AUVNetSim: Um simulador para redes acústicas subaquáticas

Joseph Jornet Montana
jmjornet@mit.edu

MIT Sea Grant College Program
Instituto de Tecnologia de Massachusetts,
Cambridge, MA, 02139

Conteúdo

Capítulo 1

Instalação

AUVNetSim é uma biblioteca de simulação para testar algoritmos de rede acústica [1]. Ele é escrito em Python padrão [2] e faz uso extensivo do pacote de simulação de eventos discreto SimPy [3]. AUVNetSim é redistribuída sob os termos da Licença Pública Geral GNU.

AUVNetSim é interessante tanto para *usuários finais* quanto para *desenvolvedores.* Um usuário disposto a executar várias simulações usando os recursos que já estão disponíveis, pode facilmente modificar vários parâmetros do sistema sem ter que lidar explicitamente com o código Python. Um desenvolvedor, que por exemplo, quer incluir um novo protocolo MAC, pode simplesmente fazê-lo aproveitando a estrutura existente.

1.1Prerequisites

Para executar este software, os seguintes pacotes são necessários antes de instalar e executar o AUVNetSim:

- Ambiente Python: o software núcleo Python (versão 2.5) [4].
- SimPy Package: um sistema de simulação de eventos discreto [5].
- MatplotLib: uma biblioteca de plotagem Python (versão 0.91 ou mais) [6].
- Numpy: um pacote que fornece funcionalidades científicas de computação [7].

Todo este software está livremente disponível sob os termos da licença GNU. Siga as instruções incluídas em cada pacote para concluir adequadamente sua instalação.

1.2Instalação

O código-fonte AUVNetSim está disponível usando a subversão (SVN). Existem vários clientes SVN para diferentes plataformas. Por exemplo, usamos TartarugaSVN [8]. Uma vez que uma ferramenta como esta é instalada, um novo usuário deve:

- Crie uma nova pasta.
- Clique com o botão direito do mouse na pasta, SVN Checkout.
- Aponte para o repositório AUVNetSim em [9].

Automaticamente, o conteúdo do AUVNetSim será baixado para a máquina local. A versão estável atual do simulador está na subpasta do *porta-malas.* Para

instalar o simulador, basta digitar setup.py python da linha de comando dessa pasta. Para verificar se o simulador foi instalado corretamente, você pode executar os dois exemplos que estão incluídos no pacote.

Observe que todas as atualizações que estão sendo feitas na subpasta do tronco podem ser vistas diretamente apenas atualizando a pasta com a ferramenta SVN. Os usuários só podem modificar sua cópia local. Se uma grande mudança for introduzida, você pode entrar em contato com o administrador do projeto e ele considerará criar um ramo para você ou incluir os recursos propostos no *porta-malas*.

Capítulo 2

AUVNetSim para usuários finais

O simulador já contém uma grande variedade de parâmetros e protocolos para redes acústicas subaquáticas que podem ser selecionadas ou modificadas. Em vez de ter que compilar o código cada vez que uma nova simulação é necessária, o usuário só precisa configurar o arquivo de simulação (*.py) e o arquivo de configuração (*.conf).

2.1 Arquivo desimulação

Este arquivo contém a função *principal* que será invocada quando o simulador for lançado. O seguinte código Python serve como exemplo:

```
simulação de importação como AUVSim# Inclusão dos recursos
importação pylab# Inclusão da classe de visualização def

aSimulação():# Função principal

    se (len(sys.argv) < 2):
        imprimir "uso: ", sys.argv[0], "ConfigFile" # Um arquivo de configuração é esperado
    exit(1) config =

    AUVSim.ReadConfigFromFile(sys.argv[1])

    imprimir "Simulação de execução"
    nóress = AUVSim.RunSimulation(config) # A simulação é lançada
    imprimir "Done"

    para x em nó:
        nodes.append(nodess[x])
    PlotScenario3D(nodes)    # Visualização do cenário para simulação
    PlotConsumption(nódulos) # Visualização do consumo por nó
    PlotDelay(nodes)         # Visualização do atraso por nó
    pylab.show()
se __name__ == "__main__":
    aSimulação()
```

Após a inclusão da biblioteca AUVNetSim, a simulação é lançada. Alguns dos resultados ou estatísticas que são monitorados ao longo da simulação são

exibidos. O usuário pode usar as funções de visualização já definidas, criar novas ou salvar as informações necessárias em arquivos de texto simples que mais tarde poderiam ser lidos usando, por exemplo, o MATLAB. Vários exemplos estão incluídos com o pacote para download.

2.2 Arquivo de configuração

Vários parâmetros devem ser especificados no arquivo de configuração antes de cada simulação. Nas linhas a seguir, há um exemplo do conteúdo desse tipo de arquivos.

```
Duração da simulação (segundos)
SimulaçãoDuração = 3600,00

# Largura de banda disponível (kHz)
Largura de banda = 10,00

# Eficiência de largura de banda (bps/Hz)
Largura de bandaBitrateRelation = 1,00

# Frequência (kHz)
Frequência = 10,00

# Potência máxima de transmissão -> intensidade acústica (dB re uPa)
TransmitirPower = 250,00

# Receber energia (dB) -> consumo de bateria (dB W)
ReceivePower = -10.00

# Ouvir energia (dB) -> consumo de bateria (dB W)
ListenPower = -30.00

# DataPacketLength (bits)
DataPacketLength = 9600,00 #bits

# PHY: definir parâmetros para a camada física
PHY = {"SIRThreshold": 15.00, "SNRThreshold": 20.00,"LISThreshold": 3.00, "variávelPower":
True, True,
"multicast2Distance": {0:1600.00,1:2300.00,2:2800.00,3:3200.00,5:6000.0}}

# MAC: defina qual protocolo estamos usando e definir parâmetros
MAC = {"protocolo":"ALOHA", "max2resend":10.0, "tentativas":4,"ACK_packet_length":24,
"RTS_packet_length":48, "CTS_packet_length":48,"WAR_packet_length":24,
"SIL_packet_length":24,
"tmin/T":2.0,"twmin/T":0.0, "deltatdata":0.0, "deltad/T":0.0,}

# Roteamento: defina parâmetros para a camada de roteamento
Roteamento = {"Algoritmo": "Estática", "variação":2, "coneAngle":120.0}

# Nós: aqui é onde definimos nós individuais
Formato: Endereço, Posição [, período, destino]
Nós = [["SinkA", (5000.5000.500), Nenhum, Nenhum],["Sinkd",(5000,15000,500),
Nenhum, Nenhum], ["Sinkb", (15000.5000,500), Nenhum, Nenhum], ["SinkC",
(15000.15000,500), Nenhum, Nenhum]]

# NodeField: Configure um campo de nós.
Formato (grid_block_size, N_blocks_wide, N_blocks_high[, bottom_left_corner[,
node_ID_prefix])
```

```
NodeField = (5000,00, 4, 4, (0,0,0), "S")

# Altura máxima/profundidade em que os nós podem ser
Altura = 1000,00

# Por padrão, os nós no campo de nó são apenas relés, mas podemos
fazê-los # também gerar informações alterando esse valor.
Período = 240,0

# Faz mais sentido colocar nós aleatoriamente em vez de em uma grade
perfeita RandomGrid = True
# Nós no campo de nó podem estar se movendo aleatoriamente e
dentro sem estar ciente disso por causa dos fluxos
subaquáticos.
Movimento = Falso
Velocidade = 0,0
```

Todos os parâmetros possíveis que podem ser especificados atualmente estão resumidos na Tabela 2.1. A simulação pode ser iniciada apenas digitando a partir da linha de comando do SO:

```
!> python   simulation_file.py  configuration_file.conf
```

No apêndice A, há uma introdução com alguns exemplos de como controlar externamente o simulador ao usar o MATLAB. Apesar de não ser a abordagem que encorajamos, o usuário pode apenas pensar no simulador como uma caixa preta, com algumas entradas (o arquivo de configuração) e algumas saídas (que podem ser escritas em um arquivo de texto). Neste caso, o usuário raramente terá que lidar com o código Python.

