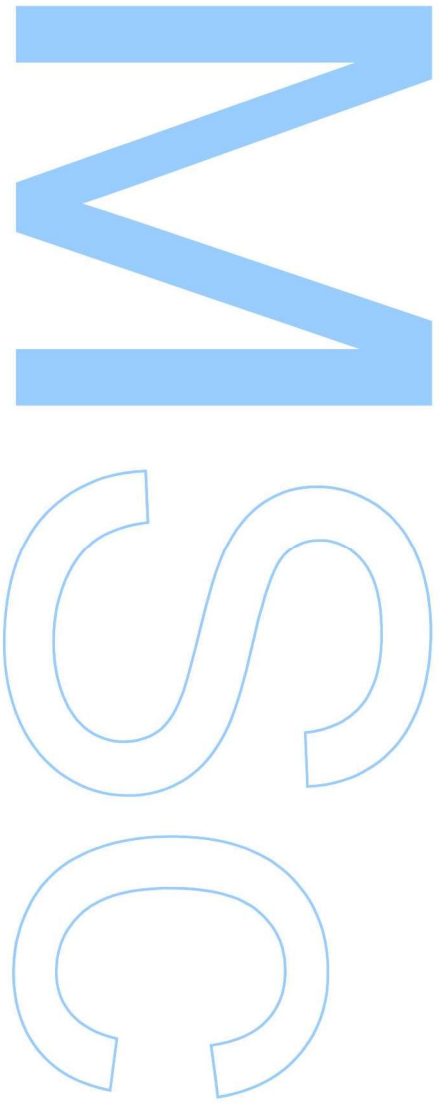
CommManager: comunicação adaptativa em redes de veículos autónomos

Pedro Seruca
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2018

Orientador
Eduardo Resende Brandão Marques, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto

Coorientador
José Carlos de Queirós Pinto, Assistente de Investigação,
Faculdade de Engenharia da Universidade do Porto



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Resumo

O Laboratório de Sistemas e Tecnologia Subaquática (LSTS) da Faculdade de Engenharia da Universidade do Porto (FEUP) tem utilizado vários tipos de veículos autónomos em diversas aplicações e cenários diferentes. Frequentemente, as missões realizadas envolvem múltiplos veículos que empregam diversos meios de comunicação, tais como modems acústicos, WiFi, GSM, ou Iridium. Estes meios de comunicação têm diferentes características e restrições, por exemplo em termos de custo, meio de propagação ou largura de banda, e empregam diferentes protocolos que são dependentes da tecnologia. Esta heterogeneidade torna complexa a sua integração coerente no software de bordo de um veículo autónomo.

Com esta motivação, foi adicionado ao software de bordo DUNE, usado em todos os sistemas do LSTS, um novo módulo denominado `CommManager` para gestão dos vários tipos de comunicação. O `CommManager` permite abstrair com uma API de mensagens o acesso a meios heterogéneos de comunicação, o roteamento dinâmico de mensagens por diferentes meios em função das condições de rede e outras do meio físico, e provém também mecanismos de *buffering* para tolerância a atrasos para entrega de mensagens, e retransmissões automáticas em caso de falha. Além da implementação do `CommManager` em si, foram feitas diversas melhorias e extensões ao software de bordo relacionado com comunicações. O software desenvolvido foi alvo de testes sistemáticos *hardware-in-the-loop* seguidos de testes de campo no Porto de Leixões.

Palavras-chave: veículos autónomos, redes heterogéneas, protocolos de comunicação, roteamento

Conteúdo

Lista de Tabelas

Lista de Figuras

Acrónimos

ACCU Android Command and Control Unit

APDL Administração dos Portos do Douro, Leixões e Viana do Castelo

ASV Unmanned Surface Vehicle

AUV Autonomous Underwater Vehicle

DTN Delay/Disruptive Tolerant Network

DUNE DUNE Uniform Navigational Environment

FEUP Faculdade de Engenharia da Universidade do Porto

GSM Global System for Mobile Communications

IMC Inter-Module-Communication

LAUV Light Autonomous Underwater Vehicle

LSTS Laboratório de Sistemas e Tecnologia Subaquática

Manta Manta Communications Gateway

ROV Remotely Operated Vehicle

RTT Round Trip Time

TCP Transmission Control Protocol

UAN Underwater Acoustic Network

UASN Underwater Acoustic Sensor Network

UAV Unmanned Aerial Vehicle

UDP User Datagram Protocol

UML Unified Modeling Language

Capítulo 1

Introdução

1.1 Motivação

A importância dos oceanos na sustentabilidade, transporte, comércio e crescimento da sociedade é incomparável, e no entanto, 95 porcento deste meio continua por explorar e descobrir. Com os avanços tecnológicos e o decréscimo dos custos de produção de componentes eletrónicos nas últimas décadas, foi possível aumentar a exploração dos recursos marinhos por parte da comunidade científica.

O Laboratório de Sistemas e Tecnologia Subaquática (LSTS) da Universidade do Porto nasceu exatamente para dar resposta à necessidade de investigação nesta área. Desde o seu início o LSTS recorre a diferentes veículos autónomos para a realização de diversas missões como mapeamento, patrulha ou investigação. Estes veículos, de total desenvolvimento por parte do laboratório, operam no ar, na superfície ou debaixo de água e partilham todos o mesmo software.

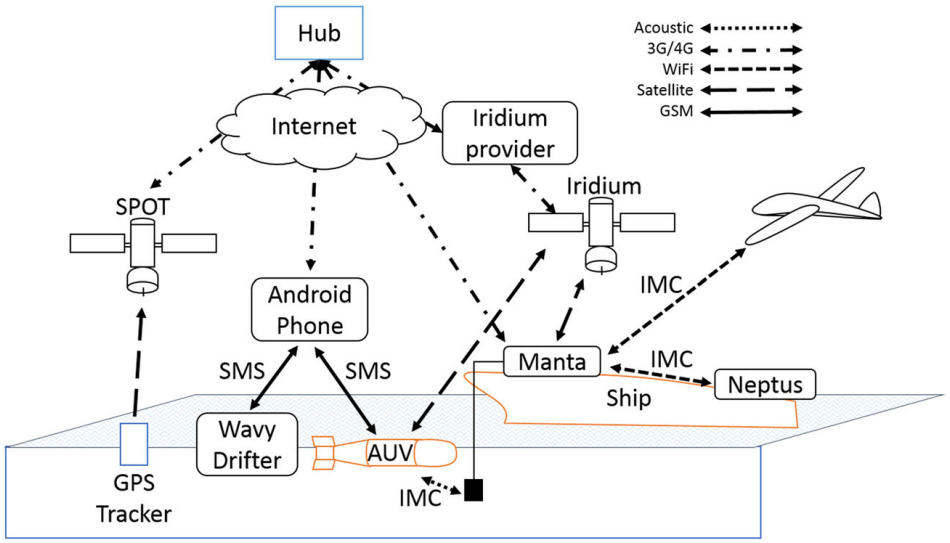


Figura 1.1: Comunicação heterogénea em redes de veículos autónomos [10]

Estes veículos comunicam numa rede complexa utilizando as diversas tecnologias de comunicação disponíveis, seja WiFi, modems acústicos, Iridium ou GSM, tal como ilustra a Figura 1.1. Esta comunicação está sujeita a diferentes tipos de falhas e restrições tais como atenuação, custo monetário, largura de banda, latência, custo energético, etc. Aliado a estas condições, alguns destes veículos costumam estar desconectados da rede por longos períodos ou demasiado longe para obter uma conexão fiável.

Esta heterogeneidade de sistemas com diferentes protocolos e restrições de comunicação exige ao software de bordo uma flexibilidade e modularidade fora do comum para dar resposta às necessidades.

1.2 Descrição do problema

A complexidade descrita acima refletia-se naturalmente no código do software de bordo dos veículos autónomos do LSTS, por exemplo com código específico para acesso a diferentes meios de comunicação, replicação de código para adaptabilidade às condições de rede e meio físico. Essencialmente, faltava um mecanismo de mediação que permitisse abstrair a heterogeneidade dos meios de comunicação através de um interface comum. Identificou-se que o mecanismo passaria por ter um gestor de comunicações com essas funcionalidades base, à qual poderiam depois, naturalmente, ser acopladas outras novas, expressas ao nível do gestor de forma abstrata em vez de replicadas e dependentes do contexto dos vários módulos responsáveis pelos diferentes meios de comunicação.

1.3 Contribuições

A dissertação descreve o desenvolvimento e validação de um gestor de comunicações chamado `CommManager` para o software de bordo DUNE usado em veículos autónomos e sistemas computacionais auxiliares do LSTS. As principais contribuições são:

- A conceção do `CommManager` como tarefa DUNE, com interface dado por uma API de mensagens que abstrai o acesso a meios heterogéneos de comunicação, algo que era feito anteriormente sem mediação e de forma diferenciada.
- A incorporação de novas funcionalidade, expressas pela mesma API, tais como o roteamento dinâmico de mensagens por diferentes meios em função das condições de rede e do meio físico, e tolerância a atrasos para entrega de mensagens.
- Alteração do código já existente para passar a usar o `CommManager` para envio de mensagens (tarefas clientes do gestor), ou para acoplar mecanismos de *buffering* e retransmissão automática em (tarefas de comunicação para os diversos meios de comunicação).

- Definição de uma nova tarefa para comunicações usando IP sobre WiFi, que permite a entrega fiável de mensagens a pedido via TCP/IP.
- Validação de toda esta funcionalidade em testes *hardware-in-the-loop* em laboratório, seguidos de testes de campo no Porto de Leixões.

1.4 Estrutura da tese

O resto da tese está organizada pelos seguintes capítulos:

- **Capítulo 2. Enquadramento:** na primeira secção damos a conhecer o LSTS e todo o seu trabalho. Depois falamos sobre projetos relacionados com o tema da dissertação. E na última secção fazemos uma análise mais aprofundada do software DUNE.
- **Capítulo 3. Desenho e implementação:** primeiro relatamos os requisitos para cumprir os objetivos propostos neste trabalho. Seguidamente falaremos da arquitetura do gestor de comunicações desenvolvido. E por fim temos uma parte mais técnica sobre a implementação.
- **Capítulo 4. Avaliação:** este capítulo descreve todos os testes ao longo dos desenvolvimentos, bem como a avaliação final aos resultados obtidos.
- **Capítulo 5. Conclusão:** no capítulo final temos uma apreciação geral do que foi conseguido, são também apresentadas ideias para desenvolvimentos futuros e termina com uma pequena opinião pessoal sobre a realização deste trabalho.

Capítulo 2

Enquadramento

Na primeira secção deste capítulo (2.1), iremos dar a conhecer o laboratório, os seus sistemas desenvolvidos, tal como o software utilizado. Na secção 2.2 falaremos sobre projetos relevantes para o tema da tese. Terminando na secção 2.3 com uma análise ao software de borde, DUNE.

2.1 Laboratório de Sistemas e Tecnologia Subaquática

Figura 2.1: Apresentação do Laboratório de Sistemas e Tecnologia Subaquática

O Laboratório de Sistemas e Tecnologia Subaquática (LSTS), criado em 1997, é um centro de investigação multidisciplinar da Faculdade de Engenharia da Universidade do Porto (FEUP), focando-se em engenharias como eletrotécnica, mecânica e informática. O laboratório é especializado em desenhar, construir, e operar veículos autónomos aéreos, terrestres, de superfície ou subaquáticos, assim como desenvolver ferramentas e tecnologias para permitir a operação destes

veículos em rede. Nos últimos 20 anos, desde o Pacífico ao Atlântico, o LSTS já completou com sucesso inúmeras missões com âmbito cientifico, militar ou arqueológico, com objetivos como, recolha de dados biológicos (salinidade, temperatura e clorofila), deteção de minas subaquáticas e vídeo vigilância, ou mapeamento e reconhecimento de áreas arqueológicas e artefactos. Esta experiência permitiu que, tanto a nível de hardware como de software, os veículos utilizados estejam em constante melhoramento e adaptados às tecnologias atuais. Damos nesta secção uma perspectiva do trabalho do LSTS.

2.1.1 Veículos Autónomos

Um veículo autónomo é um veículo não tripulado e capaz de navegar sem interação humana. Apesar do conceito não ser novo, apenas recentemente tem havido grandes avanços nesta área, impulsionados pelo desenvolvimento e acesso a melhores processadores e baterias com maior capacidade. Estes veículos são uma ferramenta indispensável à investigação em meios de difícil acesso. Habitualmente estão equipados com sensores e/ou uma câmara, podendo armazenar a informação ou transmitir em direto. Diferentes tecnologias de comunicação podem ser usados consoante o meio e o objetivo da operação, sendo o WiFi, GSM, Iridium e acústica as opções utilizadas pelo LSTS.

Os **Light Autonomous Underwater Vehicles (LAUVs)**, ilustrados na Figura 2.2, deslocam-se debaixo de água através dos seus sensores e atuadores. São considerados modulares pois permitem acrescentar sensores extra (chamados *payload*) à sua base e podem incluir os quatro protocolos enumerados acima para comunicar. Têm a particularidade de poder estar bastante tempo sem transmitir ou responder ao operador, uma vez que debaixo de água apenas as transmissões acústicas resultam e estas têm elevados custos energéticos e são lentas.

