

# Universidade do Minho Departamento de Sistemas de Informática

**MIETI, UMINHO 2022/2023**

**Módulo de Gestão Redes**

**Relatório do Projeto de Grupo**

Grupo:



José Gomes

A93XXX

Diogo Cerqueira

A93108

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

Índice

[1. Introdução 6](#_Toc125119599)

[2. Requisitos 8](#_Toc125119600)

[3. Implementação 9](#_Toc125119601)

[3.1 Linguagem de Programação 9](#_Toc125119602)

[3.2 MIB 10](#_Toc125119603)

[3.3 Classes 13](#_Toc125119604)

[4. Teste, Resultados e Discussão 22](#_Toc125119658)

[4.1 Discussão de Resultados 24](#_Toc125119659)

[5. Autoavaliação 25](#_Toc125119660)

[6. Referências 26](#_Toc125119661)

[7. Conclusão 27](#_Toc125119662)

# Introdução

No âmbito da Unidade Curricular de Gestão de Redes foi proposta a realização de um trabalho prático, que tem como objetivo a implementação de um agente *proxy* SNMPv2c com recurso ao conhecimento que foi adquirido ao longo do semestre.

O presente relatório tem comos objetivos principais a apresentação do problema proposto, a formulação do mesmo, a descrição das decisões efetuadas pelo grupo, e uma reflecção crítica sobre os resultados obtidos.

De forma a realizar o trabalho com sucesso, anteriormente, foi importante e necessário relembrar alguns conceitos e postulados da linguagem de programação *Java*, visto que foi a linguagem adotada pelo grupo para a realização do presente projeto.

# Requisitos

De forma a garantir o planeamento prévio e adequado da exequibilidade do presente trabalho torna-se essencial identificar claramente os seus requisitos, uma vez que os mesmos servirão de base ao trabalho executado posteriormente.

* 1. **Requisitos funcionais**

Para que o sistema trabalhe com o funcionamento desejado é necessário que cumpra os seguintes requisitos:

* + - Possuir um pacote *freeware* instalado com suporte ao protocolo SNMP *(Simple Network Management Protocol)* – o grupo decidiu usufruir do pacote “Net*-*SNMP”.
    - Possuir um módulo JDK *(Java Development Kit)* instalado na respetiva máquina (versão 19 ou superior) - o projeto foi construído em linguagem *Java,* logo, é necessário um módulo capaz de compilar o código implementado.
  1. **Requisitos não funcionais**

Os próximos requisitos, que serão apresentados nos tópicos seguintes, são não funcionais o que significa que os mesmos não estão ligados diretamente às funcionalidades do sistema, mas sim relacionados com o tempo de execução/resposta e da fiabilidade. Os requisitos são definidos por:

* Código simples e otimizado, que proporcionará um menor tempo de resposta;
* Código bem comentado para facilitar a sua compreensão;
* Repartição do código por vários *packages* (diretorias), que representam uma determinada entidade do problema;

# Lista de Figuras

[Figura 1 - Esquema que representa a MIB 7](file:///C:\Users\diogo\Downloads\Relatorio_GR.docx#_Toc125387289)

[Figura 2 - Exemplo de um snmpget 10](file:///C:\Users\diogo\Downloads\Relatorio_GR.docx#_Toc125387290)

[Figura 3 - Exemplo do envio de um comando pela Socket 11](file:///C:\Users\diogo\Downloads\Relatorio_GR.docx#_Toc125387291)

[Figura 4 - Funções de encriptação e desencriptação 13](file:///C:\Users\diogo\Downloads\Relatorio_GR.docx#_Toc125387292)

# Implementação

# 4.1 Linguagem de Programação

De forma a efetuar a realização do código do projeto, foi decidido a utilização da linguagem de programação *Java*, pois, além de ser a linguagem para a qual os membros do grupo estão confortáveis e entendem os conceitos vitais, graças a várias pesquisas que foram realizadas pelo grupo, foi possível descobrir que esta possui bibliotecas nativas, que facilitam a implementação da primitiva SNMPv2c.

