



Universidade Federal de Pernambuco

Centro de Informática



Análise de Tráfego de Rede Auxiliado por Processadores Gráficos

Dissertação de Mestrado

Aluno: Ademir José de Carvalho Junior (ajcj@cin.ufpe.br)

Orientador: Prof. Dr. Djamel Sadok (djamel@cin.ufpe.br)

Recife, Março de 2011

Catálogo na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Carvalho Junior, Ademir José de
Análise de tráfego de rede auxiliado por
processadores gráficos / Ademir José de Carvalho
Junior - Recife: O Autor, 2011.
x, 63 folhas: il., fig., tab.

Orientador: Djamel Sadok.
Dissertação (mestrado) - Universidade Federal de
Pernambuco. CIn, Ciência da Computação, 2011.

Inclui bibliografia.

1. Redes de computadores. 2. Análise de tráfego. I.
Sadok, Djamel (orientador). II. Título.

004.6
176

CDD (22. ed.)

MEI2011 –

Dissertação de Mestrado apresentada por **Ademir José de Carvalho Junior** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Análise de Tráfego de Rede Auxiliado por Processadores Gráficos**", orientada pelo **Prof. Djamel Fawzi Hadj Sadok** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Stênio Flavio de Lacerda Fernandes
Centro de Informática / UFPE

Prof. Eduardo Lúzeiro Feitosa
Departamento de Ciência da Computação / UFAM

Prof. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 11 de março de 2011.

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Universidade Federal de Pernambuco
Centro de Informática

Análise de Tráfego de Rede Auxiliado por Processadores Gráficos

Dissertação de Mestrado

Dissertação apresentada ao Centro de
Informática da Universidade Federal de
Pernambuco, como requisito para obtenção do
Grau de Mestre em Ciência da Computação.

Orientador: *Prof. Dr.* Djamel Sadok

Recife, Março de 2011

À minha família.

Agradecimentos

Agradeço primeiramente a Deus, por ter me dado força e coragem de lutar pelos meus objetivos, e por ter guiado meus caminhos para que eu pudesse viver este momento.

Aos meus pais Sr. Ademir e D. Nilza, por todo amor, paciência e por nunca permitirem que eu desistisse do meu sonho, sendo exemplos de esforço e dedicação ao trabalho. Pela ótima base educacional que me proporcionaram, e pelos valores e princípios ensinados que me acompanharão pelo resto de minha vida.

Aos amigos em geral, que souberam entender a minha ausência tantas vezes e que também me proporcionaram muitos momentos de alegria. Entre eles, a turma do “projêto”, a turma do GPRT e a turma do Janga.

Aos meus professores e orientadores, Djamel Sadok e Judith Kelner, por todos os ensinamentos e apoio, e pela oportunidade da realização desta dissertação de Mestrado e aos companheiros do grupo de pesquisa, que elucidaram muitas dúvidas sobre o tema desta dissertação.

E não menos importante, àqueles que não citei, mas que contribuíram de alguma forma para a realização deste trabalho, ou que contribuíram para que eu chegasse até este momento.

Meus sinceros agradecimentos.

Abstract

Computer Networks are elements of great importance for the technological progress of society in general. It is the duty of the managers of large networks present in our daily life keep them free of problems, to prevent harm to users of networks and organizations involved. In this context, tools capable of monitoring networks and provide information through the analysis of network traffic is very helpful. But with the increasing speed of current networks to the order of gigabits per second, some of these tools cannot cope with such data throughput. To solve this problem, researchers in the computer networks field have been studying alternative techniques for dealing with high outflow of current networks links. Some new solutions employ special hardware such as Network Processors, Field Programmable Gate Arrays (FPGAs) and even the modern and powerful graphics processing units (GPUs). Among the technologies mentioned, the GPUs are highlighted by its high processing power, lower cost and easy programming. With an eye on this research niche, this paper proposes a new methodology based on the use of GPUs for computer networks traffic monitoring of and providing support information for network administration. A prototype able to perform analysis such as the volume of traffic between hosts and traffic speed was developed using CUDA platform, one of the main platforms for GPU programming. Statistical tests were applied in order to compare the prototype with a similar implementation that does not use GPUs. Results show significant performance gains in processing the traffic.

Key-Words: Traffic Analysis, High-Speed Networks, GPU, GPGPU, CUDA.

Resumo

Redes de Computadores constituem um elemento de grande relevância para o avanço tecnológico da sociedade em geral. É dever dos administradores das grandes redes presentes em nosso cotidiano mantê-las livres de problemas, para evitar prejuízos aos usuários e às organizações envolvidas. Neste contexto, ferramentas capazes de monitorar as redes e fornecer informações através da análise do tráfego da rede são bastante úteis. Porém, com o aumento da velocidade das redes atuais para a ordem dos gigabits por segundo, algumas destas ferramentas não conseguem lidar com tamanha vazão de dados. Para resolver este problema, pesquisadores da área de redes de computadores vêm estudando técnicas alternativas para lidar com a alta vazão dos enlaces de redes atuais. Algumas das novas soluções empregam hardwares especiais como *Network Processors*, *Field Programmable Gate Arrays* (FPGAs) e até mesmo as modernas e poderosas unidades de processamento gráficas (GPUs). Dentre as tecnologias citadas, as GPUs se destacam pelo seu alto poder de processamento, menor custo e fácil programação. De olho neste nicho de pesquisa, este trabalho propõe uma nova metodologia, baseada na utilização de GPUs, para o monitoramento do tráfego de redes de computadores e fornecimento de informações de suporte para a administração da rede. Um protótipo capaz de realizar análises como o volume de tráfego entre hosts e a velocidade do tráfego foi desenvolvido utilizando a plataforma CUDA, uma das principais plataformas de programação para GPUs. Testes estatísticos foram aplicados no intuito de comparar o protótipo com uma implementação similar que não utiliza GPUs. Os resultados mostraram um ganho significativo de desempenho no processamento do tráfego.

Palavras-Chave: Análise de Tráfego, Redes de Alta Velocidade, GPU, GPGPU, CUDA.

Sumário

1.	INTRODUÇÃO.....	13
1.1	Problema e Motivação	14
1.2	Objetivos	15
1.3	Estrutura do Documento	15
2.	ANÁLISE DE TRÁFEGO	16
2.1	<i>Sniffers</i>	16
2.1.1	libpcap	17
2.2	SNMP & NetFlow.....	17
2.3	RRDTool & MRTG.....	18
2.4	NIDS – Network Intrusion Detection Systems.....	19
2.4.1	Snort.....	20
2.5	DPI – Deep Packet Inspection.....	21
2.5.1	CAM - Content Addressable Memory.....	22
3.	GPU - GRAPHICS PROCESSOR UNITS.....	24
3.1	GPGPU	25
3.2	CUDA	27
4.	TRABALHOS RELACIONADOS	30
4.1	BV-TCAM.....	30
4.2	NG-MON (Next Generation Monitor).....	31
4.3	PacketShader	35
4.4	GNORT	36
4.5	Discussão	38
5.	IMPLEMENTAÇÃO DO SOFTWARE	39
5.1	Visão Geral.....	39
5.2	Fluxos.....	40
5.3	Banco de Dados	41
5.4	Coletor CPU	44
5.5	Coletor GPU	45
5.6	Detalhes de Implementação do Coletor	47
5.7	Cliente	47
6.	AVALIAÇÃO DE DESEMPENHO	50
6.1	Métricas Existentes.....	50

6.2	Métricas Utilizadas	50
6.2	Teste Estatístico	53
6.3	Cenário	54
6.4	Resultados.....	54
6.4.1	Análise Descritiva da Primeira Amostra	55
6.4.2	Análise Descritiva da Segunda Amostra.....	56
6.4.3	Resultado do Teste de Hipóteses	57
6.5	Discussão	59
7.	CONCLUSÕES E TRABALHOS FUTUROS	60
7.1	Contribuições	60
7.2	Dificuldades Encontradas.....	60
7.3	Trabalhos Futuros	60
8.	REFERÊNCIAS	62

Índice de Ilustrações

Figura 1 – Exemplo de Gráfico Produzido pelo RRDTool.....	19
Figura 2 – Comparação entre GPU/CPU.....	24
Figura 3 – Transistores na CPU/GPU.	25
Figura 4 - Modelo de Execução de CUDA.....	28
Figura 5 – Diagrama FPGA do BV-TCAM.	31
Figura 6 – Fases do NG-Mon.....	32
Figura 7 – Captura de Pacotes no NG-Mon.....	33
Figura 8 – Geração do Fluxo no NG-Mon.	33
Figura 9 – Armazenamento do Fluxo no NG-Mon.....	34
Figura 10 – Análise e Apresentação de Dados no NG-Mon.	34
Figura 11 – Arquitetura do PacketShader.....	35
Figura 12 – Fluxo do PacketShader.	36
Figura 13 – Arquitetura do Gnort.	37
Figura 14 – Visão Geral do Software.....	39
Figura 15 – Modelo conceitual do Banco de Dados.....	42
Figura 16 – Tabelas MySQL.	43
Figura 17 – Exemplo simplificado de coleta.	43
Figura 18 – Comportamento do Coletor (Versão CPU).....	44
Figura 19 – Pseudo-Código da 1ª Thread na Versão CPU.....	45
Figura 20 – Pseudo-Código da 2ª Thread na Versão CPU.....	45
Figura 21 – Comportamento do Coletor (Versão CPU + GPU).....	46
Figura 22 - Pseudo-Código da 1ª Thread na Versão CPU + GPU.	46
Figura 23 - Pseudo-Código da 2ª Thread na Versão CPU + GPU.	46
Figura 24 - Pseudo-Código da 3ª Thread na Versão CPU + GPU.	46
Figura 25 – Interface Gráfica da Aplicação.	48
Figura 26 – Exemplo de Funcionamento ao Longo do Tempo.	51
Figura 27 – Exemplo de Arquivo de Log.	52
Figura 28 – Amostra do desempenho na execução da primeira versão.	55
Figura 29 – Função de densidade da primeira amostra.....	56
Figura 30 – Amostra do desempenho na execução da segunda versão.	56
Figura 31 – Função de Densidade da Amostra GPU.	57

Índice de Tabelas

Tabela 1 – Configuração adequada no NG-Mon.....	34
Tabela 2 – Configuração de Hardware do PacketShader.	36
Tabela 3 – Parâmetros da primeira amostra.	55
Tabela 4 – Parâmetros da segunda amostra.....	57

1. INTRODUÇÃO

Redes de Computadores constituem atualmente um elemento de grande relevância para o avanço tecnológico da sociedade em geral. Elas permitem facilidades que não se imaginavam há algum tempo atrás: conversação remota entre pessoas, incluindo áudio e vídeo; notícias em tempo real e ensino a distância (Kozierok 2005).

As redes de computadores são, tecnicamente falando, formadas por computadores interconectados entre si e que interagem seguindo um determinado padrão, ou protocolo. A rede mais conhecida atualmente é a Internet, sendo uma rede pública, que conecta diversas outras redes de computadores.

Alguns exemplos práticos do uso das redes de computadores, e consequentemente da Internet, são citados em (Tanenbaum 2003). Estes exemplos se resumem à aplicações comerciais e domésticas. O principal representante da categoria comercial são as aplicações de compartilhamento de recursos, que visam principalmente tornar programas, equipamentos e dados ao alcance de todas as entidades que participam de determinada rede. Como as redes não são limitadas a um espaço físico, existe uma verdadeira quebra de barreiras geográficas. Por exemplo, pessoas em países diferentes podem consultar e trocar dados e informações como se estivessem fisicamente próximas.

Já no âmbito doméstico estão as aplicações de acesso às informações remotas, representadas principalmente pela *World Wide Web*, aplicações que permitem acesso às mais diversas informações como artes, negócios, cultura, governo, saúde, etc. Estão também as aplicações que permitem a comunicação pessoal, como o Messenger e o Skype, as aplicações de entretenimento interativo e também as aplicações de comércio eletrônico.

Então não só as pessoas comuns, mas também as organizações se beneficiam das redes pelo grande potencial para alavancar seus negócios. Por exemplo, *web sites* que expõem seus serviços e produtos, canais de interação direta com clientes (como *chats* e emails), realização de transações financeiras, entre outros benefícios (Cronin 1995).

Tais benefícios criam uma dependência às redes de computadores à medida que elas se tornam mais presentes em nosso cotidiano. Isso significa que

quaisquer problemas que afetem o funcionamento das redes, podem gerar contratempos ou até prejuízos para seus usuários.

Neste contexto, pode-se citar alguns riscos: como o de existir pessoas mal-intencionadas que rastreiam brechas para realizar atividades maliciosas como apropriação indevida de informações confidenciais, degradação e até indisponibilização de serviços (Richardson 2008). Um exemplo deste fato é a existência de técnicas como o DoS (*Denial of Service* ou Negação de Serviço), que consiste em uma atividade coordenada para tirar uma aplicação ou serviço do ar. Outro risco possível é a má configuração das redes, o que pode causar problemas de congestionamento, e assim prejudicar seu funcionamento.

1.1 Problema e Motivação

Com o crescimento das redes de computadores, torna-se indispensável o seu bom gerenciamento, para evitar os riscos anteriormente citados. Assim, em grandes organizações surge o papel da administração da rede, que é a entidade responsável pelo monitoramento, segurança e resolução de problemas que porventura venham a acontecer em uma rede. A administração também é responsável pelo planejamento de operações na rede, como por exemplo, a captura de perfis de uso e a avaliação do impacto de novos serviços sendo implantados na mesma (Hunt 1998).

O avanço tecnológico experimentado recentemente e o consequente aumento na velocidade das redes atuais, trazem um desafio para esta categoria de sistemas. Estes precisam realizar processamento de tráfego com uma alta vazão de pacotes. A consequência quando não se consegue lidar com esta restrição é o fornecimento de informações que não são coerentes com a realidade da rede em questão (Shomura, Watanabe e Ikeda 2009).

Para resolver este problema, pesquisadores da área de redes de computadores vêm estudando técnicas alternativas para lidar com a alta vazão dos enlaces de redes atuais. Algumas das novas soluções empregam hardwares especiais como *Network Processors*, *Field Programmable Gate Arrays* (FPGAs) e até mesmo as modernas e poderosas unidades de processamento gráficas (GPUs) (Baker e Prasanna 2004).

Dentre as tecnologias citadas, as GPUs (Owens, et al. 2008) se destacam

pelo seu alto poder de processamento, menor custo e fácil programação. GPUs são dispositivos originalmente concebidos para realização de computações gráficas. Esta arquitetura, quando devidamente utilizada, proporciona bons ganhos de desempenho em relação às CPUs tradicionais. Por este motivo, muitos trabalhos tentam utilizar GPUs, para tentar obter ganhos de desempenho em áreas fora do escopo de computação gráfica como, Inteligência Artificial, Economia e Biologia. Este é um campo de pesquisa conhecido como GPGPU (*General Purpose Computation on GPU*).

1.2 Objetivos

Os objetivos deste trabalho são: Realizar uma revisão de literatura nas áreas de Análise de Tráfego e GPUs (*Graphics Processing Units*); implementar um protótipo de software de análise de tráfego utilizando GPUs; e avaliar o desempenho deste software.

O protótipo a ser implementado será capaz de realizar análises como o volume de tráfego entre hosts e a velocidade do tráfego, e será desenvolvido utilizando a plataforma CUDA, uma das principais plataformas de programação para GPUs. Testes estatísticos serão aplicados no intuito de comparar o protótipo com uma implementação similar que não utiliza GPUs. Espera-se obter um aumento da capacidade de inspeção de tráfego, minimizando o problema da inspeção de tráfego em tempo real em redes de alta velocidade, o que poderá ser confirmado pelos resultados dos testes.

1.3 Estrutura do Documento

Assim, este trabalho está organizado da seguinte forma. As seções 2 e 3 apresentam alguns conceitos necessários para o entendimento da proposta. A seção 4 resume alguns trabalhos relacionados. A seção 5 descreve a implementação do software desenvolvido neste trabalho. A seção 6 contém a avaliação de desempenho e os resultados obtidos. A seção 7 conclui o trabalho e sugere alguns trabalhos futuros e por fim, a seção 8 contém as referências utilizadas.

2. ANÁLISE DE TRÁFEGO

Existem dois grandes assuntos abordados neste trabalho: a análise de tráfego, o qual este capítulo está dedicado, e as GPUs, tratada no próximo capítulo. Assim, a função destes dois capítulos é introduzir os conceitos necessários para fundamentar o restante do trabalho.