Figura 2.2: LAUVs modelo Noptilus

Os **Unmanned Aerial Vehicles (UAVs)** são veículos voadores de baixo custo, exemplos nas Figuras 2.3 e 2.4, que tipicamente comunicam por WiFi. Podem ser remotamente controla-

Figura 2.3: UAV modelo Cularis

Figura 2.4: UAV modelo Skywalker X8

dos, ou, tal como os LAUV, autónomos depois de indicado um plano de manobras. Para além das típicas missões de reconhecimento, estes podem servir como expansão de cobertura numa rede de veículos ou até fazer *data muling*.

2.1.2 Sistemas de suporte

A **Manta Communications Gateway (Manta)**, ilustrados na Figura 2.5, é um ponto de acesso portátil à prova de água e está presente em quase todas as missões. A sua principal tarefa é conectar o software de operador com os veículos autónomos e isto é conseguido através das ligações sem fios. A Manta também pode servir como ponte entre meios de comunicação heterogéneos, como por exemplo, receber informação por via acústica (debaixo de água) e retransmitir via WiFi esses mesmo dados para os outros nós da rede. Também tem capacidade de servir como hotspot usando 3G ou 4G.

Figura 2.5: Manta

O **Ripples** é um serviço web que pode ser acedido através de uma interface Web (REST API). Este serviço é usado para aceder, guardar e distribuir informação em tempo real dos sistemas em uso. O objetivo é difundir dados relevantes de um veículo para outros sistemas que não consigam

estabelecer contacto direto. Para além disto, no endereço http://ripples-web.appspot.com está disponível uma interface web que mapeia os veículos em lançamentos.

2.1.3 Software toolchain

Além dos veículos autónomos em si, o LSTS também desenvolveu de raiz um toolchain de software *open-source*[9] com 3 componentes principais: IMC, DUNE, e Neptus.

O **Inter-Module-Communication (IMC)** é um protocolo de mensagens e é especificado através de ficheiro único XML onde estão descritos todos os formatos de serialização das mensagens [6]. A partir deste mesmo ficheiro é possível gerar código C++ ou Java tornando a sua integração nos sistemas transparente. Isto permite utilizar este protocolo de comunicação tanto internamente (entre tarefas) como externamente (entre sistemas).

O **DUNE Uniform Navigational Environment (DUNE)** é o software de bordo instalado nos veículos e em outros sistemas do LSTS. É escrito na linguagem C++, de maneira a poder ser utilizado em sistemas com poucos recursos, e é independente do sistema operativo ou arquitetura em utilização. É usado para comandar, comunicar, navegar, controlar sensores e atuadores, invocando tarefas em tempo real. Essas tarefas são concorrentes, podendo trocar mensagens IMC através de um *bus* comum que as distribui. Esta distribuição é conseguida usando uma arquitetura *publish-subscribe* que funciona de maneira assíncrona. Para além disso, algumas tarefas também são modulares, ou seja, diferentes veículos (diferente hardware) podem utilizar diferentes tarefas. Isto permite que uma grande heterogeneidade de veículos tenham grande facilidade em trocar informação, assim como ter uma boa integração como o software de controlo. Descrevemos o DUNE em mais detalhe na secção 2.3.

O **Neptus** é o programa para planeamento, controlo e revisão das missões dos veículos não tripulados. É escrito em Java com distribuição para Linux e Microsoft Windows e tem uma interface gráfica para realizar todos os comandos. Foi desenhado para ser flexível, e por isso compatível com todos os veículos autónomos, sensores, cenários e operações, e para tal cada operador pode acrescentar à interface principal ferramentas personalizadas e ajustáveis para cada tipo de missão. Podem ainda ser instalados *plugins* independentes do projeto principal para adicionar funcionalidades.

Numa fase antes da realização da missão, o Neptus permite planear, simular e enviar os planos a serem executados pelos veículos. Estes planos são transmitidos como uma sequência de manobras através de uma (ou várias) mensagem IMC. Durante a missão o Neptus permite estar a par do progresso e do estado do veículo em tempo real. O operador pode controlar o desenrolar das manobras através de ações como abortar, recomeçar ou redefinir. Após a missão, os relatórios dos veículos podem ser analisados. Todas as mensagens IMC trocadas podem ser observadas, assim como gráficos sobre o estado do veículo e todos os dados recolhidos pelos sensores a bordo.

2.1.4 Comunicações / Operação em rede

Para o funcionamento de uma rede de veículos, a comunicação entre os nós é essencial. Como visto no capítulo anterior, a arquitetura desenvolvida pelo laboratório utiliza o seu próprio protocolo para transmitir mensagens entre sistemas. A Figura 2.6 representa cenários possíveis onde a troca de mensagens é essencial para o sucesso das missões [7].

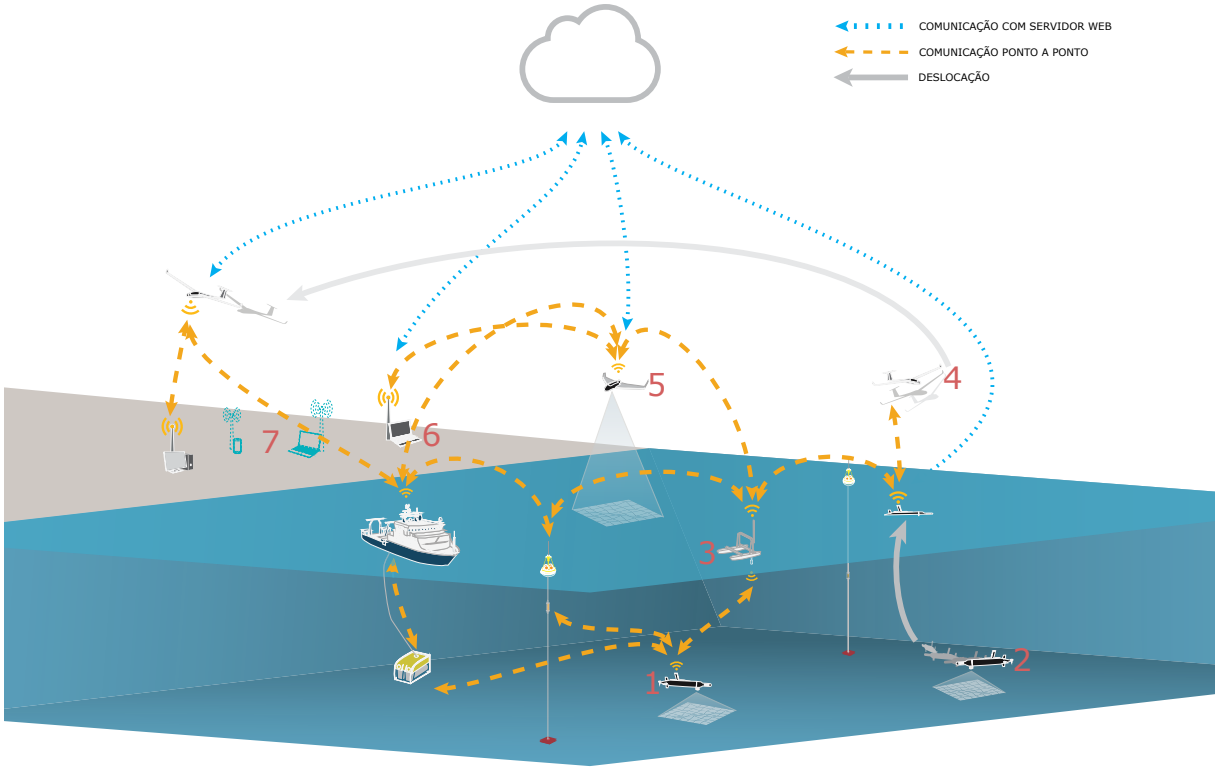


Figura 2.6: Ilustração geral de cenários de operação

Para descrevermos estas ligações e identificarmos os sistemas iremos utilizar a numeração da figura.

Como já foi referido, qualquer sistema com ligação à Internet ou através de Iridium pode comunicar com o serviço web disponível para armazenar e espalhar a informação pelos outros sistemas. Este serviço está representado pela nuvem e as trocas de mensagens pelas setas azuis. Num cenário real, dois sistemas sem comunicação direta podem partilhar dados relevantes, tais como a sua localização.

Na figura estão representados dois sistemas LAUV, identificados por 1 e 2. O LAUV 1 está a fazer mapeamento do fundo oceânico e pode transmitir os dados obtidos em tempo real para um ASV (3) operando à superfície usando comunicações acústicas. O ASV é responsável por re-transmitir os dados por uma ligação WiFi ou Iridium, por exemplo a um UAV (5), que por sua vez poderá novamente re-transmitir também os dados a um operador humano a usar o

Neptus (6). O LAUV 2 representa a combinação de dois cenários: rastreamento de espécies e *Data Muling*. Após terminar a deteção e seguimento de animais marinhos, esses dados devem de ser reportados ao operador, no entanto, como podemos ver não existem ligações disponíveis nessa localização. Assim sendo, o LAUV 2 move-se até à superfície, comunica a sua posição a um operador e este último comanda um UAV (4) para ir ao encontro e descarregar os dados do LAUV (estas deslocações estão representadas pelas setas a cinzento na figura).

O UAVs como o 4 e 5, para além de serem usados como ponte de comunicação, são também necessários para certas missões de mapeamento e patrulhamento. Finalmente, o cenário pode também contemplar (7) o uso do Android Command and Control Unit (ACCU), uma aplicação Android que permite teleguiar e monitorizar veículos autónomos.

2.2 Trabalho relacionado

2.2.1 Delay/Disruptive Tolerant Networks

Na vida real, uma missão levada a cabo pelo LSTS está sujeita a deslocações constantes, zonas com pouca ou sem cobertura de rede, distâncias entre pontos de comunicação elevadas e meios de transmissão pouco propícios à propagação de ondas. Estes fatores influenciam na qualidade da transmissão de dados, levando a muitos erros ou à perda total da conexão. Nestas situações protocolos de comunicação como o habitual TCP ou UDP têm taxas de transmissão muito baixas, ou até nem conseguem estabelecer comunicação, pelo que não são viáveis[2].

A arquitetura de rede Delay/Disruptive Tolerant Network (DTN), não assume caminhos *end-to-end*, qualquer taxa de transmissão ou de erros, ou RTTs mínimos, pelo que se aplica perfeitamente às caraterísticas de uma rede em plena missão. Para que esta nova arquitetura fosse bem sucedida era indispensável que se mantivesse a interoperabilidade com o restos dos protocolos da rede. Como tal, as DTNs situam-se entre a camada de aplicação e de transporte, criando uma nova camada chamada *Bundle*, e têm as suas próprias tabelas para identificar os nós[3].

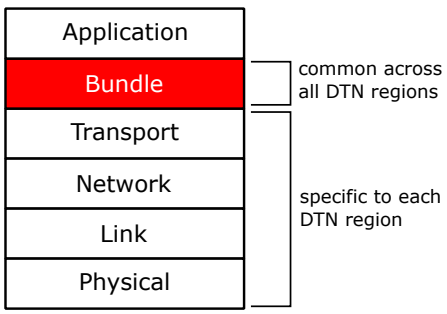


Figura 2.7: DTN *bundle layer*

Com esta nova camada é possível fazer *store and forward* das mensagens, como pode ser visto na Figura 2.8, através de um mecanismo de transferência de custódia. Neste mecanismo,

as mensagens são guardadas no nó que a enviou até que o nó recetor envie uma confirmação especial. Nesta altura o nó que recebeu a mensagem passa a ser o responsável por a encaminhar para o próximo nó, e o nó inicial pode então libertar a mensagem da memória. Desta forma, a cada salto, a mensagem fica guardada no nó até este ter a certeza que a transmitiu com sucesso, resultando num protocolo de transmissão fiável.

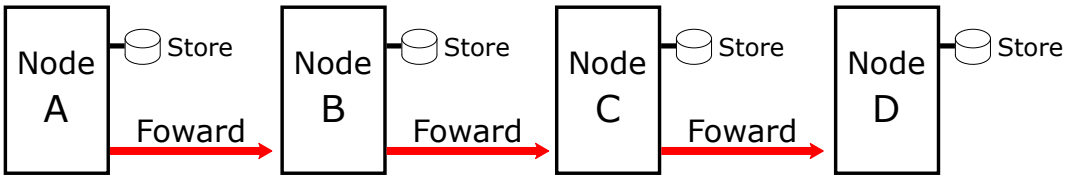


Figura 2.8: *Store and foward*

Atualmente já existem distribuições *open-source* com implementações deste protocolo, como a disponibilizada em https://sourceforge.net/projects/ion-dtn/. Em [1] o autor implementa e testa o protocolo DTN em comunicações acústicas debaixo de água.

