

**UNIVERSIDADE DE BRASÍLIA  
FACULDADE DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**IMPLEMENTAÇÃO DE UM SISTEMA DE DETECÇÃO DE  
INTRUSÃO POR ANOMALIA COMPORTAMENTAL PARA  
REDES MÓVEIS AD-HOC**

**ALAN SALLES VIEIRA PINTO  
FAUSTO BARROS DE SÁ TELES**

**ORIENTADOR: RICARDO STACIARINI PUTTINI**

**PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA  
ELÉTRICA**

**PUBLICAÇÃO: XXX/2007**

**BRASÍLIA / DF: JUNHO/2007**

**UNIVERSIDADE DE BRASÍLIA  
FACULDADE DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**IMPLEMENTAÇÃO DE UM SISTEMA DE DETECÇÃO DE  
INTRUSÃO POR ANOMALIA COMPORTAMENTAL PARA  
REDES MÓVEIS AD-HOC**

**ALAN SALLES VIEIRA PINTO  
FAUSTO BARROS DE SÁ TELES**

PROJETO FINAL DE GRADUAÇÃO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO.

APROVADA POR:

---

**RICARDO STACIARINI PUTTINI, Doutor, ENE-UnB  
(ORIENTADOR)**

---

**ANDERSON CLAYTON ALVES NASCIMENTO, Ph.D. , ENE-UnB  
(EXAMINADOR INTERNO)**

---

**JACIR LUIZ BORDIM, Doutor, CIC-UnB  
(EXAMINADOR EXTERNO)**

**DATA: BRASÍLIA/DF, 26 DE JUNHO DE 2007.**



## FICHA CATALOGRÁFICA

PINTO, ALAN SALLES VIEIRA; SÁ TELES, FAUSTO BARROS. Uma implementação em Java em um ambiente real para validação de um sistema de detecção de intrusão por anomalia comportamental para redes ad-hoc [Distrito Federal] 2007. xxi, 92p., 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2007).

Projeto Final de Graduação – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. IDS 2. Ad-Hoc

3. Segurança

I. ENE/FT/UnB. II. Título (Série)

## REFERÊNCIA BIBLIOGRÁFICA

PINTO, ALAN SALLES VIEIRA; SÁ TELES, FAUSTO BARROS (2007). Uma implementação em Java em um ambiente real para validação de um sistema de detecção de intrusão por anomalia comportamental para redes móveis ad-hoc. Projeto Final de Graduação, Publicação XXX/2007, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília , DF, 92p.

## CESSÃO DE DIREITOS

NOME DO AUTOR: Alan Salles Vieira Pinto; Fausto Barros de Sá Teles.

TÍTULO DA DISSERTAÇÃO: Uma implementação em Java em um ambiente real para validação de um sistema de detecção de intrusão por anomalia comportamental para redes móveis ad-hoc.

GRAU/ANO: Engenheiro/2007.

É concedida à Universidade de Brasília permissão para reproduzir cópias deste Projeto Final de Graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desta dissertação de mestrado pode ser reproduzida sem a autorização por escrito do autor.

---

Alan Salles Vieira Pinto  
SQS 212 Bloco J, Apartamento 504.  
CEP 70275-100 – Brasília – DF - Brasil

---

Fausto Barros de Sá Teles  
SGAN 912 Módulo D, Bloco D, Apartamento 14.  
CEP 70790-120 – Brasília – DF - Brasil

A Deus, nossos pais, irmãos, amigos e a todos que nos ajudaram nessa conquista.



## **AGRADECIMENTOS**

Ao nosso orientador Prof. Ricardo Staciarini Puttini, pelo apoio, incentivo e dedicação essenciais para o desenvolvimento deste trabalho.

Ao Mestre Fábio Miziara Silveira, verdadeiro co-orientador deste trabalho, que, apenas por razões de regulamentação, não pode ser registrado formalmente como tal. Sua presença constante foi fundamental para a conclusão deste trabalho.

À Universidade de Brasília por fornecer os equipamentos necessários à realização dos testes deste trabalho.

A todos, os nossos sinceros agradecimentos.





## RESUMO

**Este trabalho apresenta uma implementação de um IDS em Java testada em um ambiente real, mais especificamente uma rede ad-hoc com protocolo de roteamento OLSR (Optimized link-state routing). Tal implementação teve como objetivo a realização de testes para validar um modelo de um sistema de detecção de intrusão por anomalia comportamental. Esse modelo de detecção anteriormente feito foi simulado em um ambiente computacional, sendo assim este trabalho uma continuação para validação em ambiente real. Foi utilizado um ambiente de testes específicos, com uma topologia previamente definida no qual testamos com ferramentas geradoras de tráfego, simulando ataques com estas, bem como utilizando ferramentas específicas de ataque. A validação do IDS foi feita mostrando um caso específico de topologia sem mobilidade e os resultados encontrados foram satisfatórios para o proposto.**



## **ABSTRACT**

**This work presents an implementation of an IDS using the Java programming language tested in a real environment, more specifically in an ad-hoc network with routing protocol OLSR (Optimized link-state routing). This implementation aims at validating a model for an anomaly intrusion detection system by testing its performance. The previous model was tested on a simulated environment, and this work now, implemented and tested it on a real environment. It was used a specific topology and some traffic generation tools plus specific attack tools. For the presented topology we could get satisfactory results regarding to the proposed.**



# ÍNDICE

Capítulo	Página
<u>1. INTRODUÇÃO.....</u>	<u>19</u>
<u>1.1. ESTUDO BIBLIOGRÁFICO.....</u>	<u>21</u>
<u>1.2. ORGANIZAÇÃO DO TRABALHO.....</u>	<u>24</u>
<u>2. SISTEMA DE DETECÇÃO DE INTRUSÃO.....</u>	<u>26</u>
<u>2.1. DETECÇÃO DE INTRUSÃO POR COMPORTAMENTO.....</u>	<u>26</u>
<u>2.1.1. Modelos de Mistura de Distribuições para Caracterização Estatística do Comportamento.....</u>	<u>28</u>
<u>2.1.1.1. Algoritmo EM.....</u>	<u>30</u>
<u>2.1.1.2. Principais problemas na aplicação do algoritmo EM e soluções propostas.....</u>	<u>33</u>
<u>2.1.1.3. Estimação automática da ordem ótima do modelo.....</u>	<u>34</u>
<u>2.1.1.4. Algoritmo de detecção.....</u>	<u>36</u>
<u>2.1.1.5. Algoritmo de detecção para operação em tempo-real com GMM.....</u>	<u>37</u>
<u>2.1.2. Caracterização de Tráfego Normal em uma Manet e Construção do Modelo de Comportamento Normal.....</u>	<u>39</u>
<u>2.1.3. Detecção de Ataques de DoS e Scanner de Portas.....</u>	<u>41</u>
<u>2.1.4. Caracterização do Modelo de Tráfego dos Ataques.....</u>	<u>46</u>
<u>3. IMPLEMENTAÇÃO DO IDS.....</u>	<u>47</u>
<u>3.1. ARQUITETURA DO L-IDS.....</u>	<u>47</u>
<u>3.1.1. Framework de Detecção de Intrusão .....</u>	<u>48</u>

3.1.2. Mensagens Geradas pelos L-IDS .....	49
3.2. ESCOLHAS DE PROJETO.....	50
3.2.1. Escolha de dados de auditoria.....	51
3.2.2. Processo de abstração de eventos .....	53
3.2.3. Definição de fases de treinamento e detecção.....	55
3.2.4. Construção do modelo estatístico.....	56
3.2.5. Definição de limiares de detecção.....	56
4. RESULTADOS EXPERIMENTAIS.....	58
4.1. AMBIENTE DE TESTES.....	58
4.1.1. Notebooks.....	58
4.1.2. Topologia da Rede ad-Hoc.....	59
4.1.3. Protocolo de roteamento OLSR.....	62
4.1.4. Net-SNMP.....	62
4.1.5. Geradores de Tráfego.....	63
4.1.6. Ferramentas de Ataque.....	67
4.2. TESTES E ANÁLISE DE RESULTADOS.....	68
4.2.1. Modelo UDP.....	69
4.2.1.1. Detecção de ataques.....	73
4.2.2. Modelo TCP.....	75
4.2.2.1. Detecção de ataques.....	79

<b><u>5. CONCLUSÃO.....</u></b>	<b><u>82</u></b>
<b><u>6. REFERÊNCIAS BIBLIOGRÁFICAS.....</u></b>	<b><u>84</u></b>
<b><u>7. ANEXO I.....</u></b>	<b><u>86</u></b>
<b><u>8. ANEXO II.....</u></b>	<b><u>87</u></b>

## ÍNDICE DE TABELAS

**Tabela**

**Página**



## ÍNDICE DE FIGURAS

**Figura**

**Página**



# 1.INTRODUÇÃO

Redes *ad-hoc* são redes de comunicação sem fio que não possuem uma estrutura de rede pré-definida. No contexto de redes sem fio são conhecidas como redes Manet (Mobile Ad-Hoc Networks), ou redes móveis Ad-Hoc. Nestas redes cada nodo é capaz de rotear informações aos seus nodos vizinhos formando assim uma topologia.

Em uma Manet, os nodos podem a qualquer instante, aparecer, desaparecer ou mover-se dentro da rede. Conseqüentemente, a topologia da rede está sujeita a freqüentes mudanças. A possibilidade de mobilidade dos nodos, aliada a pouca confiabilidade e largura de banda limitada dos enlaces *wireless*, faz com que a disponibilidade de um nodo específico não possa ser garantida. Assim, os serviços de rede não podem ser concentrados em entidades centrais, devendo estar disponíveis de forma distribuída. A comunicação entre nodos geralmente utiliza-se das redundâncias resultantes do modelo de comunicação, o que acaba, de certo modo, compensando a ausência de confiabilidade acerca da disponibilidade dos nodos individualmente.

Com tantas características de mobilidade e adaptabilidade, devemos nos preocupar com um fator que acaba se tornando crítico nesse tipo de rede: a segurança. Nas Manets, devemos ter um serviço de segurança que se adeque às necessidades particulares desse tipo de rede, ou seja, o serviço deve ser escalável, distribuído e promover monitoramento *on time*. Quando se há nodo comprometido nesse tipo de rede, devido a sua estrutura, é provável que esse comprometimento venha a se espalhar por toda a rede caso alguma medida não seja tomada. Nesse cenário o uso de *Intrusion Detection Systems* (IDS) é difundido para se trabalhar com mecanismos corretivos e preventivos de segurança.

Basicamente, a literatura separa os IDS em duas categorias de acordo com o método de detecção utilizado: detecção por uso incorreto e detecção por anomalia comportamental. A detecção por uso incorreto usa assinaturas para validar um ataque. No ato de um ataque algumas ações previamente conhecidas e definidas são sequencialmente efetuadas pelo nodo atacante e o IDS é capaz de reconhecer o ataque pela assinatura de ações deixada. Na detecção por anomalia não se tem assinaturas. Define-se um comportamento considerado normal para a rede e este é comparado com o comportamento atual. Nesse caso, o IDS deve aprender o que é um comportamento considerado normal, e a partir daí comparar o

comportamento atual com o aprendido. No ato de um ataque, o comportamento da rede foge dos padrões usuais aprendidos pelo IDS e então é feita a detecção.

Neste trabalho será descrita uma implementação de um IDS por anomalia utilizando a linguagem Java. Também serão descritos os testes dessa implementação em um ambiente real de rede Manet simples. Nosso principal objetivo é a validação das simulações feitas em [1].

Existem muitos ataques passíveis de serem realizados numa rede. Nessa implementação de IDS iremos abordar os ataques de inundação de pacotes (DoS e DDoS) como também ataques do tipo scanner de portas.

Para validar as simulações, além da implementação do IDS devemos também definir um ambiente para testes. Nesse ambiente precisamos definir algumas configurações para que a validação aconteça o mais próximo possível da simulação realizada em [1].

Primeiramente devemos escolher o sistema operacional. Nesse caso, foi utilizado Linux Fedora 5 para rodar o IDS e alguns softwares de geração de tráfego. A escolha de um sistema como o Linux deve-se ao fato de se poder customizar detalhadamente os parâmetros para a rede e para o IDS.

Outra variável importante a ser definida para esse ambiente de simulação é como a Manet será construída. Nesse trabalho será implementada com *notebooks*, uma rede sem fio, usando o protocolo de roteamento OLSR (*Optimized Link State Routing Protocol*) [17] por meio de um *daemon* (olsrd) [17] para Linux. Esse é um protocolo pró-ativo de roteamento aberto e o *daemon* está disponível para download gratuitamente na Internet.

Além do protocolo de roteamento, os nodos devem possuir um agente SNMP para que se possa ter acesso a base de dados (MIB). Para uma validação coerente, devemos utilizar um agente standard. O NETSNMP [19] foi usado por ser um pacote de utilitários SNMP bem difundido para sistemas operacionais Linux e seu *daemon*, o SNMPD [19], de grande customização.

Devemos pensar também em como gerar o tráfego para rede. Existem diversas ferramentas para se gerar tráfego em uma rede, dentre elas programas geradores de tráfego, como o PACKIT [21] e IPERF [22]. Esses programas possuem diversas funções para geração de pacote, que vão desde geração de taxas fixas de envio de dados até taxas aleatórias com

diversas distribuições temporais diferentes. Uma boa geração de pacotes é fundamental para se ter controle do ambiente de testes e se obter dados aproveitáveis.

Num ambiente computacional onde são realizadas simulações, por mais detalhada que esta seja, podem-se omitir algumas características importantes que aparecem num ambiente real. Neste tipo de ambiente, essas variáveis ignoradas se tornam parte do experimento, influenciando sempre de alguma forma no resultado final obtido. Nesse trabalho, devemos levar em conta que existem imperfeições nos enlaces wireless entre os nodos, como efeitos de múltiplo acesso, desvanecimento (*fading*), ruídos e interferências decorrentes de fontes eletromagnéticas exógenas ao sistema, entre outros [2]. Tendo em vista tudo que foi dito até aqui, que contempla desde o projeto e implementação do IDS até a criação do ambiente de testes, podemos então iniciar a tentativa de validação de [1]. Essa análise deve cobrir a observação do comportamento do IDS implementado e também a comparação dessa observação com a simulação já realizada.

## **1.1. ESTUDO BIBLIOGRÁFICO**

Para termos uma boa noção de qual o estado da arte no qual a segurança de redes Manet se encontra, devemos fazer um pequeno estudo dos trabalhos anteriores relacionados ao nosso projeto. Isso é importante pra mostrar em que contexto de desenvolvimento tecnológico esse trabalho é proposto.

A segurança em Manet é um assunto bastante discutido atualmente, por tratar-se de contexto e problemática novos, com muitos pontos ainda em aberto. A maioria dos estudos até a data de realização deste trabalho teve foco em mecanismos de proteção preventiva dos protocolos básicos (e.g. protocolos de roteamento), utilizando mecanismos de autenticação. Em geral essa abordagem é bastante interessante, mas muito vulnerável à presença de nodos comprometidos na rede. A segurança deve portanto ser reforçada utilizando mecanismos corretivos de segurança, tais como sistemas de detecção de intrusão.

Sabe-se que é muito difícil garantir que um sistema de informação seja mantido seguro durante todo seu tempo de utilização. Os sistemas de detecção de intrusão têm objetivo de monitorar a utilização destes sistemas com a intenção detectar o aparecimento de estados inseguros.

O comportamento em caso de detecção descreve a resposta do IDS aos ataques detectados. Quando o IDS responde a um ataque executando ações corretivas (fechando brechas) ou pró-ativas (registrando possíveis atacantes, fechando serviços), o sistema é classificado como ativo. Se o sistema apenas gera e envia alertas ele é dito passivo. Nosso projeto implementou um IDS passivo, já que nosso objetivo era medir a capacidade do motor de detecção, para validar [1].

A fonte de dados discrimina os IDS com base no tipo de dado de auditoria que eles analisam. Esse dado pode ser trilha de auditoria (e.g. log de sistema), pacotes de rede, log de aplicativos ou mesmo alertas gerados por outros sistemas de detecção de intrusão. Nossa escolha quanto a essa classificação será abordada no capítulo 3.

O paradigma de detecção descreve o mecanismo de detecção usado pelo IDS. Estes sistemas podem avaliar estados (seguro/inseguro) ou transições (de seguro para inseguro). Nosso IDS avalia estados, baseando-se em limiares de probabilidade.

Por fim, a Frequência de uso descreve a utilização do IDS. Certos IDS são usados na monitoração contínua e em tempo real do sistema alvo, enquanto outros são executados periodicamente. O IDS apresentado neste trabalho é projetado para uso contínuo.

A tabela 1-1 mostra as principais propostas de sistemas de detecção de intrusão distribuídos até aqui apresentadas.

IDS	Fonte de Dados	Método de Detecção	Pré-processamento distribuído	Detecção centralizada	Análise em tempo-real	Tipo de Resposta
AAFID [50]	Sistema	Uso incorreto	Sim	Sim	Sim	Passivo
DIDS [52]	Sistema/Rede	Híbrido	Sim	Sim	Sim	Passivo
Grids [51]	Sistema/Rede	Híbrido	Sim	Sim	Não	Passivo
CSM [55]	Sistema	Anomalia	Sim	Não	Sim	Ativa
JiNao [22]	MIB/Rede	Híbrido	Sim	Sim	Sim	Passivo
EMERALD [43]	Sistema/Rede	Híbrido	Sim	Não	Sim	Ativo
IDA [10]	Sistema	Uso incorreto	Agentes móveis	Sim	Sim	Passivo
SPARTA [32]	Sistema/Rede	Uso incorreto	Agentes móveis	Não	Sim	Passivo

**Tabela 1-1 – Trabalhos anteriores**

(Fonte: F. Miziara [1] p.29)

A análise dessa tabela mostra com clareza o estado da arte atual dos IDS voltados para redes Manet. Dentro desse contexto, nosso IDS apresenta uma arquitetura que pretende preencher parte desses requisitos dos ambientes Manet, adaptada da proposta apresentada em [2].

## **1.2. ORGANIZAÇÃO DO TRABALHO**

Esse trabalho está dividido em 5 capítulos. O segundo capítulo é uma síntese da descrição do modelo de IDS proposto por [1]. O terceiro capítulo trata do projeto de implementação do IDS, mencionando as escolhas de projeto e a arquitetura do software. No

quarto capítulo encontramos os resultados experimentais bem como as descrições detalhadas do ambiente de testes e análises desses resultados. Por fim, no capítulo 5 temos as conclusões tiradas do trabalho, com uma análise de tudo que foi feito durante o projeto, considerações finais e propostas para trabalhos futuros.



## **2. SISTEMA DE DETECÇÃO DE INTRUSÃO**

Este capítulo apresenta a base teórica do motor de detecção do IDS por anomalia. Este não é exatamente o escopo do nosso trabalho, de forma que este capítulo pretende ser meramente uma referência ao que foi tratado e desenvolvido em [1].

Como visto, o IDS pode ser um sistema indispensável para redes móveis ad-Hoc. Ele tem como função prover monitoração pró-ativa do ambiente, identificando violações da política de segurança. O IDS também pode interagir com diversos serviços de segurança da rede para eliminar as causas de um ataque. O objetivo deste capítulo é mostrar uma proposta de sistema de detecção de intrusão para redes Manet.