2.1 *Sniffers*

Sniffing consiste na atividade de capturar todos os pacotes que trafegam em uma rede (Ansari, Rajeev e Chandrashekar 2002). Neste contexto, *Sniffer* é o termo utilizado para designar as aplicações que realizam *Sniffing*. Os usos típicos para este tipo de aplicação incluem:

- Captura de tráfego de rede;
- Análise de desempenho de rede;
- Detecção de intrusões na rede.

Uma Interface de Rede (NIC - *Network Interface Card*), quando recebe quaisquer dados da rede, primeiramente checa se estes lhe são endereçados. Se sim, aceita os dados, se não os dados são descartados. O “truque” dos *Sniffers* é colocar a Interface de Rede em modo Promíscuo. Neste modo, a Interface de Rede não descarta os dados que não lhe são endereçados, mas os aceita silenciosamente. Com a ajuda de artifícios como o espelhamento de tráfego¹, os *sniffers* conseguem desempenhar corretamente seu papel. Pois estas aplicações ganham acesso a todos os pacotes da rede, podendo interpretá-los e realizar as devidas atividades.

Existem vários *Sniffers* disponíveis na Internet. Um dos mais conhecidos é o Wireshark, (Combs 2007). Esta ferramenta detalha todos os pacotes capturados. Pode-se definir esta aplicação como um navegador de pacotes, onde é possível selecionar um pacote do tráfego e inspecionar todo o seu conteúdo.

Outro *Sniffer* conhecido é o TNV (*Time-based Network Traffic Visualizer*) (Goodall, et al. 2005), que também permite visualizar em detalhes o tráfego, porém a um nível maior de. Sua interface contém ilustrações que mostram quantitativamente

¹ Espelhamento de Tráfego consiste em configurar um equipamento de rede, como um Switch, por exemplo, para espelhar todo o tráfego que passa pelo dispositivo para uma porta específica.

atividades entre hosts, atividades entre portas, etc. Informações que permitem analisar um contexto mais amplo da rede.

2.1.1 libpcap

Uma das principais bibliotecas para a implementação de *Sniffers* é a `libpcap` (Garcia 2008), uma biblioteca open-source que fornece uma interface de alto nível para a captura de pacotes. Foi desenvolvida em 1994 por pesquisadores do *Lawrence Berkeley National Laboratory*, como parte de um projeto de pesquisa para a investigação e melhoria do protocolo TCP.

O principal objetivo dos seus autores foi criar uma API (*Application Programming Interface*) independente de plataforma para eliminar a dependência de recursos do sistema operacional ou de módulos específicos. Esta API foi projetada inicialmente para ser utilizada em aplicações C/C++, porém existem *wrappers* para diversas outras linguagens. A biblioteca executa na maioria dos sistemas operacionais UNIX-like, como Linux, Solaris e BSD, e existe também uma versão para Windows, a WinPcap¹. Atualmente a biblioteca é mantida pelo grupo *Tcpdump*².

2.2 SNMP & NetFlow

Existem outras alternativas além dos *Sniffers* para monitorar o comportamento da rede. Uma delas é o protocolo SNMP (*Simple Network Management Protocol*) (Stallings 1998). Este protocolo, implementado por dispositivos de rede, permite, através de certos comandos, checar o estado atual da rede. O NetFlow (Cisco IOS NetFlow 2002), similar ao SNMP, é um protocolo proprietário desenvolvido pela Cisco e implementado apenas nos seus dispositivos. O NetFlow introduziu o conceito de fluxo de rede, uma sequência unidirecional de pacotes que compartilham os mesmos seguintes valores (denominados 7-upla comum):

- Endereço IP de Origem
- Endereço IP de Destino
- Porta de Origem

¹ Para mais informações, visite <http://www.winpcap.org/>.

² Para mais informações, visite <http://www.tcpdump.org/>.

- Porta de Destino
- Protocolo IP
- Interface de Ingresso
- Tipo de Serviço IP

Assim, os dispositivos são capazes de exportar informações sobre os fluxos de rede como quantidade de pacotes e bytes associados ao fluxo. Apesar de serem opções viáveis de monitoramento, o uso destes protocolos foi descartado neste trabalho por não oferecerem controle sobre a coleta do tráfego, o que não permitiria customizar esta etapa no software, para incluir o uso de GPUs, por exemplo.

2.3 RRDTool & MRTG

O RRDTool (*Round-Robin Database Tool*) (Oetiker 2004) se enquadra na categoria de ferramentas auxiliares aos *Sniffers*. O RRDTool é uma ferramenta para coleta de séries numéricas, que em sua versão inicial, o MRTG (*Multi-Router Traffic Grapher*) (T. Oetiker 1998), foi projetada para coletar dados sobre o estado de redes de computadores, porém agora já possui flexibilidade suficiente para ser empregada em outros tipos de séries de dados como temperatura, uso de CPU, etc.

Como o seu nome sugere, esta ferramenta segue uma estratégia *Round-Robin*, possuindo uma estrutura similar a um *buffer* circular: uma quantidade fixa de elementos, onde ao se preencher todos os elementos do *buffer*, sobrescreve-se os elementos mais antigos. Uma característica interessante do RRDTool é a possibilidade de possuir vários *buffers*, cada um contendo uma série com uma resolução diferente da original. Neste caso, as resoluções podem possuir médias, máximos ou mínimos dos dados originais em relação aos intervalos de tempo em que a série original esta sendo coletada. Esta é uma característica interessante quando se quer analisar dados em diferentes granularidades.

O RRDTool também tem a capacidade de produzir gráficos que permitem ter uma idéia visual das séries armazenados, podendo estes gráficos serem utilizados em outros sistemas, como por exemplo, o Cacti (Cacti Group 2006). O Cacti é uma espécie de *front-end* para sistemas baseados no RRDTool.

A Figura 1 ilustra um gráfico produzido pelo RRDTool, contendo duas

séries e mais algumas informações auxiliares.

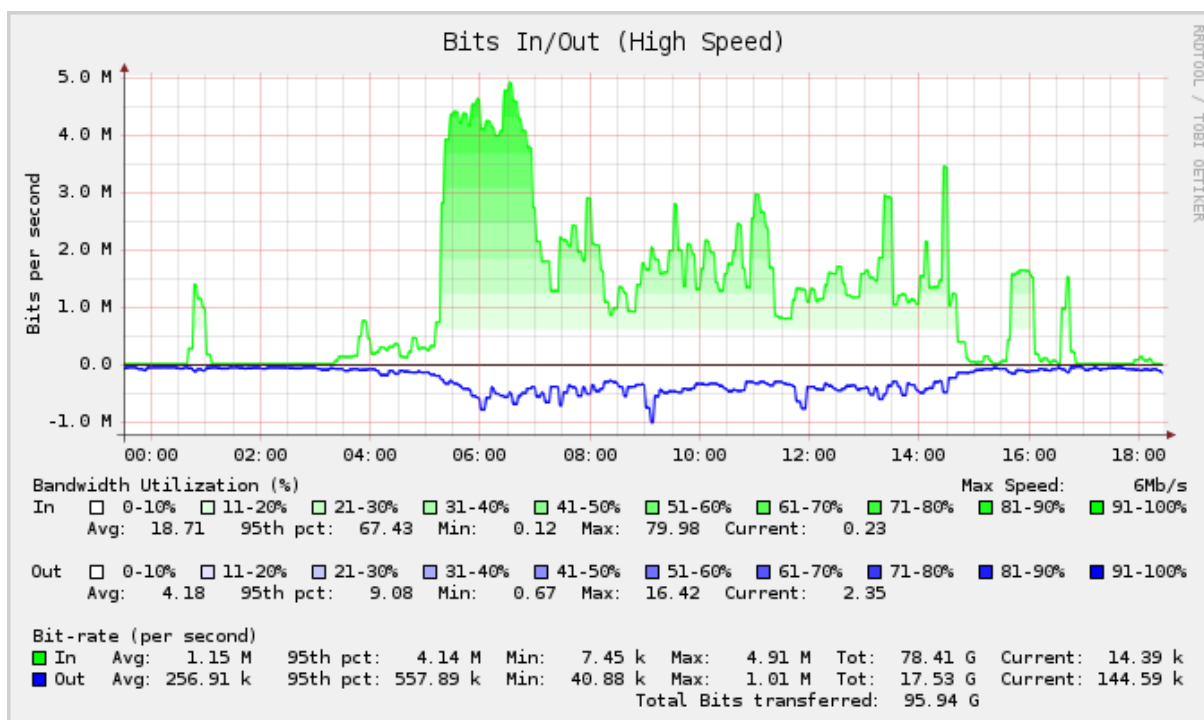


Figura 1 – Exemplo de Gráfico Produzido pelo RRDTool.

Sobre o MRTG, que ficou limitado apenas à análise de dados de rede, este funciona utilizando o protocolo SNMP para consultar informações sobre a rede e alimentar as séries de dados correspondentes. Em conjunto com o Cacti, o MRTG constitui uma ferramenta bastante utilizada para monitoramento de enlaces de rede.

Apesar de constituírem ferramentas de análise de tráfego, estas aplicações não se enquadram nos trabalhos relacionados por não terem o foco no desempenho desta atividade, característica em que este trabalho se baseia.

2.4 NIDS – Network Intrusion Detection Systems

Uma importante aplicação da análise de tráfego é a busca por atividades maliciosas (Fuchsberger 2005). A este tipo de aplicação dá-se o nome de NIDS (*Network Intrusion Detection Systems*).

A motivação para o surgimento deste tipo de aplicação foi o aumento dos ataques as redes de computadores, tanto em quantidade, como em qualidade (Richardson 2008), atrelada ao aumento da dependência das organizações às suas redes de computadores. Assim, evitar problemas à rede significa evitar prejuízos à

organização.

Os NIDS operam examinando os pacotes da rede e verificando as informações de seu cabeçalho e de seu *payload*, denominação que designa o conteúdo útil de um pacote. Este conteúdo é comparado com regras previamente armazenadas, conhecidas como assinaturas. Estas assinaturas descrevem o comportamento malicioso de uma aplicação específica, podendo incluir portas utilizadas e determinados padrões no *payload*. O banco de dados de assinaturas constitui parte vital de qualquer NIDS, pois a precisão das assinaturas implica na precisão do sistema. Outro ponto importante aqui, e que será melhor detalhado na próxima seção, é a forma de inspeção do *payload* de um pacote. Tal procedimento é denominado *Deep Packet Inspection* (Kumar, et al. 2006).

Quando o NIDS realiza um *match*, ou seja, encontra um pacote de acordo com as informações de uma assinatura, ele executa uma ação de alerta, e dependendo do NIDS, pode realizar também uma ação de resposta à esta atividade maliciosa. As ações de alertas podem ser emails enviados ao administrador do sistema, alertas visuais na interface gráfica do software, entre outros. Já as ações de resposta podem ser a elaboração de novas regras no *firewall*, a reinicialização de uma conexão, etc. Estas ações variam bastante de um sistema para outro, e geralmente são configuráveis pelo usuário. Apesar do foco deste trabalho não ser segurança de rede, é importante detalhar os NIDS já que muitos dos trabalhos relacionados se baseiam em NIDS.

2.4.1 Snort

O Snort (Snort 2011) é um NIDS *open-source* bastante popular. O Snort é capaz de detectar uma grande variedade de ataques, como *buffer overflows* e *port scanners*. Além disso, o Snort é capaz de gerar alertas em tempo real, podendo registrar os alertas no *syslog* (padrão para mensagens de log em redes IP), exibir *Popups* do Windows, ou mesmo gerar arquivos de alertas. É programado utilizando uma sintaxe própria que descreve alguns testes nos pacotes e ações relacionadas. Esta simplicidade facilita o desenvolvimento de novas regras de detecção. Um exemplo deste fato foi quando o *exploit* do IIS *Showcode* foi revelado. As regras de detecção no Snort demoraram poucas horas para serem lançadas (Parcens e Lopht 1999).

O Snort é composto por três subsistemas principais: um decodificador de

pacotes; um motor de detecção; e um subsistema de log e alerta (Snort 2011). A principal funcionalidade do decodificador de pacotes, baseada na libpcap, é a captura dos pacotes e a preparação para que estes sejam manipulados pelos outros subsistemas. Já o motor de detecção mantém as regras em uma estrutura do tipo lista ligada com duas dimensões: a cadeia de cabeçalhos e a cadeia de opções. A cadeia de cabeçalhos contém os filtros para os endereços IP e portas de origem e destino. Já a cadeia de opções contém dados mais avançados como as assinaturas. Assim, para cada pacote itera-se recursivamente pelas regras em ambas as dimensões. A primeira regra que se aplica ao pacote tem sua ação executada e encerra-se a iteração. O subsistema de log e alerta possui algumas opções de log e outras de alerta, que são definidas via linha de comando. Os pacotes podem ser gravados em um formato legível em diretórios baseados nos endereços IP dos pacotes, ou em um único arquivo no formato similar ao do tcpdump. A terceira opção é desativar o log, deixando o sistema com um melhor desempenho. Os alertas podem ser enviados para o *syslog*, para arquivos texto, para popups Windows ou, assim como o log, serem completamente desativados.

As regras do Snort são simples e flexíveis. Há três diretivas de ação possíveis quando uma regra se aplica a um pacote: *pass*, *log* e *alert*. A diretiva *pass* simplesmente descarta o pacote. A diretiva *log* grava o pacote de acordo com a opção de log selecionada e a diretiva *alert* gera um evento utilizando a opção de alerta selecionada. A seguir, um exemplo de regra do Snort, é ilustrado:

```
alert tcp any any -> 10.1.1.0/24 80 (content: "/cgi-bin/phf"; msg: "PHF probe!";)
```

Esta regra detecta tentativas de acesso ao serviço PHF em qualquer dos servidores web (porta 80) locais, que estão na faixa de endereço 10.1.1 classe C. Neste caso, se um pacote é detectado, um evento de alerta é gerado e o pacote é gravado através do mecanismo selecionado.

2.5 DPI – Deep Packet Inspection

Como já mencionado na seção anterior, um dos pontos que merecem destaque é a forma de inspeção do *payload* dos pacotes.

Apesar das técnicas de DPI atuais examinarem a carga útil dos pacotes, vale ressaltar que ferramentas antigas utilizavam estratégias mais simples para realizar esta tarefa. Uma destas era a baseada em portas conhecidas (*well-known*

ports), que exploram o fato de que algumas aplicações e serviços utilizam portas específicas em sua operação, como por exemplo, o protocolo HTTP que utiliza a porta 80. Estudos já confirmam a ineficiência em relação à precisão desta estratégia (Moore e Papagiannaki 2005).

Outras estratégias aplicam estatísticas para classificar os fluxos (conceito introduzido pelo NetFlow) em uma classe de fluxos, através de algoritmos de aprendizagem de máquina (McGregor, et al. 2004), ou agrupamento (Moore e Zuev 2005), (Roughan, et al. 2004). Nestas técnicas os fluxos são agrupados em *clusters*, levando em consideração aspectos como tamanho dos pacotes, tempo médio de duração dos fluxos e o tempo entre a chegada de pacotes. Estas técnicas aplicam conhecimento de estudos que examinaram como identificar aplicações através destas informações (Hernandez-Campos, et al. 2003).

Já as técnicas mais recentes de *Deep Packet Inspection* (Kumar, et al. 2006) envolvem o uso de expressões regulares na comparação do *payload* com as assinaturas. Alguns algoritmos de comparação como Aho-Corasick (Aho e Corasick 1975), Commentz-Walter (Commentz-Walter 1979), e Wu-Mamber (Wu e Mamber 1994) usam abordagens de pré-processamento para realizar esta comparação com alto desempenho. Estas técnicas são mais populares por serem mais precisas.

Porém, o número, o tamanho e a complexidade destas expressões aumentam de acordo com a complexidade das aplicações. Deste modo, é exigido cada vez mais das técnicas de DPI, já que estas precisam lidar com um numero maior de expressões, e expressões mais complicadas (Becchi, Franklin e Crowley 2008). Neste contexto, Autômatos Finitos Determinísticos (Vespa e Weng 2011) são explorados por diversos trabalhos para atender a estas restrições. Estes autômatos possuem capacidade para avaliar diversas expressões simultaneamente, possuindo desempenho linear em função apenas do tamanho de um pacote. O seu grande desafio é o uso de memória, já que assinaturas complexas podem levar a uma quantidade muito de grande de estados a serem representados, o que significa alto consumo de memória.

