UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA
ELÉTRICA E DE COMPUTAÇÃO

Identificação Remota de Sistemas Operacionais Utilizando Análise de Processos Aleatórios e Redes Neurais Artificiais

João Paulo de Souza Medeiros

Orientador: Prof. Dr. Agostinho de Medeiros Brito Júnior

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de ordem PPgEEC: M232
Natal, RN, junho de 2009

Divisão de Serviços Técnicos

Catalogação da publicação na fonte. UFRN / Biblioteca Central Zila Mamede

Medeiros, João Paulo de Souza.

Identificação remota de sistemas operacionais utilizando análise de processos aleatórios e redes neurais artificiais / João Paulo de Souza Medeiros - Natal, RN, 2009

103 f.

Orientador: Agostinho de Medeiros Brito Júnior.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-graduação em Engenharia Elétrica e de Computação.

1. Sistemas operacionais – identificação remota – Dissertação. 2. TCP/IP – Identificação de pilhas – Dissertação. 3. Seguranças em redes industriais – Dissertação. 4. Identificação de Honeypots – Dissertação. I. Brito Júnior, Agostinho de Medeiros. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM CDU 004.451(043.3)

Identificação Remota de Sistemas Operacionais Utilizando Análise de Processos Aleatórios e Redes Neurais Artificiais

João Paulo de Souza Medeiros

Dissertação de Mestrado aprovada em 24 de junho de 2009 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Agostinho de Medeiros Brito Júnior (orientador) ..... DCA/UFRN

Prof. Dr. Eduardo Bráulio Wanderley Netto ..... IFRN

Prof. Dr. Adrião Duarte Dória Neto ..... DCA/UFRN

Prof. Dr. Jorge Dantas de Melo ..... DCA/UFRN

Prof. Dr. Paulo Sérgio da Motta Pires ..... DCA/UFRN

Aos meus pais, amigos e minha companheira, pela confiança, suporte e paciência em mim depositados.

Agradecimentos

Ao meu orientador, Prof. Agostinho de Medeiros Brito Júnior, sou grato pela orientação e por todo o precioso conhecimento a mim repassado. Conste que este cumpriu seu papel com excelência e de forma amigável. Sou orgulhoso por ter sido seu primeiro orientado.

Ao Prof. Paulo Sérgio da Motta Pires, pelo incentivo, compreensão e apoio. Sua exigência ajuda a descobrirmos de que somos feitos e do quão somos capazes. Sou grato também pelos valiosos conselhos e oportunidades, além do exemplo de profissionalismo.

Aos Professores Adrião Duarte Dória Neto e Jorge Dantas de Melo pela clara demonstração de como conduzir a transferência do conhecimento de professor para aluno.

À minha mãe, Maria Nerivan de Souza Medeiros, pela benção e incentivo, e por ter mostrado a mim que mãe é definição de esforço, determinação e lealdade.

Ao meu pai, Josias Martinho de Medeiros. Por me influenciar com seus dons da paciência, superação e discernimento, além do apoio e suporte indispensáveis.

Ao meu irmão, Luiz Paulo de Souza Medeiros, pela presença enriquecedora e por compartilhar de forma não linear os pontos críticos desta jornada.

À minha companheira, Graciele Saionara Linhares de Lima, por sempre me ajudar a recuperar as forças e refletir sobre meus objetivos. Sua atenção e compreensividade foram fundamentais.

Aos meus amigos e colegas de trabalho, Aguinaldo Bezerra Batista Junior, José Macedo Firmino Filho e Tiago Hiroshi Kobayashi, pelos comentários e sugestões pertinentes e pelas horas de trabalho compartilhadas.

Ao meu amigo, João Batista Borges Neto, pelo apoio e discussões esclarecedoras, além do claro exemplo de coerência e objetividade.

Ao grupo de desenvolvedores do Umit (especialmente ao Adriano Monteiro Marques e ao Luís A. Bastião Silva) pelo enriquecedor envolvimento no meu trabalho e pela oportunidade de inserção no mundo do desenvolvimento de código aberto.

Ao grupo de desenvolvedores do Nmap (especialmente ao Gordon “Fyodor” Lion) pelos comentários, oportunidades e conhecimento ofertados.

À Petrobras e a REDIC pelo suporte financeiro e ao LabSIN pela disponibilidade de equipamentos e local de trabalho.

Resumo

É proposto um novo método para identificação remota de sistemas operacionais que operam em redes TCP/IP. Este método possui diversas aplicações relacionadas à segurança em redes de computadores e é normalmente adotado tanto em atividades de ataque quanto de defesa de sistemas. O método proposto é capaz de obter sucesso em situações onde diversas soluções atuais falham, inclusive no tratamento com dispositivos possivelmente vulneráveis ao processo de identificação. O novo método realiza a análise dos geradores de números aleatórios usados nas pilhas TCP/IP e, através do uso de redes neurais artificiais, cria mapas que representam o comportamento destes geradores. Tais mapas são usados para comparação com mapas rotulados que representam sistemas já conhecidos, concretizando o processo de identificação.

Palavras-chave: Identificação Remota de Sistemas Operacionais, Identificação de Pilha TCP/IP, Segurança em Redes de Computadores, Identificação de Honeypots.

Abstract

A new method to perform TCP/IP fingerprinting is proposed. TCP/IP fingerprinting is the process of identify a remote machine through a TCP/IP based computer network. This method has many applications related to network security. Both intrusion and defence procedures may use this process to achieve their objectives. There are many known methods that perform this process in favorable conditions. However, nowadays there are many adversities that reduce the identification performance. This work aims the creation of a new OS fingerprinting tool that bypass these actual problems. The proposed method is based on the use of attractors reconstruction and neural networks to characterize and classify pseudo-random numbers generators.

Keywords: OS fingerprinting, TCP/IP fingerprinting, Network Security, Honeypots Indentification.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Exemplos

Lista de Algoritmos

Glossário

Convenções

1 – A utilização de letras minúsculas em negrito em equações, denotam vetores ou variáveis aleatórias (por exemplo, **x**). A utilização de letras maiúsculas em negrito, denotam matrizes (por exemplo **A**).

2 – Números precedidos por `0x` estão representados na base hexadecimal.

3 – A média amostral $\overline{\mu}_{\mathbf{x}}$ apresentada em vários pontos deste documento é calculada como segue:

$$\overline{\mu}_{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

A variância $\sigma^2_{\mathbf{x}}$ é calculada com base na média amostral $\overline{\mu}_{\mathbf{x}}$ e é definida como segue:

$$\sigma^2_{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \overline{\mu}_{\mathbf{x}})^2.$$

Em ambas as equações apresentadas, N indica o número de amostras da variável aleatória **x** e x_i a i-ésima amostras da mesma variável aleatória.

4 – Em formulações do tipo $\|\mathbf{a} - \mathbf{b}\|$ assumir que a métrica utilizada é a distância Euclidiana, a menos que, seja indicado o contrário.

5 – O conteúdo das mensagens dos protocolos de rede é apresentado de forma similar ao encontrado no exemplo a seguir:

```
1  00 00 BC 35  7A 8B 00 16  76 21 A1 16  08 00 45 00      ...5z... v!....E.
2  00 62 1E 35  40 00 80 06  5A DD C0 A8  00 32 C0 A8      .b.5@... Z....2..
```

Esta ilustração deve ser interpretada da seguinte forma: do lado esquerdo, exterior ao quadro tem-se a numeração de blocos de 128 *bits* (ou 16 *bytes*). Ainda do lado esquerdo, porém na parte interior ao quadro, tem-se o conteúdo da mensagem. Finalmente do lado direito é ilustrado o conteúdo no formato ASCII, quando o valor correspondente na mensagem pode ser representado, caso contrário, um ponto (“.”) é utilizado.

Siglas

ADSL	*Asymmetric Digital Subscriber Line*
ASCII	*American Standard Code for Information Interchange*
DDoS	*Distributed Denial of Service*
DNS	*Domain Name System*
DoS	*Denial of Service*
EOL	*End of Line*
FTP	*File Transfer Protocol*
GCD	*Greatest Common Divisor*
GNG	*Growing Neural Gas*
HTTP	*HyperText Transfer Protocol*
IANA	*Internet Assigned Numbers Authority*
ICMP	*Internet Control Message Protocol*
IDS	*Intusion Detection System*
IP	*Internet Protocol*
ISN	*Initial Sequence Number*
LAN	*Local Area Network*
NAT	*Network Address Translation*
NAPT	*Network Address Port Translation*
MDC	Máximo Divisor Comum
OS	*Operating System*
PPPoE	*Point-to-Point Protocol over Ethernet*
PRNG	*Pseudo-Random Number Generator*
RFC	*Request for Comments*
SO	Sistema Operacional
SOM	*Self-Organizing Map*
PAT	*Port Address Translation*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
VoIP	*Voice over Internet Protocol*
WLAN	*Wireless Local Area Network*

Simbologia

$\mu_{\mathbf{x}}$	Média da variável aleatória $\mathbf{x}$
$\overline{\mu}_s$	Média amostral de s
σ_s	Desvio padrão de s
σ_s^2	Variância de s
$\hat{f}(x)$	Estimativa da função $f(x)$
$\overline{XY}$	Plano composto pelos eixos X e Y

Capítulo 1

Introdução

A identificação remota de sistemas operacionais, ou OS *fingerprinting* (*Operating System fingerprinting*), é um processo que tem como finalidade a descoberta do sistema operacional de uma máquina remota. Entenda-se por remota uma máquina que é acessível por meio de uma rede de computadores. Esta identificação é realizada a partir da utilização de dados originados da máquina remota. As técnicas usadas para esse fim diferem entre si de acordo com os dados que eles utilizam e com a forma como esses dados são adquiridos. A forma com que estes dados são adquiridos depende da aplicação fim da identificação. Neste capítulo, serão apresentadas as aplicações e será definido todo o processo de identificação, além dos requisitos a serem considerados no projeto de uma ferramenta. Alguns métodos clássicos de identificação também são exemplificados.

1.1 Aplicações e motivação

Diversas são as aplicações do processo de OS *fingerprinting*. Geralmente estão relacionadas ao gerenciamento e segurança de redes de computadores baseadas na pilha de protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*) e podem estar envolvidas tanto em procedimentos de ataques (ou intrusão) quanto de gerenciamento e proteção de redes de computadores. Porém, os próprios procedimentos de intrusão podem ser utilizados pelos gerentes de rede a fim de descobrir a vulnerabilidades da rede e tomar as devidas precauções.

1.1.1 Processo seletivo de intrusão

As motivações que levam um indivíduo a invadir computadores são muitas vezes desconhecidas. Porém, o procedimento para realizar esta tarefa possui etapas bem definidas, apresentadas na Tabela 1.1. Nessa tabela, o procedimento de intrusão é dividido em 6 etapas distintas, sendo que cada uma delas depende de alguma forma da anterior. Nem sempre todas as etapas estão presentes, mas algumas delas são obrigatórias, pois sem elas os parâmetros necessários para a realização do ataque não seriam conhecidos. Mesmo assim, em alguns casos, etapas importantes são descartadas por serem passíveis de identificação, dando à máquina alvo a chance de se defender do possível ataque. A não execução das

etapas 3, 4, ou 5 da Tabela 1.1 faz com que um ataque seja feito as cegas, implicando em um número maior de tentativas e, conseqüentemente, em uma perda de tempo maior.

Nº	Etapa	Motivação
1	Escolher alvo	Pessoal, profissional, etc.
2	Identificar alvo	Descobrir endereço do alvo (Internet).
3	*Port-scan*	Descobrir portas abertas.
4	*Service fingerprinting*	Identificar serviços.
5	OS *fingerprinting*	Identificar sistema operacional.
6	Atacar alvo	Utilizar programas que exploram vulnerabilidades.

Tabela 1.1: Etapas de um procedimento de intrusão.

Para ilustrar uma situação básica de intrusão, considere um exemplo em que um atacante tem como objetivo encontrar máquinas na Internet que possuam as portas 80/TCP e 22/TCP abertas e que executem algum sistema operacional baseado em Unix. Neste sentido, o atacante utiliza um programa que tenta identificar essas duas portas abertas em endereços quaisquer na Internet. O interesse na porta 80/TCP está relacionado a uma vulnerabilidade de uma versão especifica de um servidor HTTP (*HyperText Transfer Protocol*) que possibilita acesso a arquivos protegidos do sistema. Sistemas operacionais baseados em Unix possuem um arquivo localizado em `/etc/shadow` ou `/etc/passwd` que possui os nomes de usuários e suas senhas encriptadas. O atacante quer, de posse desse arquivo, desvendar as senhas dos usuários utilizando ferramentas adequadas. Ao desvendar uma senha, o atacante poderá utilizá-la para obter acesso à máquina remota através do serviço SSH (*Secure Shell*) disponível pela porta 22/TCP.

Neste exemplo, as etapas de escolha do alvo e sua identificação na Internet são realizados de forma aleatória. Porém, as etapas de *port-scan*, *service* e OS *fingerprinting* foram utilizadas e são primordiais para desempenhar o ataque em questão. *Port-scan* consiste na tarefa de procurar por portas abertas em uma máquina remota [Fyodor 1997]. Mesmo após descobrir que as portas almejadas estão abertas, o atacante tem que verificar se a versão da aplicação que oferece o serviço associado àquela porta é vulnerável. Esta tarefa, denominada *service fingerprinting*, ou identificação de serviço, é normalmente realizada através de técnicas de inspeção de protocolos [Fyodor 2009*b*]. Nesse exemplo particular, a detecção do sistema operacional (OS *fingerprinting*) da máquina remota teve como finalidade descartar máquinas que provavelmente não possuiriam os arquivos desejados.

1.1.2 Auxílio na exploração de vulnerabilidade

Mesmo quando é descoberta uma vulnerabilidade que é passível de exploração, é preciso saber qual sistema operacional hospeda o serviço vulnerável. Isto se deve ao fato de que, para explorar vulnerabilidades como estouro de pilha, ou *buffer overflow*, é geralmente necessária a utilização de código em *assembly*, que por sua vez, é dependente da arquitetura e do sistema operacional utilizado na máquina vulnerável. É comum o serviço deixar de operar normalmente se o código *assembly* enviado não for o adequado.

Caso o atacante conheça o sistema operacional do alvo, este contratempo é evitado, dando a chance de explorar a vulnerabilidade e e de não ter que que esperar a reativação do serviço para tentar invadir novamente [Fyodor 1998, Fyodor 2006, Fyodor 2009*b*].

1.1.3 Engenharia social

Saber quais dispositivos são utilizados pela organização que se pretende atacar pode também facilitar o processo de invasão. Considere o papel de um atacante que tem como objetivo obter uma lista das ligações VoIP (*Voice over Internet Protocol*) originadas e recebidas de uma determinada empresa. Ao realizar OS *fingerprinting* nas máquinas da rede de uma empresa ele encontra um *gateway* VoIP. De posse dessa informação o atacante cria uma atualização maliciosa para esse dispositivo que envia uma mensagem de texto para um e-mail pré-definido com a lista de chamadas realizadas na última hora. Logo após, o atacante contacta o setor de gerência do sistema VoIP da empresa dizendo ser um engenheiro da empresa que fabrica o *gateway* VoIP encontrado e que foi detectada uma falha de segurança no *firmware* do equipamento em questão. Em seguida o atacante pergunta para qual endereço de e-mail ele deve enviar o programa de atualização do dispositivo. Convencido pelas informações fornecidas pelo atacante, o administrador resolve repassar o endereço para envio da atualização e posteriormente instalá-la [Fyodor 2009*b*].

1.1.4 Determinação de máquinas vulneráveis

Eventualmente, são descobertas falhas de segurança em serviços básicos instalados por padrão em alguns sistemas operacionais. Em dezembro de 2004, por exemplo, a Sun publicou um alerta (Sun Alert 57659) que descreve um problema de segurança no serviço `in.rwhod` presente no sistema operacional Solaris. Para solucionar esse problema é necessário aplicar uma correção que remove a vulnerabilidade.

Suponha que em uma determinada rede de servidores de uma grande empresa de serviços de hospedagem existem dezenas de computadores que utilizam diferentes versões desse sistema operacional. Sabe-se que a vulnerabilidade é presente nas versões 7, 8 e 9 do Solaris.

Caso o administrador dessa rede opte por verificar uma máquina de cada vez, ele pode desperdiçar um tempo considerável com máquinas que possuam uma versão 10 ou uma versão mais antiga do Solaris. Enquanto o administrador verifica cada um de seus servidores, um atacante, que acaba de escrever um programa malicioso que provê acesso de superusuário às máquinas com a vulnerabilidade publicada, pode invadir um de seus servidores. O administrador pode evitar um desperdício de tempo substancial se utilizar OS *fingerprinting* para descobrir quais de seus servidores utilizam as versões afetadas do sistema operacional [Fyodor 2009*b*].

1.1.5 Inventário e detecção de dispositivos não-autorizados

Em redes controladas e com políticas de segurança rígidas podem existir restrições relacionadas à utilização de sistemas operacionais considerados inseguros. Por exem-

plo, em uma determinada rede é autorizada apenas a utilização de sistemas operacionais Linux, cuja versão do *kernel* seja igual ou superior à 2.6. Para garantir que essa restrição está sendo atendida, é possível agendar periodicamente a identificação remota de todos os sistemas operacionais da rede em questão [Fyodor 2009*b*].

Em outros casos também é necessário conhecer os sistemas operacionais presentes na rede. Por exemplo, uma organização passa por um período de transição relacionada à substituição do sistema operacional das máquinas de seu funcionários. Para estimar a adaptação dos funcionários, o administrador realiza OS *fingerprinting* diariamente e verifica quantos funcionários já estão utilizando o sistema operacional novo. De acordo com esse inventário diário, é possível acompanhar o processo de transição até o momento em que não seja mais necessário utilizar o SO antigo.

1.2 Funcionamento, componentes e subprocessos

Apesar da quantidade considerável de técnicas desenvolvidas, todas elas envolvem dois subprocessos em comum, denominados neste trabalho de *caracterização* e *classificação*. O processo de OS *fingerprinting*, ilustrado na Figura 1.1, possui quatro componentes e dois subprocessos. Nessa figura é ilustrado que o processo de OS *fingerprinting* é dividido em dois subprocessos, denominados *caracterização* e *classificação* (*fingerprinting* e *matching*).

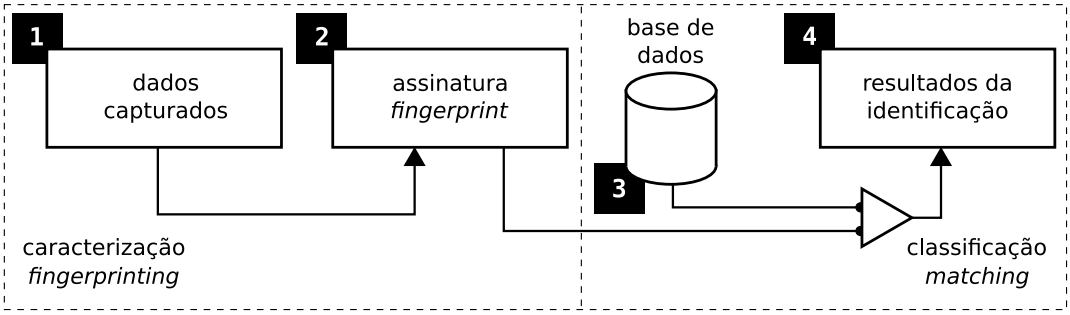

Figura 1.1: Representação gráfica do processo de OS *fingerprinting*.

A primeira etapa do processo de identificação remota de sistemas operacionais consiste na aquisição dos dados que caracterizam a máquina remota. De acordo com a forma como estes dados são criados e posteriormente capturados, pode-se classificar os métodos que realizam OS *fingerprinting* em dois tipos:

- **Ativos** (*active* OS *fingerprinting*): a máquina que realiza a identificação envia mensagens para a máquina remota. As respostas as estas mensagens são capturadas e utilizadas no processo de identificação;

- **Passivos** (*passive* OS *fingerprinting*): a máquina que realiza a identificação não emite mensagens na rede. Os dados relacionados a máquina remota são capturados quando a máquina remota se comunica com uma terceira máquina.

Para ilustrar o funcionamento dos dois tipos de OS *fingerprinting* é apresentado um protocolo simplista de troca de mensagens, cujo o formato da mensagem e os dispositivos da rede utilizada no exemplo são ilustrados na Figura 1.2.

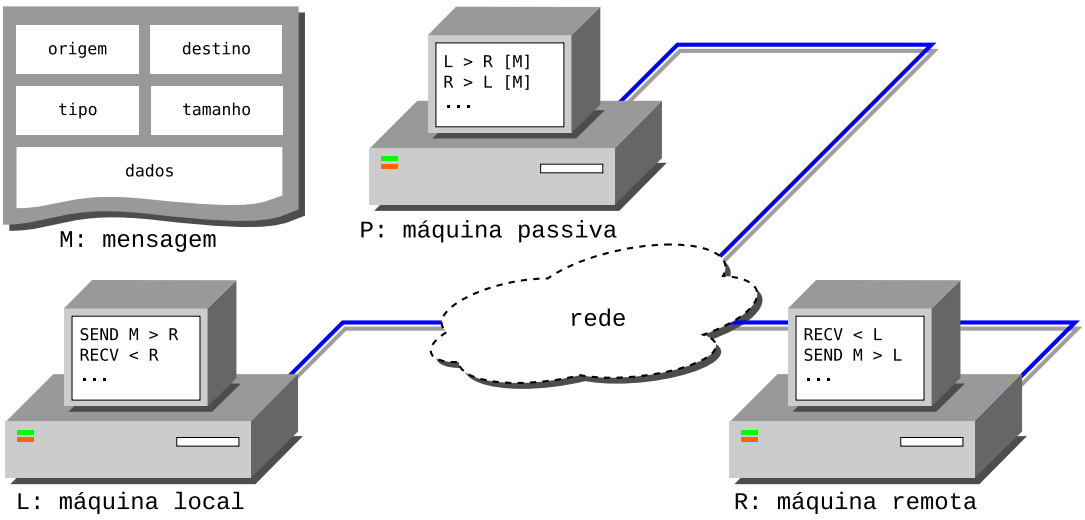

Figura 1.2: Ilustração dos dois tipos de aquisição de dados.

Na rede apresentada na Figura 1.2, é utilizado o formato de mensagem `M` (representado no canto superior esquerdo da mesma figura) para troca de dados. Os quatros primeiros campos possuem um tamanho equivalente a um *byte* (oito *bits*). Os dois primeiros campos da mensagem indicam os endereço da máquina de origem e da máquina de destino, respectivamente. O tipo da mensagem indica qual sua finalidade. São três os tipos de mensagem: requisição (`0x00`), controle (`0x01`) e transmissão (`0x02`). O campo relativo ao tamanho da mensagem indica o tamanho do campo de dados.

Dessa forma, adota-se um exemplo em que a máquina `L` (cuja representação em hexadecimal de acordo com a tabela ASCII é `0x4C`) deseja enviar um texto para a máquina `R` (`0x52`). De acordo com o protocolo da mensagem `M`, antes de enviar a mensagem com o texto deve-se (por meio de uma mensagem de controle) descobrir qual o conjunto de caracteres utilizados para indicar o final de uma linha, ou EOL (*End of Line*). Tal requisito se deve ao fato de que diferentes SOs utilizam caracteres específicos para este fim.

Nesse sentido, a máquina `L` envia através da rede a mensagem ilustrada no Exemplo 1.1. Pode-se notar que o campo de dados da mensagem contém o conteúdo "`EOL?`". Em acordo com o protocolo hipotético utilizado nesse exemplo, está é a forma da máquina `L` perguntar a máquina `R` qual o indicador de fim de linha por ela utilizado.

```
1  4C 52 01 04  45 4F 4C 3F                                  LR..EOL?
```

Exemplo 1.1: Mensagem enviada pela máquina `L` para a máquina `R`.

Em resposta, a máquina `R` envia a mensagem ilustrada no Exemplo 1.2. Pode-se notar que o conteúdo da resposta é caracterizado pelos últimos cinco *bytes* da mensagem, qual seja, "`EOL=\n`" (o *byte* `0x0A` equivale ao caractere especial "`\n`").

```
1 | 52 4C 01 05   45 4F 4C 3D   0A                                  LR..EOL= .
```

Exemplo 1.2: Resposta enviada pela máquina R para a máquina L.

O que se deseja demonstrar é que, para obter a informação do caractere utilizado para delimitar uma linha na máquina R, foi necessário enviar a mensagem de controle apresentada no Exemplo 1.1. Nesse caso, se a máquina L enviou tal mensagem com o objetivo de estimar o SO da máquina R, diz-se que a máquina L realizou OS *fingerprinting* ativo. Por outro lado, caso a máquina P ficasse monitorando a rede, também poderia obter a mesma informação em relação ao caractere indicador de final de linha da máquina R. Nesse caso, dizemos que a máquina P realizou OS *fingerprinting* passivo. Supera-se, assim, o primeiro ponto do processo de identificação remota de sistemas operacionais.

A etapa seguinte consiste no subprocesso de caracterização. Nessa etapa, é necessária a elaboração de um método que crie uma representação de um dado SO com base nos dados que esse último propaga pela rede. Esta representação é denominada assinatura (*fingerprint*). O termo *fingerprint*, em computação, é definido como uma assinatura que tem como finalidade representar uma dada informação de forma única [Broder 1993]. No Algoritmo 1.1, é descrito o procedimento que representa o exemplo de subprocesso de caracterização para realizar OS *fingerprinting* ativo.

requer: r {endereço da máquina remota}
requer: c {controlador de acesso a rede}
1: $s \leftarrow c.enviar(r, \texttt{MSG\_CONTROLE\_EOL\_PERGUNTA})$
2: **se** s = MENSAGEM_ENVIADA **então**
3: $m \leftarrow c.receber(r, \texttt{MSG\_CONTROLE\_EOL\_RESPOSTA})$
4: $a \leftarrow m.dados$
5: **else**
6: $a \leftarrow$ NULO
7: **fim se**
retorna: a {assinatura de r}

Algoritmo 1.1: Exemplo de procedimento de caracterização de OS *fingerprinting* ativo.

Como pode ser visto no Algoritmo 1.1, o subprocesso de caracterização, para realização de OS *fingerprinting* ativo, é composto de três passos básicos: enviar dados, coletar dados e criar assinatura. No Algoritmo 1.2, é ilustrado o procedimento de caracterização para realização de OS *fingerprinting* passivo. Comparando o Algoritmo 1.1 com o 1.2 pode-se notar que a diferença entre ambos está relacionada à não geração de tráfego de rede pelo método passivo. Por esse motivo, métodos que realizam OS *fingerprinting* passivo têm a vantagem de não serem detectáveis, ao contrário dos métodos ativos, que quase sempre podem ser detectados. Por outro lado, os métodos ativos não precisam esperar que as mensagens necessárias para realizar a caracterização sejam enviadas à rede, pois essas mensagens são incitadas pelo método. Os métodos ativos podem ainda explorar formatos de mensagens que não são previstos pelo protocolo a fim de facilitar a identificação.

requer: r {endereço da máquina remota}
requer: c {controlador de acesso a rede}
1: $m \leftarrow c.interceptar(r, \texttt{MSG\_CONTROLE\_EOL\_RESPOSTA})$
2: $a \leftarrow m.dados$
retorna: a {assinatura de r}

Algoritmo 1.2: Exemplo de procedimento de caracterização de OS *fingerprinting* passivo.

Terminado o subprocesso de caracterização, é a vez do subprocesso de classificação (*matching*) ser executado. O subprocesso de classificação tem como componentes uma base de dados de assinaturas de sistemas operacionais e o resultado do procedimento de classificação. Na Tabela 1.2, vê-se o exemplo de base de dados de assinaturas para o exemplo adotado nessa seção. A partir dessa tabela pode-se notar que é possível a identificação de três grupos de sistemas operacionais. O procedimento de classificação é ilustrado no Algoritmo 1.3.

Nº	Assinatura	Sistema Operacional
1	EOL=$\backslash r$	Mac OS X
2	EOL=$\backslash n$	Linux e BSDs
3	EOL=$\backslash r\backslash n$	Windows

Tabela 1.2: Exemplo de base de dados de assinaturas.

requer: a {assinatura da máquina remota}
requer: d {tabela de assinaturas}
1: $r \leftarrow (0, \texttt{SISTEMA\_DESCONHECIDO})$
2: **para cada** linha $l \in d$ **faça**
3: **se** $l.assinatura = a$ **então**
4: $r \leftarrow (l.n, l.sistema)$
5: **fim se**
6: **fim para**
retorna: r {resultado}

Algoritmo 1.3: Exemplo de procedimento de classificação (*matching*).

O procedimento de classificação, ilustrado no Algoritmo 1.3, tem como resultado o nome do sistema operacional e o número da linha da base de dados em que a assinatura igual foi verificada. Nesse caso, a classificação utilizou uma métrica baseada apenas na igualdade entre as assinaturas. Porém, é possível utilizar outras métricas que expressem de forma quantitativa a similaridade entre assinaturas.

A escolha da métrica está relacionada diretamente à natureza dos dados contidos na assinatura. Quando é o caso da assinatura ser composta por um vetor de valores reais é possível utilizar como métrica, por exemplo, a distância Euclidiana. Nesse caso, a utilização de uma métrica baseada em distância oferece a possibilidade de comparar duas

assinaturas e saber o quão próximas são uma da outra. Isto faz com que a identificação apresente uma classificação mesmo não havendo na base de dados uma assinatura igual à capturada.

O exemplo adotado para ilustrar o processo de OS *fingerprinting* teve como objetivo apresentar o conceitos básicos presentes neste processo. Porém, as técnicas utilizadas para realizar a identificação remota do SO vão além em complexidade e variedade, quando comparadas com a apresentada no exemplo com um protocolo hipotético.

1.3 Planejamento e desafios

Alguns aspectos são relevantes no desenvolvimento de uma ferramenta que realiza OS *fingerprinting*. Cada um destes aspectos está relacionado à finalidade da ferramenta. Nesta seção, serão descritos estes aspectos e serão apresentados casos em que é conveniente a criação de ferramentas que realizem a identificação em acordo com cada aspecto.

1.3.1 Eficiência

Quantifica-se a eficiência de um processo de OS *fingerprinting* quanto ao número de pacotes que este injeta na rede (no caso ativo), quanto à quantidade de informações necessárias para criar a assinatura do SO da máquina remota e classificá-la e quanto ao tempo decorrido para realização dessa tarefa. A preocupação quanto a eficiência é importante quando o objetivo de utilização da ferramenta está relacionado principalmente a sua utilização em larga escala. A quantidade de tráfego gerado, a necessidade de serviços disponíveis na máquina remota e o tempo necessário para criar e classificar a assinatura são fatores que podem inviabilizar o uso de uma ferramenta. Pela descrição, pode-se discernir que a eficiência de uma ferramenta de identificação pode ser medida nos dois subprocessos (de caracterização e identificação) e depende dos três primeiros componentes (dados capturados, assinatura e base de dados de assinaturas) do processo de OS *fingerprinting*.

1.3.2 Eficácia

A eficácia de uma ferramenta está relacionada diretamente a qualidade do resultado. Para que a qualidade do resultado seja adequada é necessário, primeiramente, que as informações necessárias para criar a assinatura sejam freqüentemente disponíveis e suficientes para diferenciação entre sistemas operacionais. Em seguida, é necessário que a informação capturada seja devidamente representada na assinatura. Eventualmente, métodos de mineração de dados podem ser utilizados para otimizar este processo. A terceira condição está associada ao método de classificação de assinaturas. O método adotado deve ser robusto a problemas físicos, lógicos ou numéricos relacionados às fases anteriores. Por último, pode ser conveniente apresentar um resultado mesmo quando a igualdade entre assinaturas não acontece, a fim de situar o dispositivo testado no conjunto de SOs catalogados. Nestes casos, técnicas de visualização de informação podem ser utilizadas para auxiliar o usuário na identificação do sistema.

1.3.3 Detectabilidade

A característica de não detectabilidade em uma ferramenta de OS *fingerprinting* é, em muitos casos, desejável, isto porque, quando a detecção é factível, a máquina remota pode agir de forma incomum com o objetivo de atrapalhar a identificação. Existe também o caso em que um Sistema de Detecção de Intrusão, ou IDS (*Intrusion Detection System*), pode inibir a comunicação que possui como origem a máquina que realiza a identificação, caso a identificação seja detectada. Inicialmente, a detectabilidade é um aspecto do projeto que está relacionado à escolha de como os dados são criados e capturados. As ferramentas passivas possuem a vantagem da não detecção, enquanto os desenvolvedores de ferramentas ativas precisam tomar uma série de cuidados para concretizar esse fim.

1.3.4 Tratabilidade

A tratabilidade é definida como a capacidade que uma ferramenta possui de não danificar o estado do sistema operacional da máquina remota. Esta é uma preocupação presente apenas quanto ao desenvolvimento das ferramentas que realizam OS *fingerprinting* ativo, visto que este aspecto está relacionado diretamente à construção das mensagens enviadas a máquina remota pelo identificador. Muitas vezes, as ferramentas que realizam OS *fingerprinting* ativo exploram detalhes da especificação de protocolos que não são cobertas pela especificação. Nesses testes exploratórios, é comum a criação de mensagens malformadas ou com formação incomum. Como esses detalhes não são firmemente tratados pela especificação dos protocolos, cada SO pode implementar sua reação a essas mensagens de forma própria. Porém, a utilização de mensagens que exploram estes detalhes pode danificar o estado do sistema operacional da máquina remota, fazendo com que o serviço em questão trave, e em casos mais graves a máquina torne-se incomunicável. É comum dispositivos com sistemas proprietários e embarcados apresentar problemas quando recebem mensagens dessa natureza. Outra desvantagem relacionada ao uso dessas mensagens é que elas tornam a utilização da ferramenta detectável, pois raramente mensagens desse tipo são enviadas por uma máquina que deseja estabelecer uma comunicação normal.

1.3.5 Confiabilidade

É importante notar que o processo de OS *fingerprinting* utilizado nesta subseção pode ser enganado facilmente. Para tanto, é suficiente que a resposta à mensagem de controle do Exemplo 1.1 seja configurável. Nesse caso, o administrador da máquina remota é capaz de modificar o valor padrão de EOL na mensagem de controle de resposta. Por exemplo, em uma máquina Windows alterar o valor de EOL para "$\backslash n$" (EOL associado a sistemas baseados em Unix). As ferramentas que têm como objetivo enganar um processo de OS *fingerprinting* são denominadas OS *fingerprinting deceiving tools*. As técnicas utilizadas por essas ferramentas estão diretamente relacionada às técnicas utilizadas pelo processo de OS *fingerprinting* que ela pretende enganar.

1.4 Histórico

Os primeiros métodos utilizados para realização de OS *fingerprinting* baseavam-se na inspeção de informações em serviços disponibilizados pela máquina remota. Estas informações são obtidas, geralmente, apenas conectando-se ao serviço em questão. Outras vezes, é necessária uma interação mínima com o serviço para obter tais informações. Serão ilustrados nesta subseção os dois procedimentos.

Para exemplificar o segundo caso, em que uma interação mínima é necessária, será analisado o serviço HTTP de máquinas com configurações comumente encontradas na Internet. Para realizar está tarefa será utilizado o programa `telnet`. Está aplicação oferece um terminal de comunicação que pode ser utilizado para troca de mensagens com serviços, desde que o usuário tenha conhecimento do protocolo utilizado pelo mesmo. Utilizaremos no primeiro exemplo o endereço `maquina1.example.com`[1] para verificar que informações o servidor HTTP disponibiliza, como ilustrado no Exemplo 1.3.

```
1 # telnet maquina1.example.com 80
2 Trying 192.0.2.31...
3 Connected to universidade.example.com.
4 Escape character is '^]'.
5 GET / HTTP/1.1
6 Host: universidade.example.com
7 User-Agent: Mozilla/5.0 Gecko/2008102920 Firefox/3.0.4
8
```

Exemplo 1.3: Acesso ao serviço HTTP de `maquina1.example.com`.

Os comandos utilizados no Exemplo 1.3 (linhas 5 à 8) são relacionados ao protocolo HTTP versão 1.1 [Fielding et al. 1999]. Estes comandos são semelhantes aos que um navegador Web envia para um servidor HTTP durante o acesso a um sítio qualquer na Internet. O que se pretende fazer é enviar esta requisição para um servidor HTTP e verificar o conteúdo do cabeçalho de resposta. No Exemplo 1.4, é possível visualizar a resposta enviada pelo servidor de `maquina1.example.com`. A partir de uma análise desta resposta pode-se inferir (através da linha 3) que o sistema operacional do servidor HTTP é baseado em Linux, mais precisamente na distribuição Debian Etch. Embora clássica, esta técnica, ainda é eficaz nos casos em que estas informações, desnecessárias do ponto de vista de um usuário, não são retiradas do arquivo de configuração do serviço.

```
1 HTTP/1.1 302 Found
2 Date: Wed, 08 Apr 2009 00:56:17 GMT
3 Server: Apache/2.2.3 (Debian) mod_python/3.2.10 Python/2.4.4 PHP/5.2.0-8+etch13
4 X-Powered-By: PHP/5.2.0-8+etch13
5 Location: example
6 Content-Length: 0
7 Content-Type: text/html
```

Exemplo 1.4: Resposta da `maquina1.example.com` à requisição HTTP.

[1] Os endereços e nomes utilizados estão em acordo com a RFC 3330 [IANA 2002] e 2606 [Eastlake & Panitz 1999], respectivamente.

Como contra exemplo, foi utilizado a resposta de uma outra máquina executando outro servidor HTTP, cujo endereço é `maquina2.example.com`. A resposta obtida, enviando o mesmo cabeçalho utilizado do Exemplo 1.3, é apresentada no Exemplo 1.5. Como pode ser notado, a resposta do outro servidor retorna bem menos informações que a resposta do servidor da primeira máquina analisada. Como se pode notar pelo Exemplo 1.5 (linha 3) a única informação relevante disponível é que o servidor HTTP utilizado é o Apache. Vê-se que o segundo servidor HTTP foi configurado de forma a remover as informações desnecessárias no cabeçalho de resposta do serviço HTTP.

```
1  HTTP/1.1 200 OK
2  Date: Wed, 08 Apr 2009 01:35:55 GMT
3  Server: Apache
4  Set-Cookie: PHPSESSID=pojr4oa77attgi59r39nerpu56; path=/
5  Expires: Mon, 26 Jul 1997 05:00:00 GMT
6  Cache-Control: private, no-cache
7  Pragma: no-cache
8  Connection: close
9  Transfer-Encoding: chunked
10 Content-Type: text/html; charset=ISO-8859-1
```

Exemplo 1.5: Resposta da `maquina2.example.com` à requisição HTTP.

Também é possível realizar OS *fingerprinting* em outros tipos de serviços, como FTP (*File Transfer Protocol*) [Postel & Reynolds 1985] e Telnet [Postel & Reynolds 1983]. Nos Exemplos 1.6 e 1.7, são ilustrados o processo de inspeção dos serviços. No primeiro caso, o serviço FTP da máquina associada ao endereço 192.0.2.1 foi utilizado para realizar acesso anônimo (linhas 5 e 8). Posteriormente através da diretiva SYST do protocolo FTP foi obtida a informação presente na linha 12. Uma pesquisa na Internet pode mostrar que a versão descrita como "BSD-199506" pertence ao FreeBSD.

```
1  # telnet 192.0.2.1 21
2  Trying 192.0.2.1...
3  Connected to 192.0.2.1.
4  Escape character is '^]'.
5  USER anonymous
6  220 example.com FTP server (Version 6.00LS) ready.
7  331 Guest login ok, send your email address as password.
8  PASS me@somewhere
9  230- Your welcome message here.
10 230 Guest login ok, access restrictions apply.
11 SYST
12 215 UNIX Type: L8 Version: BSD-199506
```

Exemplo 1.6: Acesso anônimo ao serviço FTP.

Já no teste relacionado ao serviço Telnet da máquina associada ao endereço 192.0.2.2 não foi preciso realizar qualquer interação para que a informação necessária fosse obtida (linha 6 do Exemplo 1.7). Através de uma busca na Internet, é possível descobrir que "AP1100F" é um modelo de *gateway* utilizado em VoIP, produzido pela empresa AddPac, e o sistema operacional utilizado neste equipamento é o AddPac VoiceFinder Operating System (APOS).

```
1 # telnet 192.0.2.2 23
2 Trying 192.0.2.2...
3 Connected to 192.0.2.2.
4 Escape character is '^]'.
5
6 Welcome to AP1100F !
7
8 login:
```

Exemplo 1.7: Acesso ao serviço Telnet.

Atualmente, é prática comum esconder as informações utilizadas nesta seção para identificar dispositivos, motivo pelo qual estes métodos tendem a se tornar obsoletos. Mesmo assim, como visto, ainda é possível encontrar serviços que possibilitam a utilização deste tipo de procedimento.

1.5 Objetivo e organização do trabalho

É finalidade desse trabalho apresentar uma ferramenta capaz de realizar OS *fingerprinting* onde as ferramentas atuais não obtêm sucesso. Para tanto, inicialmente, deve-se identificar as deficiências ou ineficiências das ferramentas e técnicas de OS *fingerprinting* existentes e, posteriormente, procurar e avaliar soluções para os problemas encontrados. Os resultados obtidos com o uso das soluções propostas serão avaliadas com base nos aspectos de planejamento descritos na Seção 1.3.

No Capítulo 2 serão apresentadas as ferramentas utilizadas para realizar OS *fingerprinting* em redes que utilizam os protocolos da pilha TCP/IP. Serão destacadas, também, as limitações e as deficiências destas ferramentas. No final do segundo capítulo, serão apresentadas propostas que podem ser utilizadas para resolver os problemas encontrados. No Capítulo 3, será apresentada a fundamentação teórica necessária para implementar as propostas apresentadas no Capítulo 2. No Capítulo 4 é confirmada a funcionalidade do método proposto através da apresentação dos resultados. Finalmente, a conclusão e trabalhos futuros são abordados no Capítulo 5.

Capítulo 2

Identificação da pilha TCP/IP

Para entender as definições e exemplos apresentados neste capítulo, é necessário entender alguns conceitos relacionados às camadas de transporte e de rede da pilha TCP/IP. Será definido o conceito de TCP/IP *stack fingerprinting* e descrito seu funcionamento. Serão apresentadas, também, ferramentas que contribuem para a identificação de sistemas por meio desta técnica. As virtudes e limitações de cada uma destas ferramentas serão abordadas e descritos os desafios associados às técnicas de TCP/IP *stack fingerprinting* atuais. Finalmente, serão apresentadas as propostas desse trabalho.

2.1 Funcionamento

TCP/IP *stack fingerprinting* é uma forma de realizar OS *fingerprinting*. Está técnica tira proveito de detalhes que diferem entre si de implementação para implementação da pilha de protocolos TCP/IP [Fyodor 1998]. Antes de iniciar o estudo sobre a técnicas de OS *fingerprinting* que fazem uso dessa pilha de protocolos, será descrito de forma breve como é organizado o modelo TCP/IP [Postel 1981*c*, Stevens 1993], apresentado na Figura 2.1. Pode-se notar que o modelo divide-se em quatro camadas. Cada camada possui seus próprios protocolos, de maneira que os dados são repassados de forma transparente entre elas.

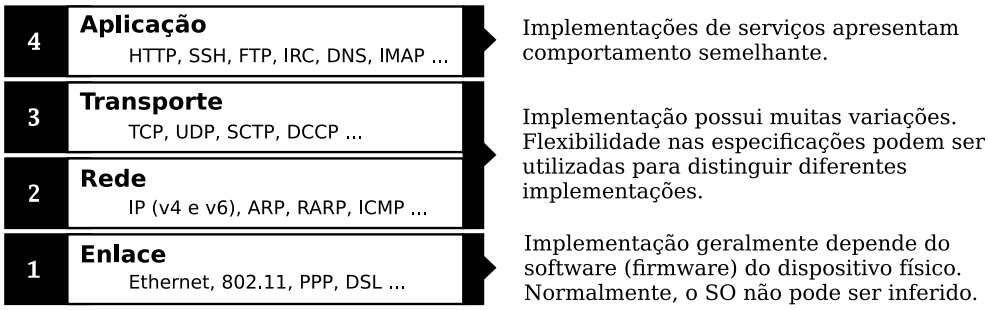

Figura 2.1: Camadas do modelo TCP/IP e sua influência em OS *fingerprinting*.

Do lado direito na Figura 2.1, são feitas algumas considerações em relação à importância de cada camada na tarefa de identificação remota de sistema operacional. Como

descrito nesta figura, as camadas de transporte e rede do modelo TCP/IP são as mais convenientes para explorar técnicas para OS *fingerprinting*. Para ilustrar um exemplo de detalhe de implementação que pode ser utilizado para realizar TCP/IP *stack fingerprinting*, é apresentada na Figura 2.2 uma rede constituída por três máquinas.

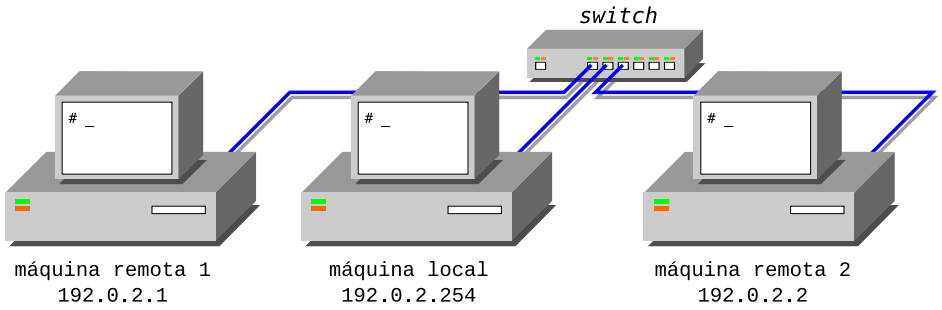

Figura 2.2: Três máquinas interligadas através de uma rede local.

Para realizar o experimento será feita a utilização do programa `ping`. Este programa utiliza o protocolo ICMP (*Internet Control Message Protocol*) para enviar um tipo de mensagem denominada ICMP *Echo Request* para uma máquina destino especificada. De acordo com a RFC 792 [Postel 1981*a*], quando a máquina destino recebe a mensagem *Echo Request*, deve responder a máquina de origem com uma mensagem *Echo Reply*. O formato da mensagem ICMP utilizada para desempenhar esta tarefa é apresentado na Figura 2.3 [Postel 1981*a*]. Deve-se acrescentar que, quando se trata da mensagem *Echo Request*, o campo `Type` da mensagem ICMP contém o valor 8 (oito), enquanto que no caso da mensagem *Echo Reply* este valor deve ser 0 (zero). E, em ambos os casos, o valor de `Code` deve ser 0 (zero) [Postel 1981*a*].

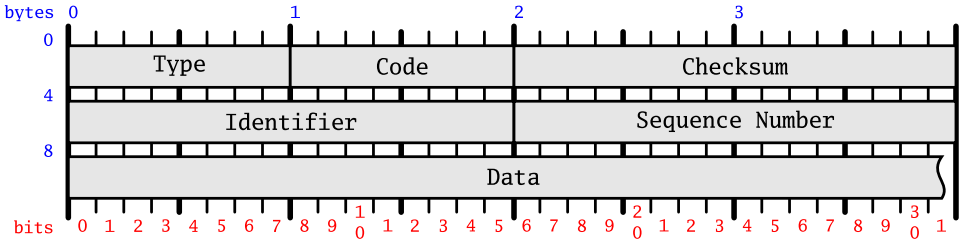

Figura 2.3: Formato da mensagem ICMP *Echo Request* e *Echo Reply* [Postel 1981*a*].

Toda mensagem que compõe o protocolo ICMP é encapsulada em um datagrama IP. O formato do cabeçalho do datagrama IP é apresentado na Figura 2.4 [Postel 1981*b*]. O encapsulamento da mensagem ICMP é discriminado atribuindo-se o valor 1 (um) ao campo `Protocol` do datagrama IP [Postel 1981*b*]. É importante notar nesse exemplo o campo `Time to Live` (ou TTL). Ele indica o número máximo de *hops*[1] que uma mensagem pode trafegar na rede antes de ser descartada. Será verificado a seguir que este

[1]O termo *hop* é utilizado para designar uma unidade de distância quando se refere a roteamento em redes de computadores.

campo é preenchido de forma diferente por diferentes sistemas operacionais em resposta a uma mensagem ICMP *Echo Request*.