O projeto de tal IDS deve levar em conta as características desse tipo de rede (e.g. banda passante, problemas nos enlaces sem fio, interferências e ausência de um nodo central). Para o IDS proposto, temos um procedimento de detecção de intrusão que funciona primeiramente coletando-se e analisando-se dados de auditoria da rede, e em seguida, caso a análise aponte um ataque, emite-se um aviso para a gerência de segurança.

O escopo desse IDS é um IDS local a ser colocado em cada nodo de uma rede ad-Hoc, que seja capaz de detectar um ataque no próprio nodo ou um ataque que esteja passando pelo nodo com o IDS implementado. Este tipo de topologia é o usual em redes Manet, onde cada nodo pode servir de roteamento para outros nodos. Como dito, um estudo mais completo da teoria que possibilitou a construção do IDS apresentado neste trabalho pode ser encontrado em [1] e [2].

### **2.1. DETECÇÃO DE INTRUSÃO POR COMPORTAMENTO**

Uma intrusão em uma rede pode ser detectada observando-se um desvio no comportamento usual desta. Para determinar o que é usual ou não, deve-se ter um padrão considerado normal, que é obtido por uma observação prévia da rede. Desta forma, o IDS pode comparar o comportamento atual com o prévio, usado para o aprendizado, e enfim gerar alarmes em caso de discrepância.

De maneira geral a construção deste tipo de IDS pode ser descrito em 3 fases: Construção de um modelo de comportamento normal válido; Detecção e Atualização do modelo de comportamento assumido [3].

Na etapa de construção do modelo é onde o treinamento do sistema acontece. Esse treinamento, ou aprendizado, deve ser feito de forma a abranger as características usuais desejadas para o sistema. Nessa etapa devemos identificar que tipo de informação de auditoria deve ser coletada para que o sistema seja descrito. Essas informações devem então ser usadas no estágio de detecção, de forma a construir o modelo de comportamento. O modelo de comportamento pode ser desenvolvido usando-se modelagem estatística [4], redes neurais ou algoritmos genéticos e outras técnicas.

A arquitetura desses modelos possui características e algoritmos de aprendizagem que funcionam de forma a otimizar a atualização dos parâmetros da modelagem. Podem ainda ser feitas mudanças nos parâmetros de ajuste para que o sistema se adeque a um novo comportamento.

Após se escolher a arquitetura e quais informações de auditoria usar, o grande desafio é realizar um bom treinamento inicial, ou seja, que contenha uma representação adequada do sistema a ser representado.

Além de se considerar a arquitetura do modelo e o tipo de informação de auditoria a ser usado, devemos levar em conta o desempenho do sistema, ainda mais se tratando de um IDS que deve emitir alertas em tempo real.

Para um IDS comportamental, a definição do que é ataque ou não pode ser modelada de forma binária ou através de limiares de probabilidade. Como numa detecção por comportamento não se tem uma assinatura de um uso incorreto, mas sim de uma indicação de desvio comportamental, os alertas gerados pelo IDS podem ser pós-processados para eliminação de falsos positivos.

Ao longo do tempo de utilização de uma rede, o comportamento considerado usual pode ser modificado em relação ao aprendido inicialmente pelo IDS. Para que o IDS continue tendo uma referência do que é comportamento normal, a modelagem do sistema deve ser refeita. Essa atualização pode ser contínua ou periódica.

Esse tipo de mudança é um dos problemas da detecção por comportamento. Deve-se tomar cuidado para que um atacante não se utilize disso para induzir um ataque como comportamento normal. Isso é um tópico em aberto. Já foram propostas formas de atualização recursiva dos parâmetros do modelo, mas este trabalho não implementa tal solução, por conta do risco embutido em utilizá-la.

Para o IDS local utilizado nesse trabalho, usaremos o método comportamental com a modelagem do comportamento sendo feita estatisticamente. Para isso utilizamos algumas informações de auditoria do sistema na forma de variáveis aleatórias.

Inicialmente, são observados valores de informações que mostram o comportamento de aplicações e protocolos na Manet. Como a modelagem estatística é sempre complicada, utilizou-se o modelo de multi misturas para então ajustá-lo aos dados de tráfego. Tendo como referência trabalhos similares [5], o IDS será modelado estatisticamente para detecção de ataques DoS, DDoS e Scanner de portas, pois tais ataques geram mudanças significativas no comportamento do tráfego.

### **2.1.1. Modelos de Mistura de Distribuições para Caracterização Estatística do Comportamento**

Dada uma variável aleatória  $\mathbf{y}$  n-dimensional, um modelo de mistura de distribuições é utilizado para modelar sua função de distribuição de probabilidade (f.d.p.). As realizações da variável  $\mathbf{y}$  são tiradas do domínio de dados de auditoria.

Seja  $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n]^T$  um vetor de realização observável de  $\mathbf{y}$ . A combinação linear de funções nucleares básicas define uma f.d.p. para  $\mathbf{y}$ , conforme a equação 2-1.

$$p(\mathbf{y}_i) = \sum_{k=1}^K w_k g_k(\mathbf{y}_i, \boldsymbol{\theta}_k) \quad \text{Eq. 2-1}$$

onde:  $g$  representa cada função nuclear,  $w_k$  são os fatores de ponderação de cada função nuclear e  $\boldsymbol{\theta}_k$  são os parâmetros das funções nucleares.

O vetor  $\Psi = [w_1, w_2, \dots, w_k, \theta_1, \theta_2, \dots, \theta_k]$  representa todos os parâmetros desconhecidos do modelo, que devem ser ajustados para enquadrar  $\Psi$  à  $Y$ . O número  $K$  é a ordem do sistema, geralmente fixo.

Um algoritmo iterativo para otimização, por um critério de máxima verossimilhança (ML) foi apresentado em [7] e é denominado algoritmo de estimação-maximização (EM).

Os dados de referência devem conter informações sobre diferentes padrões de comportamentos válidos. É interessante que o modelo promova a clusterização desses dados. A adoção de um modelo de mistura parametrizado [8] permite obtenção automática da clusterização.

Cada função nuclear representa individualmente a f.d.p. de um *cluster* diferente no conjunto de dados. Assim, um modelo de mistura de ordem  $K$  é diretamente aplicável a uma mistura populacional de  $K$  grupos.

Dessa maneira, os coeficientes  $w_k$  equivalem à probabilidade de cada *cluster*  $p(k)$ . Do mesmo modo, as probabilidades posteriores de cada realização  $p(y_i | k)$  podem ser obtidas, dado os valores de cada função nuclear em  $\bar{y}$ . Pode-se obter uma estimativa para as probabilidades posteriores na forma  $p(k | y_i) = p(y_i | k)p(k)/p(y_i)$ .

Para determinação automática de  $K$ , há um algoritmo baseado em uma otimização por um critério de maximização de entropia. Este algoritmo, adaptado de [9] para modelos paramétricos, é descrito na seção seguinte.

Para dados multivariados, o caso especial de funções nucleares gaussianas multivariadas forma um modelo conhecido como modelo de mistura de gaussianas (GMM). Este modelo, em particular, pode ser facilmente ajustado iterativamente pelo algoritmo EM, com boas propriedades de convergência, dado uma correta estimativa de  $K$ .

Assim, neste trabalho, considera-se o caso de um GMM. Portanto, a descrição do algoritmo EM apresentada aqui está particularizada para este caso. Em [7,6] pode-se encontrar os detalhes de uma descrição mais genérica do algoritmo EM.

Para o caso de GMM,  $g_k$  em Eq. é substituído por  $\phi(\mathbf{y}_i, \boldsymbol{\mu}_k, \mathbf{R}_k)$ , que denota uma f.d.p. normal multivariada com média  $\boldsymbol{\mu}_k$  e matriz de covariância  $\mathbf{R}_k$ . A Eq. pode ser reescrita como Eq. 2-2:

$$p(\mathbf{y}_i) = \sum_{k=1}^K w_k \phi(\mathbf{y}_i, \boldsymbol{\mu}_k, \mathbf{R}_k) \quad \text{Eq. 2-2}$$

onde:  $\boldsymbol{\Psi} = [w_1, w_2, \dots, w_K, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K, \mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_K]$

#### 2.1.1.1. Algoritmo EM

A estimação por máxima verossimilhança consiste de encontrar um  $\boldsymbol{\Psi}^*$  para  $\boldsymbol{\Psi}$  que maximize a verossimilhança de  $\mathbf{y}$  para um dado conjunto de observações  $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n]^T$ . Assumindo-se arbitrariamente que  $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n$  sejam realizações independentes do vetor característico  $\mathbf{Y}$ , a função de verossimilhança logarítmica como função de  $\boldsymbol{\Psi}$  é dada por Eq. 2-3:

$$\log L(\boldsymbol{\Psi}) = \sum_{j=1}^n \log \left( \sum_{k=1}^K w_k \phi(\mathbf{y}_j, \boldsymbol{\mu}_k, \mathbf{R}_k) \right) \quad \text{Eq. 2-3}$$

Uma estimativa de  $\boldsymbol{\Psi}$  com máxima verossimilhança é dada pelas raízes da Eq. 2-3, que corresponde a um máximo local de Eq. 2-:

$$\frac{\partial \log L(\boldsymbol{\Psi})}{\partial \boldsymbol{\Psi}} = 0 \quad \text{Eq. 2-4}$$

Para facilitar a otimização são introduzidas variáveis ocultas (não observadas)  $z_{jk}$ , onde  $z_{jk}$  é definido como 1 ou 0 se  $\mathbf{y}_j$  é proveniente ou não do k-ésimo componente do modelo de mistura ( $j = 1, \dots, n$ ;  $k = 1, \dots, K$ ), respectivamente. O vetor de dados completo (não observado)  $\mathbf{X}$  é formado por  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T$ , onde  $\mathbf{z}_j = [z_{j1}, z_{j2}, \dots, z_{jK}]^T$  são os vetores de

variáveis ocultas para uma realização  $\mathbf{y}_j$  com  $\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n$  sendo realizações independentes de uma distribuição multinomial consistindo de um experimento em  $K$  categorias, com respectivas probabilidades  $w_1, \dots, w_K$ . As realizações  $\mathbf{x}_1 = (\mathbf{y}_1^T, \mathbf{z}_1^T)^T, \dots, \mathbf{x}_K = (\mathbf{y}_K^T, \mathbf{z}_K^T)^T$  são consideradas independentes e identicamente distribuídas.

Para esta especificação, a função de verossimilhança logarítmica para o vetor completo  $\mathbf{X}$  é dada por:

$$\log L_c(\Psi) = \sum_{k=0}^K \sum_{j=1}^n z_{jk} \log \{w_k \phi(\mathbf{y}_j, \boldsymbol{\mu}_k, \mathbf{R}_k)\} \quad \text{Eq. 2-5}$$

O algoritmo EM [7] é considerado efetivo quando maximizar a verossimilhança do vetor de dados completos ( $\mathbf{X}$ ) é mais simples que maximizar a verossimilhança dos dados incompletos (Eq. 2-3).

A execução iterativa do algoritmo EM consiste de dois passos pra cada iteração: passo E (estimação) e passo M (maximização). Considerando  $\Psi^i$  como uma estimativa de  $\Psi$  na  $i$ -ésima iteração, o passo E requer o cálculo de 2-6:

$$Q(\Psi; \Psi^i) = E_{\Psi^i}(\log L_c(\Psi) | \mathbf{Y}) \quad \text{Eq. 2-6}$$

onde:  $Q(\Psi; \Psi^i)$  é o valor esperado condicional de  $\log L_c(\Psi)$ , dado os dados observados  $\mathbf{Y}$  e o ajuste atual  $\Psi^i$  para  $\Psi$ .

Uma vez que  $\log L_c(\Psi)$  é uma função linear das variáveis ocultas  $z_{jk}$ , o passo E é executado substituindo-se  $z_{jk}$  por seu valor esperado condicional, dado  $\mathbf{y}_j$ , usando-se  $\Psi^i$  para  $\Psi$ . Isto é,  $z_{jk}$  é substituído em Eq. 2-6 por Eq. 2-7:

$$\tau_k(\mathbf{y}_j; \Psi^i) = E_{\Psi^i}(z_{jk} | \mathbf{y}_j) = \frac{w_k^i \phi(\mathbf{y}_j, \boldsymbol{\mu}_k^i, \mathbf{R}_k^i)}{\sum_{k'=1}^K w_{k'}^i \phi(\mathbf{y}_j, \boldsymbol{\mu}_{k'}^i, \mathbf{R}_{k'}^i)} \quad \text{Eq. 2-7}$$

Pode-se reconhecer  $\tau_k(\mathbf{y}_j; \Psi^i)$  na 2-7 como a estimativa corrente da probabilidade posterior da j-ésima realização ( $\mathbf{y}_j$ ) ter vindo do k-ésimo grupo, i.e.  $p(k | \mathbf{y}_j)$ . A Eq. 2-7 pode ser, então, reescrita como:

$$p(k | \mathbf{y}_i) = \frac{w_k^i \phi(\mathbf{y}_i, \boldsymbol{\mu}_k^i, \mathbf{R}_k^i)}{\sum_{k=1}^K w_k^i \phi(\mathbf{y}_i, \boldsymbol{\mu}_k^i, \mathbf{R}_k^i)} = \frac{p(k) p(\mathbf{y}_i | k)}{p(\mathbf{y}_i)} \quad \text{Eq. 2-8}$$

A estimativa da probabilidade *a priori* de cada *cluster* ( $p(k)$ ) é dada pela estimativa corrente do fator de ponderação  $w_k^i$ .

Substituindo-se Eq. 2-8 em Eq. 2-7, obtêm-se a expressão para o passo E:

$$Q(\Psi; \Psi^i) = \left( \sum_{j=1}^n \log \sum_{k=1}^K w_k^i \phi(\mathbf{y}_j, \boldsymbol{\mu}_k^i, \mathbf{R}_k^i) \right) / n = \left( \sum_{j=1}^n \log(p(\mathbf{y}_j)) \right) / n \quad \text{Eq. 2-9}$$

No passo M na (i + 1)-ésima iteração, o objetivo é escolher  $\Psi^{i+1}$  que maximize  $Q(\Psi; \Psi^i)$ . Assim, o ajuste atual para as proporções de mistura ( $w_k^{i+1}$ ), os componentes de média ( $\boldsymbol{\mu}_k^{i+1}$ ) e as matrizes de covariância ( $\mathbf{R}_k^{i+1}$ ) são dadas explicitamente por Eq. 2-10:

$$\begin{aligned} w_k^{i+1} &= \sum_{i=1}^n p(k | \mathbf{y}_i) / n \\ \boldsymbol{\mu}_k^{i+1} &= \sum_{i=1}^n p(k | \mathbf{y}_i) \mathbf{y}_i / \sum_{i=1}^n p(k | \mathbf{y}_i) \\ \mathbf{R}_k^{i+1} &= \sum_{i=1}^n p(k | \mathbf{y}_i) (\mathbf{y}_i - \boldsymbol{\mu}_k^{i+1})(\mathbf{y}_i - \boldsymbol{\mu}_k^{i+1})^T / \sum_{i=1}^n p(k | \mathbf{y}_i) \end{aligned} \quad \text{Eq. 2-10}$$

No uso do algoritmo EM, a verossimilhança da mistura  $L(\Psi)$  não pode nunca decrescer após uma sequência EM. Assim,  $L(\Psi^{i+1}) \geq L(\Psi^i)$ , implicando na convergência de  $L(\Psi)$  para um certo valor  $L^*$ , se a sequência de valores para a verossimilhança for limitada. Os passos E e M são alternados repetidamente até a verossimilhança ou as estimativas para os

parâmetros mudem de um valor arbitrariamente pequeno, indicando a convergência do algoritmo.

O algoritmo EM pode ser sintetizado da seguinte forma:

<b>Algoritmo 2 – EM</b>
1: Inicia-se $\Psi^0$ com valores aleatórios para $w_1^0, w_2^0, \dots, w_K^0, \mu_1^0, \mu_2^0, \dots, \mu_K^0, R_1^0, R_2^0, \dots, R_K^0$ 2: Para $i = 0$ , calcula-se $L^i$ conforme a Eq. 2- 3: Para $i = i+1$ , calcula-se $\Psi^{i+1}$ (Eq. 2-10) e $L^{i+1}$ (Eq. 2-9) 4: Se $L^{i+1} - L^i > \delta$ (constante de convergência), repete-se 3 5: Atualiza-se os valores reais dos parâmetros $\Psi^* = \Psi^i$ .

#### 2.1.1.2.Principais problemas na aplicação do algoritmo EM e soluções propostas

O primeiro problema na aplicação do algoritmo EM, conforme descrito na última seção, está relacionado com o fato da função de verossimilhança ter, em geral, múltiplos máximos locais, resultando em ajustes sub-ótimos ou mesmo inadequados. Para tratar deste problema, diversos procedimentos de iniciação são propostos [6,10,8].

Além disso, no caso de componentes das matrizes de covariância não terem restrições a função de verossimilhança não é limitada [6]. Além disso, cuidados especiais devem ser tomados para os casos onde um componente (*cluster*) ajustado possui uma variância generalizada, o que acarreta em valores relativamente grandes para o máximo local. Este componente corresponde a um *cluster* contendo poucos pontos, que estão relativamente próximos uns dos outros. Há, portanto, uma necessidade de se monitorar o tamanho relativo das proporções das misturas ajustadas, os componentes das variâncias generalizadas com objetivo de eliminar estes máximos locais espúrios. Deve-se também uma necessidade de se monitorar as distâncias euclidianas entre as médias de componentes ajustados, com objetivo de verificar se os *clusters* implicados representam uma separação real entre as médias ou se



trata-se de um ou mais *clusters* que caíram quase em um sub-espço do vetor característico original [6].

Levando em conta o que foi dito nos parágrafos acima, uma versão modificada do algoritmo EM é sumarizada a seguir:

### Algoritmo 3 – EM Modificado

- 1: Inicia-se um contador  $c = 0$
- 2:  $c = c+1$
- 3: Inicia-se  $\Psi^0$  com valores aleatórios para  $w_1^0, w_2^0, \dots, w_K^0, \mu_1^0, \mu_2^0, \dots, \mu_K^0, R_1^0, R_2^0, \dots, R_K^0$
- 4: Para  $i = 0$ , calcula-se  $L^i$  conforme a Eq. 2-.
- 5: Para  $i = i+1$ , calcula-se  $\Psi^{i+1}$  (Eq. 2-) e  $L^{i+1}$  (Eq. 2-)
- 6: Se  $L^{i+1} - L^i > \delta$  (constante de convergência), repete-se 5
- 7: Se o determinante de qualquer uma das matrizes de covariâncias  $< \epsilon$  (uma constante pequena), repete-se 2
- 8: Se  $(c = 1)$  ou  $(L^i > L_{opt})$  então faz-se  $L_{opt} = L^i$  e  $\Psi_{opt} = \Psi^i$
- 9: Se  $c \leq C_{max}$ , repete-se 2
- 10: Atualiza-se os valores reais dos parâmetros  $\Psi^* = \Psi_{opt}$ .