Camada Física
Frequência central[kHz] Largura de banda[kHz] Eficiência da largura de banda[bit/Hz] Transmissão do modo de energia máxima[dBreμPa] Recebimento de consumo de energia [dBW] Consumo de energia de escuta [dBW] Limiar de audição [dB] Limite de recebimento [dB] Controle de energiaTrue/Falso Nome de níveis de energia:valor[km]Somente se o controle de energia for usado
Controle de **Acesso** Médio
ProtocolCS-ALOHA, DACAP, CSMA Comprimento RTS[bit]Somente se o DACAP for usado comprimento CTS[bit]Somente se o DACAP for usado comprimento ACK[bit] Comprimento de guerra[bit]Somente se o DACAP for usado comprimento SIL[bit]Somente se o DACAP for usado Comprimento de DADOS[bit] Tentativas de retransmissão Tempo máximo de espera[s]Somente se a ALOHA for usada TminOnly se o DACAP for usado TwminOnly se o DACAP for usado Região de interferência apenas se o DACAP for usado
Camada de roteamento
ProtocoloSObre rotas, Rotas Estáticas, FBR Variação0,1,2 Apenas quando rotas estáticas são usadas Abertura de cone[grau]Somente se a FBR for usada tentativas de retransmissão Somente se o FBR for usado
Nós
Nome Period[s]Pode ser Nenhum Destino[nó]Pode ser Nenhum Posição ou Lista de Caminhos de pontos Posição aleatória True/False Random Moving True/False Velocidadede movimento [m/s]
Duração dasimulação [s]

Tabela 2.1: Parâmetros AUVNetSim que devem ser especificados no arquivo de configuração

Capítulo 3

AUVNetSim para Desenvolvedores

Antes de ler esta seção, encorajamos o usuário a se familiarizar com a linguagem de programação Python e a biblioteca SimPy [2, 3].

A forma como o AUVNetSim é programado facilita a tarefa de incluir novos recursos. Como muitos outros simuladores de rede sem fio, a descrição das diferentes funcionalidades de camadas é especificada em diferentes classes ou arquivos. Um programador disposto a introduzir, por exemplo, uma nova técnica de roteamento não precisa lidar com o MAC ou a camada física.

A comunicação entre camadas é realizada pela troca de mensagens curtas. Por exemplo, um pacote proveniente da camada de aplicativo é enviado para a camada de roteamento, que atualizará o cabeçalho do pacote e, por sua vez, o enviará para a camada MAC. Por fim, a mensagem será transmitida ao canal através da camada física, seguindo a política de protocolo.

Nas linhas a seguir, é oferecida uma visão geral de cada um dos arquivos que compõem o simulador.

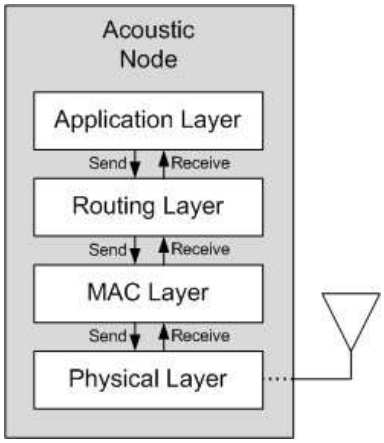

Figura 3.1: AUVNetSim: estrutura de programação de nó.

3.1Simulation.py

Este é o arquivo principal de um projeto. Aqui, o cenário 3D para simulação é criado e a simulação é realizada.

```
importar SimPy.Simulação como Sim# Inclusão do mecanismo de evento discreto
da Importação Acústica Node# Contém a definição de um nó

def RunSimulation (config_dict):
    Sim.initialize()
    # Este é o evento inicial que acionará a simulação Evento
    Acústico = Sim.SimEvent ("Evento Acústico")

    # Nós são configurados de acordo com config_dict, que é o conteúdo do arquivo de
    configuração. nodes = SetupNodesForSimulation (config_dict, AcousticEvent)

    # A simulação é executada até o final dos eventos ou pelo tempo

    máximo de duração Sim.simular (até=config_dict["Simulaçãoduração"])
```

Um cenário é definido de acordo com a descrição no arquivo de configuração. Existem duas maneiras de especificar os nós que o sistema contém:

- Cada nó pode ser especificado pelo seu nome e posição (ou um caminho se for um nó móvel). Se for um nó ativo, a taxa de geração de pacotes e o destino dos pacotes (um novo destino pode ser escolhido aleatoriamente antes de cada transmissão) também devem ser incluídos.

- Um campo de nó pode ser definido pela região 3D que está cobrindo e pelo número de nós que estão posicionados nele. Os nós podem ser completamente posicionados aleatoriamente ou posicionados aleatoriamente dentro de uma grade. Os nós em um campo de nó geralmente são apenas relés, mas é fácil fazê-los gerar informações também. Além disso, a deriva involuntária também pode ser modelada, apenas indicando que eles estão se movendo e especificando sua velocidade.

```
configuração defNodesForSimulation (config_dict,
   acoustic_event): nodes = [ ]

   # Nós únicos se "Nós" em
   config_dict.keys():
      para n em
         config_dict["Nodes"]: cn
         = [config_dict,] + n
         nodes.append (AcousticNode(acoustic_event, *cn))

   # Campo de nó se "NodeField" em
   config_dict.keys():
      nós += CreateRandomNodeField (acoustic_event,

   config_dict,*config_dict["NodeField"])
```