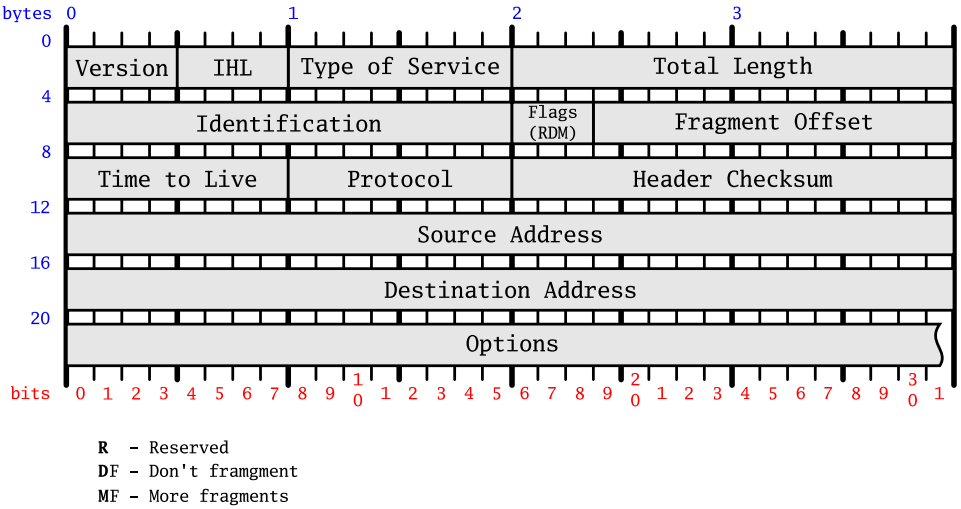

Figura 2.4: Formato do cabeçalho do datagrama IP [Postel 1981*b*].

A partir da máquina local (`192.0.2.254`), são enviadas quatro mensagens ICMP *Echo Request* (controlado pelo parâmetro "`-c 4`" do comando `ping`) para cada uma das máquinas remotas apresentadas na Figura 2.2. Neste experimento, a máquina cujo endereço é `192.0.2.1` possui instalado o sistema operacional o Slackware 12.1 (Linux 2.6.24), enquanto a máquina `192.0.2.2` possui o Windows XP SP2. O procedimento de envio de mensagens é ilustrado no Exemplo 2.1 (linhas 1 e 11).

```
1  # ping 192.0.2.1 -c 4
2  PING 192.0.2.1 (192.0.2.1) 56(84) bytes of data.
3  64 bytes from 192.0.2.1: icmp_seq=1 ttl=128 time=0.483 ms
4  64 bytes from 192.0.2.1: icmp_seq=2 ttl=128 time=0.357 ms
5  64 bytes from 192.0.2.1: icmp_seq=3 ttl=128 time=0.355 ms
6  64 bytes from 192.0.2.1: icmp_seq=4 ttl=128 time=0.359 ms
7
8  --- 192.0.2.1 ping statistics ---
9  4 packets transmitted, 4 received, 0% packet loss, time 2997ms
10 rtt min/avg/max/mdev = 0.355/0.388/0.483/0.058 ms
11 # ping 192.0.2.2 -c 4
12 PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
13 64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.022 ms
14 64 bytes from 192.0.2.2: icmp_seq=2 ttl=64 time=0.020 ms
15 64 bytes from 192.0.2.2: icmp_seq=3 ttl=64 time=0.022 ms
16 64 bytes from 192.0.2.2: icmp_seq=4 ttl=64 time=0.029 ms
17
18 --- 192.0.2.2 ping statistics ---
19 4 packets transmitted, 4 received, 0% packet loss, time 2997ms
20 rtt min/avg/max/mdev = 0.020/0.023/0.029/0.004 ms
```

Exemplo 2.1: Uso do comando `ping` para verificar o valor utilizado no campo TTL do cabeçalho IP produzindo pelo Linux 2.6.24 (linhas 1 a 10) e pelo Windows XP SP2 (linhas 11 a 20).

Pode-se verificar no Exemplo 2.1 que, para uma mesma mensagem ICMP *Echo Request*, cada uma das duas máquinas retornou uma mensagens ICMP *Echo Reply* com valor

do campo `ttl` diferente. No caso do Linux este valor foi igual a 128 (da linha 3 a 6) e, no caso do Windows, igual a 64 (da linha 13 à 16). Fica evidente, então, um detalhe de implementação da pilha de protocolos TCP/IP que pode ser utilizado para diferenciar estes dois sistemas operacionais. As ferramentas que realizam TCP/IP *stack fingerprinting* utilizam vários destes testes para criar a assinatura de um SO. Este exemplo específico é indicado apenas quando a máquina remota se encontra a poucos *hops* da máquina que realiza a identificação (seja ela passiva ou ativa), visto que o valor do campo TTL de um datagrama IP é decrementado a cada *hop*.

2.2 Metodologias

Para apresentar os métodos que realizam TCP/IP *stack fingerprinting* serão descritas ferramentas e técnicas que as compõem. Estes métodos utilizam detalhes que diferem entre si na implementação da pilha de protocolos TCP/IP de cada SO. Os protocolos utilizados são o ICMP, IP, TCP e UDP (*User Datagram Protocol*). Após a apresentação de algumas características desses protocolos que são relevantes para o processo de OS *fingerprinting*, serão apresentados os métodos encontrados na literatura.

O protocolo IP (cabeçalho apresentado na Figura 2.4) contribui de forma significativa para realização de OS *fingerprinting*. A especificação do IP não é mandatória em relação aos valores de alguns campos como, por exemplo, o tempo de vida do pacote e o algoritmo pelo qual deve ser gerado o valor do campo de identificação. Uma vantagem dos métodos que utilizam informações relacionadas ao protocolo IP possuem é a de que, por se tratar de um protocolo da camada de rede, seu cabeçalho estará presente para qualquer outro protocolo encapsulado, seja ele ICMP, TCP ou UDP. Portanto, mesmo que a máquina remota não disponibilize qualquer tipo de serviço, ainda é possível utilizar informações do cabeçalho IP através do envio e recebimento de mensagens ICMP.

A especificação do protocolo ICMP descreve nove tipos de mensagens. Na Figura 2.3, é apresentado o formato de mensagem *Echo Request/Reply* do protocolo ICMP. Ao contrário da especificação do protocolo IP, a do ICMP é bem incisiva. Geralmente, os métodos que utilizam dados do ICMP aproveitam-se de falhas em sua implementação. Por exemplo, de acordo com a RFC 792, o campo `Code` deve ser igual a zero em ambas as mensagens [Postel 1981*a*]. Por motivos desconhecidos, algumas implementações respondem à mensagens *Echo Request* com o campo `Code` com valores diferentes de zero.

O formato do cabeçalho do segmento do protocolo TCP é apresentado na Figura 2.5. O TCP é um dos protocolos mais complexos de toda a pilha de protocolos TCP/IP. Este fato, associado à quantidade de campos no cabeçalho onde a especificação não é mandatória, e à "obrigatoriedade" da implementação deste protocolo, faz com que ele seja o protocolo que mais contribui para o processo de OS *fingerprinting*. Tópicos da especificação que tratam de forma superficial de geração de valores aleatórios, tamanho inicial da janela, opções adicionais, campos reservados e combinação de *flags* tendem a fazer com que cada implementação seja diferente uma da outra. Por esse motivo a ausência de uma porta TCP aberta na máquina remota dificulta de forma considerável a classificação correta de seu sistema operacional via TCP/IP *stack fingerprinting*.

O formato do cabeçalho do datagrama do protocolo UDP é apresentado na Figura 2.6.

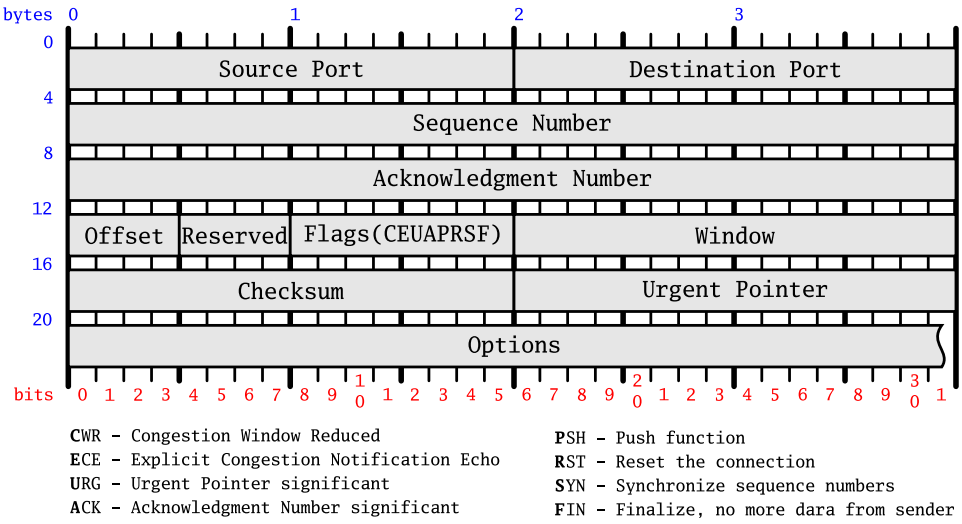

Figura 2.5: Formato do cabeçalho do segmento TCP [Postel 1981*c*].

Devido à simplicidade deste protocolo, é difícil estabelecer qualquer método que extraia alguma informação relevante dentre as diferentes implementações. O que acontece normalmente é que este protocolo é utilizado para gerar mensagens de resposta de outros protocolos como o ICMP. Por exemplo, quando uma mensagem TCP ou UDP é enviada para uma porta fechada na máquina remota, está deve responder com uma ICMP *Destination Unreachable* com o campo `Code` igual a 3 (*Port unreachable*). Esta mensagem ICMP pode conter informações relevantes para o processo de OS *fingerprinting*. Quando possível, os métodos de identificação dão preferência ao uso de mensagens UDP, pois mensagens TCP têm maiores chances de serem bloqueadas por alguns dispositivos de segurança de redes.

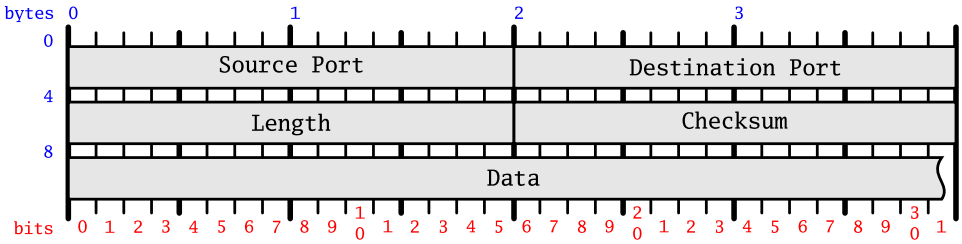

Figura 2.6: Formato do cabeçalho do datagrama UDP [Postel 1980].

Os métodos utilizados pelas ferramentas OS *fingerprinting* atuais são semelhantes aos aqui descritos. Mais adiante, estas ferramentas são utilizadas, porém, os métodos que estas utilizam não são detalhados. Nesse trabalho, serão analisadas as ferramentas que utilizam diferentes abordagens em relação aos dados utilizados, porém todas elas são do tipo ativo de OS *fingerprinting*, pois geralmente são mais eficazes.

Dentre as ferramentas disponíveis, foram selecionadas aquelas que possuem maior aceitação pela comunidade de segurança [Nmap Hackers Mailing List 2008] e são consideradas as mais eficientes [Fyodor 2009*b*] e em que as técnicas que utilizam são, no

mínimo, citadas em outros trabalhos [Auffret 2008]. Na Tabela 2.1, são apresentadas estas ferramentas, as versões utilizadas nos testes, a lista de protocolos utilizados por cada uma delas e a bibliografia especificando as técnicas utilizadas por cada uma delas.

Ferramenta	Versão	Protocolos	Referência
Nmap	4.85	IP, ICMP, TCP e UDP	[Fyodor 2009*b*]
SinFP	2.06	TCP e IP	[Auffret 2008]
Xprobe2	0.3	UDP e ICMP	[Arkin & Yarochkin 2002]

Tabela 2.1: Discriminação das ferramentas de TCP/IP *stack fingerprinting* utilizadas.

Cada uma das ferramentas citadas na Tabela 2.1 foi submetida a uma série de testes relacionados a sua capacidade de identificação e robustez quanto a presença de dispositivos de segurança de rede como, por exemplo, *firewalls*. Inicialmente, os testes foram realizados em um ambiente controlado sem a presença de *firewalls*. Neste caso, as ferramentas estão, teoricamente, sob condições ideais. Assim sendo, os resultados relacionados a estes testes expressam os melhores resultados possíveis para cada uma das ferramentas. Esse ambiente de testes inicial é apresentado na Figura 2.7.

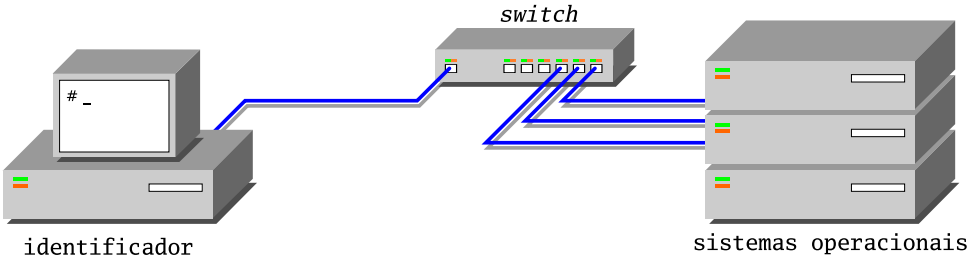

Figura 2.7: Primeiro ambiente de testes utilizado.

O sistemas operacionais instalados nas máquinas à direita na Figura 2.7, foram escolhidas de acordo com as disponibilidades relacionadas à obtenção do próprio sistema operacional e de equipamentos. Os sistemas utilizados são listados na Tabela 2.2.

Sistema operacional	Endereço	Versão detalhada
Debian	`192.0.2.1`	`Linux debian 2.6.26-1-686`
FreeBSD	`192.0.2.2`	`6.4-RELEASE i386`
NetBSD	`192.0.2.3`	`4.0.1 GENERIC i386`
OpenBSD	`192.0.2.4`	`4.4 GENERIC#1021 i386`
OpenSolaris	`192.0.2.5`	`SunOS 5.11 snv_101b i86pc`
Windows 2000	`192.0.2.6`	`5.00.2195 Service Pack 4`
Windows XP	`192.0.2.7`	`Version 2002 Service Pack 2`

Tabela 2.2: Sistemas operacionais utilizados, suas respectivas versões e endereços.

Após a realização dos testes iniciais, são realizados os testes na presença de uma *firewall*. Esse dispositivo tem como objetivo proteger uma determinada rede de acesso não autorizado. Dentre suas tarefas, pode-se citar: permitir e bloquear, encriptar e decriptar e intermediar o tráfego entre duas redes de computadores. Geralmente essas tarefas são arbitradas por um conjunto de regras pré-definidas pelo administrador da *firewall*. Três funcionalidades das *firewalls* que podem comprometer o resultado da identificação remota. São eles: bloqueio de pacotes, normalização de tráfego e intermediação de sincronização (tradução do termo em inglês SYN *proxy*. Para demonstrar essa característica, foi utilizada a *firewall* PF [OpenBSD PF 2009] do sistema operacional OpenBSD [*OpenBSD* 2008]. A configuração da *firewall* foi realizada conforme o guia exemplo de utilização para uso doméstico e em pequenas empresas [Knight 2004]. Para exemplificar a utilização desta *firewall*, foi adotada a rede apresentada na Figura 2.8.

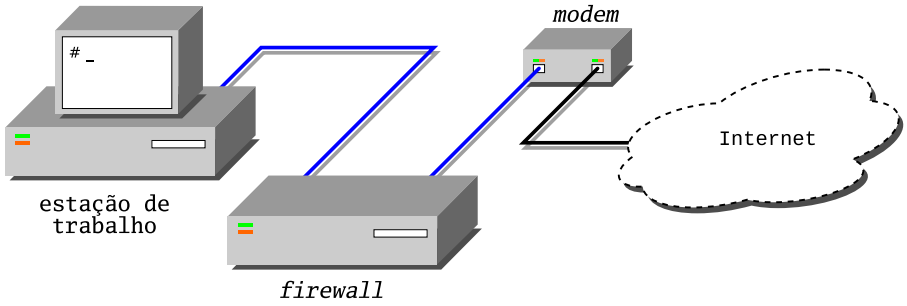

Figura 2.8: Exemplo de utilização de *firewall*.

Do lado esquerdo na Figura 2.8 está a estação de trabalho de um usuário. O acesso à Internet é provido por um *modem* ADSL (*Asymmetric Digital Subscriber Line*), sendo o acesso realizado utilizando PPPoE (*Point-to-Point Protocol over Ethernet*). Considerando um ambiente em que a *firewall* tem como objetivo proteger um determinado conjunto de serviços, por exemplo HTTP e SSH, todo o tráfego não associado a estes serviços deve ser bloqueado. Este bloqueio faz com que de imediato todo tráfego ICMP e UDP originado da Internet seja descartado. Quanto à normalização de tráfego, praticamente todas as peculiaridades exploradas pelos métodos ativos, presentes nos pacotes maliciosos, enviados por algumas ferramentas para a máquina remota, são retiradas ou fazem com que o pacote seja descartado [Handley et al. 2001]. A normalização de pacotes é realizada nos protocolos IP, TCP, UDP e ICMP. Por último, existe a intermediação de sincronização (ou SYN *proxy*). A utilização de SYN *proxies* faz com que os pacotes SYN+ACK de sincronização do TCP enviados em resposta a requisições SYN não sejam originados da máquina alvo, mas sim, da própria *firewall*. Este procedimento é utilizado para proteger servidores de ataques de DoS (*Denial of Service*) [Eddy 2007]. Porém, a utilização de SYN *proxies* também afeta diretamente as ferramentas de identificação que utilizam o protocolo TCP. Desta forma, a utilização de *firewalls* pode comprometer o resultado da identificação.

Há ainda outros problemas relacionados à utilização de NAT (*Network Address Translation*) e PAT (*Port Address Translation*) – também conhecido como NAPT (*Network Address Port Translation*) [Egevang & Francis 1994, Srisuresh & Egevang 2001]. NAT é o processo que modifica o endereço de rede no cabeçalho de datagramas IP com o objetivo

de mapear um espaço de endereçamento em outro. Inicialmente o NAT foi desenvolvido para solucionar o problema de esgotamento de endereços [Egevang & Francis 1994]. Porém, ele também pode ser utilizado para proteger uma rede privada por trás de uma única máquina conectada a Internet. Desta forma, a única máquina suscetível a invasões, originadas da Internet, é aquela conectada diretamente a Internet e que realiza NAT para as demais máquinas da rede privada. PAT é uma tecnologia semelhante ao NAT, porém, o mapeamento entre a porta do dispositivo da rede interna e a porta do dispositivo exposta a Internet é feita de forma explícita. O uso de PAT dificulta o processo de identificação porque o sistema operacional a ser identificado depende de qual porta a ferramenta coleta as informações. Se a ferramenta de identificação utilizar mais que uma porta TCP aberta para criar sua assinatura, provavelmente esta assinatura não representará a pilha TCP/IP de nenhum dos dois sistemas. Como o PF possui a capacidade de realizar PAT, serão também realizados testes acerca da utilização de PAT.

Além da utilização de *firewalls* existem ferramentas que têm como objetivo primário enganar ferramentas que realizam OS *fingerprinting* [Smart et al. 2000]. O Honeyd é uma ferramenta que tem como propósito a simulação de máquinas, serviços e sistemas operacionais em rede [Provos 2007]. Neste sentido, esta ferramenta simula diferentes implementações da pilha TCP/IP. Atualmente, todas as ferramentas que realizam TCP/IP *stack fingerprinting* são, de alguma forma, enganadas por esta ferramenta [Provos & Holz 2008]. Neste sentido, considerando a utilização de *firewalls* e a presença do Honeyd, a arquitetura do segundo ambiente de testes é apresentada na Figura 2.9.

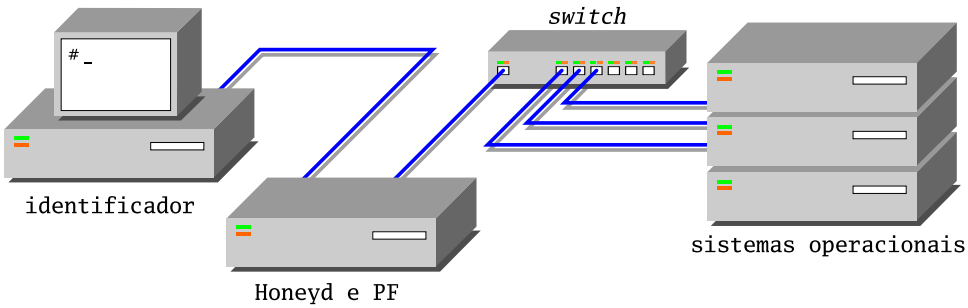

Figura 2.9: Segundo ambiente de testes utilizado (utilizando Honeyd e PF).

Espera-se, inicialmente, confirmar, com os testes, a deficiência de cada uma das ferramentas/técnicas analisadas. Posteriormente, será verificada quais informações podem ser utilizadas para realizar OS *fingerprinting* mesmo na presença do PF e do Honeyd.

2.3 Testes de eficácia

A realização dos testes consistirá, inicialmente, na tentativa de identificação dos sistemas operacionais listados na Tabela 2.2. A arquitetura utilizada é aquela apresentada na Figura 2.7. Na Tabela 2.2, são definidos os endereços IP das máquinas associadas a cada sistema operacional. A seguir cada ferramenta apresentada na Tabela 2.1 será utilizada na identificação de cada um dos sistemas operacionais. Os testes realizados têm como

objetivo apresentar resultados que servirão como referência. Isto porque, quando adotada a arquitetura da Figura 2.9 a ferramenta que apresentar resultado semelhante ao desta seção não foi afetada significativamente. Outro objetivo é verificar se as ferramentas são capazes de distinguir os sistemas operacionais adotados. Na Tabela 2.3 são discriminadas as portas/serviços utilizados na execução das ferramentas de identificação.

Sistema operacional	Serviço	Porta aberta	Porta fechada
Debian	OpenSSH	22	135
FreeBSD	OpenSSH	22	135
NetBSD	OpenSSH	22	135
OpenBSD	OpenSSH	22	135
OpenSolaris	OpenSSH	22	135
Windows 2000	Microsoft RPC	135	22
Windows XP	Microsoft RPC	135	22

Tabela 2.3: Portas utilizadas nos testes de eficácia.

Em todas as máquinas, com exceção das que possuem Windows instalado, a porta 22 estava aberta. Já nas duas máquinas Windows a porta aberta utilizada foi a 135. As portas fechadas são utilizadas pelo processo de identificação do Nmap [Fyodor 2006].

2.3.1 Nmap

A primeira ferramenta submetida aos testes é denominada Nmap [Fyodor 2009*a*]. Esta ferramenta é a única ferramenta de código livre [Free Software Foundation 1996] que possui seu módulo de identificação constantemente atualizado. Este fato será verificado de acordo com os resultados apresentados nesta seção. O método de identificação do Nmap é habilitado com a opção "`-O`". O Nmap envia um total de 16 pacotes para a máquina alvo explorando características da implementação dos protocolos TCP, UDP, IP e ICMP [Fyodor 2006, Fyodor 2009*b*]. Nos Exemplos 2.2, 2.3, 2.5, 2.4, 2.6, 2.7 e 2.8 são apresentados os resultados para os sistemas operacionais da Tabela 2.2.

```
1 # nmap -O -p 22,135 192.0.2.1
2 (...)
3 Device type: general purpose
4 Running: Linux 2.6.X
5 OS details: Linux 2.6.13 - 2.6.27
6 (...)
```

Exemplo 2.2: Resultado do Nmap para o Debian.

```
1 # nmap -O -p 22,135 192.0.2.2
2 (...)
3 Device type: general purpose
4 Running: FreeBSD 6.X
5 OS details: FreeBSD 6.2-STABLE - 6.4-STABLE
6 (...)
```

Exemplo 2.3: Resultado do Nmap para o FreeBSD.

```
1 # nmap –O –p 22,135 192.0.2.3
2 (...)
3 Device type: general purpose
4 Running: NetBSD 4.X
5 OS details: NetBSD 4.99.4
6 (...)
```

Exemplo 2.4: Resultado do Nmap para o NetBSD.

```
1 # nmap –O –p 22,135 192.0.2.4
2 (...)
3 Device type: general purpose
4 Running: OpenBSD 3.X|4.X
5 OS details: OpenBSD 3.9 – 4.4
6 (...)
```

Exemplo 2.5: Resultado do Nmap para o OpenBSD.

```
1 # nmap –O –p 22,135 192.0.2.5
2 (...)
3 Device type: general purpose
4 Running: Sun Solaris 9|10
5 OS details: Sun Solaris 9 or 10
6 (...)
```

Exemplo 2.6: Resultado do Nmap para o OpenSolaris.

```
1 # nmap –O –p 22,135 192.0.2.6
2 (...)
3 Device type: general purpose
4 Running: Microsoft Windows 2000
5 OS details: Microsoft Windows 2000 SP0/SP1/SP2 or Windows XP SP0/SP1
6 (...)
```

Exemplo 2.7: Resultado do Nmap para o Windows 2000.

```
1 # nmap –O –p 22,135 192.0.2.7
2 (...)
3 Device type: general purpose
4 Running: Microsoft Windows XP
5 OS details: Microsoft Windows XP SP2 or SP3
6 (...)
```

Exemplo 2.8: Resultado do Nmap para o Windows XP.

Como se pode notar, o Nmap foi capaz de reconhecer cada um dos sistemas operacionais adotados. Deve-se acrescentar apenas que a versão do Windows 2000 relatada no Exemplo 2.7 não corresponde à utilizada e que o OpenSolaris foi reconhecido como Solaris. Isto acontece, provavelmente, porque, em ambos os casos, as versões dos sistemas na base de dados do Nmap possuem a mesma implementação da pilha TCP/IP.

2.3.2 SinFP

O SinFP é uma ferramenta de identificação cujo objetivo é superar os problemas que o Nmap apresenta em ambiente onde se faz presente a utilização de NAT/PAT [Auffret 2006]. O sistema de identificação desta ferramenta utiliza apenas três mensagens TCP.

Uma única porta TCP aberta na máquina de destino é suficiente para realizar a identificação. Por este motivo, o uso de PAT não exerce influência no desempenho do SinFP. Nos Exemplos 2.9, 2.10, 2.12, 2.11, 2.13, 2.14 e 2.15 são apresentados os resultados para os sistemas operacionais da Tabela 2.2.

```
# sinfp.pl -i 192.0.2.1 -p 22
(...)
IPv4: HEURISTIC0/P1P2P3: GNU/Linux: Linux: 2.6.x
(...)
```

Exemplo 2.9: Resultado do SinFP para o Debian.

```
# sinfp.pl -i 192.0.2.2 -p 22
(...)
IPv4: HEURISTIC0/P1P2P3: BSD: FreeBSD: 6.0
IPv4: HEURISTIC0/P1P2P3: BSD: FreeBSD: 6.1
IPv4: HEURISTIC0/P1P2P3: BSD: FreeBSD: 6.2
(...)
```

Exemplo 2.10: Resultado do SinFP para o FreeBSD.

```
# sinfp.pl -i 192.0.2.3 -p 22
(...)
IPv4: HEURISTIC0/P1P2P3: BSD: NetBSD: 3.0
(...)
```

Exemplo 2.11: Resultado do SinFP para o NetBSD.

```
# sinfp.pl -i 192.0.2.4 -p 22
(...)
IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.5
IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.6
IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.7
IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.8
IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.9
IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 4.0
(...)
```

Exemplo 2.12: Resultado do SinFP para o OpenBSD.

```
# sinfp.pl -i 192.0.2.5 -p 22
(...)
IPv4: HEURISTIC0/P1P2P3: Unix: SunOS: 5.10
IPv4: HEURISTIC0/P1P2P3: Unix: SunOS: 5.9
(...)
```

Exemplo 2.13: Resultado do SinFP para o OpenSolaris.

```
# sinfp.pl -i 192.0.2.6 -p 135
(...)
IPv4: HEURISTIC0/P1P2P3: Windows: Windows: 2000
(...)
```

Exemplo 2.14: Resultado do SinFP para o Windows 2000.

```
# sinfp.pl -i 192.0.2.7 -p 135
(...)
IPv4: HEURISTIC0/P1P2P3: Windows: Windows: 2000
(...)
```

Exemplo 2.15: Resultado do SinFP para o Windows XP.

Os resultados obtidos com o SinFP também são consistentes. Novamente, apenas as versões dos sistemas operacionais apresentadas não são inteiramente corretas e, no caso do Windows XP, errada.

2.3.3 Xprobe2

A ferramenta Xprobe2 [Yarochkin et al. 2005] é resultado de uma pesquisa iniciada em 2000 por Ofir Arkin [Arkin 2002]. As ferramentas que realizavam OS *fingerprinting* naquela época dependiam quase sempre da utilização do protocolo TCP por parte da máquina alvo. A preocupação emergente em relação a segurança em redes de computadores faz com que o número de máquinas com portas expostas na rede seja cada vez menor. Pode-se confirmar isto pela evolução das versões do Windows XP. Em suas primeiras versões, SP0 e SP1, várias portas são abertas por padrão. Tal fato não se repete nas novas distribuições a partir do SP2. Pensando neste fato, Ofir Arkin projetou o Xprobe2 para utilizar apenas o protocolo ICMP para criar as mensagens utilizadas para extrair informações da máquina remota [Arkin et al. 2003, Arkin & Yarochkin 2002]. Nos Exemplos 2.16, 2.17, 2.18, 2.19, 2.20, 2.21 e 2.22 são apresentados os resultados para os sistemas operacionais da Tabela 2.2.

```
# xprobe2 192.0.2.1
(...)
[+] Primary guess:
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.23" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.21" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.20" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.19" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.24" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.25" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.26" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.27" (Guess probability: 100%)
[+] Host 192.0.2.1 Running OS: "Linux Kernel 2.4.28" (Guess probability: 100%)
(...)
```

Exemplo 2.16: Resultado do Xprobe2 para o Debian.

```
# xprobe2 192.0.2.2
(...)
[+] Primary guess:
[+] Host 192.0.2.2 Running OS: "FreeBSD 5.2" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.2 Running OS: "FreeBSD 5.2.1" (Guess probability: 100%)
[+] Host 192.0.2.2 Running OS: "FreeBSD 5.3" (Guess probability: 100%)
[+] Host 192.0.2.2 Running OS: "FreeBSD 5.4" (Guess probability: 100%)
[+] Host 192.0.2.2 Running OS: "Apple Mac OS X 10.2.6" (Guess probability:
    100%)
[+] Host 192.0.2.2 Running OS: "Apple Mac OS X 10.2.7" (Guess probability:
    100%)
[+] Host 192.0.2.2 Running OS: "Apple Mac OS X 10.2.8" (Guess probability:
    100%)
[+] Host 192.0.2.2 Running OS: "Apple Mac OS X 10.3.3" (Guess probability: 96%)
[+] Host 192.0.2.2 Running OS: "Apple Mac OS X 10.3.7" (Guess probability: 96%)
[+] Host 192.0.2.2 Running OS: "Apple Mac OS X 10.4.1" (Guess probability: 96%)
(...)
```

Exemplo 2.17: Resultado do Xprobe2 para o FreeBSD.

```
# xprobe2 192.0.2.3
(...)
[+] Primary guess:
[+] Host 192.0.2.3 Running OS: "NetBSD 2.0" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.3 Running OS: "NetBSD 1.6.2" (Guess probability: 100%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.6.1" (Guess probability: 100%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.6" (Guess probability: 100%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.5.3" (Guess probability: 96%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.5.2" (Guess probability: 96%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.5.1" (Guess probability: 96%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.5" (Guess probability: 96%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.4.3" (Guess probability: 96%)
[+] Host 192.0.2.3 Running OS: "NetBSD 1.4.2" (Guess probability: 96%)
(...)
```

Exemplo 2.18: Resultado do Xprobe2 para o NetBSD.

```
# xprobe2 192.0.2.4
(...)
[+] Primary guess:
[+] Host 192.0.2.4 Running OS: "OpenBSD 3.7" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.4 Running OS: "OpenBSD 3.6" (Guess probability: 100%)
[+] Host 192.0.2.4 Running OS: "OpenBSD 3.5" (Guess probability: 100%)
[+] Host 192.0.2.4 Running OS: "OpenBSD 3.4" (Guess probability: 100%)
[+] Host 192.0.2.4 Running OS: "OpenBSD 2.5" (Guess probability: 96%)
[+] Host 192.0.2.4 Running OS: "OpenBSD 2.9" (Guess probability: 100%)
[+] Host 192.0.2.4 Running OS: "NetBSD 1.4" (Guess probability: 96%)
[+] Host 192.0.2.4 Running OS: "NetBSD 1.4.1" (Guess probability: 96%)
[+] Host 192.0.2.4 Running OS: "NetBSD 1.4.2" (Guess probability: 96%)
[+] Host 192.0.2.4 Running OS: "NetBSD 1.4.3" (Guess probability: 96%)
(...)
```

Exemplo 2.19: Resultado do Xprobe2 para o OpenBSD.

Neste ponto, pode-se notar através de uma análise dos Exemplos 2.16, 2.17, 2.18 e 2.19 que a base de dados de assinaturas do Xprobe2 não foi recentemente atualizada. Isto porque não estão presentes as versões 2.6.X do Linux, 6.X do FreeBSD, 3.X e 4.X do NetBSD e 4.X do OpenBSD.

```
# xprobe2 192.0.2.5
(...)
[+] Primary guess:
[+] Host 192.0.2.5 Running OS: "HP UX 11.0" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.5 Running OS: "Sun Solaris 10 (SunOS 5.10)" (Guess
    probability: 100%)
[+] Host 192.0.2.5 Running OS: "HP UX 11.0x" (Guess probability: 96%)
[+] Host 192.0.2.5 Running OS: "Sun Solaris 9 (SunOS 5.9)" (Guess probability:
    96%)
[+] Host 192.0.2.5 Running OS: "Sun Solaris 8 (SunOS 2.8)" (Guess probability:
    96%)
[+] Host 192.0.2.5 Running OS: "Sun Solaris 7 (SunOS 2.7)" (Guess probability:
    96%)
[+] Host 192.0.2.5 Running OS: "Sun Solaris 2.5.1" (Guess probability: 96%)
[+] Host 192.0.2.5 Running OS: "Sun Solaris 6 (SunOS 2.6)" (Guess probability:
    96%)
[+] Host 192.0.2.5 Running OS: "Apple Mac OS X 10.3.8" (Guess probability: 88%)
[+] Host 192.0.2.5 Running OS: "OpenBSD 3.6" (Guess probability: 88%)
(...)
```

Exemplo 2.20: Resultado do Xprobe2 para o OpenSolaris.

```
# xprobe2 192.0.2.6
(...)
[+] Primary guess:
[+] Host 192.0.2.6 Running OS: "Microsoft Windows XP" (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Server Service Pack 4"
    (Guess probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Server Service Pack 3"
    (Guess probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Server Service Pack 2"
    (Guess probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Server Service Pack 1"
    (Guess probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows XP SP1" (Guess probability:
    100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Server" (Guess
    probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Workstation SP4" (Guess
    probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess
    probability: 100%)
[+] Host 192.0.2.6 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess
    probability: 100%)
(...)
```

Exemplo 2.21: Resultado do Xprobe2 para o Windows 2000.

```
# xprobe2 192.0.2.7
(...)
[+] Primary guess:
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2003 Server Standard Edition"
    (Guess probability: 100%)
[+] Other guesses:
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2003 Server Enterprise
    Edition" (Guess probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows XP SP2" (Guess probability:
    100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Workstation" (Guess
    probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Workstation SP1" (Guess
    probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess
    probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess
    probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Workstation SP4" (Guess
    probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Server" (Guess
    probability: 100%)
[+] Host 192.0.2.7 Running OS: "Microsoft Windows 2000 Server Service Pack 1"
    (Guess probability: 100%)
(...)
```

Exemplo 2.22: Resultado do Xprobe2 para o Windows XP.

O Xprobe2 também não consegue distinguir claramente entre o Windows 2000 e o Windows XP SP2. Porém, o Xprobe2 apresenta uma vantagem que é a não necessidade de portas abertas na máquina destino para realizar a identificação.

2.4 Testes de eficácia na presença de *firewall*

A realização de outros testes é feita utilizando a *firewall* PF e a arquitetura proposta na Figura 2.9. Como o objetivo é demonstrar a influência da *firewall* no processo de identificação de cada ferramenta, apenas o sistema operacional Debian fará o papel da máquina remota. O arquivo utilizado na configuração do PF é apresentado no Exemplo 2.23[2].

```
1  ##############################################################################
2  # Conjunto de regras utilizadas para proteger os sistemas operacionais da    #
3  # rede de testes proposta.                                                   #
4  ##############################################################################
5
6  # - Macros ---------------------------------------------------------------- #
7  ext_if="pcn0"              # interface da rede externa, desmilitarizada
8  int_if="pcn1"              # interface da rede interna, protegida
9  tcp_services="{ 22, 113 }" # portas da firewall abertas para Internet
10 icmp_types="echoreq"       # mensagens ICMP liberadas
11 web_server="192.0.2.1"     # endereco do servidor HTTP interno
12
13 # - Opcoes ---------------------------------------------------------------- #
14 set block-policy return    # definindo a politica padrao para bloqueio
15 set loginterface $ext_if   # habilitando log na interface de saida
16 set skip on lo             # firewall nao interferira na interface de loopback
17
18 # - Normalizacao ---------------------------------------------------------- #
19 scrub in
20
21 # - NAT/PAT --------------------------------------------------------------- #
22 nat on $ext_if from !($ext_if) to any -> ($ext_if)              # NAT
23 rdr on $ext_if proto tcp from any to any port 80 -> $web_server # PAT
24
25 # - Regras ---------------------------------------------------------------- #
26 block in                          # bloqueia tudo por padrao (deny all)
27 pass out keep state               # permite a passagem do trafego de saida
28
29 pass in on $ext_if inet proto tcp \            # libera acesso aos servicos
30     from any to ($ext_if) port $tcp_services \ # disponibilizados pela
31     flags S/SA keep state                      # prorpria firewall
32
33 pass in on $ext_if inet proto tcp \    # habilita protecao contra ataques de
34     from any to $web_server port 80 \  # SYN flooding no servidor HTTP da rede
35     flags S/SA synproxy state          # interna
36
37 pass in inet proto icmp \        # libera a passagem de pacotes ICMP echo
38     all icmp-type $icmp_types \  # request
39     keep state
40
41 pass in quick on $int_if   # libera todo trafego originado da rede interna
42
43 ##############################################################################
```

Exemplo 2.23: Arquivo de configuração (`/etc/pf.conf`) da *firewall* PF.

Nas linhas 7 e 8 são definidos rótulos para as duas interfaces de rede da *firewall*. A primeira (`pcn0`) é definida como a interface externa (`ext_if`), que fornece acesso à máquina de identificação (à esquerda na Figura 2.9). A segunda (`pcn1`) é definida como a interface interna (`int_if`), onde são acessíveis as máquinas cujos sistemas operacionais serão submetidos ao processo de identificação (à direita na Figura 2.9). O endereço

[2] A acentuação nos comentários (trechos precedidos por "#") foi suprimida intencionalmente. Para que a *firewall* funcione corretamente deve-se suprimir os comentários presentes nas linhas 29, 30, 33, 34, 37 e 38, que estão dispostas no exemplo desta forma apenas para descrever os comandos de forma compacta.

atribuído a *firewall* foi `192.0.2.254`, enquanto os demais endereços permanecem associados aos sistemas operacionais citados anteriormente. Considerando a Figura 2.9, a rede onde a estação responsável pela identificação se encontra é definida, utilizando a notação CIDR (*Classless Inter-Domain Routing*) [Fuller & Li 2006], como sendo `192.0.2.128/25`, e a rede onde os sistemas operacionais foram instalados é a `192.0.2.0/25`.

2.4.1 Tradução de endereço

Inicialmente será analisado apenas a influência da utilização de NAT/PAT na identificação. Para que isto seja possível deve-se comentar a linha 19 do Exemplo 2.23 e substituir "`synproxy`" na linha 35 por "`keep`". A porta 80 da máquina Debian (`192.0.2.1`) está vinculada agora a porta 80 da *firewall* (`192.0.2.254`). Desta forma, o indivíduo utilizando a máquina de identificação não sabe a priori que a porta 80 da *firewall* é redirecionada para o serviço HTTP de outra máquina. Primeiramente, a ferramenta Nmap será submetida aos testes. Desta forma o usuário identificador utiliza o comando apresentado no Exemplo 2.24.

```
1  # nmap –O 192.0.2.254
2  (...)
3  Interesting ports on 192.0.3.1:
4  Not shown: 997 closed ports
5  PORT     STATE SERVICE
6  22/tcp   open  ssh
7  80/tcp   open  http
8  113/tcp  open  auth
9  (...)
10 Device type: general purpose
11 Running: OpenBSD 4.X
12 OS details: OpenBSD 4.4
13 (...)
```

Exemplo 2.24: Resultado do Nmap na presença de PAT.

O sistema operacional discriminado foi o OpenBSD porque a porta TCP utilizada pelo Nmap foi a 22 ou 113. A partir do exemplo de configuração sabe-se que as portas 22 e 113 pertencem de fato a *firewall*. Porém, o usuário pode assumir erroneamente que a porta 80 pertence ao mesmo sistema operacional. Uma forma de contornar este problema é forçar o Nmap a utilizar a porta 80 na identificação, como apresentado no Exemplo 2.25.

```
1  # nmap –O –p 80,135 192.0.2.254 ––osscan–guess
2  (...)
3  Interesting ports on 192.0.3.1:
4  PORT     STATE  SERVICE
5  80/tcp   open   http
6  135/tcp  closed msrpc
7  (...)
8  Device type: general purpose
9  Running (JUST GUESSING) : Linux 2.6.X (85%), OpenBSD 4.X (85%)
10 Aggressive OS guesses: Linux 2.6.13 – 2.6.27 (85%), OpenBSD 4.4 (85%), Linux
11 2.6.22 – 2.6.23 (85%)
12 No exact OS matches for host (If you know what OS is running on it, see
13 http://nmap.org/submit/ ).
14 (...)
```

Exemplo 2.25: Resultado do Nmap na presença de PAT (porta 80).

A utilização do parâmetro "`-osscan-guess`" é necessário porque o Nmap, ao não conseguir uma classificação exata, não discriminaria o sistema operacional mais parecido com a assinatura coletada. A utilização da porta 135 é feita apenas porque o Nmap também utiliza uma porta TCP fechada para a identificação.

Os testes para as portas 22 e 80 utilizando o SinFP são apresentados nos Exemplos 2.26 e 2.27. É assim confirmado que o SinFP não é afetado pela utilização de PAT.

```
1  # sinfp.pl -i 192.0.2.254 -p 22
2  (...)
3  IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.5
4  IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.6
5  IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.7
6  IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.8
7  IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 3.9
8  IPv4: HEURISTIC0/P1P2: BSD: OpenBSD: 4.0
9  (...)
```

Exemplo 2.26: Resultado do SinFP na presença de PAT (porta 22).

```
1  # sinfp.pl -i 192.0.2.254 -p 80
2  (...)
3  IPv4: HEURISTIC0/P1P2: GNU/Linux: Linux: 2.6.x
4  (...)
```

Exemplo 2.27: Resultado do SinFP na presença de PAT (porta 80).

Para finalizar os testes relacionados à tradução de endereços, é apresentado no Exemplo 2.28 o resultado da identificação realizada pela ferramenta Xprobe2.

```
1  # xprobe2 192.0.2.254
2  (...)
3  [+] Primary guess:
4  [+] Host 192.0.3.1 Running OS: "OpenBSD 3.7" (Guess probability: 100%)
5  [+] Other guesses:
6  [+] Host 192.0.3.1 Running OS: "OpenBSD 3.6" (Guess probability: 100%)
7  [+] Host 192.0.3.1 Running OS: "OpenBSD 3.5" (Guess probability: 100%)
8  [+] Host 192.0.3.1 Running OS: "OpenBSD 3.4" (Guess probability: 100%)
9  [+] Host 192.0.3.1 Running OS: "OpenBSD 2.5" (Guess probability: 96%)
10 [+] Host 192.0.3.1 Running OS: "OpenBSD 2.9" (Guess probability: 100%)
11 [+] Host 192.0.3.1 Running OS: "NetBSD 1.4" (Guess probability: 96%)
12 [+] Host 192.0.3.1 Running OS: "NetBSD 1.4.1" (Guess probability: 96%)
13 [+] Host 192.0.3.1 Running OS: "NetBSD 1.4.2" (Guess probability: 96%)
14 [+] Host 192.0.3.1 Running OS: "NetBSD 1.4.3" (Guess probability: 96%)
15 (...)
```

Exemplo 2.28: Resultado do Xprobe2 na presença de PAT.

A ferramenta Xprobe2 apresenta então o resultado menos eficaz, pois o SinFP apresentou os mesmos resultados obtidos nos testes iniciais (com o ambiente ideal para identificação), e o Nmap, apesar de se mostrar inadequado para utilização em ambientes de rede onde PAT é utilizado, possui mecanismos para contornar suas limitações.

2.4.2 Normalização de pacotes

Os testes agora serão realizados para avaliar o impacto ao acrescentar na configuração da *firewall* a normalização de pacotes. Para tanto basta substituir "`synproxy`" na linha 35 por "`keep`" no Exemplo 2.23. Os resultados para a ferramentas SinFP e Xprobe2

permanecem inalterados em relação aos testes anteriores relacionados a PAT. Porém, a ferramenta Nmap apresenta uma sensibilidade à normalização de pacotes. Esses resultados são apresentados nos Exemplos 2.29 e 2.30 para as portas 22 e 80, respectivamente.

```
1  # nmap -O -p 22,135 192.0.2.254 --osscan-guess
2  (...)
3  Device type: general purpose
4  Running (JUST GUESSING) : OpenBSD 4.X|3.X (98%)
5  Aggressive OS guesses: OpenBSD 4.2 - 4.4 (98%), OpenBSD 3.9 - 4.2 (96%),
6  OpenBSD 4.1 (x86) (95%), OpenBSD 4.2 (94%), OpenBSD 4.3 (94%), OpenBSD 4.0
7  (94%), OpenBSD 4.1 (93%), OpenBSD 3.5 (92%), OpenBSD 4.0 (x86) (92%), OpenBSD
8  4.4 (92%)
9  No exact OS matches for host (If you know what OS is running on it, see
10 http://nmap.org/submit/ ).
11 (...)
```

Exemplo 2.29: Resultado do Nmap na presença de Normalização (porta 22).

```
1  # nmap -O -p 80,135 192.0.2.254 --osscan-guess
2  (...)
3  Device type: general purpose
4  Running (JUST GUESSING) : Linux 2.6.X (87%), FreeBSD 7.X (85%)
5  Aggressive OS guesses: Linux 2.6.15 - 2.6.26 (87%), Linux 2.6.27 (Ubuntu 8.10)
6  (87%), Linux 2.6.22 (Ubuntu, x86) (85%), FreeBSD 7.0-RELEASE (85%)
7  No exact OS matches for host (If you know what OS is running on it, see
8  http://nmap.org/submit/ ).
9  (...)
```

Exemplo 2.30: Resultado do Nmap na presença de Normalização (porta 80).

A normalização de pacotes fez com que o sistema operacional da *firewall* deixasse de ser reconhecido de forma exata pelo Nmap. A identificação do sistema operacional da máquina Debian também sofreu alterações quanto a classificação.

2.4.3 Intermediação de sincronização

A intermediação do processo de sincronização via *three-way hand shake*, também conhecida como SYN *proxy* e TCP *proxy* [Postel 1981*c*], tem como objetivo impedir ataques de negação de serviço, ou DoS (*Denial of Service*), e ataques distribuídos de negação de serviço, ou DDoS (*Distributed Denial of Service*). Essa intermediação atrapalha o processo de identificação porque a resposta do pacote SYN+ACK enviada pela máquina remota é utilizada para criar a assinatura do sistema. O resultados para a ferramenta Nmap e SinFP são apresentados nos Exemplos 2.31 e 2.32, respectivamente.

```
1  # nmap -O -p 80,135 192.0.2.254 --osscan-guess
2  (...)
3  Device type: general purpose
4  Running (JUST GUESSING) : OpenBSD 4.X|3.X (90%), FreeBSD 6.X|7.X (89%), DEC
5  Digital UNIX 5.X (85%)
6  Aggressive OS guesses: OpenBSD 4.2 - 4.4 (90%), OpenBSD 4.2 (89%), FreeBSD
7  6.2-RELEASE-p2 (pf with scrub enabled) (89%), FreeBSD 7.0-RELEASE (88%),
8  OpenBSD 4.1 (88%), OpenBSD 4.0 (87%), FreeBSD 7.0-BETA4 (87%), OpenBSD 3.5
9  (85%), OpenBSD 3.9 (85%), OpenBSD 3.9 - 4.2 (85%)
10 No exact OS matches for host (If you know what OS is running on it, see
11 http://nmap.org/submit/ ).
12 (...)
```

Exemplo 2.31: Resultado do Nmap na presença de SYN *proxy* (porta 80).

```
# sinfp.pl -i 192.0.2.254 -p 80
P1: B11113 F0x12 W0 O0204ffff M64
P2: B11113 F0x12 W0 O0204ffff M1460
P3: B11123 F0x14 W0 O0 M0
IPv4: unknown

*** File [sinfp4-127.0.0.1.anon.pcap] generation done.
*** Please send it to sinfp@gomor.org if you think this is not
*** the good identification, or if it is a new signature.
*** In this last case, please specify 'uname -a' (or equivalent)
*** from the target host.
```

Exemplo 2.32: Resultado do SinFP na presença de SYN *proxy* (porta 80).