2.2.2 Projeto SUNRISE

O SUNRISE é um projeto financiado pela União Europeia e coordenado pela universidade de Roma, La Sapienza. O objetivo deste projeto é ligar diferentes intervenientes no desenvolvimento de tecnologias para redes subaquáticas para que estas partilhem e testem os seus produtos em cooperação. o LSTS é um dos cinco parceiros iniciais do SUNRISE [8].

2.2.3 Web Enabling Ocean Vehicles

O laboratório do LSTS, inspirado nos conceitos das DTNs, desenvolveu um módulo para guardar e enviar dados de forma oportunista, com o objetivo de conectar os seus veículos ao servidor, e assim disseminar a informação mais rapidamente, dando mais controlo ao operador.

O `DataStore` é compatível com o DUNE, o Neptus e o Ripples, utilizando mensagens IMC para ser acedido. Este módulo serve de intermediário entre a criação de dados (data recolhida pelos sensores dos veículos) e a sua transmissão. O sua tarefa é partir, reduzir e priorizar o envio de dados quando requisitados, de maneira que, quando uma ligação é estabelecida apenas dados selecionados são transmitidos.

O diagrama acima, Figura 2.9 (imagem de [4]), demonstra a sequencia de mensagens IMC trocadas pelo módulo `DataStore`. Para uma análise mais aprofundada, dos algoritmos implementados neste módulo para a gestão dos dados, pode ser consultada em [4].

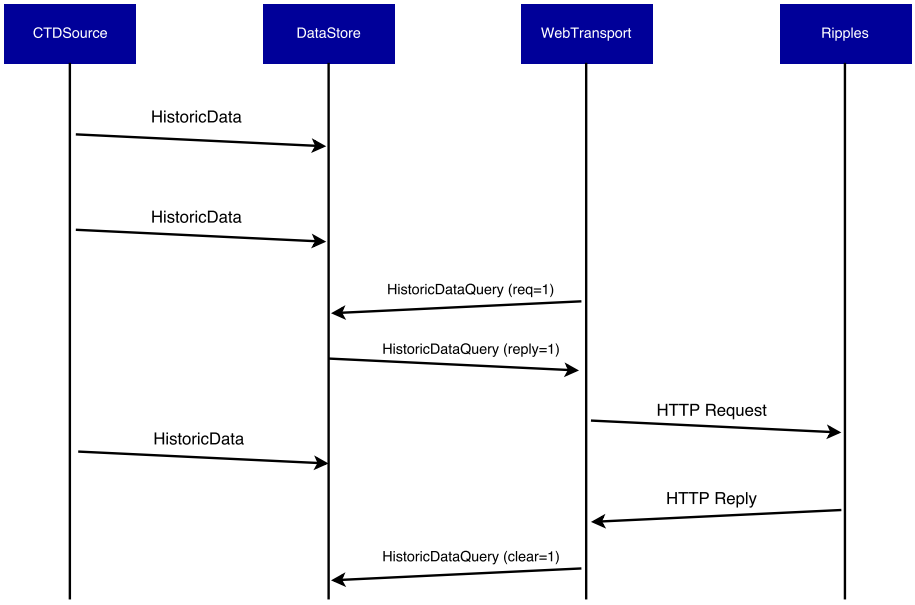


Figura 2.9: Diagrama da transmissão de dados desde a fonte até ao servidor Web

2.2.4 Protocolos de roteamento para AUVs

Desenvolver protocolos de comunicação para Underwater Acoustic Sensor Networks (UASNs) é um desafio que tem de ter em conta as características desfavoráveis do meio aquático. Em [5] investigadores de várias instituições juntaram-se para desenhar e testar novos algoritmos que se adequassem a estas necessidades. Foram testados 2 algoritmos diferentes em 3 cenários que obtiveram altas taxas de transmissões bem sucedidas apesar da sua simplicidade.

Para os testes foram utilizados três Mantas e dois LAUVs, um dos veículos teve um comportamento móvel e outro estático, simulando operações de *Data Muling*. Dois tipos de mensagem foram utilizados para este teste, `REPORT` e `DATA`, que correspondentemente são para requisitar o envio e para enviar dados. O nó móvel é o único a transmitir mensagens do tipo `REPORT`, periodicamente, e os nós fixos apenas transmitem mensagens `DATA` quando requisitado.

No primeiro algoritmo apenas é suportado comunicação entre nós diretos, ou seja, apenas um salto. Neste caso os nós estáticos descartam mensagens do tipo `DATA` e respondem ou não ao receberem `REPORT`, sendo que esta decisão é baseada da distância ao nó móvel. Desta forma quando o LAUV móvel envia uma mensagem `REPORT`, os nós fixos (dependendo da distância) respondem com mensagens `DATA` e assim o nó móvel guarda essas mensagens.

O segundo algoritmo é semelhante ao primeiro, mas é possível comunicar através de vários saltos. Para isto alguns nós fixos foram designados como roteadores. Estes nós passam a ler mensagens `DATA` e a guardar-las. Os outros nós fixos deixam de responder a mensagens `REPORT` e passam a enviar mensagens `DATA` regularmente. Desta forma, os nós fixos comunicam apenas com os nós roteadores, e estes por sua vez são responsável por retransmitir a mensagem quando o LAUV móvel passa.

No primeiro cenário o nó móvel andou sempre a volta da região de teste e foi utilizado o algoritmo 1. No segundo cenário o nó móvel começou parado, a uma distância favorável à comunicação com todos os nós fixos. Algum tempo depois começou a andar para trás e para a frente num movimento linear. Neste caso também foi usado o primeiro algoritmo. No terceiro cenário o LAUV fez o mesmo movimento que no segundo teste mas foi usado o algoritmo 2, com dois dos quatro nós fixos como roteadores.

No primeiro teste 100% das mensagens foram bem transmitidas, que pode ser explicado pela inexistência de colisões devido aos nós fixos apenas comunicarem quando o nó móvel está perto. No segundo caso a percentagem desceu para os 95 sendo que no inicio do teste todos os nós fixos estavam ao alcance do LAUV e puderam enviar mensagens simultaneamente, provocando colisões. O terceiro cenário teve uma prestação mais baixa causada pelo excesso de mensagens a enviar pelos nós roteadores por cada passagem do veículo.

2.2.5 LSTS toolchain para missões marítimas de longa duração

Numa publicação recente [7] realizada pelos investigadores do LSTS é relatado o funcionamento e desempenho do seu software em várias missões reais.

Este teste à escalabilidade do número de veículos a operar em simultâneo obteve resultados positivos visto que, diferentes sistemas conseguiram interoperabilidade com o Neptus por diversos meios de comunicação. Estes resultados provam o sucesso do protocolo IMC.

No entanto também se pôde concluir que, não só o aumento do número de veículos, como a heterogeneidade destes, contribuíram para um crescimento significativo da complexidade de gestão. Aliado a esta condição, a falta de informação em tempo real por parte de alguns veículos (por exemplo, quando estão submersos) resulta num excesso de carga de trabalho para o operador, ao qual não consegue dar resposta.

Estas falhas podem ser colmatadas através de melhorias no design do Neptus e desenvolvimentos no ACCU e Ripples.

2.3 Análise de implementação ao nível do DUNE

O DUNE foi desenhado para ser o mais versátil possível, tendo em conta que são vários os sistemas que utilizam este software, tais como, ASVs, ROVs, AUVs, UAVs e também a Manta. Como já referido anteriormente, o programa apresenta modularidade de tarefas, sendo que, cada sistema pode usar as tarefas que melhor se adaptam ao seu hardware (sensores, atuadores, módulos de comunicação). Certas tarefas são comuns a todos os sistemas, por exemplo, as classes dentro do namespace `DUNE`.

O DUNE utiliza IMC para trocar mensagens assincronamente entre tarefas e, por isso, iremos explorar a estrutura destas mensagens para explicarmos como o *bus* e a arquitetura *publish-*

subscribe funcionam. Tal como outros protocolos a estrutura de uma mensagem IMC tem um *header*, um *body* e um *footer*. O *footer* serve apenas para verificar a integridade dos dados. O *body* é o conteúdo da mensagem a ser transmitida que pode ter vários campos dependendo do tipo de mensagem. Esta mensagem tem que ser interpretada e desserializada dependendo desse mesmo tipo. O *header* está dividido em vários campos, sendo alguns deles relevantes para análise. Existem dois campos para identificar a fonte e dois para o destinatário, sendo que para cada par destes, um é o endereço, referente ao veículo, e outro a entidade, referente à tarefa. Na Tabela 2.1 podemos ver uma descrição mais técnica sobre o formato desta mensagens.

	Field	Type	Description
Header	sync	uint16_t	Used for stream synchronization and determining endianess
	mgid	uint16_t	Identifies the message type (payload fields)
	size	uint16_t	Payload size in bytes, can be used for skipping this message
	timestamp	fp64_t	Time when message was generated (UNIX epoch)
	src	uint16_t	Unique identifier of the system that generated this message
	src_ent	uint8_t	Identifier of the subsystem that generated this message
	dst	uint16_t	Unique identifier of the system this message is addressed to
	dst_ent	uint8_t	Identifier of the subsystem this message is addressed to
Body	...		
Footer	checksum	uint16_t	CRC16

Tabela 2.1: Estrutura de uma mensagem IMC.

Assim sendo, no caso do veículo estar a passar informação internamente (entre tarefas), o endereço da fonte será o mesmo que o de destino. Outro campo relevante é o de identificação do tipo, ou Message Identification Number (representado na Tabela 2.1 como `mgid`), que serve para o recetor da mensagem saber como interpretar e desserializar os dados recebidos.

A arquitetura *publish-subscribe* pode ser observada no DUNE por um par de funções que utilizam o *bus* como mediador. A função `dispatch(IMC::Message)` permite publicar uma mensagem no fim da fila do *bus*. Em contrapartida, quando uma tarefa tem declarada uma função `consume(IMC::x)`, onde x é o tipo de mensagem, está a subscrever-se para receber todas as mensagens desse tipo. Quando o *bus* retira a mensagem da fila faz a distribuição.

O *namespace* que iremos ver mais ao pormenor será `Transports`, no qual é responsável pela troca de mensagens com outros sistemas integrados na rede LSTS. Nestes casos, as mensagens são enviadas por algum dos módulos de comunicação que os sistemas disponham.

Capítulo 3

Desenho e implementação

Neste capítulo começaremos por descrever os requisitos para alcançar os objetivos propostos, secção 3.1. Seguiremos por uma visão da arquitetura do gestor de comunicações desenvolvido na secção 3.2. A implementação do trabalho é uma secção um pouco mais técnica que se pode encontrar em 3.3.

3.1 Requisitos

Devido ao número crescente de meios de comunicação suportados pelos sistemas do LSTS e diferentes implementações específicas tornou-se cada vez mais difícil desenvolver código robusto para transmissão de dados através de diferentes meios. Assim, este trabalho de dissertação teve como propósito aplicar melhorias nas comunicações entre os sistemas utilizados pelo LSTS. Desta forma, a maior parte do trabalho incidiu em dois componentes essenciais da toolchain: O DUNE, responsável pela troca de mensagens, e o IMC, base do protocolo de comunicações.

No âmbito deste trabalho, o valor acrescentado será na troca de mensagens com outros sistemas e não internamente. Para cumprir com os objetivos propostos foram definidos vários requisitos de implementação:

- Adicionar mensagens ao protocolo IMC que suportem de forma genérica a transmissão de informação por qualquer tecnologia, abstraindo detalhes específicos das várias possíveis e evitando o uso de mensagens IMC diferenciadas.
- Suportar a associação de "deadlines" a mensagens, por forma a que estas possam ser "buffered" se necessário, até uma transmissão bem sucedida para o sistema destino.
- Desenvolver uma forma de comunicação sem especificar o meio, permitindo roteamento adaptativo ou disseminação por vários meios simultaneamente.
- Dar suporte "legacy" às APIs de comunicação anteriores fazendo conversão automática dessas mensagens para a nova API.

3.2 Arquitetura

3.2.1 Organização

O gestor de comunicações, denominado `CommManager`, é uma tarefa DUNE que recebe pedidos genéricos de transmissão de mensagens e roteia esses pedidos para outras tarefas responsáveis por meios de comunicação concretos, da forma ilustrada na Figura 3.1. Este gestor é portanto uma nova camada entre os "clientes" e os módulos de comunicação.

A arquitectura é ilustrada na Figura 3.1. No topo, podemos ver várias tarefas a comunicar com o `CommManager`, que correspondem a tarefas clientes. Os clientes pedem ao `CommManager` para este transmitir informação, informação essa que é encapsulada numa mensagem específica para o meio de comunicação correspondente. Estas tarefas são responsáveis por controlar os módulos de comunicação, à exceção da tarefa `UAN` que é uma camada extra de abstração, uma vez que existem vários modelos suportados de modems acústicos, incluindo `Evologics`, `MicroModem` e `Seatrac`.