A forma como os nós no campo de relé são móveis involuntários é mostrada abaixo. Quando iniciados, quatro pontos aleatórios são criados e, seguindo a velocidade especificada, eles viajarão ao longo deles.

```
def CreateRandomNodeField (acoustic_event, config, box_len, nWide, nHigh,
                      bottom_left_position=(0,0,0), Prefixo="StaticNode"):
   importação
   aleatória
   aleatória
   random.seed()
   node_positions=
   [ ]

   # Nós podem ser localizados aleatoriamente dentro de uma grade
   virtual e movendo-se aleatoriamente se não config ["RandomGrid"]
   e não config["Moving"]:
   [node_positions.append((bottom_left_position[0]+box_len*x,
                        bottom_left_position[1]+box_len*y,
         bottom_left_position[2])) para x em intervalo(nWide) para y em alcance (nHigh)]
   elif config["RandomGrid"] e não config["Moving"]:
      [node_positions.append((bottom_left_position[0]+box_len*random.random()+b
                           ox_len*x,
                           bottom_left_position[1]+box_len*random.random()+box_l
                           en*y,
                           bottom_left_position[2]+config["Height"]*random.rando
                           m()) para x in range(n)Wide para y in range(nHigh)]
```

```
    mais:
        [node_positions.append(([bottom_left_position[0]+box_len*random.random()+box_len*x,
                              bottom_left_position[1]+box_len*random.random()+box_len*y,
                              bottom_left_position[2]+config["Height"]*random.random()),
                              (bottom_left_position[0]+box_len*random.random()+box_len*x,
                              bottom_left_position[1]+box_len*random.random()+box_len*y,
                              bottom_left_position[2]+config["Altura"]*random.random()),
                              (bottom_left_position[0]+box_len*random.random()+box_len*x,
                              bottom_left_position[1]+box_len*random.random()+box_len*y,
                              bottom_left_position[2]+config["Altura"]*random.random()),
                              (bottom_left_position[0]+box_len**random.random()+box_len*x,
                              bottom_left_position[1]+box_len*random.random()+box_len*y,
                              bottom_left_position[2]+config["Height"]*random.random()],
                              config["Speed"])) ) para x in range(nWide) para y in
                              range(nHigh)]
    nodes={}
    j =
    enumerar(node_positions)
    para num, pos in j:
        nome = %s%03d" % % (Prefixo,num+1)
        nodes[nome]=AcousticNode(acoustic_event, config, nome, pos, config["Period"],
                               Nenhum, config["Move"])
    acenos de retorno
```

3.2AcousticNode.py

Este arquivo contém a descrição de um nó acústico. Dentro desta classe, as diferentes funcionalidades de um único nó são inicializadas de acordo com o arquivo de configuração. Um nó é determinado por:

- Nome e posição.
- Características de sua camada física.
- Protocolo de Controle de Acesso Médio em uso.
- Técnica de roteamento.
- Taxa de geração de pacotes e destino de pacotes.

```
classe AcústicoNode():
    def __init__ (self, event, config, label, position_or_path, period=None,
                destination=None, involuntary_moving=False):
        ""Inicialização do nó acústico.
        """
        auto.config = config
        self.name = rótulo
        eu. self.involuntary_moving de setupPath
        (position_or_path) = involuntary_moving

        # Camada física
        self.physical_layer = PhysicalLayer (self, config["PHY"], evento)

        # Camada MAC
        eu. MACProtocol = SetupMAC (self, config["MAC"])

        Camada de roteamento
```

```
        self.routing_layer = SetupRouting (self, config["Roteamento"])

        # Camada de aplicação
        auto.app_layer = ApplicationLayer(self)
        se (o período não é Nenhum):
            eu. Configuração Detransmissãoperiódica (período, destino)
```

A mobilidade de nodes é abordada da seguinte forma. Um nó móvel é definido como um conjunto de pontos e a velocidade com que ele viaja ao longo deles. A partir da classe de caminho, a posição específica de cada vez pode ser obtida. Levando em conta que os nós podem estar movendo tanto estar cientes ou sem saber, definimos duas funções diferentes:

```
def GetCurrentPosition (self):
    se não self.involuntary_moving:
        # Retorna a posição real dos nós, que não são retorno em movimento
        involuntário self.path.get_position_at_time(Sim.now())
    mais:
        # Retorna a posição inicial, a única que os nós podem saber se são retornos móveis
        involuntários self.path.get_initial_position()

def GetRealCurrentPosition (self):
    # Retorna a posição real, que pode mostrar o efeito da deriva involuntária se esse for
    o caso retornar self.path.get_position_at_time(Sim.agora())
```

3.3PhysicalLayer.py

A camada física é o arquivo chave de todo o simulador. A correção de seu conteúdo afetará toda a simulação. É por isso que um nível extra de detalhes é dado nesta seção. A camada física modela ambos, o modem e o transdutor de cada usuário, e o canal comum de todos os usuários. Este arquivo contém as seguintes classes Python.

Vários parâmetros de desempenho do sistema são medidos neste nível, incluindo a energia consumida nas transmissões, a energia consumida na escuta do canal e o número de colisões detectadas.

O modelo de canal também está contido neste arquivo. Um pacote que está sendo transmitido será atrasado de acordo com a propagação acústica e sua energia será atenuada de acordo com o modelo de perda de caminho acústico definido no mesmo arquivo.

Um desenvolvedor pode facilmente introduzir novos modelos de canal, variáveis para monitorar, funcionalidades de modem, mas há duas funções que devem ser sempre preservadas. Estes são os que são usados para se comunicar de e para a camada imediatamente acima, neste caso, a camada MAC.

```
def TransmitirPacket (self, pacote):
    ''' Função chamada das camadas superiores para transmitir um pacote.
    '''
    # É dever do protocolo MAC verificar antes de transmitir se o canal
    está ocioso # usando a função IsIdle().
    se eu mesmo. IsIdle()===Falso: self. PrintMessage("Eu não deveria
        fazer isso ... o canal não estava ocioso!")

    auto.colisão = Falso # Inicializando a
    bandeira se self.variable_power:
        distância =
        auto.multicast2distance[packet["level"]] power =
```

```
        distance2Intensity (self.bandwidth, self.freq,
        distance, self. SNR_threshold)
    outra coisa: potência = self.transmit_power #
        Potência máxima padrão

    new_transmission = Pacote de saída (self)
    Sim.activate (new_transmission, new_transmission.transmit(pacote, energia))

def OnSuccessfulReceipt (self, packet):
    ''' Função chamada das camadas inferiores quando um pacote é recebido.
    '''
    self.node.MACProtocol.OnNewPacket(pacote)
```

3.3.1 Camada física

Esta é a classe que permite a interação entre esta camada, o protocolo MAC e outras camadas acima
(por exemplo, quando se pensa em um design de camada cruzada). Aqui, todos os parâmetros especificados no arquivo de configuração sobre a camada física são inicializados. Vários parâmetros de desempenho do sistema são medidos neste nível, incluindo a energia consumida nas transmissões, a energia consumida na escuta do canal e o número de colisões detectadas.

A camada física não *pensa*, ou seja, se a camada MAC pedir a esta camada para transmitir um pacote, ele simplesmente o fará, mesmo que o canal esteja ocupado. O status do canal pode ser verificado nas camadas acima para obter informações valiosas como:

- Se o canal estiver ocioso ou não (isso deve ser verificado antes de transmitir).
- Se houve recentemente uma colisão (pode ser interessante depois de esperar por duas vezes o tempo máximo de transmissão, por exemplo).