Nota-se que ambas as ferramentas (Nmap, no Exemplo 2.31, e SinFP, no Exemplo 2.32) não foram capazes de identificar os sistema operacional da máquina presente na porta 80. Para entender o motivo desta classificação incorreta, no caso de Nmap, e inviável, no caso do SinFP, o funcionamento de um SYN *proxy* é ilustrado na Figura 2.10. Neste sentido, é compreensível que o Nmap discrimine o sistema operacional presente na porta 80 como sendo OpenBSD, pois os pacotes de sincronização enviados pelo *proxy* são deste sistema. Como as informações relativas ao sistema operacional da máquina remota não são acessíveis diretamente, é conveniente que ferramentas identifiquem a utilização destes *proxies*.

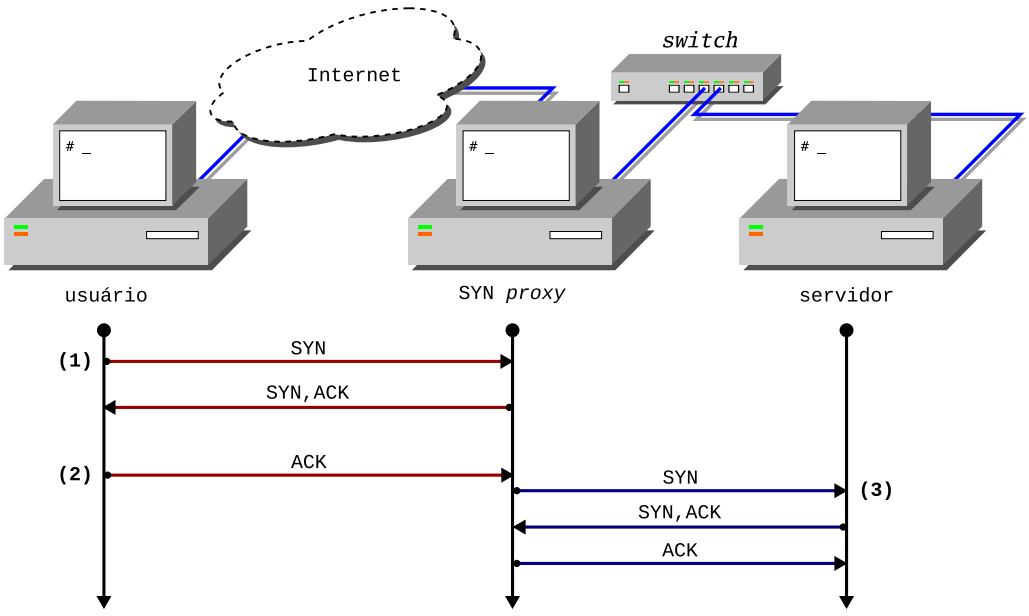

Figura 2.10: Ilustração do funcionamento de um SYN *proxy*: (1) a mensagem de sincronização enviada pelo usuário para um servidor é interceptada pelo *proxy*, que por sua vez assume o papel do servidor na sincronização; (2) no caso de ataques DoS e DDoS o processo de sincronização não é confirmado nem cancelado pela máquina de origem; (3) com a confirmação do processo de sincronização por parte do usuário o *proxy* realiza outro processo de sincronização com o servidor afim de estabelecer a conexão do usuário com o mesmo.

A não realização de testes com o Xprobe2 se deve ao fato de que esta ferramenta não utiliza mensagens de sincronização em seu processo de identificação.

2.5 Testes de confiabilidade

Como descrito na Subseção 1.3.5 do Capítulo 1, a confiabilidade é a capacidade de uma ferramenta em não ser enganada por métodos de mimetismo TCP/IP. Para por a prova a confiabilidade de cada uma das ferramentas utilizou-se o Honeyd. O Honeyd utiliza as bases de dados de assinaturas do Nmap e do Xprobe2 para imitar sistemas operacionais de acordo com suas implementações da pilha TCP/IP. Para configurar uma rede virtual em que existe uma máquina Linux e um outro grupo de máquinas Windows utilizou-se o arquivo de configuração apresentado no Exemplo 2.33.

```
1  ############################################################################
2  # Conjunto de regras utilizadas para criar um honeypot virtual             #
3  ############################################################################
4
5  # Criacao da maquina virtual Linux que representa o servidor
6  create linux
7  set linux personality "Linux 2.4.16 - 2.4.18"
8  set linux default tcp action reset
9  set linux default udp action reset
10 add linux tcp port 22 "sh scripts/test.sh"
11 add linux udp port 53 "sh scripts/test.sh"
12 bind 192.0.2.8 linux
13
14 # Criacao da maquina virtual default que representa as estacoes de trabalho
15 create default
16 set default personality "Microsoft Windows XP SP1"
17 set default default tcp action reset
18 add default tcp port 135 "sh scripts/test.sh"
19
20 ############################################################################
```

Exemplo 2.33: Arquivo de configuração (`/etc/honeyd.conf`) do Honeyd.

Desta forma a máquina cujo endereço é `192.0.2.8` se comportará como o Linux (linha 7 do Exemplo 2.33) e as demais máquinas (como por exemplo `192.0.2.9`) se comportarão como Windows (linha 17 do mesmo exemplo).

2.5.1 Nmap

O Honeyd utiliza a base de dados da primeira geração do sistema de identificação do Nmap [Provos & Holz 2008, Fyodor 1998]. O Nmap utilizado neste trabalho utiliza a segunda geração deste sistema de identificação [Fyodor 2006, Fyodor 2009*b*]. Por este motivo o Honeyd não consegue (ainda) enganar da mesma forma as novas versões do Nmap. Porém, utilizando a opção "`-osscan-guess`" é possível verificar que o Nmap ainda é susceptível à imitação exercida pelo Honeyd. Nos Exemplos 2.34 e 2.35 são apresentados os resultados do Nmap para as duas emulações de sistema operacional.

```
1  # nmap -O -p 22,135 192.0.2.8 --osscan-guess
2  (...)
3  Device type: general purpose
4  Running (JUST GUESSING) : Linux 2.4.X (86%)
5  Aggressive OS guesses: Linux 2.4.20-grsec (86%)
6  No exact OS matches for host (If you know what OS is running on it, see
7  http://nmap.org/submit/ ).
8  (...)
```

Exemplo 2.34: Resultado do Nmap na presença do Honeyd emulando Linux.

```
1  # nmap -O -p 22,135 192.0.2.9 --osscan-guess
2  (...)
3  Aggressive OS guesses: Microsoft Windows 2000 Pro SP4 (90%), Microsoft Windows
4  2000 SP4 (89%), Sony PSP game console (modified, running Custom Firmware 3.90
5  M33-2) (88%), Microsoft Windows 2000 Server SP4 (87%), Microsoft Windows 2000
6  SP0/SP1/SP2 or Windows XP SP0/SP1 (87%), Microsoft Windows 2000 SP3/SP4 or
7  Windows XP SP1/SP2 (87%), Microsoft Windows XP Professional SP1 (87%),
8  Microsoft Windows XP SP1 (87%), Microsoft Windows XP SP3 (87%), NetBSD 1.4.2 -
9  1.5.2; Lanier LS232c, NRG DSc428, or Savin 8055 printer; or Panasonic Network
10 Camera (BB-HCM331, BB-HCM381, BCL-30A, BL-C1CE, or BL-C10CE) (87%)
11 No exact OS matches for host (If you know what OS is running on it, see
12 http://nmap.org/submit/ ).
13 (...)
```

Exemplo 2.35: Resultado do Nmap na presença do Honeyd emulando Windows.

Nos dois casos foi possível enganar o Nmap quando a opção "`-osscan-guess`" foi utilizada. Provavelmente, as próximas versões do Honeyd devem melhorar seu desempenho utilizando a base de dados da segunda geração do sistema de identificação de sistemas operacionais do Nmap [Fyodor 2006].

2.5.2 SinFP

Os resultados dos testes com o SinFP são apresentados nos Exemplos 2.36 (Honeyd emulando Linux) e 2.37 (Honeyd emulando Windows). Nos dois testes realizados a ferramenta SinFP não foi enganada pelo Honeyd, provavelmente porque esta ferramenta é nova e certamente não foi considerada ainda no sistema de mimetismo do Honeyd [Provos & Holz 2008].

```
1  # sinfp.pl -i 192.0.2.8 -p 22
2  P1: B11113 F0x12 W5792 O0204ffff0101080affffffff0000000001030300 M1460
3  P2: B11113 F0x12 W5792 O0204ffff0101080affffffff4445414401030300 M1460
4  P3: B11020 F0x04 W0 O0 M0
5  IPv4: unknown
6  (...)
```

Exemplo 2.36: Resultado do SinFP na presença do Honeyd emulando Linux.

```
1  # sinfp.pl -i 192.0.2.9 -p 135
2  P1: B11113 F0x12 W64240 O0204ffff010303000101080affffffff00000000 M1460
3  P2: B11113 F0x12 W64240 O0204ffff010303000101080affffffff44454144 M1460
4  P3: B11020 F0x04 W0 O0 M0
5  IPv4: unknown
6  (...)
```

Exemplo 2.37: Resultado do SinFP na presença do Honeyd emulando Windows.

Através das linhas 2 e 3 de ambos os testes do SinFP, vemos que, apesar de não classificar os sistemas operacionais mimetizados pelo Honeyd, ele apresentada valores diferentes para cada alvo em seus testes `P1` e `P2`. Este fato pode ser levado em consideração para acreditar que alguns ajustes no Honeyd podem fazer com que seja possível também enganar o SinFP.

2.5.3 Xprobe2

O Xprobe2 foi o que obteve piores resultados. Isto porque o grau de certeza atribuído a sua classificação supera aquele o do Nmap. Nos dois casos essa ferramenta discriminou com um grau de certeza razoável os sistemas operacionais imitados pelo Honeyd. No caso do teste com a simulação do sistema Linux (Exemplo 2.38) este grau de certeza foi igual a 92%. No caso do teste com a simulação do sistema Windows (Exemplo 2.39) este grau foi igual a 96%.

```
# xprobe2 192.0.2.8
(...)
[+] Primary guess:
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.19" (Guess probability: 92%)
[+] Other guesses:
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.5" (Guess probability: 92%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.17" (Guess probability: 92%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.7" (Guess probability: 92%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.15" (Guess probability: 92%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.9" (Guess probability: 96%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.13" (Guess probability: 92%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.11" (Guess probability: 96%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.11" (Guess probability: 96%)
[+] Host 192.0.4.1 Running OS: "Linux Kernel 2.2.13" (Guess probability: 92%)
(...)
```

Exemplo 2.38: Resultado do Xprobe2 na presença do Honeyd emulando Linux.

```
# xprobe2 192.0.2.9
(...)
[+] Primary guess:
[+] Host 192.0.4.2 Running OS: "Microsoft Windows XP SP1" (Guess probability:
    96%)
[+] Other guesses:
[+] Host 192.0.4.2 Running OS: "Microsoft Windows XP" (Guess probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Server Service Pack 4"
    (Guess probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Server Service Pack 3"
    (Guess probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Server Service Pack 2"
    (Guess probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Server Service Pack 1"
    (Guess probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Server" (Guess
    probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Workstation SP4" (Guess
    probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Workstation SP3" (Guess
    probability: 96%)
[+] Host 192.0.4.2 Running OS: "Microsoft Windows 2000 Workstation SP2" (Guess
    probability: 96%)
(...)
```

Exemplo 2.39: Resultado do Xprobe2 na presença do Honeyd emulando Windows.

Na Tabela 2.4 é apresentado um resumo do desempenho de cada ferramenta avaliada. Em termos gerais todas elas falham a partir do ponto em que o SYN *proxy* é utilizado. Além disso, nota-se que o SinFP foi a ferramenta que demonstrou ser mais robusta a utilização de ambientes hostis ao processo de TCP/IP *fingerprinting*. Da Tabela 2.4 retira-se ainda os requisitos da ferramenta proposta neste trabalho. Neste sentido, busca-se um

método de identificação imune a utilização de PAT, normalização de pacotes e capaz de identificar SYN *proxies* e o Honeyd.

Configuração da rede	Nmap	SinFP	Xprobe2
Ausência de adversidades	●	●$^{(a)}$	●$^{(a)}$
Alvo acessados via PAT	◑$^{(b)}$	●	○$^{(c)}$
Normalização de pacotes	◑$^{(d)}$	●	○
Utilização de SYN *proxy*	○	○	○
Alvo criado com Honeyd	○	○$^{(e)}$	○

Tabela 2.4: Resultado a avaliação das ferramentas. Sendo ●, sucesso; ◑, imprecisão; e ○, falha. Onde a notas indicam: (a) incapacidade de diferenciar Windows 2000 e Windows XP; (b) Debian reconhecido como: Linux 2.6.X – 85%, OpenBSD 4.X – 85%; (c) deficiência por utilizar somente informações da camada de rede, e assim não consegue diferenciar por porta (camada de transporte); (d) Debian reconhecido como: Linux 2.6.X – 87%, FreeBSD 7.X – 85%; e (e) imitação não reconhecida, sistemas classificados como desconhecidos.

Alguns testes foram realizados a fim de encontrar alguma informação que trafegasse livremente pela rede e não fosse alterada pelo PAT e pela normalização de pacotes. Uma informação específica presente no cabeçalho TCP não sofreu influência do PAT da normalização de pacotes implementados pela *firewall* do OpenBSD PF. Posteriormente, foi verificado também que esta informação é gerada de forma singular pelo SYN *proxy* do PF e pelo Honeyd, podendo, assim, caracterizar a utilização destes. Esta informação é denominada TCP ISN (*Initial Sequence Number*). O TCP ISN também é utilizado pelas ferramentas testadas, porém, essas ferramentas não exploram toda a capacidade de caracterização oferecida por esse dado.

2.6 Proposta

O Polonês Michal Zalewski, publicou um trabalho onde é realizada uma análise dos geradores de número de seqüência inicial, ou ISN, da implementação do protocolo TCP de diferentes sistemas operacionais [Zalewski 2001]. Os TCP ISNs são responsáveis por manter a coerência em comunicações TCP, de forma a evitar segmentos duplicados originados da reutilização de seqüências de conexões anteriores (*previous connection incarnations*) [Postel 1981*c*]. A forma como a geração desses números é implementada pode acarretar em problemas de segurança. Após a descoberta desses problemas, foi criada, em 1996, uma nova recomendação para geração desses números, descrita na RFC 1948 [Bellovin 1996]. Em seu trabalho, Michal Zalewski, mostrou, dentre outras coisas, que vários sistemas operacionais tem uma forma singular de implementar a geração desses números [Zalewski 2001, Zalewski 2002].

Para utilizar os TCP ISNs como dados para criação de assinaturas para realização de OS *fingerprinting* serão analisados os geradores de números pseudo-aleatórios, ou PRNG (*Pseudo-Random Number Generator*), dos SOs analisados. Para extrair esta informação

é necessário compreender como TCP ISNs são gerados, pois o termo gerado pelo PRNG está, geralmente, embutido nesses números. A recomendação atual para a geração destes números através de uma função $G_{isn}(t)$ é expressa como [Bellovin 1996]:

$$G_{isn}(t) = M(t) + F(\cdot) \tag{2.1}$$
$$M(t) = M(t-1) + R(t) \tag{2.2}$$
$$F(\cdot) = f(connection\_id, secret\_key) \tag{2.3}$$

onde $G_{isn}(t)$ é a função responsável pela geração do número inicial de seqüência no instante t, $M(t)$ é uma função composta pelo seu valor anterior acrecido do valor da função $R(t)$ e $F(\cdot)$ consiste em aplicar uma função $f(\cdot)$ ao identificador da conexão, composto pelos endereços e pelas portas de origem e destino e a uma chave secreta (opcional).

Tem-se como objetivo estimar a função $R(t)$ utilizando apenas amostras de $G_{isn}(t)$. Para realizar esta tarefa é importante notar que $F(\cdot)$ é constante para um mesmo identificador de conexão (*connection_id*) dada uma instância do sistema operacional (entenda-se por instância uma inicialização em um determinado instante no tempo). Porém, em alguns sistemas, a chave secreta (*secret_key*) é modificada periodicamente. Desta forma, pode-se obter a partir das Equações 2.1, 2.2 e 2.3 uma estimativa, $\hat{R}(t)$, da função $R(t)$:

$$\hat{R}(t) = G_{isn}(t) - G_{isn}(t-1). \tag{2.4}$$

O processo de aquisição de amostras é ilustrado na Figura 2.11. Uma característica a ser considerada é que, intervalos de envio de pacotes SYN suficientemente curtos, podem caracterizar um ataque SYN *flooding* caso mensagens RST não sejam enviadas em resposta para a mensagem SYN+ACK da máquina alvo [Eddy 2007].

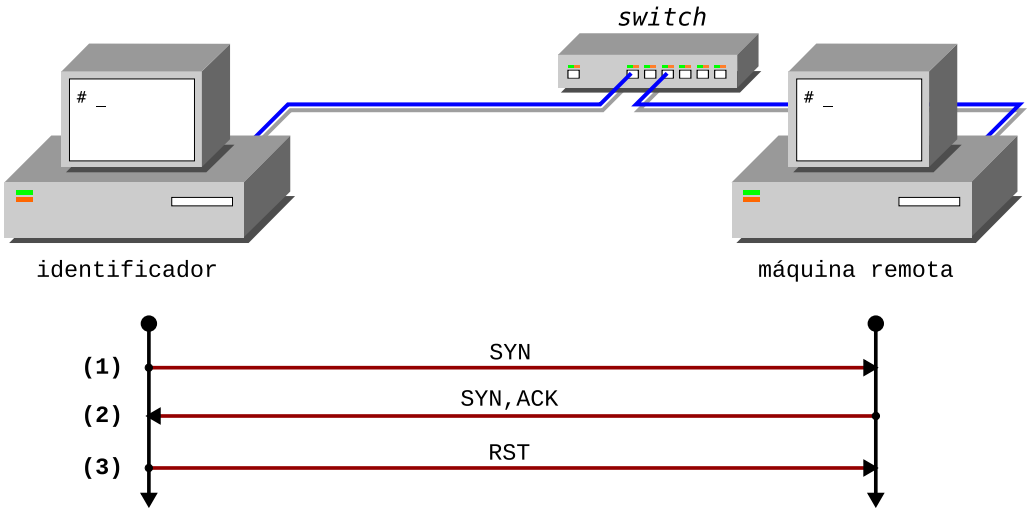

Figura 2.11: Ilustração da aquisição de amostras de TCP ISNs: (1) envio da mensagem de sincronização SYN; (2) recebimento da mensagem que confirma a sincronização SYN+ACK e aquisição do TCP ISN; (3) envio da mensagem RST para cancelar a sincronização a fim de evitar SYN *flooding*.

Enquanto em alguns sistemas a mudança do ISN aparenta ser feita por demanda (sempre que um ISN diferente era recebido, não importando o quão pequeno era o intervalo entre o envio das mensagens), em outros sistemas o intervalo de mudança do ISN aparenta ser periódico. O valor de amostragem escolhido deve levar em consideração os dois casos. Além disso, a freqüência com é desejável que este valor seja o menor possível, pois isto implica na diminuição do tempo de aquisição das amostras. Após alguns testes, foi verificado que o valor 10^{-2}s (dez milissegundos) satisfaz esses requisitos para os sistemas operacionais analisados um uma rede local. Na Tabela 2.5, são apresentadas vinte amostras da função $G_{isn}(t)$ para cada um dos sistemas operacionais analisados.

Debian	FreeBSD	NetBSD	OpenBSD	OpenSolaris	Win. 2000	Win. XP
473803361	1611390248	1648042687	1399492081	3238261495	1857333441	4170711277
473815220	1611390248	1667717403	4107840975	3238291347	1857367314	4170741137
473826508	1611390248	1679801257	3875137240	3238509855	1857420076	4170796548
473838502	1611390248	1704048339	1138545661	3238611494	1857460833	4170825212
473850798	1611390248	1716241872	3730234074	3238689975	1857503030	4170849444
473863174	1611390248	1729958926	1160744693	3238834359	1857558205	4170865951
473874586	1611390248	1760341144	144331379	3239039977	1857608086	4170895235
473886573	1611390248	1761977139	4018493	3239092293	1857696402	4170914864
473898490	55578985	1795012198	3788692605	3239269660	1857738357	4170931957
473910483	55578985	1810843641	3775967235	3239324619	1857802611	4170963250
473923114	55578985	1825938559	1024752262	3239494978	1857863071	4170981602
473934562	55578985	1840781929	80982006	3239644709	1857897850	4171036393
473946502	55578985	1859998782	2565329183	3239815233	1857957682	4171055806
473958521	55578985	1869942211	2748466912	3239911720	1858006144	4171076290
473971199	55578985	1888065985	2722553507	3240001761	1858067366	4171095073
473982479	55578985	1904640747	4184720619	3240089279	1858136363	4171119700
473994630	55578985	1923261815	4213820032	3240261499	1858201580	4171138461
474006501	55578985	1930456100	1516844101	3240344973	1858238104	4171164595
474018933	55578985	1952214833	598741424	3240480599	1858278853	4171185077
474030472	55578985	1969833702	2225294446	3240693935	1858320087	4171232517

Tabela 2.5: Amostras da função $G_{isn}(t)$ dos sistemas operacionais analisados.

Ao se analisar os valores apresentados nesta tabela, nota-se que os sistemas operacionais FreeBSD e OpenBSD não aparentam seguir a recomendação proposta pela RFC 1948. As amostras desses dois sistemas não apresentam um comportamento incremental que é causado pelo uso da função $M(t)$ (Equação 2.2) na geração do inicial número de seqüência. Além do fato dos outros sistemas operacionais apresentarem o comportamento incremental nos números de seqüência iniciais, foi verificado que, alterando o valor das portas de origem, o valor da seqüência perdia o comportamento incremental. Esse fato indica que a função $F(\cdot)$ também é utilizada pelo gerador de ISNs desses sistemas operacionais. Portanto, concluiu-se que as versões analisados dos sistemas operacionais Debian, NetBSD, OpenSolaris, Windows 2000 e Windows XP adotam a recomendação proposta pela RFC 1948. Nos casos em que a recomendação proposta pela RFC 1948 não for adotada, serão utilizadas as próprias amostras da função $G_{isn}(t)$ no lugar da estimativa $\hat{R}(t)$. Na Tabela 2.6, são apresentadas as estimativas para amostras do PRNG de cada sistema operacional.

Debian	FreeBSD	NetBSD	OpenBSD	OpenSolaris	Win. 2000	Win. XP
235335	1611390248	19674716	1399492081	29852	33873	29860
249909	1611390248	12083854	4107840975	218508	52762	55411
250028	1611390248	24247082	3875137240	101639	40757	28664
249975	1611390248	12193533	1138545661	78481	42197	24232
250001	1611390248	13717054	3730234074	144384	55175	16507
250053	1611390248	30382218	1160744693	205618	49881	29284
249949	1611390248	1635995	144331379	52316	88316	19629
249988	1611390248	33035059	4018493	177367	41955	17093
250027	55578985	15831443	3788692605	54959	64254	31293
249988	55578985	15094918	3775967235	170359	60460	18352
250002	55578985	14843370	1024752262	149731	34779	54791
250014	55578985	19216853	80982006	170524	59832	19413
250001	55578985	9943429	2565329183	96487	48462	20484
250080	55578985	18123774	2748466912	90041	61222	18783
249988	55578985	16574762	2722553507	87518	68997	24627
249962	55578985	18621068	4184720619	172220	65217	18761
250001	55578985	7194285	4213820032	83474	36524	26134
250001	55578985	21758733	1516844101	135626	40749	20482
250027	55578985	17618869	598741424	213336	41234	47440
249988	55578985	17432720	2225294446	133922	53862	22556

Tabela 2.6: Amostras estimadas $\hat{R}(t)$ dos sistemas operacionais analisados.

Na Tabela 2.7, são apresentadas a média amostral e o desvio padrão de um conjunto de amostras de $\hat{R}(t)$ para cada sistema operacional analisado a partir de 100000 (cem mil) amostras de ISNs.

Sistema	$\hat{R}(t)$	$\overline{\mu}$	$\hat{\sigma}$
Debian	$\Delta G_{isn}(t)$	≈ 228332	≈ 119817
FreeBSD	$G_{isn}(t)$	≈ 2139331259	≈ 1237528682
NetBSD	$\Delta G_{isn}(t)$	≈ 17221761	≈ 7380255
OpenBSD	$G_{isn}(t)$	≈ 2144262074	≈ 1242845940
OpenSolaris	$\Delta G_{isn}(t)$	≈ 127999	≈ 53423
Windows 2000	$\Delta G_{isn}(t)$	≈ 52437	≈ 12668
Windows XP	$\Delta G_{isn}(t)$	≈ 27650	≈ 9606

Tabela 2.7: Média amostral e desvio padrão das amostras de $\hat{R}(t)$.

O valor da média e do desvio padrão destas séries seria uma informação que talvez pudesse ser utilizada para classificar estes sistemas. Porém, como pode ser verificado na Tabela 2.7, o OpenBSD e o FreeBSD, apesar de bem distintos em relação forma como geram os ISNs, possuem valores próximos para esses dois parâmetros. Na Figura 2.12, são apresentados esboços das 100 primeiras amostras da função $\hat{R}(t)$ de cada sistema operacional e do Honeyd. Essa representação gráfica de cada uma destas séries evidencia o quão cada uma é diferente das outras.

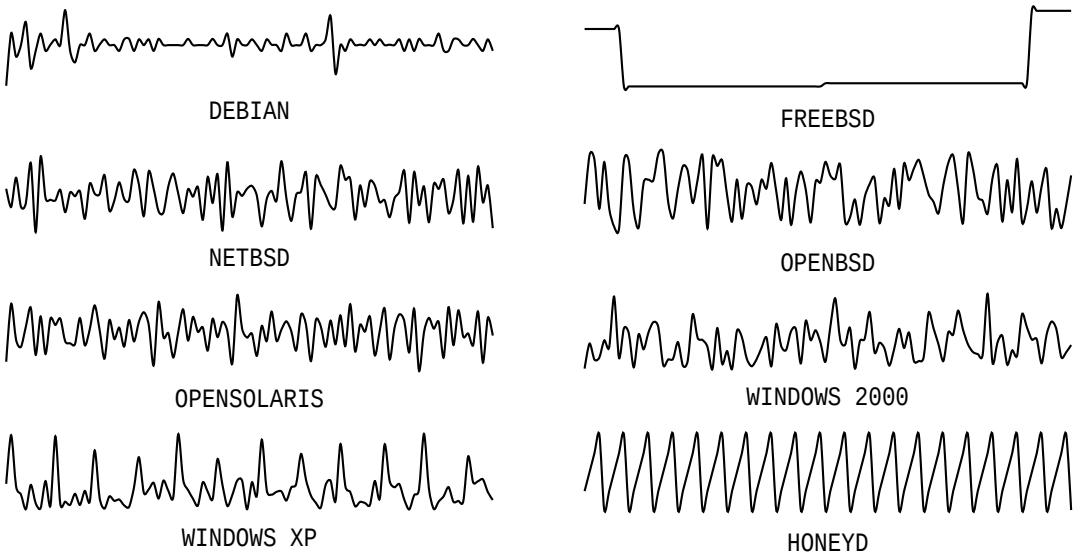

Figura 2.12: Esboço das series formadas por 100 amostras de $\hat{R}(t)$.

A proposta deste trabalho consiste, então, em criar assinaturas de sistemas operacionais e classificá-las utilizando amostras estimadas do gerador de números pseudo-aleatórios associados ao TCP ISN. No Capítulo 3, será apresentada a fundamentação teórica necessária para realizar esta tarefa.

Capítulo 3

Fundamentação

Este capítulo tem como objetivo apresentar a fundamentação teórica necessária para criação do método de identificação proposto. Para tanto, serão apresentados métodos para caracterização de sistemas dinâmicos e algoritmos de classificação.

3.1 Atratores e Espaço de Fase

Neste trabalho, o gerador de números pseudo-aleatórios é tratado como um sistema dinâmico cuja única informação disponível é uma série de amostras da saída do sistema. Neste caso, essas amostras são extraídas dos números de seqüências iniciais do cabeçalho TCP. Em acordo com o processo de identificação apresentado na Figura 1.1, deve-se criar uma forma de caracterizar sistemas operacionais utilizando a informação disponível, neste caso, as mostras de saída dos PRNGs. A utilização de amostras do ISNs impõem, ainda, a restrição de que a representação deve ser baseada apenas nos dados coletados experimentalmente. Estas caracterizações podem ser feitas utilizando atratores.

Atratores representam o espaço de estados de um sistema e como este sistema evolui ao longo do tempo. O espaço utilizado para representar atratores é denominado Espaço de Fase (*Phase Space*), onde cada ponto do espaço de fase pode representar um estado de um sistema [Ott 2002]. A construção de atratores a partir de amostras da saída do sistema em análise, representado por $s(t)$, é feita utilizando um método denominado Coordenadas de Atraso (ou *Delay Coordinates*) [Baker & Gollub 1996].

O método de criação de atratores baseado em coordenadas de atraso é utilizado quando não se conhece as equações que representam o sistema dinâmico, mas é possível observar a saída $s(t)$ do sistema. Cada ponto $\mathbf{x}$ de um espaço de fase de dimensão m é construído a partir de atrasos seguidos aplicados à função $s(t)$ [Alligood et al. 1997]:

$$[x_1, x_2, x_3, \ldots, x_m] = [s(t), s(t-\tau), s(t-2\tau), \ldots, s(t-(m-1)\tau)] \tag{3.1}$$

onde para cada coordenada do espaço de fase é atribuído um atraso de razão τ a saída observada do sistema $s(t)$. O atrator é obtido como resultado de sucessivas utilizações da Equação 3.1 a medida que o tempo t evolui. Tem-se que, para n amostras de $s(t)$ e um espaço de fase de dimensão m, o número de pontos do atrator é dado por $n-(m-1)$.

Geralmente, quando se trata de reconstrução de atratores, a dimensionalidade do espaço de fase é desconhecida a priori. Porém, pode estimar esta dimensão de forma iterativa, aumentando gradativamente a dimensionalidade do espaço de fase, e estimam assim a dimensão do atrator gerado [Grassberger & Procaccia 1983]. O aumento na dimensionalidade é realizado até que a dimensão do atrator reconstruído não sofra alterações significativas [Baker & Gollub 1996]. Exemplos de dimensões que podem ser utilizadas são: a dimensão de capacidade (*capacity dimension* ou Minkowski-Bouligand *dimension*), a dimensão de correlação (*correlation dimension*), a dimensão de informação (*information dimension*) e, de forma geral, as dimensões de Rényi [Baker & Gollub 1996]. Em se tratando de dados experimentais, a dimensão de correlação é a mais utilizada [Grassberger & Procaccia 2004].

Como o resultado desejado neste trabalho é apenas criar uma caracterização para o PRNG dos sistemas operacionais utilizados, é suficiente que os atratores sejam distinguíveis entre si. Por este motivo, foi utilizada a mesma dimensão para o espaço de fase para os atratores de cada sistema operacional, qual seja, bidimensional. Porém, para efeito de comparação, construções de atratores de cada sistema operacional em espaço de fase tridimensional são apresentadas no Apêndice A. Sendo assim, para cada conjunto de amostras de $\hat{R}(t)$ será construído uma representação bidimensional onde cada ponto (x,y) dessa representação é definida como descrito da seguinte forma:

$$[x,y] = [s(t), s(t-\tau)] \tag{3.2}$$

$$[x,y] = [\hat{R}(t), \hat{R}(t-1)] \tag{3.3}$$

onde, em acordo com os experimentos realizados no Capítulo 2, $\tau = 10^{-2}$, ou seja, a equivalência entre as Equações 3.2 e 3.3 se deve ao fato de que o intervalo de amostragem utilizado para construir a estimativa $\hat{R}(t)$ foi de 10^{-2} segundos.

Assim como a dimensionalidade, o intervalo de amostragem τ pode ser estimado com base nos dados. Métodos baseados na correlação média [Albano et al. 1988], entropia [Fraser 1989] e informação mútua [Fraser & Swinney 1986] dos pontos que compõem o atrator são bem definidos na literatura [Baker & Gollub 1996]. Porém, assim como a dimensão do espaço de fase, o valor de amostragem utilizado deve ser o mesmo para cada atrator. Por este motivo, e considerando a motivação apresenta no final do Capítulo 2, o valor utilizado para τ foi 10^{-2} segundos.

Para construção dos atratores foram capturadas 100.000 (cem mil) amostras de números iniciais de seqüência de cada sistema operacional listado na Tabela 2.2. Este número de amostras foi escolhido porque acredita-se ser suficientemente grande para representar estatisticamente o PRNG de cada SO. Apenas para efeito de comparação, em seu trabalho, Michal Zalewski utilizou aproximadamente 50.000 (cinqüenta mil) amostras [Zalewski 2001]. Utilizando os critérios de estimação da função $R(t)$ apresentados no final do Capítulo 2 foram criadas as estimativas $\hat{R}(t)$ de cada sistema operacional. Detalhes sobre estas estimativas e a apresentação dos atratores de cada sistema operacional são apresentados a seguir. Após a apresentação dos dados de cada sistema, serão ainda analisadas as amostras de números iniciais de seqüência do Honeyd e do SYN *proxy* implementado pelo *firewall* PF.

3.1.1 Debian

O incremento aleatório nas amostras de números iniciais de seqüência do Debian sugerem que sua implementação adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída em acordo com a Equação 2.4. Os 100 primeiros valores desta estimativa são apresentados na Figura 3.1.

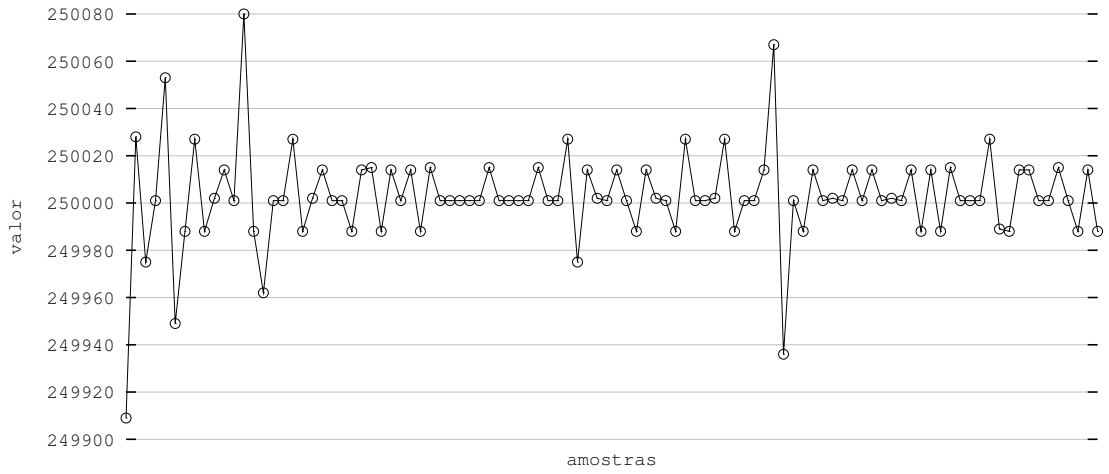

Figura 3.1: 100 amostras da função $\hat{R}(t)$ do Debian.

A estimativa $\hat{R}(t)$ do Debian possui média amostral $\overline{\mu}_{debian} = 228332,460505$ e desvio padrão $\sigma_{debian} = 119817,124687$. O atrator reconstruído é apresentado na Figura 3.2.

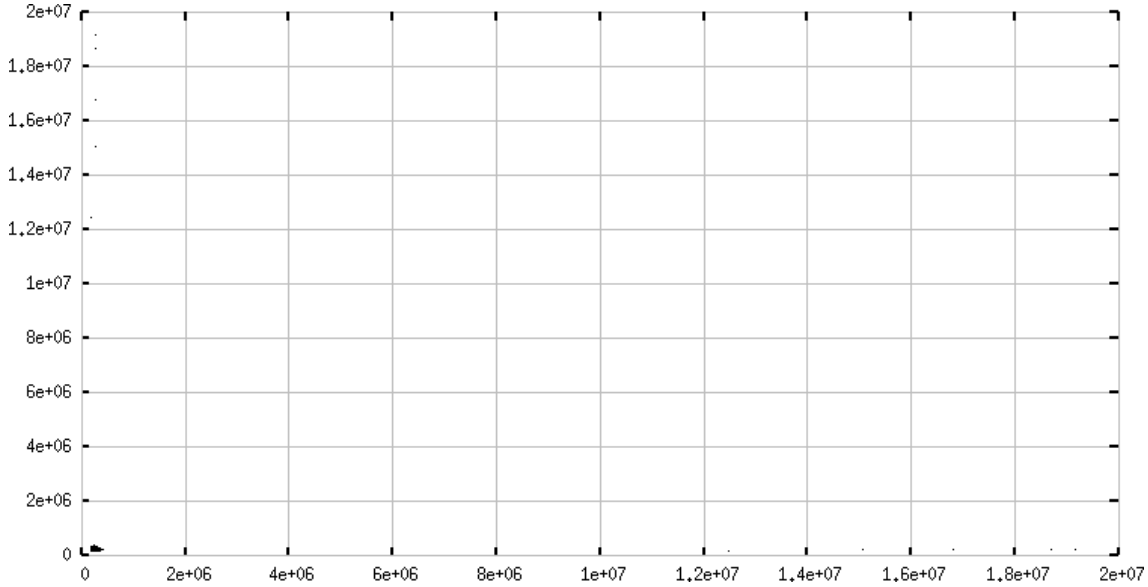

Figura 3.2: Atrator do Debian.

A presença de alguns valores altos na estimativa $\hat{R}(t)$ fazem com que a parte onde há maior concentração de pontos no atrator não seja bem expressa na Figura 3.2. Por este motivo este atrator foi ampliado próximo ao eixo de origem nas Figuras 3.3 e 3.4.

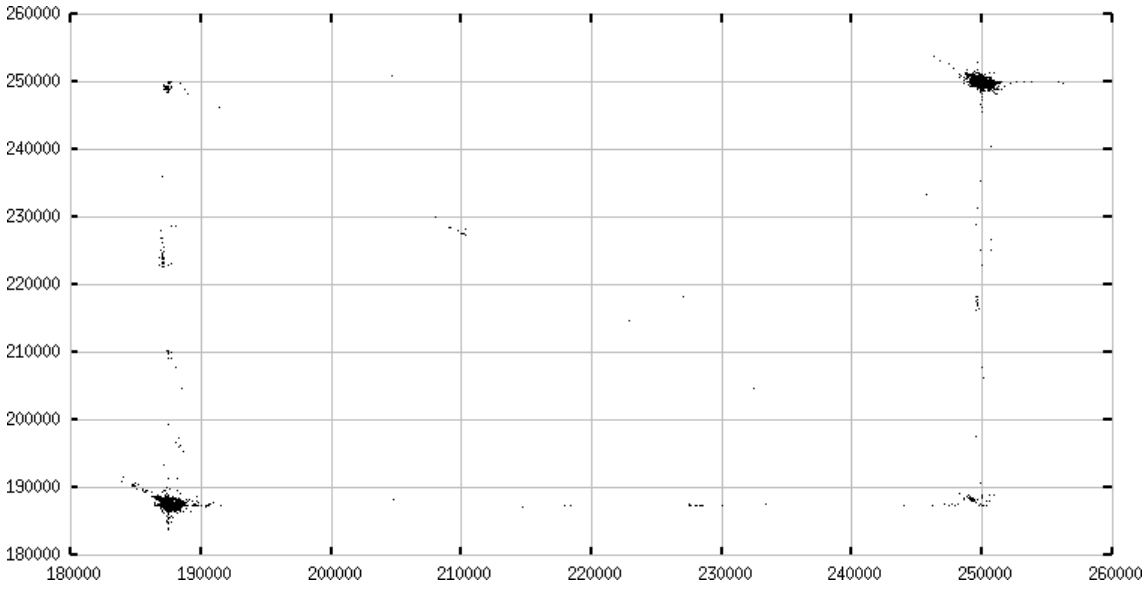

Figura 3.3: Atrator do Debian (ampliado de 180.000 a 260.000).

Sendo o centro da Figura 3.3 aproximadamente a média amostral de $\hat{R}(t)$, pode-se afirmar que a maioria dos pontos do atrator é apresentada nesta figura.

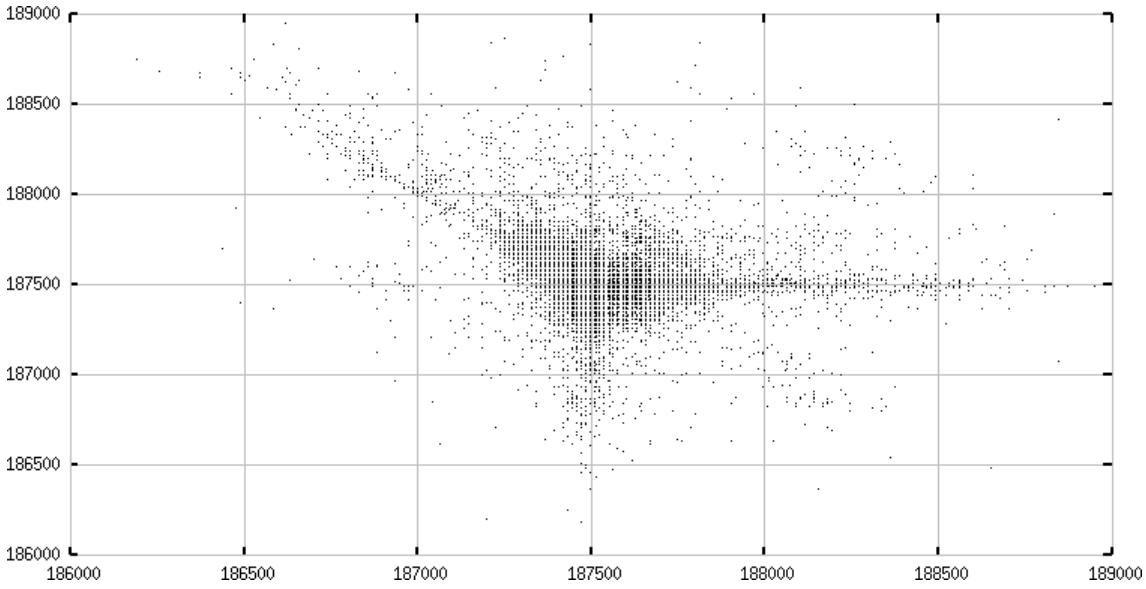

Figura 3.4: Atrator do Debian (ampliado de 186.000 a 189.000).

O atrator do Debian apresenta um comportamento singular em relação aos outros sistemas operacionais. Isto ocorre porque dois pontos de atração distintos e bem definidos podem ser visualmente determinados. Um deles em $(187.500, 187.500)$ e o outro em $(250.000, 250.000)$. Existem outros pontos de atração, porém, de menor influência. A Figura 3.4 sugere ainda que a função aleatória não segue uma distribuição de probabilidade condicional uniforme.

3.1.2 FreeBSD

As amostras do gerador de números iniciais de seqüência do FreeBSD sugerem que sua implementação não adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída utilizando as próprias amostras de $G_{isn}(t)$, sendo os 100 primeiros valores apresentados na Figura 3.5.

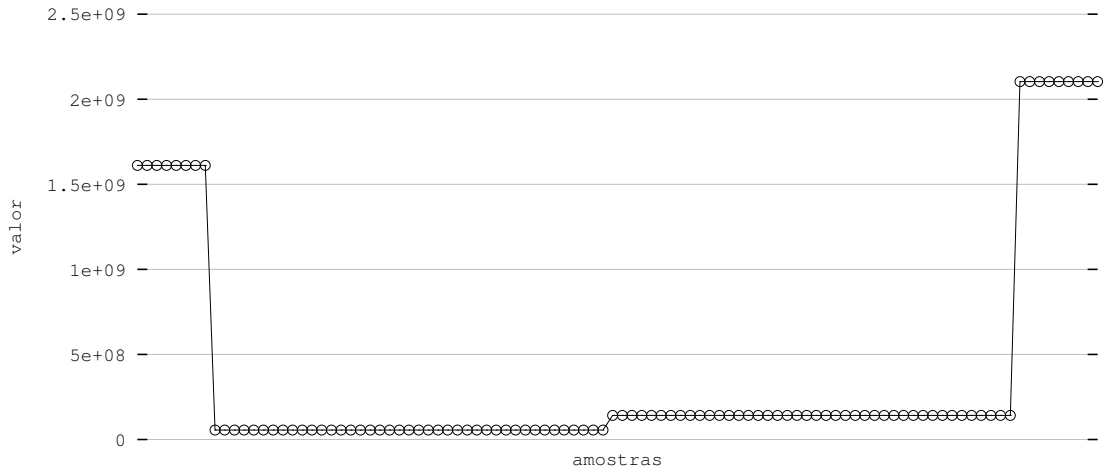

Figura 3.5: 100 amostras da função $\hat{R}(t)$ do FreeBSD.

A Figura 3.5 mostra que o período de mudança do número aleatório é de aproximadamente $0,42s$ (42×10^{-2}). A estimativa possui média $\overline{\mu}_{freebsd} = 2139331259,15$ e desvio padrão $\sigma_{freebsd} = 1237528682,85$. O atrator reconstruído é apresentado na Figura 3.6.

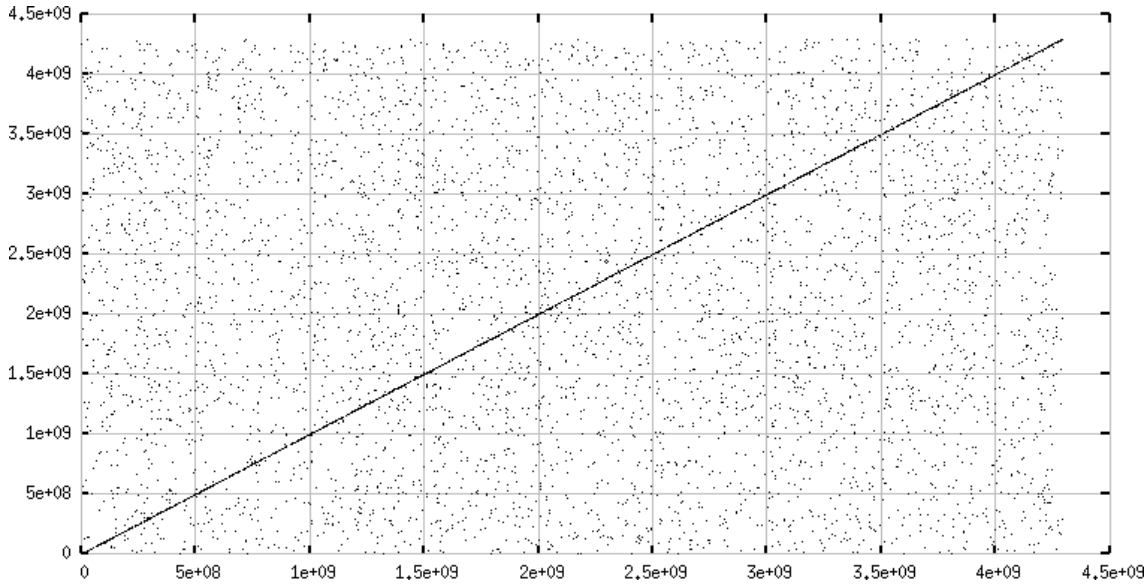

Figura 3.6: Atrator do FreeBSD.

A Figura 3.6 sugere que a função aleatória segue uma distribuição de probabilidade condicional uniforme. A diagonal deve-se ao baixo valor (relativo) de τ utilizado.

3.1.3 NetBSD

As amostras do gerador de números iniciais de seqüência do NetBSD sugerem que sua implementação adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída em acordo com a Equação 2.4. Os 100 primeiros valores desta estimativa são apresentados na Figura 3.7.

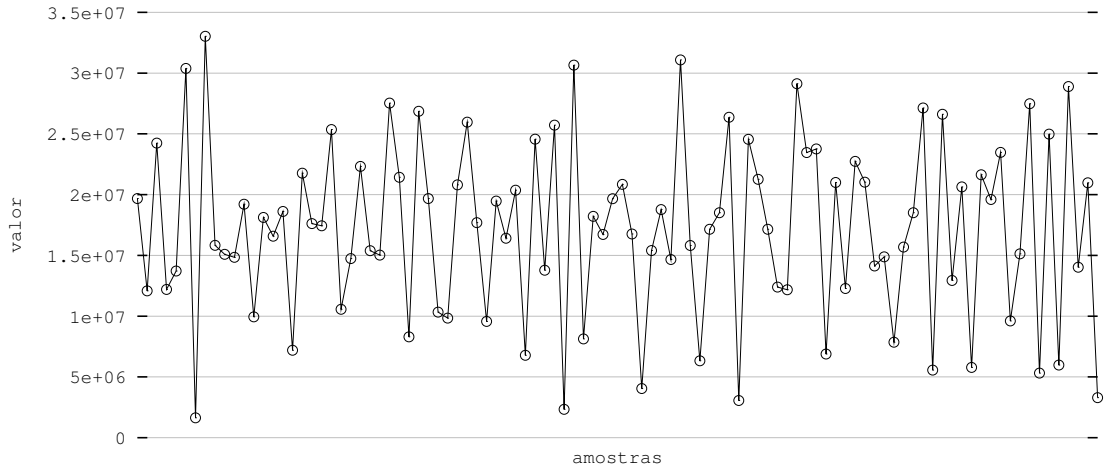

Figura 3.7: 100 amostras da função $\hat{R}(t)$ do NetBSD.