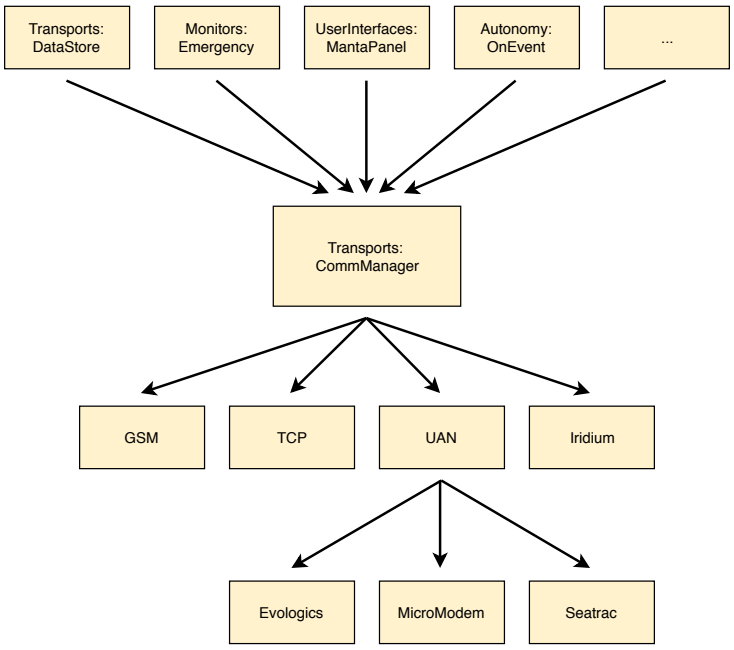


Figura 3.1: Tarefas de comunicação

Tal como qualquer outra tarefa no DUNE, a tarefa `CommManager` utiliza a estrutura "Publish/subscribe" para receber e enviar mensagens do/para o *bus*. No esquema da Figura 3.2 está representada essa transação de mensagens. Os clientes devem apenas enviar `TransmissionRequest` e esperar por uma resposta do tipo `TransmissionStatus`. As tarefas de comunicação escutam uma mensagem de pedido específica, e respondem com uma resposta correspondente, formando uma espécie de par pedido/resultado, tal como `SmsRequest` e `SmsStatus`.

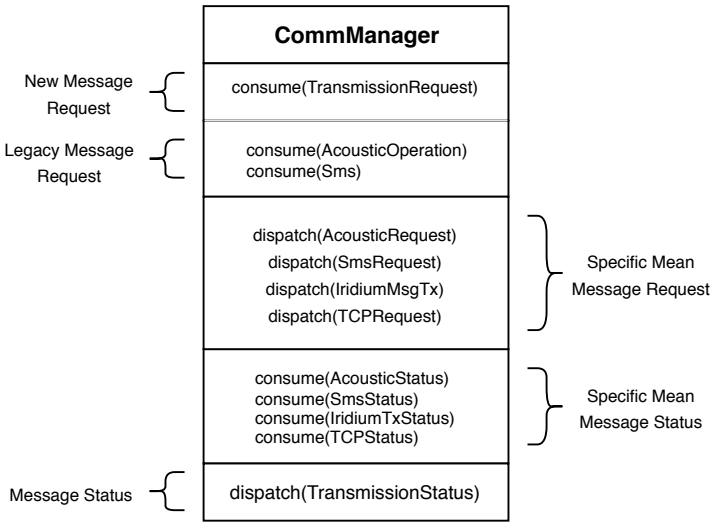


Figura 3.2: Mensagens trocadas pelo `CommManager`

Podemos também observar a subscrição do gestor de tarefas a duas outras mensagens: `AcousticOperation` e `Sms`. Estas mensagens são modelos descontinuados nesta nova versão do DUNE, contudo, devido à complexidade das operações e o número crescente de veículos que usam a LSTS toolchain (dentro e fora do LSTS), não é possível garantir que todos os sistemas tenham a última versão do software, pelo que é um requisito que estas mensagens ainda consigam ser interpretadas e respondidas. Neste caso, o `CommManager` converte-as em `TransmissionRequest` que por sua vez recebem o tratamento normal.

3.2.2 Mensagens IMC

Como já dito anteriormente, as mensagens IMC juntamente com o modelo de subscrição perfazem o sistema de comunicação do DUNE. Os melhoramentos do `CommManager`, e as modificações nas tarefas de comunicação, necessitaram também de alterações em certas mensagens e da criação de outras.

3.2.2.1 `TransmissionRequest` / `TransmissionStatus`

Quer seja um operador ou uma tarefa do DUNE a pedir para transmitir dados, com as mudanças no `CommManager`, deve ser sempre utilizada a mensagem IMC `TransmissionRequest`. Isto vai de acordo com um dos objetivos deste trabalho, uniformizar as comunicações. Este motivo faz com que esta (e a mensagem em resposta) seja a mensagem mais complexa e mais importante, pelo que vamos aprofundar um pouco mais.

Estas mensagens, apesar de já existentes no protocolo IMC não eram ainda usadas pelo sistema DUNE, tendo sido criadas apenas como rascunho de uma API de comunicações unificada. Nesta tese essas mensagens foram revistas tomando a forma descrita em seguida. Nestas tabelas

	Field	Type	Description
Body	req_id	uint16_t	The unique identifier for this message request.
	comm_mean	uint8_t (Enumeration)	Communication mean to be used to transfer these data.
	destination	plaintext	The name of the system where to send this message.
	deadline	fp64_t	Deadline for message transmission (seconds since epoch).
	data_mode	uint8_t (Enumeration)	Type of data to be transmitted.
	msg_data	message	Data to be transmitted if selected *data_mode* is *INLINEMSG*.
	txt_data	plaintext	Data to be transmitted if selected *data_mode* is *TEXT*.
	raw_data	rawdata	Data to be transmitted if selected *data_mode* is *RAW*.

(a) `TransmissionRequest`.

	Field	Type	Description
Body	req_id	uint16_t	The unique identifier for this message status.
	status	uint8_t (Enumeration)	Status of the requested message transmission.
	range	fp32_t	Distance from the ranged system in meters.
	info	plaintext	Extra information about the status.

(b) `TransmissionStatus`.

Tabela 3.1: Estrutura das mensagens `TransmissionRequest` e `TransmissionStatus`.

as linhas a verde representam os campos modificados ou adicionados.

- `TransmissionRequest`:
  - `req_id`: É um valor único, identificador da mensagem. Este valor deve apenas coincidir com o valor na mensagem de resposta.
  - `comm_mean`: É uma enumeração que identifica por que meio a mensagem deve ser enviada. Os valores possíveis são:

    * `ACOUSTIC` * `GSM` * `ANY`
    * `WIFI` * `SATELLITE` * `ALL`

  - `destination`: É o nome do sistema a qual esta mensagem deve ser transmitida.
  - `deadline`: É a validade (timestamp) absoluta, ou seja, o momento até ao qual esta mensagem deve ser transmitida.
  - `data_mode`: É uma enumeração que identifica o modo como os dados devem ser enviados. Os valores possíveis são:

    `INLINEMSG`**:** Os dados são enviados encapsulados numa mensagem IMC.

    `TEXT`**:** Os dados são texto corrido.

    `RAW`**:** Os dados são comprimidos para base 64

    `RANGE`**:** Específico para modem acústico, envia um pedido para o destino, que responde com a distância.

    `REVERSE_RANGE`**:** Específico para modem acústico, indica ao destino que deverá medir a distância a este sistema para actualizar a sua posição.

    `ABORT`**:** Específico para modem acústico, envia um pedido para o destino, que faz abortar o plano correntemente em execução..

  - `msg_data`: É o campo para introduzir dados no formato `INLINEMSG`.
  - `txt_data`: É o campo para introduzir dados no formato `TEXT`.
  - `raw_data`: É o campo para introduzir dados no formato `RAW`.

- `TransmissionStatus`:
  - `req_id`: É um valor único, identificador da mensagem. Este valor deve apenas coincidir com o valor na mensagem de pedido.
  - `status`: É uma enumeração que identifica o estado da mensagem enviada. Os valores possíveis são:

    `IN_PROGRESS`**:** A mensagem entrou em fila de espera para ser enviada. Está a ser processada.

    `SENT`**:** A mensagem foi enviada com sucesso.

    `DELIVERED`**:** A mensagem foi recebida com sucesso (nem todas as comunicações têm esta confirmação).

    `RANGE_RECEIVED`**:** A mensagem de resposta ao pedido `RANGE` foi recebida.

    `INPUT_FAILURE`**:** Erro de formação da mensagem. Tem algum campo inválido.

    `TEMPORARY_FAILURE`**:** Erro temporário. Sendo temporário, o cliente deveria tentar nova transmissão posterior.

    `PERMANENT_FAILURE`**:** Erro permanente.
  - `range`: É um valor (em metros) com a distância relativa ao sistema de destino. Este valor só está presente no estado `RANGE_RECEIVED`.
  - `info`: É a descrição aprofundada do estado da transmissão.

3.2.2.2 `AcousticRequest` / `AcousticStatus`

No panorama das comunicações acústicas, a mensagem previamente existente `AcousticOperation`, Tabela 3.2a, não cumpria os requisitos para se enquadrar no novo `CommManager`, e portanto, foi criada uma separação entre pedido/resposta resultando num par `AcousticRequest` e `AcousticStatus`, Tabelas 3.2b e 3.2c, em substituição. A mensagem de pedido passou a incluir um campo de validade (`timeout`) e um campo de identificação único (`req_id`), assim como a mensagem de resposta passou a incluir todos os campos necessários. Estas modificações não só melhoram a visualização das comunicações, como permite um tratamento muito mais rico por parte do DUNE.

3.2.2.3 `TCPRequest` / `TCPStatus`

As mensagens para transmitir por TCP também foram criadas de raiz uma vez que os sistemas não tinham comunicação utilizando este protocolo entre eles. Na Tabela 3.3 podemos ver os campos que constituem o par de pedido/resposta e entender a sua simplicidade.

	Field	Type	Description
body	operation	uint8_t (Enumeration)	Operation type. This can be a request or a status type.
	system	plaintext	The name of the system where to send this message.
	range	fp32_t	Distance from the ranged system in meters.
	msg	message	Argument for message send ('MSG') requests.

(a) `AcousticOperation`. API antiga.

	Field	Type	Description
Body	req_id	uint16_t	The unique identifier for this message request.
	destination	plaintext	The name of the system where to send this message.
	timeout	fp64_t	Period of time to send message (in seconds).
	type	uint8_t (Enumeration)	Type of data to be transmitted.
	msg	message	Argument for message send ('MSG') requests.

(b) `AcousticRequest`.

	Field	Type	Description
Body	req_id	uint16_t	The unique identifier for this message request.
	range	fp32_t	Distance from the ranged system in meters.
	status	uint8_t (Enumeration)	Status of the requested message transmission.
	info	plaintext	Extra information about the status.

(c) `AcousticStatus`.

Tabela 3.2: Estrutura de uma mensagem `AcousticOperation` e `AcousticRequest` / `AcousticStatus`.

	Field	Type	Description
Body	req_id	uint16_t	The unique identifier for this message request.
	destination	plaintext	The name of the system where to send this message.
	timeout	fp64_t	Period of time to send message (in seconds).
	msg	message	Argument for message send ('MSG') requests.

(a) `TCPRequest`.

	Field	Type	Description
Body	req_id	uint16_t	The unique identifier for this message request.
	status	uint8_t (Enumeration)	Status of the requested message transmission.
	info	plaintext	Extra information about the status.

(b) `TCPStatus`.

Tabela 3.3: Estrutura de uma mensagem `TCPRequest` / `TCPStatus`.

3.2.3 Fluxo de mensagens entre tarefas

A centralização da gestão das mensagens adicionou uma nova etapa no percurso desde que um "cliente" pede para enviar uma mensagem até que a tarefa do módulo de comunicações específico realmente envia. O impacto negativo desta adição (acréscimo no tempo de processamento) pode ser desprezado, uma vez que nunca ultrapassa os 5ms, valor que é irrelevante comparado com o tempo de transmissão das mensagens usando qualquer tipo de transmissão disponível aos

veículos (WiFi, GSM, Iridium, modem acústico).