Quando a camada MAC quiser transmitir um pacote, ele invocará a função de transmissão:

```
Função chamada das camadas acima. A camada MAC precisa transmitir um
pacote def TransmitPacket (self, packet):

    se eu mesmo. IsIdle()==Falso:
        # O protocolo MAC é aquele que deve verificar isso antes de
    transmitir a si mesmo. PrintMessage ("Eu não deveria fazer isso...
    o canal não está ocioso!") auto.colisão = Falso

    se self.variable_power:
        distância = auto.level2distance[packet["nível"]]
  potência = distância2Intensity (auto.largura de banda, self.freq, distância, auto.
                                  SNR_threshold)
    mais:
        poder = self.transmit_power

    new_transmission = Pacote de saída (self)
    Sim.ativar (new_transmission, new_transmission.transmit(pacote,
    alimentação)) Quando um pacote é recebido corretamente, este é
    passado para a camada MAC:
```

```
# Quando um pacote for recebido, devemos passá-lo para a
camada MAC def OnSuccessfulRecece (self, packet):
    self.node.MACProtocol.OnNewPacket(pacote)
```

3.3.2 Pacote contínuo

Na camada física, um pacote transmitido é modelado usando a classe 'Pacote de saída'. Como mostrado nas linhas a seguir, um pacote de saída ocupará o modem do nó para o tempo de transmissão, que é obtido de acordo com o comprimento do pacote e a taxa de bits do modem (isso é obtido a partir da largura de banda que está sendo usada e da eficiência da largura de banda, ambas especificadas no arquivo de configuração). Ao mesmo tempo, um evento será acionado em todos os nós da rede, contabilizando a recepção do pacote. Tudo isso é feito na função de transmissão na classe 'Pacote de saída'.

```
def transmitir (self, packet, power):
    # Segure o modem para modelar a operação semi-duplex de modems acústicos subaquáticos.
    yield Sim.request, self, self.physical_layer.modem

    # Taxa de bits real
    largura de banda = bitrate de largura de
    banda self.physical_layer.largura de banda =
    largura de banda *1e3*
    self.physical_layer.band2bit duração =
    pacote["comprimento"]/bitrate

    self.physical_layer.transducer.channel_event.signal(
                  {"pos": self.physical_layer.node.GetRealCurrentPosition(),
                   "poder": potência, "duração": duração, "frequência":
                   self.physical_layer.freq, "pacote": pacote})
    Segure o transdutor durante a duração da transmissão
    self.physical_layer.transducer.OnTransmitBegin()
    yield Sim.hold, self, duração
    self.physical_layer.transducer.OnTransmitComplete()

    # Solte o modem quando feito
    yield Sim.release, self, self.physical_layer.modem

    Isso é usado para tomar estáticas sobre o consumo de energia
    power_w = DB2Linear (AcousticPower(power))
    self.physical_layer.tx_energy += (power_w * duração)
```

3.3.3Arrival Scheduler

Para programar um evento sobre a recepção do pacote em cada nó, definimos a classe ArrivalScheduler. Em cada nó, isso calculará o tempo em que a recepção do pacote será iniciada e a energia que o pacote terá. Isso é feito usando a posição real dos nódulos (aquele que leva em conta movimentos involuntários), de acordo com a velocidade de propagação das ondas acústicas e modelando a perda de caminho acústico subaquático usando a expressão de Thorp para absorção acústica.

```
def schedule_arrival (self, transducer, params, pos):
    distância =  pos.distanceto(params["pos"])

    se a distância > 0,01: # Não devo receber minhas próprias transmissões
        receive_power = params["power"] - Atenuação(params["frequência"],
```

```
        distância) travel_time = distância/1482,0 # Velocidade do som na
        água = 1482,0 m/s rendimento Sim.hold, self, travel_time

        new_incoming_packet = IncomingPacket(DB2Linear(receive_power),
        params["packet"], transducer.physical_layer) Sim.activate
        (new_incoming_packet.new_incoming_packet. Receber(params["duração"))
```

3.3.4Empas

A recepção de um pacote é modelada usando a classe IncomingPacket. Neste, uma parte do conteúdo do usuário, informações sobre seu próprio status em termos de interferência também estão incluídas, ou seja, a Relação Sinal-Ruído ou SIR é monitorada durante a vida útil do pacote. Quando um pacote de entrada estiver sendo recebido, ele segurará o transdutor do usuário para o tempo de recepção.

```
def UpdateInterference (self, interference):
    se (interferência > eu. MaxInterferência):
        eu. MaxInterferência = interferência

def Receber (self, duração):
    se auto.power >= self.physical_layer.listening_threshold:
       Caso contrário, nem vou notar que existem pacotes na rede
       yield Sim.request, self,
       self.physical_layer.transducer yield Sim.hold, self,
       duração
       yield Sim.release, self, self.physical_layer.transducer

       Mesmo que um pacote não seja recebido corretamente, consumimos energia
       self.physical_layer.rx_energy += DB2Linear(self.physical_layer.receive_power) *
       duração

def GetMinSIR(self):
    retornar self.power/(self. MaxInterference-self.power)
```

3.3.5O Transdutor

Esta classe é usada para modelar o transdutor de cada usuário. O transdutor pode ser visto como uma lista de pacotes de entrada e saída. Quando um novo pacote chegar (um Pacote de Entrada),todos os pacotes da lista, que são aqueles que estão sendo recebidos simultaneamente, atualizarão seu SIR. Lembre-se que o ArrivalScheduler já calculou o poder de cada pacote em cada destino. Ao mesmo tempo, observe que se a camada MAC decidir transmitir um pacote (um Pacote de Saída) mesmo tendo otransdutor não vazio, levando em conta a operação semi-duplex de modems acústicos subaquáticos, todos os pacotes que estão sendo recebidos serão descartados ou condenados.

```
# Substituir a função "_request" do SimPy Resource para atualizar o SIR para todas as
mensagens recebidas.
def _request (self, arg):
    Devemos atualizar o activeQ
    Sim.Resource._request(self, arg)

    # "arg[1] é uma referência à instância do Pacote de Entrada que acaba de ser
    criada new_packet = arg[1]

    # Condenar quaisquer pacotes recém-recebidos à falha se o transdutor
    estiver transmitindo se auto.transmitindo:
        new_packet. Destino()
```

```
    # Atualize a auto-interferência sir de todos
    os pacotes recebidos += new_packet.power

    [i.UpdateInterference(self.interference, new_packet.packet) para i em self.activeQ]
```