A estimativa $\hat{R}(t)$ do NetBSD possui média $\overline{\mu}_{netbsd} = 17221761,1994$ e desvio padrão $\sigma_{netbsd} = 7380255,24101$. O atrator reconstruído é apresentado na Figura 3.8.

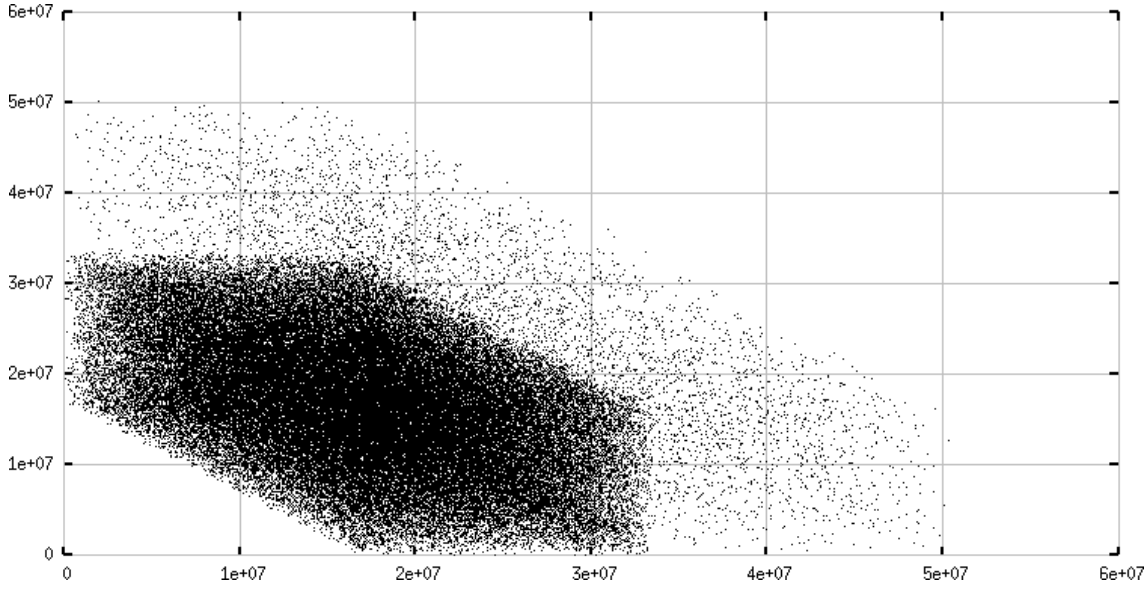

Figura 3.8: Atrator do NetBSD.

A Figura 3.8 sugere que a função aleatória segue uma distribuição de probabilidade condicional aproximadamente gaussiana, porém, com uma dispersão aproximadamente uniforme para valores entre 3.3×10^7 e 5×10^7.

3.1.4 OpenBSD

As amostras do gerador de números iniciais de seqüência do OpenBSD sugerem que sua implementação não adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída utilizando as próprias amostras de $G_{isn}(t)$, sendo os 100 primeiros valores apresentados na Figura 3.9.

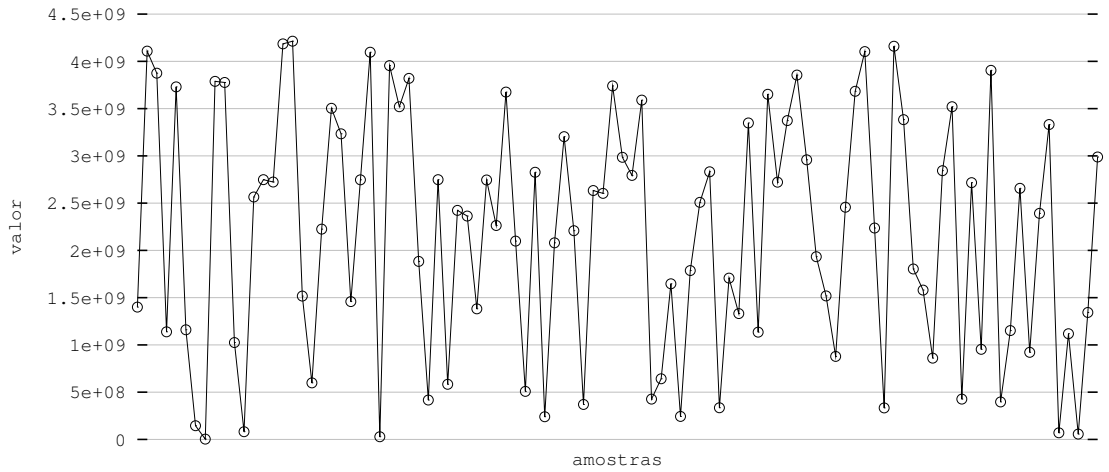

Figura 3.9: 100 amostras da função $\hat{R}(t)$ do OpenBSD.

A estimativa $\hat{R}(t)$ do OpenBSD possui média $\overline{\mu}_{openbsd} = 2144262074,92$ e desvio padrão $\sigma_{openbsd} = 1242845940,91$. O atrator reconstruído é apresentado na Figura 3.10.

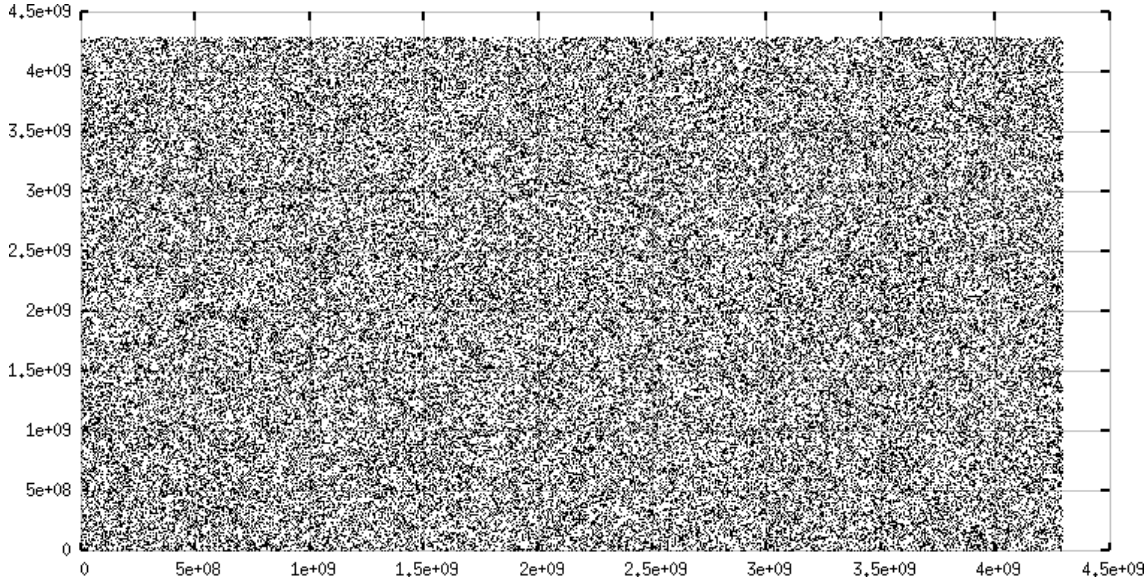

Figura 3.10: Atrator do OpenBSD.

A Figura 3.10 sugere que a função aleatória segue uma distribuição de probabilidade condicional uniforme, que caracteriza seu PRNG como próximo do ideal.

3.1.5 OpenSolaris

As amostras do gerador de números iniciais de seqüência do OpenSolaris sugerem que sua implementação adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída em acordo com a Equação 2.4. Os 100 primeiros valores desta estimativa são apresentados na Figura 3.11.

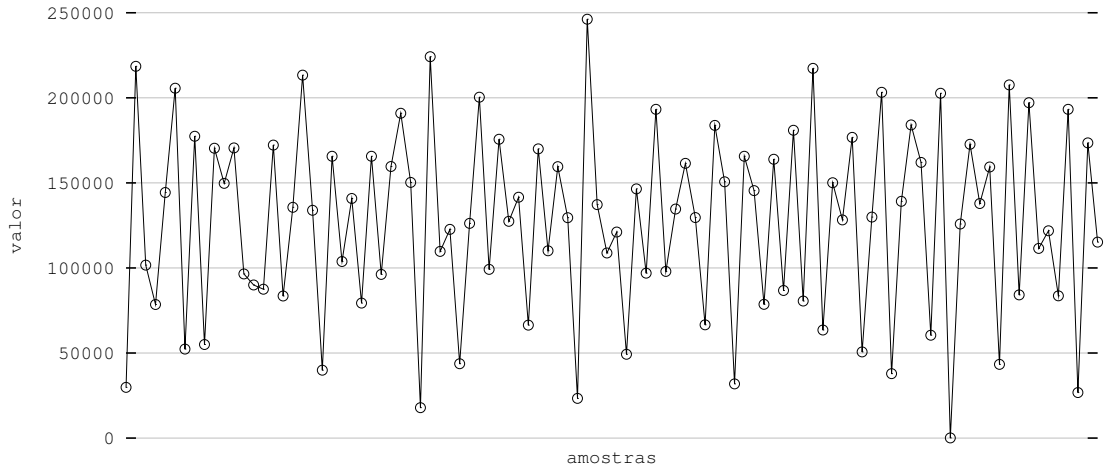

Figura 3.11: 100 amostras da função $\hat{R}(t)$ do OpenSolaris.

A estimativa $\hat{R}(t)$ do OpenSolaris possui média $\overline{\mu}_{opensolaris} = 127999,696517$ e desvio padrão $\sigma_{opensolaris} = 53423,5592826$. O atrator reconstruído é apresentado na Figura 3.12.

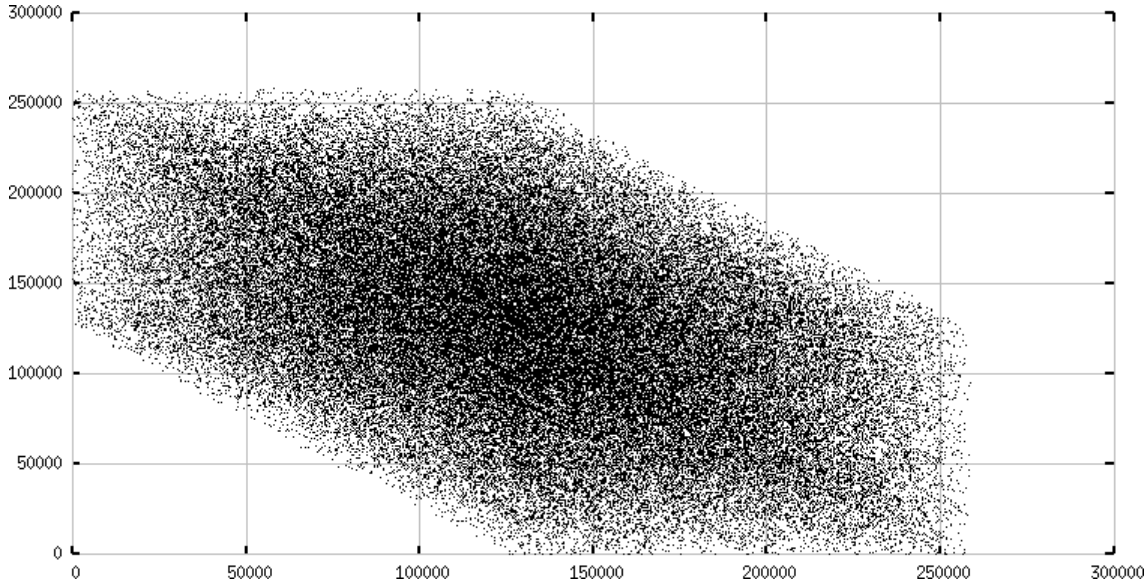

Figura 3.12: Atrator do OpenSolaris.

A Figura 3.12 sugere que a função aleatória segue uma distribuição de probabilidade condicional aproximadamente gaussiana.

3.1.6 Windows 2000

As amostras do gerador de números iniciais de seqüência do Windows 2000 sugerem que sua implementação adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída em acordo com a Equação 2.4. Os 100 primeiros valores desta estimativa são apresentados na Figura 3.13.

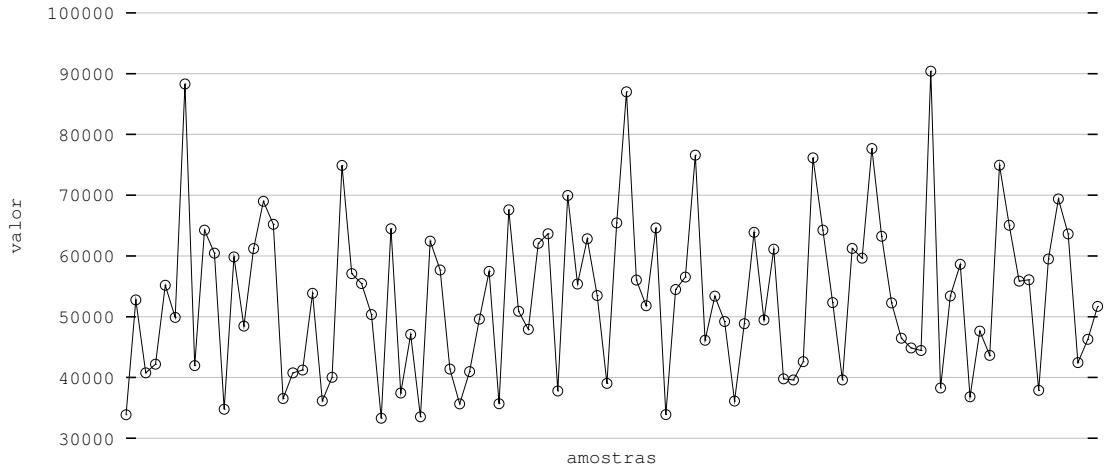

Figura 3.13: 100 amostras da função $\hat{R}(t)$ do Windows 2000.

A estimativa $\hat{R}(t)$ do Windows 2000 possui média $\overline{\mu}_{2000} = 52437,2461525$ e desvio padrão $\sigma_{2000} = 12668,7825762$. O atrator reconstruído é apresentado na Figura 3.14.

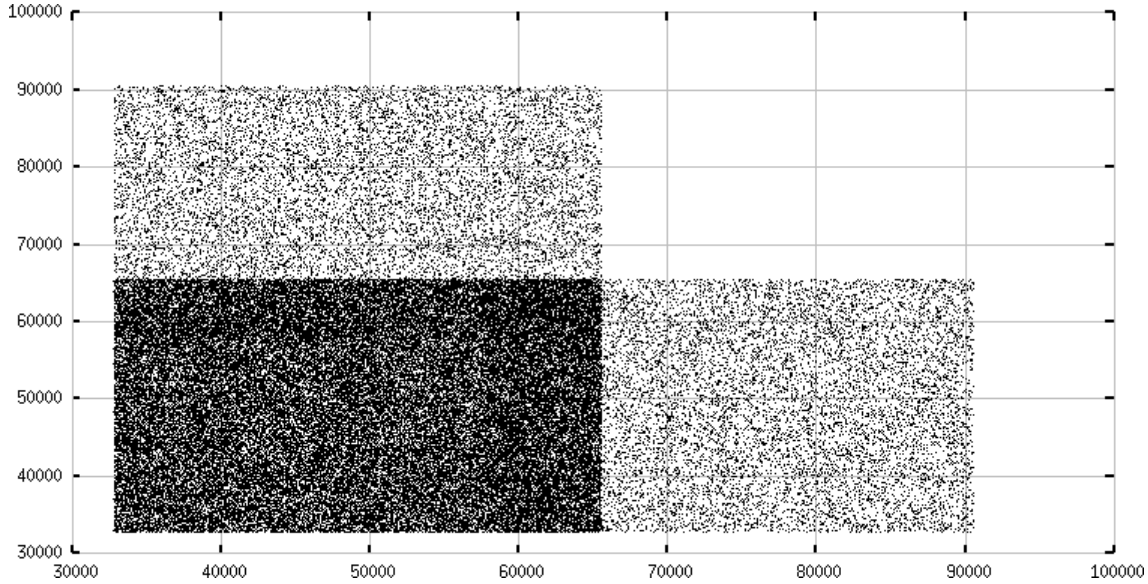

Figura 3.14: Atrator do Windows 2000.

A Figura 3.14 sugere que a função aleatória segue três distribuições de probabilidade condicional uniformes e sobrepostas. Sendo uma definida aproximadamente entre 33.000 e 66.000, e a outra aproximadamente entre 66.000 e 91.000.

3.1.7 Windows XP

As amostras do gerador de números iniciais de seqüência do Windows XP sugerem que sua implementação adota a recomendação descrita pela RFC 1948. Sendo assim, a estimativa $\hat{R}(t)$ da componente aleatória foi construída em acordo com a Equação 2.4. Os 100 primeiros valores desta estimativa são apresentados na Figura 3.15.

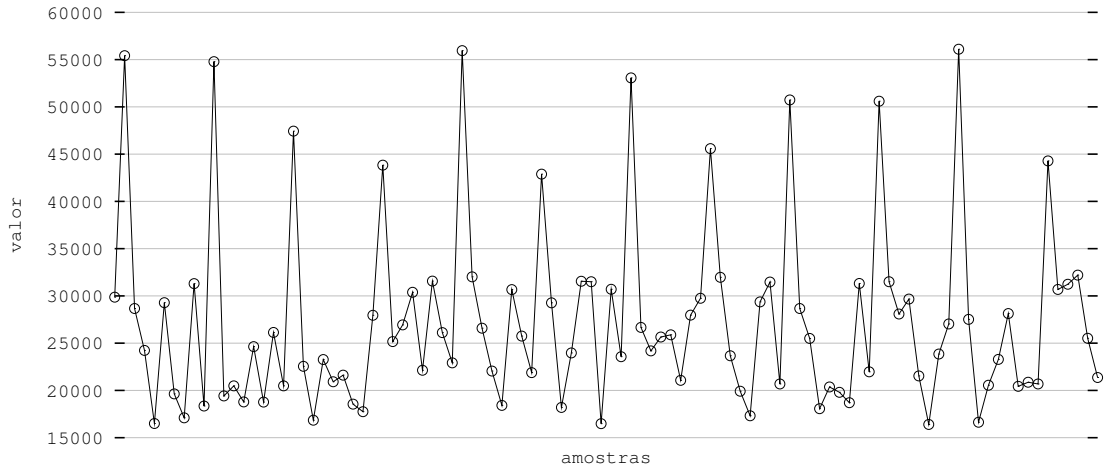

Figura 3.15: 100 amostras da função $\hat{R}(t)$ do Windows XP.

A estimativa $\hat{R}(t)$ do Windows XP possui média $\overline{\mu}_{xp} = 27650,4339943$ e desvio padrão $\sigma_{xp} = 9606,07395603$. O atrator reconstruído é apresentado na Figura 3.16.

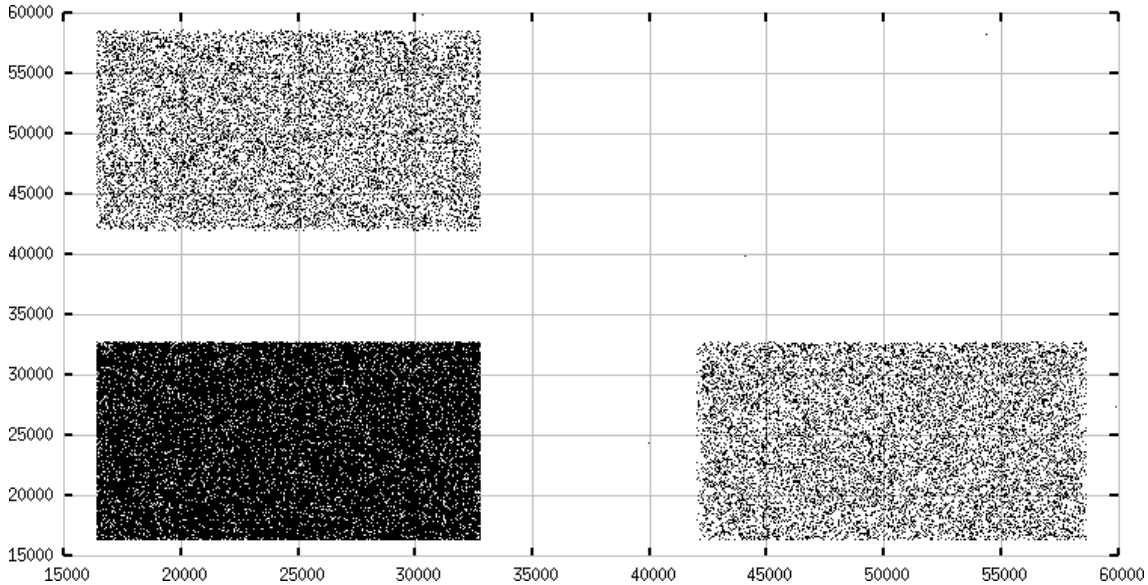

Figura 3.16: Atrator do Windows XP.

A Figura 3.16 sugere que a função aleatória segue três distribuições de probabilidade condicional, todas uniformes. Sendo uma definida aproximadamente entre 17000 e 33000, e a outra aproximadamente entre 43000 e 58000.

3.1.8 Honeyd

O PRNG do Honeyd possui uma característica singular. Como o objetivo de seu gerador de TCP ISNs é enganar o sistema de identificação do Nmap, ele é projetado para explorar a caracterização usada pelo Nmap para representar este campo do cabeçalho TCP. A caracterização dos TCP ISNs na criação da primeira geração do sistema de identificação do Nmap é baseada no incremento entre amostras consecutivas [Fyodor 1998]. Neste sentido, o Honeyd, de posse deste valor, para um dado sistema operacional, gera uma seqüência que produz o mesmo resultado. Posteriormente, o sistema de identificação do Nmap começou a utilizar o máximo divisor comum (MDC), ou *Greatest Common Divisor* (GCD), dentre os incrementos das amostras de TCP ISNs coletadas para caracterização deste. Logo, o Honeyd, de posse deste MDC, precisa apenas gerar incrementos múltiplos deste valor. Os dois processamentos realizados pelo Honeyd descritos anteriormente, podem ser verificados no Exemplo 3.1, nas linhas 22 e 24, respectivamente.

```
1  static uint32_t
2  get_next_isn(struct template *tmpl, const struct personality *person)
3  {
4      uint32_t prev_isn = tmpl->seq;
5      uint32_t num_isns = tmpl->seqcalls;
6      uint32_t gcd = person->gcd;
7      double amin = person->seqindex_amin;
8      double amax = person->seqindex_amax;
9      double a;
10
11     if ((amax - amin) >= 2.0) {
12         /* Select an 'a' halfway between acceptable limits.  The
13          * value of 'a' should always be the same, not random.
14          */
15         a = ((amax - amin) / 2.0) + amin;
16     } else {
17         /* To get a valid number in a small range */
18         a = (uint32_t)ceil(((amax - amin) / 2.0) + amin);
19     }
20     /* Constant ISN difference */
21     if ((amax - amin) < 1 || gcd > (5 / 2 * person->seqindex_aconst))
22         return (prev_isn + gcd);
23
24     return (prev_isn + ((((num_isns - 1) % 5) + 1) * (uint32_t)a - 1)*gcd);
25 }
```

Exemplo 3.1: Função do Honeyd responsável pela geração de TCP ISNs [Provos 2007].

No primeiro caso (linha 22 do Exemplo 3.1), a estimativa do PRNG será uma função constante. Incrementos constantes em geradores de TCP ISNs estão geralmente relacionados ao incremento do valor da função em intervalos bem definidos. Mesmo que a amostragem aqui utilizada seja igual ao período de incremento destes sistemas, ela não será a amostragem real devido aos tempos de transmissão e recepção de mensagens pela rede. Neste sentido, à medida em que o número de amostras aumenta, tende a 0 (zero) a probabilidade de que a função estimada $\hat{R}(t)$ seja constante para sistemas reais. Portanto, se o desvio padrão $\sigma_{\hat{R}(t)}$ for igual a 0 (zero), a máquina remota é uma imitação feita pelo Honeyd. No caso de TCP ISN constantes, este tipo de estratégia não funcionará, porém, apenas $\approx 3.8\%$ (38 de 989) dos sistemas presentes na base de dados do Honeyd usam esse tipo de geração de ISN. De forma geral, são raros os sistemas operacionais modernos que utilizam incrementos fixos ou valores constantes para geração dos TCP ISNs.

No segundo caso (linha 24 do Exemplo 3.1) o comportamento da função $\hat{R}(t)$ será semelhante ao que é apresentado na Figura 3.17. O período de 5 (cinco) amostras apresentado na Figura 3.17 está relacionado à operação de módulo presente na linha 24 do Exemplo 3.1. Este determinismo do gerador de TCP ISNs do Honeyd pode ser explorado de modo a estimar se um conjunto de amostras de ISNs foram gerados por esta ferramenta.

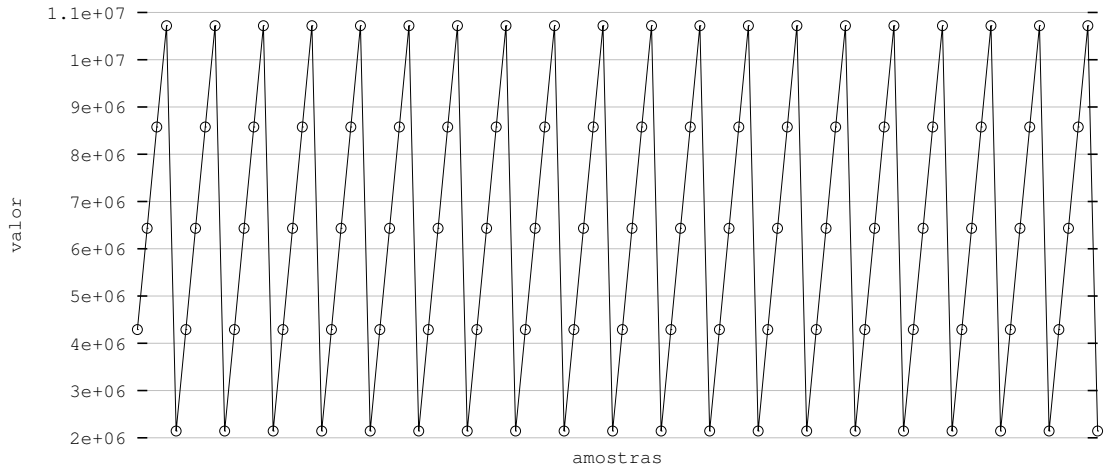

Figura 3.17: 100 amostras da função $\hat{R}(t)$ do Honeyd.

A princípio pode-se levar em consideração a criação de um modelo que considere este determinismo para detectar o Honeyd. Porém, a função de geração de número de seqüência do Honeyd não é exclusiva para uma única conexão. Se dois usuários se comunicam simultaneamente com uma mesma máquina do Honeyd, a função $\hat{R}(t)$ estimada por um deles será corrompida pela ação do outro. Na Figura 3.18 é ilustrado este caso.

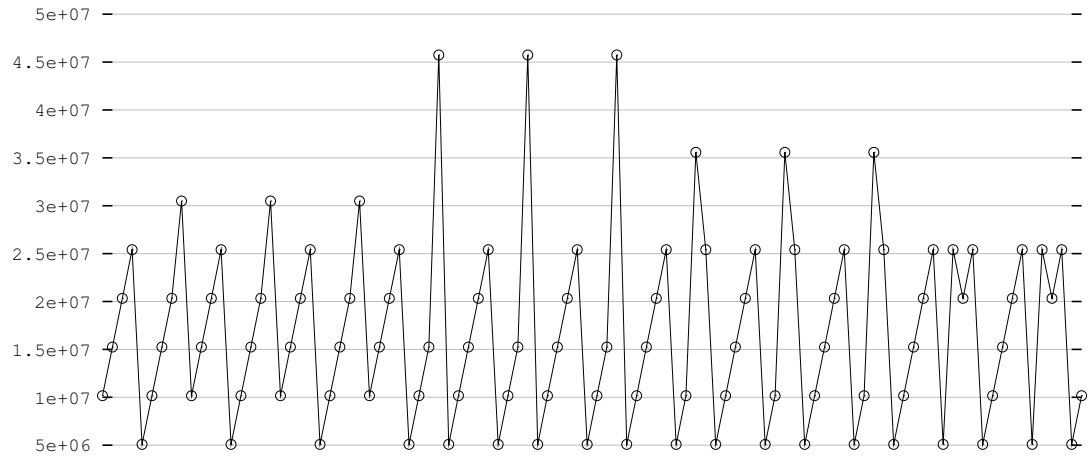

Figura 3.18: 100 amostras ruidosas da função $\hat{R}(t)$ do Honeyd.

Por este motivo, assim como acontecem ruídos relacionados a estimativa de $\hat{R}(t)$ em sistemas operacionais reais, a estimativa $\hat{R}(t)$ do Honeyd também pode ser ruidosa. O atrator do Honeyd referente a Figura 3.17 é apresentado na Figura 3.19.

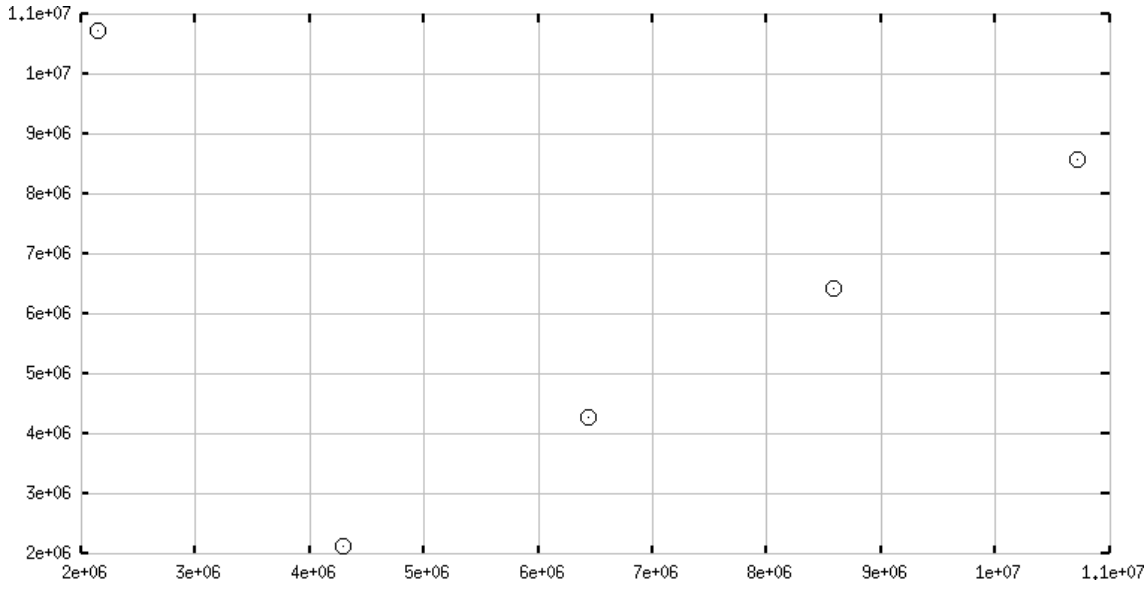

Figura 3.19: Atrator do Honeyd (pontos destacados usando $\odot$).

O atrator referente as amostras da Figura 3.18 é apresentado na Figura 3.20.

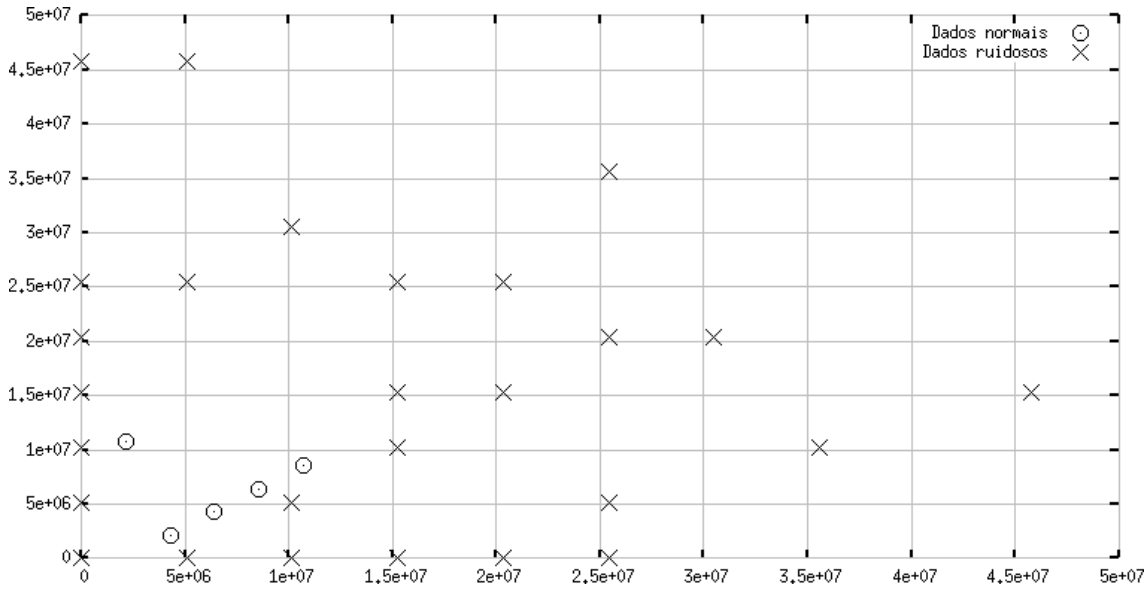

Figura 3.20: Atrator do Honeyd (pontos normais em $\odot$ e ruidosos em $\times$).

Esta interferência, no caso do Honeyd, pôde ser verificada de forma clara porque seu sistema de geração de TCP ISNs é determinístico. Nos outros casos, utilizando amostras de sistemas operacionais reais, esta interferência (ou ruído) está associada a não constância da função $F(\cdot)$ da Equação 2.1. Seja qual for a origem desta interferência, é desejável que a técnica de caracterização minimize seu efeito na avaliação do dado analisado.

3.2 Detecção do SYN *proxy* do PF

SYN *proxies* fazem a intermediação do processo de sincronização via TCP *three-way hand shake*, por este motivo o ISN enviado a máquina de origem não é o da máquina real de destino, mas sim, o do sistema que realiza a intermediação. O sistema de intermediação de sincronização do PF gera TCP ISNs de uma forma peculiar. Na Figura 3.21 é apresentada a função $\hat{R}(t)$ do SYN *proxy* do PF.

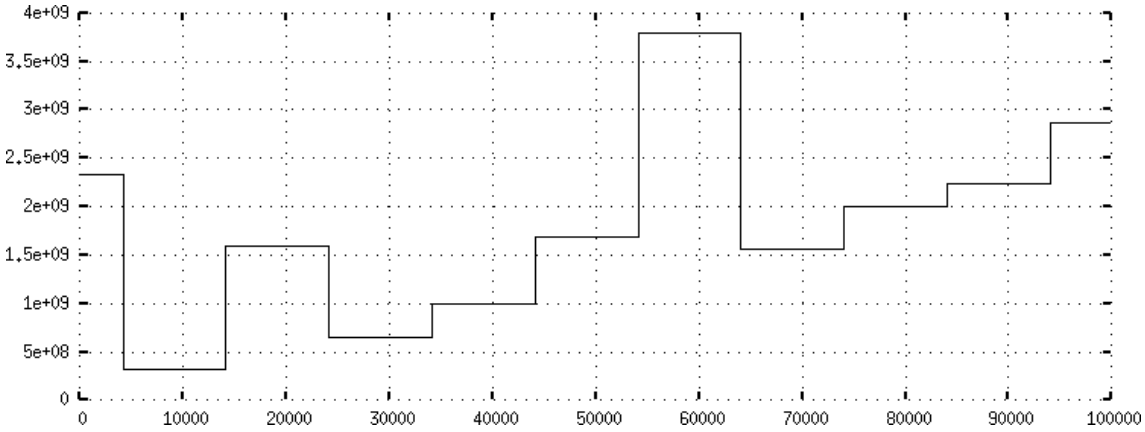

Figura 3.21: 100.000 (cem mil) amostras da função $\hat{R}(t)$ do SYN *proxy* do PF.

O período de mudança do ISN é de aproximadamente $100s$ (10.000×10^{-2}). Analisar este sistema por meio de atratores requereria um quantidade inviável de amostras. Outra forma de detecção é partir da análise do comportamento constante de $\hat{R}(t)$, porém, isto requereria uma quantidade de amostras da superior a 10.000, pois, existem SOs que possuem ISN fixo [Fyodor 1998] e que poderiam ser confundidos com SYN *proxies*. Uma forma mais eficiente de realizar esta tarefa é utilizar o fato de que sistemas operacionais que geram ISNs fixos não seguem a RFC 1948. Desta forma, para estes sistemas, duas requisições TCP SYN com a porta de origem diferente retornarão o mesmo TCP ISN, enquanto que no SYN *proxy* do PF estes ISNs serão diferentes. Para confirmar a classificação do SYN *proxy* deve-se enviar um terceiro pacote com a porta de origem igual a umas das anteriores. No Exemplo 3.2 são ilustradas as mensagens de resposta de um SYN *proxy* para este procedimento.

```
1 [18:57:18]                [18:57:20]                [18:57:22]
2 + IPv4                    + IPv4                    + IPv4
3 | (src 192.0.2.1)        | (src 192.0.2.1)        | (src 192.0.2.1)
4 |_(dst 192.0.2.254)      |_(dst 192.0.2.254)      |_(dst 192.0.2.254)
5 + TCP                     + TCP                     + TCP
6 | (srcport 80)           | (srcport 80)           | (srcport 80)
7 | (dstport 23)           | (dstport 48252)        | (dstport 23)
8 |_(seq 385720742)        |_(seq 1118434370)       |_(seq 385720742)
```

Exemplo 3.2: Mensagens de resposta de um SYN *proxy* para portas de origem distintas.

Note que o intervalo de envio das mensagens deve ser maior que $0,42s$ (ver Subseção 3.1.2) para que este procedimento não classifique um FreeBSD como SYN *proxy* (verifica-se que na linha 1 do Exemplo 3.2 as respostas foram recebidas em um intervalo de aproximadamente 2 segundos).

Após esta descrição de como detectar SYN *proxies* utilizando um método heurístico acerca da geração do ISN, será retomado o problema de caracterização dos atratores. Neste trabalho, a rede SOM foi utilizada por causa das seguintes propriedades presentes na rede após seu treinamento: aproximação do espaço de entrada, ordenação topológica, resolução baseada em densidade, e extração de características.

3.3 Mapas auto-organizáveis

Um mapa auto-organizável, também conhecido como *Self-Organizing Map* (SOM), é o produto de uma rede neural proposta por Teuvo Kohonen em 1982 [Kohonen 1982]. Ele é capaz de mapear um padrão de entrada de dimensão arbitrária em uma mapa de características (*feature map*) que expressam a ordem topológica do espaço de entrada [Kohonen 2001]. Na Figura 3.22, é ilustrada a estrutura da rede neural SOM.

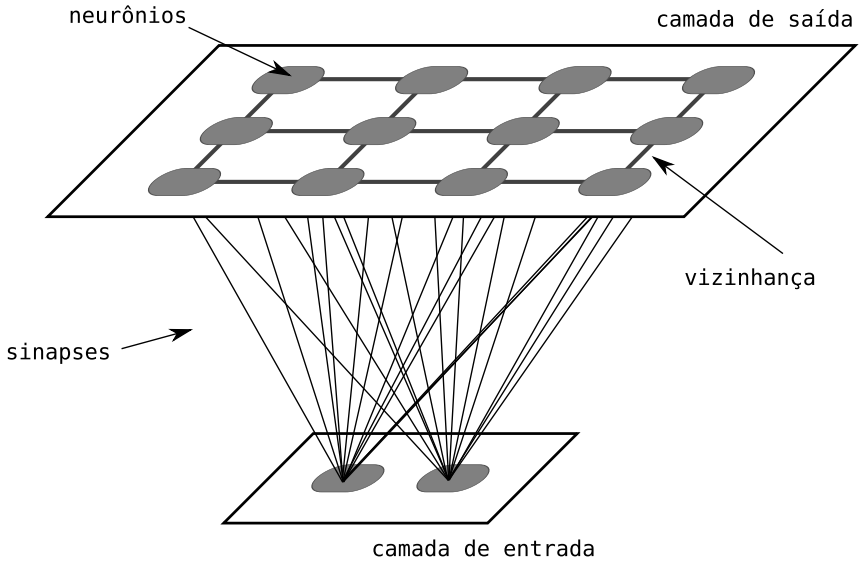

Figura 3.22: Estrutura da rede neural SOM 4×3 para uma entrada bidimensional.

A estrutura da rede neural SOM é composta por um vetor, ou camada, de entrada *n*-dimensional e uma camada de saída composta por neurônios cujo número de sinapses é da mesma dimensão do vetor de entrada. A entrada da rede é descrita por um vetor $\mathbf{x}$ de dimensão m (na Figura 3.22 tem-se $m = 2$), como descrito a seguir:

$$\mathbf{x} = [x_1, x_2, \ldots, x_m]^T \tag{3.4}$$

onde $\mathbf{x}^T$ indica a operação de transposição sobre o vetor $\mathbf{x}$. Para uma camada de saída que possui l neurônios (na Figura 3.22 tem-se uma camada de saída 4×3, totalizando $l = 12$), todo neurônio j possui um vetor de sinapses $\mathbf{w}_j$, onde cada sinapse está associada a uma posição do vetor de entrada $\mathbf{x}$:

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T \qquad j = 1, 2, \ldots, l. \tag{3.5}$$

Cada iteração do algoritmo clássico de treinamento da rede SOM é composta por três processos, executados nesta ordem: competitivo, cooperativo e adaptativo.

3.3.1 Processo competitivo

No processo competitivo, para uma dada entrada $\mathbf{x}$, cada neurônio $\mathbf{w}_j$ compete com os outros para representar a informação apresentada a rede de acordo com a seguinte equação [Haykin 1999]:

$$i(\mathbf{x}) = \arg\min_j \|\mathbf{x} - \mathbf{w}_j\|_\lambda \qquad j = 1, 2, \ldots, l. \tag{3.6}$$

onde, o neurônio vencedor será aquele cujo vetor de pesos sinápticos minimiza a métrica λ em relação a $\mathbf{x}$. A princípio não há restrição quanto à métrica adotada para desempenhar este processo. Porém, a mais utilizada é a distância Euclidiana [Kohonen 2001].

3.3.2 Processo cooperativo

No processo cooperativo, o neurônio vencedor compartilha com os vizinhos topológicos a capacidade de representar a atual entrada. Na rede SOM, isto é realizado utilizando uma função Gaussiana centrada no neurônio vencedor. A Equação 3.7 descreve a função utilizada para ponderar o processo de cooperação [Haykin 1999]:

$$h_{j,i(\mathbf{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right) \tag{3.7}$$

onde o numerador da exponencial na Equação 3.7 faz com que quanto maior a distância

$$d_{j,i}^2 = \|\mathbf{w}_j - \mathbf{w}_i\|^2 \tag{3.8}$$

entre um dado neurônio j e o neurônio vencedor i, menor seja a influência da entrada para o neurônio j. O denominador da exponencial da função $h_{j,i(\mathbf{x})}(n)$ depende da iteração n. A função $\sigma(n)$, apresentada na Equação 3.9 tem como objetivo diminuir a cooperação entre os neurônios a medida que o treinamento evolui:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) \tag{3.9}$$

ou seja, quanto maior o número da iteração menor será a cooperação entre os neurônios.

3.3.3 Processo adaptativo

É no processo adaptativo que cada neurônio têm os pesos ajustados de acordo com a ponderação realizada no processo cooperativo. O processo de aprendizado é baseado na hipótese Hebianna [Hebb 1949] e em um fator denominado termo de esquecimento (*forgetting term*) [Haykin 1999]. A função de aprendizado é descrita pela Equação 3.10:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)[\mathbf{x} - \mathbf{w}_j(n)] \tag{3.10}$$

onde $[\eta(n)h_{j,i(\mathbf{x})}(n)\mathbf{x}]$ é o termo de aprendizado Hebianno, $[-\eta(n)h_{j,i(\mathbf{x})}(n)\mathbf{w}_j(n)]$ o termo de esquecimento. Tal característica se deve ao fato de que, no caso onde $\mathbf{w}_j(n)$ é o neurônio vencedor, os valores de $\mathbf{x}$ tendem a ser incorporados à $\mathbf{w}_j(n+1)$, já que seu valor atual tende a ser cancelado pelo termo de esquecimento. Por último define-se $\eta(n)$ como sendo a função taxa de aprendizado, definida como:

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right). \tag{3.11}$$

3.3.4 Algoritmo e considerações

O Algoritmo 3.1 descreve o treinamento da rede SOM, onde N representa o tamanho do conjunto de vetores de entrada, e l o número de neurônios na camada de saída.

requer: $\{\mathbf{x}_i\}_{i=1}^{N}$ {conjunto de vetores de entrada}
1: $\{\mathbf{w}_j(0)\}_{j=1}^{l}$ {iniciar pesos de cada neurônio}
2: **repita**
3: $\mathbf{x}(n) \leftarrow \mathbf{x}_{random()}$ {escolhe aleatoriamente uma amostra do conjunto de entrada}
4: $i(\mathbf{x}) \leftarrow \arg\min_j \|\mathbf{x}(n) - \mathbf{w}_j\|$ {calcula neurônio vencedor (distância Euclidiana)}
5: $\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n)h_{j,i(\mathbf{x})}(n)[\mathbf{x}(n) - \mathbf{w}_j(n)]$ {ajusta os pesos}
6: **enquanto** o critério de parada não seja satisfeito
retorna: $\{\mathbf{w}_j\}_{j=1}^{l}$ {mapa de características}

Algoritmo 3.1: Algoritmo de treinamento da rede SOM.

Inicialmente os pesos da rede podem ser iniciados aleatoriamente ou instanciados a partir de amostras do conjunto de entrada. A primeira estratégia é a mais utilizada, porém, a segunda pode acelerar o processo de aprendizado. A cada iteração do algoritmo de aprendizado é aconselhado o embaralhamento do conjunto de entrada para que sua ordem não influencie no treinamento.

3.3.5 Utilização

Nas Equações 3.12, 3.13 e 3.14 são apresentados os valores das constantes utilizadas para o treinamento da rede. Estes valores são recomendações da literatura [Haykin 1999] e foram utilizados com sucesso nos experimentos realizados, são eles:

$$\eta_0 = 0,1 \tag{3.12}$$

$$\tau_2 = 1.000 \tag{3.13}$$

$$\tau_1 = \frac{\tau_2}{\log \sigma_0}. \tag{3.14}$$

Foi atribuído a σ_0 o valor do maior raio do mapa topológico, isto é, para uma rede $n \times m$, tem-se $\sigma_0 = 0,5\max(n,m)$. Para o caso da Figura 3.22 (onde a topologia é 4×3)

$\sigma_0 = 2$. No treinamento para caracterização de cada atrator foi utilizada uma camada de saída 30×10 interligados em uma malha regular (como ilustrado na camada de saída da Figura 3.22), e o critério de parada utilizado foi o de número de iterações (no caso, 1500). Os mapas resultantes do treinamento mostrando as posições dos vetores de pesos sinápticos para cada sistema operacional e para o Honeyd são apresentados na Figura 3.23.