#### 2.1.1.3. Estimação automática da ordem ótima do modelo

Para aplicação do algoritmo EM, a ordem do modelo ( $K$ ) deve ser assumida *a priori*. No entanto, em muitos casos, o número de partições não é conhecido *a priori*. Deve-se, portanto, ter um mecanismo para se descobrir o número de partições mais provável, para um dado modelo.

Queremos construir uma estimativa para  $K$  que implique em uma “partição ideal”, isto é,  $p(k | y_i)$  é próximo da unidade para um valor de  $k$  e próximo de zero, para todos os outros

valores, para cada realização  $\mathbf{y}_i$ . Como descrito em [9], esta partição ideal deve ser obtida pela minimização da entropia de Shannon dado os dados observados<sup>1</sup> ( $\mathbf{Y}$ ), que deve ser avaliada para cada observação como Eq. 2-1:

$$H_K = -\sum_{k=1}^K p(k | \mathbf{y}_i) \log(p(k | \mathbf{y}_i)) \quad \text{Eq. 2-11}$$

O valor esperado desta entropia é avaliado tirando-se a média de  $H_K$  sobre todos os dados observados Eq. 2-2:

$$E^*(H_K) = -\sum_{i=1}^n \sum_{k=1}^K p(k | \mathbf{y}_i) \log(p(k | \mathbf{y}_i)) / n \quad \text{Eq. 2-22}$$

onde:  $E^*$  denota o estimador da esperança.

Então, procede-se ao ajuste de  $K_{\max}$  modelos com ordens diferentes ( $K = 1, 2, \dots, K_{\max}$ ) e avalia-se a entropia esperada (Eq. 2-22) para cada um deles. O modelo que resultar em uma medida mínima é considerado como o modelo ótimo.

O algoritmo EM com estimação automática de ordem ótima é sumarizado a seguir:

<b>Algoritmo 4 – EM com Estimação de Ordem Ótima</b>
1: Inicia-se $K = 0$ , $H_{\text{opt}} = 0$ , $K_{\text{opt}} = 1$ 2: $K = K+1$ 3: Ajusta-se o modelo de ordem $K$ aos dados $\mathbf{Y}$ , usando-se o algoritmo EM modificado (algoritmo 3). 4: Estima-se a esperança de $H_K$ (Eq. 2-22) 5: Se $(K = 1)$ ou $(H_K < H_{\text{opt}})$ então faz-se $H_{\text{opt}} = H_K$ ; $K_{\text{opt}} = K$ ; e $\Psi_{\text{opt}} = \Psi^*$ 6: Se $K < K_{\max}$ (uma constante fixa), repete-se 2 7: Atualiza-se a ordem real do modelo com o valor ótimo: $K = K_{\text{opt}}$

---

<sup>1</sup>

8: Atualiza-se os valores reais dos parâmetros  $\Psi^* = \Psi_{\text{opt}}$ .

#### 2.1.1.4. Algoritmo de detecção

Na fase de detecção, o modelo de comportamento já está computado e disponível para a realização de inferências sobre o comportamento sistema. O objetivo consiste em definir alguma penalidade  $\lambda$  (e.g.  $0 \leq \lambda \leq 1$ ), indicando o grau de normalidade de uma realização de certamente anômalo ( $\lambda = 0$ ) a certamente normal ( $\lambda = 1$ ).

Neste trabalho, define-se um procedimento de detecção formado por duas etapas: uma classificação (Bayesiana) e uma inferência acerca da pertinência em um determinado *cluster*.

A classificação consiste da avaliação das probabilidades posteriores de cada *cluster* condicionadas ao novo dado  $\mathbf{y}'$ , isto é,  $p(k | \mathbf{y}')$  (Eq. 2-8) para  $k = (1, 2, \dots, K)$ .

A inferência acerca da pertinência em um determinado cluster consiste em avaliar a probabilidade de o novo dado estar contido em algum intervalo de pertinência ( $\Pi_k$ ), definido como uma função da nova observação  $\mathbf{y}'$  e dos parâmetros da distribuição do *cluster* (e.g.  $\mu_k$  e  $\mathbf{R}_k$ ), o que pode ser formalmente expresso como Eq. 2-33:

$$p(\mathbf{y}' \in \Pi_k | k) = \int_{\Pi_k} g_k(\mathbf{y}, \boldsymbol{\theta}_k) d\Pi_k \quad \text{Eq. 2-33}$$

De fato, a probabilidade definida na Eq. 2-33 se parece com uma função de distribuição acumulativa (f.d.a), se  $\Pi_k$  for definido conforme mostrado na Eq. 2-44 abaixo [11]:

$$\Pi_k = \left\{ \mathbf{y} \in \mathbb{R}^d \mid \frac{\|(\mathbf{y} - \mu_k)\|^2}{\|\mathbf{R}_k\|} \geq \gamma^2 \right\} \quad \text{Eq. 2-44}$$

onde:  $\| \|^2$  e  $\| \|$  denotam algum tipo de operadores para calculo da norma e  $\gamma$  é uma constante que depende de  $\mathbf{y}'$ .

Finalmente, a função de penalidade para a detecção pode ser definida, conforme a Eq. 2-55:

$$\lambda(\mathbf{y}') = \sum_{k=1}^K p(k | \mathbf{y}') p(\mathbf{y}' \in \Pi_k | k) \quad \text{Eq. 2-55}$$

#### 2.1.1.5. Algoritmo de detecção para operação em tempo-real com GMM

Em geral, é desejável que os estágios de detecção e atualização do modelo de comportamento possam ser executados continuamente. Dessa forma, os algoritmos para detecção e atualização do modelo devem ser projetados para operação em tempo-real.

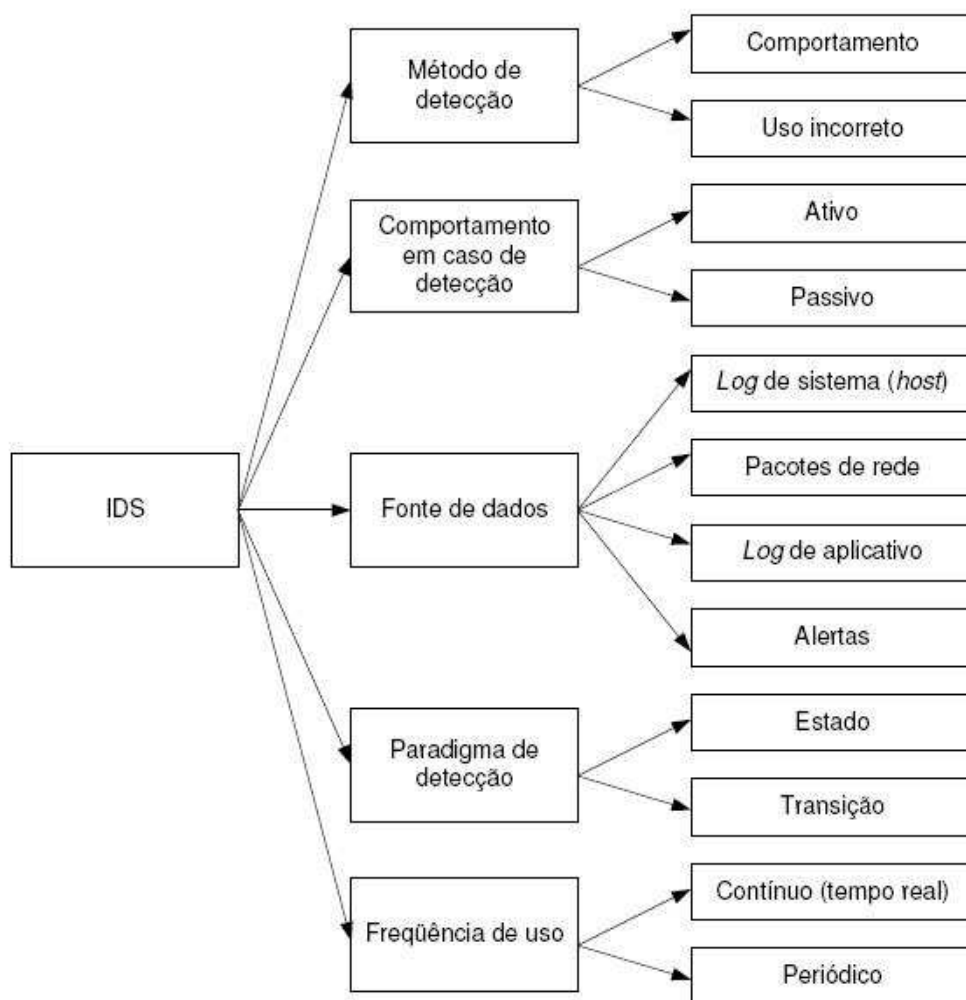
A Eq. 2-33 não pode ser sempre avaliada analiticamente. Uma solução geral seria avaliar esta equação integral numericamente, mas isto pode ser proibitivo, pois tal avaliação numérica é computacionalmente muito intensiva [12]. No entanto, um algoritmo computacionalmente eficiente para avaliação desta integral por ser estabelecido no caso especial de distribuições Gaussianas.

Assim, quando utiliza-se um GMM, a avaliação da Eq. 2-33 pode ser feita escolhendo-se convenientemente os elementos indefinidos desta equação, isto é, o operador de norma e  $\gamma$ . Define-se, portanto,  $\Pi_k$  como o espaço complementar (côncavo) da elipsóide de isodensidade (em  $\mathcal{R}^d$ ), cuja fronteira contém  $\mathbf{y}'$  e tem como centro de gravidade  $\boldsymbol{\mu}_k$ . Isso significa que  $\Pi_k$  é limitado internamente por uma superfície elipsoidal d-dimensional, formada por todos os pontos que possuem o mesmo valor de densidade que  $\mathbf{y}'$  (i.e.  $\phi(\mathbf{y}, \boldsymbol{\mu}_k, \mathbf{R}_k) = \phi(\mathbf{y}', \boldsymbol{\mu}_k, \mathbf{R}_k)$ ). Assim, reescrevendo-se a Eq. 2-44, tem-se a Eq. 2-6:

$$\Pi_k = \left\{ \mathbf{y} \in \mathcal{R}^d \mid \sum_{\alpha\beta} (y_\alpha - \mu_\alpha) [R_k^{-1}]_{\alpha\beta} (y_\beta - \mu_\beta) \geq \gamma^2 \right\} \quad \text{Eq. 2-66}$$

onde:  $\mathbf{y} = (y_1, y_2, \dots, y_d)^T$ ;  $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_d)^T$ ;  $[R_k^{-1}]_{\alpha\beta}$  é o elemento da  $\alpha$ -ésima linha e da  $\beta$ -ésima coluna da matriz de covariância inversa, e  $\gamma$  é dada pela Eq. 2-7:

Para a classificação dos sistemas de detecção de intrusão atuais H. Debar et al. [3] propõe a taxionomia mostrada abaixo:



**Figura 1-1 – Tipos de IDS**

(Fonte: H. Debar et al. [3], com adaptações)

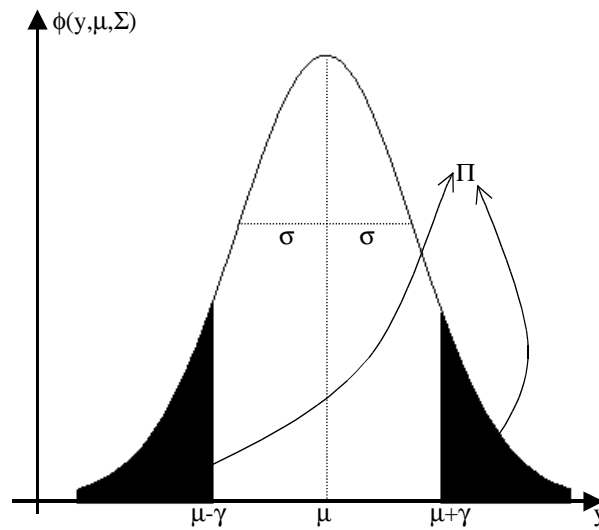
Como mostrado na figura 1-1, os critérios de classificação são: método de detecção, comportamento em caso de detecção, fonte de dados, paradigma de detecção e frequência de uso.

O método de detecção já foi abordado previamente nesse capítulo. Como dito, nesse projeto foi implementado um IDS cujo método de detecção é a detecção por comportamento.

$$\gamma^2 = \sum_{\alpha\beta} (y'_\alpha - \mu_\alpha) [R_k^{-1}]_{\alpha\beta} (y'_\beta - \mu_\beta)$$

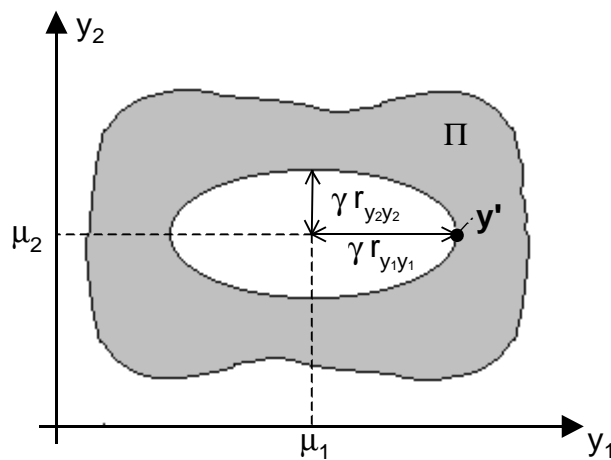
Eq. 2-77

Essa estratégia pode ser ilustrada para os espaços uni e bidimensionais, conforme mostrados nas Figura 2-1 e Figura 2-2, respectivamente. Esta última foi desenhada para uma distribuição Gaussiana bivariada, com matriz de covariância diagonal (não correlacionada).



**Figura 2-1 -  $\Pi$  para um *cluster* com distribuição Gaussiana unidimensional**

(Fonte: R. Puttini [2], p.127)



**Figura 2-2 -  $\Pi$  para um *cluster* com distribuição Gaussiana bivariada e matriz de covariância diagonal**

(Fonte: R. Puttini [2], p.127)

Como os dados observados podem pertencer a um espaço multidimensional ( $\Re^d$ ), uma distância generalizada  $\gamma'$  é definida na Eq. 2-88. Isto possibilita a normalização das probabilidades expressas na Eq. 2-33 para dados pertencentes a espaços dimensionais diferentes, o que permite a redução da computação para o espaço unidimensional, o que pode ser realizado por um procedimento simples de *lookup* em tabela.

$$\gamma' = \gamma / \sqrt{d} \quad \text{Eq. 2-88}$$

### **2.1.2. Caracterização de Tráfego Normal em uma Manet e Construção do Modelo de Comportamento Normal**

Como já dito anteriormente, não existe um consenso sobre como seria um tráfego normal em uma rede ad-Hoc. De fato, devemos observar que na realidade esse tráfego sempre irá variar de uma rede para outra, pois o tráfego é algo inerente às aplicações que são rodadas na rede. Considera-se também difícil obter dados reais de uma rede ad-Hoc em funcionamento que garantidamente não contenham vestígios de intrusão, então muitos trabalhos tem sido feitos usando-se simulações de tráfego.

O objetivo desse trabalho é validar num ambiente real as simulações feitas anteriormente em [1]. Entretanto tal validação não pode ser estendida a qualquer Manet genérica. Essa validação apenas trás para um ambiente real um tráfego selecionado de um caso particular de Manet, pois como já dito, cada Manet possui seu tráfego particular dependente das aplicações rodadas.

Definimos 2 aspectos para caracterizar o tráfego na Manet: Controle e sinalização e aplicações executadas. O tráfego de controle e sinalização é gerado em UDP pelo protocolo de roteamento (i.e. OLSR) e também pelo tráfego ARP. Na simulação em [1], foi simulado tráfego de aplicações usuais em uma Manet. Na ocasião, para ser caracterizado um tráfego

representativo foi usado telnet, transferência de dados em rajada (FTP), transferência contínua de dados CBR (Video e audio-conferência) e ping.

Foi definido como cada uma dessas aplicações atuaria na simulação:

Telnet:

- Usa TCP;
- Tráfego bidirecional;
- Múltiplas sessões origem/destino;
- Tempo de início da seção: processo de Poisson;
- Duração da Seção: normalmente distribuída;
- Intervalo entre mensagens: processo de Poisson;

FTP:

- Usa TCP;
- Arquivo de tamanho aleatório;
- Múltiplas transferências origem/destino;
- Tamanho de arquivo: normalmente distribuído
- Tempo de início da seção: processo de Poisson;
- Duração da Seção: normalmente distribuída;



Transferência de dados CBR:

- Usa UDP;
- Taxa de 128kbps;
- Múltiplas transferências origem/destino;
- Tempo de início: processo de Poisson;
- Duração da seção: normalmente distribuída

Ping:

- Usa ICMP;
- 4 requisições espaçadas por 1 segundo;
- Envia resposta;
- Tempo de início: processo de Poisson;

Em [1], nas simulações em ambiente computacional também é introduzido um modelo de mobilidade para a Manet. Nesse trabalho não podemos introduzir esse modelo pois estamos trabalhando com um ambiente real (i.e. *Notebooks* conectados por uma rede ad-Hoc), assim deixamos de lado esse modelo devido a dificuldade em se ter mobilidade simulada em ambientes reais. Usaremos uma topologia física fixa que será discutida mais a frente que é capaz de simular multi-saltos entre máquinas diferentes.

### **2.1.3. Detecção de Ataques de DoS e Scanner de Portas**

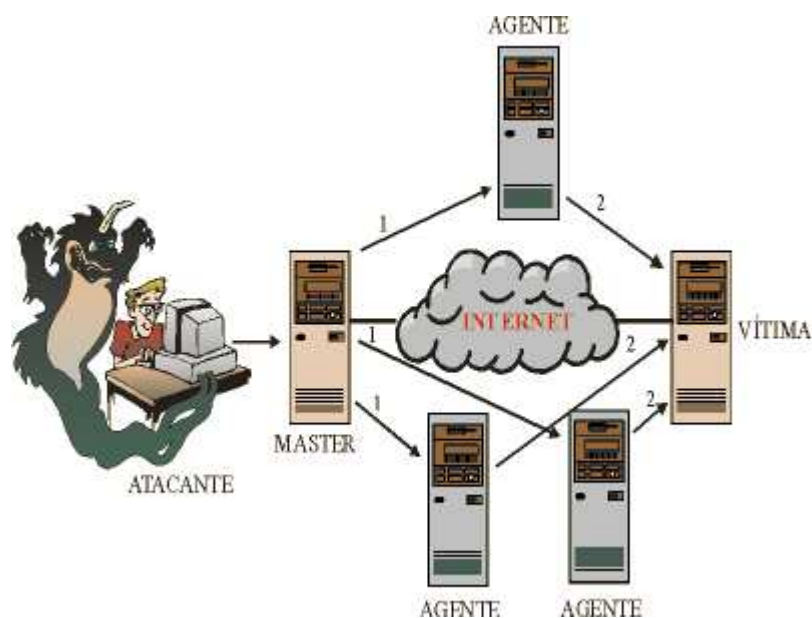
Os ataques DoS são bastante conhecidos no âmbito da comunidade de segurança de redes. Estes ataques, através do envio indiscriminado de requisições a um computador alvo, visam causar a indisponibilidade dos serviços oferecidos por ele.