Quando o tempo de recebimento tiver sido concluído, um pacote será: devidamente recebido (potência suficiente, sir suficiente), descartado devido a interferência por causa de uma colisão (potência suficiente, não suficiente SIR) ou descartado (sem energia suficiente).

```
# Substituir a função "_release" do SimPy Resource para atualizar o SIR para todas as
mensagens recebidas.
def _release (self, arg):
    # "arg[1] é uma referência à instância do Pacote de Entrada que
    acabou de completar condenado = arg[1].condenado minSIR = arg[1].
    GetMinSIR() new_packet = deepcopy(arg[1].packet)

    # Reduza a interferência geral pela auto-interferência de poder
    desta mensagem -= arg[1].power

    # Exclua isso da fila do transdutor chamando a forma Pai de "_release"
    Sim.Resource._release(self, arg)

    se não condenado e Linear2DB (minSIR) >= self.
       SIR_thresh e arg[1].power >=
       self.physical_layer.receiving_threshold:

       # Devidamente recebido: potência suficiente, não
       interferência suficiente auto.colisão = falsa
       self.on_success(new_packet)

    elif arg[1].power >= self.physical_layer.receiving_threshold:
         Muita interferência, mas energia suficiente para recebê-la: sofreu uma colisão
         se self.physical_layer.node.name ==
            new_packet["através"] ou
            self.physical_layer.node.name ==
            new_packet["dest"]:

             auto.colisão = Auto.collisions
             true.append(new_packet)
             self.physical_layer. PrintMessage("Um pacote "+new_packet["tipo"]+" para "
                                       + new_packet["através"]
                                         +" foi descartado devido à interferência.")
    mais:
       Não há energia suficiente para ser devidamente recebido: basta ouvir.
       self.physical_layer. PrintMessage ("Este pacote não foi endereçado a mim.")
```

Por fim, note que a camada física não se preocupa com a abordagem. Todos os pacotes que estão sendo recebidos serão transmitidos para a camada MAC, que novamente, é o único que toma decisões (aceitar um pacote e reagir de acordo, ou apenas descartá-lo).

3.4MAC.py

O protocolo MAC é capaz de *comandar* a camada física. Como observado anteriormente, é sua responsabilidade verificar se o canal está ocioso ou não

antes de transmitir, retransmitir, se necessário e após um determinado tempo, ou considerar ou não um pacote que tenha sido recebido, entre outros. Ao mesmo tempo, ele interagirá com as camadas acima, ou seja, a camada de roteamento.

Qualquer protocolo MAC pode ser descrito como um conjunto de regras. Diferentes protocolos MAC são definidos neste arquivo. CSALOHA, DACAP e CSMA e suas adoções para o protocolo FBR já estão incluídas na biblioteca. Não cabe a este documento explicar a forma como estes são implementados. Como no caso anterior, existem algumas funções que devem ser sempre preservadas:

```
def IniciarTransmissão (self, OutgoingPacket):
    ''' Função chamada das camadas superiores para transmitir um pacote.
    '''

    self.outgoing_packet_queue.append (OutgoingPacket) self.fsm.process
    ("send_data")


def OnNewPacket (self, IncomingPacket):
    ''' Função chamada das camadas inferiores quando um pacote é recebido.
    '''
    self.incoming_packet = Pacote de
    entrada se eu mesmo. IsForMe():
        self.fsm.process (self.packet_signal[IncomingPacket["tipo"])
    mais:
        eu. Ouvir ()
```

A classe FSM.py é usada para implementar máquinas de estado finita (comuns entre os protocolos MAC). Qualquer diagrama estatal pode ser facilmente reproduzido definindo os diferentes estados e todas as transições possíveis entre eles.

Também é comum nos protocolos MAC fazer uso de temporizadores para agendar períodos de espera ou recuo. Um temporizador acionará um evento se não for interrompido assim que o tempo for consumido.

```
classe InternalTimer (Sim.Process):
    def __init__ (self, fsm):
        Sim.Process.__init__            (self,
        name="MAC_Timer")        random.seed()
        self.fsm = fsm

    def Ciclo de vida (self, Request):
        enquanto Verdadeiro:
            yield Sim.waitevent, self, Request
            yield Sim.hold, self,
            Request.signalparam[0]
            if(self.interrupted()):
                # Apenas ignora o tempo de finalização
                self.interruptReset()
            mais:
                # Desencadeia uma nova transição no diagrama de estado
                self.fsm.process(Request.signalparam[1])
```

3.5 Camada de rodada

A camada de roteamento contém uma descrição de diferentes técnicas de roteamento. Neste momento, o algoritmo de Dijkstra pode ser usado para obter rotas mínimas

de energia, bem como o novo protocolo de roteamento de feixe focado. Como nas camadas anteriores, independentemente do protocolo, as funções que interagem com o protocolo MAC e a camada de aplicação devem ser preservadas:

```
classe SimpleRoutingTable (dict):

    def SendPacket (self, packet):
        pacote["nível"]=0.0
        pacote["rota"].append((self.node.name,
        self.node.GetCurrentPosition()) tente:
            pacote["através"] = auto[pacote["dest"]] ]
        exceto KeyError:
            pacote["através"] = pacote["dest"]

        self.node.MACProtocol.InitiateTransmission(pacote)

    def OnPacketReception (self, packet):
        Se este for o destino final do pacote,
        passe-o para a camada de aplicativo # caso
        contrário, envie-o em...
        se o pacote ["dest"] == self.node.name:
            auto.node.app_layer. OnPacketReception(pacote)
        mais:
            SendPacket (pacote)
```

Observe que à medida que o acoplamento entre o protocolo MAC e a técnica de roteamento aumenta, há mais funcionalidades cruzadas entre as classes.

3.6 Camada de aplicação

Na camada de aplicação, os pacotes podem ser gerados periodicamente e retransmitidos para as camadas inferiores. Diferentes fluxos de informação podem ser modelados. Neste momento, uma fonte de informação poisson é usada, mas isso pode ser facilmente alterado. Além disso, alguns parâmetros de desempenho do sistema são monitorados, como o atraso de ponta a ponta do pacote ou o número de saltos que um pacote fez antes de chegar ao seu destino final.

```
defTransmissão periódica (auto, período, destino):
    enquanto Verdadeiro:
        self.packets_sent+=1
        packet_ID = self.node.name+str(self.packets_sent)

        se o destino==Nenhum:
             destino = "AnySink"

        pacote = {"ID": packet_ID, "dest": destino, "fonte": self.node.name, "rota": [ ],
                 "tipo": "DATA", "initial_time": Sim.now(),
                 "comprimento":self.node.config["DataPacketLength"]}
        self.node.routing_layer. SendPacket (pacote)
        seguinte =
        poisson(período) yield
        Sim.hold, self, next
        def OnPacketReception
        (self, packet):
        self.log.append(packet)
        origin =
```

```
        packet["route"][0][0]
        se a origem em
        self.packets_received.k
        eys():
        self.packets_received[origem]+=1
    mais:
        self.packets_received[origem]=1

    atraso = Sim.now()-
    packet["initial_time"] hops =
    len(packet["route"])

    eu. PrintMessage("Pacote "+pacote["ID"]+" recebido sobre "+str(hops)+
                    " lúpulo com um atraso de
                    "+str(delay)+ "s
                    (delay/hop="+str(delay/hops)+")."
                    )

    self.packets_time.append(delay)
    self.packets_hops.append(hops)
    self.packets_dhops.append(delay/hops)
```

3.7 Funcionalidades devisualização

Por último, mas não menos importante, existem várias funções que estão incluídas no pacote para download que podem ser usadas para ilustrar os resultados e verificar rapidamente o desempenho geral do sistema. Todos eles fazem um uso extensivo do pacote MatPlotLib/Pylab e podem ser encontrados no arquivo Sim.py. A ideia principal é processar as diferentes variáveis que são monitoradas durante a simulação, a partir da estrutura contendo todos os nós.