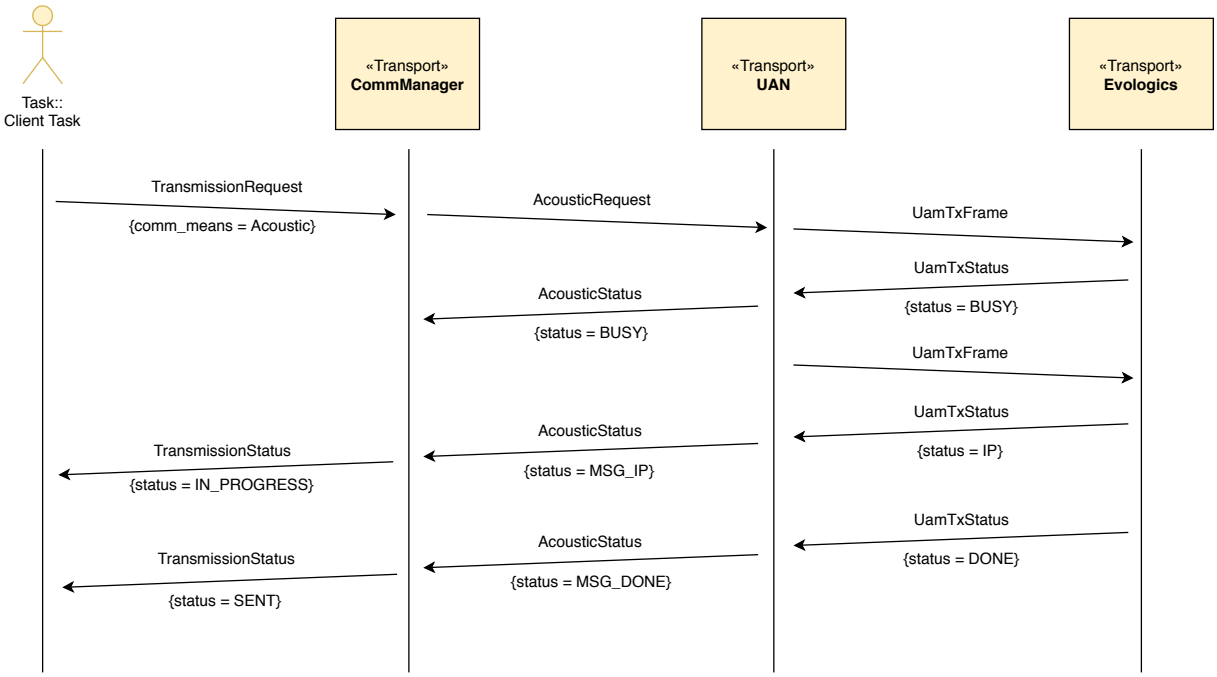


Figura 3.3: Fluxo de mensagens para uma transmissão acústica

Para facilitar a visualização e compreensão deste percurso, na Figura 3.3 está demonstrado um diagrama de sequência para uma transmissão acústica. Como ilustrado, uma tarefa cliente pede para enviar informação ao criar uma mensagem `TransmissionRequest` e publicando-a no *bus* do sistema DUNE. Um operador utilizando o Neptus deverá também utilizar esta mensagem quando pretende transmitir alguma informação. Neste diagrama de exemplo, vemos que foi pedido explicitamente para que esta mensagem fosse enviada via acústica, uma vez que o `comm_mean` está definido com valor `ACOUSTIC`. Assim sendo, a tarefa `CommManager` verifica se os vários campos são válidos. Se tudo estiver bem formado, a mensagem é convertida para o formato do meio selecionado (no exemplo, `AcousticRequest`) e mais uma vez, despachada para o *bus*. Estas mensagens são particulares para cada meio de comunicação, ou seja, cada tarefa só consome a mensagem de pedido que lhe interessa. Sendo uma amostra disso, a tarefa `UAN` apenas consome `AcousticRequest` ou, da mesma forma, a tarefa `GSM` só consome mensagens `SmsRequest`.

Nesta altura, as tarefas de comunicação de baixo nível recebem o pedido de transmissão e respondem de volta a indicar que estão a tratar do envio, como podemos ver na segunda mensagem `AcousticStatus` na figura, onde `MSG_IP` significa "*message in progress*".

Quando a tarefa que trata de enviar reporta um desfecho final, sendo sucesso ou erro, cria outra mensagem com esse estado e volta a enviar ao `CommManager` (mensagem endereçada à tarefa `CommManager` e enviada através do *bus* de mensagens do DUNE). O `CommManager` por sua vez trata de uniformizar esse estado e responder ao cliente o resultado de toda esta operação. No exemplo que temos seguido podemos ver o resultado final como uma `TransmissionStatus`

com o valor de `SENT`.

Na Figura 3.3 existe uma camada extra a todos os outros meios de comunicação. Isto deve se ao facto de os sistemas terem vários modelos de modems acústicos, o que obriga a existir tarefas específicas para cada. No entanto o percurso de uma mensagem é semelhante ao descrito acima, à exceção que a tarefa `UAN` ainda comunica com as tarefas abaixo através do par de mensagens `UamTxFrame` e `UamTxStatus`.

3.2.4 Funcionalidades

1. **Transporte específico**

Até à data deste trabalho todas as comunicações ocorriam desta forma: um cliente queria enviar uma mensagem para um destino, selecionava o meio de comunicação que achava mais indicado e pedia para transmitir. Estas mensagens podiam ser simples como um "Abort", ou planos de missão complexos.

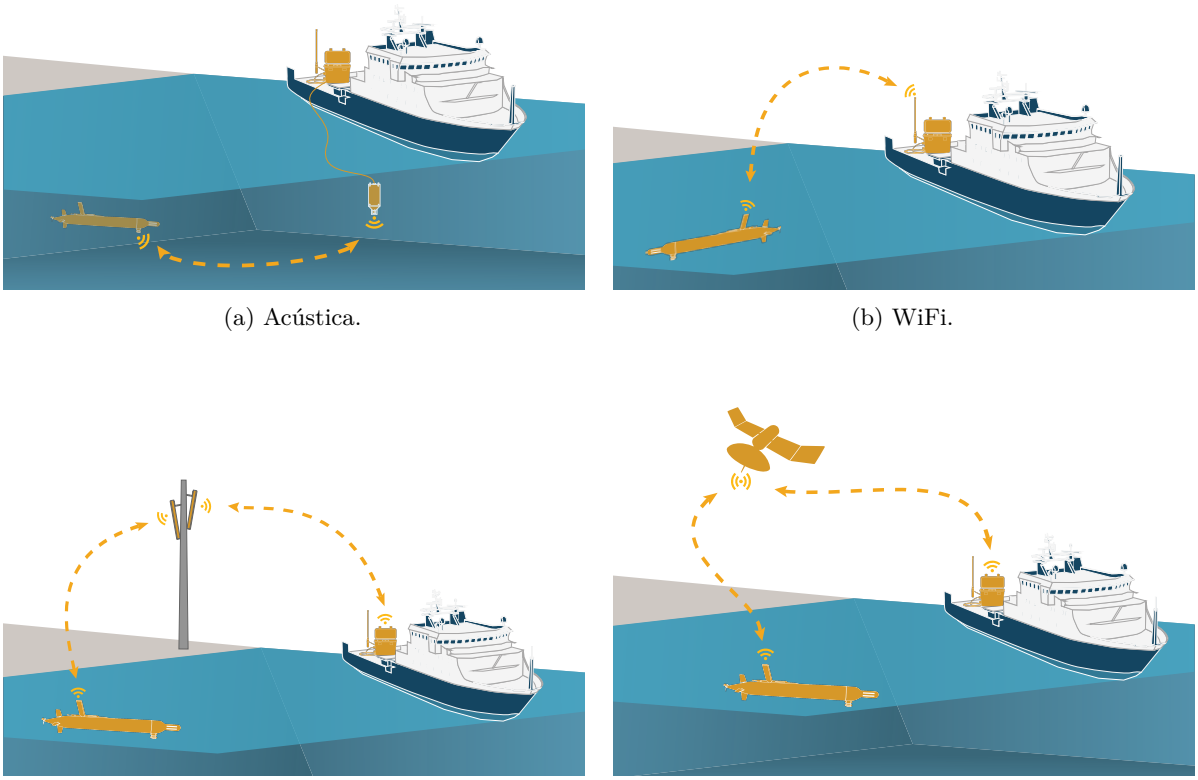
No sentido de acrescentar fiabilidade a esta funcionalidade, várias mudanças nas tarefas de comunicação e estrutura das mensagens (já mencionadas anteriormente) foram aplicadas. Uma das alterações principais, foi adicionar a possibilidade de utilizar TCP para os sistemas comunicarem entre eles. A principal vantagem (em relação ao UDP) é obtermos um "feedback" do estado da transmissão.

2. **Transporte** `Any`

A envio de dados sem especificar o meio de transmissão foi introduzido nos sistemas com os desenvolvimentos deste trabalho. Utilizando a mensagem `TransmissionRequest` e definindo o meio para `ANY`, o `CommManager` passa a ser o responsável por escolher qual o meio de transmissão mais apropriado. Durante uma missão, os sistemas utilizados pelo LSTS não têm nenhum meio de comunicação garantido, seja pelo ambiente em que se encontra, a distância aos pontos de acesso, ou simplesmente má conetividade. No processo manual, o operador tem que considerar todos estes fatores e "apostar" qual será a melhor forma de transmitir, processo que muitas vezes tem de ser repetido. O sistema implementado considera os parâmetros relevantes e opta pelo que se enquadra melhor no preciso momento, demonstrado na Figura 3.5. Desta forma tudo o que a tarefa cliente tem fazer é pedir para transmitir uma mensagem sem se preocupar como irá ser enviada.

Num contexto prático isto significa que, uma tarefa que envie mensagens pode despachá-las por vários meios sem dar conta que isso está a acontecer. Da mesma forma, permite a um operador enviar uma mensagem de maneira transparente.

3. **Transporte** `All`



(a) Acústica.

(b) WiFi.

(c) GSM.

(d) Satelite.

Figura 3.4: Meios de comunicação

Para disseminar informação por vários meios foi implementado no `CommManager` um sistema que verifica se a mensagem pode efetivamente ser enviada por cada um deles, e transmite em todos os que conseguir, representado no fluxograma da Figura 3.6. A aplicabilidade direta desta funcionalidade é em casos de emergência, onde não é relevante se o sistema recebe a mesma mensagem mais que uma vez, mas sim, garantir que ele a recebe o mais rapidamente possível.

Outro caso é no atual sistema de monitorização. Este sistema é responsável por reportar periodicamente caso um veículo esteja parado sem nenhum plano ou ação definida (entre outras condições). Este relatório está a ser despachado apenas por GSM e satélite, mas deve passar a usar o método `ALL`.