Uma categoria de ataques de rede tem-se tornado bastante conhecida: a intrusão distribuída. Neste novo enfoque, os ataques não são baseados no uso de um único computador para iniciar um ataque, e sim de vários computadores lançando ataques coordenados a um alvo.

Os ataques DDoS são na verdade uma mistura do DoS com ataques distribuídos. Os ataques DDoS podem ser definidos como ataques DoS diferentes partindo de várias origens, disparados simultânea e coordenadamente sobre um ou mais alvos.

O ataque é dividido em duas fases: infecção de outros computadores com ferramentas que geram ataques e, posteriormente, inundação da rede com pacotes. O IDS idealizado nesse trabalho atuará na segunda fase apenas, detectando essa sobrecarga de pacotes.

A figura 2-3 abaixo ilustra como é feito o ataque DDoS. Nela o atacante tem controle do nodo master e de mais 3 agentes remotos, utilizando todos para atacar uma única vítima.



**Figura 2-3: Exemplo de ataque DDoS.**

(Fonte: <http://www.rnp.br> [14])

A tabela abaixo faz um resumo de algumas ferramentas usadas para esse tipo de ataque:

Ataque de DoS	Tipo de tráfego
Smurf	Inundação de pacotes ICMP echo-replay
Trinoo	Inundação de datagramas UDP em portas aleatórias
TFN e TFN2K	inundação de pacotes ICMP, UDM e TCP syn (flag syn setado); pacotes errôneos; smurf
TFN2K (ping flood)	inundação pacotes ICMP e smurf
TFN2K Targa 3	Pacotes IP inválidos
stacheldraht v.2.666	inundação de pacotes ICMP, UDP, TCP syn (flag syn setado), TCP null (nenhum flag setado), TCP ack (flag ack setado) e smurf
Shaft	inundação de pacotes ICMP, UDP, TCP syn (flag syn setado)
mstream	inundação de pacotes TCP ack (flag ack setado)

--	--

**Tabela 2-1: Ataques DoS**

(Fonte: F. Miziara [1], p. 57)

O ataque de scanner de portas também gera inundação de pacotes na rede aumentando o tráfego. Esse ataque pode ter uma abordagem distribuída ou não.

A tabela abaixo mostra o tipo de tráfego gerado por ataques scanner:

Tipo de Scanner	Tipo de tráfego
Scanner de portas TCP	Pedidos sucessivos de abertura de conexões TCP, em portas diferentes
Scanner de portas UDP	datagramas UDP sucessivos, em portas diferentes.

**Tabela 2-2: Tráfego gerado por Scanner de Portas**

(Fonte: F. Miziara [1], p. 58)

Seguindo o modelo de detecção proposto anteriormente, foram criados modelos separados para cada tipo de tráfego (i.e. TCP, UDP, ICMP e IP). Na MIB II [5] podemos monitorar as variáveis correspondentes a cada tipo de tráfego. Foi escolhido esse tipo de monitoramento por dois motivos: agentes SNMP são facilmente habilitados em computadores de forma padronizada e o módulo sensor pode ser utilizado. No capítulo 3 abordaremos o papel do módulo sensor no IDS implementado.

A tabela abaixo faz um resumo do tipo de tráfego, variáveis utilizadas para monitoramento e tipo de ataque detectável.

Modelo de Comportamento	Variáveis a serem monitoradas	Ataques possivelmente detectados
TCP	<ul style="list-style-type: none"> <li>- número/taxa de conexões ou entrantes</li> <li>- duração de uma conexão</li> <li>- tcpInErrs</li> <li>- tcpNoPorts</li> </ul>	<ul style="list-style-type: none"> <li>- TFN e TFN2K</li> <li>- stacheldraht</li> <li>- shaft</li> <li>- mstream</li> <li>- scanner de TCP</li> </ul>
UDP	<ul style="list-style-type: none"> <li>- udpInDatagrams</li> <li>- udpInErrs</li> <li>- udpNoPorts</li> </ul>	<ul style="list-style-type: none"> <li>- trino</li> <li>- TFN e TFN2K</li> <li>- stacheldraht</li> <li>- shaft</li> <li>- scanner de UDP</li> </ul>
ICMP	<ul style="list-style-type: none"> <li>- icmpInEchos</li> <li>- icmpOutEchos</li> <li>- icmpInErrs</li> </ul>	<ul style="list-style-type: none"> <li>- smurf</li> <li>- TFN (ping flood)</li> <li>- stacheldraht</li> <li>- shaft</li> </ul>
IP	<ul style="list-style-type: none"> <li>- ipReasmFails</li> </ul>	<ul style="list-style-type: none"> <li>- TFN2K (Targa 3)</li> </ul>

**Tabela 2-3: Modelo de comportamento, variáveis monitoradas e tipos de ataque.**

(Fonte: F. Miziara [1], p. 59)

#### **2.1.4. Caracterização do Modelo de Tráfego dos Ataques**

Caracterizar um modelo de ataque em uma rede Manet é complicado. Como já citado anteriormente quando estávamos descrevendo o modelo de tráfego da rede, não existe um perfil típico para ser utilizado.

Como feito anteriormente para caracterizar o tráfego na Manet, iremos também usar simulações de tráfego para tentarmos reproduzir o que foi feito em [1]. Para isso, utilizamos as ferramentas de geração de tráfego que serão descritas no capítulo 4. O tráfego com ataque simulado será correspondente ao tráfego normal adicionado do tráfego gerado pelo ataque.

Descrição dos ataques:

- DoS: Partindo-se de um nodo em direção ao nodo rodando o LIDS, adicionamos um tráfego UDP CBR (8Mbps).
- DDoS: Igualmente ao DoS, entretanto com 2 nodos (4Mbps cada) gerando o ataque no mesmo nodo rodando o LIDS.
- Scanner: Um nodo qualquer da rede fazendo 10 requisições TCP por segundo no nodo com o LIDS.

### 3.IMPLEMENTAÇÃO DO IDS

Na implementação apresentada neste trabalho houve reutilização de dois códigos. O primeiro código é um L-IDS completo, originalmente um IDS local, por assinatura, com características de projeto voltadas para aplicação em redes Manet, mostrado em [2]. O segundo software é um motor de detecção por comportamento, que foi construído de acordo com a teoria apresentada no capítulo 2 e testado em ambiente simulado, conforme [1].

Do primeiro software foi reaproveitada toda a arquitetura, incluindo a interface com usuário e interface para coleta de dados de auditoria. O segundo software foi inteiramente reaproveitado, ou seja, o motor de detecção por comportamento foi reutilizado.

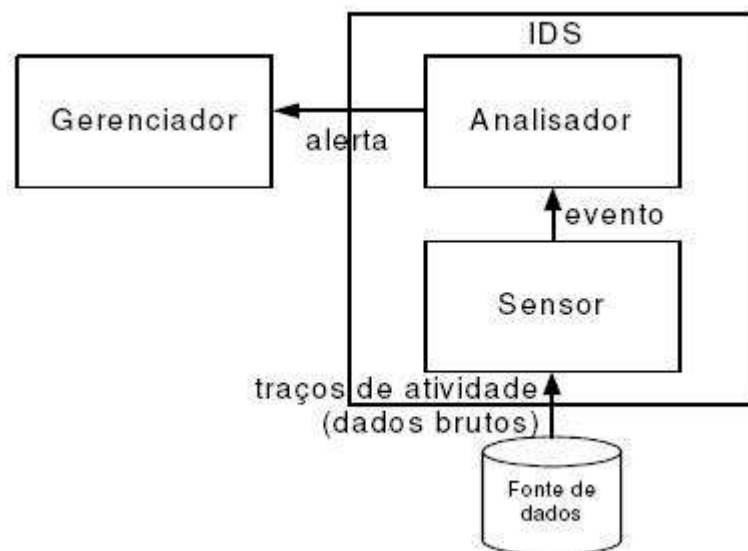
Assim, o desenvolvimento de software deste trabalho tratou basicamente da substituição do kernel do L-IDS no primeiro código. Acoplamos o segundo software ao primeiro constituindo um IDS compatível com o modelo de simulação de [1].

Os detalhes desse trabalho de desenvolvimento são mostrados nesse capítulo.

#### 3.1. ARQUITETURA DO L-IDS

Como dito anteriormente a arquitetura do L-IDS foi mantida e corresponde à figura 3-1. Esta arquitetura é uma adaptação do *framework* de detecção de intrusão proposto pelo grupo de trabalho sobre detecção de intrusão do IETF [15].

Ele é formado por sensores, analisadores e gerentes. Em cada L-IDS temos a seguinte modularização: o módulo sensor coleta dados a partir das fontes de dados de auditoria e os pré-processa, o módulo analisador processa esses dados para detectar situações que podem constituir-se violações da política de segurança e o módulo gerente realiza a interface de gerenciamento do processo de detecção, além de executar as tarefas de correlação de alertas e ativação de resposta automática às intrusões, caso exista. No nosso trabalho as facilidades de correlação de alertas e de resposta ativa à intrusão não foram implementadas.



**Figura 3-1 – Arquitetura L-IDS**

(Fonte: M. Wood e M.Erlinger [15], com adaptações)

### 3.1.1. **Framework de Detecção de Intrusão**

Os dados de auditoria coletados pelo L-IDS podem vir de diferentes fontes, tais como uma interface de captura de pacotes de rede (nível de rede), um sistema de *log* (nível de sistema) ou mesmo uma MIB (nível de rede, sistema ou aplicativo). Estes dados são usualmente brutos e têm pouca semântica. Além disso, o formato desses dados depende da sua fonte. Assim, um pré-processamento é aplicado a esses dados, transformando-os em mensagens semanticamente mais ricas para serem usadas pelo módulo analisador, denominadas eventos. Esta transformação realizada nos dados brutos é denominada abstração de eventos.

Devido aos fatos explicados nos parágrafos acima, o módulo sensor é decomposto em dois módulos neste trabalho: Extrator de Eventos e Coletor de Dados. Esses módulos separam as tarefas de recuperação de dados e de abstração de eventos em duas entidades diferentes. Isso possibilita múltiplas implementações para o coletor de dados, que podem operar paralelamente coletando dados de diversas fontes de auditoria. Da mesma forma, pode-se ter múltiplas implementações para o extrator de eventos, permitindo o uso simultâneo de diferentes técnicas de abstração.



No âmbito particular deste trabalho o módulo Coletor de Dados busca os dados de auditoria diretamente na MIB-II. O trabalho de abstração de eventos é simples, mas extremamente significativo. Ambas as atividades serão detalhadas posteriormente neste capítulo.

O analisador processa os eventos de acordo com alguma estratégia de detecção definida. Como abordado anteriormente, pelo menos duas metodologias de detecção de intrusão estão em discussão atualmente: detecção por uso incorreto e por desvio de comportamento (anomalia) [3]. De uma maneira geral, essas metodologias são complementares entre si. Na arquitetura do L-IDS, cada implementação de um algoritmo de detecção específico pode ser encapsulada em um módulo núcleo de IDS. É igualmente possível que se tenha múltiplas instâncias deste módulo, cada qual implementando um algoritmo de detecção específico.

No nosso trabalho não utilizamos de múltiplas instâncias de nenhum dos módulos. O L-IDS original tinha um motor de detecção por uso incorreto, o software resultante tem um novo motor de detecção por anomalia. Ou seja, houve completa reconstrução do módulo analisador, ao invés do uso de múltiplas instâncias.

O módulo gerente está relacionado com a interpretação dos alertas gerados pelo analisador e com a comunicação com outros L-IDS localizados em nodos vizinhos. Neste trabalho tais facilidades não foram exploradas, apesar de estarem presentes na arquitetura original do software, como concebido em [2].

### **3.1.2. Mensagens Geradas pelos L-IDS**

O L-IDS gera vários tipos de mensagens, que podem ser consumidas internamente ou compartilhadas remotamente. Essas mensagens estão completamente especificadas em [2]. Aqui abordaremos somente as mais relevantes dentro do escopo deste trabalho. Elas podem ser especificadas em termos das seguintes cláusulas:

- As mensagens referem-se a um conjunto de entidades do sistema, possivelmente vazio, que é formado por pares atacante↔alvo. Um atacante consiste de um conjunto, possivelmente vazio, de identificadores de entidades da rede que são consideradas como possíveis causadoras do ataque. Do mesmo modo, um alvo

consiste de um conjunto, possivelmente vazio, de identificadores de entidades da rede que são consideradas como possíveis afetados pelo ataque.

- Mensagens evento, consulta e alerta possuem os seguintes atributos: identificador, identificador da entidade de origem e um conjunto, possivelmente vazio, de pares atacante $\leftrightarrow$ alvo.
- Uma mensagem consulta periódica é um tipo especial da mensagem consulta, que é gerada automaticamente de maneira periódica. Esta mensagem possui um atributo adicional: período, indicando de quanto em quanto tempo tal mensagem será gerada.
- Mensagens alerta possuem um atributo adicional: identificador do ataque, identificando qual o ataque está sendo monitorado quando da geração destas mensagens.

As mensagens consulta estão associadas à coleta de dados brutos de auditoria e as mensagens evento estão relacionadas ao resultado da abstração desses dados. Os alertas resultam de análises efetuadas sobre os eventos.

## 3.2. ESCOLHAS DE PROJETO

Até aqui o IDS foi abordado sob a perspectiva dos trabalhos anteriores nos quais ele é baseado. O capítulo 2 explicou a teoria na qual se respalda o novo motor de detecção, enquanto este capítulo até aqui tratou da arquitetura que foi reutilizada na sua construção.

De agora em diante este capítulo abordará detalhes da nossa implementação.

Algumas escolhas de projeto foram necessárias na construção efetiva do software. Elas estão relacionadas a problemáticas inerentes à detecção por anomalia de comportamento e à arquitetura do software.

Tais escolhas tratam-se basicamente de:

1. Quais dados coletar para auditoria, mais especificamente quais variáveis da MIB-II utilizar;

2. Como promover a abstração de eventos sobre os dados escolhidos para coleta;
3. Como separar adequadamente as etapas de treinamento e de detecção do IDS;
4. Como construir os modelos de comportamento de rede;
5. Como escolher limiares de detecção apropriados para que o analisador gere o mínimo possível de alertas incorretos;

### **3.2.1. Escolha de dados de auditoria**

Para ter sucesso na detecção é necessário analisar dados que representem adequadamente o comportamento da rede. Obviamente, quanto mais variado o conjunto de dados, melhor a representação do sistema. Por outro lado, quanto maior o número de variáveis processadas, maior a complexidade de codificação e maior o consumo de processamento por parte do software.

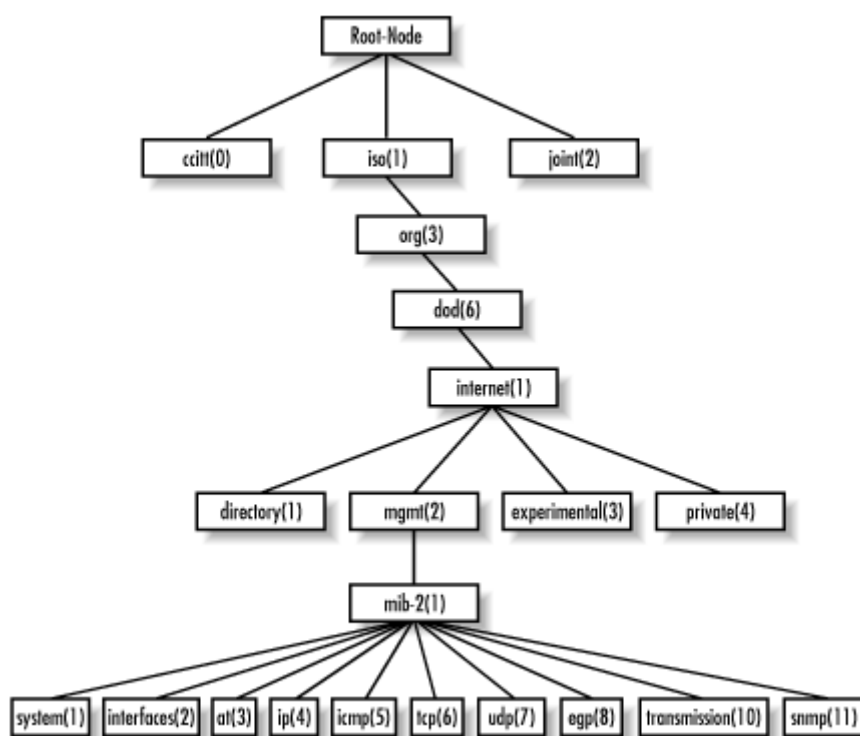
Dessa forma, apoiados em [1], escolhemos um conjunto de variáveis pequeno, mas que bem representa o comportamento da rede dentro do nosso objetivo: detectar ataques DoS, DDoS e Scanner de portas TCP.

Para entender bem o valor desse conjunto de variáveis é necessária uma compreensão mínima de como funciona protocolo de gerência SNMP e de como se organiza a MIB. Faremos então um breve estudo desses temas.

O SNMP (Simple Network Management Protocol) é um protocolo da camada de aplicação voltado para a gerência de redes. Ele permite aos administradores gerenciar o desempenho de rede, encontrar e resolver problemas de rede. Para que os dados de auditoria estejam disponíveis em um nodo é necessária a presença de um agente SNMP. Uma vez acionado, o agente pode então enviar dados de auditoria para o serviço que os requisitou.

Esses dados de auditoria são guardados na forma de um banco de dados. Esse banco de dados mantém um conjunto de variáveis escalares e tabelas conhecido como MIB (Management Information Base). Atualmente são usualmente utilizados três tipos de MIBs: MIB II, MIB experimental, MIB privada. Nosso IDS coleta variáveis escalares disponíveis na MIB-II, que é uma evolução da primeira MIB utilizada pelo protocolo SNMP.

Na MIB os objetos são organizados em uma hierarquia em árvore. Cada objeto é reconhecido por um OID (object identifier) único. Essa estrutura é a base do esquema de atribuição de nomes do SNMP. Um OID de objeto é composto por uma sequência de inteiros baseada nos nós das árvores, separada por pontos. Por exemplo, o objeto internet pode ser referenciado como iso.org.dod.internet ou como 1.3.6.1, conforme a figura 3-2 abaixo.



**Figura 3-2 – Exemplo do objeto 1.3.6.1**

(Fonte: [16])

Tendo em mente essas informações básicas sobre o protocolo SNMP e a organização da MIB-II, podemos novamente voltar o foco às nossas escolhas de projeto.

O processo de coleta de dados é simples. O módulo Analisador do IDS envia a requisição de quatro consultas periódicas ao módulo Sensor. Em consequência, o módulo sensor busca na MIB-II periodicamente quatro variáveis distintas. São elas:

- tcpInSegs

OID: 1.3.6.1.2.1.6.10

Representa o número de segmentos TCP recebidos até um dado momento, inclusive com erro. A análise desta variável é interessante para a detecção do ataque de Scanner TCP, pois na varredura de portas o nodo atacante envia um número anormal de segmentos ao nodo alvo.

- tcpPassiveOpens

OID: 1.3.6.1.2.1.6.6

Indica o número de conexões abertas passivamente no nodo até um dado instante. Essa variável também é interessante na análise do ataque Scanner TCP, pois a varredura de portas que encontrar portas TCP abertas causará alteração abrupta nessa variável.