2.5.1 CAM - Content Addressable Memory

As técnicas citadas anteriormente são puramente algorítmicas, porém existem também as técnicas arquiteturais, projetadas para proverem um desempenho

ainda maior, pois envolvem elementos de hardware em sua concepção. Uma solução neste âmbito são as CAMs (*Content Addressable Memories*) (Kempke e McAuley 1998), que constituem uma memória endereçável por conteúdo.

CAM é um tipo de memória especial, conhecida também como *memória associativa*, *armazenamento associativo* ou *array associativo*. Este último em referência a uma estrutura similar existente nas linguagens de programação. Seu funcionamento é oposto ao da memória RAM convencional. Enquanto a memória RAM aceita endereços como entrada e retorna os dados situados neste endereço, uma CAM aceita dados como entrada e então retorna uma lista de endereços onde estes dados estão armazenados. A busca é realizada em toda a memória em uma única operação, desta forma a sua velocidade de acesso é bem superior a velocidade de uma memória RAM, sendo da ordem de nanosegundos.

CAMs possuem uma baixa densidade de dados e possuem um alto consumo de energia, assim são caras e não são encontradas em computadores comuns. Costumam ser encontradas em equipamentos de rede como roteadores e *switches*, sendo utilizada para evitar problemas de desempenho no encaminhamento e na classificação de pacotes. As CAMs mais comuns são as binárias, que realizam buscas apenas por zeros e uns. Porém existem as TCAMs (*Ternary Content Addressable Memory*), que permitem buscar um terceiro elemento denominado *don't care*. Este representa um coringa e significa que seu valor pode ser tanto “zero” quanto “um”. Isto é útil ao lidar com máscaras de rede, onde os endereços IP constituídos pela parte da rede e pela parte do host podem ser encontrados de forma mais eficiente.

3. GPU - GRAPHICS PROCESSOR UNITS

Após apresentar alguns conceitos relacionados à análise de tráfego, este capítulo explora os conceitos relativos as GPUs, elemento central na realização deste trabalho. GPUs (*Graphics Processing Units*) (Owens, et al. 2008) (Luebke e Humphreys 2007) são dispositivos que vêm ganhando destaque nos últimos anos, principalmente pela sua eficiência em processamento. Esta eficiência é obtida através de sua arquitetura, com alto grau de paralelismo, característica necessária para a execução de sistemas gráficos.

De acordo com (NVIDIA 2009a), as recentes arquiteturas das GPUs proporcionam, além de um vasto poder computacional, uma grande velocidade na transferência de dados. Isso possibilita desempenho superiores às CPUs comuns em determinadas situações, como ilustra a Figura 2.

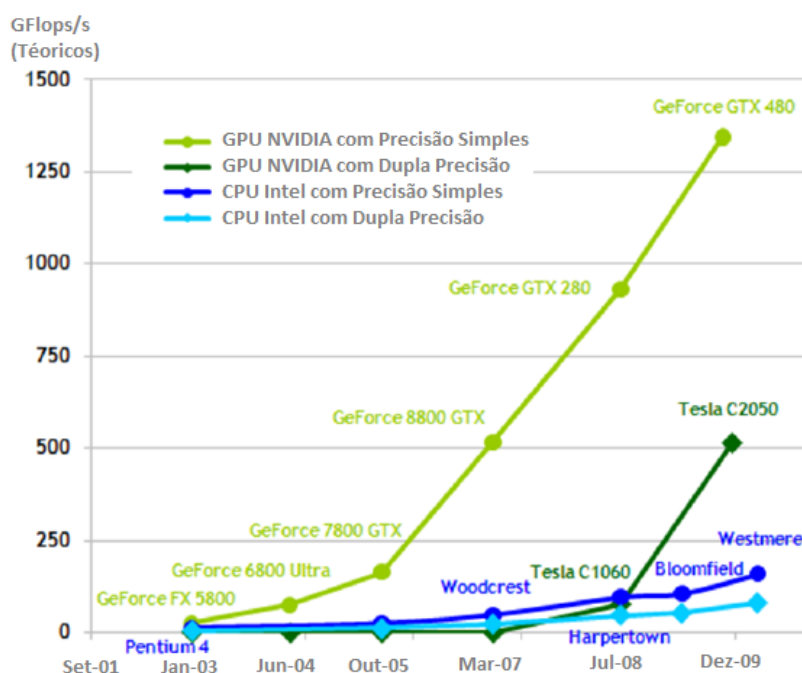


Figura 2 – Comparação entre GPU/CPU¹.

Um conceito importante é o modo de execução das instruções. GPUs são processadores orientados a execução paralela de instruções e são otimizados para processar operações sobre vetores, executando no modo denominado SIMD (*Single*

¹ Figura extraída e traduzida de (NVIDIA 2009a).

Instruction, Multiple Data)¹. Isto se reflete na arquitetura das GPUs. De modo geral, uma GPU inclui mais transistores dedicados ao processamento de dados, em detrimento da realização do *caching* de dados e fluxo de controle, quando comparado a uma CPU comum, como ilustrado na Figura 3.

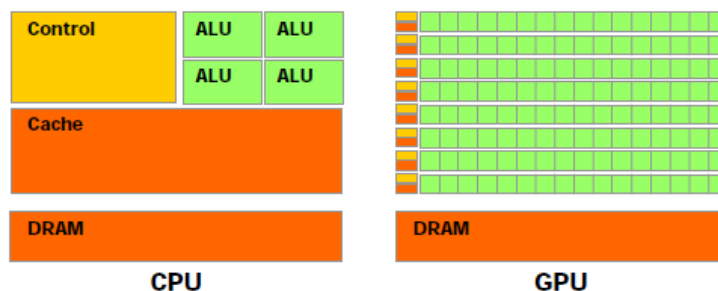


Figura 3 – Transistores na CPU/GPU.

Vale ressaltar que a arquitetura de uma GPU é bastante relacionada ao denominado pipeline gráfico, conjunto de operações simultâneas necessárias para a renderização de imagens na tela. A arquitetura das GPUs foi projetada de modo a realizar as operações contidas neste pipeline de forma simultânea. Porém, recentemente, novas funcionalidades foram adicionadas às GPUs, incluindo a habilidade de modificar o processo de renderização através dos *shaders* (Shirley 2005). *Shaders* consistem em funções definidas pelo usuário criadas no intuito de customizar o processo de renderização. Esta flexibilidade é considerada o ponto de entrada para realizar computações fora do escopo da computação gráfica, e consiste no pilar básico para o GPGPU.

3.1 GPGPU

GPGPU (Owens, et al. 2007) é um conceito que visa explorar as vantagens das GPUs em áreas não necessariamente relacionadas ao processamento gráfico. Seu principal desafio é identificar os problemas que podem ser tratados pelas GPUs, e como mapear estes problemas para a plataforma, já que ela foi inicialmente desenvolvida com o foco em aplicações gráficas.

Nem todos os problemas se encaixam no contexto das GPUs, mas alguns

¹ SIMD é uma técnica aplicada para obtenção de paralelismo em processadores. No SIMD, uma mesma instrução é executada em paralelo em diferentes partes do hardware e sobre diferentes dados.

tipos de aplicações conseguem um aumento significativo de desempenho através do uso do hardware gráfico. Uma boa métrica para avaliar a chance de uma aplicação tirar proveito das vantagens de uma GPU é a sua intensidade aritmética. Essa métrica mede a razão entre a quantidade de operações matemáticas e a quantidade de acessos à memória existentes em uma aplicação. Quando mapeadas para as GPUs, aplicações que possuem alta intensidade aritmética obtêm melhor desempenho.

Porém, a natureza gráfica das GPUs torna árdua a programação para outros fins, exigindo do programador conhecimentos adicionais, como a adaptação da aplicação ao pipeline gráfico. Por exemplo, no caso de uma aplicação em GPGPU que utiliza *shaders*, é necessária a renderização de alguma primitiva gráfica (por exemplo, um quadrilátero texturizado) para que o aplicativo seja inicializado, assim como a obtenção do *feedback* da aplicação, que é realizada através da leitura de texturas.

Conceitualmente, uma aplicação comum não possui relação com o desenho de gráficos e manipulação de texturas, mas as linguagens existentes forçam os programadores a pensarem desta forma. As plataformas mais recentes, como Cg (C for Graphics) (Fernando e Kilgard 2003), Accelerator (Tarditi, Puri e Oglesby 2006), GLSL (OpenGL Shading Language) (Rost 2004) e HLSL (High Level Shading Language) (Microsoft 2006), tentam lidar com este problema abstraindo alguns detalhes da GPU para o programador. A plataforma utilizada neste trabalho será o CUDA, a ser apresentada na próxima seção.

O Brook (Buck, et al. 2004), uma linguagem para GPGPU, é uma extensão da linguagem ANSI C. Esta extensão contém conceitos de um modelo de programação denominado *streaming*. Um *stream*, conceitualmente, significa um *array*, exceto pelo fato de que todos os seus elementos podem ser operados em paralelo através de funções conhecidas como *kernels*. Esta linguagem mapeia automaticamente *kernels* e *streams* em *shaders* e memórias de texturas. O Folding@Home (Larson, et al. 2002), um projeto de computação distribuída para o estudo do comportamento de proteínas, é um exemplo de aplicação que utiliza o Brook.

Até então, as plataformas citadas, são fortemente relacionadas à hardwares específicos (aplicações CUDA executam apenas em placas gráficas da NVIDIA, assim como Brook foi projetada para as GPUs da AMD/ATI). Este fato abriu

espaço para o desenvolvimento de plataformas independente do hardware: OpenCL (Khronos 2009) e o DirectCompute (Microsoft 2009).

3.2 CUDA

Apesar das linguagens independentes de hardware tenderem para um padrão (como OpenCL), estes ainda não estão consolidados o bastante. CUDA é a plataforma com mais tempo de uso, e por isso, mais familiar para a maioria dos desenvolvedores, além de possuir uma maior quantidade de aplicações e bibliotecas desenvolvidas. Para as placas da NVIDIA, o CUDA sempre irá apresentar o melhor desempenho entre as demais plataformas, pois foi desenvolvida objetivando especificamente estas placas gráficas. Além disso, existem ferramentas de programação, *guidelines*, e uma boa comunidade de suporte. Estas razões motivaram o uso de CUDA neste trabalho.

A arquitetura de CUDA (NVIDIA 2009a) foi lançada pela NVIDIA em Novembro de 2007 para ser utilizada em conjunto com suas placas gráficas. Os primeiros processadores compatíveis com CUDA são os da série G80.

O modelo de software proposto por CUDA e pela maioria das linguagens para GPGPU enxerga a GPU como um co-processador de dados paralelo. Neste contexto, a GPU é considerada como o dispositivo (ou *device*) e a CPU como anfitrião (ou *host*). Alguns conceitos existentes nesta plataforma são *threads*, *blocks*, *grids* e *kernels*, conforme ilustrado na Figura 4. *Threads* são unidades de execução paralela em uma GPU. Elas são agrupadas em *blocks*, onde *threads* pertencentes a um mesmo *block* podem sincronizar sua execução e compartilhar um mesmo espaço de memória (*shared memory*). Um conjunto de *blocks* representa um *grid*. E um *kernel* consiste no código que é executado por uma *thread*. Cada chamada a um *kernel* precisa especificar uma configuração contendo o número de *blocks* em cada *grid*, o número de threads em cada *block* e opcionalmente a quantidade de memória compartilhada a ser alocada (NVIDIA 2009a).

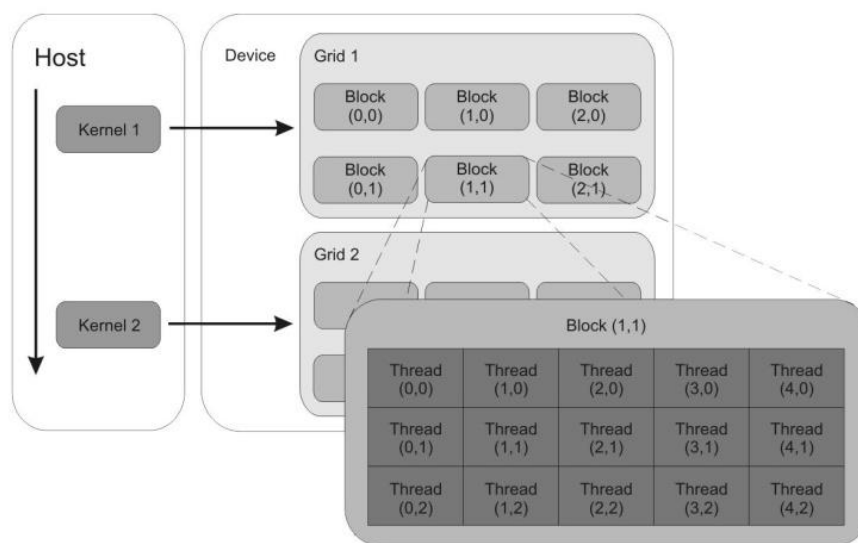


Figura 4 - Modelo de Execução de CUDA.

A sintaxe da linguagem é similar à sintaxe da linguagem C, com algumas extensões que controlam aspectos intrínsecos à programação paralela como sincronização, identificação das *threads*, tipos vetoriais e operações atômicas. Porém, se o usuário é familiarizado com a linguagem C, a curva de aprendizado é baixa (NVIDIA 2009a). A versão 3.0 de CUDA adiciona suporte a C++, que permite o usuário utilizar o paradigma orientado a objetos.

CUDA executa na maioria dos sistemas operacionais (Windows, Linux e MacOS), e fornece ferramentas como o Nexus (NVIDIA 2010a), um ambiente de desenvolvimento que suporta programação tanto em código para CPU quanto para CUDA, depuração de hardware e análise de desempenho diretamente no Microsoft Visual Studio (Microsoft 2005). Entretanto, na ausência de uma IDE mais complexa como o Nexus, um simples editor de texto e uma linha de comando são suficientes para programar em CUDA.

Existem tanto produtos comerciais como produtos acadêmicos como exemplos de aplicações que tiram benefícios da plataforma CUDA. No âmbito comercial, existem tendências para maximizar o desempenho de várias aplicações como motores físicos (NVIDIA PhysX (NVIDIA 2009b)), decodificadores de vídeo (Roxio Creator 2010 (Roxio 2010)) e processamento de som (ACUSTICA Nebula3 (Acustica 2009)). De forma similar, na área acadêmica GPUs são utilizadas para aumentar o desempenho de aplicações como em (Bakkum e Skadron 2010), onde a GPU é utilizada para acelerar um conjunto de instruções de banco de dados em um

DBMS (Database Management System), com um ganho de 28 a 36x. Outros exemplos acadêmicos de CUDA podem ser encontrados em (Farias, et al. 2008b), (Santos, et al. 2009) e em (Leite, et al. 2009), que modelam algoritmos e estruturas de dados complexos que utilizam informação espacial para tarefas de renderização.

4. TRABALHOS RELACIONADOS

Alguns trabalhos têm explorado a questão do aperfeiçoamento na vazão de processamento de pacotes (Dharmapurikar e Lockwood 2005). Outros (Baker e Prasanna 2004) (Yu, Katz e Lakshman 2004) propõem hardwares especializados para inspecionar vários pacotes concorrentemente, incluindo TCAMs e FPGAs. Apesar de eficientes, eles são complexos de programar e modificar. Além de possuírem uma baixa flexibilidade, já que suas abordagens são ligadas a uma implementação de hardware em específico.

(Nottingham e Irwin 2009) fazem um estudo sobre a viabilidade no uso de GPUs na classificação de pacotes. Este estudo faz uma avaliação positiva neste ponto, e ilustra algumas técnicas de classificação passíveis de serem utilizadas em uma solução baseada em GPU. Nas próximas seções os trabalhos relacionados mais relevantes serão descritos em maiores detalhes.

4.1 BV-TCAM

O BV-TCAM é uma arquitetura para classificação de pacotes, baseada em FPGA e utilizada para Sistemas de Detecção de Intrusão (NIDS). O BV-TCAM combina o TCAM (*Ternary Content Addressable Memory*) e um algoritmo de Bit-Vector (BV)¹ para reduzir a representação dos dados e assim aumentar a vazão no processamento de pacotes.