As bibliotecas que foram utilizadas pelo grupo são as seguintes:

* ***Crypto***– *package* importada, que é crucial para a implementação da cifra AES, que faz a cifragem da informação que é comunicada entre o *Agent* e o *Manager.*
* ***Socket***– classe necessária para o estabelecimento de um canal de comunicaçãoentre o *Agent* e o *Manager*.
* ***Process –*** classe necessária para a execução das primitivas SNMP.
* ***LocalDateTime***– classe utilizada para funcionalidades inerentes ao domínio do tempo.
* ***Objects***– classe usada para facilitar a utilização dos objetos do programa.

# 4.2 MIB

A nossa MIB é constituída por uma tabela (OperTable) cuja é constituída por variadas entradas. Na figura seguinte é possivel observar a sua constituição:

Diagrama

Descrição gerada automaticamente

Figura 1 - Esquema que representa a MIB

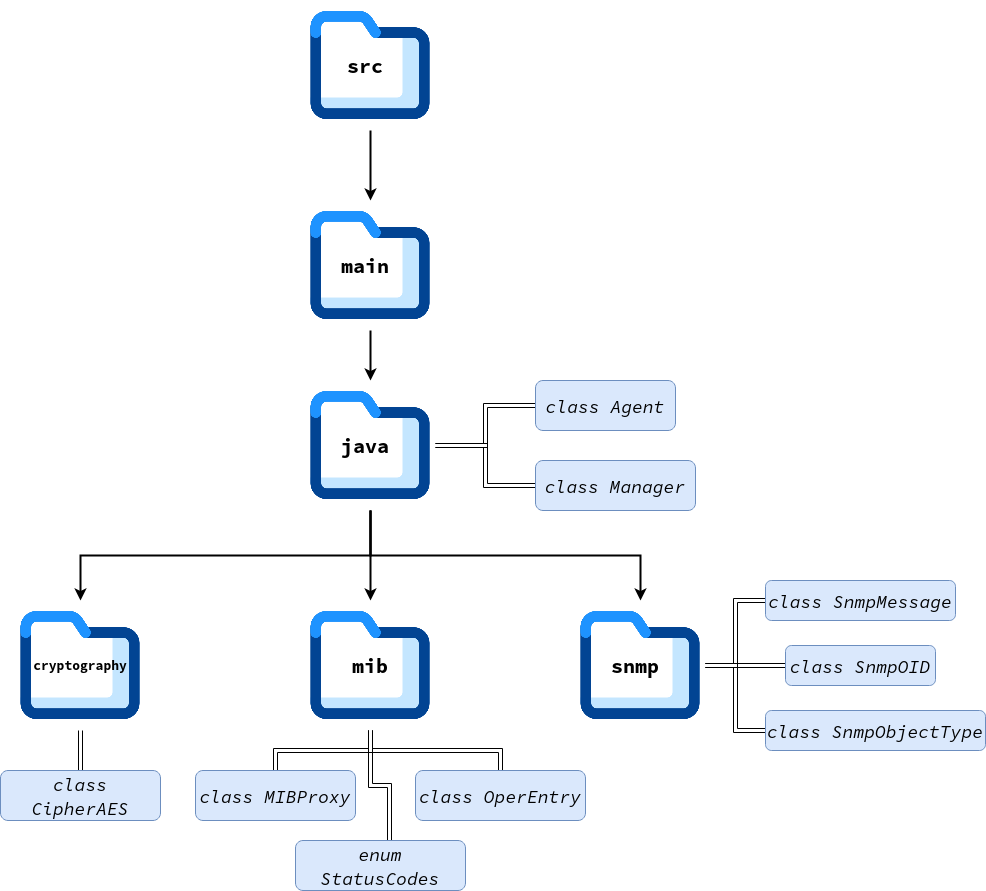
**Objetos da MIB:**

* **OperTable**: Estrutura de dados que armazena as entradas
* **OperEntry:** Conjunto que possui os objetos
* **idOper:** é o identificador da operação recebida pelo agente. É a chave da tabela.
* **typeOper:** mostra o tipo de operação SNMP.
* **operArg1:** 1º argumento de uma determinada operação a ser processada.
* **operArg2:** 2º argumento de uma determinada operação a ser processada.
* **idSource:** identificador da fonte do request.
* **idDestination:** identificador do destido do request e onde a operação será executada.
* **idArg:** OID (*Object Identifier*) do objeto da MIB e cujo é o argumento da operação a ser executada pelo agente.
* **valueArg:** valor do objeto referido pelo idArg e cujo é o resultado recebido no agente proxy vindo do agente SNMP.
* **typeArg:** tipo de dados do valueArg.
* **sizeArg:** tamanho em bytes do valueArg
* **ttl:** tempo de vida restante da entrada na tabela. Quando este objeto possuir o valor zero a entrada é retirada e o Manager não poderá aceder aos valores.
* **status:** é um número que corresponde ao estado do request (accepted, created, entre outros)

# 4.3 Árvore de diretorias

Para promover a organização e categorização do código desenvolvido no decorrer do projeto, numa fase inicial o grupo procedeu à construção de uma “árvore” de diretorias*/packages* para distribuir ou separar o código desenvolvido. Cada diretoria da árvore representa um domínio do sistema, onde alberga as classes relativas a esse domínio.

A figura y demonstra a árvore de diretorias construída pelo grupo.