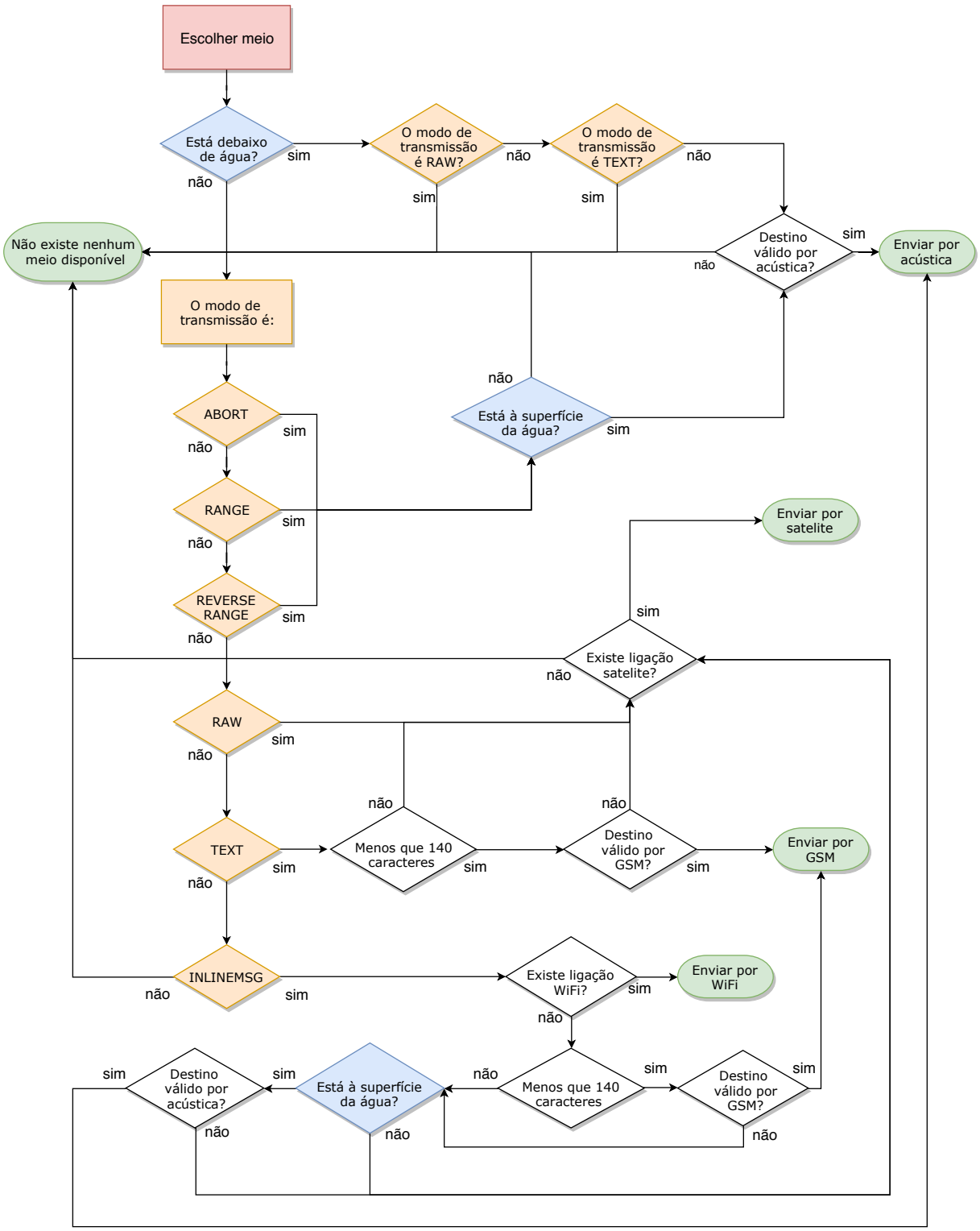


Figura 3.5: Fluxograma do algoritmo para escolher o meio de transmissão

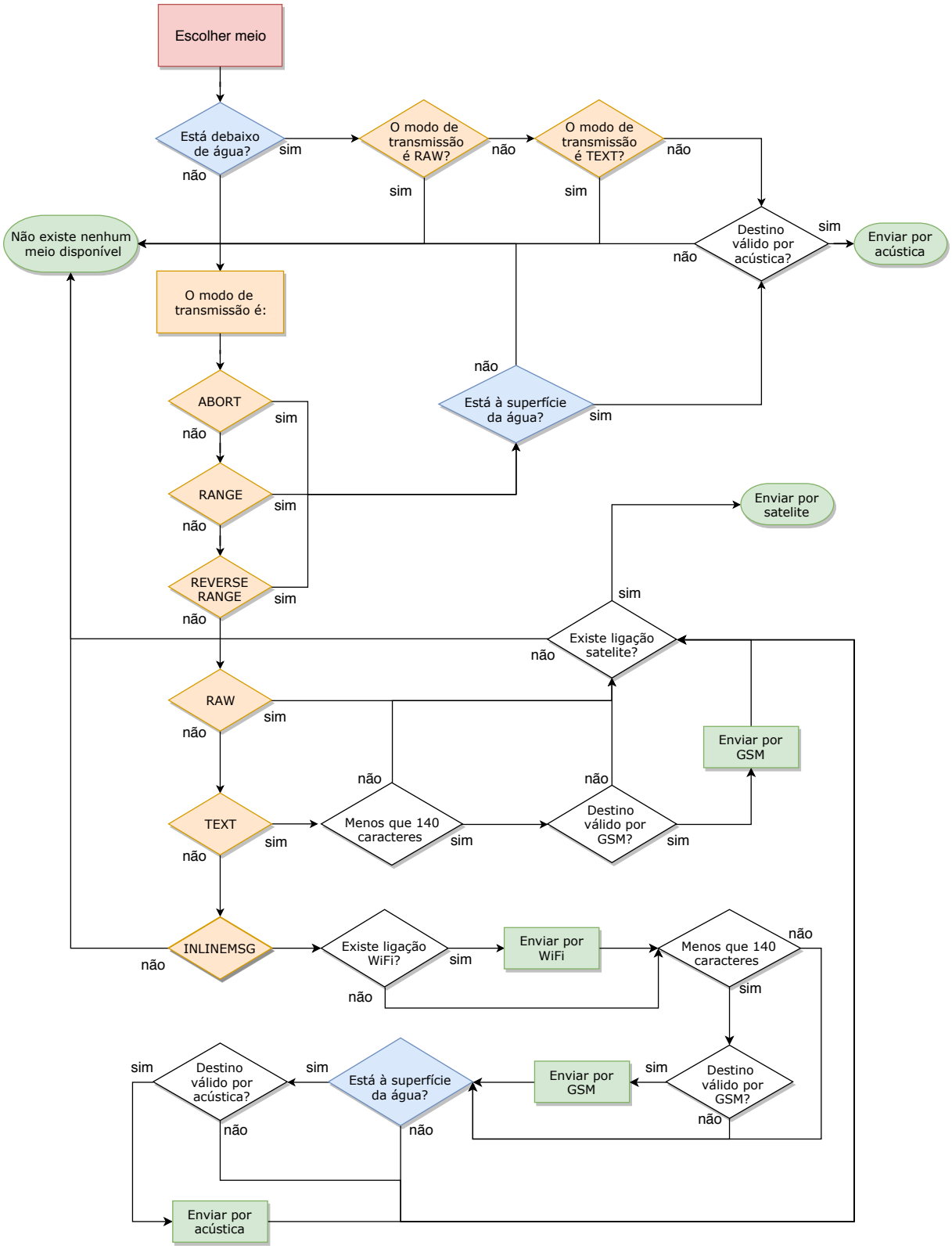


Figura 3.6: Fluxograma do algoritmo para transmitir por todos os meios

4. **Data muling**

Data muling é um dos cenários mais apetecíveis quando falamos em grupos de robôs a cooperar. Existem vários casos onde a única forma de comunicar com um sistema remoto é através do movimento de uma plataforma para "ir ao seu encontro". Por exemplo, se um submarino estiver muito longe, pode ser enviado um veículo de superfície para transmitir comandos e receber resultados. Graças às novas funcionalidades, neste momento podemos realizar facilmente planos para o conseguir. Numa explicação simples, Data mulling é quando um sistema se desloca de maneira a transmitir informação a outro sistema que não tinha conexão inicialmente. Com o novo gestor de comunicações para conseguir isto é tão simples como enviar uma mensagem com uma validade superior ao tempo de deslocamento necessário até haver ligação e enviar um plano de deslocamento para o sistema.

A validade da mensagem e o mecanismo de retransmissões fazem com que o sistema em movimento tente transmitir a mensagem periodicamente até ter tido sucesso ou a validade expirar.

5. **Algoritmo de escolha de meios de comunicação**

Para o transporte de informação `ANY` (sem especificar meio) funcionar, não basta apenas a criação de uma nova mensagem. O algoritmo que adapta as comunicações, processa vários parâmetros para considerar o meio que melhor se enquadra no momento de transmissão. Existem algumas variáveis que limitam as escolhas possíveis, como por exemplo, o meio em que o sistema se encontra, facilitando um pouco este mecanismo.

Na imagem da Figura 3.5 está representado uma forma simplificada do algoritmo. Para uma mais fácil compreensão, os elementos foram coloridos consoante o seu significado:

- Vermelho - Início do fluxograma.
- Azul - Condições pela localização que o veículo se encontra.
- Amarelo - Condições pelo modo de transmissão selecionado.
- Branco - Condições diversas.
- Verde - Fim do algoritmo / resultado do algoritmo.

3.3 Implementação

Neste subcapítulo iremos falar sobre o código acrescentado/modificado nas várias tarefas do DUNE. Podemos esperar que esta seja uma parte mais técnica do trabalho, com especial cuidado ao descrever o `CommManager` por este se tratar do centro dos desenvolvimentos.

3.3.1 Tarefa `UAN`

As transmissões acústicas eram conhecidas por ter pouca fiabilidade e (muito) baixa velocidade de transmissão, o que faziam evitar ao máximo este tipo de comunicações. No decorrer da análise ao DUNE foi detetada uma série de problemas e falhas de funcionalidades que tornavam o mau desempenho da tarefa `UAN` percetível.

Esta tarefa utilizava a mensagem `AcousticOperation` para receber pedidos de envio e respondia com a mesma mensagem o seu resultado. Isto não se enquadrava com o modelo do par pedido/resposta referido na secção 3.2.2. Além disso, esta mensagem não tinha um identificador único, nem validade. Por este motivos, esta mensagem foi logo substituída pelo novo par `AcousticRequest` e `AcousticStatus`.

Um dos maiores defeitos era o mecanismo que guardava as mensagens na fila para serem enviadas. Como estas mensagens não tinham um campo de identificação, a maneira que estava implementado só guardava a última mensagem posta na fila e assumia que a próxima resposta era dessa mesma mensagem. Na prática o que acontecia era que, esta tarefa ao receber dois ou mais pedidos de transmissão num curto espaço de tempo, apenas a última iria ter resposta (ou melhor, várias respostas, uma vez que todas as respostas seriam para a última mensagem na fila).

Para resolver todas estas falhas o processamento dos pedidos recebidos foi quase todo refeito. Desta forma, todos os pedidos são guardados numa fila e despachados à vez, as respostas com o estado de transmissão vindo do modem acústico são apenas para o pedido com a identificação correspondente e foi acrescentada uma verificação para a validade das mensagens.

3.3.2 Tarefa `TCPOnDemand`

Esta tarefa foi criada de raiz e teve como objetivo permitir aos sistemas comunicarem entre eles por TCP. Em termos de funcionamento, sempre que esta tarefa recebe um pedido de transmissão deve criar uma nova ligação TCP e, se tiver sucesso, transmitir a mensagem e fechar a conexão.

Para tal, cada sistema precisa de fazer um mapa dos outros sistemas e os seus endereços IP. Isto foi conseguido através da mensagem `Announce`. Esta mensagem é enviada periodicamente por qualquer sistema por UDP em *broadcast* e *multicast*. sendo que, como o nome diz, anunciam serviços. Desta forma, cada sistema consegue anunciar-se, dando a conhecer o seu IP e porta do serviço TCP. Ao receber uma mensagem destas, a tarefa `TCPOnDemand` procura se está presente nela este serviço, e faz um mapa com o nome do sistema que enviou e o par IP/porta. Esta mensagem tem uma validade predefinida de 15 segundos, ou seja, a tarefa não considera haver conexão se a última mensagem de anúncio recebido por parte de um sistema for há mais de 15 segundos.

Na tarefa foi implementada uma fila de prioridade para o *buffering* de mensagens, onde a comparação é a validade da mensagem, ou seja, as mensagens que tenham menor validade são

despachadas primeiro. Para além disso a tarefa tem o seu próprio sistema de retransmissão onde é configurável o número de tentativas e o intervalo entre elas.

3.3.3 Adaptações às tarefas clientes

Nas várias tarefas que pedem para transmitir mensagens foram feitas as alterações para passar a usar o `CommManager`, substituindo as mensagens antigas por `TransmissionRequest`. Consequentemente, nestas tarefas foi feita a subscrição para receber `TransmissionStatus` e o seu devido tratamento.

3.3.4 Programa de testes: dune-sendmsg

Ao longo dos desenvolvimentos nas tarefas de comunicação, foram feitos testes para ir validando as mudanças. Testes mais básicos foram conseguidos utilizando um software adicional, no entanto, havia interesse em fazer testes mais repetitivos e configuráveis. No capítulo 4 está descrito ao pormenor a utilização destes softwares.

O dune-sendmsg faz parte do DUNE e o seu objetivo é fazer testes unitários às mensagens IMC. Este programa apenas permitia selecionar uma mensagem e o endereço do sistema a injetar. A nossa ideia era modificar de maneira a que pudesse ser injetado mais que uma mensagem. Para acrescentar flexibilidade a esta ferramenta foram acrescentados novos parâmetros que podem ser visto na Tabela 3.4.

Arguments	Type	Description
address	String	Ip of the receiving system
port	int	Port of the receiving system
message	String	Message to be injected
system id	int	Id of the receiving system
destination	String	Name of the destination system
optional: count	int	Number of messages to inject
optional: interval	double	Interval between injetions

Tabela 3.4: Parametros do dune-sendmsg

Como o propósito destas modificações seria fazer testes para os sistemas transmitirem mensagens para outros, foi adicionado o campo destino, o número total de mensagens a enviar, e o intervalo entre elas. O número de mensagens e intervalo são argumentos opcionais, sendo que estando predefinidos com 1 e 0, respetivamente.

Numa análise de implementação, primeiro o programa cria todas as mensagens e coloca-as numa fila. Depois tem um ciclo que retira as mensagens dessa fila até ficar vazio, e a cada mensagem envia-a por UDP. Este ciclo tem um simples `sleep` no inicio para fazer o intervalo

entre transmissões.

Um exemplo para testar as comunicações WiFi de um sistema local para o sistema com nome "manta-1", em que são enviadas 100 mensagens com um intervalo de 2 segundos seria:
`./dune-sendmsg 127.0.0.1 6002 teste_wifi_1 36870 manta-1 100 2`

Os testes criados utilizaram todos `TransmissionRequest` com os vários meios de transmissão possíveis.

3.3.5 Tarefa `CommManager`

O `CommManager` foi, propositadamente, a tarefa central desta dissertação. Um dos principais objetivos do trabalho era ter um só gestor de comunicações, que abstraísse toda a complexidade das transmissões de um "cliente". Isto realizou-se no `CommManager`. Desde então, um cliente apenas deve pedir para transmitir informação através de `TransmissionRequest` e esperar por uma resposta `TransmissionStatus`.

A tarefa em questão já existia, mas era somente um "esqueleto" em que apenas se podia transmitir por GSM e satélite, e não estava activada no branch estável de desenvolvimento. O funcionamento do esqueleto foi utilizado como base para os desenvolvimentos e, os seus componentes principais, foram mantidos. Antes de falarmos sobre as modificações implementadas, iremos falar deste esqueleto e do que foi aproveitado.

Inicialmente o `CommManager` apenas servia para converter uma mensagem `TransmissionRequest` para uma mensagem específica, neste caso, `SmsRequest` ou `IridiumMsgTx`. Como esta tarefa na prática não transmite nenhuma mensagem, não existe nenhum fila de espera, assim sendo, estas mensagens são imediatamente processadas para a conversão e então despachadas. No entanto, a tarefa que ficou de transmitir a mensagem irá responder com o estado do envio, e então é preciso saber a que mensagem esta resposta corresponde.