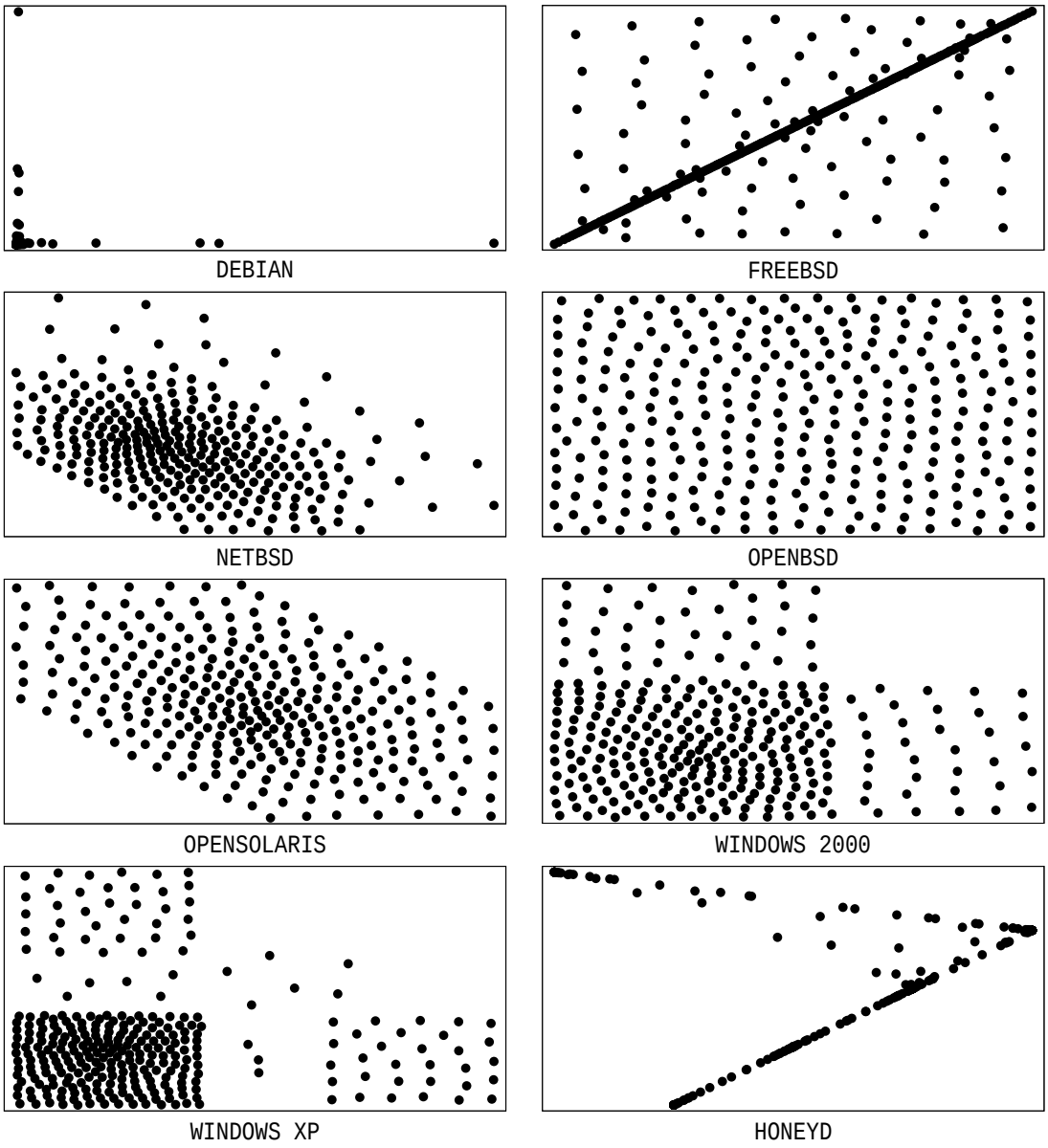

Figura 3.23: Resultado do treinamento da rede SOM para cada atrator.

3.3.6 Pós-processamento

A fim de caracterizar os atratores, buscou-se suprimir o efeito de ruídos, possivelmente inseridos na captura dos dados, e reforçar a noção de densidade em cada neurônio. Foi

utilizado um pós-processamento que adiciona um fator de densidade $\phi(\mathbf{p})$ para cada ponto $\mathbf{p}$ do mapa de características, como descrito na Equação 3.15:

$$\phi(\mathbf{p}) = \sum_{i=1}^{N} \exp\left(-\frac{\|\mathbf{p}-\mathbf{x}_i\|^2}{2\sigma_d^2}\right) \tag{3.15}$$

onde $\mathbf{x}_i$ representa o i-ésimo ($i = 1,\ldots,N$) ponto do atrator. Os resultados gerados para $\sigma_d = 0,07$ (escolhido de forma empírica) são apresentados na Figura 3.24.

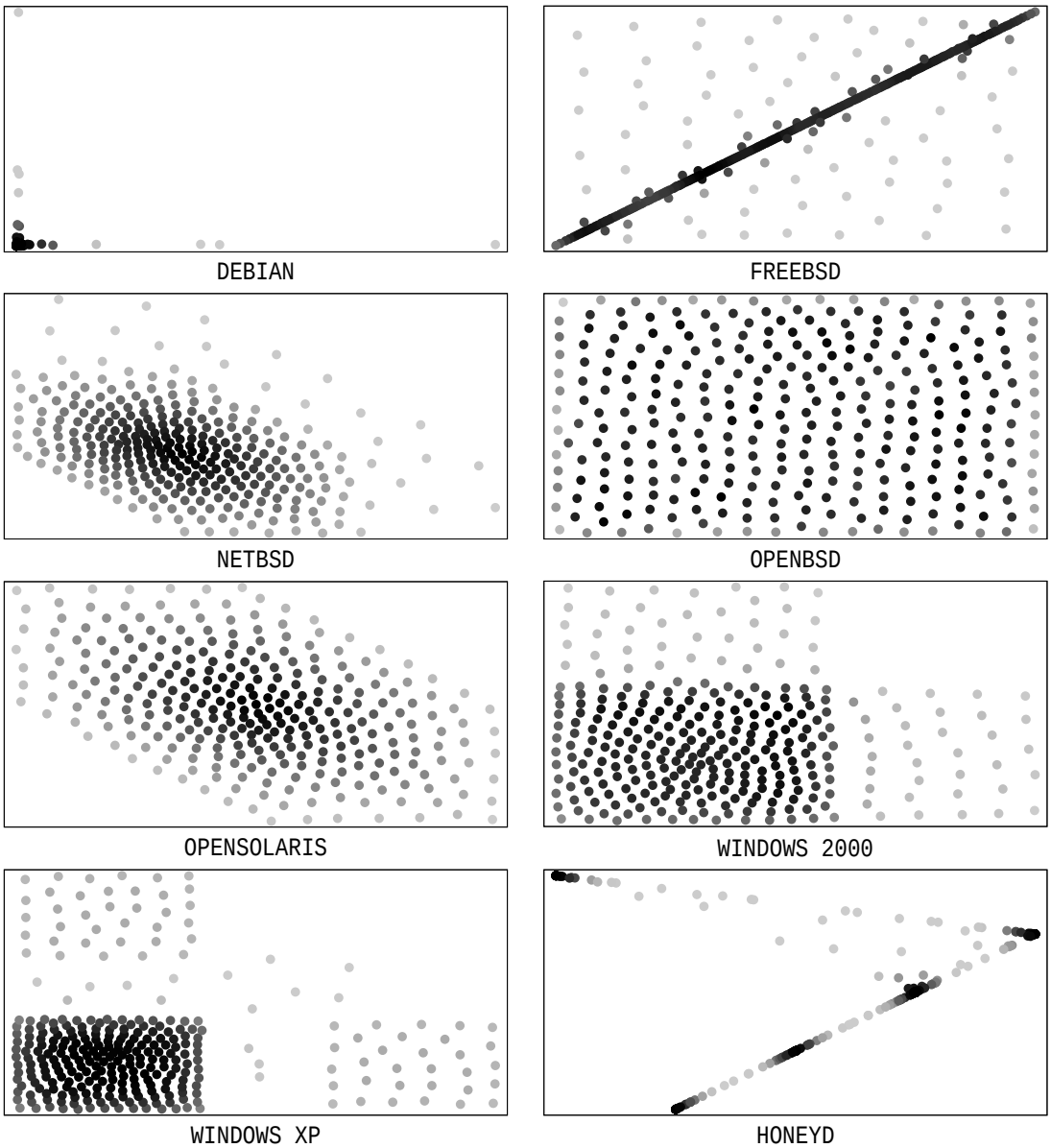

Figura 3.24: Pós-procedimento da rede SOM em relação à densidade do atrator.

Outro pós-processamento realizado tem como objetivo descrever o fluxo dos pontos de cada atrator. Esta tarefa é realizada analisando a direção que cada ponto da entrada

próximo a um ponto no mapa segue, como descrito na Equação 3.16:

$$\theta(\mathbf{p}) = \tan^{-1}\left[\sum_{i=1}^{N-1} \exp\left(-\frac{\|\mathbf{p}-\mathbf{x}_i\|^2}{2\sigma_f^2}\right)(\mathbf{x}_{i+1}-\mathbf{x}_i)\right] \tag{3.16}$$

onde $\mathbf{x}_i$ representa o i-ésimo ($i = 1,\ldots,N-1$) ponto do atrator. Os resultados gerados para $\sigma_f = 0,5$ (escolhido de forma empírica) são apresentados na Figura 3.25.

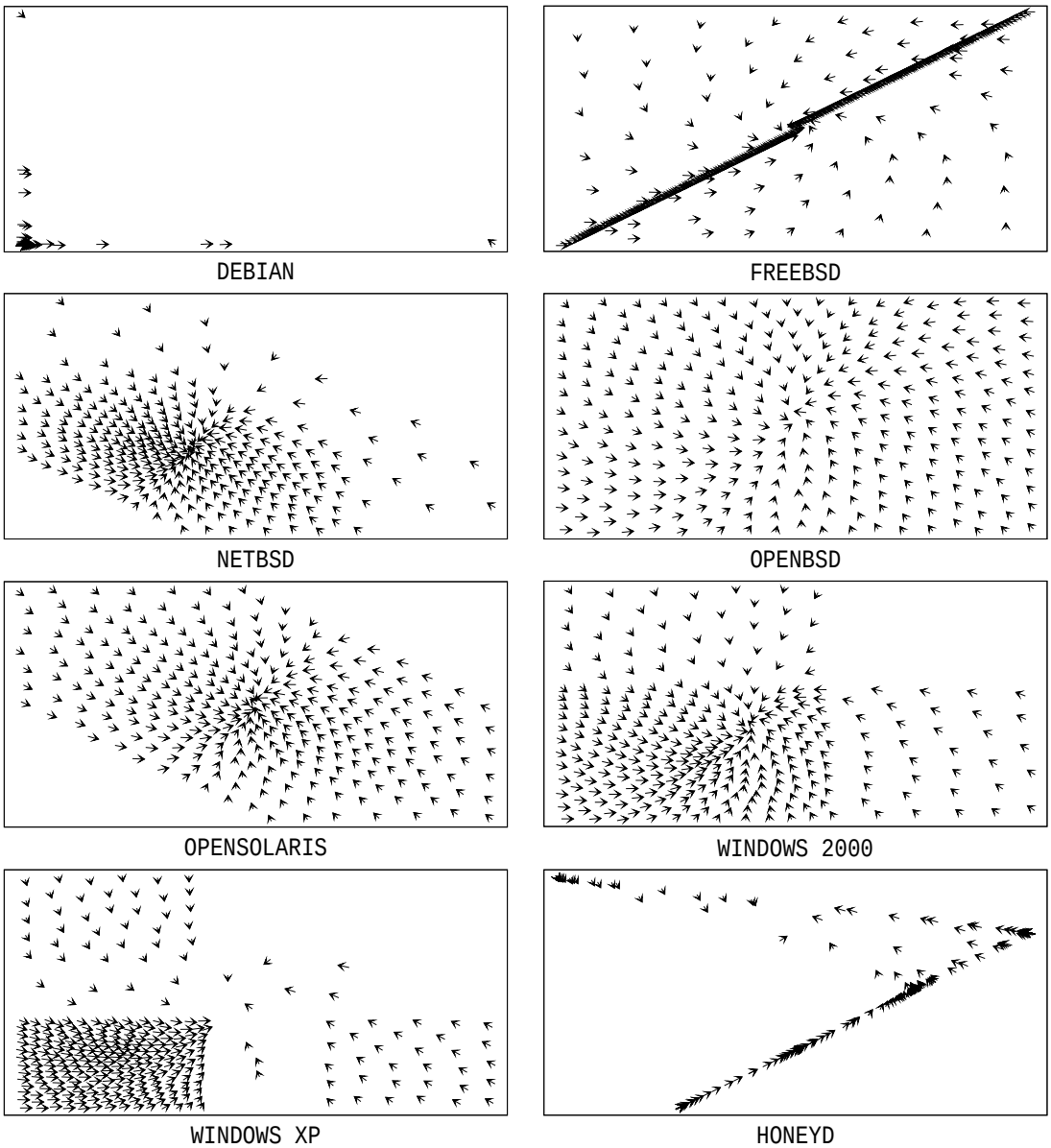

Figura 3.25: Pós-procedimento da rede SOM em relação ao fluxo do atrator.

Desta forma, cada atrator é descrito por uma tripla $\langle \mathbf{P},\Phi,\Theta \rangle$ onde $\mathbf{P}$ é uma matriz que simboliza o conjunto de pontos obtidos a partir do treinamento da rede SOM, Φ a densidade estimada $\phi(\mathbf{p})$ em cada ponto $\mathbf{p} \in \mathbf{P}$, e Θ a orientação estimada $\theta(\mathbf{p})$ em cada ponto

$\mathbf{p} \in \mathbf{P}$. Dada esta descrição dos atratores de cada sistema operacional, será apresentado a seguir um método para classificar estas descrições. A distância de Hausdorff é conhecida pela sua capacidade de classificar formas (conjuntos de pontos). Neste trabalho uma adaptação da distância de Hausdorff será utilizada para classificar as representações dos atratores dos sistemas operacionais.

3.4 Distância de Hausdorff

A distância de Hausdorff é, informalmente, definida como a maior distância mínima entre dois conjuntos de pontos. Esta descrição é definida formalmente para um par de conjuntos X e Y pela Equação 3.17 [Deza & Deza 2006]:

$$H(X,Y) = \sup_{x \in X} \inf_{y \in Y} d(x,y) \tag{3.17}$$

onde "inf" indica a operação ínfimo e "sup" a operação supremo. Na Figura 3.26, as distâncias $H(X,Y)$ e $H(Y,X)$ para dois conjuntos de pontos X e Y são apresentadas.

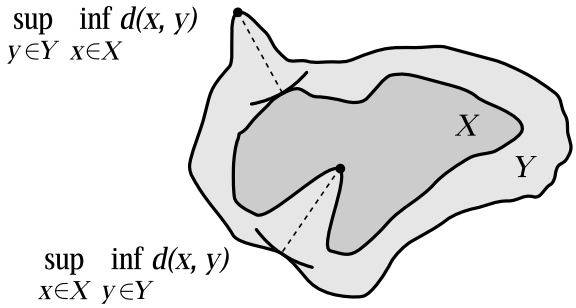

Figura 3.26: Ilustração das distâncias de Hausdorff entre dois conjuntos de pontos.

Através desse exemplo pode-se notar que as duas distâncias podem assumir valores diferentes. Por este motivo diz-se que a distância de Hausdorff, como apresentada na Equação 3.17, não é simétrica. Para contornar a não simetria intrínseca da Equação 3.17 é adotada a distância simétrica de Hausdorff, apresentada pela Equação 3.18:

$$H_s(X,Y) = \max\{H(X,Y), H(Y,X)\} \tag{3.18}$$

onde "max" indica a operação máximo. Esta adaptação consiste basicamente em calcular o valor da Equação 3.17 nos dois sentidos e optar pelo maior deles como resultado.

3.4.1 Extensões

É importante lembrar que será utilizada uma estimativa da função $R(t)$ para gerar os atratores de cada sistema operacional. Esta estimativa está sujeita a ruídos relacionados ao atraso na transmissão de pacotes e a mudança no valor da chave secreta da função $F(\cdot)$. Como a distância de Hausdorff é baseada em supremos e ínfimos, não é difícil qualquer

ruído associado à criação do atrator invalidar o resultado. Para contornar este problema, deve-se adotar um método que priorize a estatística do atrator de forma a minimizar a influência desses possíveis ruídos. Uma solução para este problema seria utilizar, como fator de distância, o número de máximas distâncias mínimas que fossem maior que um certo valor β, como descrito na Equação 3.19:

$$M(X,Y,\beta) = \left| \{x \in X : \inf_{y \in Y} d(x,y) \geq \beta\} \right| \tag{3.19}$$

onde $|\cdot|$ indica a cardinalidade do conjunto, X e Y são duas representações de atratores e $d(x,y)$ representa a distância Euclidiana. O modelo geométrico usado para representar atratores possui duas informações adicionais em relação ao que é contemplado pela distância de Hausdorff. Este modelo, inclui informações relacionadas à densidade e à orientação de cada no mapa gerado pela rede neural auto-organizável. Para efeitos de utilização desta métrica, a densidade pode ser considerada a terceira coordenada do ponto $\mathbf{p} \in \mathbf{P}$. Sendo assim, a distância $d(x,y)$ é definida como:

$$d(x,y) = \| \langle \mathbf{p}_x, \phi_x \rangle - \langle \mathbf{p}_y, \phi_y \rangle \| \tag{3.20}$$

onde $\langle \mathbf{p}_x, \phi_x \rangle$ representa um vetor tridimensional formado pelas coordenadas do ponto $\mathbf{p}_x$ e pela densidade ϕ_x. Uma segunda métrica foi criada para utilização do fator de orientação presente na representação dos atratores:

$$N(X,Y,\alpha) = \left| \{x \in X : [x \cdot \arg\inf_{y \in Y} d(x,y)] \leq \alpha\} \right|. \tag{3.21}$$

Para considerar a orientação é necessário incorporar alguma métrica capaz de verificar a semelhança entre duas orientações. Neste sentido o produto interno foi utilizado para desempenhar tal tarefa:

$$x \cdot y = \cos\theta_x \cos\theta_y + \sin\theta_x \sin\theta_y. \tag{3.22}$$

A Equação 3.21 difere da Equação 3.19 na utilização do produto interno como filtro de contagem para a métrica $N(X,Y,\alpha)$. Como as métricas M e N também não são simétricas, define-se as Equações 3.24 e 3.23:

$$M_s(X,Y,\beta) = \max\{H(X,Y,\beta), H(Y,X,\beta)\} \tag{3.23}$$

$$N_s(X,Y,\alpha) = \max\{N(X,Y,\alpha), N(Y,X,\alpha)\}. \tag{3.24}$$

Serão estas as duas métricas utilizadas para classificar as representações dos atratores apresentadas neste capítulo. No próximo capítulo serão apresentados os resultados de classificação para cada uma das métricas.

Capítulo 4

Implementação e resultados

Neste capítulo serão apresentados os resultados e os procedimentos necessários para classificação dos atratores de cada um dos sistemas operacionais. Primeiramente, os procedimentos de caracterização e identificação relativos a Figura 1.1 são descritos, e em seguida os resultados obtidos utilizando estes procedimentos.

4.1 Caracterização

A transformação de amostras de TCP ISNs em uma estimativa de $\hat{R}(t)$ e a criação e representação de atratores compõem o subprocesso de caracterização do método de identificação proposto. O subprocesso de caracterização, definido com base na fundamentação apresentada no Capítulo 3, é apresentada de forma contextualizada na Figura 4.1.

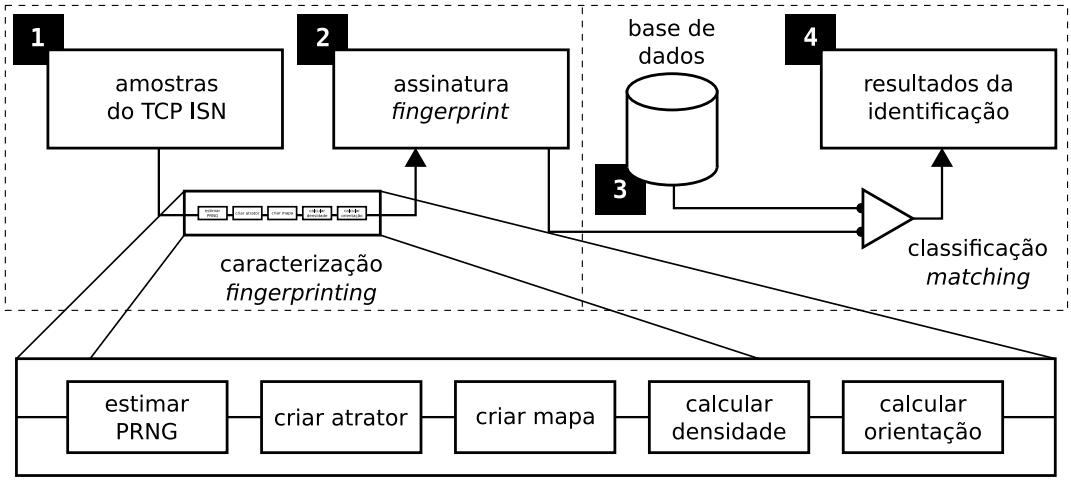

Figura 4.1: Contextualização do subprocesso de caracterização.

O procedimento de caracterização proposto é descrito no de forma mais detalhada no Algoritmo 4.1. Como pode ser verificado, o algoritmo de caracterização consiste em obter amostras do PRNG da máquina remota (linhas de 1 à 7), criar o atrator a partir

da função estimada $\hat{R}(t)$ (linha 8) e criar uma caracterização utilizando a rede SOM e o pós-processamento (linhas de 9 à 12).

requer: r {endereço da máquina remota}
requer: c {controlador de acesso a rede}
requer: s {rede neural auto-organizável}
requer: l {lista vazia para armazenar as amostras}
1: **para** i de 1 até n **faça**
2: $c.enviar(r, \texttt{TCP\_SYN})$
3: $r \leftarrow c.recebe(r, \texttt{TCP\_SYN\_ACK})$
4: $c.enviar(r, \texttt{TCP\_RST})$
5: $l.adicionar(i, r.isn)$
6: esperar(τ) {espera τ segundos antes de retomar}
7: **fim para**
8: $a \leftarrow atrator(estimar(l))$
9: $s.treinar(a, \texttt{DIMENSÃO}, \texttt{NÚMERO\_DE\_ITERAÇÕES})$
10: $\mathbf{P} \leftarrow s.mapa$
11: $\Phi \leftarrow densidade(s.mapa, \sigma_d)$
12: $\Theta \leftarrow fluxo(s.mapa, \sigma_f)$
retorna: $\langle \mathbf{P}, \Phi, \Theta \rangle$ {representação do atrator para r}

Algoritmo 4.1: Procedimento de caracterização proposto.

A quantidade de amostras, n, o tempo de amostragem, τ, a dimensão da rede auto-organizável e o número de iterações no treinamento da rede foram determinados de forma empírica. Utilizou-se 100.000 (cem mil) amostras de ISNs de cada sistema operacional para construir as caracterizações que compõem a base de dados de assinaturas. Mais 10.000 (dez mil) amostras foram capturadas para realizar os testes de classificação. O tempo de amostragem utilizado foi $\tau = 10^{-2}$ segundos, e a dimensão da rede utilizada foi de 30×30 neurônios. O número de interações utilizado para o algoritmo SOM foi de 1500. Foi constado que, após esta iteração, as mudanças no mapa de características praticamente inexistem. O valor de σ_d e σ_f utilizados nas linhas 12 e 13 do Algoritmo 4.1, respectivamente, foram mantidos em relação aos valores utilizados no Capítulo 3, ou seja, $\sigma_d = 0,07$ e $\sigma_f = 0,5$.

4.2 Classificação

O subprocesso de classificação do método de identificação proposto consiste na utilização das métricas $M_s(X, Y, \beta)$ e $N_s(X, Y, \alpha)$, desenvolvidas no Capítulo 3. Este subprocesso é ilustrado de forma contextualizada na Figura 4.2. Este procedimento retorna o nome do sistema operacional e a distância que possui para a representação mais próxima na base de dados de assinaturas. Base de dados esta formada por representações de atratores construídas a partir de 100.000 amostras de TCP ISNs de cada sistema operacional e do Honeyd.

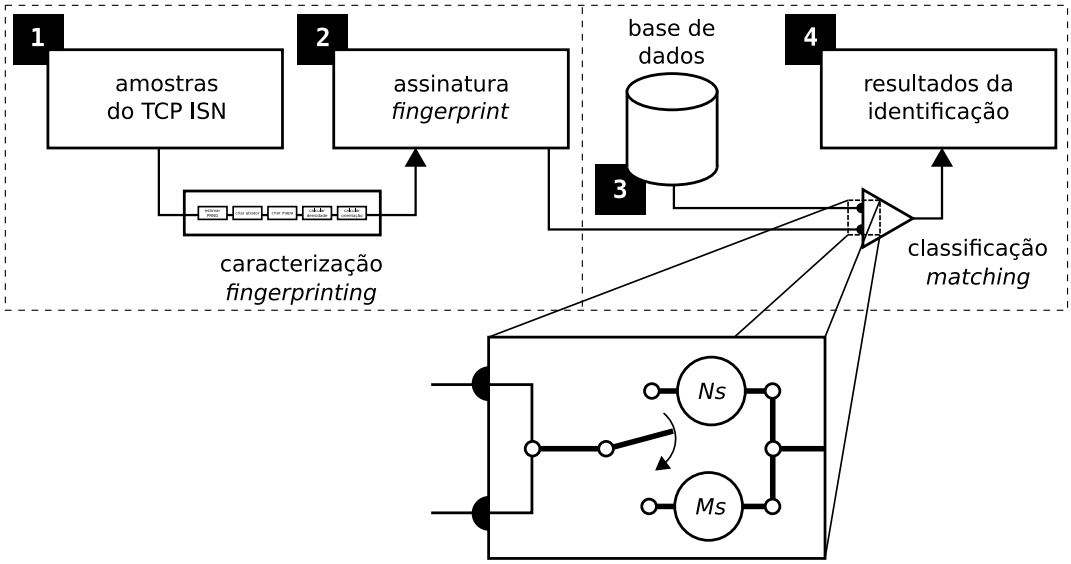

Figura 4.2: Contextualização do subprocesso de classificação.

O procedimento de classificação proposto é descrito no Algoritmo 4.2. Para finalizar o processo de classificação, basta instanciar qual das métricas descritas no Capítulo 3 será utilizada. Deve-se então decidir por uma das das duas métricas, $M_s(X,Y,\beta)$ descrita pela Equação 3.23 ou $N_s(X,Y,\alpha)$ descrita pela Equação 3.24.

requer: m {assinatura da máquina remota}
requer: b {base de dados de assinaturas}
1: $d_{min} = \infty$
2: $e_{min} = \texttt{NULO}$
3: **para cada** entrada e em b **faça**
4: $d \leftarrow D(m, e.assinatura)$ {distância simétrica}
5: **se** $d < d_{min}$ **então**
6: $d_{min} \leftarrow d$
7: $e_{min} \leftarrow e.nome$
8: **fim se**
9: **fim para**
retorna: $\langle e_{min}, d_{min} \rangle$ {resultado}

Algoritmo 4.2: Procedimento de classificação proposto.

Para avaliar o desempenho de ambas as métricas foram realizados testes de classificação para representações de atratores geradas a partir de 10.000 (dez mil) amostras de TCP ISNs variando o valor de α e β de 0 a 1. Para cada caracterização de atrator na base de dados de assinaturas, construídas a parir de 100.000 (cem mil) amostras de TCP ISNs, será avaliado o desempenho de ambas as métricas para classificação outras representações de atratores gerados a partir da amostra de 10.000 (dez mil) TCP ISNs. As avaliações do parâmetro α e β para todos os sistemas operacionais e para o Honeyd são apresentadas nas Figuras 4.3 e 4.4, e no Apêndice B de forma mais detalhada.

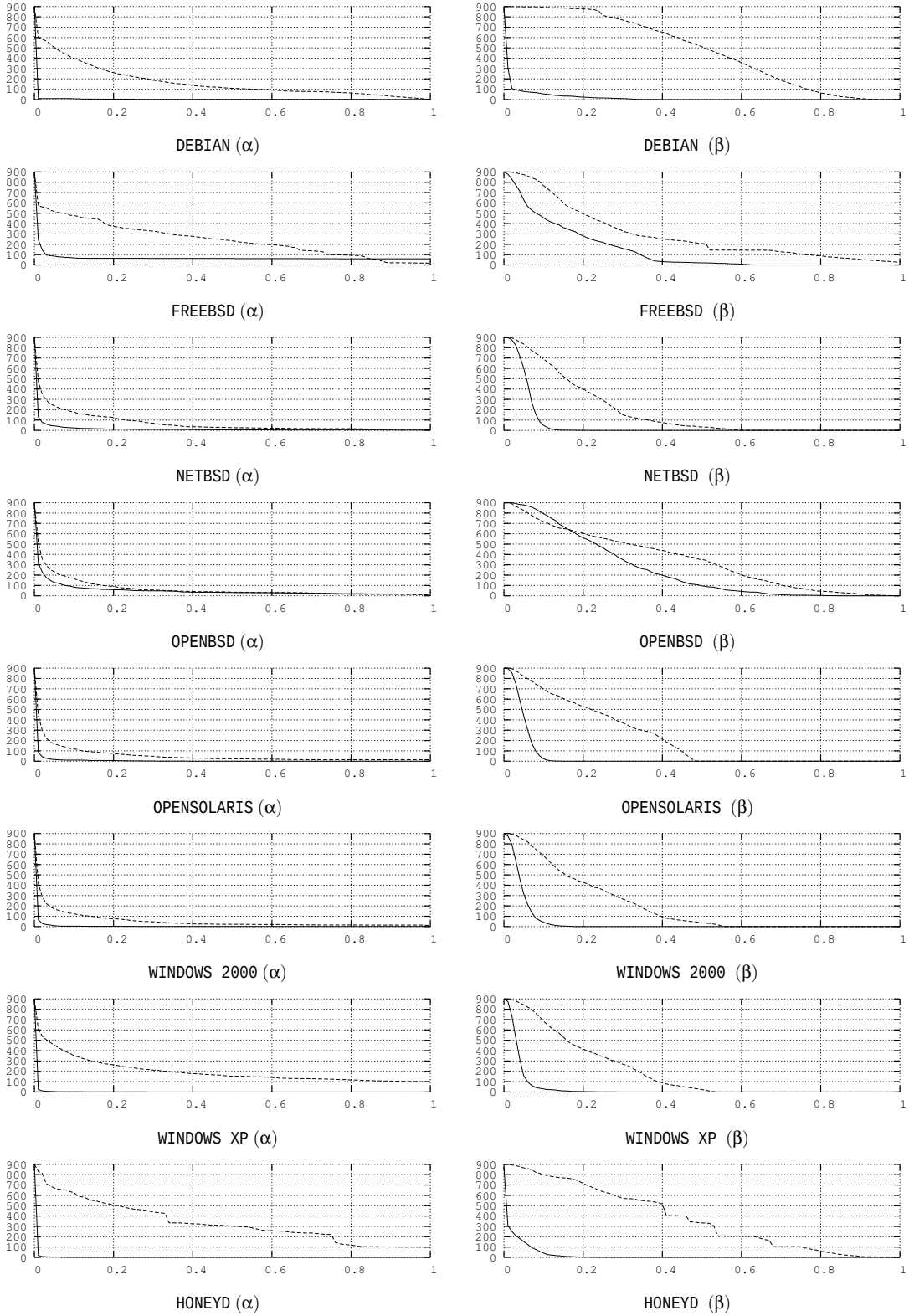

Figura 4.3: Avaliação relativa das métricas $N_s(X,Y,\alpha)$ e $M_s(X,Y,\beta)$ variando α e β entre 0 e 1. Onde a linha tracejada indica o classificação errada de menor valor e a linha cheia a classificação correta.

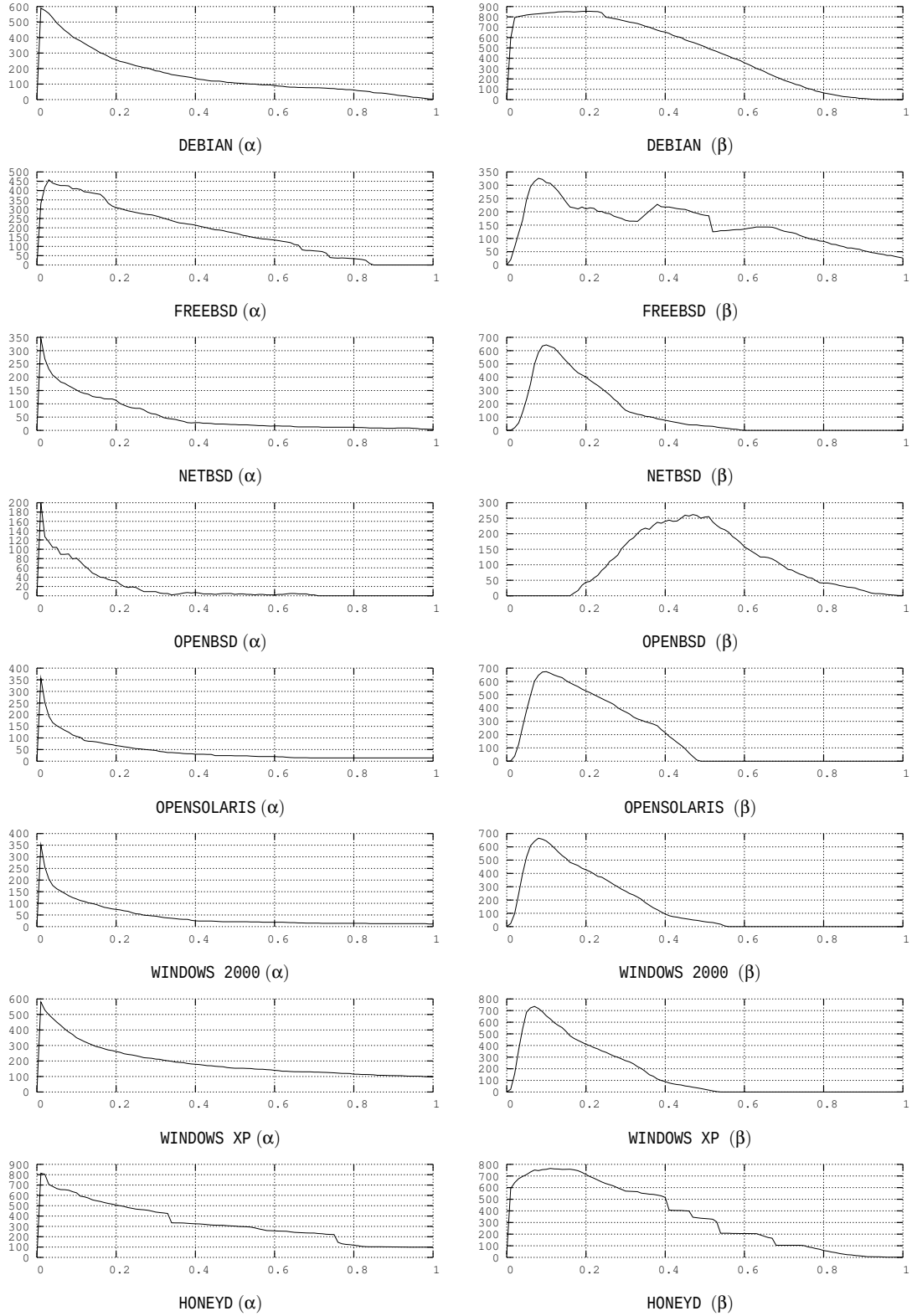

Figura 4.4: Avaliação absoluta das métricas $N_s(X,Y,\alpha)$ e $M_s(X,Y,\beta)$ variando α e β entre 0 e 1. Onde a linha indica a distância entre o valor correto e o errado mais próximo.

Pode ser verificado nas figuras relacionadas à variável α que seu valor quando escolhido próximo de $0,1$ é suficiente para classificar de forma correta todos os sistemas operacionais. Já para β, este valor se aproxima de $0,3$, isto porque, para o caso do OpenBSD (Figura B.10), valores inferiores a 0.2 classificam este sistema de forma incorreta. Os valores das métricas utilizadas para os valores de $\alpha = 0,1$ e $\beta = 0,3$ são apresentados na Tabela 4.1. Os menores valores para cada classificação são destacados com um "*".

A	B	$M_s(\cdot)$	$N_s(\cdot)$	A	B	$M_s(\cdot)$	$N_s(\cdot)$
Debian	Debian	*10	*8	OpenBSD	Debian	866	610
	FreeBSD	760	538		FreeBSD	518	611
	Honeyd	866	899		Honeyd	644	625
	NetBSD	856	845		NetBSD	491	402
	OpenBSD	861	808		OpenBSD	*351	*84
	OpenSolaris	873	834		OpenSolaris	371	118
	Windows 2000	857	757		Windows 2000	412	198
	Windows XP	725	503		Windows XP	619	457
FreeBSD	Debian	878	542	OpenSolaris	Debian	890	886
	FreeBSD	*160	*68		FreeBSD	508	641
	Honeyd	700	755		Honeyd	716	663
	NetBSD	425	759		NetBSD	522	238
	OpenBSD	624	636		OpenBSD	601	168
	OpenSolaris	457	597		OpenSolaris	*0	*11
	Windows 2000	445	607		Windows 2000	566	312
	Windows XP	510	513		Windows XP	607	424
Honeyd	Debian	890	894	Windows 2000	Debian	885	680
	FreeBSD	710	664		FreeBSD	328	630
	Honeyd	*0	*0		Honeyd	570	709
	NetBSD	583	808		NetBSD	152	176
	OpenBSD	616	577		OpenBSD	516	165
	OpenSolaris	691	687		OpenSolaris	549	311
	Windows 2000	574	710		Windows 2000	*0	*4
	Windows XP	684	647		Windows XP	269	352
NetBSD	Debian	891	805	Windows XP	Debian	769	399
	FreeBSD	449	738		FreeBSD	479	478
	Honeyd	591	778		Honeyd	610	628
	NetBSD	*0	*24		NetBSD	416	518
	OpenBSD	701	332		OpenBSD	699	620
	OpenSolaris	419	206		OpenSolaris	651	433
	Windows 2000	267	123		Windows 2000	297	340
	Windows XP	375	519		Windows XP	*0	*2

Tabela 4.1: Classificação utilizando $\alpha = 0,1$ e $\beta = 0,3$.

Fica assim demonstrado que é factível a utilização das técnicas propostas neste trabalho para desempenhar a tarefa de identificação em situações onde as outras ferramentas analisadas falham. Na Tabela 4.1 e nas figuras relacionadas à analise dos parâmetros de classificação α e β é possível notar que cada métrica se mostrou mais adequada para classificar um dado sistema operacional. Por exemplo, a utilização da métrica $M_s(\cdot)$ para classificação do Debian, NetBSD e OpenSolaris, e a utilização da métrica $N_s(\cdot)$ para classificação do FreeBSD, Honeyd e OpenBSD. Nos outros dois casos as métricas possuem desempenho de classificação semelhantes. Este fato sugere que uma criação de uma métrica híbrida possa melhorar o desempenho de classificação.

Capítulo 5

Conclusão

Neste trabalho foram apresentados os aspectos relacionados a eficácia e a confiabilidade de uma ferramenta de identificação remota de sistemas operacionais. Em relação à eficácia, foram apresentados resultados que demonstram a viabilidade da utilização do método desenvolvido para classificação dos sistemas operacionais analisados. A utilização de dados não influenciados pela utilização de normalização de pacotes e utilização de tradução de endereços foi a chave para o sucesso neste aspecto.

A confiabilidade do método desenvolvido foi confirmada através da possibilidade de identificação da ferramenta Honeyd e do SYN *proxy* implementado pela *firewall* PF. Em ambos os casos, o gerador do número inicial de seqüência possui um comportamento diferenciado daqueles apresentados por cada um dos sistemas operacionais analisados.

A tratabilidade é uma características que pode ser facilmente incorporada ao método de identificação proposto. Esta característica é particularmente importante na identificação de dispositivos sensíveis, como pode ser verificado em dispositivos de automação industrial, que, quando submetidos ao processo de identificação do Nmap, apresentaram problemas relacionados a funcionamento e perda de comunicação. Para incorporar a tratabilidade ao método aqui proposto pode-se adotar duas estratégias, sendo ambas relacionadas ao intervalo de amostragem τ. Na primeira deve-se realizar um estudo afim de determinar o intervalo τ necessário para não estourar o *buffer* de recebimento de mensagens de sincronização da máquina remota. Já na segunda, evita-se o esgotamento deste *buffer* enviando cada nova requisição de sincronização apenas quando a anterior tenha sido finalizada através do envio de uma mensagem RST (*reset*). A primeira estratégia, embora exija um estudo adicional relacionado à capacidade de recebimento de mensagens de sincronização simultâneas de cada sistema operacional, não garante que o recebimento das mensagens na máquina alvo seja compatível com a taxa enviada pela máquina de origem. A segunda estratégia possui o problema de não determinismo do valor τ. Porém, testes preliminares apontam que este fato não influencia significativamente o resultado.

Em relação a eficiência, duas etapas do processo de identificação devem ser melhoradas. Primeiramente, esforços em busca de uma quantidade menor de amostras de números iniciais de seqüência necessários para classificação devem ser realizados. O outro ponto que deve ser considerado é o tempo de processamento para criação da caracterização do atrator e para a classificação destes.

Por utilizar pacotes válidos e comuns no processo de estabelecimento de comunicações em redes TCP/IP, esta técnica não pode ser detectada por ferramentas baseadas em assinatura de pacotes. Porém, a quantidade atual de amostras necessárias pode caracterizar este processo de identificação. Na medida em que este número de amostras necessárias for reduzido, pode-se criar mecanismos que torne o processo de captura dos dados semelhante ao tráfego usual de uma rede.

Outros protocolos, como é o caso do *Domain Name System* (DNS) [Mockapetris 1987], também utilizam componentes aleatórias na comunicação. Nestes casos, o método proposto neste trabalho também pode ser utilizado para identificar as diferentes implementações destes protocolos.

De uma forma geral, este trabalho apresenta uma nova abordagem para caracterização e classificação de sistemas dinâmicos. Isto se deve ao fato de que o único requisito para tornar possível a caracterização de um sistema é um conjunto de amostras da saída do mesmo. Portanto, a aplicabilidade do método desenvolvido pode ser estendida diretamente para outras áreas da ciência e engenharia.

5.1 Trabalhos futuros

A quantidade de recursos matemáticos utilizados para desempenhar esta tarefa de classificação e a complexidade de obtenção de informações significativas para caracterização de sistemas operacionais de forma remota sugerem possíveis extensões deste trabalho. A seguir, são listados alguns tópicos que têm como objetivo aprimorar a técnica de identificação desenvolvida.

- Verificar os números de seqüência de cada sistema operacional em sua forma binária visando extrair possíveis características não visíveis, a princípio, quando utilizada a representação decimal;
- Analisar e adicionar mais sistemas operacionais na base de dados de assinaturas;
- Verificar que outras informações, tão singulares e robustas quanto o ISN, podem ser utilizadas para classificação de sistemas operacionais de forma remota, principalmente no caso em que dois sistemas operacionais distintos apresentem o mesmo atrator;
- Utilizar métodos de otimização para encontrar parâmetros ótimos para as equações de estimação da densidade e da orientação dos atratores;
- Analisar estratégias para reduzir a quantidade de amostras necessárias para criação de atratores representativos;
- Verificar a dimensão do atrator de cada sistema operacional a fim de criar representações mais expressivas;
- Criar uma métrica híbrida que incorpore as vantagens das duas métricas utilizadas;

- Verificar a influência da utilização de *proxies* de serviços na aquisição de amostras.
- Verificar a utilização de outros algoritmos de caracterização, ou extensão do algoritmo SOM utilizado, para o caso de espaços de fase de dimensão superior a $\mathbb{R}^2$. Por exemplo, verificar se a utilização dos atratores apresentados no Apêndice A, caracterizados pelo algoritmo *Growing Neural Gas* (GNG) [Fritzke 1993, Fritzke 1995], melhora os resultados do processo de identificação.

Apêndice A

Atratores em Espaço de Fase 3D

Neste apêndice são apresentados os atratores em espaço de fase tridimensional para cada sistema operacional analisado. A fim de proporcionar uma visão auxiliar do atrator tridimensional, são apresentados os planos bidimensionais $\overline{XY}$, $\overline{YZ}$ e $\overline{XZ}$ para cada atrator. Para o caso do atrator do Debian, que possui uma concentração de pontos não favorecida na visualização, são aplicados dois níveis de ampliação do atrator. Vê-se que, em alguns casos, a utilização da terceira dimensão apresenta informações ao atrator original. Como é o caso do plano $\overline{XZ}$ dos atratores do Debian, NetBSD e OpenSolaris. Para estes sistemas, o plano $\overline{XZ}$ mostrou um comportamento diferente dos outros planos. Em trabalhos futuros esta dimensão adicional pode ser utilizada para melhorar a caracterização do sistema.

A.1 Debian

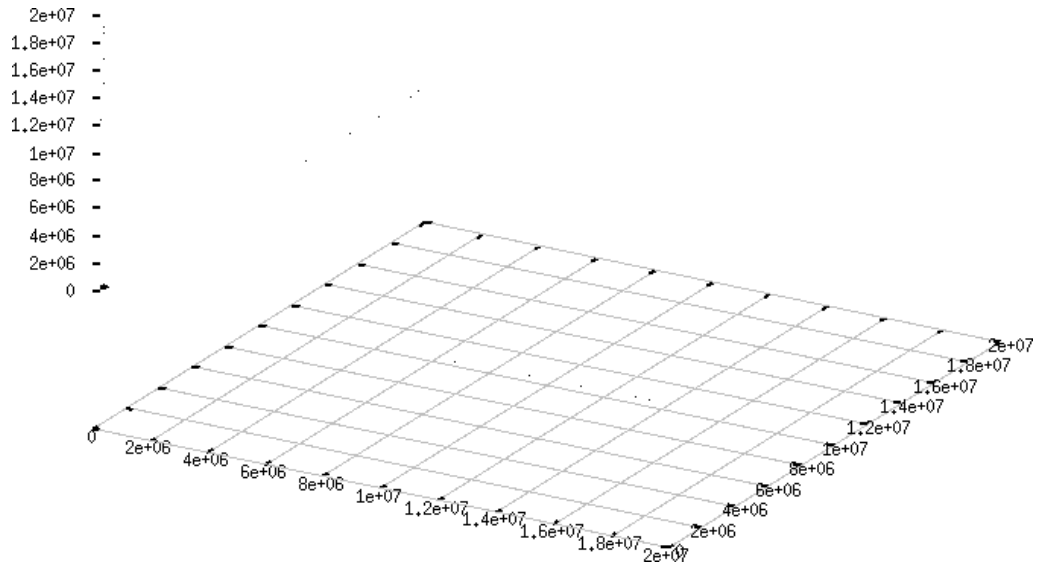

Figura A.1: Atrator do Debian.

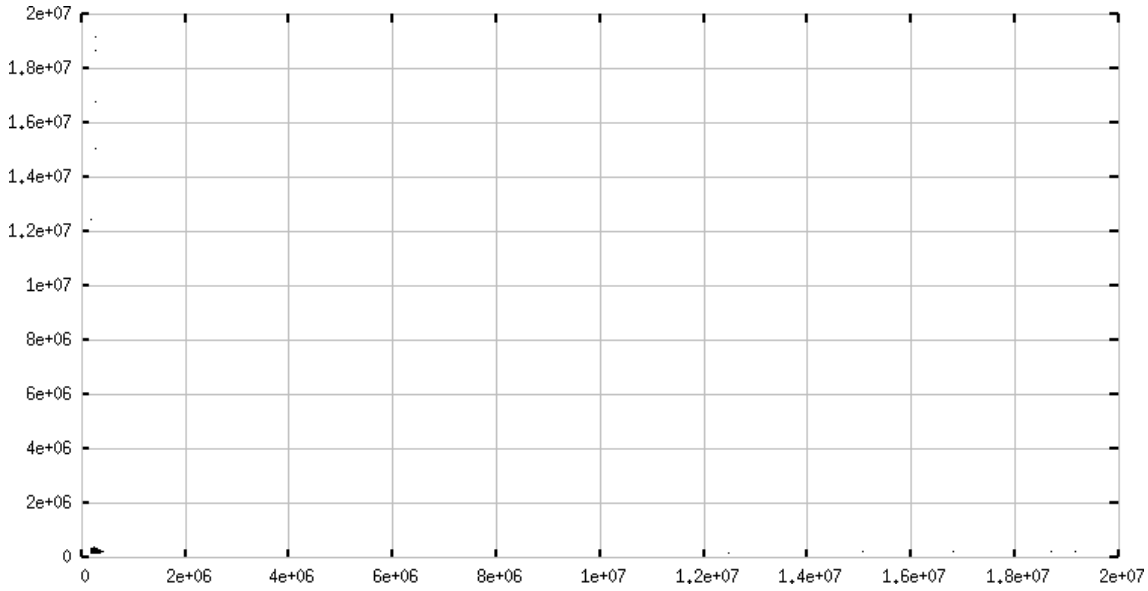

Figura A.2: Plano $\overline{XY}$ do atrator do Debian.