- udpInDatagrams

OID: 1.3.6.1.2.1.7.1

Representa o número de datagramas UDP recebidos no nodo até um dado instante. A análise dessa variável é interessante na detecção dos ataques DoS e DDoS. Por motivos já mencionados esses ataques causam um aumento anormal no valor desta variável.

- ipForwardedDatagrams

OID: 1.3.6.1.2.1.4.6

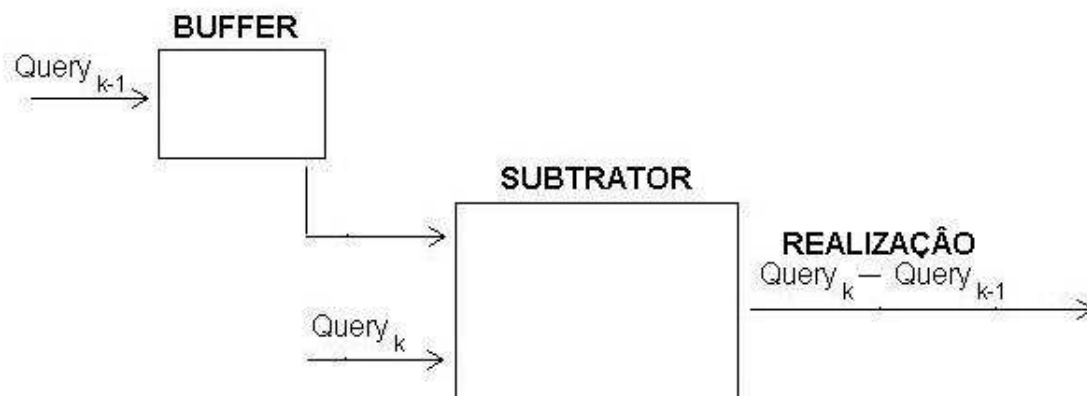
Indica o número de datagramas encaminhados com sucesso pelo nodo até um dado momento. Ou seja, datagramas recebidos pelo nodo que tinham outro nodo como destino e foram roteados. Note que essa variável é interessante para a detecção de todos os ataques e é especialmente útil em aplicações Manet. Sempre que o nodo no qual o IDS reside servir como rota para o nodo alvo, o ataque causará um aumento anormal nessa variável.

### **3.2.2. Processo de abstração de eventos**

Todas as quatro variáveis crescem indefinidamente durante o tempo de operação do sistema. Logo, o valor escalar das variáveis não representa adequadamente o comportamento da rede (não tem valor semântico).

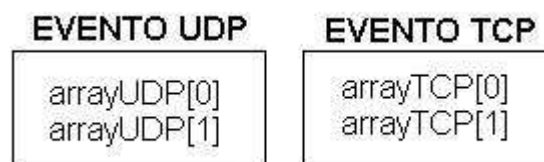
Estamos interessados em saber qual a variação dos escalares num dado intervalo de tempo, pois isso sim pode indicar anomalias no comportamento corrente da rede.

Assim, a extração de eventos consiste em subtrair-se do valor atual da consulta periódica corrente o seu valor precedente (consulta periódica precedente). Essa diferença obtida é chamada de realização. O processo é mostrado na figura 3-3:



**Figura 3-3 - Processo de abstração de eventos**

Essa subtração resulta em dois eventos distintos: um evento UDP e um evento TCP. Associado a cada evento há um array que guarda o valor das duas variáveis UDP e das duas variáveis TCP respectivamente.



**Figura 3-4 - Eventos resultantes**

Esses arrays possuem o nível de abstração adequado para serem diretamente tratados pelo módulo analisador.

É importante observar aqui que apenas as realizações não-nulas são consideradas na construção do modelo de comportamento de rede. O módulo extrator de eventos é responsável pela filtragem das realizações nulas.

### 3.2.3. Definição de fases de treinamento e detecção

Como explicado anteriormente, o IDS deve trabalhar em duas fases distintas: treinamento e detecção. É necessário, portanto, definir uma estratégia para coordenar essas duas fases.

No nosso software implementamos uma fase intermediária adicional, somando três fases distintas. A primeira fase corresponde ao treinamento, a segunda é uma fase de transição e a terceira é a detecção.

Ao ser carregado, o software começa a coletar dados na MIB-II e guarda as realizações em vetores definidos como vetores de treinamento. Após um número máximo de realizações pré-definido, o programa entra no período de transição, quando é gerado o modelo de comportamento normal, utilizando a modelagem GMM descrita no capítulo 2. É possível definir no IDS o número de realizações que irá compor esse modelo. Esse parâmetro é setado em uma variável dentro do kernel do programa. Também durante esse período de transição, os vetores de treinamento e o modelo gerado são salvos em disco. Dessa forma é possível compor novo treinamento para atualização ou reutilizar um treinamento anterior. Finalmente, passadas essas duas etapas, o IDS começa a verificar se as realizações atuais indicam um comportamento normal ou não, iniciando a fase de detecção.



**Figura 3-5 – Fases de atuação do IDS**

Há duas opções para se limitar o período de treinamento: pode-se limitar por tempo ou por número de realizações. Naturalmente, quanto maior o tempo de treinamento, maior a chance de se obter um modelo que represente adequadamente o comportamento normal da

rede. Da mesma forma, quanto maior o número de realizações que compõe o vetor de treinamento, maior a probabilidade de se obter um bom modelo.

É possível ocorrer uma situação em que mesmo um longo tempo de treinamento não defina modelo algum, caso o tráfego da rede seja nulo durante este tempo. Como dito anteriormente, as realizações nulas são descartadas durante a extração de eventos. Por isso, a nossa opção foi limitar a fase de treinamento por um número de realizações pré-determinado.

### **3.2.4. Construção do modelo estatístico**

Há dois vetores de treinamento distintos. Um guarda as realizações ligadas às variáveis UDP (`udpInDatagrams`, `ipForwardedDatagrams`) e o outro guarda as realizações ligadas às variáveis TCP (`tcpPassiveOpens`, `tcpInSegments`). Os vetores têm em cada posição os arrays da figura 3-4, correspondentes às realizações do período de treinamento.

Durante a fase de transição o IDS utiliza os dados contidos nesses vetores para a construção de dois modelos separados, um modelo TCP e um modelo UDP. Ambos usam a modelagem GMM definida no capítulo 2.

Gráficos e textos que descrevem esses modelos são gerados e salvos no disco, ao fim da fase de treinamento. Exemplos desses gráficos serão apresentados no capítulo 4.

### **3.2.5. Definição de limiares de detecção**

A cada realização o módulo analisador calcula a probabilidade de o comportamento corrente ser normal. Esta probabilidade é retornada por um método que é aplicado a todas as realizações, inclusive durante o treinamento. Vamos chamar esse método de `detection()`. O método `detection()` retorna a probabilidade numa escala de 0 a 1, onde 0 indica probabilidade máxima de ataque e 1 indica probabilidade mínima. O algoritmo de detecção implementa a modelagem apresentada em no tópico 2.1.1.5 do capítulo anterior.

É preciso definir limiares de decisão para separar comportamentos normais de comportamentos anormais, dentro da escala de 0 a 1. Apoiados em [1] definimos dois limiares de detecção: limiar de baixa probabilidade e limiar de alta probabilidade.

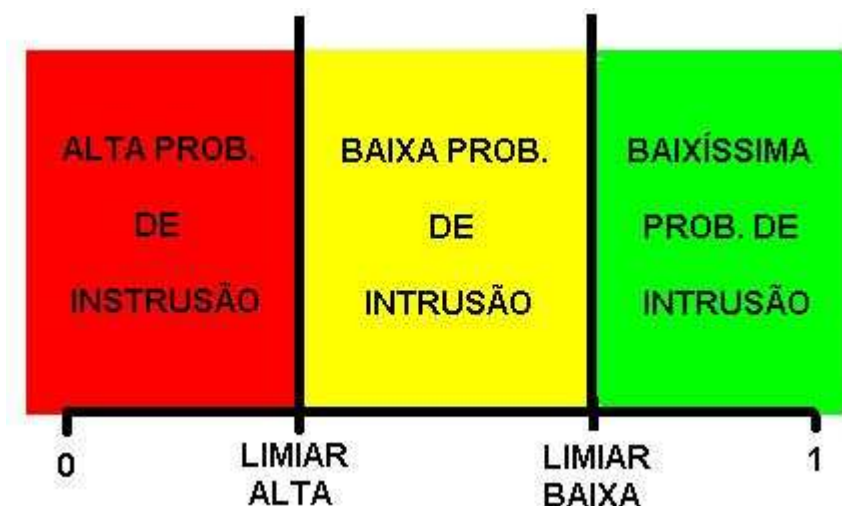


O limiar de alta probabilidade é igual ao menor valor retornado por `detection()` durante a fase de treinamento. Assim, se durante a detecção o método `detection()` retornar um valor menor que esse limiar, o IDS apontará alta probabilidade de intrusão.

O limiar de baixa probabilidade é igual à subtração da média dos valores retornados por `detection()` durante o treinamento pelo desvio-padrão do mesmo conjunto de valores. O IDS apontará baixa probabilidade de ataque caso `detection()` retorne um valor entre o limiar de baixa probabilidade e o limiar de alta probabilidade (maior que o limiar de alta e menor que o limiar de baixa).

O IDS apontará comportamento provavelmente normal se `detection()` retornar valor maior que o limiar de baixa probabilidade.

O resultado da interpretação do IDS baseada nos limiares é resumido na figura 3-6 abaixo:



**Figura 3-6 – Limiares de Detecção**

## 4.RESULTADOS EXPERIMENTAIS

Neste capítulo detalharemos as variáveis envolvidas no processo de teste de desempenho do software apresentado no capítulo 3. Na primeira parte mostramos os sistemas operacionais, configurações de hardware, geradores de tráfego dentre outros parâmetros do ambiente de rede. A segunda parte mostra os resultados obtidos nos testes além da análise desses resultados.

### 4.1. AMBIENTE DE TESTES.

Um ambiente de testes é algo complexo de se elaborar quando estamos validando uma simulação. Existem muitas variáveis do meio externo que influenciam no experimento e devem ser levadas em conta nos resultados obtidos. Para o ambiente proposto, devemos ter em mente os seguintes componentes: configuração dos *notebooks*, topologia da rede ad-hoc, geradores de tráfego, ferramentas de ataque, configuração do protocolo de roteamento e do agente SNMP.

#### 4.1.1. *Notebooks.*

Nesse experimento, trabalhamos com quatro *notebooks*. Três desses *notebooks* foram configurados com um ambiente Linux Fedora Core 5 e um deles com Windows XP Professional SP2. Devemos a partir deste momento separar esses *notebooks* por nomes distintos para que no decorrer da apresentação dos resultados obtidos seja mais fácil entender o funcionamento da implementação.





O *Notebook-1* foi configurado com sistema operacional Linux Fedora 5, OLSRD 0.4.10, Net-SNMP 5.3.1, JRE 1.5.0, Eclipse 3.2.2 e estará rodando o L-IDS. Seu processador é um Intel Centrino com interface wireless integrada Intel PRO/Wireless 3945ABG.

O *Notebook-2* foi configurado com sistema operacional Linux Fedora 5, OLSRD 0.4.10, Net-SNMP 5.3.1 e JRE 1.5.0. Seu processador é um Intel Centrino com interface wireless integrada Intel PRO/Wireless 2200BG.

O *Notebook-3* foi configurado com sistema operacional Linux Fedora 5, OLSRD 0.4.10, Net-SNMP 3.2.2, JRE 1.5.0. Seu processador é um Intel Pentium 4 com interface wireless 802.11B externa PCMCIA Avaya.

O *Notebook-4* foi configurado com sistema operacional Windows XP Professional SP2, com serviço de SNMP ativado, OLSRD 0.4.10, JRE 1.6.0. Seu processador é um Intel Centrino com interface wireless integrada Intel PRO/Wireless 2915ABG.

Para facilitar a visualização a tabela abaixo resume cada *notebook*:

 <p><b>Notebook-1</b></p> <ul style="list-style-type: none"> <li>- Linux Fedora 5</li> <li>- OLSRD 0.4.10</li> <li>- Net-SNMP 5.3.1</li> <li>- JRE 1.5.0</li> <li>- Eclipse 3.2.2</li> <li>- L-IDS</li> <li>- Intel 3945ABG</li> </ul>	 <p><b>Notebook-3</b></p> <ul style="list-style-type: none"> <li>- Linux Fedora 5</li> <li>- OLSRD 0.4.10</li> <li>- Net-SNMP 5.3.1</li> <li>- JRE 1.5.0</li> <li>- Avaya 802.11B</li> </ul>
 <p><b>Notebook -2</b></p> <ul style="list-style-type: none"> <li>- Linux Fedora 5</li> <li>- OLSRD 0.4.10</li> <li>- Net-SNMP 5.3.1</li> <li>- JRE 1.5.0</li> <li>- Intel 2200BG</li> </ul>	 <p><b>Notebook-4</b></p> <ul style="list-style-type: none"> <li>- Windows XP SP2</li> <li>- OLSRD 0.4.10</li> <li>- JRE 1.6.0</li> <li>- SNMP Ativado</li> <li>- Intel 2915ABG</li> </ul>

**Tabela 4-1 – Resumo das Configurações.**

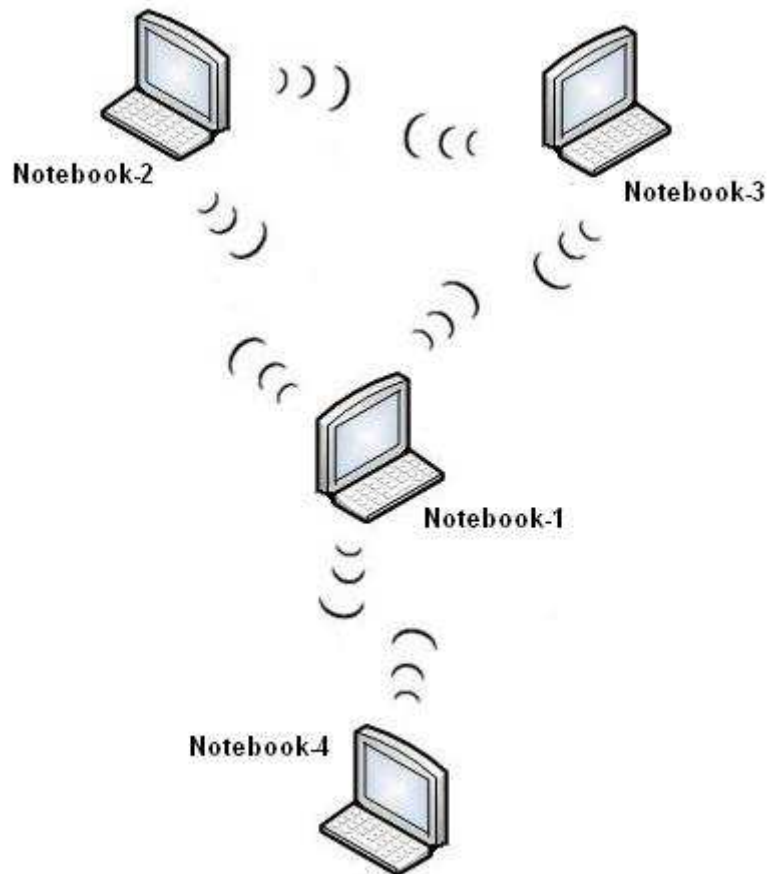
#### 4.1.2. Topologia da Rede *ad-Hoc*.

O endereçamento dos nodos (i.e. *Notebooks*) da rede foram feitos da seguinte forma:

<b>Nome:</b>	<b>Endereço IP:</b>
<i>Notebook-1</i>	192.168.1.1
<i>Notebook-2</i>	192.168.1.2
<i>Notebook-3</i>	192.168.1.3
<i>Notebook-4</i>	192.168.1.4

**Tabela 4-2 – Endereçamento IP**

Fisicamente a topologia montada foi configurada de forma que o *Notebook-1* ficasse no centro e pudesse alcançar todos os outros, os *Notebook-1* e *Notebook-2* pudessem comunicar-se entre si e com o *Notebook-1* diretamente e o *Notebook-4* pudesse apenas se comunicar diretamente com o *Notebook-1* e indiretamente (através do *Notebook-1*) com o *Notebook-2* e *Notebook-3*. A figura 4-1 abaixo ilustra a topologia:



**Figura 4-1 – Topologia da rede ad-Hoc**

Para que pudéssemos ter essa topologia, que simula saltos devido à distância do enlace sem fio, foi necessário fazer com que o *Notebook-2* e o *Notebook-3* não recebessem pacotes do *Notebook-4*. Para isso usamos o IPTABLES do Linux configurado em 2 e 3 conforme o comando a seguir:

```
# iptables -A INPUT -m mac --mac-source 00:13:CE:E6:32:B7 -j DROP
```

Onde 00:13:CE:E6:32:B7 é o endereço MAC da interface sem fio do *Notebook-4*.

Desta forma, é como se o *Notebook-4* estivesse fisicamente distante do *Notebook-2* e do *Notebook-3* (ou seja, sem comunicação direta com enlace wireless) simulando um salto e o protocolo OLSR faria com que as informações a esses destinos passassem pelo *Notebook-1*.

#### **4.1.3. Protocolo de roteamento OLSR.**

O OLSR (Optimized Link State Routing) é um protocolo padronizado pelo IETF (Internet Engineering Task Force) através de RFC 3626, muito utilizado atualmente na construção de redes ad-Hoc [17]. O OLSR é um protocolo pró-ativo, ou seja, possui uma tabela de roteamento ativa e constantemente atualizada que é modificada quando acontece a movimentação de nodos. No protocolo OLSR são selecionados dinamicamente, conforme o crescimento da rede, alguns nós vizinhos que são denominados MPR's (Multi Point Relays) onde, através deles o roteamento envia informações que têm como destino nós mais distantes, diminuindo assim o tráfego na rede e a colisão de informações na camada de transporte.

O OLSR pode também ser usado com algumas opções diferentes das estabelecidas em [18]. Neste trabalho usamos como parâmetro para escolha de rota não somente os saltos, mas também a qualidade do link anunciada por cada nodo. Tal escolha foi usada, apesar de fugir um pouco da compatibilidade estabelecida por [18], pois em determinados momentos o salto pode ser um caminho melhor que um link direto, dependendo da qualidade do mesmo.

Para a implementação do protocolo em nossa rede ad-Hoc, utilizamos o *daemon* OLSRD disponível em [17]. É um *daemon* de padrão aberto e bastante customizável. Ele possui um arquivo `olsrd.conf` onde são colocados os parâmetros de configuração a serem implementados. As configurações desse arquivo estão presentes no ANEXO I.

#### **4.1.4. Net-SNMP.**

O Net-SNMP é uma coleção de utilitários e bibliotecas para se usar o SNMP (Simple Network Management Protocol) [19]. A história do Net-SNMP vem desde o começo do SNMP, criado como um padrão a ser usado ao se tratar informações de gerência de rede, conforme descrito em [20].