Em Fig.3.2, o cenário para uma simulação contendo quatro nódulos ativos (A, B, C, D), transmitindo para uma pia comum, e 64 relés é mostrado. Dentro deste gráfico, podemos mostrar muitas informações:

1. A cor de um nó está relacionada com a energia que ele consumiu apenas ouvindo o canal. Ou seja, um nó cercado por nódulos ativos terá uma cor vermelha.

2. O tamanho de um nó é proporcional à energia que consumiu transmitindo pacotes na rede. Um nó maior participou de mais trocas de pacotes do que um nó menor.

3. As rotas podem então ser identificadas visualmente.

Alternativamente, a energia consumida tanto na transmissão quanto no recebimento de pacotes pode ser plotada em um diagrama de barra, como o mostrado em Fig.3.3. Fig.3.4 contém um histograma do atraso de ponta a ponta dos pacotes recebidos na pia comum. Gráficos 3D também são possíveis. Por exemplo, em Fig.3.5 é mostrada uma rede contendo um AUV e um campo de nó com 16 relés. No mesmo enredo, a rota que cada pacote seguiu está incluída.

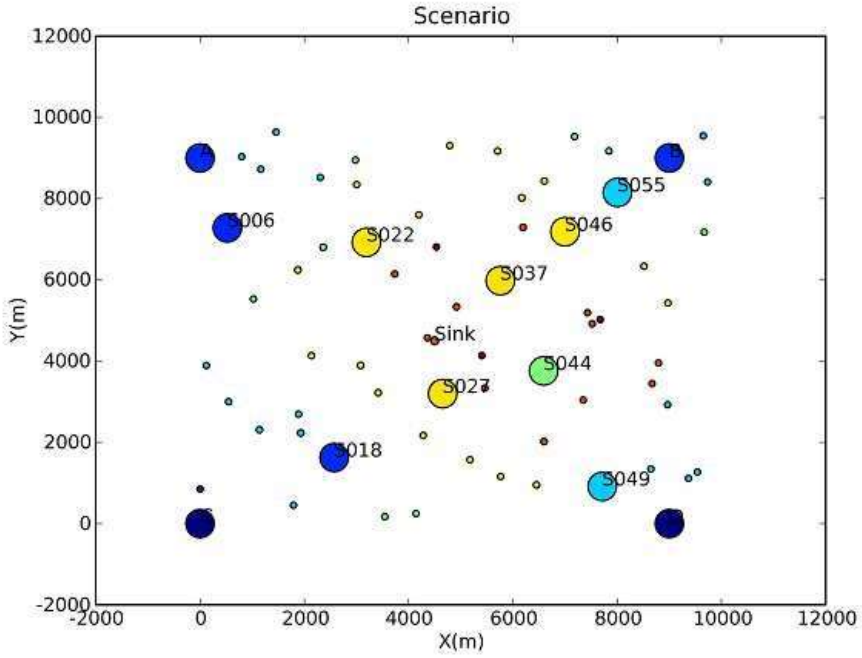

Figura 3.2: AUVNetSim: cenário para simulação e status de nó final.

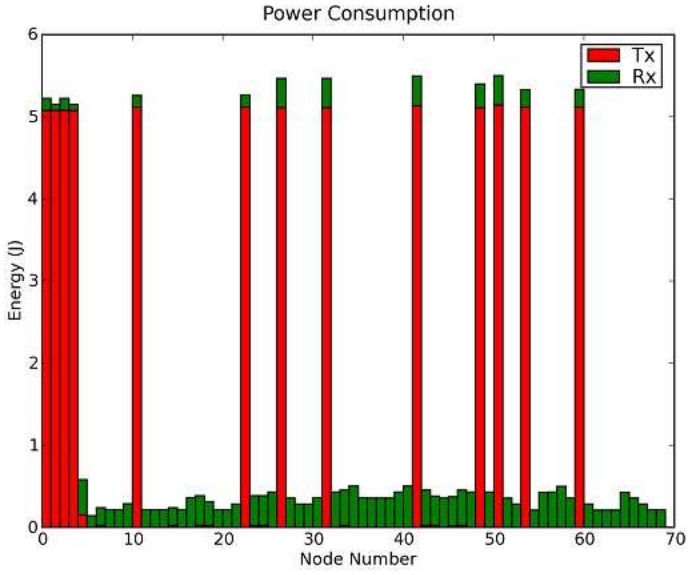

Figura 3.3: AUVNetSim: consumo de energia em cada nó do cenário.

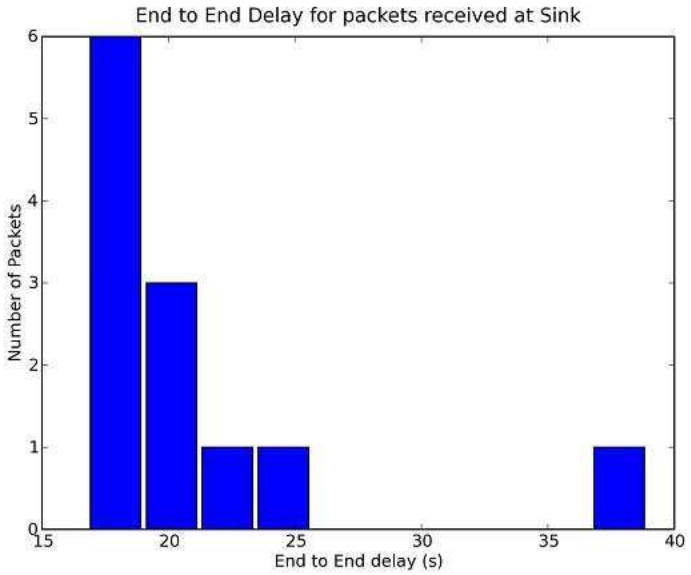

Figura 3.4: AUVNetSim: histograma do atraso de ponta a ponta dos pacotes recebidos na pia comum.

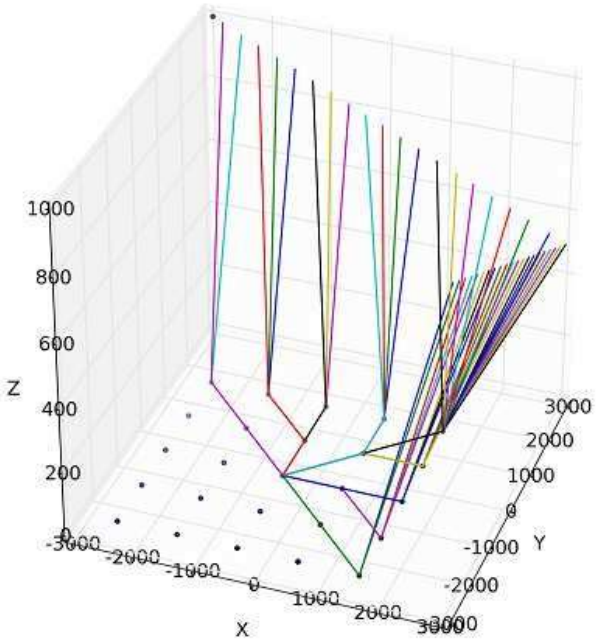

Figura 3.5: AUVNetSim: cenário 3D contendo um AUV e um campo de nó com 16 relés.