O BV-TCAM separa a classificação de pacotes em duas dimensões. A primeira é a classificação do cabeçalho, dimensão que lida com a conhecida 5-upla de um pacote (IP de origem, IP de destino, Protocolo IP, Porta de origem e Porta de destino). A segunda lida com a carga útil do pacote (*payload*). O argumento para esta separação é que os dados do cabeçalho de um pacote possuem tamanho constante e aparecem em posições fixas dentro de um pacote. Já as assinaturas para realização da comparação com o *payload* possuem tamanho variável e podem aparecer em qualquer posição no pacote.

Sendo assim, a idéia do BV-TCAM é lidar com algumas informações do pacote através do TCAM enquanto outras são tratadas através do algoritmo de Bit-

¹ Algoritmos de Bit-Vector são bastante utilizados para comparação de strings, e são adequados para serem executados paralelamente.

Vector. Os endereços IPs de origem e destino, e o Protocolo IP são processados através de uma TCAM enquanto as portas de origem e destino são tratadas através do algoritmo de BV.

Esta arquitetura foi prototipada e avaliada em um Xilinx FPGA XCV2000E na plataforma FPX. O diagrama de blocos do circuito é representado na Figura 5:

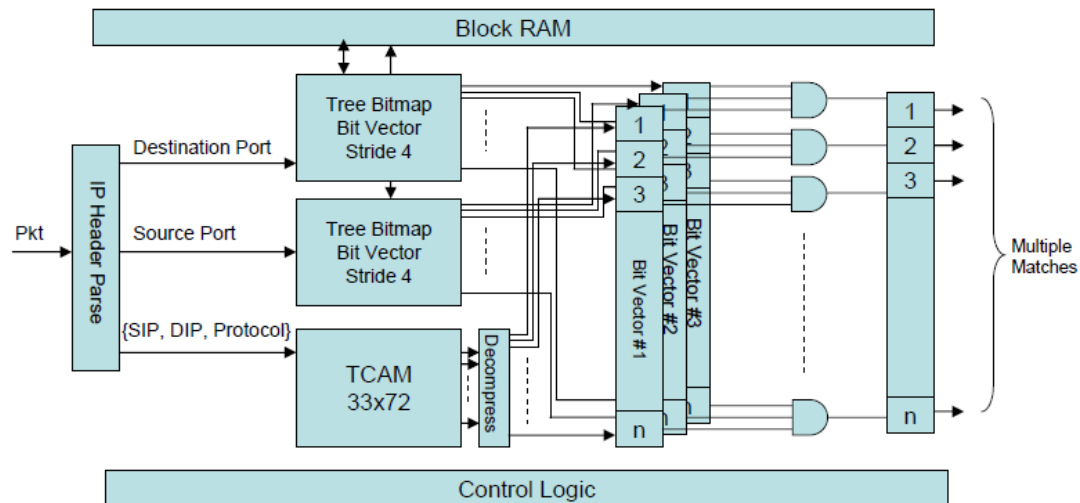


Figura 5 – Diagrama FPGA do BV-TCAM.

Assim como o trabalho proposto nesta dissertação, o BV-TCAM é alimentado pelos pacotes que trafegam por um segmento de rede. Sendo assim, as mesmas preocupações com a vazão no processamento dos pacotes existem. Ambas utilizam dispositivos especiais (TCAM no BV-TCAM e GPU neste trabalho) para obter melhores desempenhos em suas atividades.

Os resultados do BV-TCAM em relação ao desempenho não são claros. Os autores apenas argumentam que o circuito final fica bastante simples e que não ocupa parte significativa da área lógica de um FPGA. Segundo os autores a principal vantagem é o ganho em desempenho proporcionado pelo FPGA durante a classificação pacotes e a desvantagem é a baixa flexibilidade ao atualizar as regras e assinaturas de classificação.

4.2 NG-MON (Next Generation Monitor)

O NG-MON (*Next Generation MONitoring*) (Han, et al. 2002) é um trabalho que propõe uma arquitetura para análise e monitoramento do tráfego de

redes de computadores. Seu alvo são redes de alta velocidade, da ordem de 10 Gbps.

O trabalho envolve técnicas de distribuição, *pipelining* e processamento paralelo para atingir seus objetivos. Entretanto, não demanda hardware específico na sua execução, podendo ser implantado com computadores simples, constituindo assim, uma solução com um baixo custo. Além do mais, por utilizar distribuição, possui a característica de escalabilidade agregada à solução.

As atividades no NG-Mon são divididas em etapas, que podem ser executadas em diferentes *clusters* de computadores que cooperam entre si, aumentando o desempenho geral do sistema. As cinco etapas são: captura de pacotes, geração de fluxos, armazenamento dos fluxos, análise de tráfego e apresentação dos dados. As etapas podem ser visualizadas na Figura 6.

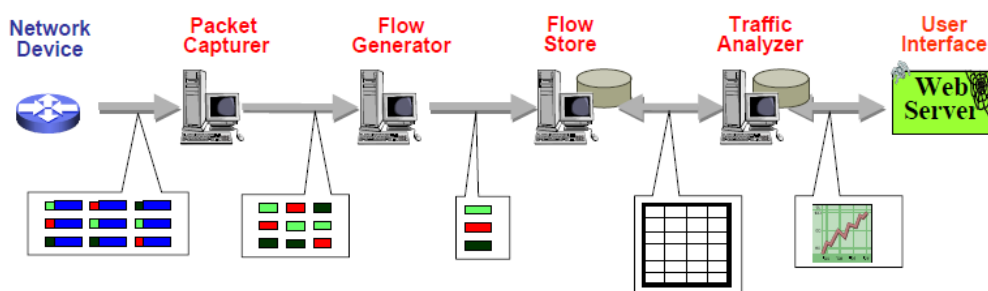


Figura 6 – Fases do NG-Mon.

Na primeira etapa, os dispositivos de rede são configurados para distribuir o tráfego para os computadores alocados para a captura de pacotes. Nesta etapa, informações são extraídas dos pacotes e armazenadas em buffers. Quando os buffers estão preenchidos, as informações são enviadas para os computadores alocados para a próxima fase.

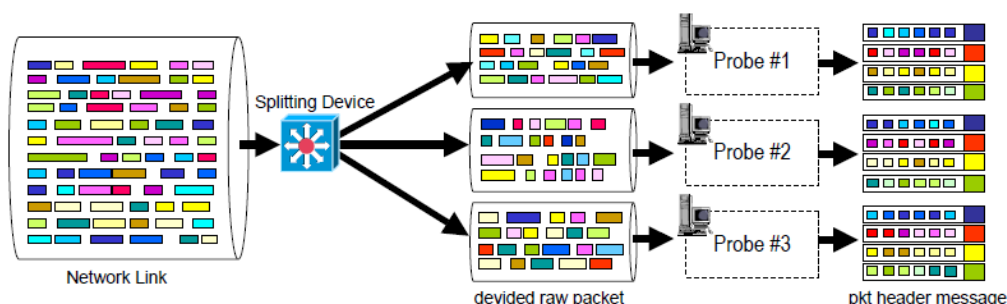


Figura 7 – Captura de Pacotes no NG-Mon.

Na segunda fase, a geração do fluxo, as informações provenientes da primeira etapa são recebidas e é criada (ou atualizada, caso já exista) uma entrada para o fluxo correspondente ao pacote. É importante ressaltar que existe o cuidado na arquitetura que pacotes do mesmo fluxo sejam enviados a mesma máquina para evitar espalhamento e consequente incoerência das informações. A entrada do fluxo é mantida em memória até que o fluxo se encerre ou expire, sendo enviada à próxima fase.

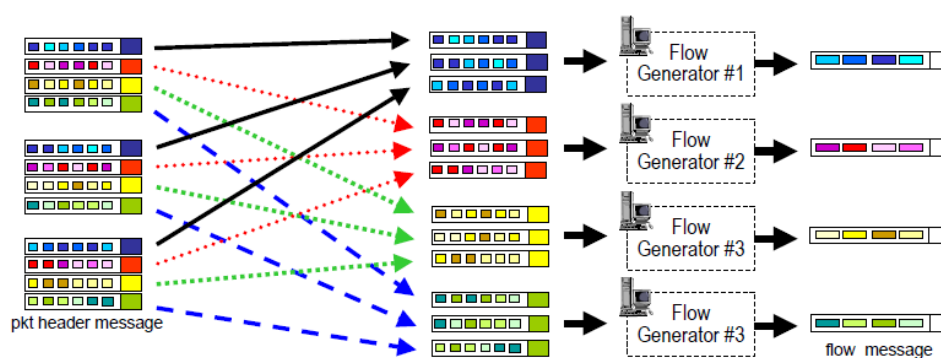


Figura 8 – Geração do Fluxo no NG-Mon.

A terceira fase, armazenamento dos fluxos, consiste na tarefa de persistir em um banco de dados as informações de fluxos recebidas. O recebimento das informações da etapa anterior é realizado seguindo uma estratégia Round-Robin para balancear a carga na operação de persistência entre os computadores alocados a esta etapa. Por ser a etapa considerada mais crítica em termos de desempenho, há o cuidado de sincronizar as escritas no banco priorizando-as em relação às consultas ao banco (realizadas na próxima fase).

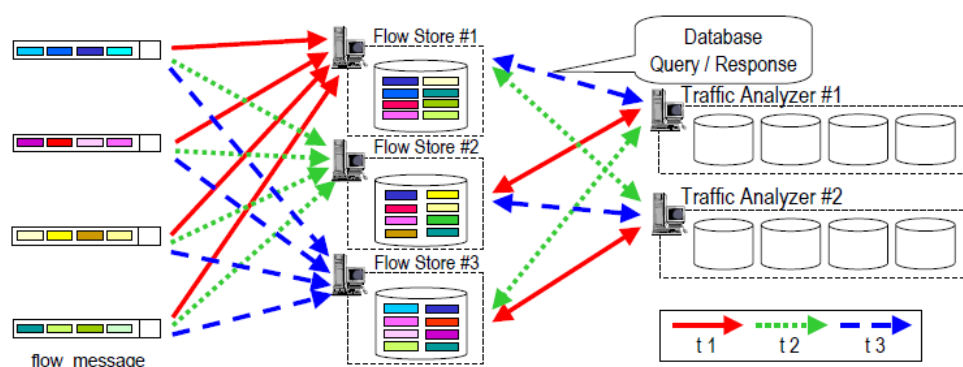


Figura 9 – Armazenamento do Fluxo no NG-Mon.

Na quarta e quinta etapas, análise e apresentação, os dados são consultados e organizados temporalmente, seguindo uma abordagem no estilo do RRDTool. Posteriormente são disponibilizados para os usuários através de um servidor Web.

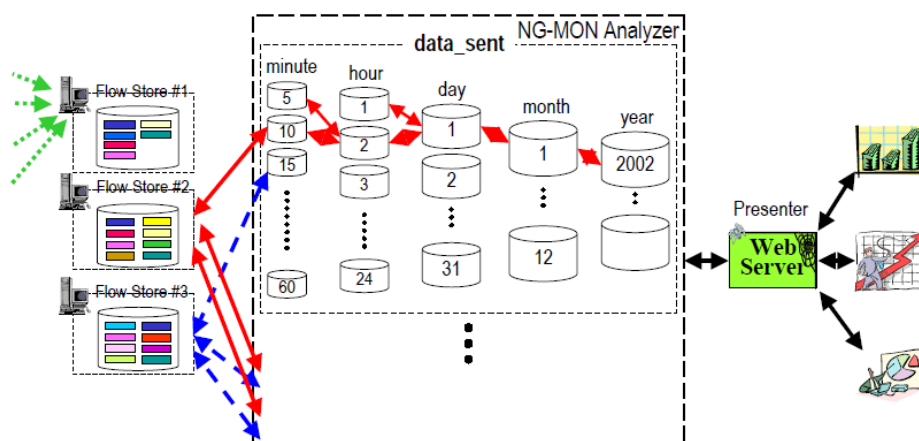


Figura 10 – Análise e Apresentação de Dados no NG-Mon.

Os autores do trabalho validaram analiticamente a arquitetura, chegando a uma conclusão de quantos computadores seriam necessários em cada fase para atender ao monitoramento de uma rede de acordo com sua velocidade. Os resultados podem ser visualizados na tabela abaixo. Os computadores considerados possuem a seguinte configuração: processador Pentium 4 2GHz, memória de 1 GB RDRAM, placa de rede gigabit Ethernet e barramento PCI de 66 MHz/64 bits.

Tabela 1 – Configuração adequada no NG-Mon

	Captura de Pacotes	Geração do Fluxo	Armazenamento do Fluxo	Total
--	--------------------	------------------	------------------------	-------

100 Mbps	1			1
1 Gbps	1		2	3
10 Gbps	7	3	5	15

4.3 PacketShader

O PacketShader (Han, et al. 2010) é um arcabouço de roteamento em software de alto desempenho para processamento de pacotes, acelerada por GPU. Consiste em uma aplicação *multi-thread*. Sua arquitetura geral pode ser vista na Figura 11. As aplicações executam no topo do arcabouço e são controladas principalmente por três funções de *callback* (funções executadas em resposta a algum evento específico), o *pre-shader*, o *shader* e o *pos-shader*.

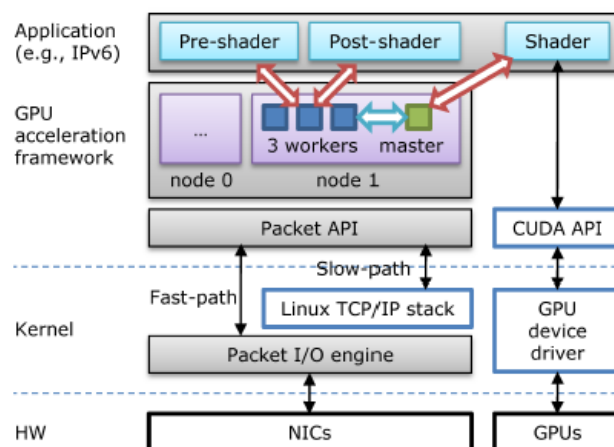


Figura 11 – Arquitetura do PacketShader.

No *pre-shading*, os pacotes são coletados e pré-processados (pacotes mal-formados são descartados, estruturas de dados inicializadas e dados copiados para uma fila de entrada). No *shading*, estas estruturas de dados são processadas via GPU e o resultado copiado para uma fila de saída. E no *pos-shading*, os dados da fila de saída são examinados, e os pacotes são modificados, descartados ou duplicados dependendo dos resultados obtidos no processamento. Finalmente, os pacotes são encaminhados às portas de destino para transmissão. A Figura 12 apresenta este comportamento.

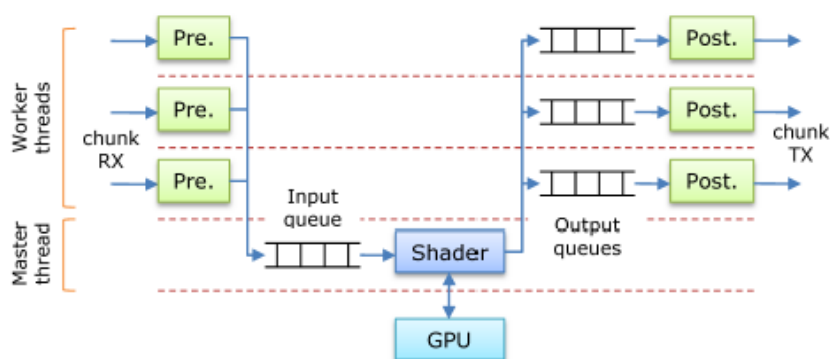


Figura 12 – Fluxo do PacketShader.

Os testes foram realizados em um servidor cuja especificação está resumida na Tabela 2. O custo total, incluindo outros componentes, foi cerca de U\$ 7.000 (sete mil dólares). No servidor foi instalada a versão 64 bits do Servidor Ubuntu Linux 9.04, com a versão 2.6.28.10 do kernel do Linux, sem modificações. O motor de E/S de pacotes é baseada no driver ixgbe 2.0.38.2 para os adaptadores Intel 10 GbE PCIe. Em relação à GPU, foi utilizado o driver 195.36.15 e o SDK 3.0 de CUDA.

Tabela 2 – Configuração de Hardware do PacketShader.