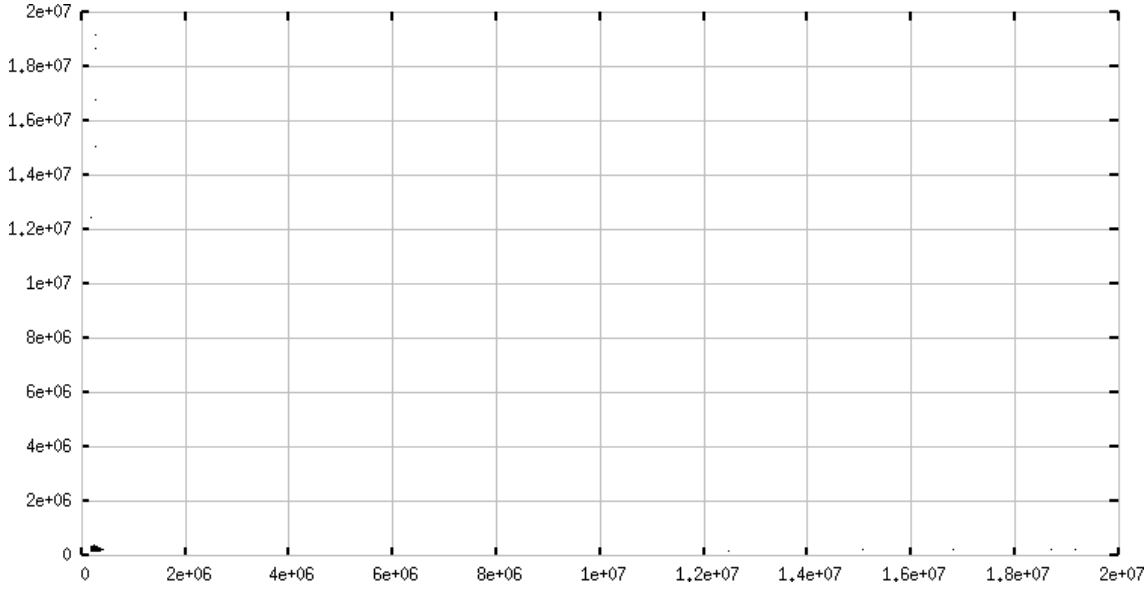

Figura A.3: Plano $\overline{YZ}$ do atrator do Debian.

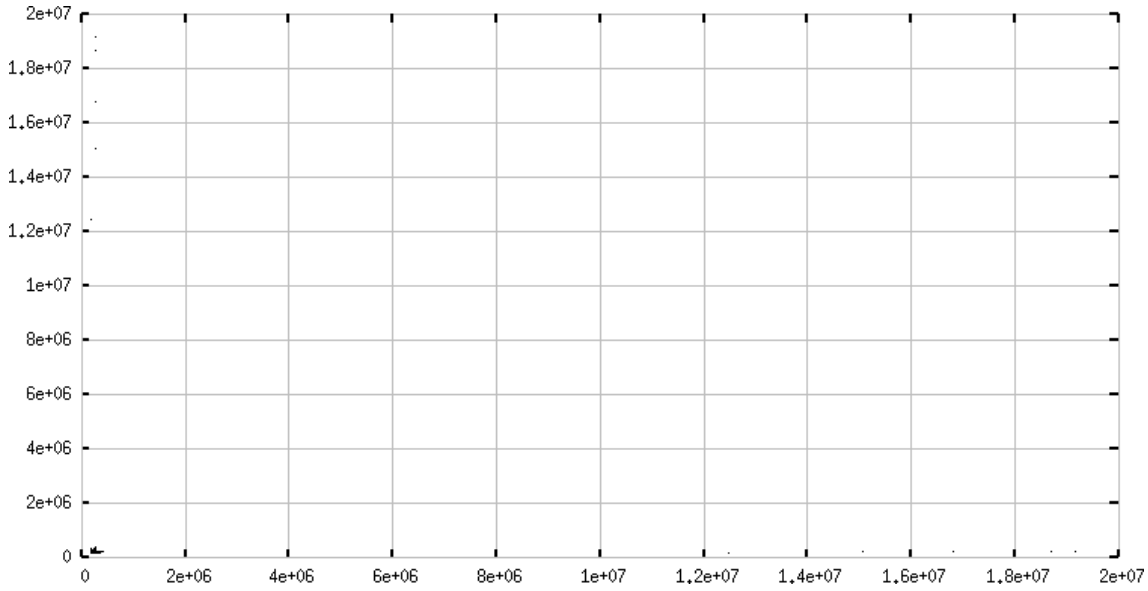

Figura A.4: Plano $\overline{XZ}$ do atrator do Debian.

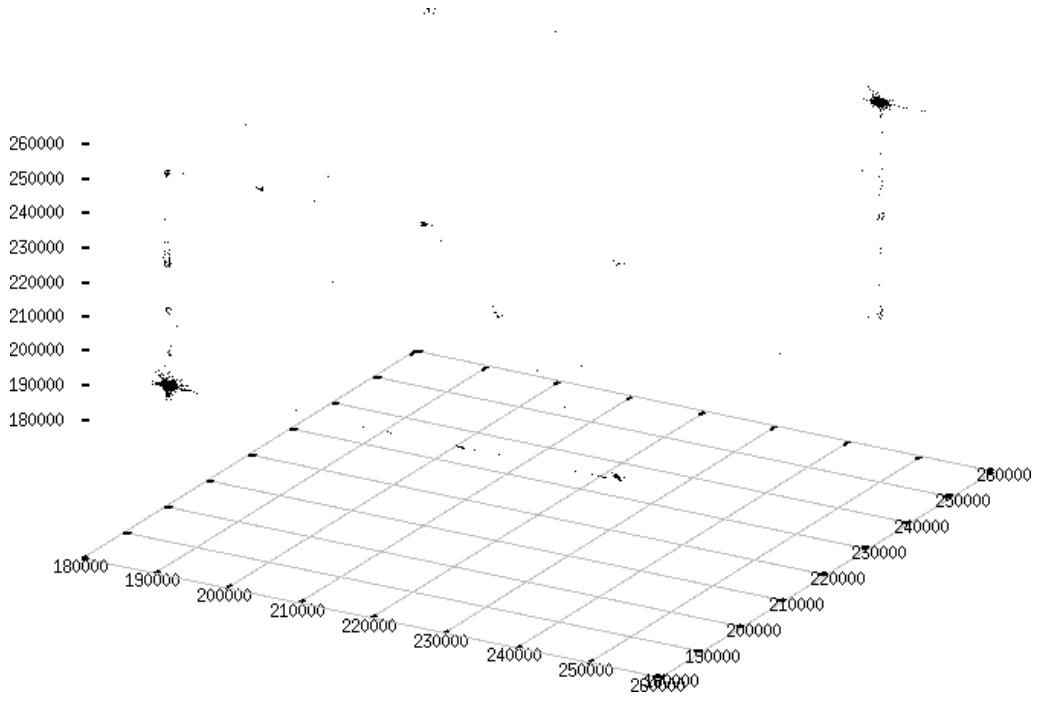

Figura A.5: Atrator do Debian (ampliado de 180.000 a 260.000).

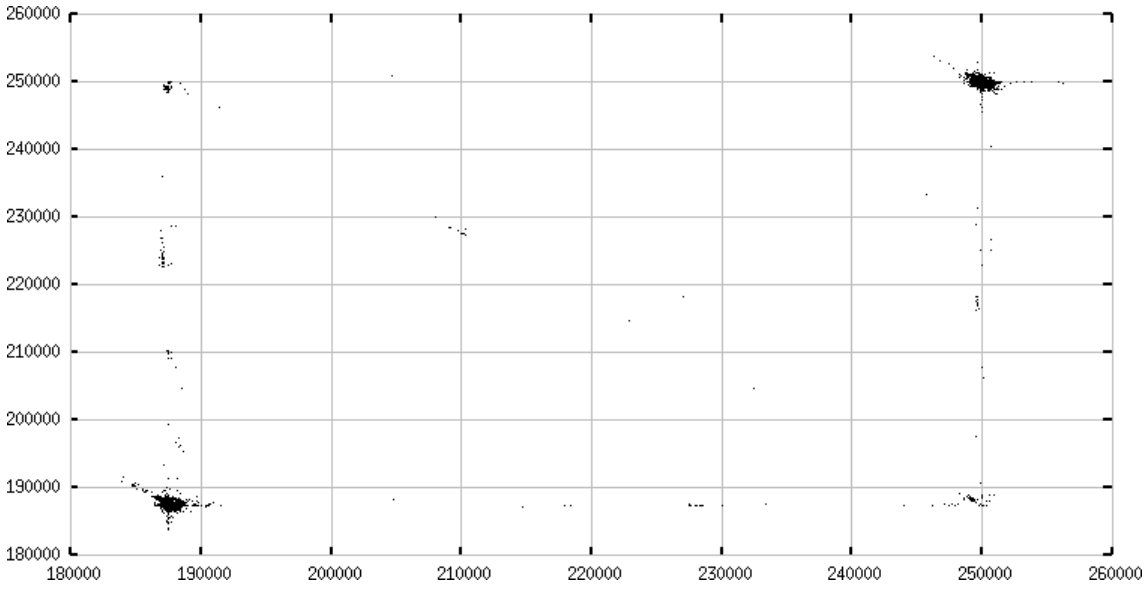

Figura A.6: Plano $\overline{XY}$ do atrator do Debian (ampliado de 180.000 a 260.000).

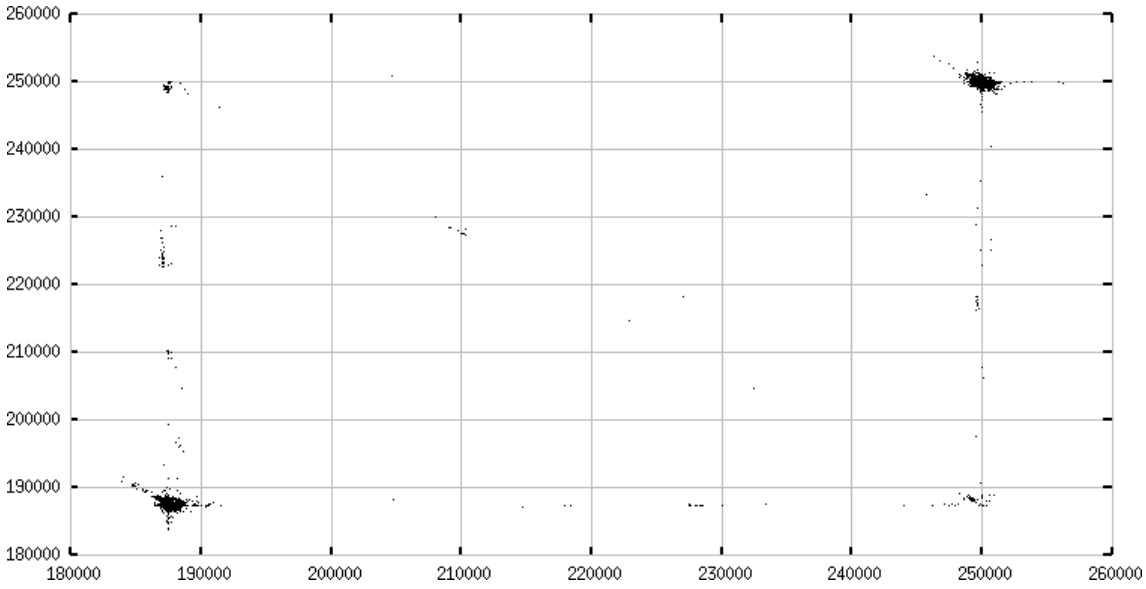

Figura A.7: Plano $\overline{YZ}$ do atrator do Debian (ampliado de 180.000 a 260.000).

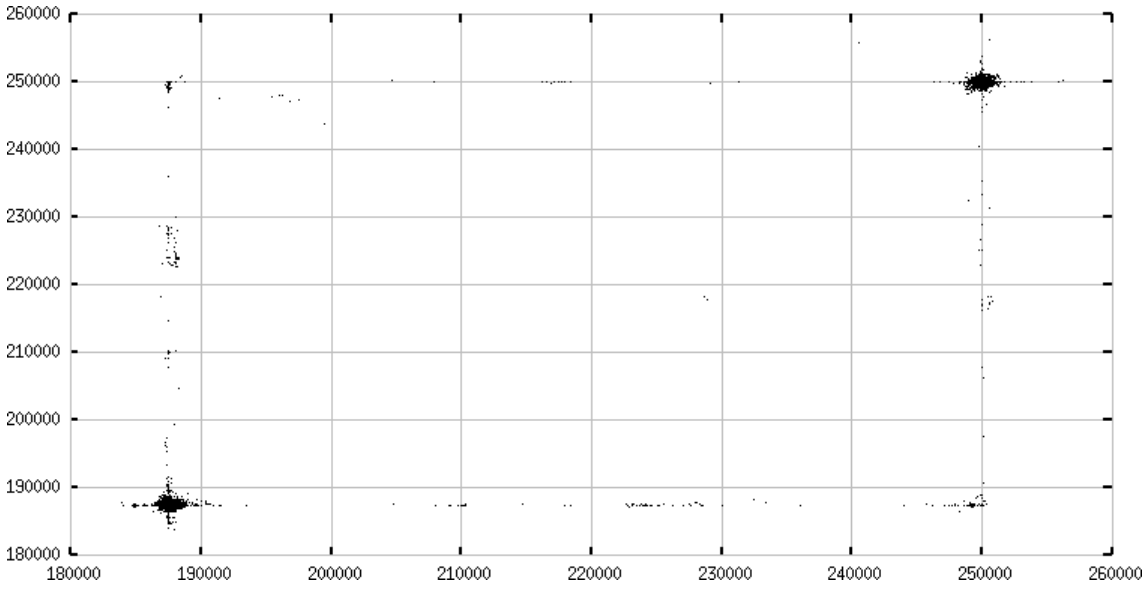

Figura A.8: Plano $\overline{XZ}$ do atrator do Debian (ampliado de 180.000 a 260.000).

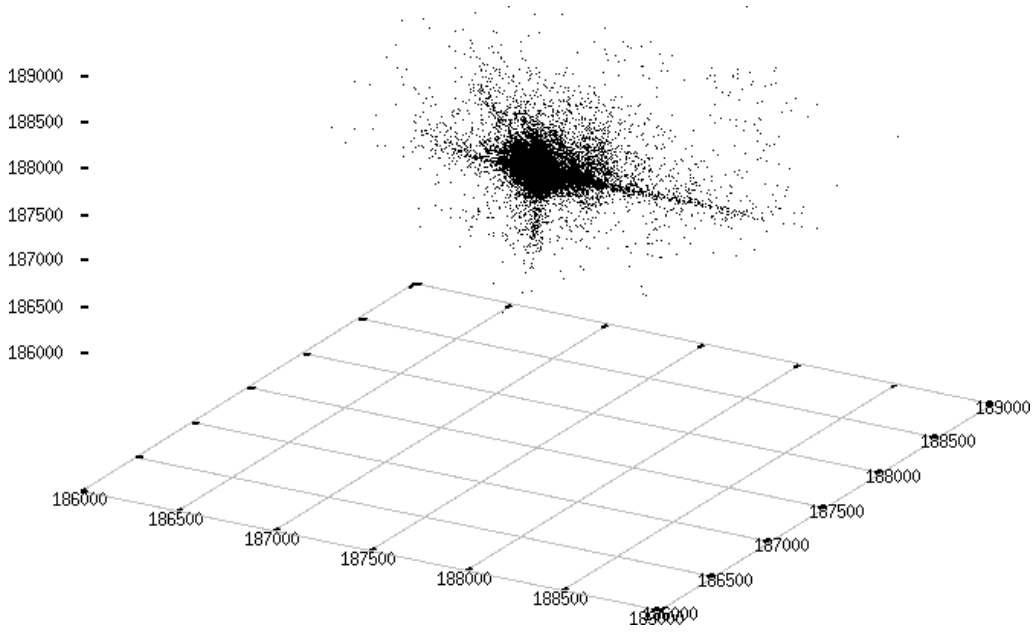

Figura A.9: Atrator do Debian (ampliado de 186.000 a 189.000).

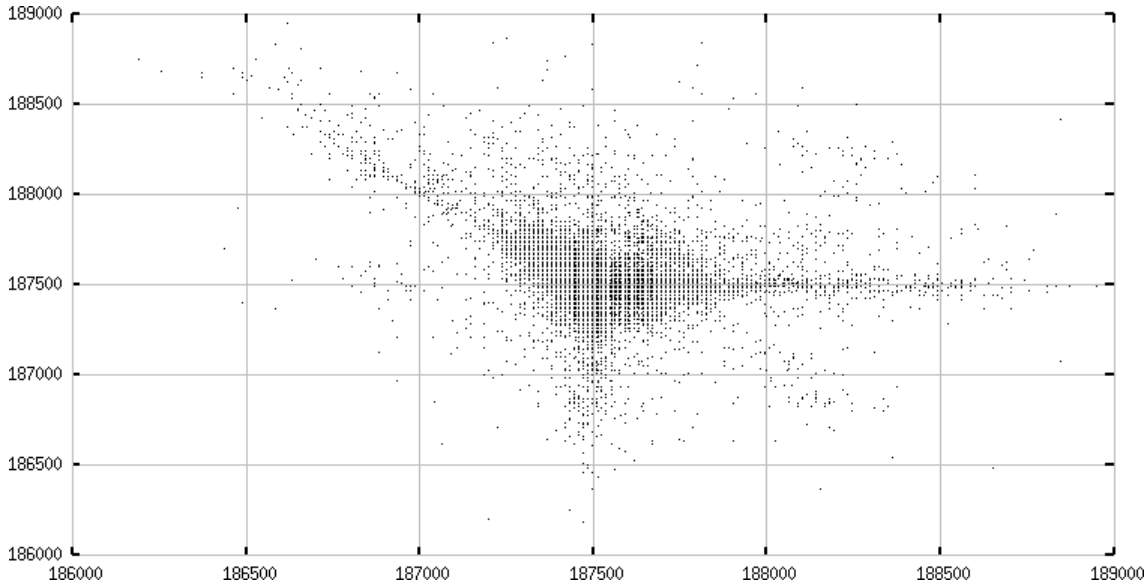

Figura A.10: Plano $\overline{XY}$ do atrator do Debian (ampliado de 186.000 a 189.000).

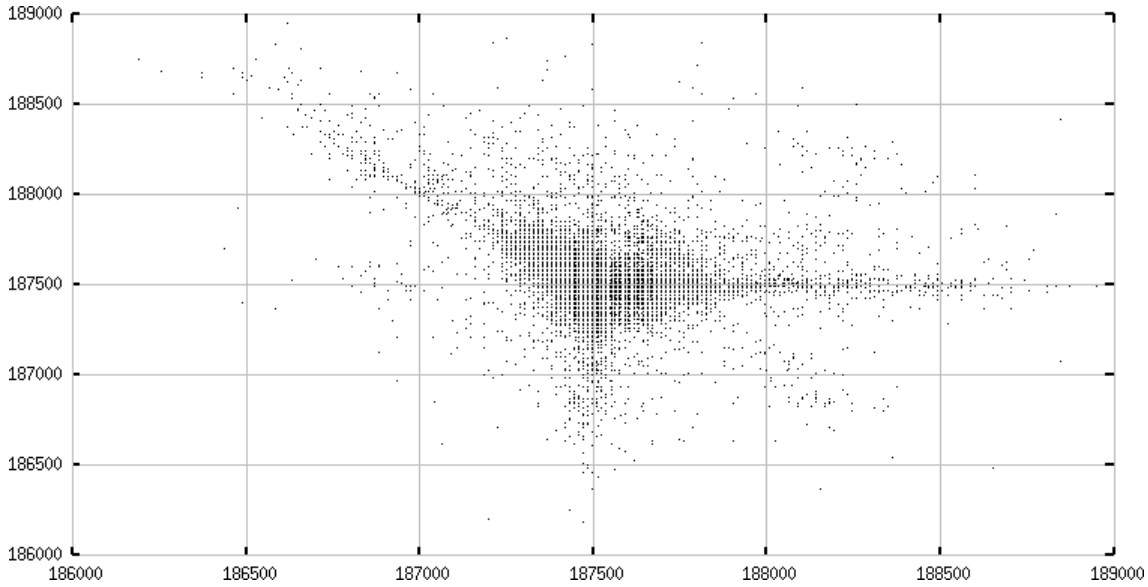

Figura A.11: Plano $\overline{YZ}$ do atrator do Debian (ampliado de 186.000 a 189.000).

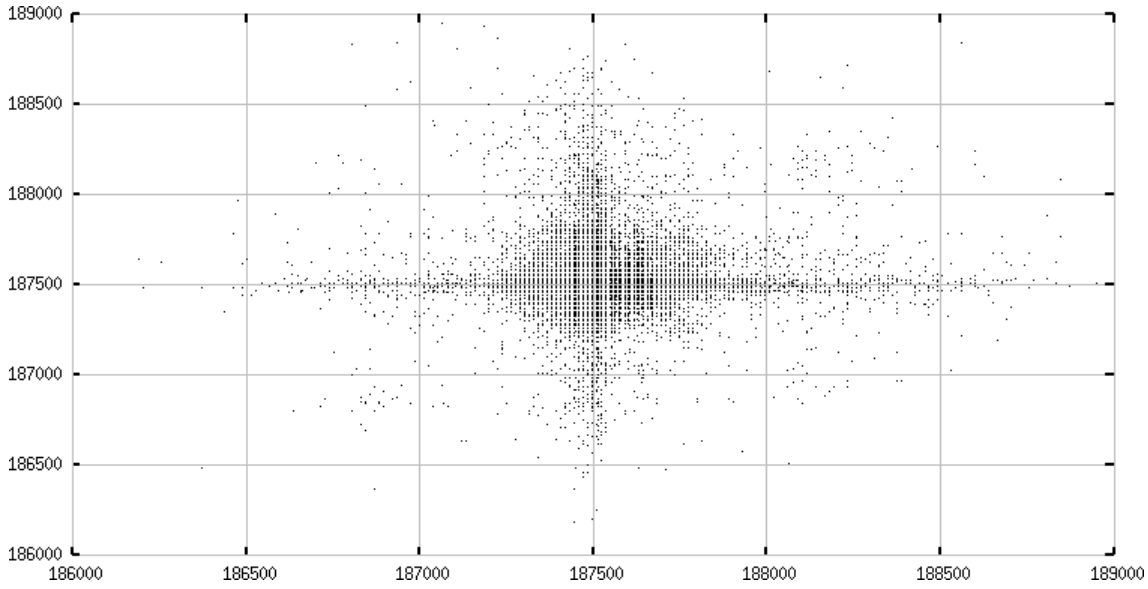

Figura A.12: Plano $\overline{XZ}$ do atrator do Debian (ampliado de 186.000 a 189.000).

A.2 FreeBSD

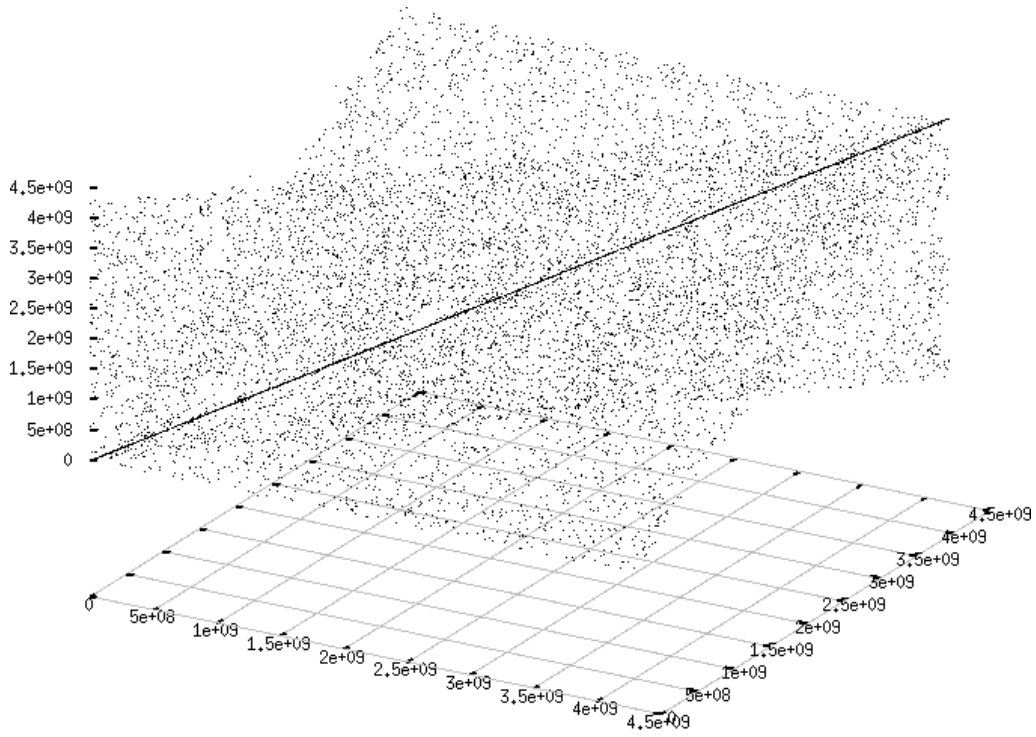

Figura A.13: Atrator do FreeBSD.

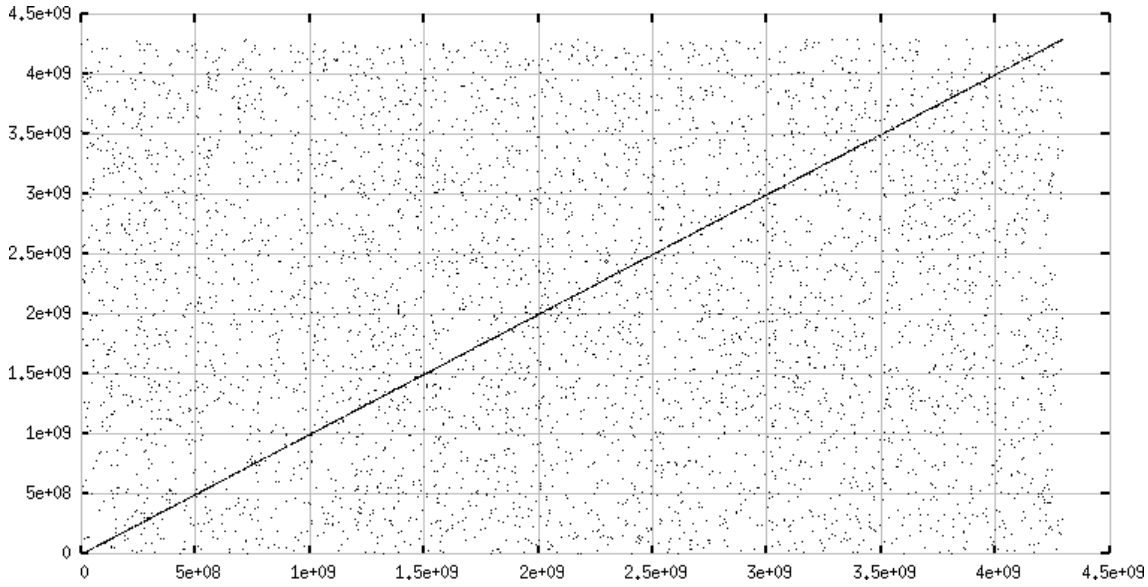

Figura A.14: Plano $\overline{XY}$ do atrator do FreeBSD.

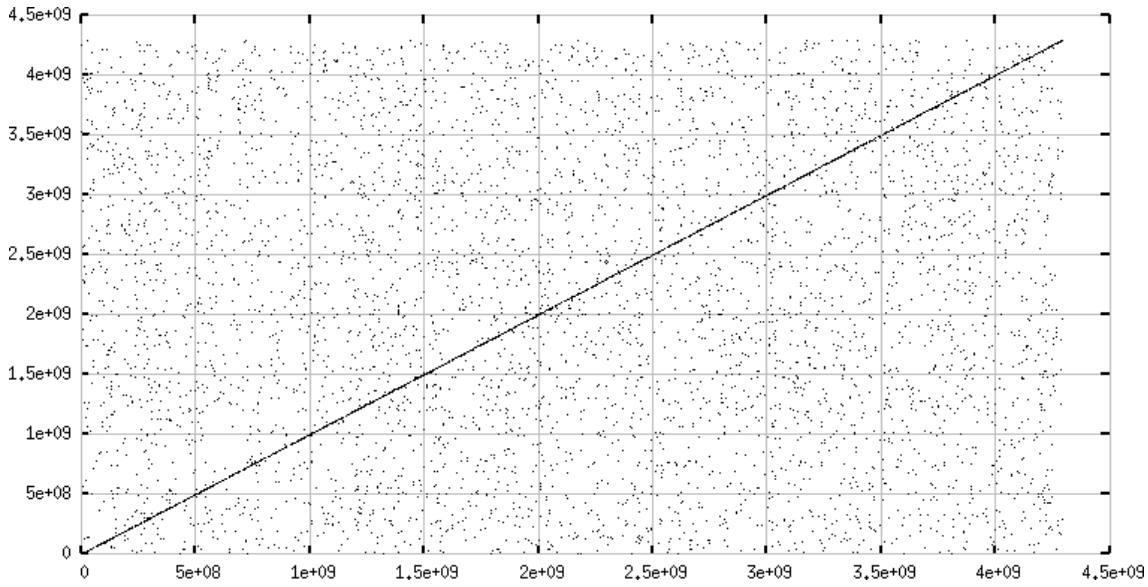

Figura A.15: Plano $\overline{YZ}$ do atrator do FreeBSD.

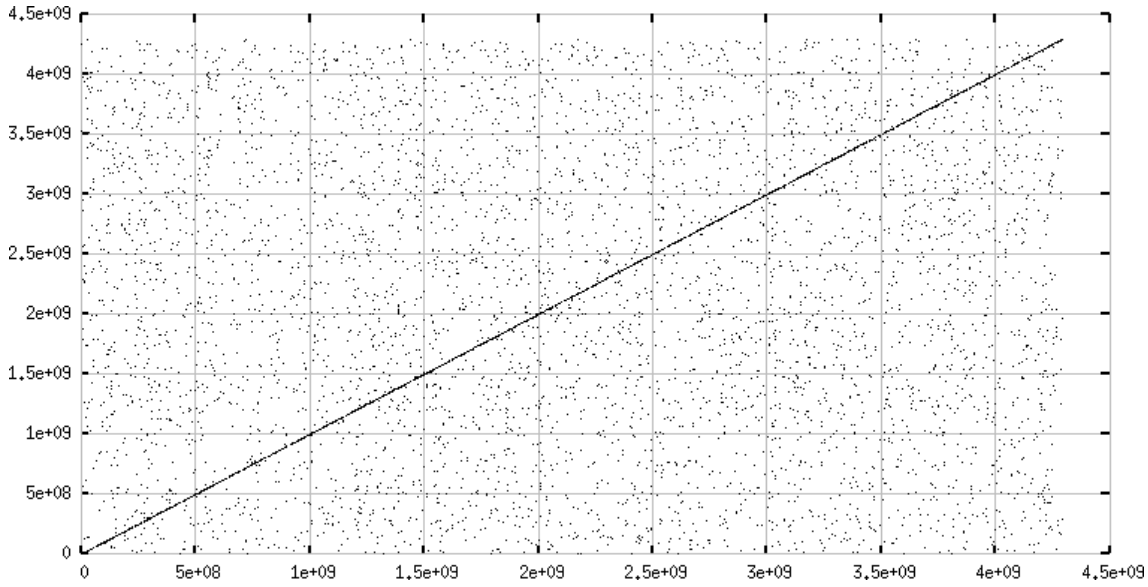

Figura A.16: Plano $\overline{XZ}$ do atrator do FreeBSD.

A.3 NetBSD

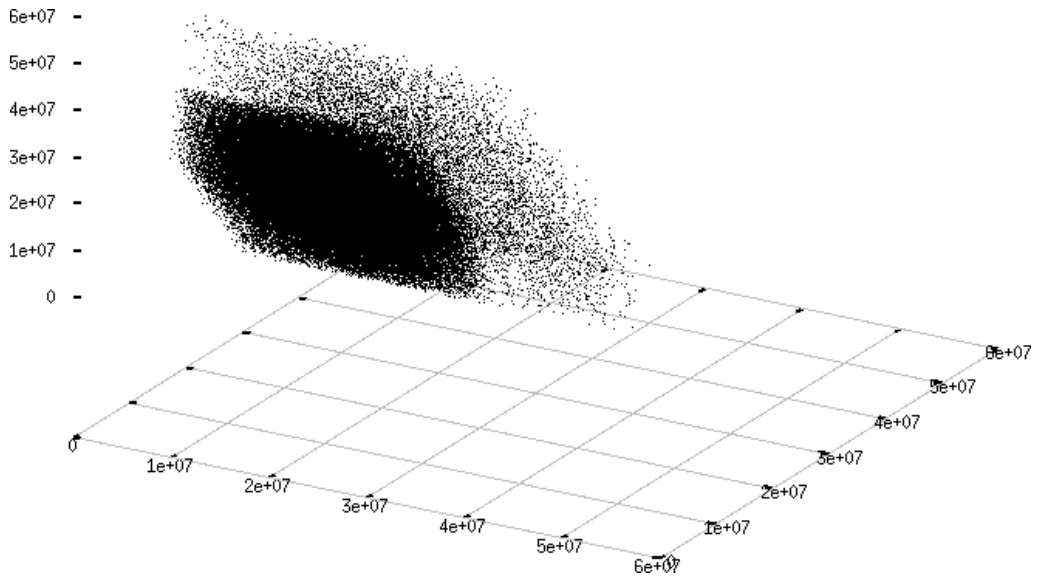

Figura A.17: Atrator do NetBSD.

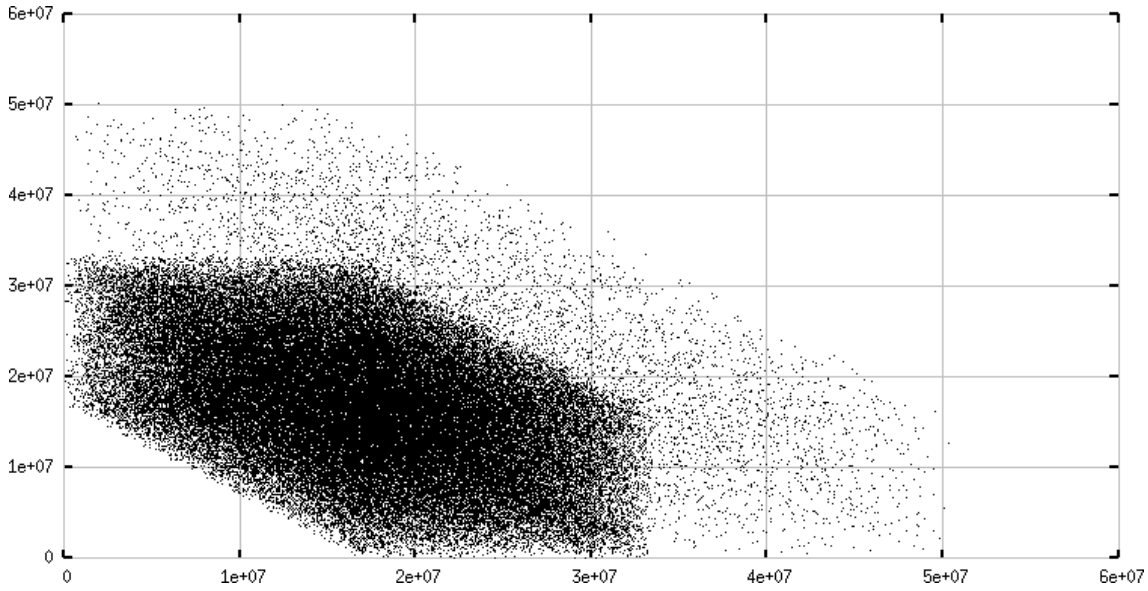

Figura A.18: Plano $\overline{XY}$ do atrator do NetBSD.

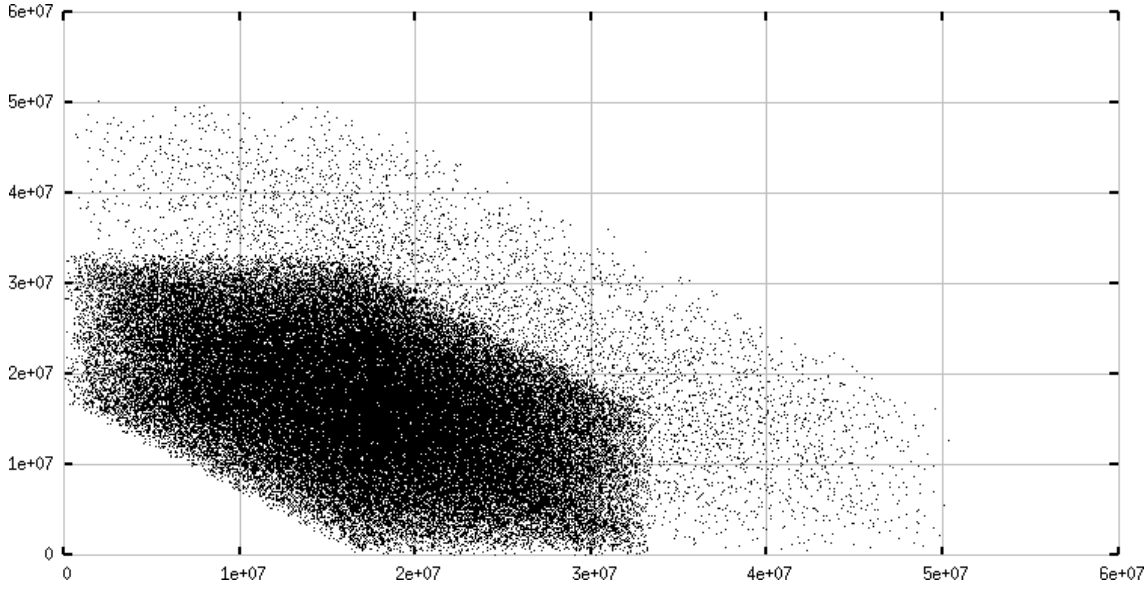

Figura A.19: Plano $\overline{YZ}$ do atrator do NetBSD.

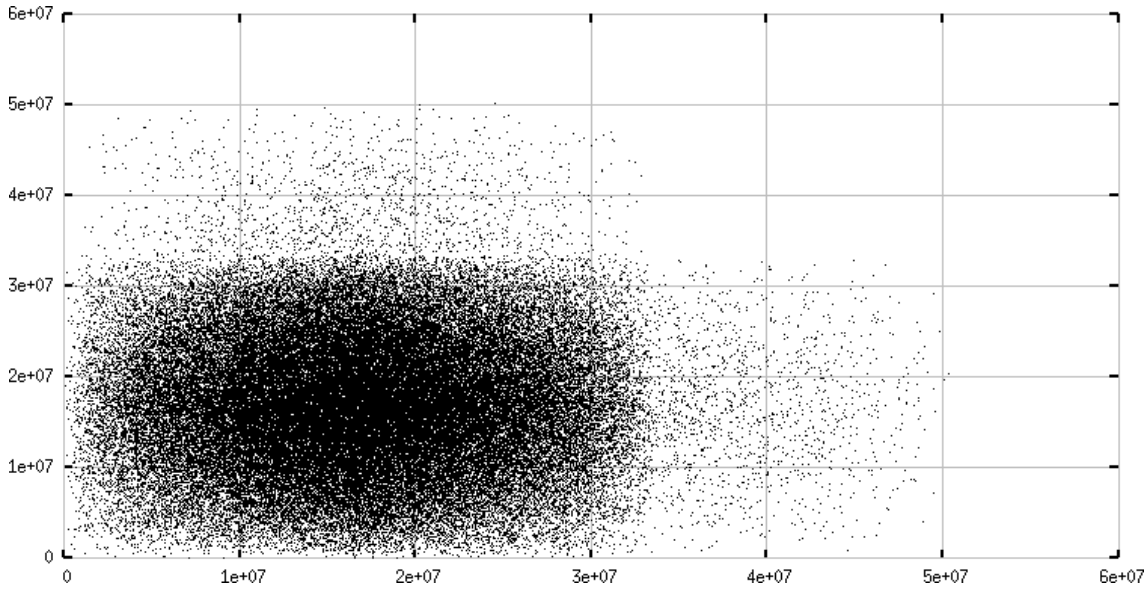

Figura A.20: Plano $\overline{XZ}$ do atrator do NetBSD.

A.4 OpenBSD

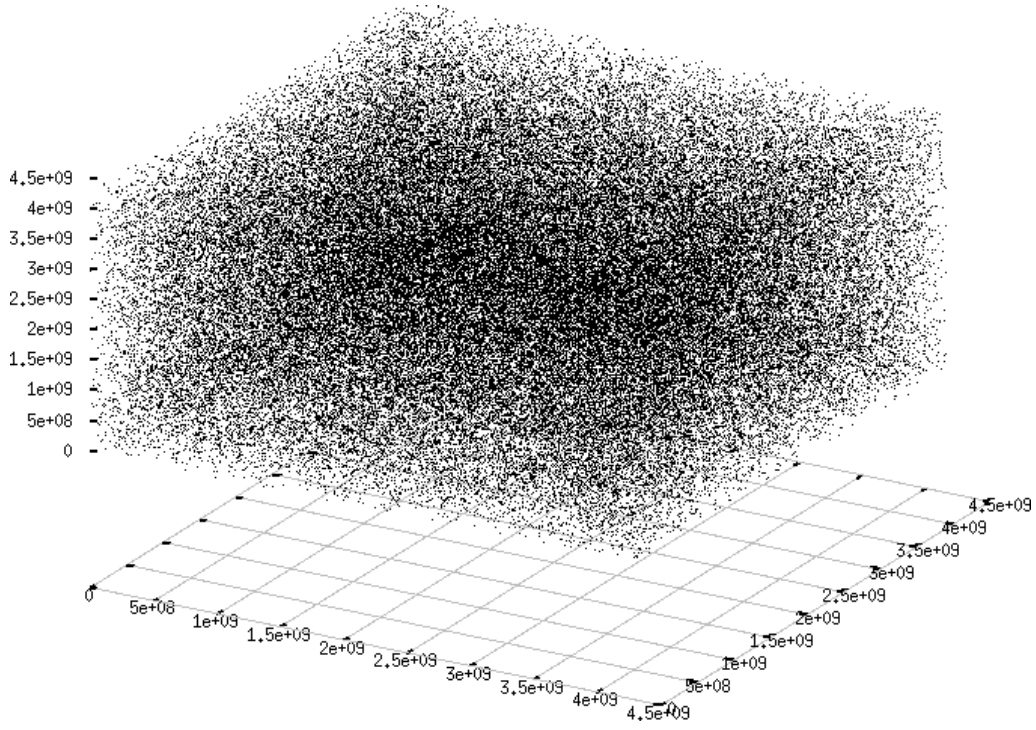

Figura A.21: Atrator do OpenBSD.

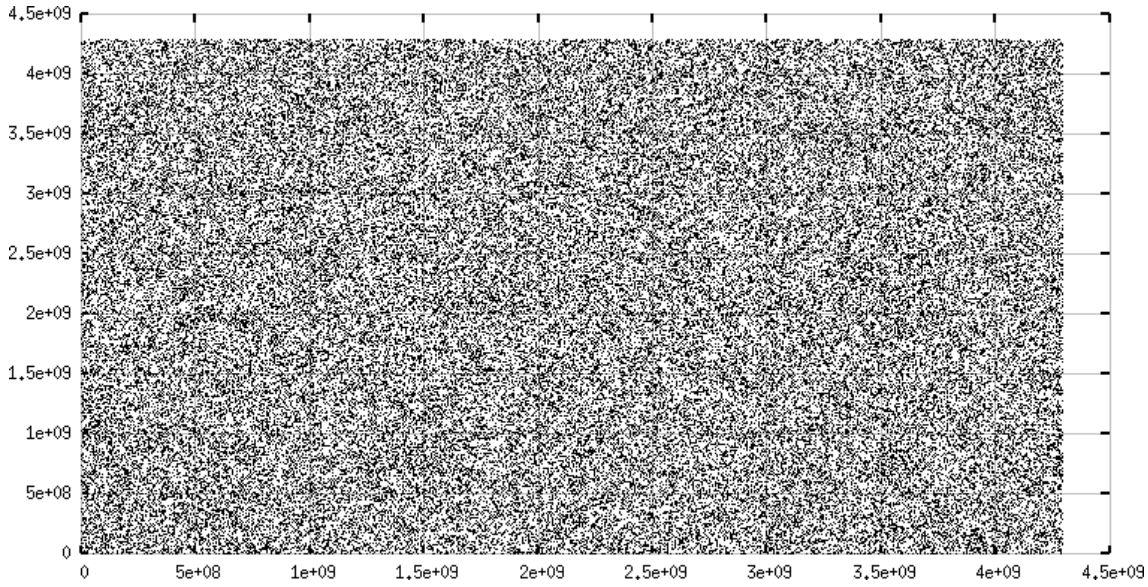

Figura A.22: Plano $\overline{XY}$ do atrator do OpenBSD.

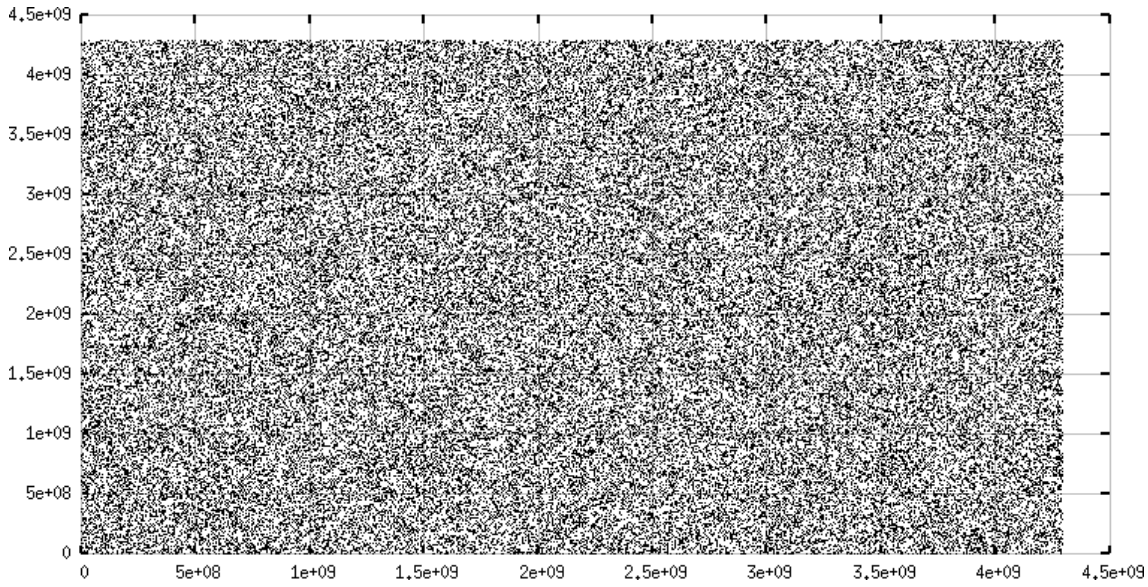

Figura A.23: Plano $\overline{YZ}$ do atrator do OpenBSD.

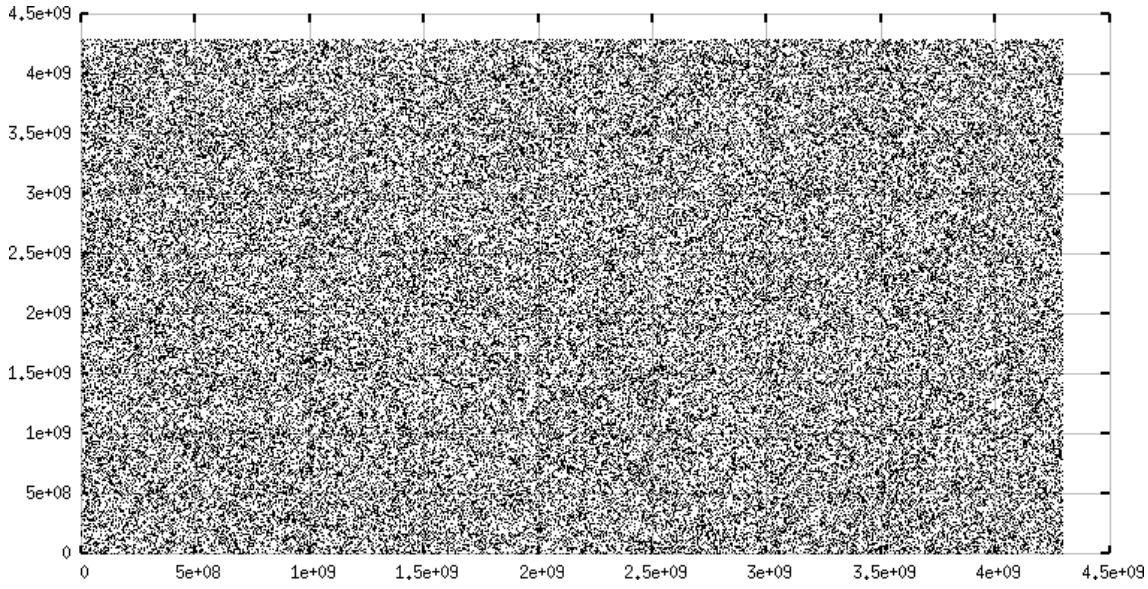

Figura A.24: Plano $\overline{XZ}$ do atrator do OpenBSD.