Para a utilização do agente SNMP do Net-SNMP, existe um *daemon* que deve ser executado, o `snmpd`. As configurações desse *daemon* ficam armazenadas no `snmp.conf`. Essa configuração pode ser observado no ANEXO II.

#### 4.1.5. Geradores de Tráfego.

A geração de tráfego para realização de simulações de tráfego real pode ser considerada desafiadora. Por mais controlado que seja o ambiente, não podemos controlar fatores externos como, por exemplo, a qualidade do enlace sem fio, que depende do meio e das possíveis interferências que podem estar acontecendo.

A geração realizada nesse trabalho foi feita usando-se alguns programas disponíveis abertamente para download na Internet. Usamos programas que fossem capaz de gerar tráfego UDP e TCP para podermos simular aplicações do tipo FTP, videoconferência, audioconferência e telnet.

O principal programa utilizado foi o Packit versão 1.0-1 para Linux. Ele é uma ferramenta de auditoria que permite que se monitore, manipule e injete tráfego IPv4 customizado na rede. É recomendado para teste de firewalls, intrusion detection systems, port scanning, e auditoria geral do TCP/IP. Atualmente possui a habilidade de definir praticamente todas opções de cabeçalhos do TCP, UDP, ICMP, IP, e Ethernet. Usa o libnet 1.1 ou mais novo e também o libpcap [21].

Além do Packit, usamos também o IPERF 2.0.2. Esse é uma ferramenta para medir a throughput de uma rede. Funciona injetando pacotes TCP ou UDP, e então mede os delays, o jitter etc. Para utilizar, deve-se em um computador executar o modo servidor e em outro em modo cliente. É possível mandar CBR (Constant Bit Rate) com velocidades pré-programadas [22].

Outro programa utilizado foi o GeradorTCP, um programa simples que nós implementamos em Java para simular abertura de portas TCP. Isso simula a abertura de conexões Telnet, FTP etc. Também pode ser usado para simular um ataque scanner pois tem a opção de abrir várias vezes uma porta num determinado intervalo de tempo randômico, fazendo com que a variável TcpPassiveOpens se altere bruscamente.

Para a realização da geração de tráfego, devemos ter essas ferramentas como variáveis a serem controladas, bem como a geração UDP fornecida pelo OLSRD e a ICMP pelo PING.

Para a geração do tráfego TCP, precisávamos gerar dois padrões diferentes de tráfego, para que dessa forma o *kernel* do LIDS pudesse reconhecer dois núcleos distintos como se

fossem duas aplicações TCP distintas. Para realizar tal geração, usamos o Packit em conjunto com o GeradorTCP.

Primeiramente com o Packit, durante a primeira metade do treinamento, geramos um tráfego que fornecia para o *Notebook-1*, a cada 2 segundos uma rajada de 7 pacotes alternando com uma outra rajada de 3 pacotes a cada segundo. Esse tráfego foi gerado usando-se os comandos abaixo:

```
# packit -m inject -t TCP -c 0 -w 2 -b 7 -h -i dev10700 -d 192.168.1.1
```

```
# packit -m inject -t TCP -c 0 -w 1 -b 3 -h -i dev10700 -d 192.168.1.1
```

Onde `-m` indica o modo utilizado, no caso injetar pacotes, `-t` indica o protocolo utilizado, o TCP, `-w` é o intervalo de espera entre cada rajada enviada, `-b` o número de pacotes injetados em `-w` segundos, `-i` é a interface usada para injetar os pacotes e `-d` é endereço do destino .

Também nessa primeira parte de treinamento, rodamos o GeradorTCP, de forma que o mesmo fizesse 1 requisição de abertura de porta a cada 10 segundos randômicamente, ou seja, entre o período de 0 a 10 segundos, o programa randômicamente (distribuição uniforme) escolhia um tempo para fazer a requisição de abertura. O comando utilizado pelo GeradorTCP está descrito abaixo:

```
# java GeradorTCP 192.168.1.1 22 10 1
```

Onde 192.168.1.1 é o endereço de destino, 22 é o número da porta TCP, 10 é o período em segundos e 1 é o número de requisições a serem feitas no período dado.

Para a segunda metade do treinamento deveríamos alterar o padrão de tráfego para que outro núcleo fosse reconhecido pelo LIDS como usual. Então alteramos as configurações do Packit para uma rajada de 70 pacotes a cada 2 segundos e mantemos a outra rajada de 3



pacotes a cada segundo. O GeradorTCP, foi alterado para fazer 2 requisições de abertura de porta usando o mesmo tipo de intervalo de tempo randômico. Os comandos abaixo exemplificam o que foi feito:

```
# packit -m inject -t TCP -c 0 -w 2 -b 70 -h -i dev10700 -d 192.168.1.1
```

```
# packit -m inject -t TCP -c 0 -w 1 -b 3 -h -i dev10700 -d 192.168.1.1
```

```
# java GeradorTCP 192.168.1.1 22 10 2
```

O tráfego UDP também deve seguir os mesmos requisitos do TCP, ou seja, uma geração diferenciada de tráfego que simule pelo menos duas aplicações UDP na rede que por fim geram dois núcleos distintos de tráfego no LIDS.

A geração de tráfego UDP foi feita usando o Packit em conjunto com o IPERF. O IPERF nesse caso serviu como servidor para que o Packit gerasse datagramas em uma porta UDP do destino.

Primeiramente, na metade inicial do treinamento, não geramos nenhum tráfego com os programas, deixando apenas com que o OLSR gerasse o tráfego UDP necessário para a formação de um núcleo no LIDS.

Nas metade final, usamos o conjunto Packit + IPERF na geração de um tráfego UDP para a formação do segundo núcleo. Para isso usamos o comando abaixo:

```
# packit -m inject -t UDP -d 192.168.1.1 -c 0 -b 25 -w 1 -i dev10700 -D 5001
```

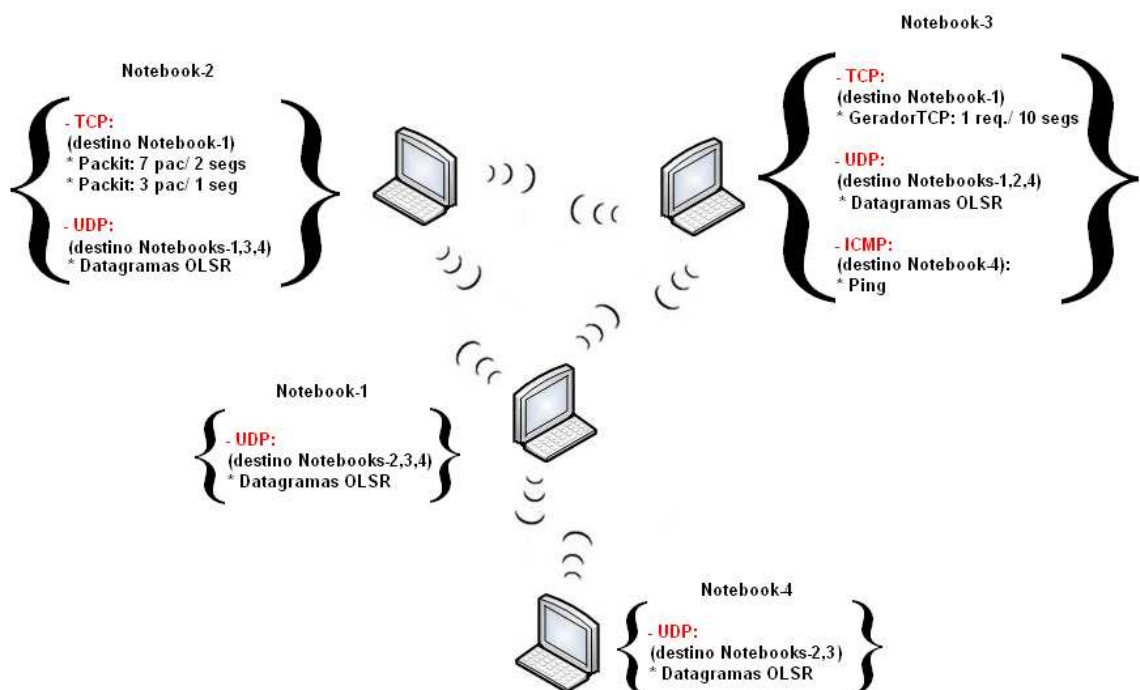
```
# iperf -s -u
```

Onde no comando do IPERF, o -s indica modo servidor e -u o modo UDP. Por padrão, o IPERF abre a porta 5001, que foi usada como destino -D no comando do Packit.

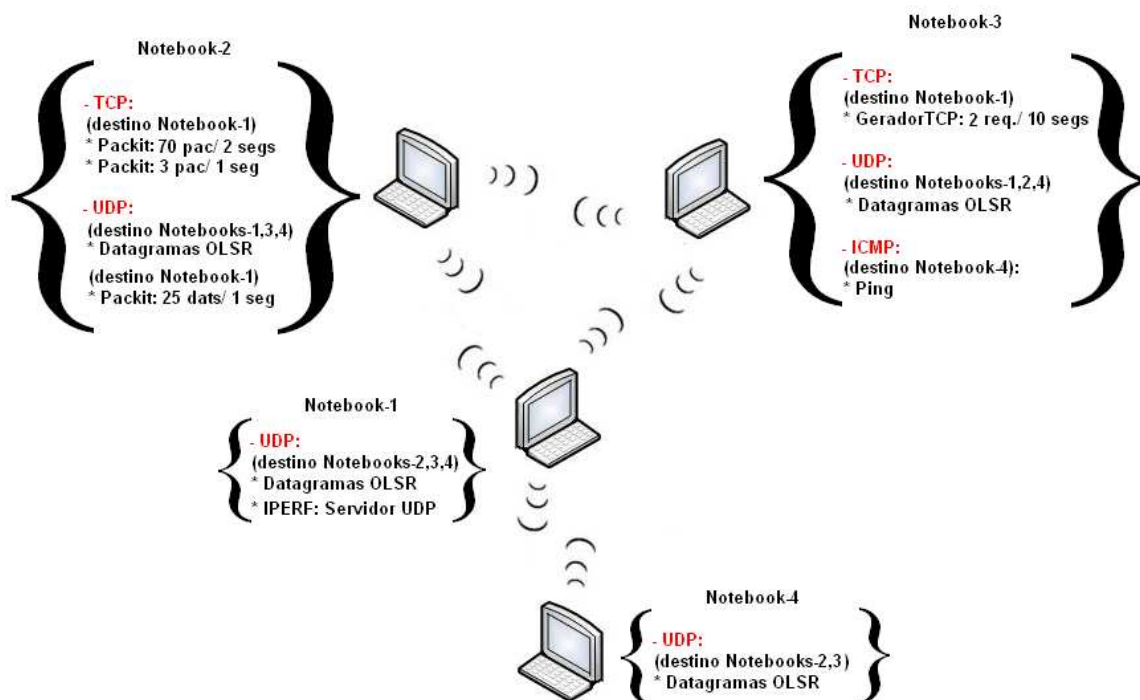
Além de tráfego TCP e UDP, devemos também gerar tráfego ICMP com destino a uma máquina qualquer mas que passe pelo *Notebook-1*, pois dessa forma a variável *IpFoward* pode ser modificada. Para isso usamos o PING:

```
#ping 192.168.1.4
```

Para entender como todo esse tráfego foi gerado em nossa topologia, as figuras 4-2 e 4-3 abaixo mostram os tráfegos gerados na primeira metade do treinamento e na segunda metade:



**Figura 4-2 – Tráfego na primeira metade do treinamento.**



**Figura 4-3 – Tráfego na segunda metade do treinamento**

#### 4.1.6. Ferramentas de Ataque.

Para simular ataques DoS, DDoS e Scanner, utilizamos algumas ferramentas disponíveis na Internet. Para ataques *DoS* e *DDoS* usamos o UDP Flood 2.0. Para o ataque do tipo *scanner*, usamos o Flash Port Scanner 1.0.

O UDP Flood é um programa que gera datagramas UDP com origem aleatória para um destino a uma taxa controlável. Na simulação usamos 200 dats/segundo [23].

O Flash Port Scanner é um programa fácil de usar que simplesmente varre um host para saber quais portas estão abertas indicando-as. Existe a opção para escolher o range de portas e o endereço IP de destino[24].

Para realizarmos o ataque DoS, usamos o UDP Flood no *Notebook-2*, atacando o *Notebook-1* que estava rodando o LIDS. Também fizemos um ataque do *Notebook-2* para o *Notebook-4*, passando pelo *Notebook-1* para que o mesmo também o detectasse.

O ataque DDoS foi feito de maneira semelhante ao DoS, porém com dois nodos realizando ataques, no caso o *Notebook-4* e o *Notebook-2* atacando o *Notebook-1* e posteriormente atacando o *Notebook-3* passando pelo *Notebook-1*.

No ataque *scanner*, procedemos da mesma maneira do ataque DoS, fazendo um *scanner* partindo do *Notebook-4* para o *Notebook-1* e depois para o *Notebook-2* passando pelo *Notebook-1*.

Para efeito de comparação com as simulações em [1], realizamos uma outra bateria de testes com ataques simulados com os geradores de tráfego. Mantemos a coerência com o capítulo 2 deste texto: usamos 8Mbps nos ataques DoS; 4Mbps nos ataques DDoS partindo de cada nodo atacante e 10 requisições de abertura de porta por segundo para ataque Scanner. Os nodos atacantes e alvo foram escolhidos na forma descrita nos parágrafos acima.

## 4.2. TESTES E ANÁLISE DE RESULTADOS

O IDS implementado constrói dois modelos de comportamento de rede na fase de transição: um modelo TCP e outro UDP. Isso faz com que os tráfegos TCP e UDP sejam discriminados na análise de detecção. Por conta disso, devemos ter geração de tráfego e análise de desempenho também distintas, para TCP e para UDP.

Como explicado no capítulo 3, temos quatro variáveis para auditoria de dados. A análise de duas delas forma o modelo TCP: *tcpInSegments* e *tcpPassiveOpens*. As outras duas variáveis, *udpInDatagrams* e *ipForwardedDatagrams*, formam juntas o modelo UDP.

Pode-se observar que em linha gerais o modelo TCP servirá para a detecção de ataque Scanner de portas TCP, enquanto o modelo UDP servirá para detecção de ataques DoS, DDoS e ataques que estejam sendo encaminhados pelo nodo.

Seguindo a idéia contida em [1] de encontrar a periodicidade ideal de realização de consultas das variáveis de auditoria, fizemos testes variando a periodicidade de amostragem. Os períodos foram: 2, 3, 4, 8, 12 e 24 segundos.

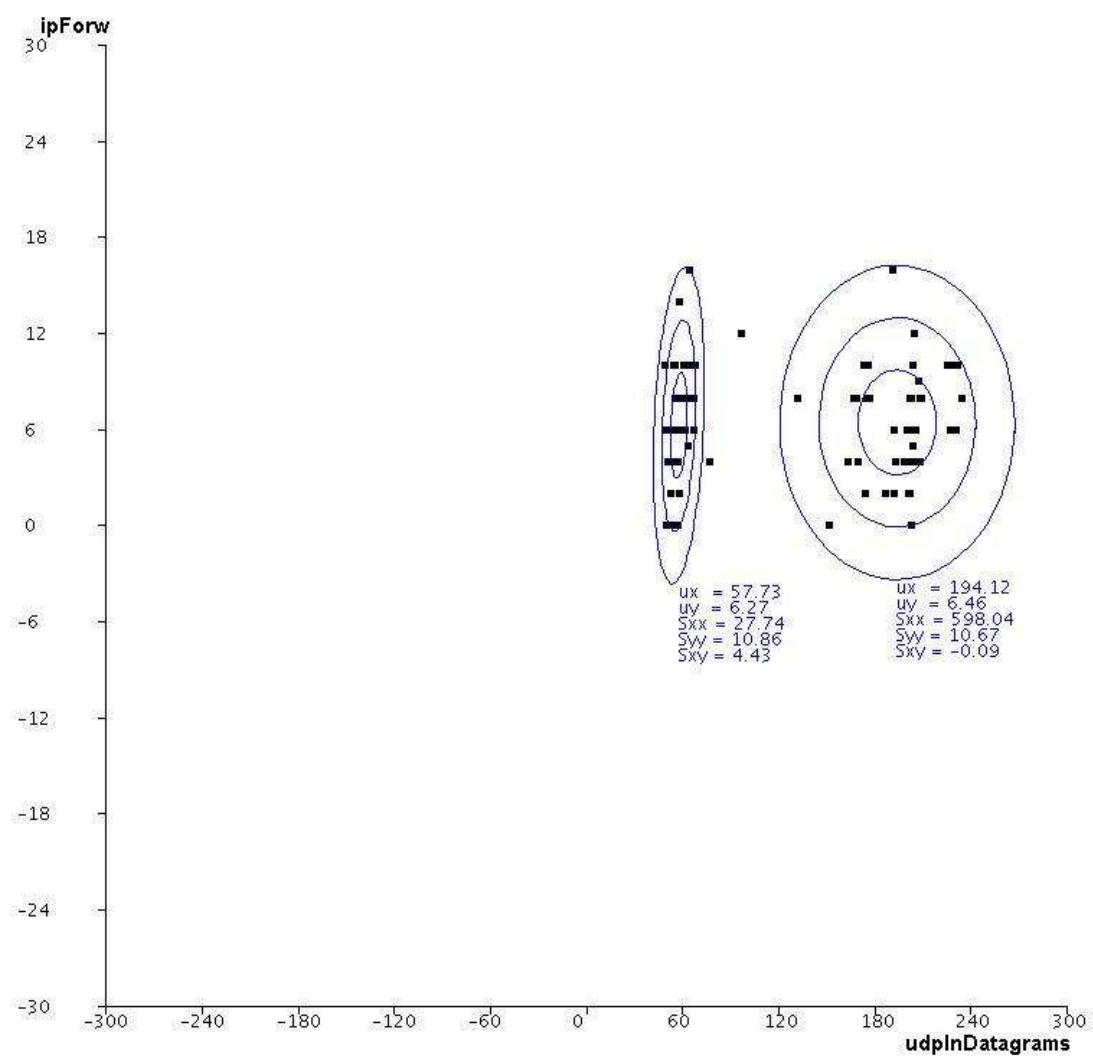
Para cada período desses foi realizado um treinamento para ajuste do GMM. Com o objetivo de prover um período de treinamento similar para todos os períodos de amostragem,

variamos proporcionalmente o número de realizações que compõem o treinamento de cada período. Por exemplo, com intervalo de polling de 3 segundos as realizações de treinamento foram 100, para 2 segundos 150, para 4 segundos 75 realizações etc.