Item	Especificação	Quantidade	Preço
CPU	Intel Xeon X5550 (4 núcleos, 2.66 GHz)	2	\$925
RAM	DDR3 ECC 2GB (1,333 MHz)	6	\$64
M/B	Super Micro X8DAH+F	1	\$483
GPU	NVIDIA GTX480 (480 núcleos, 1.4 GHz, 1.5GB)	2	\$500
NIC	Intel X520-DA2 (dual-port 10GbE)	4	\$628

Os testes atingiram um desempenho cerca de quatro vezes melhor que os softwares de roteamento existentes, conseguindo encaminhar pacotes IPv4 a uma taxa de 39 Gbps.

4.4 GNORT

O Gnort (Vasiliadis, et al. 2008) é um NIDS baseado no Snort, mas que executa algumas de suas funções mais importantes em uma GPU, na tentativa de

aumentar a vazão no processamento.

O trabalho se baseia no fato de que cerca de 75% do tempo de processamento nos NIDS atuais é despendido nas funções de DPI, utilizada para a classificação do tráfego. Na prática estas funções consistem em operações de comparação entre o *payload* de um pacote e as assinaturas das aplicações. O Snort, por exemplo, possui no seu conjunto de regras, cerca de 10.000 *strings*, e comparar cada pacote com todas estas strings utiliza recursos significantes em termos de processamento e memória.

Os autores então avaliam alguns algoritmos de *string-matching* (comparação de strings) quanto à viabilidade na execução em uma GPU. Na avaliação deles, o algoritmo KMP (*Knuth-Morris-Pratt*) e o BM (*Boyer-Moore*) se apresentaram inadequados, já o algoritmo *Aho-Corasick* apresentou o triplo do desempenho original.

A arquitetura do Gnort, ilustrada na Figura 13, é separada em três etapas: transferência dos pacotes para a GPU, *string-matching* dos pacotes na GPU, e transferência dos resultados de classificação de volta para a CPU.

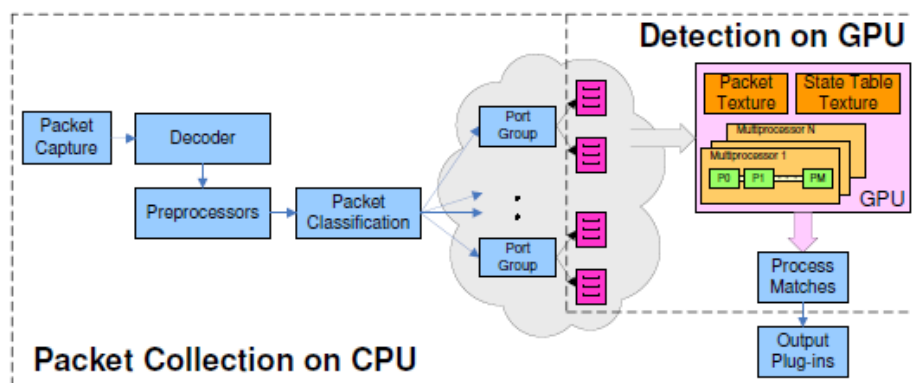


Figura 13 – Arquitetura do Gnort.

Apesar de não possuir muitos detalhes técnicos sobre a implementação, algumas considerações são feitas no trabalho. A transferência de pacotes ocorre em lote para a GPU. Existe uma organização em grupos baseado nos números das portas das regras de classificação, tendo cada grupo, um buffer associado para os respectivos pacotes. Estes buffers são transferidos para a GPU quando preenchidos ou a cada 100 milissegundos. Na etapa de processamento, o algoritmo *Aho-Corasick* portado para GPU é utilizado. Por último, na etapa da transferência dos resultados de volta para a

memória principal, um *array* com os identificadores das regras devidamente posicionados nos índices correspondentes aos pacotes é retornado para a CPU.

O Gnort foi implementado na arquitetura GeForce 8 Series, utilizando CUDA. Em testes utilizando traces de pacotes sintéticos, o Gnort obteve uma vazão de processamento de 2.3 gigabits por segundo, já em testes monitorando tráfego em tempo real de uma interface de rede Ethernet comum, ele conseguiu o dobro de desempenho do Snort.

4.5 Discussão

O BV-TCAM é um trabalho que se assemelha ao proposto, porém utiliza uma tecnologia diferente da utilizada por este trabalho, o FPGA. Isto faz com que não seja possível explorar detalhes técnicos para serem aproveitados neste trabalho.

O NG-MON utiliza uma estratégia bastante interessante, pois aproveita recursos de CPU abundantes, para distribuir a carga de processamento de tráfego. Além disso, a divisão em estágios é interessante para dar uma maior vazão ao sistema, na medida em que os estágios podem ser executados em paralelo. GPUs ainda são recursos caros se comparados as CPUs. Montar um cluster de GPUs pode ser inviável do ponto de vista financeiro, o que desviaria o foco do NG-MON que é a solução final possuir baixo custo. Por possuir uma estratégia diferente da proposta deste trabalho (processamento concentrado em uma única GPU), detalhes técnicos deste trabalho também não foram aproveitados.

Já o PacketShader e o GNORT, que utilizam GPU, possuem detalhes interessantes e que foram aproveitados pelo trabalho, como a transferência de dados em lote entre CPU e GPU e o recurso de buffer duplo que faz com que enquanto um buffer está sendo processado pela GPU, a CPU trabalha em cima do outro buffer. Estes detalhes serão melhor explicados no próximo capítulo.

5. IMPLEMENTAÇÃO DO SOFTWARE

Neste capítulo está descrita a implementação do software desenvolvido neste trabalho, incluindo detalhes sobre sua estrutura e seu comportamento. Duas versões deste mesmo software de análise de tráfego foram desenvolvidas. A principal diferença entre estas versões é a utilização ou não dos recursos de processamento de uma GPU.

5.1 Visão Geral

Nas duas versões do software existem três componentes principais.

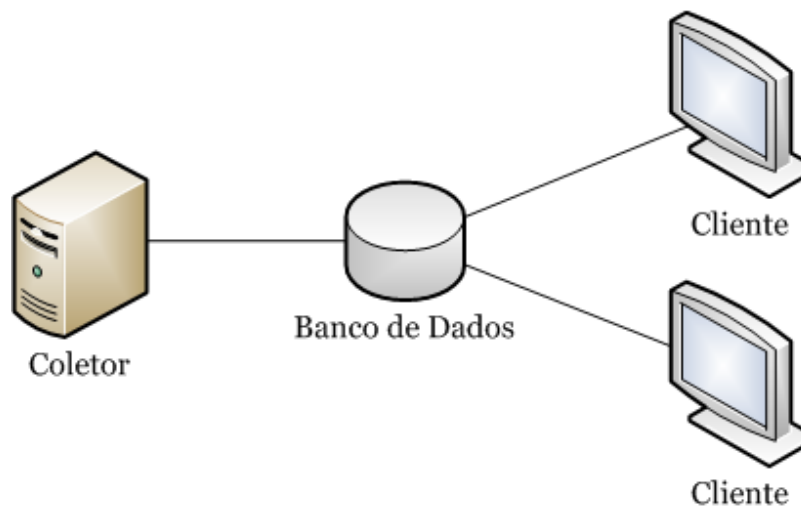


Figura 14 – Visão Geral do Software.

O componente de coleta tem função de monitorar o tráfego da rede, capturando os pacotes que servirão de entrada para todo o sistema. O banco de dados armazena toda a informação coletada e processada. Os clientes acessam as informações persistidas no banco de dados e as apresentam ao usuário (administradores de rede) em interfaces gráficas.

O componente de coleta é o ponto principal do sistema. O desempenho geral de todo sistema depende dele, uma vez que ele possui o papel de interagir com a rede, estas redes podem possuir altas velocidades, e produzirem grandes volumes de tráfego. É importante ressaltar que no início deste capítulo, quando foi mencionado que duas versões diferentes do software foram desenvolvidas, na prática duas versões diferentes do componente de coleta é que foram desenvolvidas. Estes componentes foram denominados Coletor CPU e Coletor GPU. Os demais componentes são de

caráter passivo e não possuem influência no desempenho, permanecendo inalterados nas duas versões.

5.2 Fluxos

É importante destacar a estrutura dos dados manipulados pelo software. A unidade de informação aqui é o Fluxo, conjunto de pacotes com as mesmas cinco propriedades (ou 5-upla): endereço IP de origem, endereço IP de destino, porta de origem, porta de destino e protocolo de transporte. Um fluxo representa uma conexão unidirecional entre dois computadores na rede, sendo na prática, os pacotes trocados por alguma aplicação em execução na rede. Esta definição é similar à definição de Fluxo do NetFlow, exceto pelo conjunto de propriedades.

Quando os pacotes são capturados pelo componente de coleta, uma das primeiras ações a serem executadas é identificar a qual fluxo pertence este pacote. Localizado o fluxo, este é atualizado de acordo com as informações do pacote, e então o pacote é descartado. É mais vantajoso, em termos de armazenamento, trabalhar com informações agregadas em fluxos do que com os pacotes diretamente, pois a quantidade de dados a ser trabalhada diminui significativamente, e ainda assim, um fluxo pode conter métricas representativas de milhares de pacotes.

No componente de coleta, os fluxos são armazenados temporariamente em memória até serem armazenados no banco de dados. A estrutura utilizada para este armazenamento em memória são as *Hashtables*, que mapeiam chaves em valores e são adequadas nesta situação por possuírem rápidas operações de busca, característica útil na fase de localização do fluxo de um pacote. No software, a 5-upla de um fluxo consiste na chave, enquanto as suas propriedades consistem nos valores da *Hashtable* utilizada no componente de coleta.

As propriedades armazenadas para cada fluxo são:

- Volume do tráfego em termos de pacotes;
- Volume do tráfego em termos de bytes;
- Vazão do tráfego (bytes por segundo);
- *Flags* TCP;

Estas propriedades são extraídas dos cabeçalhos e dos *payloads* dos

pacotes. E estas propriedades permitem obter análises como, por exemplo, o volume de dados originado de um determinado endereço.

5.3 Banco de Dados

O software precisa armazenar os dados coletados para posterior análise pelos administradores da rede. Existem algumas alternativas para realizar esta persistência de dados, entre elas:

- Manter os dados em memória;
- Salvar os dados em arquivos, seguindo algum formato proprietário;
- Salvar os dados em arquivos XML;
- Salvar os dados em bancos de dados.

A primeira opção tem o melhor desempenho entre as demais, favorecendo o desempenho geral do sistema. Neste estilo, não há sobrecarga causada pela transferência dos dados da memória para algum outro meio, por isso possui o melhor desempenho.

A segunda e a terceira opção consistem em armazenar os dados em arquivos, seguindo um formato proprietário ou seguindo um padrão conhecido como o XML. Apesar deste tipo de persistência ser simples, seu desempenho é a sua principal desvantagem. Operações de E/S (Entrada/Saída) são bastante lentas, o que pode comprometer o desempenho geral do sistema.

A última opção, um meio termo entre as demais, consiste em armazenar os dados em bancos de dados. Estes não possuem um desempenho baixo como os arquivos, e ainda assim os dados não são perdidos caso haja algum desligamento como acontece no caso do uso da memória como armazenamento. Por estas características, este foi o tipo de persistência escolhido.

As informações a serem persistidas no banco de dados consistem nas propriedades dos fluxos coletadas ao longo do tempo. A cada intervalo de tempo o componente de coleta transfere toda a informação contida na *Hashtable* de fluxos para o banco de dados e limpa a *Hashtable* para começar a coleta de dados para o próximo intervalo de tempo. Dado este cenário, o modelo conceitual, o qual descreve as entidades existentes e seus relacionamentos, é apresentado na Figura 15:

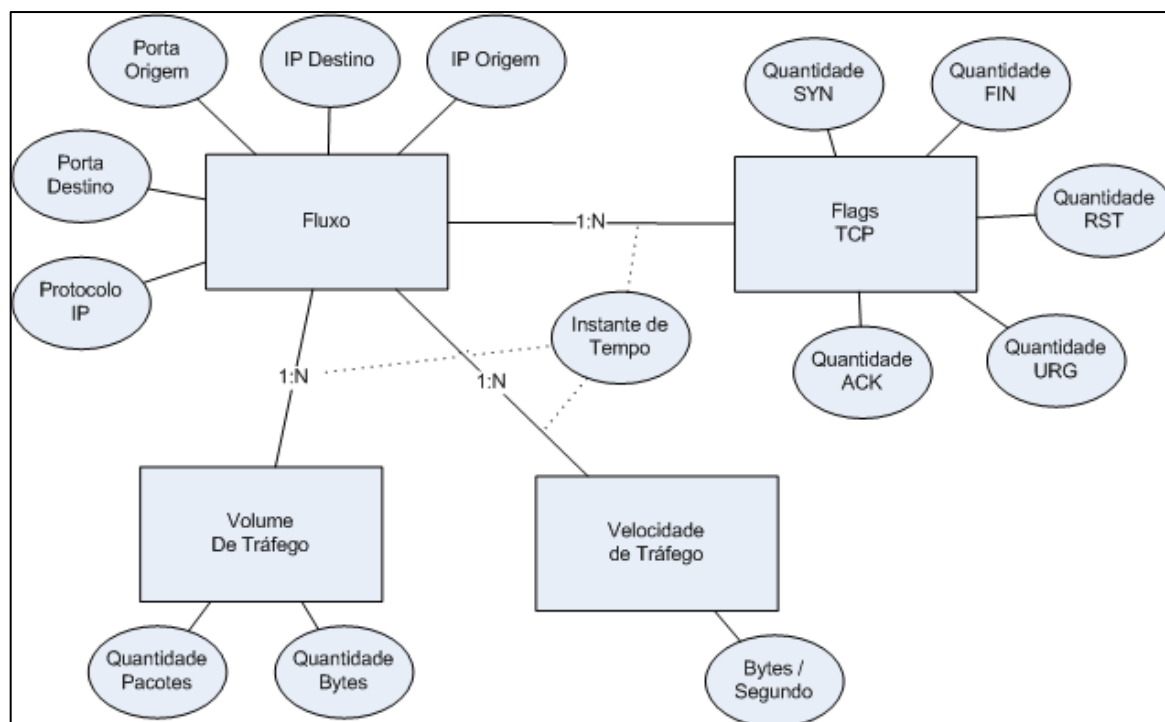


Figura 15 – Modelo conceitual do Banco de Dados.

Este modelo contém quatro entidades, a entidade Fluxo, que possui os atributos que identificam um fluxo na rede, e três entidades (Volume, Velocidade e Flags) que descrevem informações associadas ao Fluxo. Estas entidades se relacionam de forma que um fluxo pode possuir várias informações ao longo do tempo.

O banco de dados utilizado foi o MySQL, versão 5. As tabelas criadas para o modelo conceitual mostrado anteriormente estão ilustradas na Figura 16.

tb_flows		tb_amount		tb_speed		tb_tcp_flags	
PK	<u>id</u>						
	source_ip source_port destination_ip destination_port ip_protocol		flow_id timestamp packet_count bytes_count		flow_id timestamp bytes_per_second		flow_id timestamp syn fin rst urg ack

Figura 16 – Tabelas MySQL.

Existem quatro tabelas, uma para cada entidade do modelo, uma para guardar os fluxos, e as outras três para armazenar as propriedades do fluxo ao longo do tempo.

A Figura 17 exemplifica o cenário de utilização deste banco, mas por questões de simplicidade, apenas a propriedade do volume de tráfego está ilustrado no exemplo.

	t ₁	t ₂	t ₃	t ₄	t ₅
Fluxo A 192.168.0.1:2010 – 192.168.0.2:80 (TCP)	6 Pacotes 1350 Bytes	4 Pacotes 950 Bytes	2 Pacotes 120 Bytes	2 Pacotes 120 Bytes	
Fluxo B 192.168.0.3:3050 – 192.168.0.4:53 (UDP)		1 Pacotes 14 Bytes	5 Pacotes 300 Bytes		
Fluxo C 192.168.0.5:4080 – 192.168.0.6:22 (TCP)				2 Pacotes 300 Bytes	3 Pacotes 400 Bytes

Figura 17 – Exemplo simplificado de coleta.

No exemplo, tem-se em um primeiro instante apenas o fluxo A, com um tráfego de 6 pacotes, totalizando 1350 bytes. No segundo instante, além do fluxo A com 4 pacotes e 950 bytes, surge o fluxo B com apenas 1 pacote de 14 bytes. Em um terceiro instante, os dois fluxos continuam trocando pacotes. Já em um quarto instante, o tráfego do fluxo B é encerrado, o fluxo A ainda continua ativo, e surge o fluxo C, com 2 pacotes, totalizando 300 bytes. No último instante, o tráfego do fluxo A se encerra e o fluxo C troca 3 pacotes, totalizando 400 bytes. Ao final de cada

intervalo de tempo, as informações são persistidas na forma de novas entradas nas tabelas do banco de dados.