# 4.3 Classes

No total, o projeto desenvolvido pode ser repartido em nove classes:

# Agent

# A classe Agent é o paradigma do agente *proxy* que implementa a MIB especial de segurança (MIBproxy). Este desempenhará a função de *inteface* seguro para um agente SNMPv1 ou SNMPv2, normal e não seguro.

# A comunicação, entre o agente *prox*y e o *Manager,* assenta no modelo de comunicação cliente-servidor. A classe *Agent* desempenha o papel de servidor, isto é, responde a eventuais pedidos efetuados pelo cliente (classe *Manager*). Posteriormente, a resposta é impressa na *bash.*

# Visto que a classe Agent atua como servidor, esta é responsável pelo estabelecimento do canal de comunicação com o cliente. O canal de comunicação é definido com recurso à classe *ServerSocket*, da qual é passado como argumento a porta onde se estabelecerá a respetiva comunicação.

# Enquanto o cliente não transmitir a mensagem “*exit*”, o agente *proxy* continuará a atender aos *requests* recebidos.

# Após a receção de cada *request*, este passará pela cadeia de processos seguinte:

# Decifragem do *request*.

# Execução da primitiva SNMP, encapsulada pelo *request*.

# Impressão do resultado da execução da primitiva.

# Atualização do conteúdo da MIBproxy.

# Impressão da MIBproxy.

# Antes da atualização do conteúdo da MIBproxy, é necessário processar o *request*, ou seja, é preciso determinar a natureza da mensagem SNMP, analisar e extrair os parâmetros que compõe essa mensagem.

# O método “parseSnmpCommand*”* da classe Agent acomoda os requisitos acima. Numa fase inicial é processada a primitiva SNMP, após o processamento são extraídos os argumentos referentes a essa primitiva e de seguida é feita uma associação entreo tipo de mensagem SNMP com o bloco de instruções associado.

# A figura z apresenta o algoritmo do método “parseSnmpCommand”.

# 

# Tal como foi citado no parágrafo anterior, dentro do método “parseSnmpCommand” é estabelecida uma associação entre o tipo de mensagem SNMP com o bloco de instruções apropriado.

# A figura seguinte apresenta o bloco de instruções caraterístico da primitiva SNMP “GetRequest”.

# Texto Descrição gerada automaticamente

Figura 2- Exemplo de um snmpget

# 

# Manager

# Tal como foi referido na secção da classe “Agent”, a comunicação entre o agente *proxy* e o *Manager* assenta no modelo de comunicação cliente-servidor.

# A classe "Manager" desempenha o papel de cliente, ou seja, é responsável por efetuar pedidos SNMP, pela cifragem e envio dos mesmos.