Apêndice Um

Usando AUVNetSim com MATLAB

Do ponto de vista do usuário final, o AUVNetSim pode ser visto como uma caixa preta, com algumas entradas (o arquivo de configuração) e algumas saídas. Essas saídas ou resultados, em vez de apenas serem impressos na tela, também podem ser salvos em um arquivo de texto, como mostrado em um dos arquivos de simulação incluídos no pacote simulador.

Assim, o simulador pode ser controlado *externamente,* apenas escrevendo os valores apropriados no arquivo de configuração; e *monitorado*, lendo os resultados de um arquivode texto. Isso simplifica ainda mais o uso do simulador para novos usuários finais, que não terão que lidar com python em tudo (apesar de não incentivarmos essa abordagem). Um exemplo de como invocar o simulador do MATLAB e processar seus resultados está incluído nas seguintes linhas.

A.10 script principal

No script PRINCIPAL DO MATLAB, teremos que definir os parâmetros de simulação, escrever o arquivo de configuração, iniciar a simulação e mostrar os resultados. O arquivo é independente, por favor leia os comentários.

```
% % % Arquivo de amostra para governar
externamente o AUVNetSim limpar todas as %
Limpa todas as variáveis na memória do sistema
fechar todos % Feche todas as janelas
existentes clc% Limpa o conteúdo da janela de
comando % Os parâmetros são definidos da
seguinte forma:

% nl = número de níveis de energia de transmissão ao usar o
controle de energia. nn = 1;

para n = (8) % Diferentes densidades de nó podem ser
    avaliadas criando um loop % n = posições de célula de
    grade é uma dimensão. N=n.^2 significa o número de nodes
    %.

    % Cenário
    se nn ~= 1
        limpar todo o
        status de
        carga.mat
    fim

    c = 20e3; % O comprimento do lado sobre o qual os nós são implantados
    (Área total = c.^2) h = 1e3; % Diferença máxima de altura entre nós rho =
    n.^2./c.^2; % A densidade do nó
```

```
randomgrid = 'True'; % Cenário: os nós são
implantados aleatoriamente? movimento = 'Verdadeiro';
% Cenário: os nós estão se movendo aleatoriamente?
velocidade = 0,2; % Se sim, em que velocidade?
MAC = 'DACAP4FBR'; % Protocolo MAC que será
utilizado ROU = 'FBR'; % Técnica de
roteamento que será utilizada = 120,0; %
Somente se utilizar a variação do protocolo
FBR = 0; % e seu comprimento de dados de
variação = 9600; % Período de bits = 30; %
Seconds simtime = 60 * 60; % Segundos salvar
cenário.mat c h n rho

para ni = (4) % Número diferente de níveis pode ser avaliado sequencialmente
    % Para cada densidade de nó e/ou número de níveis de energia, diferentes
    % frequência central e largura de banda
    podem ser escolhidos % selfreq % NÃO
    incluído dentro do pacote

    % Ou não, podemos querer sempre usar os mesmos
    fc = 20; %
    kHz b = 1; %
    kHz

    % Podemos querer definir diferentes níveis de potência
    usando critérios diferentes de % . cenário de carga
    setlevels.mat

    % Devemos salvar o nome do arquivo contendo os resultados de diferentes
    % simulações, a fim de carregá-lo depois se n<10 se fc<10 nome (nn,:)
    = strcat ('sim_N_',num2str(n,'%2g'),'_f_',num2str(fc,'%6.2f'),
                    '_b_',num2str (b,'%6.2f'),'',MAC,''',ROU,'.txt');
       nome de outra coisa (nn,:) = strcat
           ('sim_N_',num2str(n,'%2g'),'_f_',num2str(fc,'%6.2f'),
                     '_b_',num2str (b,'%6.2f'),'_',MAC,''',ROU,'.tx');
       fim
   mais

       se fc<10 nome (nn,:) = strcat
           ('sim_N_',num2str(n,'%2g'),'_f_',num2str(fc,'%6.2f'),
                     '_b_',num2str (b,'%6.2f'),'_',MAC,''',ROU,'.tx');
       nome de outra coisa (nn,:) = strcat
           ('sim_N_',num2str(n,'%2g'),'_f_',num2str(fc,'%6.2f'),
                     '_b_',num2str (b,'%6.2f'),'_',MAC,''',ROU,''t');
       fim
   fim

   % O arquivo de configuração está
   escrito write_conf

   % O simulador é lançado
   !python Sim.py config.conf

   os chamados +1;
fim
```

```
    n=n+1; salvar
    status.mat nome n nn
    final

% Exemplo de script no qual o cenário e as rotas que
diferentes % pacotes seguiram estão incluídos. plotrous

% Exemplo de script em que o atraso médio de ponta a ponta, o número de
% colisões a energia por bit consumida é mostrada
show_results
```