5.4 Coletor CPU

Como mencionado anteriormente, o componente de coleta lida com os pacotes da rede. De forma simplificada, captura os pacotes e atualiza os fluxos correspondentes com os dados dos pacotes coletados (Figura 18). Apesar de parecer uma tarefa simples, sua precisão e eficiência são de fundamental importância, pois é este componente que produz informação para o restante do sistema.

Na implementação real, existem duas *threads* principais neste coletor. A primeira *thread* (Figura 19) captura continuamente os pacotes da interface de rede e extrai as informações úteis do cabeçalho, como endereços IP, portas, etc. Após este passo, ele localiza na *Hashtable* de fluxos a entrada correspondente ao fluxo a que este pacote pertence. Caso não exista, uma nova entrada é criada, e então as propriedades do fluxo são atualizadas de acordo com os dados do pacote. Periodicamente, a segunda *thread* (Figura 20) percorre toda a *Hashtable* de fluxos, monta um comando SQL para execução no banco de dados, executa este comando e depois limpa a *Hashtable*.

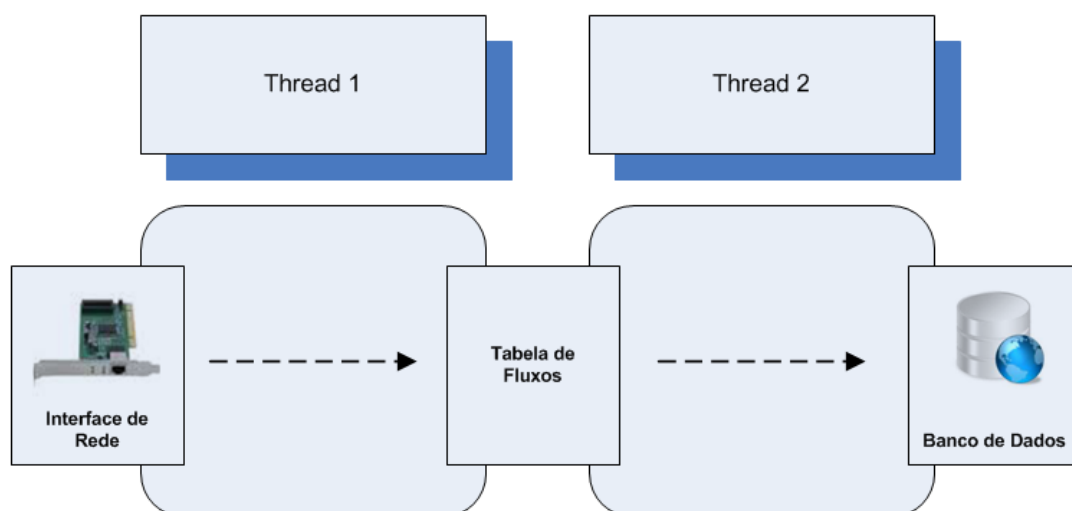


Figura 18 – Comportamento do Coletor (Versão CPU).

```
1 foreach (Packet p) {  
2     PacketInfo i = dissect_packet(p);
```

3	Flow f = get_or_create_flow(i);
4	update_flow(f, p);
5	}

Figura 19 – Pseudo-Código da 1ª Thread na Versão CPU.

1	every X time {
2	Flow[] flows = get_all_flows_and_clear();
3	String sql = build_sql_command(flows);
4	execute_sql_command(sql);
5	}

Figura 20 – Pseudo-Código da 2ª Thread na Versão CPU.

5.5 Coletor GPU

A segunda versão do componente de coleta leva em conta os principais gargalos da primeira versão e reimplementa estes pontos com o auxílio da GPU. Os principais gargalos são aqueles relacionados a grandes *loopings* e estão listados a seguir:

- O fluxo principal da primeira *thread* consiste em um *looping*, ilustrado pela linha `foreach (Packet p)` do pseudo-código (linha 1 da Figura 19);
- Na segunda *thread*, a inserção no banco de dados exige uma iteração por todos os fluxos resgatados da tabela. Isto é feito durante a elaboração do comando SQL que será executado no banco de dados (linhas 2 da Figura 20).

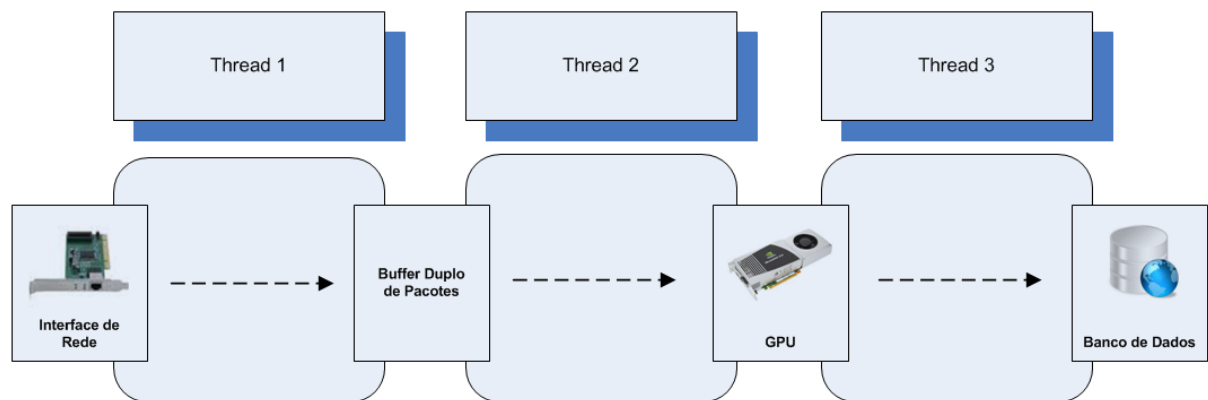


Figura 21 – Comportamento do Coletor (Versão CPU +GPU).

```
1 foreach (Packet p) {  
2     insert_in_buffer(p);  
3 }
```

Figura 22 - Pseudo-Código da 1ª Thread na Versão CPU + GPU.

```
1 while (true) {  
2     Packet[] ps = get_packets_from_buffer();  
3     copy_packets_to_gpu_and_process(ps);  
4 }
```

Figura 23 - Pseudo-Código da 2ª Thread na Versão CPU + GPU.

```
1 every X time {  
2     String sql = build_command_through_gpu();  
3     execute_sql_command(sql);  
4 }
```

Figura 24 - Pseudo-Código da 3ª Thread na Versão CPU + GPU.

Na segunda versão existem três *threads* (Figura 21). A primeira *thread* (Figura 22) coleta continuamente os pacotes da interface de rede e os insere em um *buffer*. A segunda *thread* (Figura 23) executa em *looping*, sempre resgatando os pacotes do *buffer* e processando-os paralelamente através da GPU. A terceira *thread* (Figura 24) executando periodicamente, Ela lê a tabela de fluxos, monta o comando SQL (agora através da GPU) e executa este comando no banco de dados. Vale

ressaltar que a tabela de fluxos agora reside na memória da GPU.

5.6 Detalhes de Implementação do Coletor

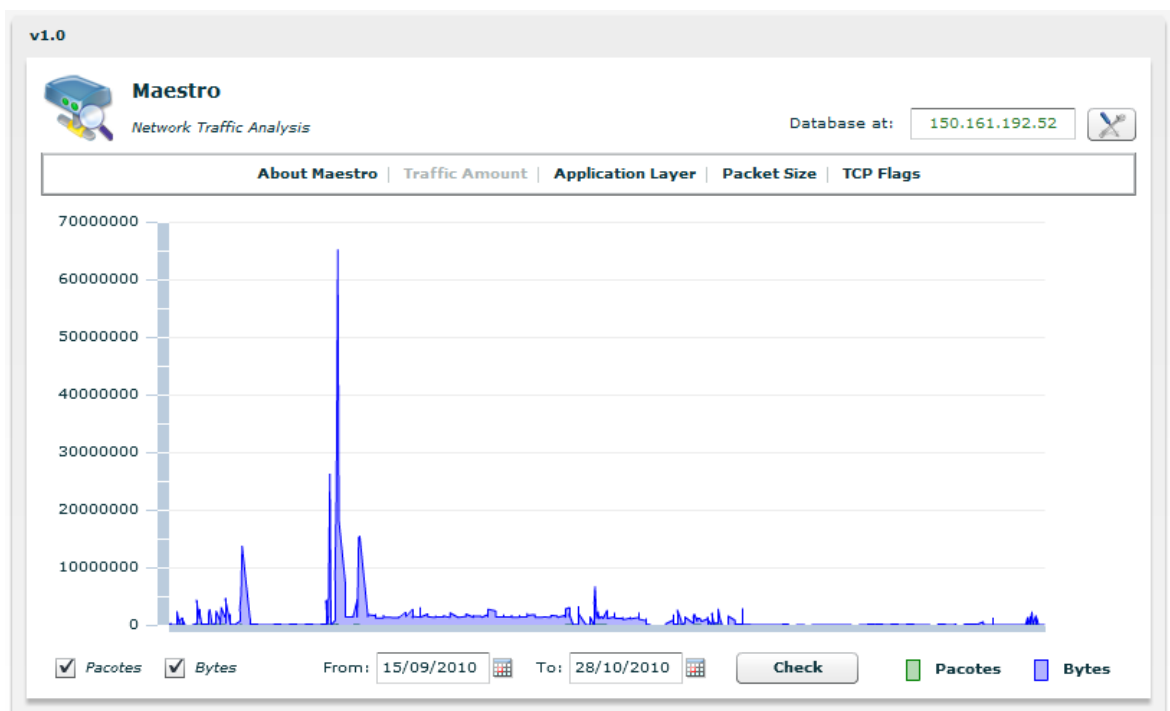
O Coletor foi implementado no ambiente Microsoft Visual Studio, em um projeto C/C++ e com o auxílio de algumas regras específicas para compilação, já que existe código CUDA no projeto. Estas regras dizem respeito à chamada do compilador `nvcc`, para compilar a parte do código envolvendo CUDA.

Outro detalhe que vale a pena ressaltar, é que devido à parte significativa do código nas duas versões do coletor ser igual, foi utilizado o recurso de configurações do Visual Studio. Em cada configuração, diferentes diretivas de compilação eram declaradas e a parte do código que difere de uma versão para outra possuía instruções `#ifdef` e `#ifndef` para compilar a versão correta. Isto facilitou bastante o gerenciamento das duas versões.

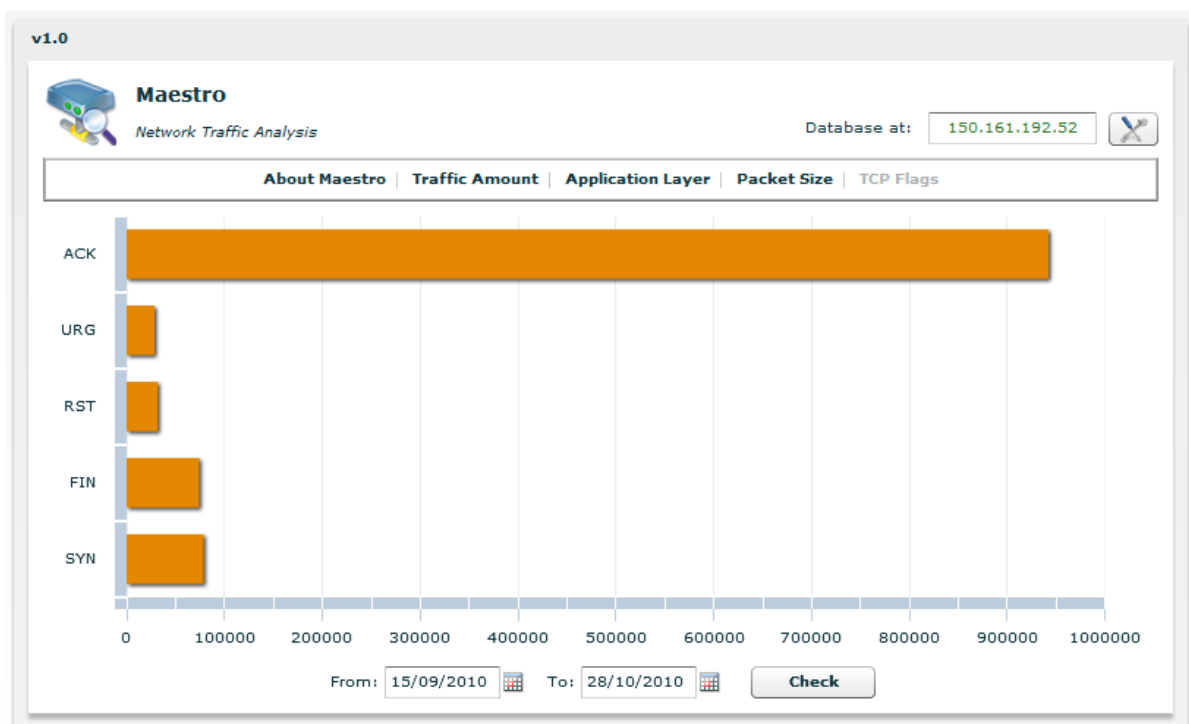
5.7 Cliente

Os clientes podem ser desenvolvidos em qualquer tecnologia que suporte acesso ao banco de dados, já que os dados coletados ficam armazenados em um banco. Mas, para fins de testes foi desenvolvida uma aplicação web para visualização dos dados. Mais especificamente, a tecnologia Adobe Flex foi utilizada nesta implementação. O Adobe Flex é um arcabouço para desenvolvimento de RIAs (*Rich Internet Applications*).

Este cliente faz consultas ao banco de dados para exibir informações e gráficos adequados para o usuário do sistema. A Figura 25 possui alguns *screenshots* da interface desta aplicação:



(a)



(b)

Figura 25 – Interface Gráfica da Aplicação.

Em (a) está um exemplo de informação possível de ser obtida através do

protótipo implementado, a quantidade de tráfego na rede ao longo de um período de tempo. Já (b) contém um indicativo das flags TCP presentes nos pacotes do tráfego em um período de tempo. Estas informações exibidas consistem nas informações coletadas pelo componente de coleta e persistidas no banco de dados.

6. AVALIAÇÃO DE DESEMPENHO

Neste capítulo está descrita a metodologia de avaliação deste trabalho. O objetivo aqui é verificar que existe um ganho de desempenho no uso da versão do software de análise de tráfego que utiliza GPU em relação à versão do software que não a utiliza. Para atingir este objetivo, cada versão foi submetida à execução em um mesmo cenário. Nesta execução, métricas de desempenho foram coletadas. Para confirmar e formalizar o ganho de desempenho e validar a hipótese deste trabalho. As duas amostras contendo as métricas coletadas foram submetidas a um teste estatístico. As próximas seções contêm detalhes sobre esta avaliação.

6.1 Métricas Existentes

Existem algumas métricas para avaliar trabalhos que utilizam GPU. Existem as mais gerais, que avaliam o desempenho de aplicações em qualquer contexto, como o *speed-up*, uma métrica que indica quantas vezes uma aplicação utilizando GPU é mais rápida que sua implementação em CPU. E também existem as mais específicas, que dependem do contexto aonde são aplicadas. Para os trabalhos que envolvem análise de tráfego existem as seguintes métricas:

- Taxa de execução: número (ou quantidade em bytes) de pacotes processados por segundo;
- Eficiência: número de pacotes processados por ciclo de *clock* do processador;
- Porcentagem da carga de processamento passada da CPU para a GPU;

As duas primeiras métricas costumam ser avaliadas realizando uma comparação entre o valor obtido na GPU contra o mesmo valor obtido em uma CPU. Neste caso o resultado é a obtenção ou não de ganhos de desempenho no uso de GPUs. A terceira métrica é um indicativo de quanto processamento foi possível delegar para a GPU, aliviando a CPU desta carga.

6.2 Métricas Utilizadas

A métrica utilizada para avaliar este trabalho foi a taxa de execução. Tomando-se como base a quantidade de dados processados e o intervalo de tempo em que este processamento foi realizado. Esta razão definirá a capacidade de processamento do software. As duas versões foram implementadas de modo a

contabilizar os pacotes que estão sendo processados e consequentemente a quantidade de bytes relativa a este conjunto de pacotes.