# O método “main” da classe “Manager” acomoda os requisitos acima. Numa fase inicial o *Manager* conecta-se ao endereço IP e à porta do canal de comunicação, que foram estabelecidos pelo agente *proxy*. De seguida é lido o *input* (via teclado) do utilizador, e numa fase terminal esse *input* será cifrado e enviado ao agente *proxy*.

# A figura X ilustra o algoritmo do método “main” da classe Manager”.

# 

# Nesta classe é criada uma Socket que vai conectar à Socket Server do Agente (Modelo Cliente-Servidor). Estas 2 Sockets conectam-se devido a estarem a usar a mesma porta. Esta classe recebe os dados escritos pelo utilizador através do terminal, encripta-os e envia-os para o Agent.

# Texto Descrição gerada automaticamente

Figura 3- Exemplo do envio de um comando pela Socket

# StatusCodes

# Esta classe os valores (códigos) que vão ser inseridos no objeto “status” para descrever o estado do request. Os valores possíveis são:

# OK – 200

# Created – 201

# Accepted – 202

# Bad Request – 400

# Unauthorized – 401

# Forbidden – 403

# Not Found - 404

# OperEntry

# 

# Nesta classe é criado um Objeto OperEntry que irá possuir vários atributos que se referem aos objetos que caracterizam o request.

# Possui uma função que compara 2 OperEntry e verifica se são iguais. Além disto, possui as tradicionais funções de get() e set() dos seus atributos e a toString().

# MIBProxy

# A figura **x** ilustra os atributos e métodos da classe MIBProxy.

# 

# A classe MIBProxy permite a definição de mecanismos de segurança indiretos, que possibilitam uma manipulação fiável e segura dos objetos que são implementados nas MIBs normais dos agentes normais.

# Através da análise da figura x é possível observar que a classe referida é composta por dois atributos, que representam respetivamente, o OID da tabela operTable e a respetiva tabela operTable. Abaixo dos atributos descritos encontram-se listados os métodos da classe MIBProxy que permitem manipular o estado desses atributos.

# Os atributos, que representam o estado da classe, descritos anteriormente, têm de ser privados de forma a respeitar os princípios e postulados da programação orientada a objetos, assim, só possível atualizar ou manipular os atributos da classe apenas com os métodos públicos definidos pelo grupo (representados pelo caractere ‘+’).

# O agente *proxy* (representado pela classe *Agent*)é responsável pela inicialização e instanciação da classe MIBProxy.

# A figura x ilustra a inicialização e instanciação da MIBProxy efetuada pelo agent *proxy,* que sucede no seu método *main*.

# 

# O modificador *final* reforça a estabilidade e segurança da MIB pois não permite que nenhuma entidade externa ao sistema, altere ou atualize a referência ao respetivo objeto.

# CipherAES

# Nesta classe são implementados os métodos de criptografia usados para encriptar o canal de comunicação entre o Manager e o Agent. A cifra escolhida para este processo foi o AES. Esta classe é muito importante pois possui o IV e a chave da cifra. Também possui as funções de encriptação e desencriptação, cujas podemos observar na figura seguinte.

# Texto Descrição gerada automaticamente

Figura 4- Funções de encriptação e desencriptação

# 

# 4.4 Canal de Comunicação

# A comunicação entre o Manager e o Agent realiza-se através de Sockets com base num modelo de Cliente - Servidor. É criada uma Socket Server no Agent que espera que a Socket Client criada no Manager se conecte a esta. A porta usada é a 5000 mas pode ser alterada, desde que a escolhida esteja livre, ou seja, não esteja a ser usada.

# Existe uma proteção do tráfego pois os dados são encriptados através da utilização de técnicas de criptografia, mais concretamente, a cifra AES. Esta é uma cifra simétrica (usa a mesma chave para encriptar e desencriptar o tráfego). Esta possui um vetor aleatório, denominado IV, que traz aleatoridade ao criptograma. Decidimos escolher esta cifra pois é uma cifra considerada inquebrável e que não traz uma sobrecarga computacional elevada ao processo.