A.5 OpenSolaris

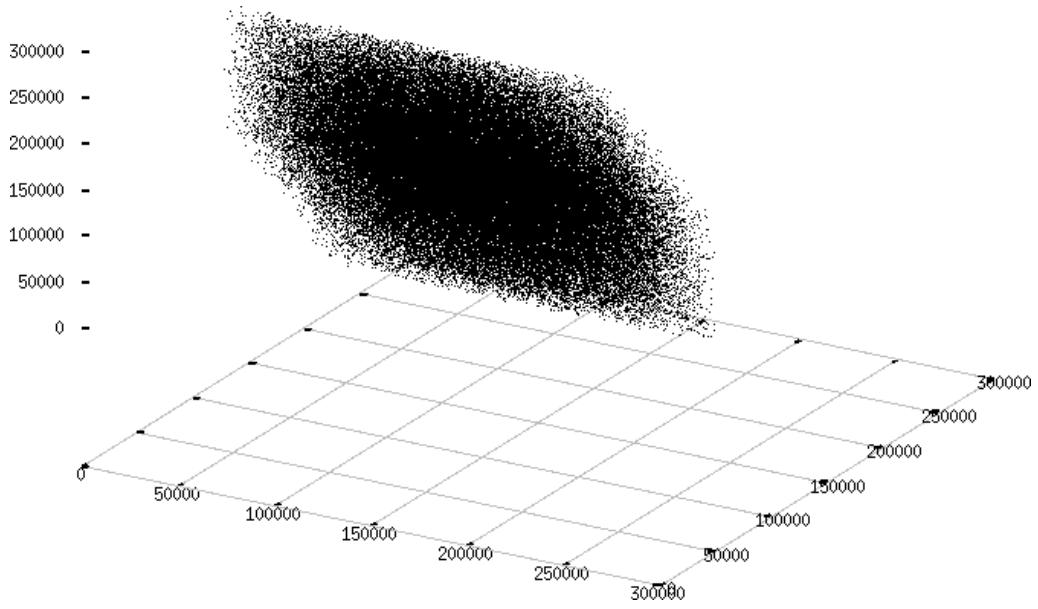

Figura A.25: Atrator do OpenSolaris.

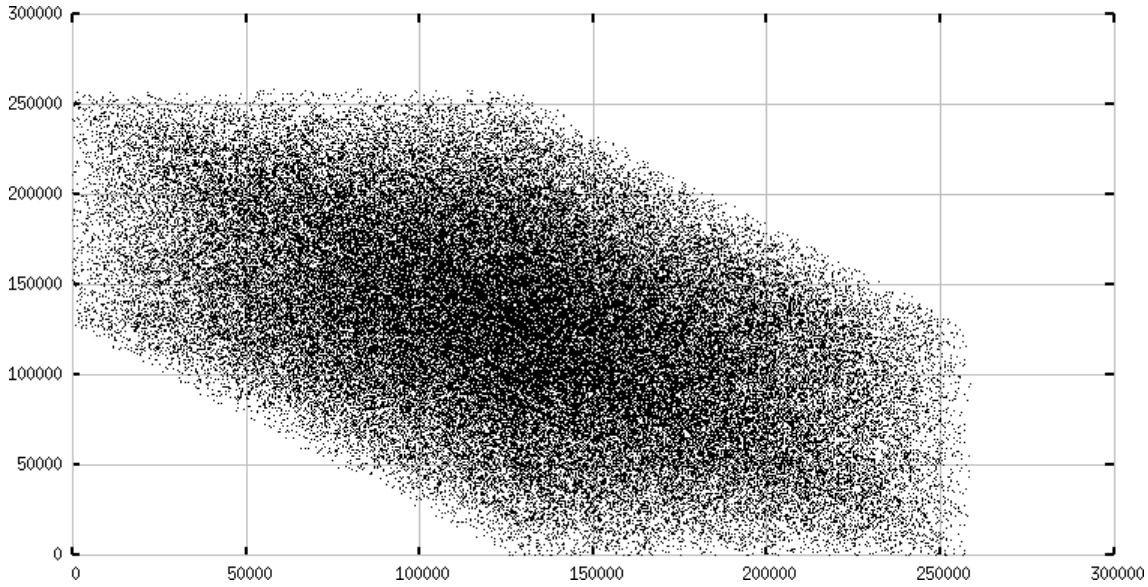

Figura A.26: Plano $\overline{XY}$ do atrator do OpenSolaris.

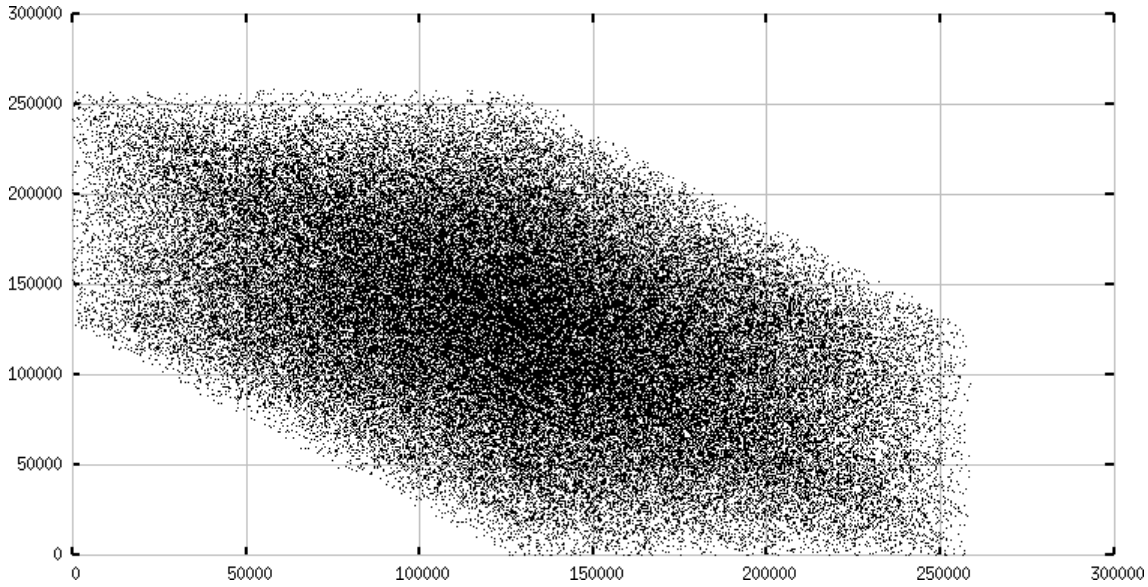

Figura A.27: Plano $\overline{YZ}$ do atrator do OpenSolaris.

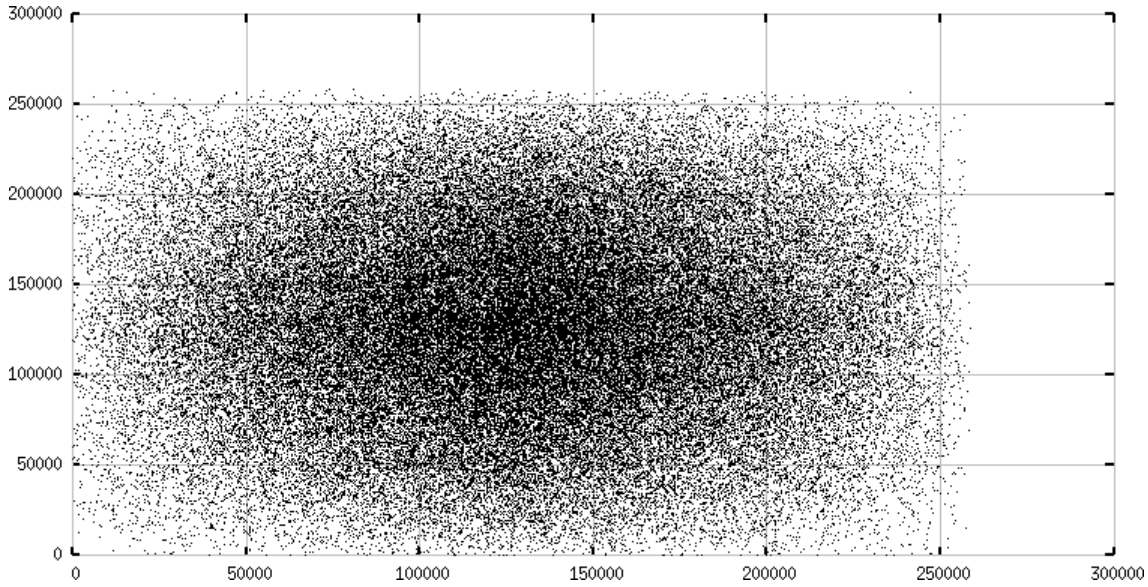

Figura A.28: Plano $\overline{XZ}$ do atrator do OpenSolaris.

A.6 Windows 2000

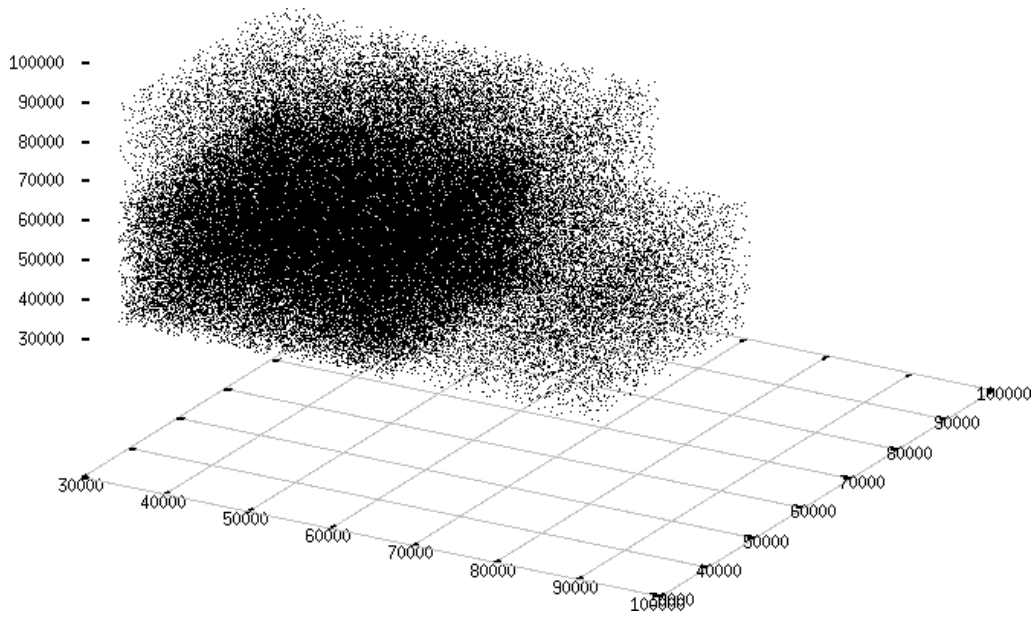

Figura A.29: Atrator do Windows 2000.

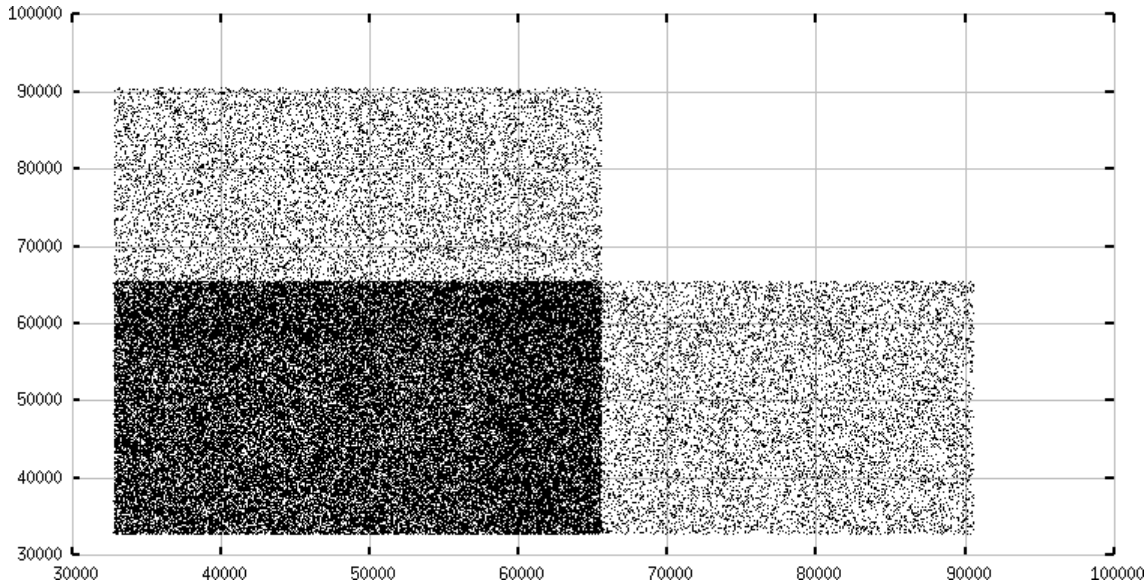

Figura A.30: Plano $\overline{XY}$ do atrator do Windows 2000.

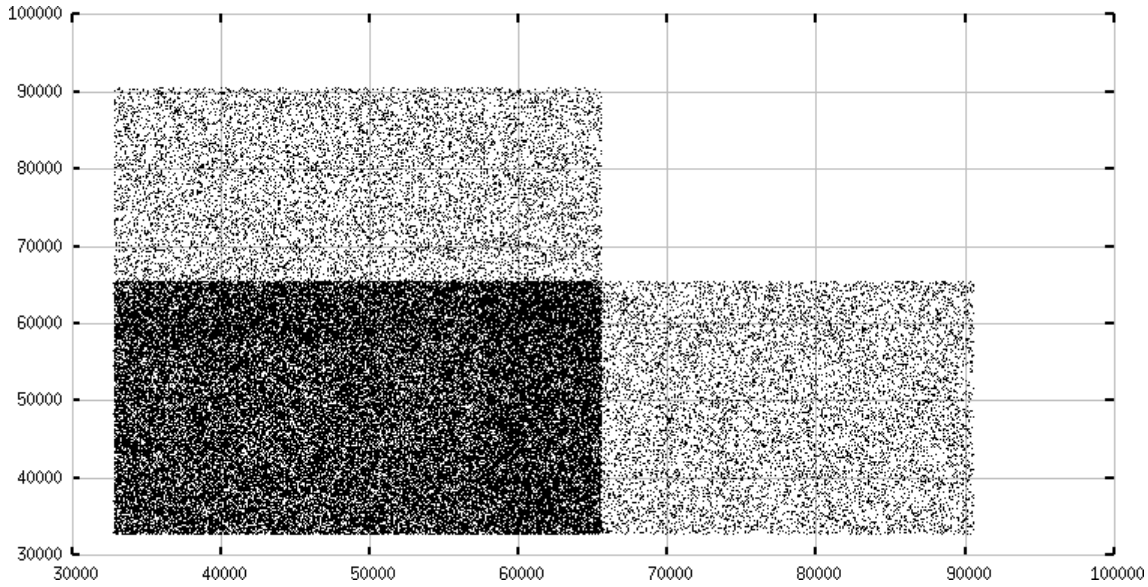

Figura A.31: Plano $\overline{YZ}$ do atrator do Windows 2000.

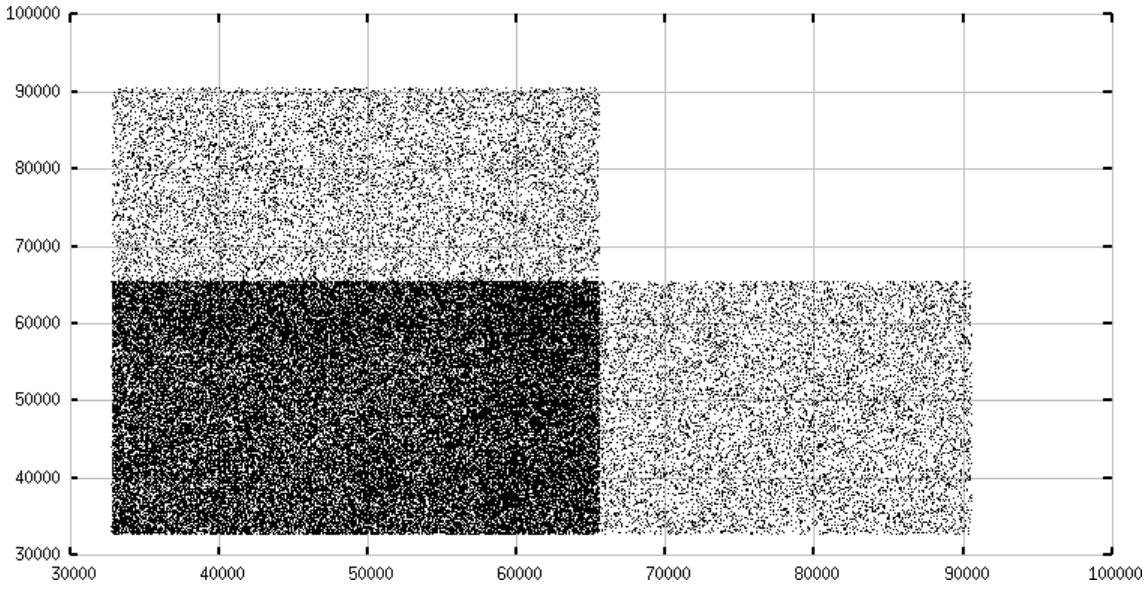

Figura A.32: Plano $\overline{XZ}$ do atrator do Windows 2000.

A.7 Windows XP

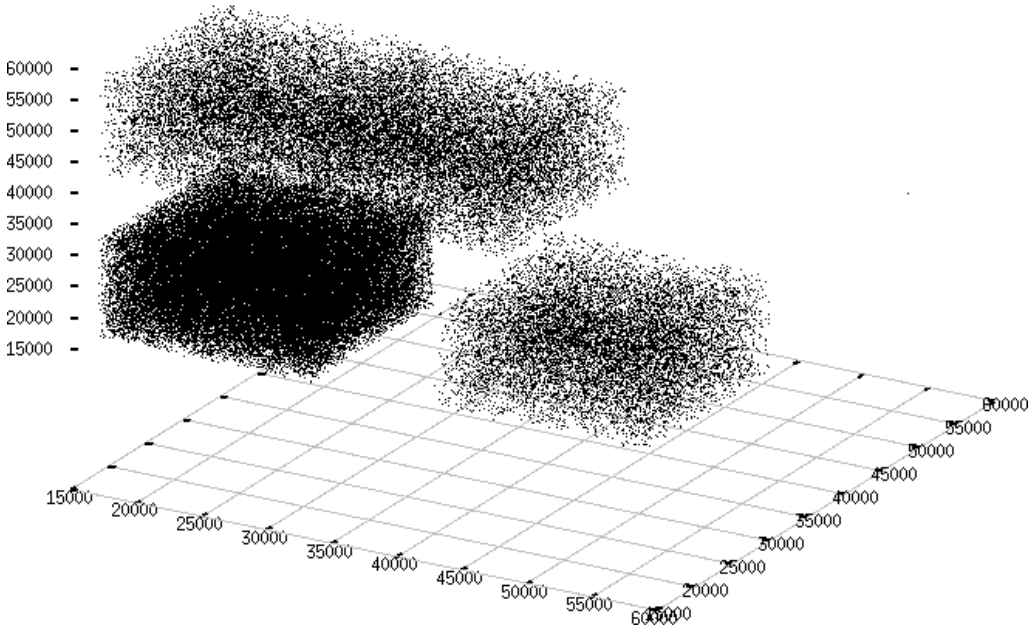

Figura A.33: Atrator do Windows XP.

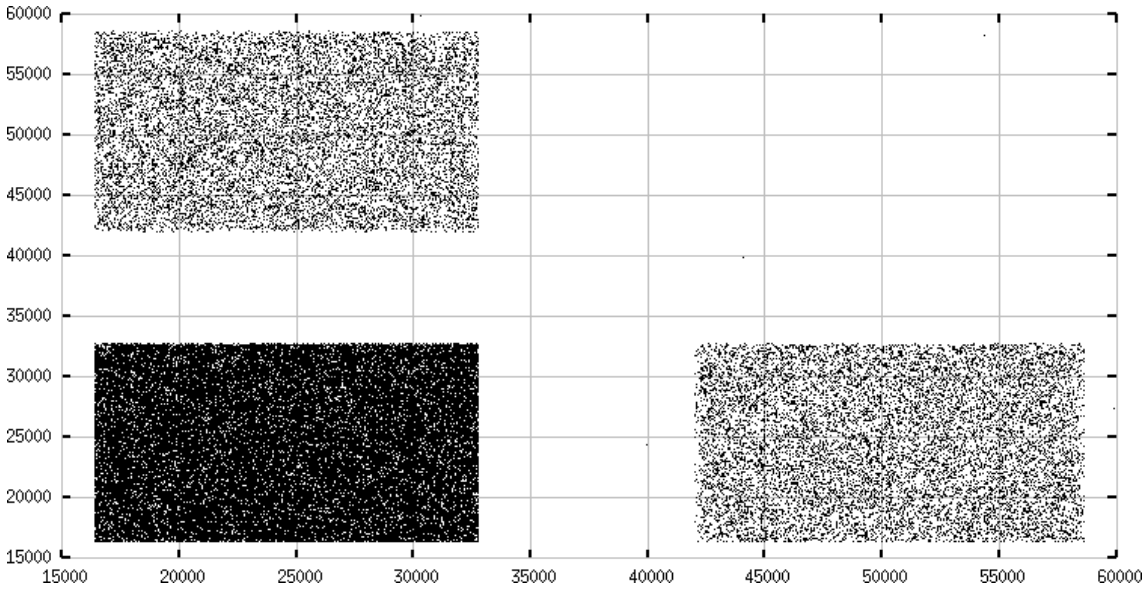

Figura A.34: Plano $\overline{XY}$ do atrator do Windows XP.

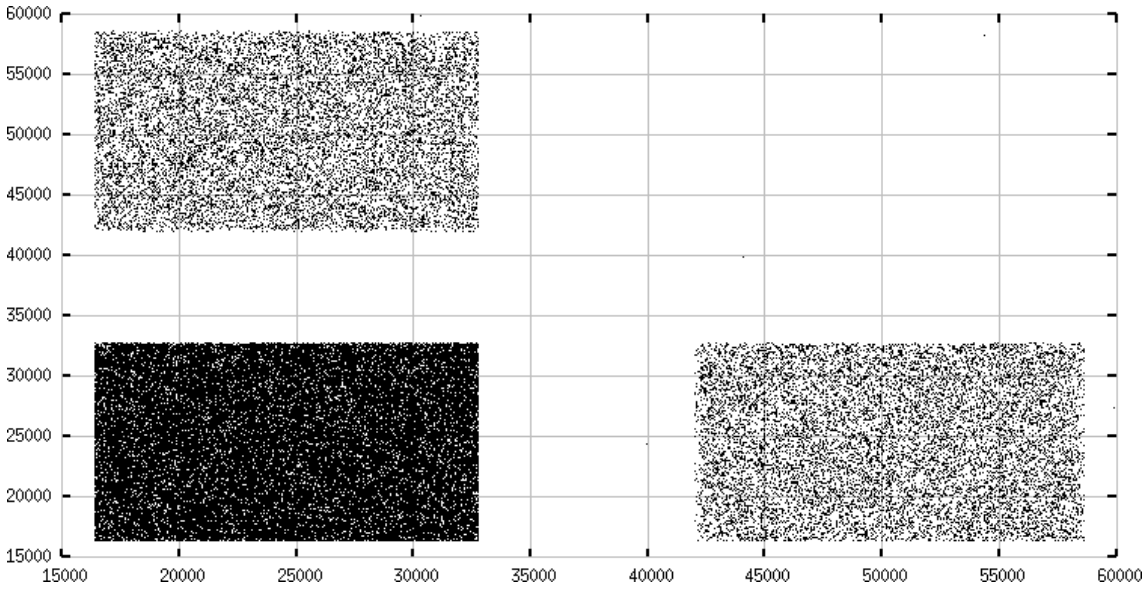

Figura A.35: Plano $\overline{YZ}$ do atrator do Windows XP.

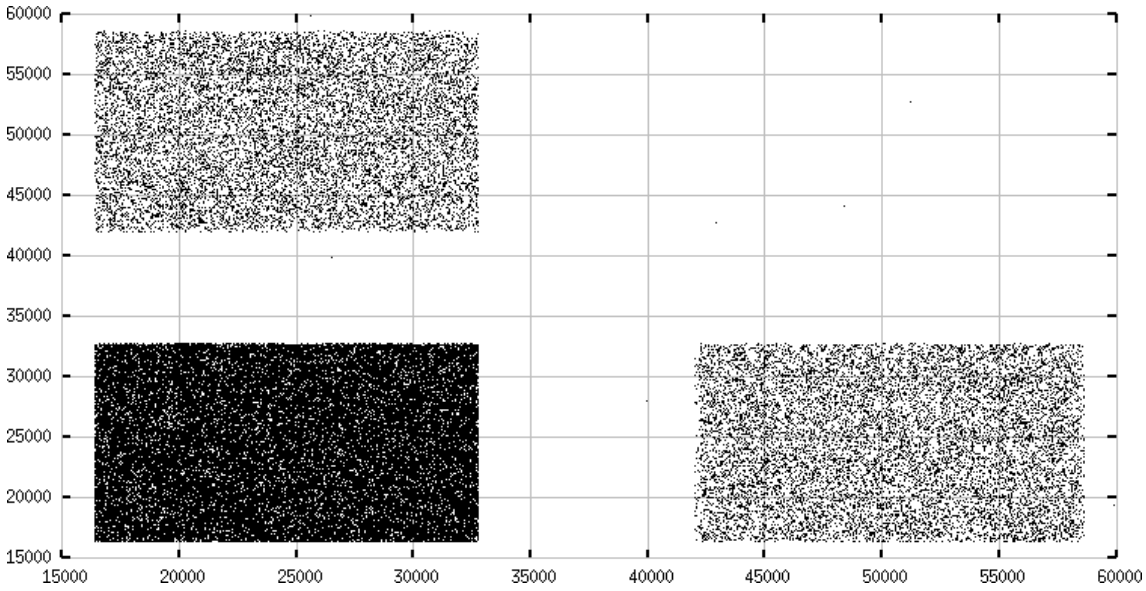

Figura A.36: Plano $\overline{XZ}$ do atrator do Windows XP.

Apêndice B

Desempenho das métricas

Os resultados de classificação apresentados nas Figuras 4.3 e 4.4 do Capítulo 4, são baseados nas curvas de classificação apresentadas neste apêndice. Em cada uma das figuras aqui apresentadas, é exibido o um gráfico que descrimina o valor de cada uma das métricas entre uma representação de atrator construída a partir de 10.000 amostras de TCP ISNs e as representações de 100.000 amostras (case de dados de assinaturas). Estes gráficos consistem na avaliação de cada métrica acerca dos parâmetros α, para à métrica $N_s(A,B,\alpha)$, e β, para as métricas $M_s(A,B,\beta)$. A avaliação é realizada para valores de α e β compreendidos dentro do intervalo $[0,1]$.

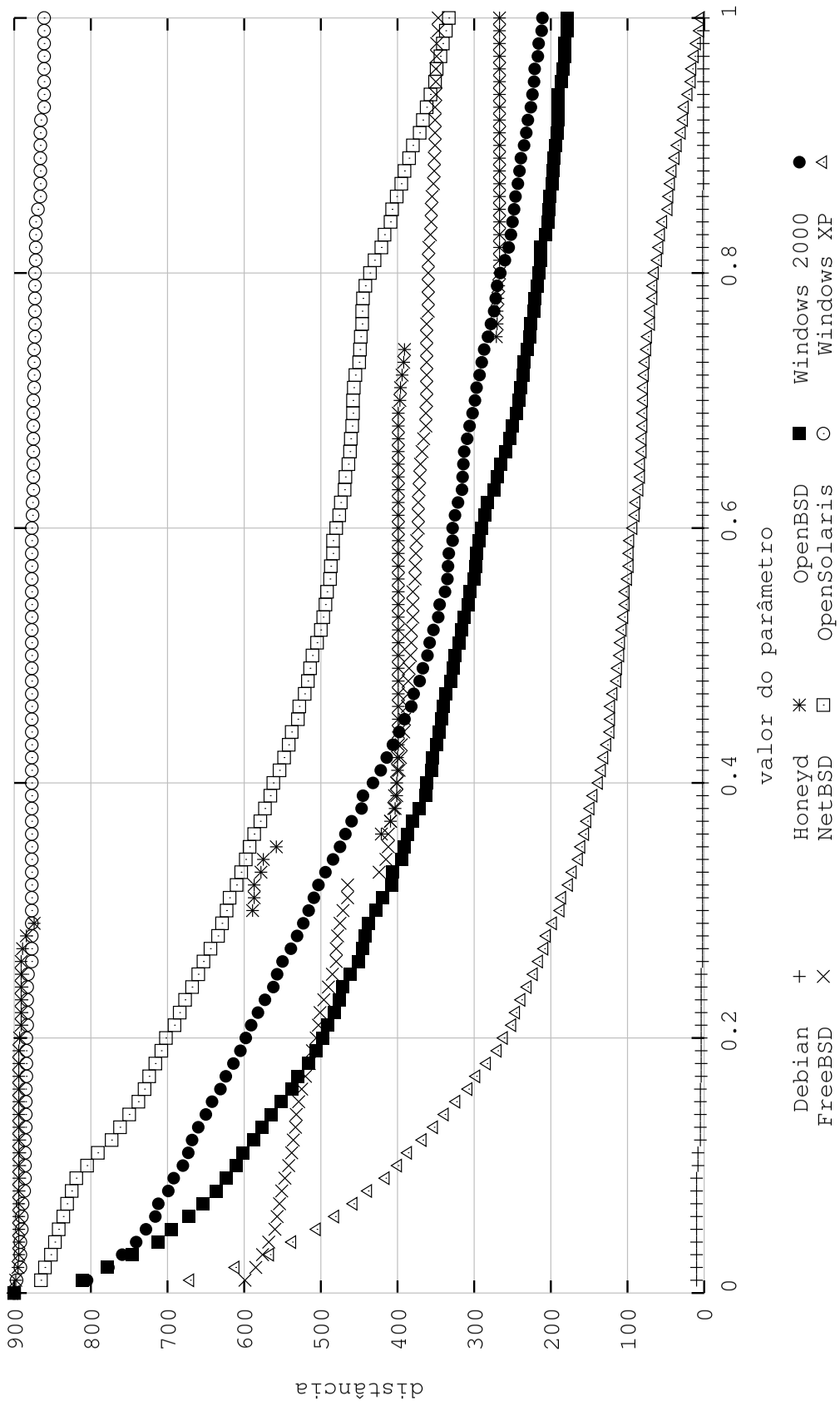

Figura B.1: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do Debian.

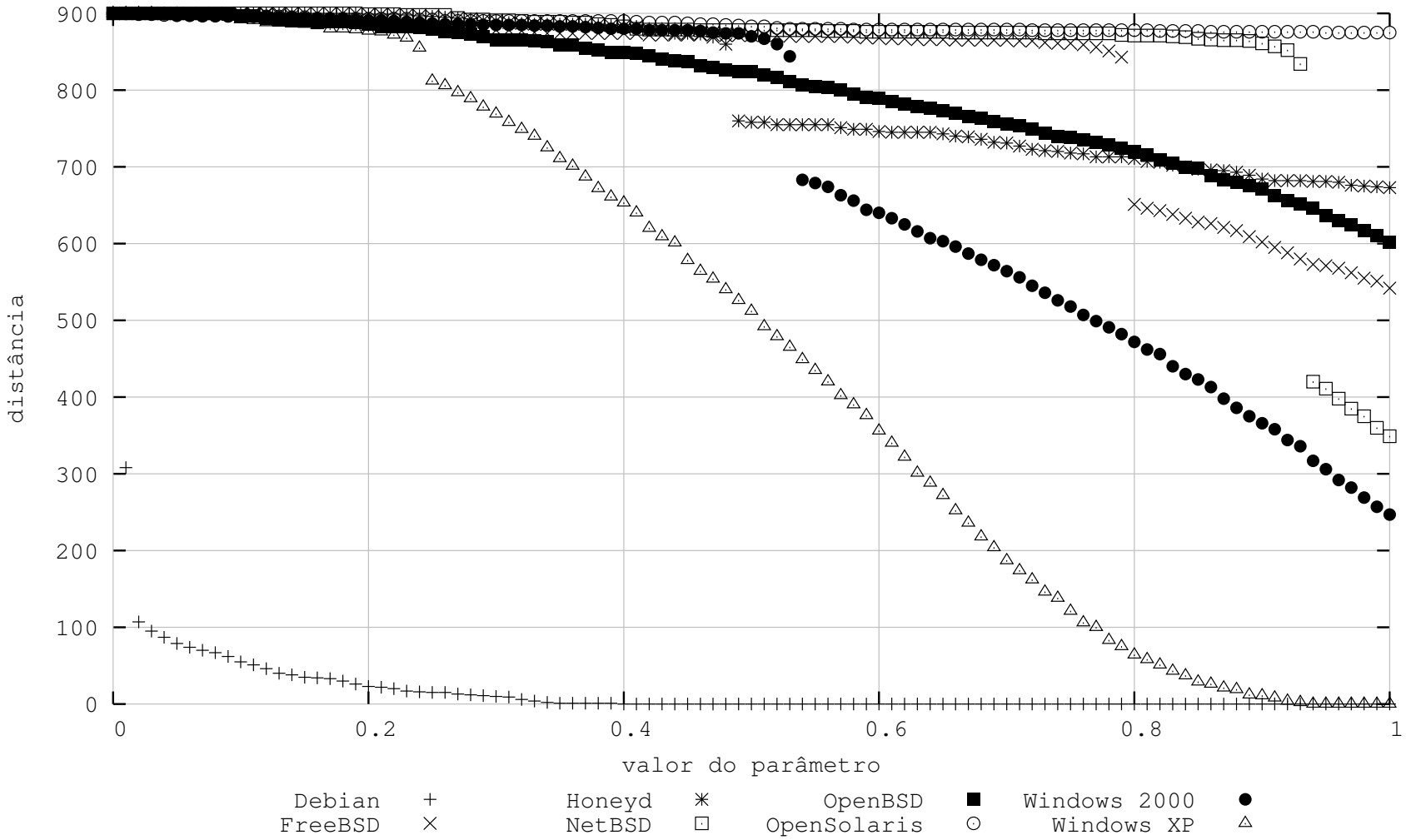

Figura B.2: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do Debian.

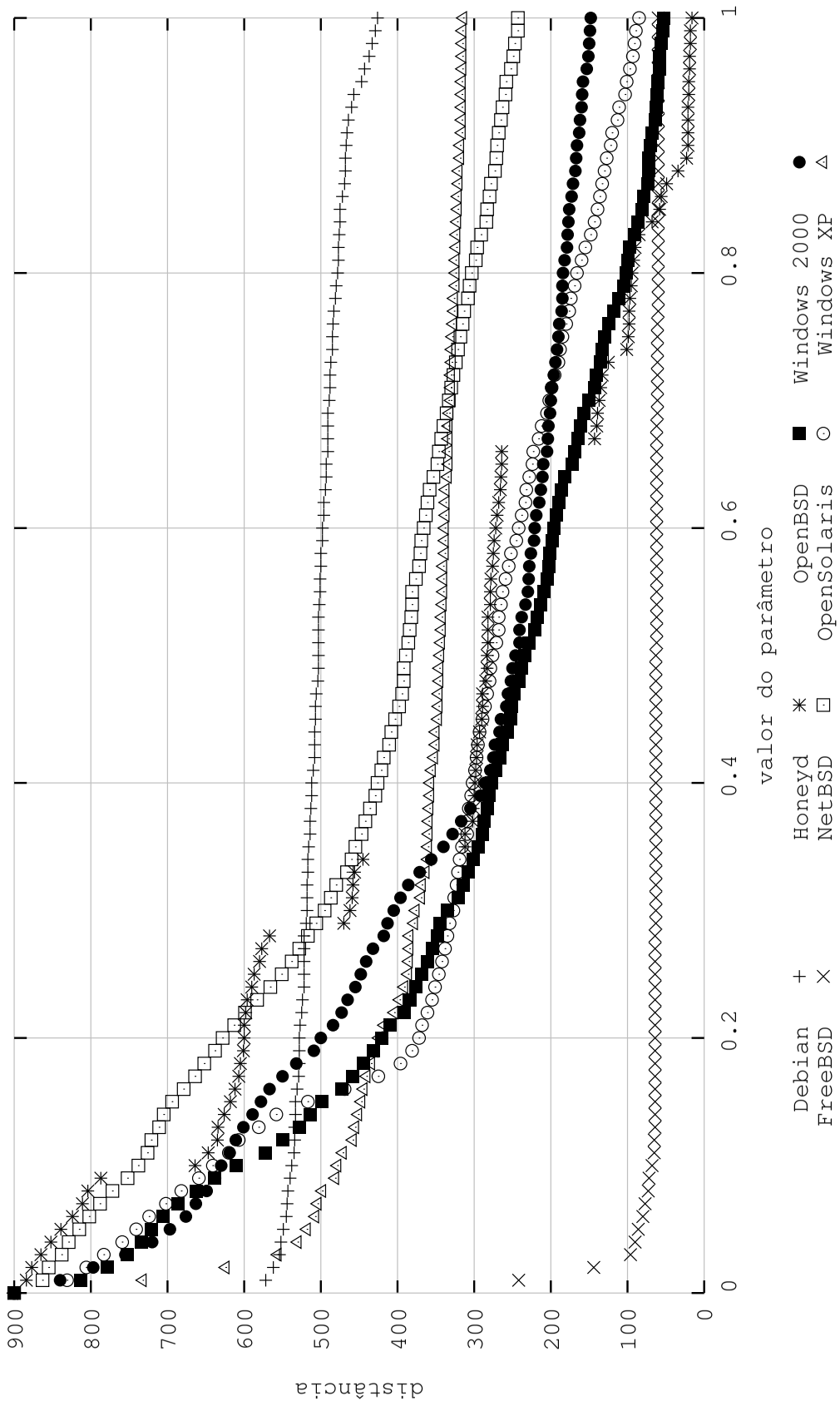

Figura B.3: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do FreeBSD.

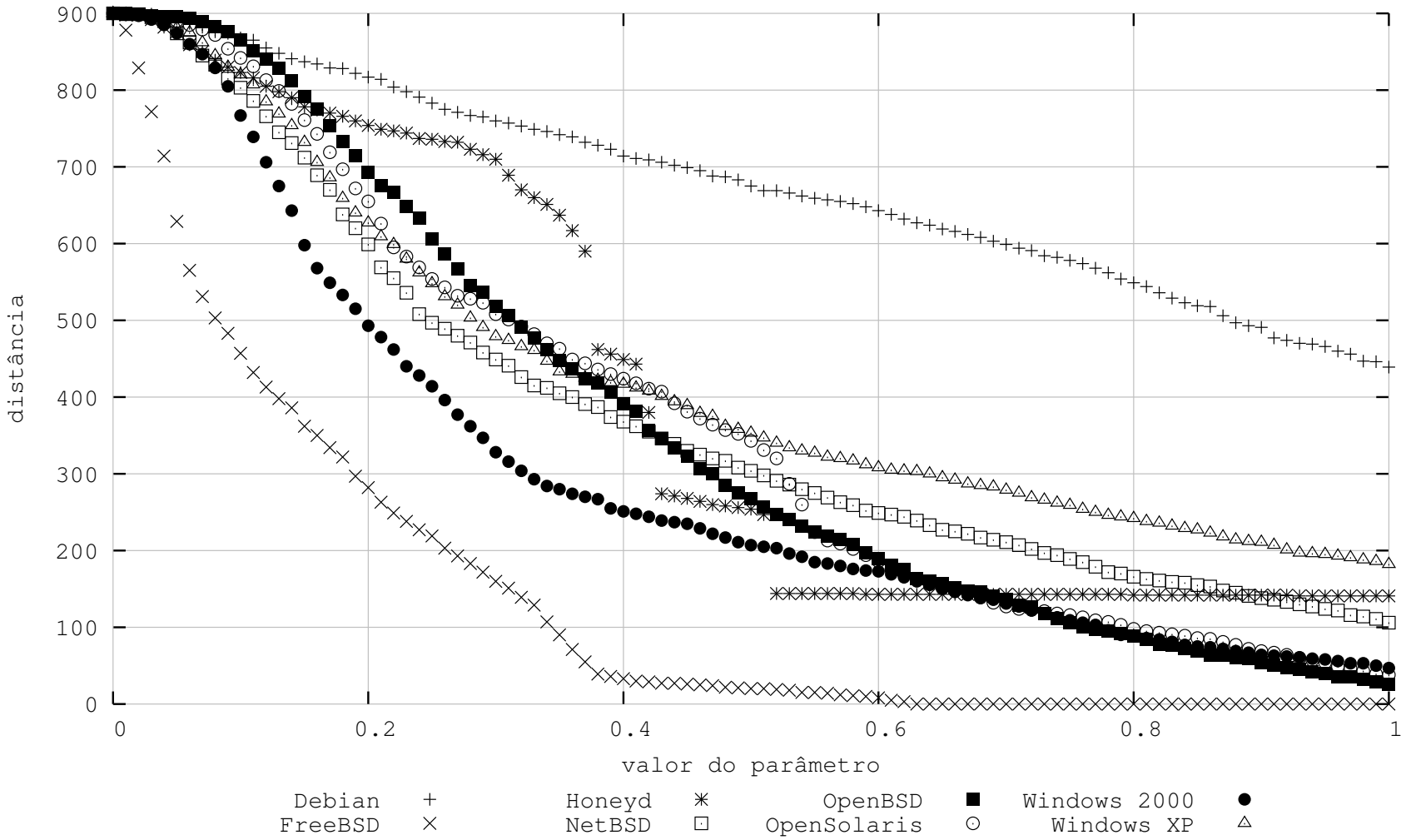

Figura B.4: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do FreeBSD.

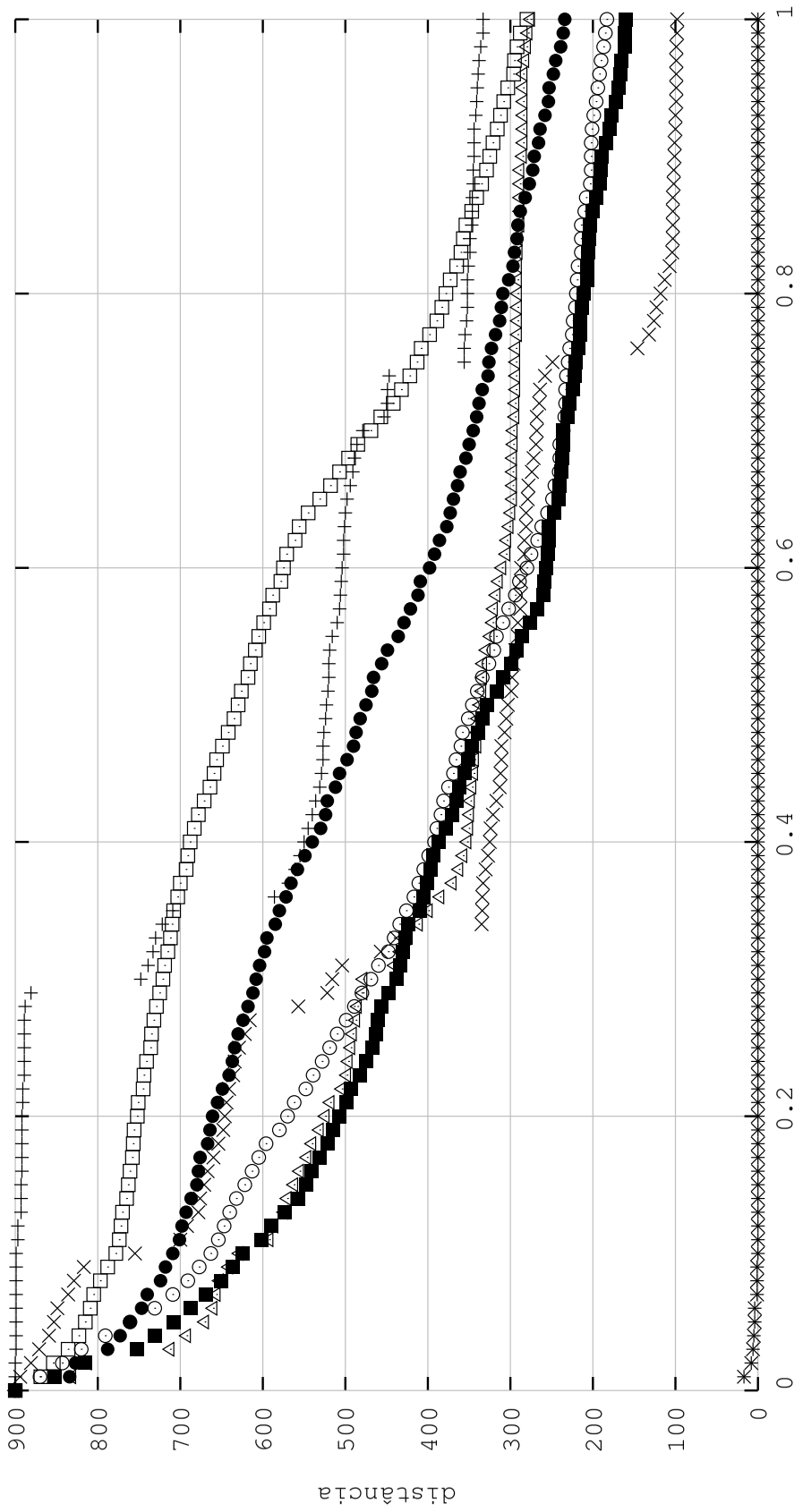

Figura B.5: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do Honeyd.

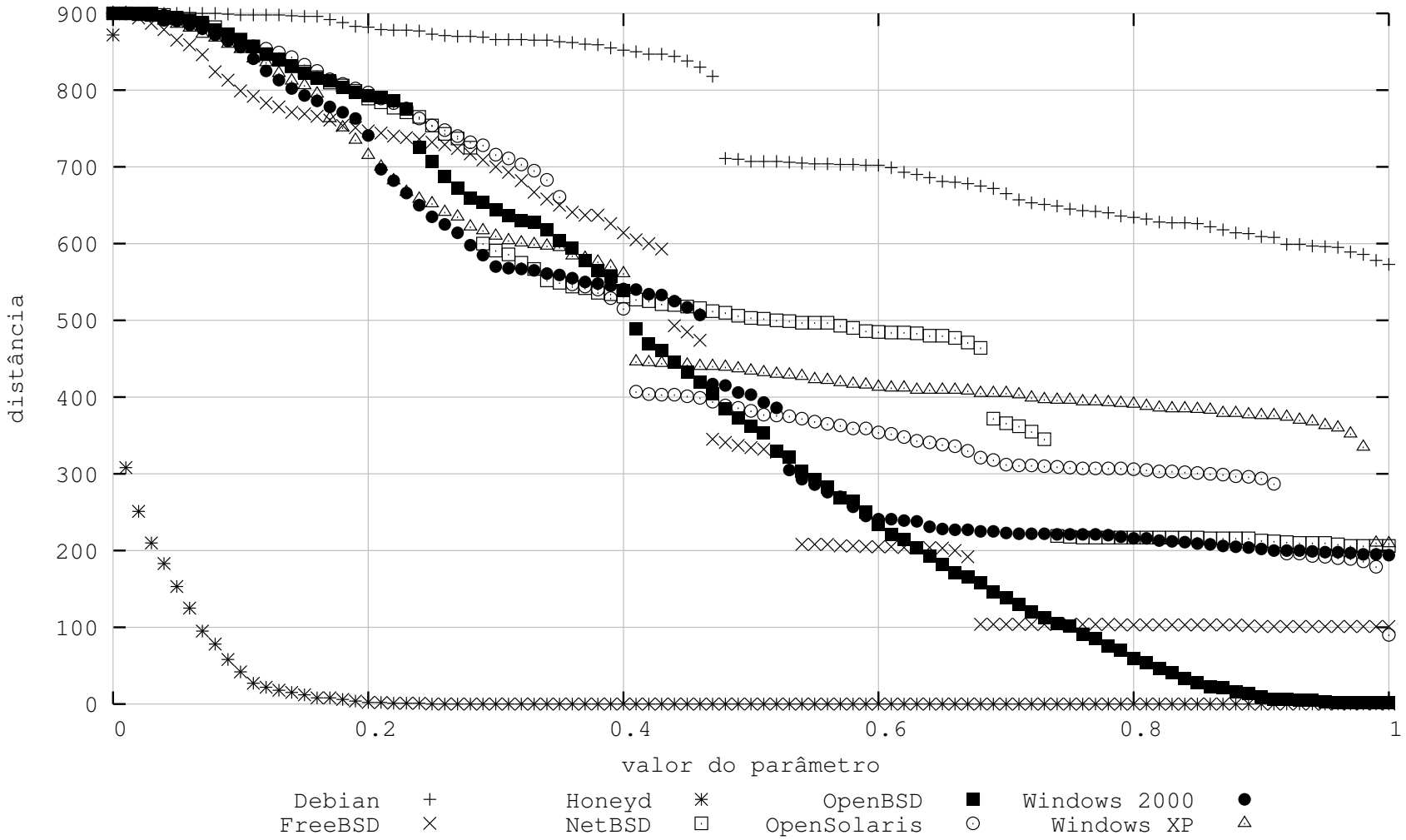

Figura B.6: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do Honeyd.