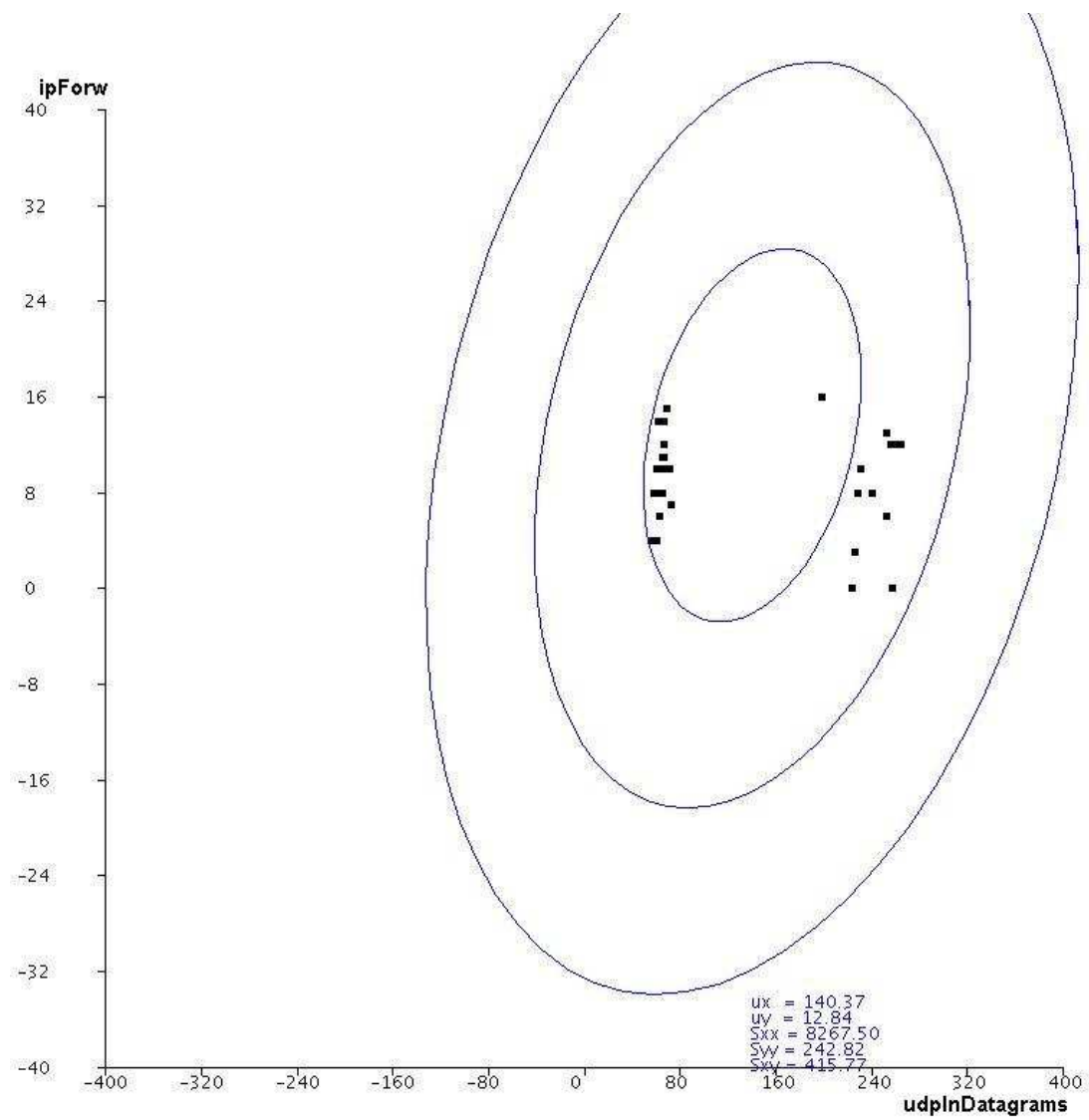
#### **4.2.1. Modelo UDP**

As figuras 4-4, 4-5 e 4-6 a seguir mostram o resultado do treinamento para as simulações com intervalo de polling de 3, 8 e 24 segundos para o modelo UDP. Os gráficos mostram as realizações que compõem os treinamentos e a formação de clusters, quando ocorrida. O eixo horizontal representa a alteração em `udpInDatagrams` e o eixo vertical a alteração em `ipForwardedDatagrams`, no período dado. Além disso, cinco dados estatísticos são apresentados: média do eixo x, média do eixo y, variância do eixo x, variância do eixo y e covariância entre x e y, respectivamente.

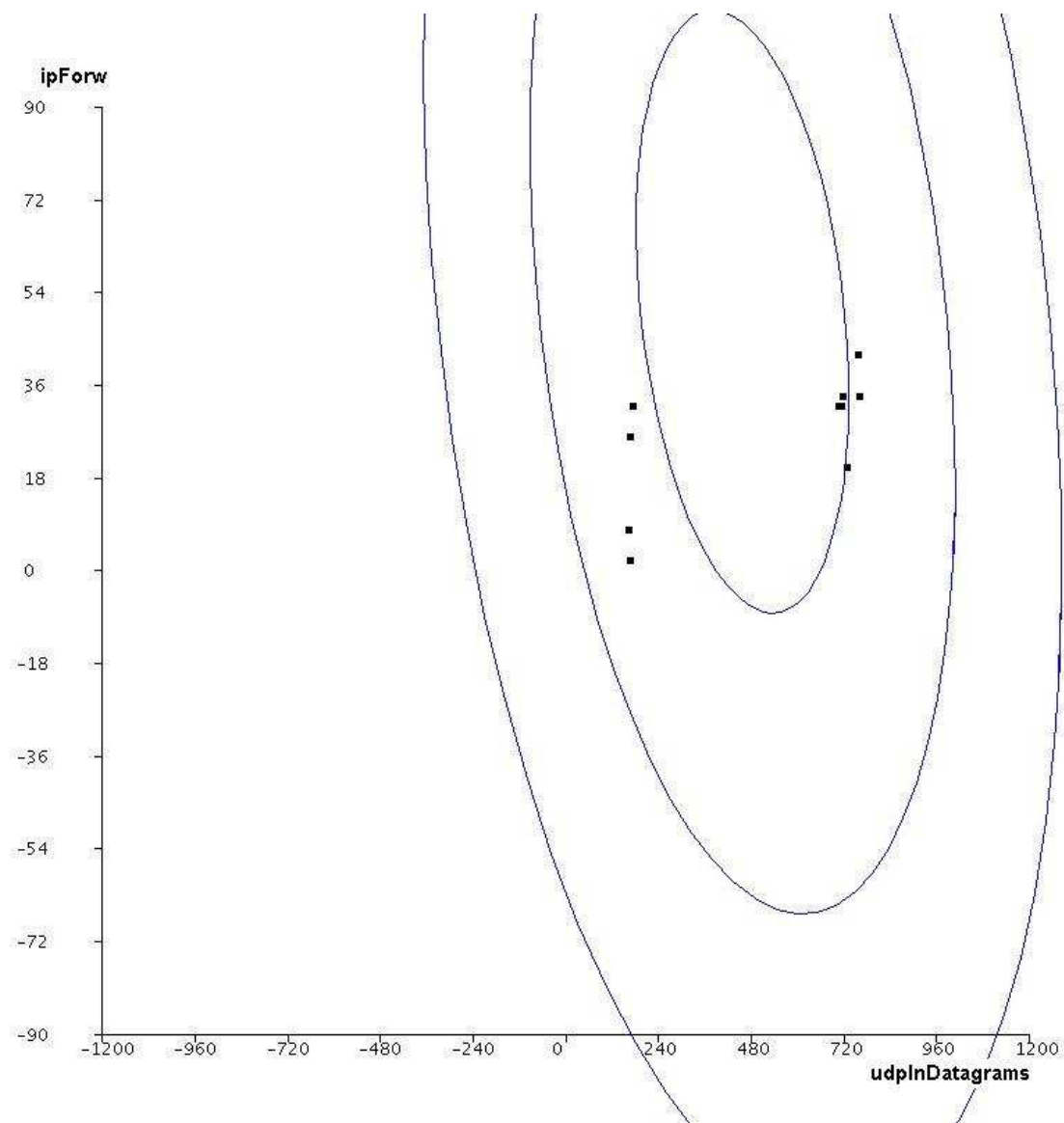
Naturalmente, os clusters estão relacionados com os dois padrões de tráfego UDP descritos anteriormente. O cluster de menor média leva a contribuição do tráfego icmp (ping) e do tráfego de roteamento (olsr). O outro cluster leva em conta, adicionalmente, o tráfego de datagramas do packit. A clusterização para UDP fica clara no caso de periodicidade de 3 segundos.



**Figura 4-4 – Modelagem UDP com intervalo de polling de 3 segundos**



**Figura 4-5 – Modelagem UDP com intervalo de polling de 8 segundos**



**Figura 4-6 – Modelagem UDP com intervalo de polling de 24 segundos**



Pode-se observar que para os três períodos apresentados apenas o de 3 segundos ocasionou em formação de clusters, ou seja, resultou em um ajuste GMM com dois *kernels* (somatório de duas funções separadas), como descrito no capítulo 2. No entanto, as realizações plotadas nos gráficos mostram que dois padrões de tráfego diferentes foram sempre utilizados. De fato, o tráfego gerado foi o mesmo em todos os testes, desconsiderando as imperfeições de um ambiente real. Isso indica que o reconhecimento adequado dos padrões de tráfego na modelagem é influenciado pelo intervalo de polling, nesse estágio de desenvolvimento do software. Além disso, a diminuição no número de realizações deve ter influenciado negativamente esse reconhecimento.

#### 4.2.1.1. Detecção de ataques

Lembramos que os resultados aqui apresentados correspondem à avaliação de um IDS localizado em um dos nodos participantes da rede (i.e. *Notebook-1*), que pode ser alvo do ataque ou servir como roteador entre o atacante e o alvo.

Falso positivo representa a detecção de ataque em um momento em que não está sendo realizado o ataque. Falso negativo significa a não detecção de ataque durante um ataque.

Os testes de falso positivo foram realizados mantendo o tráfego da rede dentro dos mesmos padrões utilizados no treinamento. Os testes de falso negativo foram realizados de quatro formas distintas: DoS utilizando o software UDP Flood, DoS utilizando IPERF com 8Mbps, DDoS com UDP Flood e DDoS com IPERF com 4 Mbps por nodo atacante.

Para todos os intervalos de polling a duração dos testes de detecção foi de 100 avaliações, ou seja, 100 realizações não nulas.

Intervalo de Polling	Falso positivo	Falso negativo (nodo alvo)	Falso negativo (nodo roteador)
2 segundos	0%	0%	0%

3 segundos	0%	0%	0%
4 segundos	0%	0%	0%
8 segundos	0%	0%	0%
12 segundos	0%	0%	0%
24 segundos	0%	0%	0%

**Tabela 4-3 – Taxas de falso positivo e falso negativo para ataque DoS e DDoS com detecção de alta probabilidade**

Intervalo de Polling	Falso positivo	Falso negativo (nodo alvo)	Falso negativo (nodo roteador)
2 segundos	2%	0%	0%
3 segundos	16%	0%	0%
4 segundos	2%	0%	0%
8 segundos	0%	0%	0%
12 segundos	12%	0%	0%
24 segundos	0%	0%	0%

**Tabela 4-4 – Taxas de falso positivo e falso negativo para ataque DoS e DDoS com detecção de baixa probabilidade**

Os resultados obtidos nos testes de ataque foram positivos. Se avaliarmos apenas detecção de alta probabilidade, vemos que não houve ocorrência de erros de avaliação. Isso decorre do fato de estarmos com um tráfego específico e numa rede também bastante específica. De fato, os ataques DoS e DDoS realizados com o software UDP Flood, tendo o IDS localizado tanto no nodo alvo como em um nodo roteador, foram sempre detectados. A mesma coisa ocorreu para os ataques simulados com os geradores de tráfego, a detecção ocorreu em todas as 100 avaliações. Isso indica que o domínio de dados está excessivamente restrito e por isso qualquer variação pode ser facilmente identificada. Obtivemos, entretanto, falsos positivos de baixa probabilidade.

Obviamente, tal taxa de acertos não é natural. Em uma rede Manet real, o comportamento mais natural é que ocorram desvios normais nos padrões de tráfego que possam ocasionar em avaliações errôneas do IDS, mesmo com um treinamento bem realizado. No entanto, o fato de usarmos um ambiente controlado para testes faz com que tal resultado seja possível.

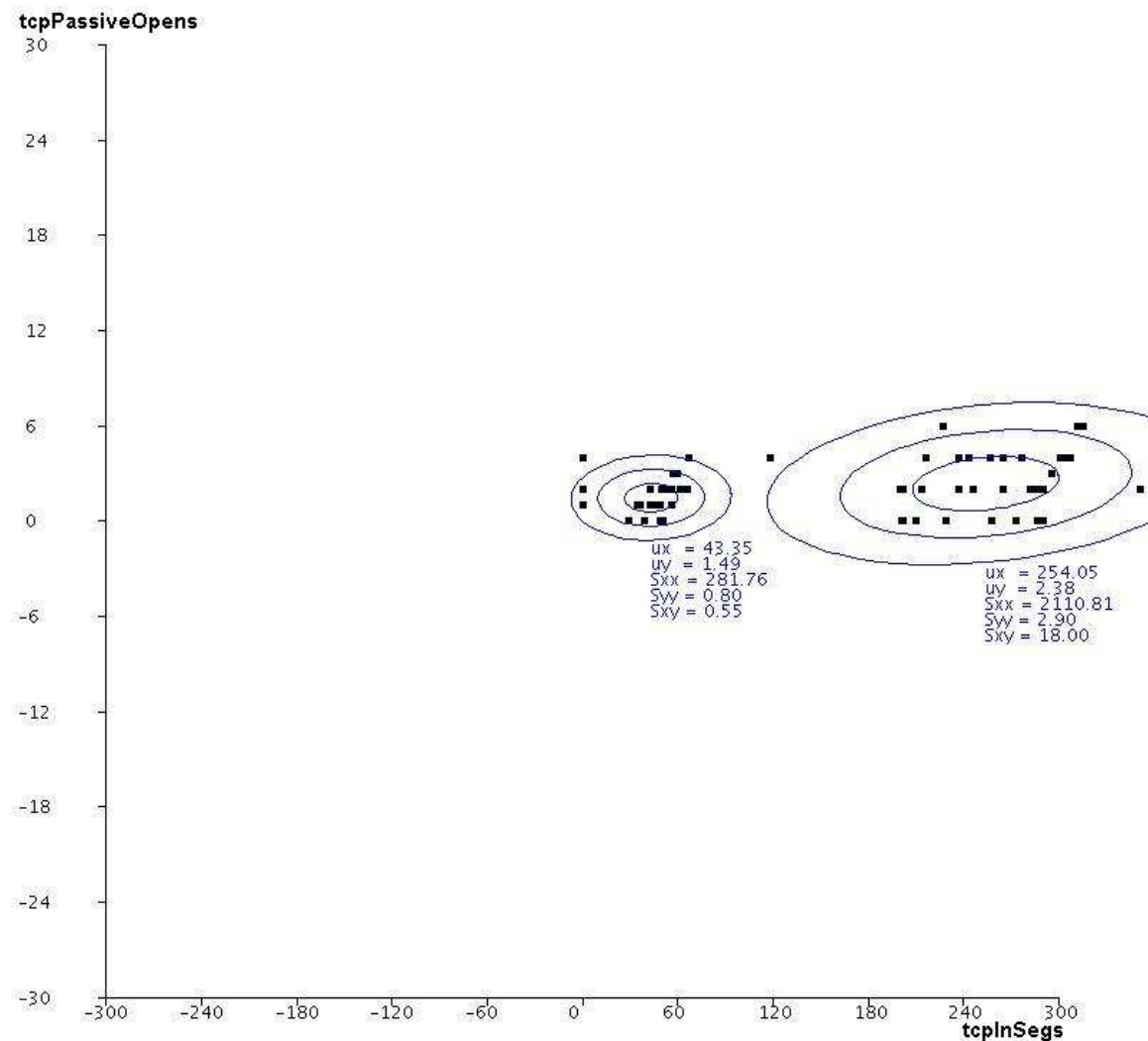
Se estudarmos mais as fundo os resultados, levando em conta os gráficos de representação dos modelos e os resultados obtidos, vemos que há uma solução de compromisso entre a taxa de falso positivo e a qualidade na percepção do tráfego. Em linhas gerais, quanto mais bem definido o tráfego, maior a possibilidade de haver falsos positivos, pois podem ocorrer pequenos desvios daquele padrão. É notório que para o período de 3 segundos o modelo reconheceu padrões de tráfego muito mais restritos, o que resultou em um índice maior de falsos positivos. Da mesma forma, o reconhecimento de padrões de tráfego mais gerais nos períodos de 8 e 24 segundos resultou num índice baixo (nulo) de falsos positivos.

#### **4.2.2. Modelo TCP**

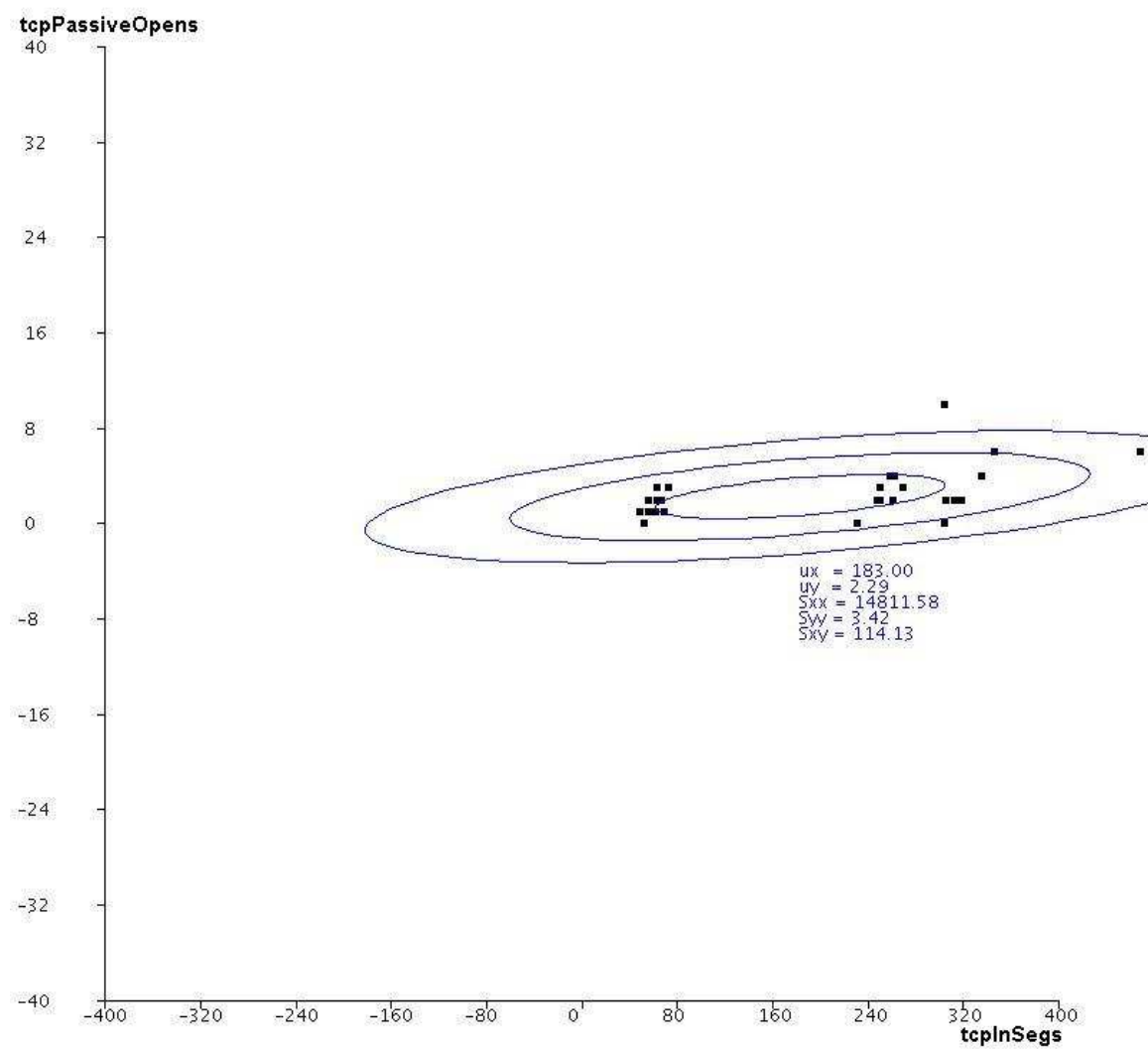
As figuras 4-7, 4-8 e 4-9 a seguir mostram o resultado do treinamento para as simulações com intervalo de polling de 3, 8 e 24 segundos para o modelo TCP. Os gráficos mostram as realizações que compõem os treinamentos e a formação de clusters, quando ocorrida. O eixo horizontal representa a alteração em `tcpInSegments` e o eixo vertical a alteração em `tcpPassiveOpens`, no período dado. Além disso, cinco dados estatísticos são

apresentados: média do eixo x, média do eixo y, variância do eixo x, variância do eixo y e covariância entre x e y, respectivamente.