Para dar como finalizado o processamento de um conjunto de pacotes, estes precisam ser coletados, analisados e as informações resultantes serem inseridas no banco de dados. Assim, o intervalo a ser considerado no cálculo do desempenho consiste no intervalo de tempo em que os pacotes foram coletados somado ao intervalo de tempo em que o software está inserindo as informações no banco de dados.

Como o software não pára de coletar pacotes enquanto está inserindo informações no banco (para evitar perdas de pacotes), há uma sobreposição entre os intervalos de tempo considerados em cada iteração, como ilustra a Figura 26.

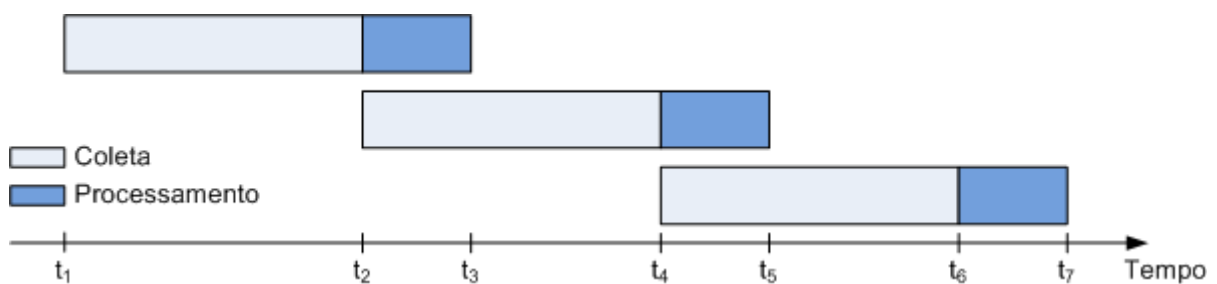


Figura 26 – Exemplo de Funcionamento ao Longo do Tempo.

A figura ilustra três iterações do software, e em cada iteração, um conjunto de pacotes é coletado e processado. Como mencionado anteriormente, a coleta e o processamento executam em paralelo.

No instante t_1 inicia-se a coleta de pacotes da primeira iteração, que ocorre até o instante t_2 . Neste instante, inicia-se o processamento dos pacotes coletados até então, mas continua-se a coletar pacotes, porém esta coleta já pertence a segunda iteração. Em t_3 , o processamento dos pacotes da primeira iteração é finalizado. Assim, a medida de desempenho para a primeira iteração se dá pela quantidade de pacotes coletados entre os instantes t_1 e t_2 , dividido pelo intervalo de tempo entre t_1 e t_3 . Aplicando a mesmo raciocínio as iterações seguintes temos:

$$desempenho(iteracao1) = \frac{\text{quantidade de bytes coletados entre } t_1 \text{ e } t_2}{\text{intervalo de tempo entre } t_1 \text{ e } t_3}$$

$$desempenho(iteracao2) = \frac{\text{quantidade de bytes coletados entre } t2 \text{ e } t4}{\text{intervalo de tempo entre } t2 \text{ e } t5}$$

$$desempenho(iteracao3) = \frac{\text{quantidade de bytes coletados entre } t4 \text{ e } t6}{\text{intervalo de tempo entre } t4 \text{ e } t7}$$

Estas medidas são registradas ao longo do tempo por ambas as versões do software. Estes escrevem os dados em um arquivo em separado. A Figura 27 ilustra um arquivo de exemplo, contendo o desempenho mencionado na coluna destacada.

1	[MAESTRO ACCOUNTING FILE]							
2								
3	KPackets	MBytes	Consol(S)	Time(S)	Dropped	KPacket/S	MByte/S	Memory(M)
4	-----							
5	4870,	406941,	0,	12,	56648,	405.83,	33911.75,	7252.00
6	2418,	189945,	1,	11,	3,	219.82,	17267.73,	7252.00
7	2110,	196075,	0,	11,	0,	191.82,	17825.00,	7320.00
8	0,	0,	0,	10,	0,	0.00,	0.00,	7320.00
9	3899,	352554,	0,	10,	0,	389.90,	35255.40,	7428.00
10	0,	0,	0,	10,	0,	0.00,	0.00,	7428.00
11	-----							
12	[6 Entries]							
13	Max:	4870,	406941,			405.83,	35255.40	
14	Min:	0,	0,			0.00,	0.00	
15	Mean:	2216,	190919,			201.23,	17376.65	
16	SD:	1987,	170715,			178.26,	15473.64	

Figura 27 – Exemplo de Arquivo de Log.

Apenas para fins informativos, outros dados também são registrados neste arquivo como a quantidade de pacotes e bytes coletados no intervalo de tempo, o tempo de processamento e o tempo total da iteração. A quantidade de pacotes perdidos na iteração também é registrada (caso haja pacotes perdidos em uma iteração esta não deve ser levada em conta para o cálculo do desempenho, além de ser

um sinal de que o software ou o hardware não está suportando a carga da rede). Por fim, um indicativo de quanta memória está sendo consumida pelo software.

6.2 Teste Estatístico

Mesmo tendo determinado como medir o desempenho do software, e supondo que os resultados demonstrem um ganho de desempenho na utilização de GPU, estatisticamente, é inviável afirmar com 100% de certeza que este ganho existe realmente. Para tanto, seria necessário conhecer a priori parâmetros acerca de toda a população da amostra, o que é algo impraticável.

Porém, existem artifícios que podem garantir com uma certa margem de confiança que os resultados são válidos. Estes artifícios se baseiam não em toda a população, mas sim em uma amostra da população, o que viabiliza a sua prática. Neste trabalho foi utilizado um teste de hipótese envolvendo as médias de desempenho dos softwares para validar o resultado do trabalho.

Para realização do teste, foi considerado o desempenho (quantidade de bytes coletados e processados por segundo) de cada versão do software como uma variável aleatória: X_{CPU} e X_{GPU} . Na prática o que deseja-se afirmar é que o desempenho médio da versão que usa GPU é superior ao desempenho médio da versão que não utiliza GPU. Em termos estatísticos:

$$E(X_{GPU}) > E(X_{CPU})$$

Isso significa que o valor médio de X_{GPU} é maior que o valor médio da X_{CPU} . Reformulando a equação, obtemos:

$$E(X_{GPU}) - E(X_{CPU}) > 0$$

Ou seja, a diferença entre o valor médio de X_{GPU} e o valor médio de X_{CPU} é superior a zero. Então a hipótese nula (H_0) foi formulada com a situação que deseja-se não ser verdadeira: A diferença entre médias ser menor ou igual à zero. A hipótese alternativa (H_1) fica sendo a última equação mostrada, conforme abaixo:

$$H_0: E(X_{GPU}) - E(X_{CPU}) \leq 0$$

$$H_1: E(X_{GPU}) - E(X_{CPU}) > 0$$

Um teste adequado para esta situação é o *Teste t de Student para Diferença de Médias*. O nível de significância adotado (α) foi de 1%, o que significa dizer que teremos um intervalo de confiança ($1 - \alpha$) de 99% para o teste.

6.3 Cenário

Vale ressaltar que para que as medidas reflitam o real desempenho do software, é necessário que os testes sejam efetuados sobre uma carga saturada de pacotes. Caso contrário, em uma situação de pouco tráfego, a quantidade de pacotes a ser coletados será reduzida, reduzindo também a medida de desempenho. Isto resulta na falsa impressão de que o software está com um baixo desempenho. Para não correr este risco, o tráfego de entrada para o software será realizado através da leitura de pacotes de um arquivo de trace, com tamanho aproximado de 95 GB. Este trace contém pacotes capturados no período de um dia em uma grande operadora de telecomunicações brasileira, contendo tráfego de diversas aplicações.

Os testes, tanto com a versão CPU, quanto com a versão GPU, foram realizados em uma máquina com Processador Intel Core i7, 8GB de Memória RAM, e placa de vídeo NVIDIA GeForce GTX 4800 com 1536 MB de memória.

6.4 Resultados

Nesta seção, os resultados obtidos através do experimento são apresentados. Duas amostras foram obtidas através da coleta das métricas de desempenho na execução de cada versão do software sobre o mesmo conjunto de tráfego em um determinado período de tempo. As seções a seguir contêm análises descritivas de cada amostra e o resultado do teste de hipóteses aplicado. Lembrando que os gráficos a seguir representam valores da taxa de execução, ou seja, a quantidade de bytes por segundo que o software foi capaz de processar ao longo do tempo. É uma espécie de “velocímetro” do software.

6.4.1 Análise Descritiva da Primeira Amostra

A Figura 28 ilustra os dados da amostra coletada através da execução da versão que não utiliza GPU no cenário especificado. A métrica utilizada em todos os gráficos desta seção é MB/s (megabytes por segundo).

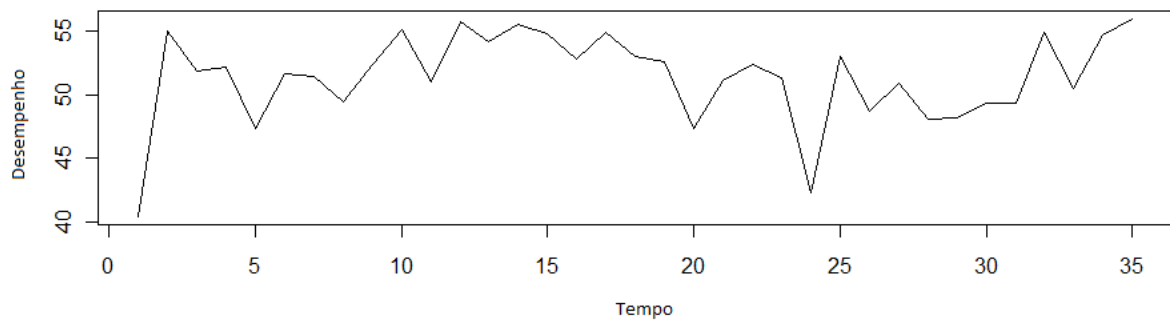


Figura 28 – Amostra do desempenho na execução da primeira versão.

A Tabela 3 resume os principais parâmetros da amostra, e inclui o resultado do *Teste de Shapiro-Wilk* aplicado à amostra. Este é um teste que fornece um indicativo da normalidade do conjunto de dados. Baixos valores obtidos neste tipo indicam forte tendência à normalidade.

Tabela 3 – Parâmetros da primeira amostra.

Parâmetro	Valor
Tamanho da Amostra	35
Máximo	55.91 MB/s
Mínimo	40.47 MB/s
Média	51.39 MB/s
Desvio-Padrão	3.53 MB/s
Variância	12.47 (MB/s) ²
Resultado (<i>p-value</i>) do Teste de Shapiro	0.003855

A Figura 29 exibe a função de densidade dos dados da amostra. Esta

função representa a distribuição de probabilidade do desempenho da aplicação.

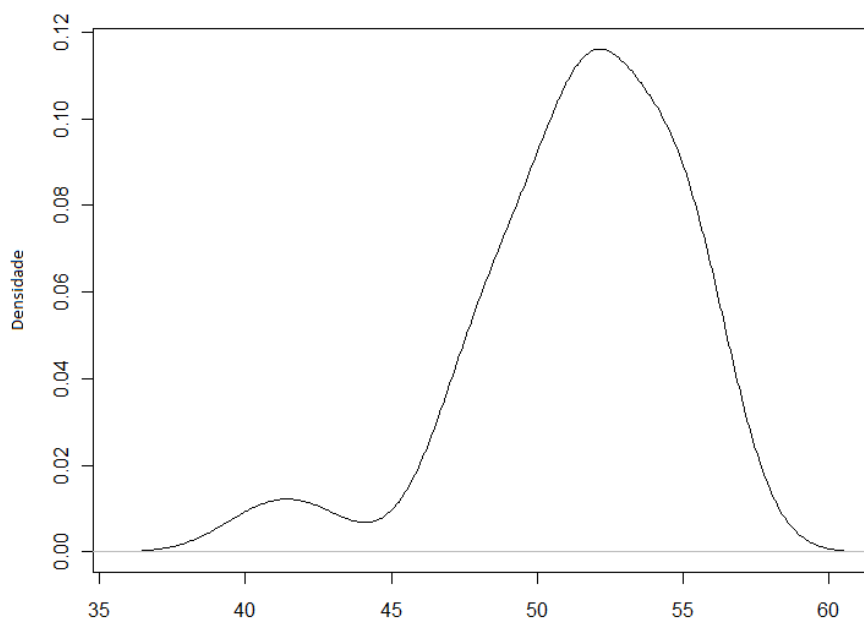


Figura 29 – Função de densidade da primeira amostra.

6.4.2 Análise Descritiva da Segunda Amostra

A Figura 30 ilustra os dados da amostra coletada através da execução da versão que utiliza GPU no cenário especificado.

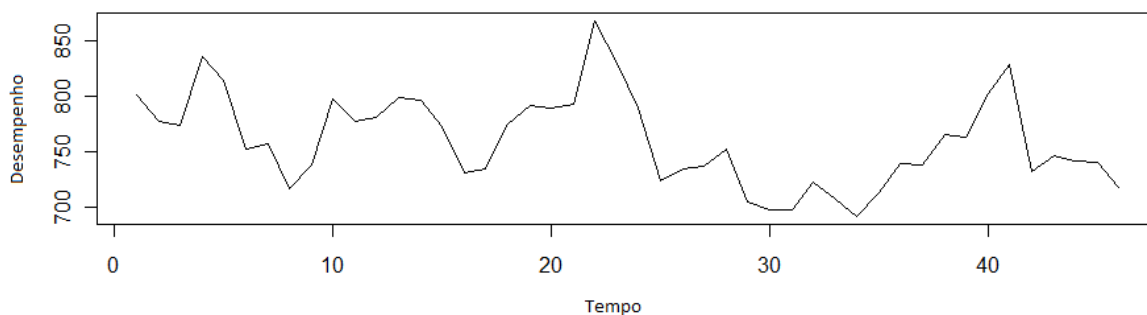


Figura 30 – Amostra do desempenho na execução da segunda versão.

A Tabela 4 resume os principais parâmetros da amostra, também incluindo o resultado do *Teste de Shapiro-Wilk*.

Tabela 4 – Parâmetros da segunda amostra.

Parâmetro	Valor
Tamanho da Amostra	46
Máximo	867.35 MB/s
Mínimo	692.65 MB/s
Média	760.79 MB/s
Desvio-Padrão	40.78 MB/s
Variância	1663.66 (MB/s) ²
Resultado (<i>p-value</i>) do Teste de Shapiro	0.3506

A Figura 31 exibe a função de densidade dos dados da amostra.

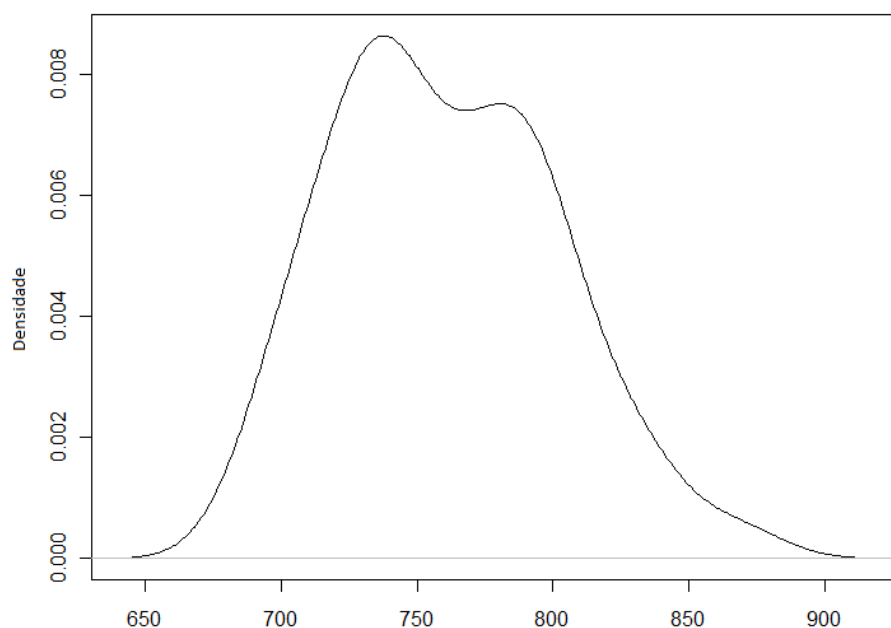


Figura 31 – Função de Densidade da Amostra GPU.