Como as tarefas clientes enviam mensagens com os Id's definidos por elas próprias, existe a possibilidade de colisão por parte das mensagens de resposta. Num sentido menos técnico, podia haver estados de mensagens falsos, ou seja, não era viável utilizar estes Id's. Para resolver esta situação foi criado um mapa de mensagens enviadas. (Este modelo também é utilizado nas restantes tarefas de comunicação)

O mapa de mensagens, como exemplificado na Figura 3.7, funciona de uma maneira bastante simples: um cliente envia uma `TransmissionRequest` que é consumida, depois esta tarefa converte para uma mensagem de meio específico, como no exemplo `SmsRequest`, e substitui o Id orinal por um Id único criado pelo `CommManager`, mais uma vez, no exemplo este é o Id 36. Para guardar esta mensagem é só acrescentar uma entrada no mapa com o par Id/`TransmissionRequest`.

Quando é recebido uma mensagem de estado terminal, ou seja, o resultado final da trans-

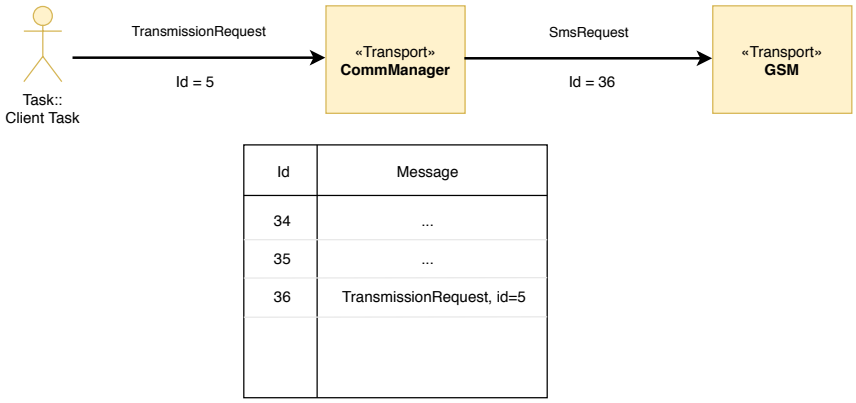


Figura 3.7: Mapa de mensagens do `CommManager`

missão, o `CommManager` procura no mapa pela "key" e encontra a mensagem de pedido que corresponde. Desta forma a tarefa sabe a quem enviar a `TransmissionStatus` de resposta, e limpa essa entrada do mapa.

Para além deste mecanismo, do esqueleto inicial foi aproveitada também a funcionalidade de remover mensagens expiradas. Como já vimos, as mensagens `TransmissionRequest` tem um campo de validade. Esta funcionalidade percorre o mapa periodicamente e verifica a validade das mensagens. Se uma mensagem já estiver expirada é despachado o resultado de "timeout" e limpa do mapa.

Nos próximos parágrafos seguiremos a ordem cronológica de implementação para dar uma melhor perspetiva das alterações que foram produzidas e os seus desafios.

Logo no primeiro contacto com esta tarefa foi acrescentado o envio por acústica seguindo as guias dos outros dois métodos. Nessa altura foi levantado um problema, já referido no subcapítulo 3.2.1, consequente da implementação de novas mensagens. Citando desse capítulo, "não é possível garantir que todos os sistemas tenham a última versão do software", mas é indispensável que as comunicações consigam ser estabelecidas, por motivos de segurança. Num exemplo concreto, o software Neptus ainda não tem compatibilidade para as novas mensagens. Numa operação real isso significaria a perda total de comunicações acústicas e por GSM.

No sentido de colmatar este problema foram introduzidas funções de conversão dos modelos descontinuados para os novos. Ainda assim isto não resolvia todos os problemas, nomeadamente a resposta acústica. Supondo que um sistema que envie uma mensagem `AcousticOperation` (descontinuada) não tem suporte para o par `AcousticRequest` / `AcousticStatus`, seria inútil responder a essa mensagem dessa forma (que seria a maneira normal de responder à mensagem convertida). Desta forma foi utilizado uma estrutura semelhante ao mapa de mensagens do `CommManager` falado acima, mas apenas para mensagens acústicas antigas. Assim, quando a tarefa consome uma mensagem `AcousticStatus` vai procurar no mapa "geral" e se não encontrar vai procurar no mapa de mensagens acústicas antigas, despachando `TransmissionStatus` ou `AcousticOperation` respetivamente. Toda esta funcionalidade pode ser desativada nos ficheiros de configuração.

Por esta altura, tínhamos os 3 meios de comunicação associados ao `CommManager` e razoavelmente testados. Foi então produzida a tarefa `TCPOnDemand` e as suas mensagens foram adicionadas como um novo meio de comunicação: `wifi`.

Para transmissão adaptativa de mensagens (`ANY`) foi necessário processar informação do estado do veículo contida nas seguintes mensagens:

- `VehicleMedium`: É a mensagem que reporta onde o sistema se encontra. Os valores possiveis são:
  - `Underwater`
  - `Ground`
  - `Unknown`
  - `Water`
  - `Air`

  Os tipos de comunicação são dependentes do meio onde se encontra o sistema. A condição inicial do algoritmo de escolha é exatamente saber se o sistema está debaixo de água, onde apenas comunicação acústica funciona.
- `RSSI`: É um valor com a potência do sinal recebido. Para GSM ou satélite é necessário saber se temos ou não rede com o ponto de acesso mais próximo. Dependendo de quem emite esta mensagem, é possível saber se conseguiremos transmitir esta mensagem ou não.
- `Announce`: É uma mensagem vinda de outros sistema que anuncia serviços. Esta mensagem ocorre periodicamente e neste caso serve para saber se temos ligação WiFi com outros sistemas.

Para GSM ainda temos uma restrição particular, o limite de carateres (ou bytes em termos absolutos). Este protocolo só permite enviar 140 carateres por mensagem, de modo que, se a mensagem tiver mais que isso, não deve ser transmitida por GSM.

Outro grande fator é o modo de transmissão da mensagem. Apesar de grande parte das transmissões utilizarem `InlineMsg`, o modo `Raw` e `Text` ainda não são compatíveis com todos os meios de comunicação, o que restringe o seu uso.

Nesta fase do desenvolvimento do algoritmo ainda havia uma falha que não tinha sido prevista. Antes desta implementação do `CommManager`, para enviar uma mensagem por GSM o campo de destino era o número de telefone pretendido. Com a uniformização das comunicações este campo deve ser sempre preenchido com o nome do sistema pelo que se torna incompatível.

A solução encontrada passou por duas implementações diferentes que se complementam uma a outra. Em primeiro lugar foi introduzido um novo ficheiro de configuração para os números de GSM, ou seja, foi criado um ficheiro para completar com os pares “nome do sistema” / “número de telefone”. Assim, a tarefa lê este ficheiro e cria um mapa com os pares. Quando uma transmissão é pedida, é verificado se existe o número para o destino escolhido, antes de ser decidido enviar por GSM. Uma grande vantagem desta implementação é não ser preciso mudar código para acrescentar ou modificar novos destinos.

Contudo, como falamos nem operações de múltiplos sistemas, isto significa que é preciso atualizar esta tabela em cada um deles. Para melhorar esta situação foi criada uma alternativa com o mesmo objetivo de criar uma mapa de números GSM. Isto exigiu alguma investigação sobre o funcionamento dos cartões SIM e comandos AT.

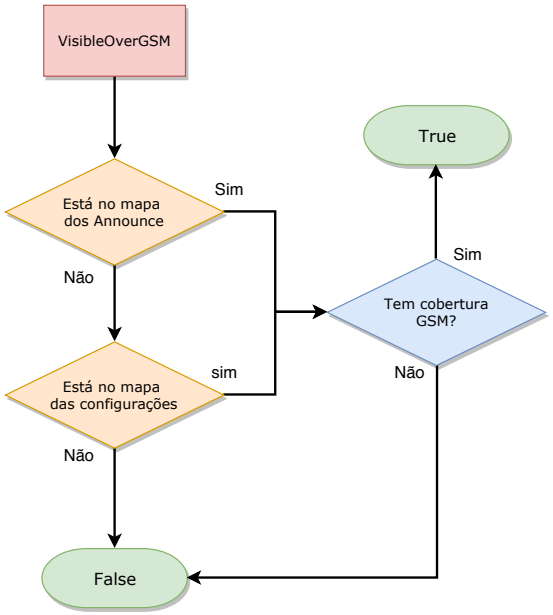


Figura 3.8: Fluxograma da funcção `VisibleOverGSM`

Os cartões SIM possuem várias listas de contactos telefónicos (não só a que estamos familiarizados), e uma das quais, serve para guardar o próprio número. O sistema desenvolvido utiliza esta lista para ler o número do cartão, e tirando partido da mensagem `Announce` falado à pouco, envia o número para todos os sistemas na rede. Mais uma vez, o `CommManager` recebe estas mensagens e constrói um mapa adicionalmente ao referido na solução anterior.

A função que verifica se um destino está disponível por GSM pode ser representada pelo diagrama da Figura 3.8. Para uma análise a este algoritmo ver a Figura 3.5.

Já referido no capítulo 3.2.4 em 3 o transporte por todos os meios disponíveis pode fazer sentido quando pretendemos, a todo custo, entregar uma mensagem, como num caso de emergência. A implementação deste método utilizou as funções criadas para o algoritmo de escolha de meio. No fluxograma da Figura 3.6 podemos ver certas diferenças. É de salientar que neste, as caixas quadradas a verde não são terminais, ou seja, têm continuidade.

Cumprindo os objetivos, a parte final do trabalho foi adicionar as retransmissões quando o envio não teve sucesso. Da mesma forma que esta tarefa passou a ser responsável pelas transmissões de mensagens para outros sistemas, foi-lhe também acrescido o reenvio de mensagens, tirando o encargo às tarefas clientes.

A nível de implementação, o código exigido para este mecanismo não foi vasto. A tarefa limitava-se a apagar a mensagem de pedido ao receber uma mensagem de estado final de insucesso. Com as alterações, ao receber uma mensagem destas, antes de apagar a mensa-

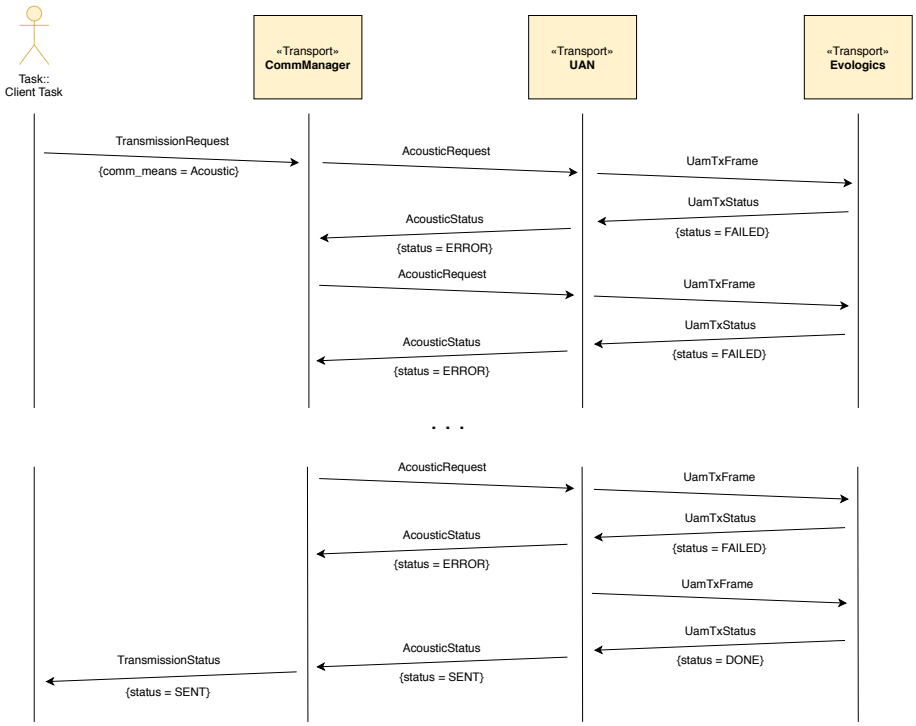


Figura 3.9: Diagrama de exemplo do mecanismo de retransmissões

gem do mapa, o `CommManager` coloca uma cópia numa lista de mensagem para retransmitir. Periodicamente, esta lista é esvaziada, e as mensagens são reintroduzidas como se fossem um pedido habitual. Estas pequenas mudanças só implicaram algumas modificações ao emitir o resultado final ao cliente (`TransmissionStatus`).

De notar, que estas mensagens estão sujeitas a todo o processamento normal, tais como verificação de validade, mapeamento de ID's e restantes mecanismos. Uma mensagem não é retransmitida se a sua validade já tiver expirado ou se tiver alguma má formação.