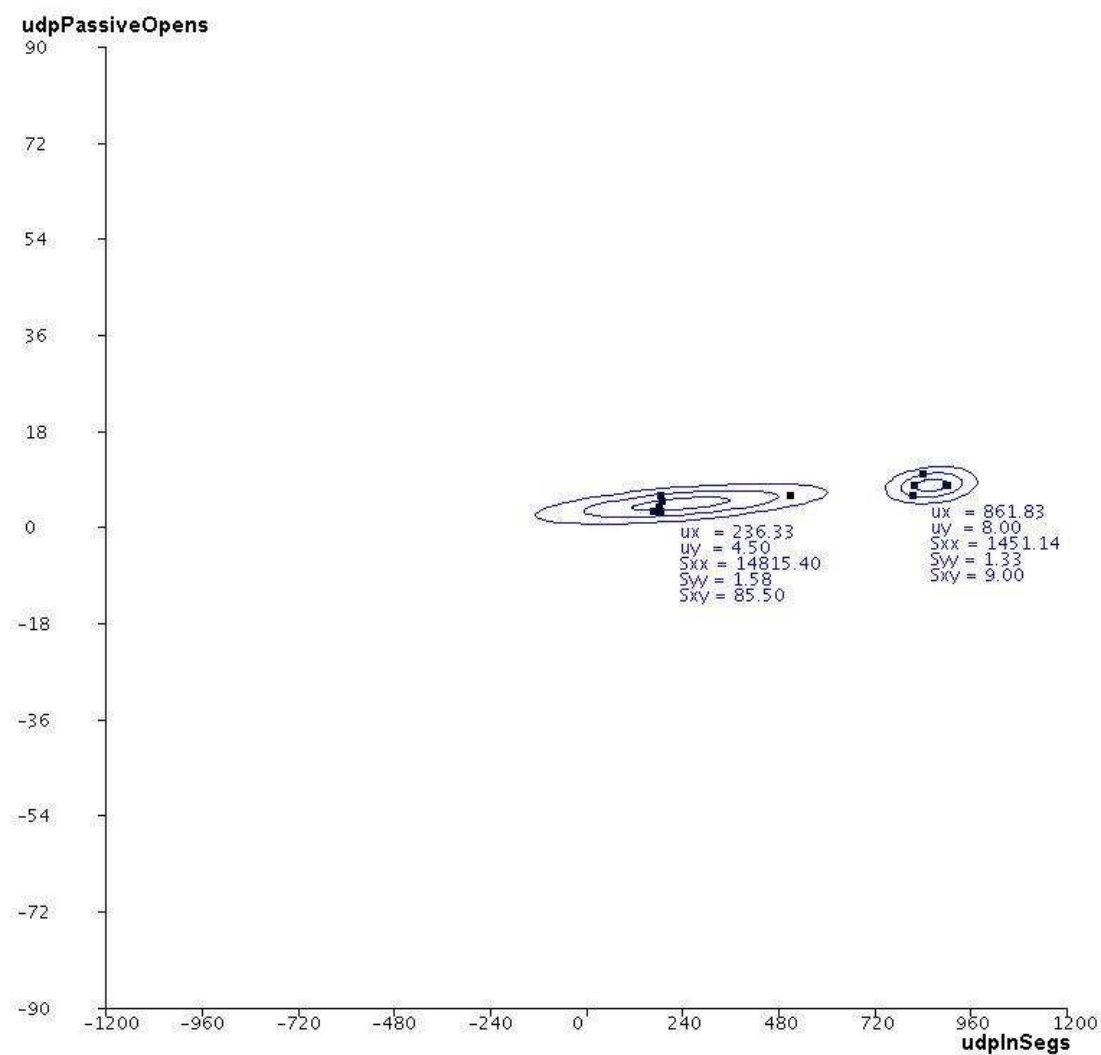
Naturalmente, os clusters estão relacionados com os dois padrões de tráfego TCP descritos anteriormente. Os dois clusters separam os dois padrões de tráfego descritos anteriormente neste capítulo, modelando duas aplicações diferentes rodando na rede.



**Figura 4-7 – Modelagem TCP com intervalo de polling de 3 segundos**



**Figura 4-8 – Modelagem TCP com intervalo de polling de 8 segundos**



**Figura 4-9 – Modelagem TCP com intervalo de polling de 24 segundos**

Assim como no caso UDP obtivemos diferenças entre os resultados, dependendo do período de consulta. No entanto, aqui conseguimos a formação de dois clusters bem definidos também para 24 segundos. Isso significa que a modelagem resultou em um ajuste GMM com dois *kernels* (somatório de duas funções separadas), como descrito no capítulo 2.

#### 4.2.2.1. Detecção de ataques

Lembramos mais uma vez que os resultados aqui apresentados correspondem à avaliação de um IDS localizado em um dos nodos participantes da rede (i.e. *Notebook-1*), que pode ser alvo do ataque ou servir como roteador entre o atacante e o alvo.

Assim como para UDP, geramos ataques reais utilizando o software Flash Port Scanner e ataques simulados. Para a simulação de ataques com o gerador usamos o programa GeradorTCP enviando 10 requisições de abertura de portas por segundo.

Novamente, para medir falsos positivos fizemos 100 avaliações gerando o mesmo tráfego gerado durante o treinamento. Para a medição de falso negativo, geramos quatro tipos de ataque: Scanner com Flash Port Scanner com o IDS no nodo alvo; Scanner com GeradorTCP com o IDS no nodo alvo; Scanner com Flash Port Scanner com IDS em nodo roteador e Scanner com GeradorTCP com o IDS em nodo roteador. Para todos os ataques obtivemos os mesmos resultados.

Intervalo de Polling	Falso positivo	Falso negativo (nodo alvo)	Falso negativo (nodo roteador)
2 segundos	0%	0%	0%
3 segundos	0%	0%	0%
4 segundos	0%	0%	0%
8 segundos	0%	0%	0%
12 segundos	0%	0%	0%
24 segundos	0%	0%	0%

**Tabela 4-5 – Taxas de falso positivo e falso negativo para ataque Scanner de portas TCP com detecção de alta probabilidade**

Intervalo de Polling	Falso positivo	Falso negativo (nodo alvo)	Falso negativo (nodo roteador)
2 segundos	0%	0%	0%
3 segundos	12%	0%	0%
4 segundos	0%	0%	0%
8 segundos	10%	0%	0%
12 segundos	2%	0%	0%
24 segundos	10%	0%	0%

**Tabela 4-6 – Taxas de falso positivo e falso negativo para ataque Scanner de portas TCP com detecção de baixa probabilidade**

Os resultados obtidos nos testes foram novamente positivos. Se avaliarmos apenas detecção de alta probabilidade, vemos que a ocorrência de erros de avaliação foi novamente nula. Isso indica que o domínio de dados está excessivamente restrito e por isso qualquer variação pode ser facilmente identificada. O ataque Scanner de portas TCP realizado com o software Flash Port Scanner, tendo o IDS localizado tanto no nodo alvo como em um nodo roteador, foram sempre detectados. O mesmo resultado foi obtido com os ataques simulados. Como nos testes UDP, tivemos falsos positivos de baixa probabilidade.

No caso da modelagem TCP esses três períodos (3, 8 e 24 segundos) foram os que resultaram em melhor reconhecimento dos padrões de tráfego e também em maior taxa de falsos positivos. Aqui cabe a mesma análise feita para os falsos positivos UDP, na seção anterior.



Vale ressaltar que detecção do ataque Scanner com o IDS em nodo roteador deve-se ao modelo UDP e não ao TCP, já que é possível graças ao desvio anormal de `ipForwardedDatagrams`.

## 5.CONCLUSÃO

A principal motivação para a realização desse trabalho foi tentar validar o trabalho de simulação descrito em [1] e testar em um ambiente real a aplicabilidade da teoria apresentada no mesmo. Para isso, resumimos a referida teoria, apresentamos uma implementação de um sistema de detecção de intrusão por anomalia de comportamento e realizamos uma série de testes para avaliar o desempenho desse sistema.

Durante o desenvolvimento do software algumas decisões importantes foram tomadas. Uma delas foi como separar as etapas de treinamento e detecção. Para isso optamos por limitar o treinamento por um número máximo de realizações de auditoria, ao invés de estipular um período de tempo.

Devido à impossibilidade, de montar um ambiente de rede com a dinâmica complexa de uma rede Manet real, fomos obrigados a definir uma arquitetura de rede simples e bem definida. Para a construção desse ambiente de rede e sua gerência fizemos uso de pacotes standard de software. Para roteamento entre nodos foi utilizado o *olsrd*, um *daemon* pra linux que implementa o protocolo de roteamento OLSR. Já para a gerência SNMP foi utilizado o pacote de utilitários NET-SNMP.

Dentro desse ambiente de rede particular definimos quais padrões de tráfego que seriam considerados normais pelo IDS. Para gerar esses padrões bem definidos voltamos a nos utilizar de ferramentas padrão, mais especificamente as ferramentas de geração de tráfego PACKIT e IPERF.

Com respeito à tentativa de validação da teoria apresentada no capítulo 2, sobre detecção por anomalia no comportamento, certamente conseguimos resultados satisfatórios. Nos testes realizados não houve ocorrência de erros de avaliação para detecção de alta probabilidade, nem de falso positivos nem de falso negativos. Conseguimos com sucesso detectar ataques DoS, DDoS e Scanner TCP, tanto com o IDS localizado no nodo alvo quanto localizado em nodos roteadores entre o atacante e o alvo. Esse resultado, particularmente, indicou um desempenho de detecção para o IDS até superior ao indicado no ambiente simulado de [1]. Esperávamos, antes da realização dos testes, que o desempenho de detecção do IDS piorasse em relação às simulações, devido aos problemas inerentes aos ambientes de redes reais. No entanto, por tratar-se de um exemplo particular e bastante simples de rede

Manet, essa expectativa demonstrou-se equivocada. De fato, apenas num ambiente bem restrito resultados tão bons podem ser obtidos.

Isso reforça o papel do referido trabalho como uma generalização da situação aqui apresentada. Nas simulações havia um número maior de nodos participantes e foi modelada a mobilidade de nodos, enquanto nossos testes passaram-se num cenário estático, com apenas quatro nodos. Além disso, a simulação previa a presença do IDS em todos os nodos da rede e nossos testes foram realizados com o IDS instalado em um nodo específico. Nesse sentido, nosso trabalho obteve sucesso na validação das simulações para um ambiente mais restrito e bem controlado.

Percebemos em nossos testes que há em termos gerais uma relação de compromisso entre a qualidade da percepção pelo IDS dos diferentes padrões de tráfego e a quantidade de falsos positivos por ele emitidos. Idealmente, gostaríamos que o sistema modelasse muitos padrões de tráfego diferentes e pudesse reconhecer cada um deles de forma diferenciada. De fato, numa rede de aplicações reais comum, o IDS deveria ser capaz de ajustar o modelo para muitos clusters diferentes e reconhecer bem cada padrão. Devido a isso, nessa situação o treinamento precisa abranger o máximo possível de aplicações passíveis de serem rodados na rede, para que modelos bem ajustados não ocasionem alta taxa de falsos positivos.

Verificou-se alteração no resultado da modelagem do comportamento de rede ao variar o intervalo de amostragem das variáveis de auditoria. Uma das sugestões pra trabalhos futuros é o uso de geradores de tráfego auto-similar. Teoricamente essas alterações não deveriam existir nesse caso.

Outra sugestão refere-se à ampliação do domínio de dados de auditoria. A análise de outras variáveis como variáveis da MIB de camada 2 poderá introduzir novas facilidades à ferramenta aqui apresentada. O uso de duas instâncias paralelas no módulo analisador também é um trabalho interessante a ser realizado, sabe-se que os métodos de detecção por anomalia no comportamento e por assinatura de ataque são complementares entre si, então poderíamos no mesmo projeto de IDS ter os 2 modelos trabalhando em conjunto para aumentar a eficiência da detecção. A implementação de resposta ativa a intrusões constituindo um IPS também seria um trabalho interessante.

Finalmente, acreditamos ter atingido nossos objetivos, dentro as restrições impostas pelo ambiente de rede controlado e bem definido proposto neste trabalho.

## 6.REFERÊNCIAS BIBLIOGRÁFICAS

- [1] F. Miziara, *Sistema de detecção de intrusão por anomalia no comportamento para redes ad-hoc*. Tese de Mestrado, Publicação ENE 293/07, Departamento de Engenharia elétrica, Universidade de Brasília, Brasília, DF, 2007.
- [2] R. Puttini, *Um Modelo de Segurança Para Redes Móveis Ad Hoc*. Tese de Doutorado, Publicação ENE 004/04, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 2004.
- [3] H. Debar, M. Dacier and A. Wespi. *A revised taxonomy for intrusion-detection systems*, IBM Research Report, Zurich, 1999.
- [4] A. S. Javits and A. Valdetts, *The SRI IDES Statistical Anomaly Detector*, Proc. Of IEEE Symposium of Research on Security and Privacy, pp. 316-326, may 1991.
- [5] J. Cabrera, L. Lewis, R. Prasanth, X. Qin, W. Lee, and R. Mehra. *Proactive detection of distributed denial of service attacks using MIB traffic variables – a feasibility study*. Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management, Seattle, WA, USA, may 2001.
- [6] G. J. McLachlan, D. Peel, K. E. Basford and P. Adams, “The EMMIX Software for the Fitting of Mixtures of Normal and t –Components”, Journal of Statistical Software, v. 04, 1999.
- [7] A. P. Dempster, N. M. Laird and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm” (with discussion). Journal of the Royal Statistical Society B 39 ,1-38, 1977.
- [8] P. Cheeseman and J. Stutz, “Bayesian classification (AutoClass): theory and results. Advances in Knowledge Discovery and Data Mining, U. M. Fayyad, G. Piatetsky-Shapiro, R. Smyth and R. Uthurusamy (Eds.), Menlo Park, California: The AAAI Press, pp. 61-83, 1996.
- [9] S. J. Roberts, R. Everson and I. Rezek, “Maximum Certainty Data Partitioning”, Pattern Recognition, 33:5, pp. 833-839, 1999.
- [10] C. Fraley and A. E. Raftery, “MCLUST: Software for Model-Based Cluster and Discriminant Analysis”, Technical Report No.342, Department of Statistics, University of Washington, 1998.
- [11] R. A. Johnson, D. A. Wichern, D. W. Wichern, “Applied Multivariate Statistical Analysis – 4<sup>th</sup> Edition”, Prentice-Hall, 1998.
- [12] A. Genz, “Numerical Computation of Multivariate Normal Probabilities”, J. Comp. Graph Statistics vol.1, pp. 141-149 (1992).
- [13] D. M. Tittertonington, "Recursive Parameter Estimation using Incomplete Data", J. R. Statist. Soc. B, n.o 46, pp. 257-267.
- [14] Rede Nacional de Ensino e Pesquisas. January, 08, 2007. <http://www.rnp.br>.
- [15] M. Wood and M. Erlinger. *Intrusion Detection Message Exchange Requirements*, IETF Internet Draft, October 22, 2002. <http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-10.txt>
- [16] The Unix system, June, 15, 2007. <http://www.unix.org>
- [17] OLSR Site, May 10, 2007. <http://www.olsr.org>
- [18] The Internet Engineering Task Force, April, 23, 2007. <http://ietf.org/rfc/rfc3626.txt>
- [19] Net-SNMP WebSite, January, 10, 2007. <http://net-snmp.sourceforge.net/>
- [20] The Internet Engineering Task Force, April, 23, 2007.

- <http://ietf.org/rfc/rfc1213.txt>
- [21] PackIt- Traffic Tool. February, 13, 2007.  
<http://www.intrusense.com/software/packit/>
  - [22] Iperf – Traffic Tool. February, 14, 2007. <http://dast.nlanr.net/Projects/Iperf/>
  - [23] UDP FLOOD. March, 26, 2007. <http://www.foundstone.com>
  - [24] FlashPortScanner. March, 26, 2007. <http://www.softpedia.com/get/Network-Tools/Network-IP-Scanner/Flash-Port-Scanner.shtml>

## 7. ANEXO I

Arquivo `olsrd.conf`. Esse é o arquivo de configuração do *daemon* `olsrd` que implementa o protocolo OLSR. Essa configuração foi utilizada nos *notebooks* para a topologia apresentada.

```
LinkQualityFishEye 1
IpVersion          4
Hna4
{
}

AllowNoInt         yes
TosValue           16
Willingness        4

IpcConnect
{
    MaxConnections  0
    Host            127.0.0.1
}

UseHysteresis      no
LinkQualityLevel   2
LinkQualityWinSize 10
Pollrate           0.5
TcRedundancy       2
MprCoverage        7

LoadPlugin "olsrd_snmpd_agentx.so.1.0"
{
    PlParam "poll" "10.0"
}

Interface "eth1"
{
    HelloInterval      0.5
    HelloValidityTime   10.0
    TcInterval         1.0
    TcValidityTime      14.0
    MidInterval        5.0
    MidValidityTime     15.0
    HnaInterval         5.0
    HnaValidityTime     15.0
    # LinkQualityMult 192.168.0.1 0.40
}
```

## 8.ANEXO II

Arquivo: snmpd.conf. Configuração mínima para funcionamento adequado do agente. Essa configuração foi utilizada nos *notebooks* durante os experimentos.

```
master agentx

trapsink localhost public 161

trapcommunity public

com2sec public default public
com2sec local localhost private

group public v1 public
group public v2c public
group local v1 local
group local v2c local

view all included 1. 80
view system included system fe
view mib2 included .iso.org.dod.internet.mgmt.mib-2 fc

access public "" any noauth exact all none none
access local "" any noauth exact all all all
```