A.2Escrevendo o arquivo de configuração

O arquivo de configuração deve especificar todos os parâmetros de simulação. Alguns deles foram definidos no arquivo principal, todos os outros podem ser alterados daqui.

```
% % Grava o arquivo de configuração

% Multi refere-se aos níveis de potência, é definido como
explicado no manual %: level_name:distance_to_cover.
multi=strcat ('0:',num2str (níveis(1),'%6.2f'));
para j=1:(nl-1) multi=strcat (multi,',',num2str(j,'%2g'),':',num2str (níveis
    (j+1),'%6.2f'));
fim

% Usamos isso para criar o arquivo de configuração para o
AUVNetSimulator fid = fopen ('config.conf','wt');
fprintf(fid,'# Arquivo Config AUTOGENERADO para
AUVNetSim\n'); fprintf(fid,'# Duração da simulação
(segundos)\n'); fprintf(fid,'SimulationDuration =
%6.2f \n\n', simtime); fprintf(fid,'# Largura de
banda disponível (kHz) \n'); fprintf(fid,'BandWidth
= %6.2f \n\n', b); fprintf(fid,'# Largura de banda e
relação bitrate (bps/Hz) \n');
fprintf(fid,'BandwidthBitrateRelation = %6.2f \n\n',
1); fprintf(fid,'# Frequência (kHz) \n');
fprintf(fid,'Frequência = %6.2f \n\n', fc);
fprintf(fid,'# Transmitir poder
fprintf(fid,'TransmitPower = %6.2f \n\n',
250.0); fprintf(fid,'# Receive Power (dB)
fprintf (fid,'ReceivePower = %6.2f \n\n',-30.0);
fprintf(fid,'# Listen Power (dB) fprintf
(fid,'ListenPower = %6.2f \n\n',-30.0);
fprintf(fid,'# DataPacketLength (bits) \n');
fprintf(fid,'DataPacketLength = %6.2f #bits \n\n', datalength);
fprintf(fid,'# PHY: definir parâmetros para a camada física \n');
fprintf(fid,"PHY = {"SIRThreshold":%6.2f, "SNRThreshold":%6.2f, "LISThreshold":%6.2f,
          "variávelPower":True,"variableBandwidth":False,"level2distance":{',15.0,20.0,3.0
          );
fprintf(fid,multi); fprintf
(fid,'}\n\n');
fprintf(fid,'# MAC: defina qual protocolo estamos usando e definir
params\n'); fprintf(fid,'MAC = {"protocol":"%s", "max2resend":10.0,
"tentativas":4,
          "ACK_packet_length":24, "RTS_packet_length":48, "CTS_packet_length":48,
```

```
          "WAR_packet_length":24, "SIL_packet_length":24, "tmin/T":2.0, "twmin/T":0.0,
          "deltatdata":0.0",deltad/T":0.0, }\n\n',MAC);
fprintf(fid,'# Roteamento: definir parâmetros para a camada de roteamento\n');
fprintf(fid,'Roteamento = {"Algoritmo": "%s", "variação":%g, "coneAngle":%2g,
"coneRadius":%6.2f,
          "maxDistance":10e3}\n\n', ROU,variação,theta,levels(1)-50.0); %#ok<LTARG>
fprintf(fid,'# Nós: aqui é onde definimos nós individuais nós\n');
fprintf(fid,'# formato: AcústicoNode(Endereço, posição[, período, destino])\n');
fprintf(fid,'Nodes = [["SinkA", (5000.5000,500), Nenhum, Nenhum,["SinkD",(5000.1500.500),
Nenhum, Nenhum],
          ["SinkB", (15000.5000.500), Nenhum, Nenhum], ["SinkC", (15000,1500.500), Nenhum,
          Nenhum]\n\n');
fprintf(fid,'# NodeField: Configurar um campo de nós\n');
fprintf(fid,'# formato (grid_block_size, N_blocks_wide, N_blocks_high[,
           bottom_left_corner[, node_ID_prefix]\n');
fprintf(fid,'NodeField = (%6.2f,%g,%g, %g, (0,0,0),
"S")\n\n',c./n,n,n,n); fprintf
(fid,'Período=%6,1f\n',período);
fprintf(fid,'RandomGrid=%s\n',randomgrid);
fprintf(fid,'Moving=%s\n',movendo-se); fprintf
(fid,'Velocidade=%6.2f\n',velocidade); fprintf
(fid,'Altura=%6,2f',h); fclose(fid);
```

A.3 Mostrando os resultados

Diferentes resultados podem ser obtidos a partir do simulador. Por exemplo, pode ser interessante replicar o cenário que é traçado ao usar a função PlotScenario3D dentro das funções de visualização. Isto é o que é feito usando as tramas de script DO MATLAB.m:

```
% O simulador armazena as posições dos nódulos em um arquivo como
este posições = arquivo de importação
('Positions_DACAP4FBR_0.txt');

% Tamanho = energia de transmissão, Cor = energia receptora
scatter3 (posições(::,1),posições (:,2), posições (::,3),round (posições(::,4)./max
       (posições(::,4)*1000+1), rodada (posições(:,5)./max
       (posições(:,5)*1000+1),'preenchido')

xlim([0 c])
topo([0 c])
zlim([0 h])

set(gca,'XTick',0:c/(n-1):c)
set(gca,'YTick',0:c/(n-1):c)
set(gca,'ZTick',0:h:h)

% O simulador armazena as posições dos nódulos em um arquivo como este rotas=importfile1
('Routes_DACAP4FBR_0.txt');

% Agora podemos apenas traçar as rotas
sobre eles segurar

para i=2:comprimento
    (rotas(:,1)) x=[];
    y=[ ]; z=[ ];
```

```
    para j=1:3:comprimento
        (rotas(1,:)) se rotas
        (i,j)==-1 quebra
        fim
        x=[x, rotas(i,j)];
        y=[y, rotas(i,j+1)];
        z=[z, rotas(i,j+2)];
    plot3 final
    (x,y,z)
fim

eixo igual
xlabel ('x [m]')
ylabel('y [m]')
zlabel('z [m]')
```

Quando a simulação foi executado para diferentes densidades de nó, também pode ser interessante mostrar o desempenho do sistema em função da densidade do nó. Por exemplo, o consumo de energia por bit, o número de colisões e o atraso médio do pacote de ponta a ponta são mostrados nas seguintes linhas:

```
% Isso só faz sentido quando a simulação foi executada sequencialmente
% para diferentes densidades de nó

n0=4; % Número inicial de nós em uma dimensão
nf=16; % Número final de nódulos em uma
dimensão

% Note que todas as linhas em uma matriz devem ter o mesmo
comprimento, é por isso que temos que começar a fazer essa coisa
de concatenação... para nnn=1:1 resultados (::,:,nnn) = arquivo
de importação (strcat(nome(nnn,:),'t'));
fim

para nnn=2:6 resultados (::,:,nnn) = arquivo de
    importação (strcat(nome(nnn,:),'xt'));
fim

para nnn=7:13 resultados (::,:,nnn) = arquivo de
    importação (strcat(nome(nnn,:),'txt'));
fim

figura()
% Na terceira coluna da matriz, temos as informações sobre a % de
energia consumida no sistema.
% Na sétima coluna da matriz, temos as informações relativas a
% o número de pacotes que foi transmitido.

plot((n0:nf).^2, 10.*log10(squeeze(resultados(1,3 :)./resultados(1,7 :)
). /9600)', '-*') ylabel ('Energia por bit [dB]') xlabel ('Número de
nós [n]') grade

figura()
% Na segunda coluna da matriz, temos as informações sobre o atraso
médio do pacote de ponta a ponta.

plot((n0:nf).^2,
squeeze(resultados(1,2:)),  '-*')  ylabel
```

```
('Atraso médio de ponta a ponta [s]')
xlabel ('Número de nós [n]') grade

figura()
% Na sexta coluna da matriz, temos a informação sobre o número de
colisões.

plot((n0:nf).^2,
squeeze(resultados(1,6:)), '-*') ylabel
('Número de colisões') xlabel ('Número de
nós [n]') grade
```

Bibliografia

[1] Site do Projeto AUVNetSim, http://sourceforge.net/projects/auvnetsim/.

[2] Guido van Rossum, "Tutorial Python", http://docs.python.org/tut/.

[3] T.Vignaux, K.Muller, "The SimPy Manual", http://simpy.sourceforge.net/.

[4] Site do Projeto Python, http://www.python.org/download/.

[5] Site do Projeto SimPy, http://simpy.sourceforge.net/archive.htm.

[6] Site do Projeto Biblioteca MatPlot, http://sourceforge.net/projects/matplotlib/.

[7] Site do Projeto NumPy, http://numpy.scipy.org/.

[8] Site da TartarugaSVN, https://sourceforge.net/projects/tortoisesvn/

[9] RepositórioAUVNetSim SVN, https://auvnetsim.svn.sourceforge.net/svnroot/auvnetsim