# Teste, Resultados e Discussão

Os testes seguintes foram realizados para o ficheiro de texto “Circuito do Mónaco”

**BFS**

Custo = 24

**DFS**

Custo = 26

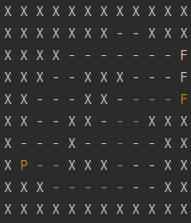
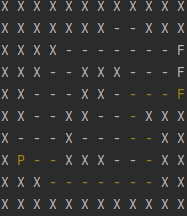


Fig 6 – Percurso da Pesquisa BFS

Fig 5 – Percurso da Pesquisa DFS

**Greedy**

Custo = 26

**A\***

Custo = 24

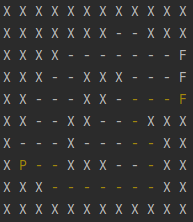
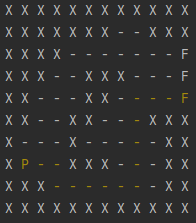
****

Fig 8 – Percurso da Pesquisa A\*

Fig 7 – Percurso da Pesquisa Greedy

# Discussão de Resultados

Como é possível observar pelas figuras 5, 6 , 7 e 8, para este circuito as melhores pesquisas são a BFS e a A\* pois possuem um menor custo final. Também é importante realçar que todas as pesquisas acabaram com uma velocidade final de 17 e que em nenhum momento tiveram uma colisão com os limites de pista.

Através destes resultados, também foi interessante observar que as pesquisas BFS e A\* apesar de possuírem um custo final igual, estas tiveram percursos diferentes.

Também, tal como esperado, conseguimos concluir que a noção geral de melhor ou pior pesquisa não existe, pois essa noção depende do problema proposto. Por exemplo, a pesquisa DFS e Greedy foram as “piores” pesquisas para o Circuito do Mónaco, mas para os circuitos de Melbourne e Suzuki já acabaram com um custo final igual à A\* e à BFS.

# Conclusão

A realização deste projeto permitiu ao grupo consolidar o conhecimento relativo aos protocolos, mecanismos e filosofias da arquitetura de gestão do INMF e dos conceitos principais sobre ameaças de segurança em aplicações e sobre as estratégias que permitem implementar uma maior segurança ao sistema. Também nos permitiu aumentar profundamente o conhecimento sobre o protocolo SNMP, atarvés da observação de como este funciona na prática. É de realçar que a execução de testes experimentais, foi fundamental para uma análise crítica dos resultados obtidos.

O grupo acabou por ter algumas dificuldades na encriptação do canal de comunicação, mais em concreto na transformação de byte para texto limpo mas após um maior estudo sobre este tema foi possível ultrapassar esta dificuldade..

# Autoavaliação

* + - Diogo Cerqueira

Neste projeto estive envolvido no desenvolvimento do código e no desenvolvimento do relatório. A maior dificuldade que encontrei durante esta fase foi na elaboração do código da função BFS.

* + - José Gomes

Neste projeto estive envolvido no desenvolvimento do código e no desenvolvimento do relatório. A maior dificuldade que encontrei foi na implementação dos códigos de pesquisa.

# Referências

[1] Russell and Norvig (2009). Artificial Intelligence - A Modern Approach, 3rd edition,

ISBN-13: 9780136042594;

[2] Costa E., Simões A., (2008), Inteligência Artificial-Fundamentos e Aplicações, FCA,

ISBN: 978-972-722-34

[3] Maria do Carmo Nicoletti, Estevan R. Hruschka, (2017), Fundamentos da Teoria dos Grafos para Computação

Fazer

# Conclusão

De modo a finalizar a fase deste relatório, gostávamos de referir que, apesar das dificuldades encontradas consideramos que conseguimos alcançar um bom resultado, pois alcançámos todos os objetivos propostos.

Este trabalho permitiu-nos compreender melhor o funcionamento dos algoritmos de pesquisa e ajudou-nos muito a melhorar a nossa capacidade de trabalhar com a linguagem de programação Python.