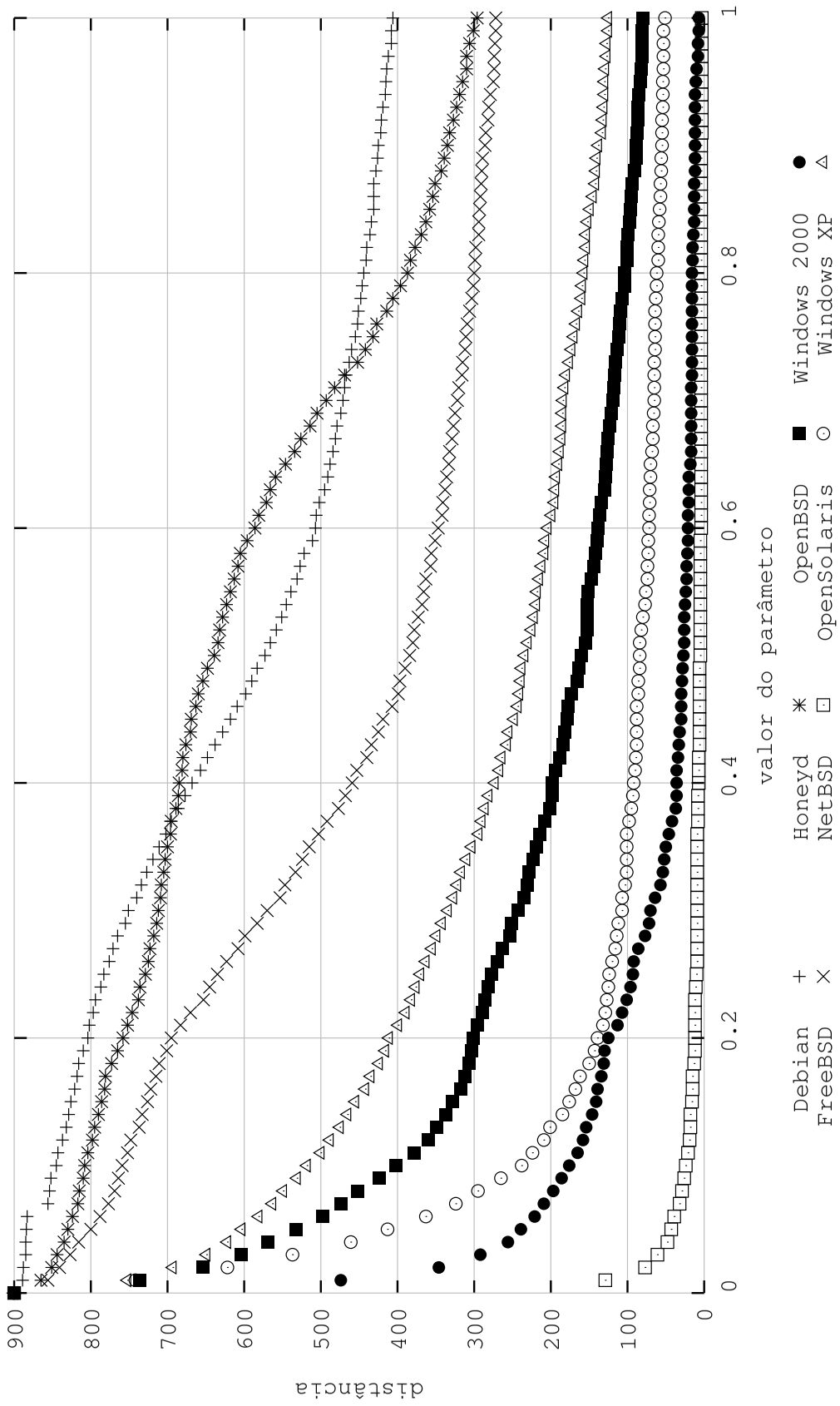

Figura B.7: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do NetBSD.

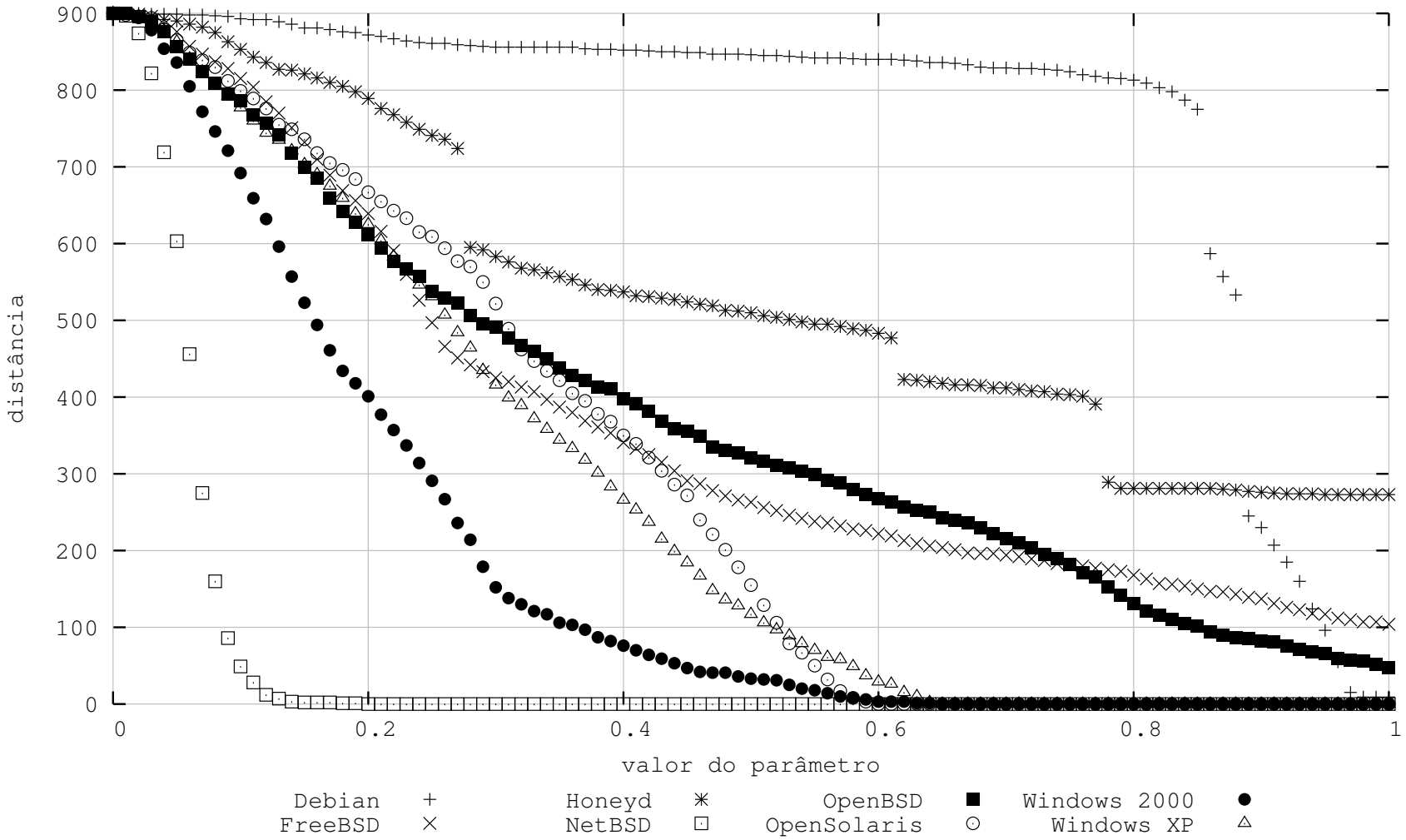

Figura B.8: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do NetBSD.

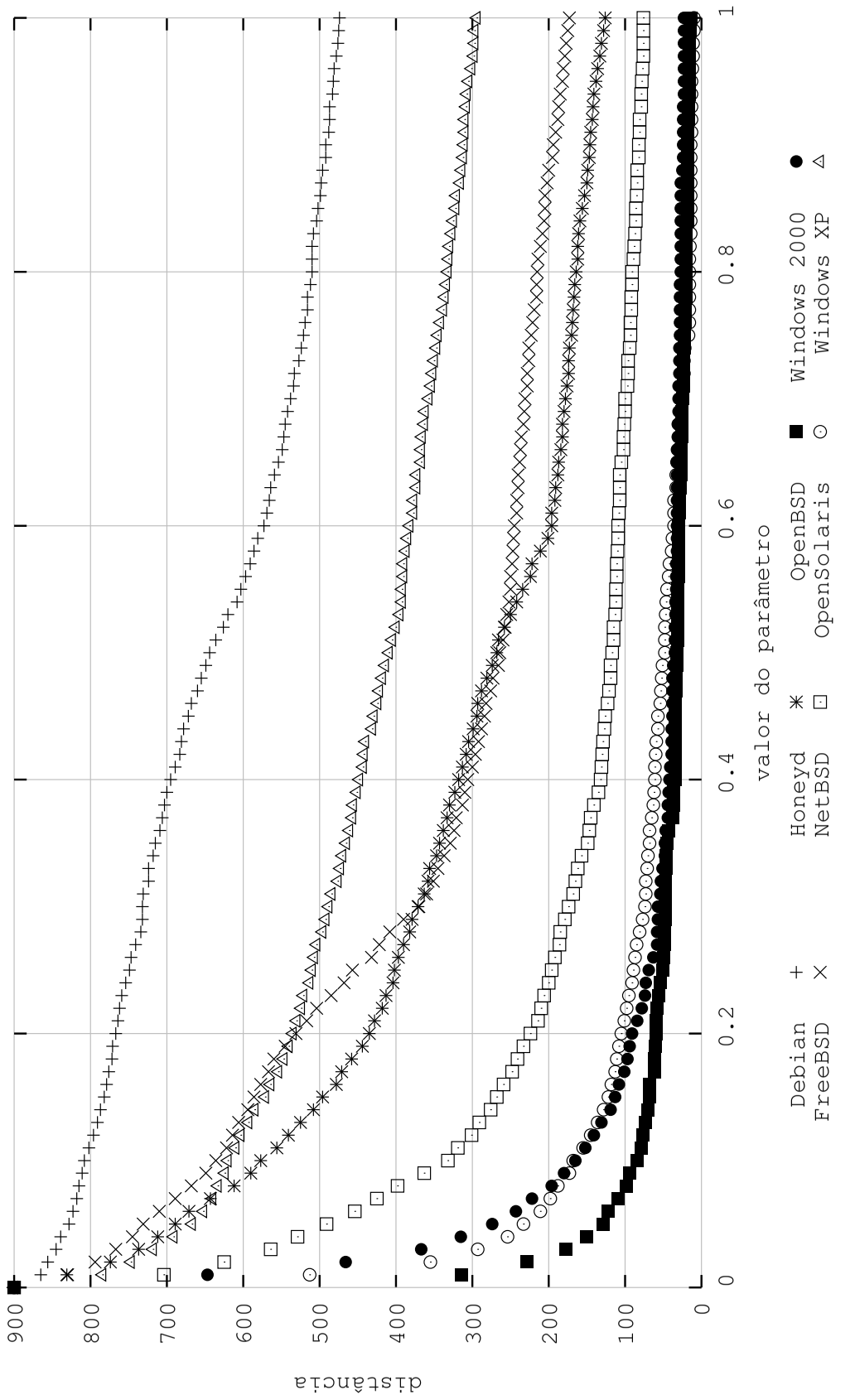

Figura B.9: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do OpenBSD.

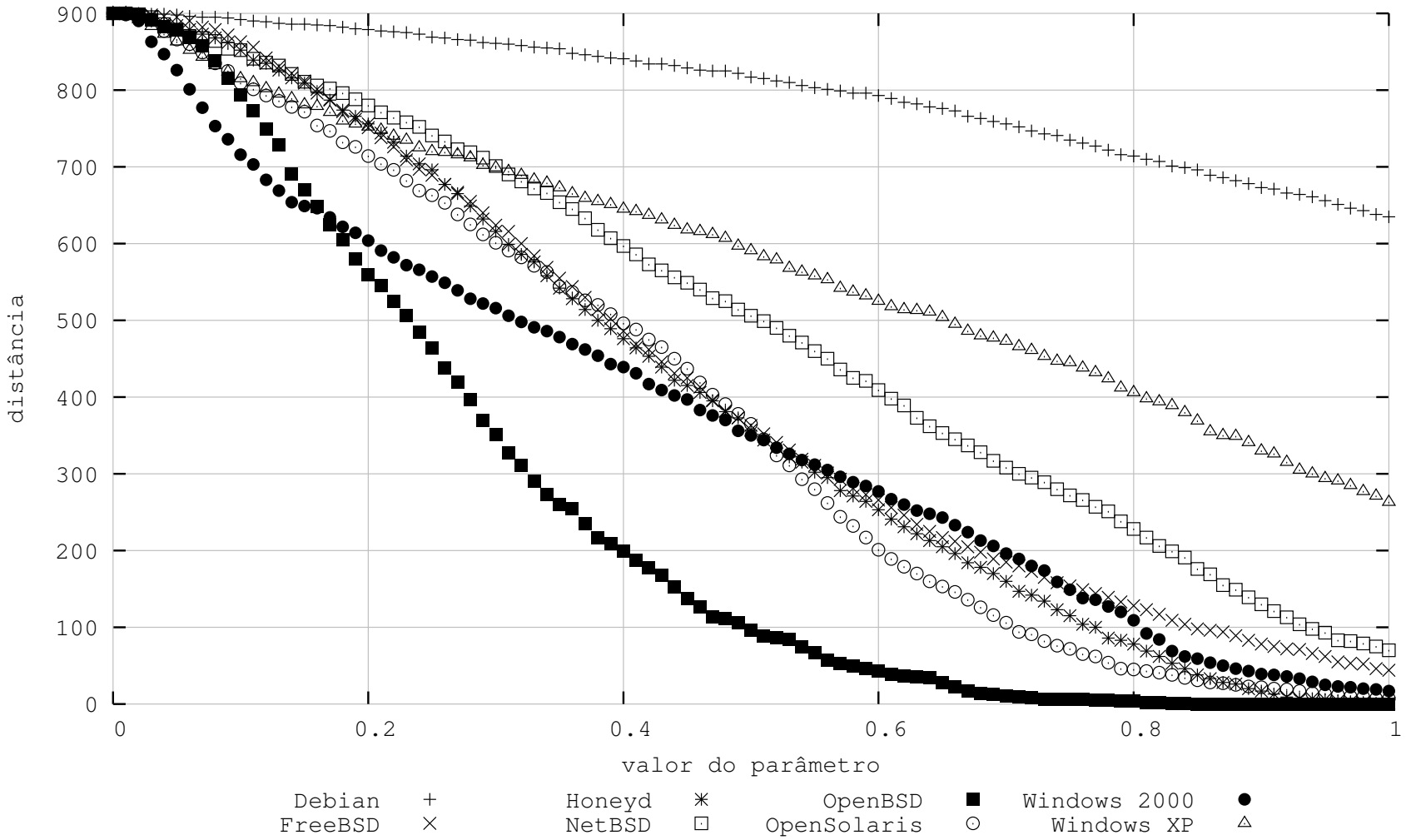

Figura B.10: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do OpenBSD.

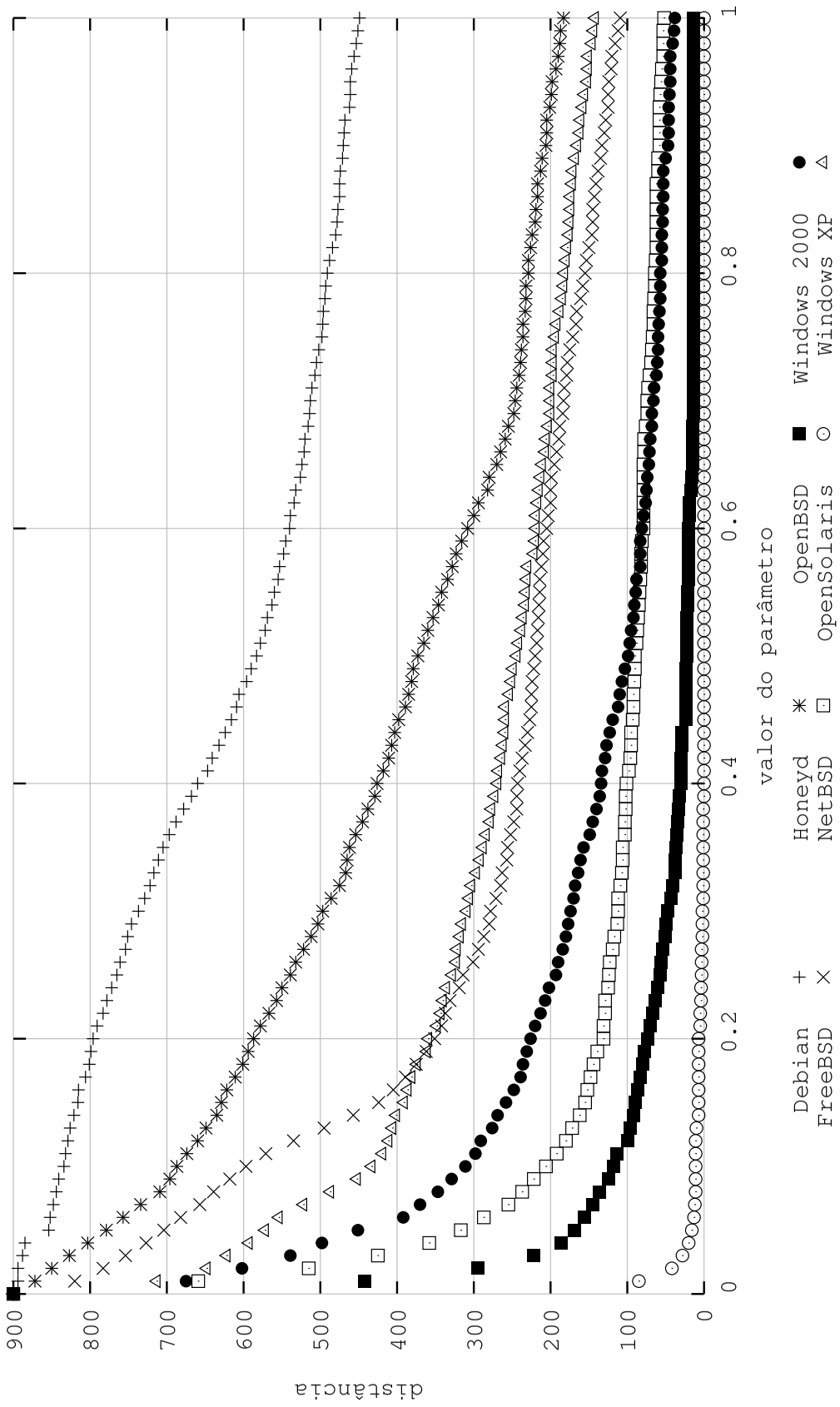

Figura B.11: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do OpenSolaris.

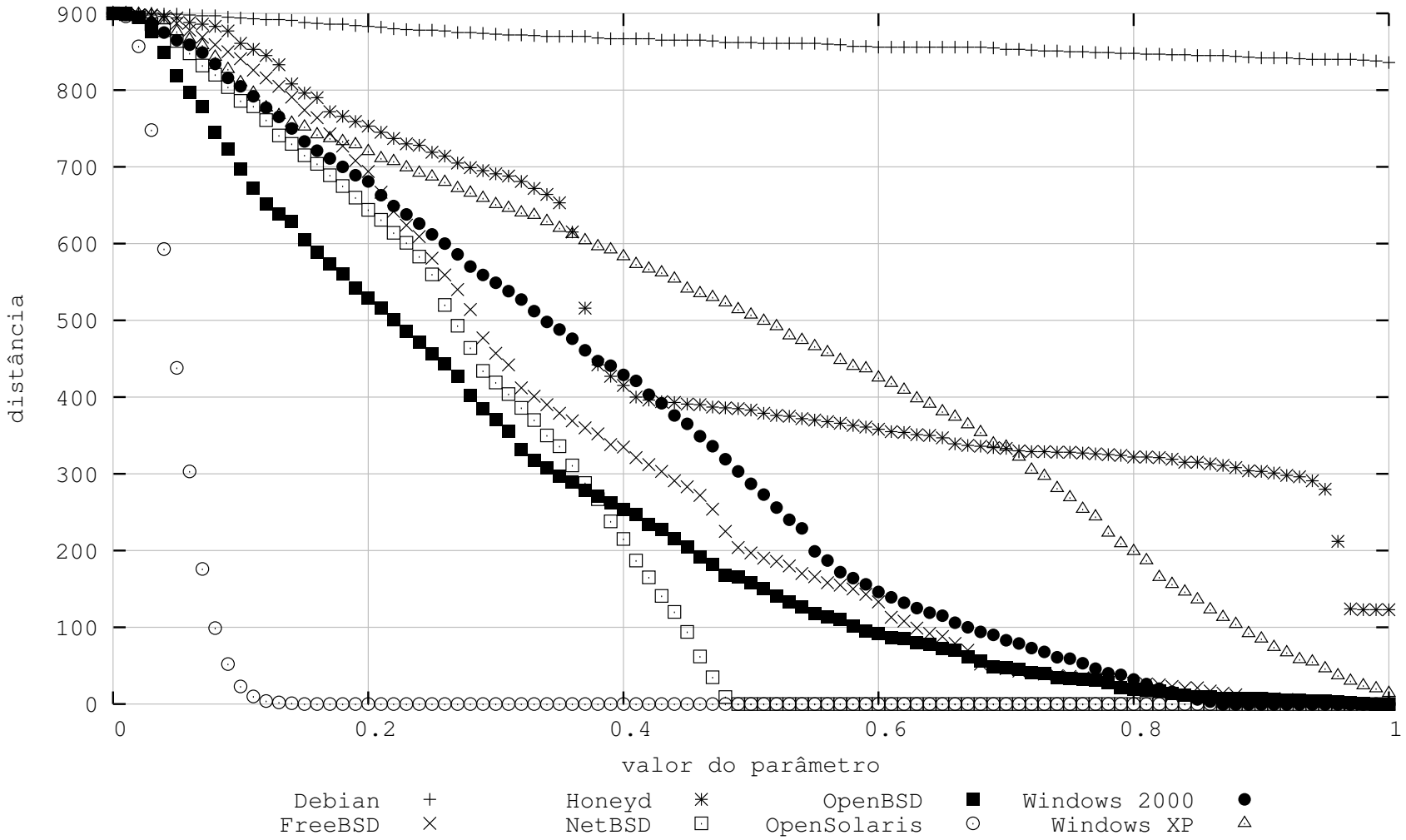

Figura B.12: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do OpenSolaris.

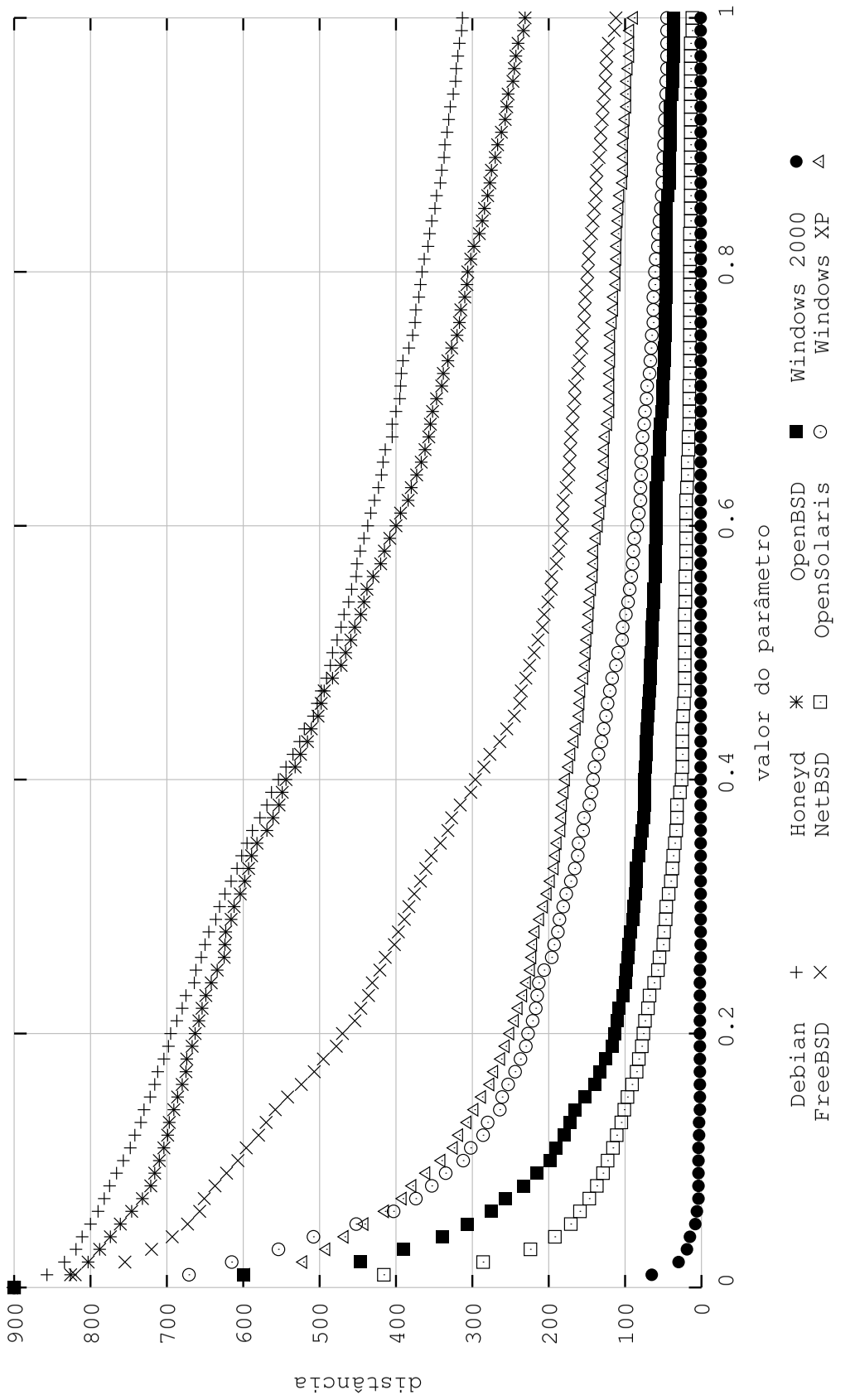

Figura B.13: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do Windows 2000.

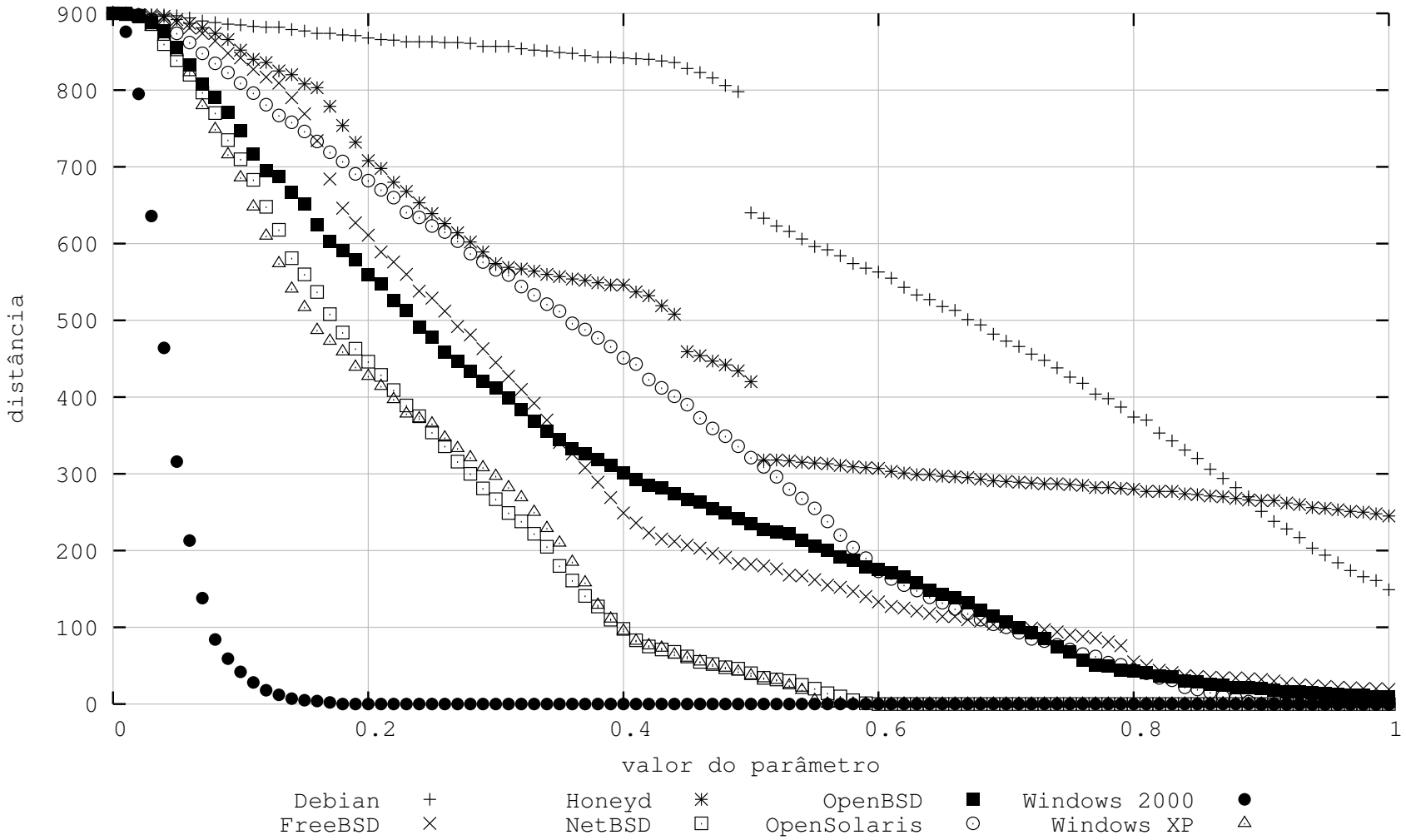

Figura B.14: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do Windows 2000.

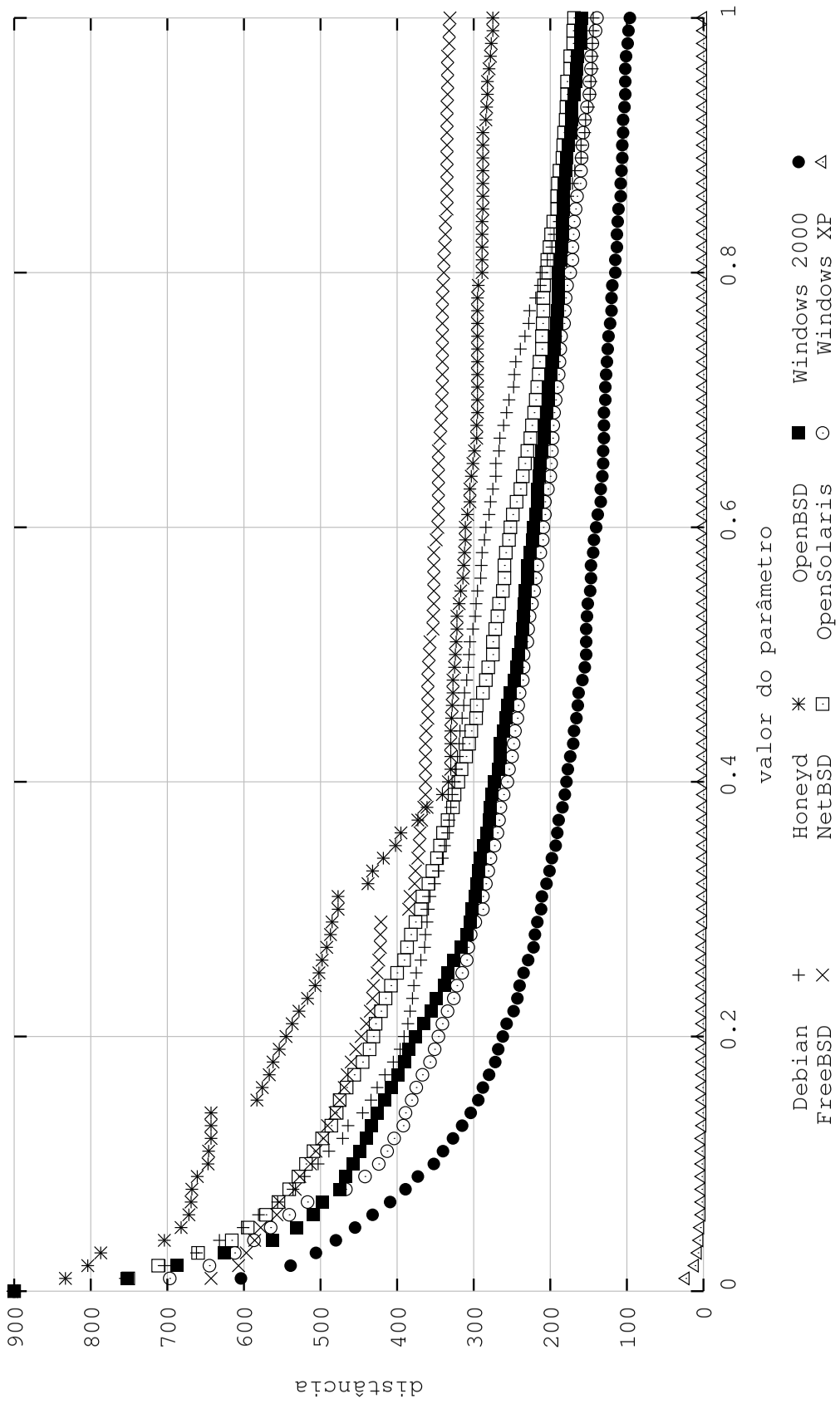

Figura B.15: Gráfico da evolução de $\alpha = [0,1]$ em relação à métrica $N_s(X,Y,\alpha)$ para o atrator de referência do Windows XP.

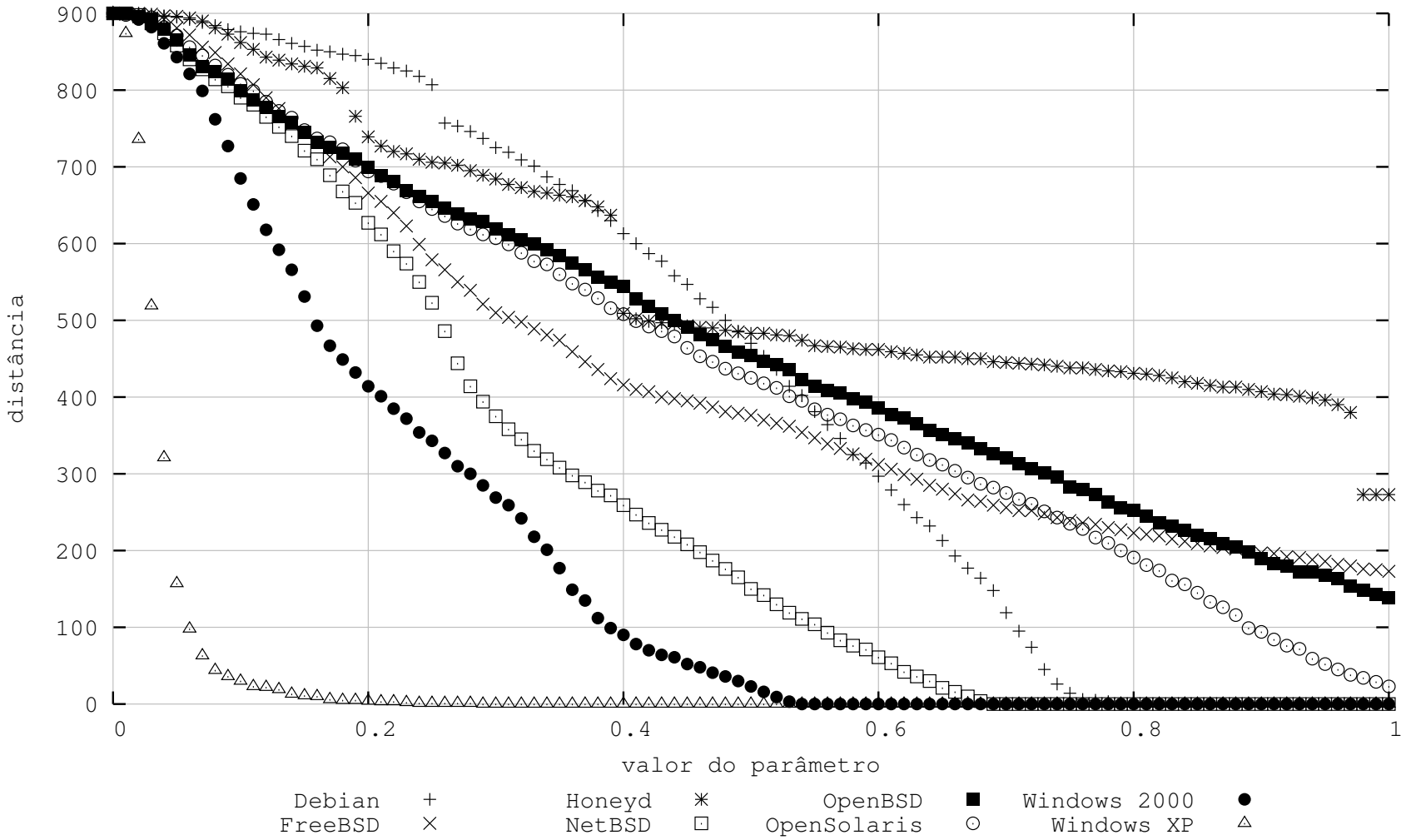

Figura B.16: Gráfico da evolução de $\beta = [0,1]$ em relação à métrica $M_s(X,Y,\beta)$ para o atrator de referência do Windows XP.

Referências Bibliográficas

Albano, AM, J. Muench, C. Schwartz, AI Mees & PE Rapp (1988), 'Singular-value decomposition and the Grassberger-Procaccia algorithm', *Physical Review A* **38**(6), 3017–3026.

Alligood, K.T., T. Sauer & J.A. Yorke (1997), *Chaos: An Introduction to Dynamical Systems*, Springer.

Arkin, Ofir (2002), 'A remote active OS fingerprinting tool using ICMP', *;login:* **27**(2). Visitado em Junho de 2009.
URL: *http://sys-security.com/xprobe/*

Arkin, Ofir & Fyodor Yarochkin (2002), XProbe2 – A 'Fuzzy' Approach to Remote Active Operating System Fingerprinting, Relatório técnico. Visitado em Junho de 2009.
URL: *http://sys-security.com/xprobe/*

Arkin, Ofir, Fyodor Yarochkin & Meder Kydyraliev (2003), The Present and Future of Xprobe2 -– The Next Generation of Active Operating System Fingerprinting, Relatório técnico. Visitado em Junho de 2009.
URL: *http://sys-security.com/xprobe/*

Auffret, Patrice (2006), 'SinFP'. Version 2.06. Visitado em Junho de 2009.
URL: *http://www.gomor.org/bin/view/Sinfp*

Auffret, Patrice (2008), SinFP, unification de la prise d'empreinte active et passive des systèmes d'exploitation, *em* 'Proc. Symposium sur La Sécurité des Technologies de L'Information et des Communications'.
URL: *http://www.gomor.org/bin/view/GomorOrg/ConfSstic2008*

Baker, G.L. & J.P. Gollub (1996), *Chaotic Dynamics: An Introduction*, 2ª edição, Cambridge University Press.

Bellovin, S. (1996), 'Defending Against Sequence Number Attacks', RFC 1948 (Informational). Obsoletes RFC 765. Updated by RFCs 2228, 2640, 2773 and 3659. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc1948.txt*

Broder, A. (1993), Some applications of Rabin's fingerprinting method, *em* 'Sequences II: Methods in Communications, Security, and Computer Science', pp. 143–152.

Deza, E. & M. M. Deza (2006), *Dictionary of Distances*, Elsevier Science.

Eastlake, D. & A. Panitz (1999), 'Reserved Top Level DNS Names', RFC 2606 (Best Current Practice). Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc2606.txt*

Eddy, W. (2007), 'TCP SYN Flooding Attacks and Common Mitigations', RFC 4987 (Informational). Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc4987.txt*

Egevang, K. & P. Francis (1994), 'The IP Network Address Translator (NAT)', RFC 1631 (Informational). Obsoleted by RFC 3022. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc1631.txt*

Fielding, Roy T., James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen & Larry Masinter (1999), 'Hypertext Transfer Protocol – HTTP/1.1', RFC 2616 (Draft Standard). Obsoletes RFC 2068. Updated by RFC 2817. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc2616.txt*

Fraser, AM (1989), 'Information and entropy in strange attractors', *IEEE transactions on Information Theory* **35**(2), 245–262.

Fraser, A.M. & H.L. Swinney (1986), 'Independent coordinates for strange attractors from mutual information', *Physical Review A* **33**(2), 1134–1140.

Free Software Foundation, Inc. (1996), 'The Free Software Definition'. Visitado em Junho de 2009.
URL: *http://www.gnu.org/philosophy/free-sw.html*

Fritzke, Bernd (1993), 'Growing Cell Structures – A Self-organizing Network for Unsupervised and Supervised Learning', *Neural Networks* **7**, 1441–1460.

Fritzke, Bernd (1995), A Growing Neural Gas Network Learns Topologies, *em* 'Advances in Neural Information Processing Systems', Vol. 7, MIT Press, pp. 625–632.

Fuller, V. & T. Li (2006), 'Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan', RFC 4632 (Best Current Practice). Obsoletes RFC 1519. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc4632.txt*

Fyodor (1997), 'The art of port scanning', *Phrack Magazine* **7**.

Fyodor (1998), 'Remote OS Detection via TCP/IP Fingerprinting', *Phrack Magazine* **8**.

Fyodor (2006), Nmap Remote OS Detection via TCP/IP Fingerprinting (2nd Generation), Relatório técnico, Insecure.org. Visitado em Junho de 2009.
URL: *http://insecure.org/nmap/osdetect*

Fyodor (2009*a*), 'Nmap'. Version 4.85BETA8. Visitado em Junho de 2009.
URL: *http://www.nmap.org/*

Fyodor (2009*b*), *Nmap Network Scanning*, Insecure.Com LLC.

Grassberger, P. & I. Procaccia (1983), 'Characterization of strange attractors', *Phys. Rev. Lett.* **50**.

Grassberger, P. & I. Procaccia (2004), 'Measuring the strangeness of strange attractors', *The Theory of Chaotic Attractors* p. 170.

Handley, M., V. Paxson & C. Kreibich (2001), Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics, *em* 'Proceedings of the 10th USENIX Security Symposium'.

Haykin, S. (1999), *Neural Networks: A Comprehensive Foundation*, Prentice Hall.

Hebb, Donald Olding (1949), *The Organization of Behavior: A Neuropsychological Theory*, Wiley.

IANA (2002), 'Special-Use IPv4 Addresses', RFC 3330 (Informational). Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc3330.txt*

Knight, Joel (2004), 'PF Example – Firewall for Home or Small Office'. Visitado em Abril de 2009.
URL: *http://www.openbsd.org/faq/pf/example1.html*

Kohonen, Teuvo (1982), 'Self-organized formation of topologically correct feature maps', *Biological Cybernetics* **43**(1), 59–69.

Kohonen, Teuvo (2001), *Self-Organizing Maps*, 3ª edição, Springer.

Mockapetris, P.V. (1987), 'Domain Names - Concepts and Facilities', RFC 1034 (Standard). Obsoletes RFC 973, 882, 883. Updated by RFC RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC2065, RFC2181, RFC2308, RFC2535, RFC4033, RFC4034, RFC4035, RFC4343, RFC4035, RFC4592. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc1034.txt*

Nmap Hackers Mailing List (2008), 'Top 2 OS detection tools'.
URL: *http://sectools.org/os-detectors.html*

OpenBSD (2008). Versão 4.4. Visitado em Junho de 2009.
URL: *http://www.openbsd.org/*

OpenBSD PF (2009), 'PF – The OpenBSD Packet Filter'. OpenBSD version 4.4.
URL: *http://www.openbsd.org/faq/pf/*

Ott, Edward (2002), *Chaos in Dynamical Systems*, 2ª edição, Cambridge University Press.

Postel, J. (1980), 'User Datagram Protocol', RFC 768 (Standard). Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc768.txt*

Postel, J. (1981*a*), 'Internet Control Message Protocol', RFC 792 (Standard). Obsoletes RFC 777. Updated by RFCs 950, 4884. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc792.txt*

Postel, J. (1981*b*), 'Internet Protocol', RFC 791 (Standard). Obsoletes RFC 760. Updated by RFC 1349. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc791.txt*

Postel, J. (1981*c*), 'Transmission Control Protocol', RFC 793 (Standard). Updated by RFCs 1122, 3168. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc793.txt*

Postel, J. & J. K. Reynolds (1983), 'Telnet Protocol Specification', RFC 854 (Standard). Obsoletes RFC 764. Updated by RFC 5198. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc854.txt*

Postel, J. & J. Reynolds (1985), 'File Transfer Protocol', RFC 959 (Standard). Obsoletes RFC 765. Updated by RFCs 2228, 2640, 2773 and 3659. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc959.txt*

Provos, Niels (2007), 'Honeyd'. Version 1.5c. Visitado em Junho de 2009.
URL: *http://www.honeyd.org/*

Provos, Niels & Thorsten Holz (2008), *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*, Addison-Wesley.

Smart, M., G.R. Malan & F. Jahanian (2000), Defeating TCP/IP stack fingerprinting, *em* 'Proceedings of the 9th USENIX Security Symposium'.

Srisuresh, P. & K. Egevang (2001), 'Traditional IP Network Address Translator (Traditional NAT)', RFC 3022 (Informational). Obsoletes RFC 1631. Visitado em Junho de 2009.
URL: *http://www.ietf.org/rfc/rfc3022.txt*

Stevens, W.R. (1993), *TCP/IP Illustrated: The Protocols*, Vol. 1, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Yarochkin, Fyodor, Meder Kydyraliev & Ofir Arkin (2005), 'Xprobe2'. Version 0.3. Visitado em Junho de 2009.
URL: *http://sys-security.com/xprobe/*

Zalewski, Michal (2001), Strange Attractors and TCP/IP Sequence Number Analysis, Relatório técnico. Visitado em Junho de 2009.
URL: *http://lcamtuf.coredump.cx/oldtcp/tcpseq.html*

Zalewski, Michal (2002), Strange Attractors and TCP/IP Sequence Number Analysis – One Year Later, Relatório técnico. Visitado em Junho de 2009.
URL: *http://lcamtuf.coredump.cx/newtcp/*

Publicações e premiações

Congressos

MEDEIROS, João Paulo de Souza; CUNHA, Allison C.; BRITO JR., Agostinho M.; PIRES, Paulo S. Motta. Automating Security Tests for Industrial Automation Devices Using Neural Networks. Proceedings of the 12th Conference on Emerging Technology and Factory Automation (ETFA), 2007.

MEDEIROS, João Paulo de Souza; BRITO JR., Agostinho M.; PIRES, Paulo S. Motta. A New Method for Recognizing Operating Systems of Automation Devices. 14th Conference on Emerging Technology and Factory Automation (ETFA), 2009.

Revista

MEDEIROS, João Paulo de Souza; CUNHA, Allison C.; BRITO JR., Agostinho M.; PIRES, Paulo S. Motta. Application of Kohonen Maps to Improve Security Tests on Automation Devices. 2nd International Workshop on Critical Information Infrastructures Security (CRITIS), Lecture Notes in Computer Science, 2007.

MEDEIROS, João Paulo de Souza; SANTOS, Selan Rodrigues. RadialNet: An Interactive Network Topology Visualization Tool with Visual Auditing Support. 3rd International Workshop on Critical Information Infrastructures Security (CRITIS), Lecture Notes in Computer Science, 2008.

MEDEIROS, João Paulo de Souza; SANTOS, Selan Rodrigues; BRITO JR., Agostinho M.; PIRES, Paulo S. Motta. Advances in Network Topology Security Visualization. International Journal of System of Systems Engineering (IJSSE), 2010 (data prevista para publicação).

MEDEIROS, João Paulo de Souza; BRITO JR., Agostinho M.; PIRES, Paulo S. Motta. An Effective TCP/IP Fingerprinting Technique Based on Strange Attractors Classification. 2nd International Workshop on Autonomous and Spontaneous Security (SETOP), Lecture Notes in Computer Science, 2009.

Capítulo de Livro

MEDEIROS, João Paulo de Souza; BRITO JR., Agostinho M.; PIRES, Paulo S. Motta. An Data Mining Based Analysis of Nmap Operating System Fingerprint Database. 2nd International Workshop on Computational Intelligence in Security for Information Systems (CISIS), Advances in Intelligent and Soft Computing Series, Volume 63, ISBN 978-3-642-04090-0, 2009.

Premiações

Projeto de desenvolvimento contemplado com uma bolsa ofertada pelo programa *Google Summer of Code* (GSoC) em Abril de 2009. Trabalho intitulado "*New OS fingerprinting Tool and RadialNet improvements*" foi aceito pelas organizações Umit e Nmap.

Índice Remissivo