6.4.3 Resultado do Teste de Hipóteses

No Teste *t* de Student, deve-se calcular a média e a variância da distribuição que representa a diferença das variáveis aleatórias. Após este passo, calcula-se a estatística *t* e compara-se com o valor da tabela *t* de student, de acordo

com o grau de liberdade. Se a estatística for superior ao resultado da tabela, rejeita-se a hipótese nula.

Calculando a diferença das médias amostrais temos:

$$\mu = E(\bar{X}_{GPU}) - E(\bar{X}_{CPU})$$

$$\mu = 760.79 - 51.39$$

$$\mu = 709.4$$

E calculando a variância combinada da amostra, temos:

$$s = \sqrt{\frac{\bar{\sigma}_{GPU}^2}{N_{GPU}} + \frac{\bar{\sigma}_{CPU}^2}{N_{CPU}}}$$

$$s = \sqrt{\frac{1663.66}{46} + \frac{12.47}{35}}$$

$$s = \sqrt{36.16 + 0.35}$$

$$s = 6.04$$

E finalmente, calculando a estatística t, temos:

$$t = \frac{\mu}{s}$$

$$t = \frac{709.4}{6.04}$$

$$t = 117.45$$

Com 34 graus de liberdade (menor valor entre $N_{GPU} - 1$ e $N_{CPU} - 1$), e o alfa (α) em 1%, o valor da tabela t de student é 2,4411. Como a estatística t é bastante superior ao valor da tabela. A hipótese nula de que $X_{GPU} - X_{CPU} \leq 0$ pode ser rejeitada. Isto confirma o argumento do trabalho com uma margem de confiança de 99% de que o desempenho médio da versão do software que utiliza GPU é superior ao desempenho médio da versão que não utiliza GPU.

6.5 Discussão

O foco do trabalho foi demonstrar que a utilização de GPUs pode trazer benefícios a área de análise de tráfego. Apesar dos resultados, mesmo com a utilização de GPUs, não apresentarem velocidades de processamento na ordem dos gigabytes, o importante é notar que houve o ganho de desempenho, ou seja o *speed-up*, de uma versão para a outra. Um ganho em média de 15x mais desempenho.

Existe também o fato de que a maioria dos trabalhos relacionados envolvendo GPU não utiliza o recurso de armazenar informações em uma memória persistente como o banco de dados utilizado neste trabalho. Sistemas para roteamento de pacotes e emissão de alertas não precisam deste recurso, que neste trabalho foi necessário para fornecer uma visão em tempo real do tráfego da rede ao longo do tempo para a administração. Sem dúvida este foi o principal gargalo de desempenho encontrado no software. Talvez no futuro quando os bancos de dados aderirem às placas gráficas para acelerar sua execução, será possível refazer os testes levando isso em consideração, e com certeza haverá velocidades e ganhos maiores do que os apresentados neste trabalho.

7. CONCLUSÕES E TRABALHOS FUTUROS

O principal objetivo do trabalho foi a implementação de um software de análise de tráfego utilizando GPUs. Alguns conceitos básicos e o estado da arte no assunto foram abordados. O protótipo implementado é capaz de coletar métricas da rede e armazenar em um banco de dados para posterior análise por outras aplicações. O software desenvolvido foi detalhado incluindo sua estrutura e o comportamento, tanto na sua versão normal quanto na sua versão com GPU. A metodologia de avaliação do trabalho também foi detalhada. Os testes de desempenho estatísticos realizados indicaram que realmente há o ganho de desempenho considerando a versão que utiliza GPU.

7.1 Contribuições

Dentre as principais contribuições deste trabalho, destacam-se:

- Implementação de um software que utiliza GPUs para acelerar o desempenho na atividade de análise de tráfego em redes de computadores, podendo servir como base para outros trabalhos que desejem utilizar GPU em suas atividades;
- Detalhamento de uma metodologia para avaliação de desempenho baseada em testes de hipóteses estatísticos, que pode ser executada sempre que se deseja checar uma melhoria de desempenho entre duas versões de um software, independentes se estes trabalham ou não com GPUs;
- Contribuição para a melhoria no suporte à atividade de administração de redes de computadores de altas velocidades

7.2 Dificuldades Encontradas

A principal dificuldade encontrada foi em relação à habilidade com a plataforma CUDA, já que esta possui peculiaridades relativas a processamento gráfico/paralelo, que não são triviais de se trabalhar. Problemas no código CUDA são difíceis de detectar, pelas ferramentas ainda não possuírem ainda boas opções para *debug* de código.

7.3 Trabalhos Futuros

Como trabalho futuro recomenda-se estudar a implementação de algumas

otimizações no código CUDA, específico para a análise de tráfego. É necessário um estudo para saber quais os recursos das GPUs, como por exemplo, memórias compartilhadas e texturas, que melhores se aplicam ao cenário. Isto poderia trazer uma maior eficiência e desempenho para o software.

Outra sugestão seria a implementação da parte do software responsável pela persistência também em GPU. Isso eliminaria o principal gargalo que consiste na persistência dos dados. Questões de armazenamento e sincronização dos dados a serem persistidos entrariam neste estudo. Isso aumentaria significativamente o desempenho.

Recomenda-se também o estudo de como organizar múltiplas GPUs para realizar a atividade da análise de tráfego. Aqui seria necessário estudar como dividir a carga a ser processada entre as GPUs e como correlacionar posteriormente o resultado de cada unidade participante. Esta seria uma abordagem que também agregaria bastante desempenho ao software.

8. REFERÊNCIAS

- Acustica. *Nebula3*. 2009.
<http://www.soundonsound.com/sos/feb08/articles/nebula3.htm> (acesso em 22 de 6 de 2010).
- Aho, V, e J Corasick. "Efficient string matching: An aid to bibliographic search." *Comm. of the ACM* 18, n. 6 (1975): 333-340.
- Ansari, S., S. G. Rajeev, e H.S. Chandrashekar. "Packet sniffing: a brief introduction." *Potentials, IEEE* 21, n. 5 (Janeiro 2002): 17-19.
- Baker, Z K, e V K Prasanna. "Time and area efficient pattern matching on FPGAs." *ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA '04)*. 2004.
- Bakkum, Peter, e Kevin Skadron. "Accelerating SQL Database Operations on a GPU with CUDA." *GPGPU '10: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. Pittsburgh, Pennsylvania, 2010. 94-103.
- Becchi, M., M. Franklin, e P. Crowley. "A workload for evaluating deep packet inspection architectures." *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium*. 2008. 79-89.
- Buck, Ian, et al. "Brook for GPUs: stream computing on graphics hardware." *ACM Trans. Graph. (ACM)* 23, n. 3 (August 2004): 777-786.
- Cacti Group. 2006. <http://www.cacti.net/> (acesso em 14 de 07 de 2010).
- Cisco IOS NetFlow. 2002. <http://www.cisco.com/web/go/netflow/> (acesso em 6 de 10 de 2010).
- Combs, G. 2007. <http://www.wireshark.org/> (acesso em 14 de 07 de 2010).
- Commentz-Walter, B. "A String matching algorithm fast on the average." *ICALP*. 1979. 118-132.
- Cronin, Mary J. *Global Advantage on the Internet: From Corporate Connectivity to International Competitiveness*. 1995.
- Dharmapurikar, S, e J Lockwood. "Fast and scalable pattern matching for content filtering." *ACM symposium on Architecture for networking and communications systems (ANCS '05)*. 2005.
- Farias, Thiago, João Marcelo Teixeira, Gabriel Almeida, Veronica Teichrieb, e Judith Kelner. "Massively parallel computing techniques: a step towards real time high definition AR applications." *Symposium on Virtual and Augmented Reality*. João Pessoa, Brazil, 2008b. 21-27.
- Fernando, Randima, e Mark Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
- Fuchsberger, Andreas. "Intrusion Detection Systems and Intrusion Prevention Systems." *Inf. Secur. Tech. Rep.* 10, n. 3 (2005): 134-139.
- Garcia, Luis Martin. "Programming with Libpcap - Sniffing the Network From Our Own Application." *Hacking* 3, n. 2 (2008): 38-46.
- Goodall, R, G Lutters, P Rheingans, e A Komlodi. "Preserving the Big Picture: Visual Network Traffic Analysis with TNV." *Workshop on Visualization for*

- Computer Security (VizSec)*, 2005: 47-54.
- Han, Sangjin, Keon Jang, KyoungSoo Park, e Sue Moon. "PacketShader: a GPU-accelerated software router." *SIGCOMM Comput. Commun. Rev.* 40, n. 4 (2010): 195-206.
- Han, Se-Hee, Myung-Sup Kim, Hong-Taek Ju, e James Won-Ki Hong. "The Architecture of NG-MON: A Passive Network Monitoring System for High-Speed IP Networks." *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications*. 2002. 16-27.
- Hernandez-Campos, F, B Nobel, D Smith, e K Jeffay. "Statistical Clustering of Internet Communication Patterns." *Computing Science and Statistics*, 2003.
- Hunt, Craig. *TCP/IP network administration (2nd ed.)*. 1998.
- Jacobsen, V., C. Leres, e S. McCanne. 2005. <http://www.tcpdump.org/> (acesso em 14 de 07 de 2010).
- Kempke, A, e J McAuley. Ed. Inc. Motorola. United States Patent Patente 5,841,874. 1998.
- Khronos. *The OpenCL Specification, Version 1.0*. April de 2009. <http://www.khronos.org/registry/cl/specs/opencl-1.0.33.pdf> (acesso em 2010 de 6 de 22).
- Kozierok, Charles. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. 2005.
- Kumar, Sailesh, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, e Jonathan Turner. "Algorithms to accelerate multiple regular expressions matching for deep packet inspection." *SIGCOMM Comput. Commun. Rev.* 36, n. 4 (2006): 339-350.
- Larson, S.M., C.D. Snow, M.R. Shirts, e V.S. Pande. "Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology." *Computational Genomics* (Horizon Press), 2002.
- Leite, Pedro, João Teixeira, Thiago Farias, Veronica Teichrieb, e Judith Kelner. "Massively Parallel Nearest Neighbor Queries for Dynamic Point Clouds on the GPU." *SBAC-PAD '09: Proceedings of the 2009 21st International Symposium on Computer Architecture and High Performance Computing*. São Paulo, Brazil, 2009. 19-25.
- Luebke, David, e Greg Humphreys. "How GPUs Work." *Computer* 40, n. 2 (2007): 96-100.
- McGregor, A, M Hall, P Lorier, e J Brunskill. "Flow Clustering Using Machine Learning Techniques." *PAM*. 2004.
- Microsoft. *DirectCompute*. 2009. <http://www.microsoft.com/windows/directx/> (acesso em 22 de 6 de 2010).
- . *HLSL*. 2006. [http://msdn.microsoft.com/en-us/library/bb509561\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bb509561(VS.85).aspx) (acesso em 22 de 6 de 2010).
- . *Microsoft Visual Studio*. 2005. <http://msdn.microsoft.com/en-us/vstudio/default.aspx> (acesso em 22 de 6 de 2010).

- Moore, A, e K Papagiannaki. "Toward the Accurate Identification of Network Applications." *PAM*. 2005.
- Moore, W, e D Zuev. "Internet Traffic Classification Using Bayesian Analysis Techniques." *ACM SIGMETRICS*. 2005.
- Nash, Kim S. *Network Monitoring Definition and Solutions*. 2009.
http://www.cio.com/article/133700/Network_Monitoring_Definition_and_Solutions (acesso em 20 de 01 de 2011).
- Nottingham, Alastair, e Barry Irwin. "GPU packet classification using OpenCL: a consideration of viable classification methods." *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. 2009. 160-169.
- NVIDIA. *Nexus*. 2010a. <http://developer.nvidia.com/object/nsight.html> (acesso em 22 de 6 de 2010).
- . *NVIDIA CUDA Programming Guide*. 2009a.
http://www.serc.iisc.ernet.in/ComputingFacilities/systems/Tesla_Doc/NVIDIA_CUDA_Programming_Guide_2.3.pdf (acesso em 20 de 05 de 2010).
- . *PhysX*. 2009b. http://www.nvidia.com.br/object/physx_new.html (acesso em 22 de 6 de 2010).
- . *PureVideo*. 2010b. <http://www.nvidia.com/page/purevideo.html> (acesso em 21 de 6 de 2010).
- Oetiker, T. 2004. <http://oss.oetiker.ch/rrdtool/> (acesso em 14 de 07 de 2010).
- Oetiker, Tobias. "MRTG: The Multi Router Traffic Grapher." *Proceedings of the 12th Conference on Systems Administration*. 1998. 141-148.
- Owens, J., M. Houston, D. Luebke, S. Green, J. Stone, e J. Phillips. "GPU Computing." *Proceedings of the IEEE* 96, n. 5 (2008): 879-899.
- Owens, John, et al. "A Survey of General-Purpose Computation on Graphics Hardware." *Computer Graphics Forum* 26, n. 1 (2007): 80-113.
- Parcens, e Lopht. *NT IIS Showcode ASP Vulnerability*. 1999.
<http://www.securityfocus.com/bid/167/info> (acesso em 22 de 01 de 2011).
- Richardson, R. "CSI Computer Crime and Security Survey." *Computer Security Institute*, 2008: 1-31.
- Rost, Randi. *OpenGL(R) Shading Language*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.
- Roughan, M, S Sen, O Spatscheck, e N Duffield. "Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification." *ACM/SIGCOMM IMC*. 2004.
- Roxio. *Roxio Creator 2010*. 2010.
<http://www.roxio.com/enu/products/creator/suite/> (acesso em 21 de 06 de 2010).
- Santos, Artur, João Teixeira, Thiago Farias, Veronica Teichrieb, e Judith Kelner. "kD-Tree Traversal Implementations for Ray Tracing on Massive Multiprocessors: A Comparative Study." *SBAC-PAD '09: Proceedings of the 2009 21st International Symposium on Computer Architecture and High Performance Computing*. Sao Paulo, Brazil, 2009. 41-48.
- Shirley, Peter. *Fundamentals of Computer Graphics*. 2º Edition. Natick, MA, USA:

- A. K. Peters, Ltd., 2005.
- Shomura, Yusuke, Yoshinori Watanabe, e Naoya Ikeda. "A Traffic Monitoring Method for High Speed Networks." *Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet*. 2009. 107-113.
- Snort. *The Open Source Network Intrusion Detection System*. 2011.
<http://www.snort.org/> (acesso em 18 de 03 de 2011).
- Spiznagel, E, D Taylor, e J Turner. "Packet Classification Using Extended TCAMs." *IEEE International Conference on Network Protocols (ICNP)*. 2003.
- Stallings, William. *SNMP, SNMPV2, Snmpv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998.
- Tanenbaum, Andrew Stuart. *Redes de Computadores*. 2003.
- Tarditi, David, Sidd Puri, e Jose Oglesby. "Accelerator: using data parallelism to program GPUs for general-purpose uses." *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. San Jose, California, USA: ACM, 2006. 325-335.
- Taylor, E, e S Turner. "Scalable Packet Classification using Distributed Crossproducting of Field Labels." *Tech. Rep. WUCSE-2004-38, Department of Computer Science and Engineering, Washington University in Saint Louis*, 2004.
- van Lunteren, J, e T Engbersen. "Fast and scalable packet classification." *IEEE Journal on Selected Areas in Communications* 21 (2003): 560-571.
- Vasiliadis, Giorgos, Spiros Antonatos, Michalis Polychronakis, Evangelos Markatos, e Sotiris Ioannidis. "Gnort: High Performance Network Intrusion Detection Using Graphics Processors." In: *Recent Advances in Intrusion Detection*, por R. Lippmann, E. Kirda e A. Trachtenberg, 116-134. Springer Berlin / Heidelberg, 2008.
- Vespa, Lucas, e Ning Weng. "Deterministic finite automata characterization and optimization for scalable pattern matching." *ACM Trans. Archit. Code Optim.*, 2011: 4:1-4:31.
- Wu, S, e U Mamber. "A fast algorithm for multi-pattern searching." *Tech. R. TR-94-17, Dept. of Comp. Science. Univ. of Arizona.*, 1994.
- Yu, F., R. H. Katz, e T. V. Lakshman. "Gigabit Rate Packet Pattern-Matching Using TCAM." *12th IEEE International Conference on Network Protocols (ICNP '04)*. 2004.