Capítulo 4

Avaliação

Ao longo do desenvolvimento das alterações às tarefas de comunicação, foram feitos testes, em simulação, para verificar o seu desempenho. Estas verificações requeriam um ambiente de teste personalizado para cada módulo de comunicação. Para tal foram utilizados vários softwares / utilitários para simular a transmissão de mensagens por parte dos sistemas:

- **IMC MessageSender:** Permite criar uma mensagem IMC (incluindo os headers) e enviar a um destino a través de UDP. Utilizando este software podemos inserir no veículo uma mensagem e assim simular que o veículo gerou uma mensagem para transmitir para outro sistema.
- **DuneSendMsg:** Este utilitário é normalmente utilizado para testes unitários, criando e adicionando no bus mensagens IMC. Contudo foi modificado para injetar mensagens nos sistemas. Adicionalmente, vários parâmetros foram acrescentados para tornar os testes mais flexíveis e escaláveis, tais como, número de mensagens a transmitir e intervalo entre elas.

Durante os testes iniciais foi utilizado o IMC MessageSender e injetada apenas uma mensagem transmissionRequest com o campo comm_mean definido para o meio a ser avaliado. De seguida, usou-se o utilitário DuneSendMsg tirando proveito da escalabilidade deste, criando testes com múltiplas transmissões.

Posteriormente, vários sistemas foram levados para o porto de Leixões, local de testes habitual do LSTS, para serem testados em execução real.

4.1 Testes em simulação - Hardware-in-the-loop

Neste subcapítulo iremos descrever os testes com sistemas simulados. Foram utilizados durante o desenvolvimento testes unitários e testes mais repetitivos para ir validando todas as alterações no código. O DUNE tem vários perfis de execução das tarefas, entre os quais "Simulação" e "Hardware". Para cada um dos casos a seguir apresentados ativou-se o modo "Hardware" apenas para os sistemas que estavam a ser testados, ficando os restantes em modo de simulação.

Aqui não iremos avaliar o desempenho destes testes mas sim descrever como foram executados num ambiente laboratorial. As próximos secções foram nomeadas por cada meio de transmissão contemplado na mensagem do tipo `TransmissionRequest`.

4.1.1 GSM

Para o DUNE poder enviar mensagens via GSM, em simulação, foi utilizado um módulo conectado por USB ao computador. Foi atribuído um canal para a placa usando o ficheiro de configuração do veículo simulado. Desta forma o DUNE pôde iniciar a tarefa e ter acesso às capacidades do módulo.

Figura 4.1: Testes - Módulo GSM

4.1.2 Acústica

A realização dos testes acústicos em laboratório necessitou de dois sistemas capazes de receber e transmitir as mensagens. Para tal, foram utilizadas duas Mantas com modems Seatrac. O DUNE foi executado em cada manta como veículos diferentes em simulação (à exceção da tarefa responsável pela comunicação com o modem acústico).

Validar as mudanças à tarefa `UAN` implicou transmitir 3 tipos de mensagens acústicas: `RANGE`, `ABORT` e `INLINEMSG`. Para além do IMC MessageSender, foi possível também testar estas mensagens utilizando os botões de interface da Manta.

4.1.3 WiFi

A tarefa criada para transmitir mensagens por TCP foi verificada da mesma forma que os outros meios de comunicação. Para a realização do teste foi apenas preciso ativar a tarefa do veículo

Figura 4.2: Testes - Mantas

Figura 4.3: Testes - Modems Seatrac

simulado nas suas configurações e injetar a mensagem a transmitir com destino a algum sistema que tenha um servidor TCP a correr, sendo que neste caso foi o Neptus.

4.1.4 Iridium

As comunicações por Iridium não foram testadas por motivos logísticos. Ainda assim, é de notar que esta tarefa não foi modificada, uma vez que o `CommManager` já tinha suporte para este meio.

4.1.5 Any

Apesar da mensagem do tipo `ANY` não ser nenhum meio de comunicação em concreto, foram feitos vários testes em simulação para validar o envio de mensagens utilizando este modo. Cada teste previa a utilização de um meio em específico.

Inicialmente, para validar a transmissão por WiFi não foi preciso nada mais do que repetir o teste realizado anteriormente mudando o meio de transmissão da mensagem para `ANY`. Como este tinha ligação e o meio WiFi é preferível a qualquer outra comunicação a mensagem foi corretamente enviada.

Para enviar por GSM, foi preciso utilizar as configurações de hardware descritas acima. Um teste simples e eficaz foi introduzir na lista de destinos GSM disponíveis um sistema único, ou seja, que não tenha mais nenhuma forma de comunicação. Neste caso foi um telemóvel pessoal introduzido. A mensagem a enviar injetada tinha esse número como destinatário pelo que foi corretamente enviada por SMS.

Os testes para acústica foram um pouco mais exigentes. Para além da montagem do hardware foi preciso fazer o sistema que envia a mensagem crer que estava dentro de água para preferir a

utilização de comunicação acústica em vez de WiFi. Para tal, utilizou-se o DuneSendMsg para injetar mensagens periodicamente do tipo `Vehicle Medium` com o valor `UNDERWATER`. Com o IMC Message Sender foi injetada a mensagem a transmitir, que foi expectavelmente enviada por acústica.

4.2 Testes na APDL e cenários de uso

Os testes em campo foram divididos em 4 etapas. As duas primeiras tiveram como objetivo validar as tarefas de comunicação no campo, enquanto que os dois últimos tiveram como objetivo simular cenários reais de operação que envolvem comunicação entre sistemas.

Estas operações foram executadas no porto de Leixões (APDL), num pontão onde é habitual o LSTS operar. Os sistemas utilizados compreenderam em 2 veículos, 1 manta e um portátil com Neptus.

4.2.1 Cenário 1: Validação do `CommManager`

O teste inicial teve como objetivo verificar que qualquer mensagem, seja por que meio for, consegue ser entregue em condições ideais. Este teste serve também para validar o trabalho desenvolvido em sistema reais utilizados em operações.

Este cenário envolveu comunicações entre um veículo na superfície da água e a Manta. Os sistemas encontravam-se separados apenas a poucas dezenas de metros. Através do software DuneSendMsg foi introduzida uma mensagem no veículo, previamente criada, para ser transmitida pelos diversos meios.

A análise dos resultados mostraram que, tal como esperado, todas as mensagens foram entregues com sucesso. Também já previsto, a mensagem enviada pelo meio `Any` foi corretamente despachada por TCP.

4.2.2 Cenário 2: Simulação de transmissão de mensagens durante um plano

O segundo teste teve as mesmas condições iniciais que o anterior, um veículo próximo da Manta parado à superfície da água. A realização deste exercício teve como propósito analisar os meios de transmissão escolhidos pelo algoritmo implementado durante um plano.

A injeção de mensagens foi feita pelo DuneSendMSg, tirando proveito das novas funcionalidade de teste, que permitiu enviar várias mensagens espaçadas por 30 segundos.

Este plano pretendia simular eventos de uma possível missão real, assim sendo, o veículo inicialmente deslocou-se à superfície, depois afundou até aos 2 metros de profundidade e continuou o seu trajeto. Após escassos minutos regressou à superfície estando, estimativamente, a

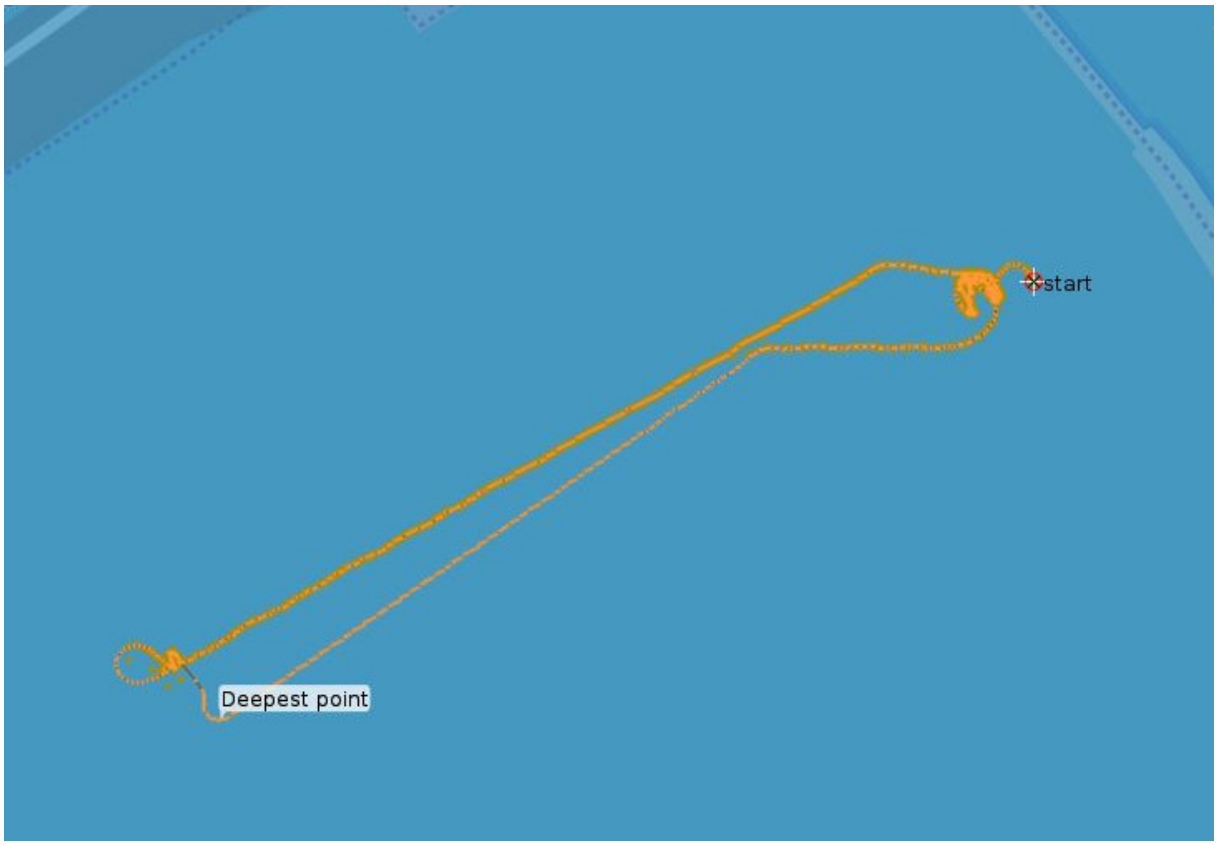


Figura 4.4: Cenário 2 - Trajeto

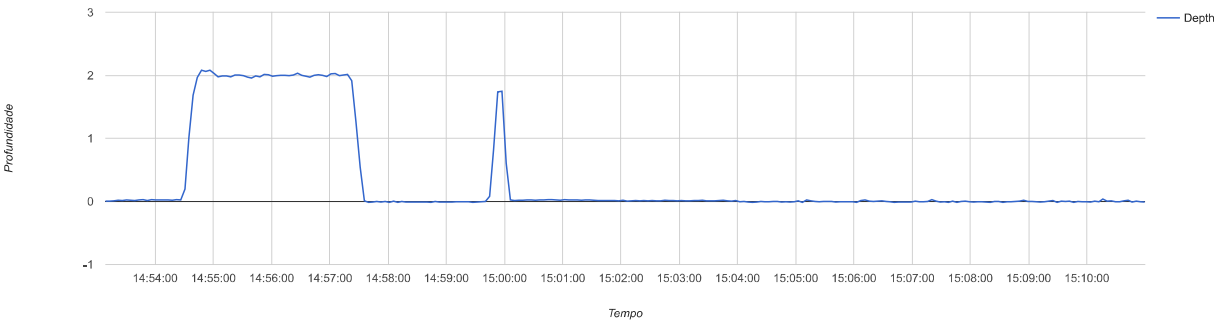


Figura 4.5: Cenário 2 - Profundidade

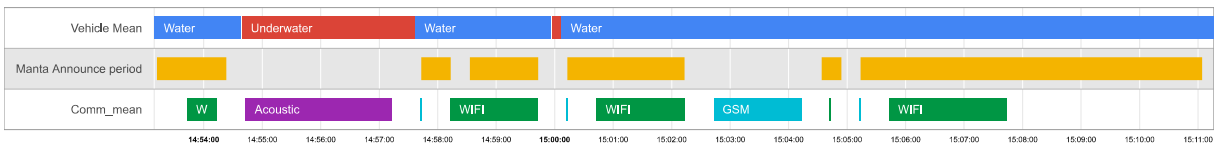


Figura 4.6: Cenário 2 - Mensagens

uma centena de metros do ponto inicial, onde ficou parado algum tempo. No percurso de volta, durante um período de tempo, foi retirada a alimentação do modem WiFi da Manta, com o intuito de simular grandes distâncias em alto mar, e assim fazendo o veículo perder a ligação por WiFi (e consequentemente TCP).

As deliberações da tarefa corresponderam exatamente ao esperado, ou seja, as mensagens foram enviadas pelos meios certos nos diferentes instantes. No entanto, os resultados mostram que houve 3 mensagens "soltas"a serem despachadas por SMS quando inicialmente era previsível que fossem por TCP. Após análise, podemos observar que nos instantes anteriores a cada uma dessas 3 mensagens não existe `Announce` e o veículo encontrava-se à superfície, confirmando que o algoritmo, de facto, escolheu o meio de transmissão corretamente.

É de notar que depois do veículo submergir, houve várias interrupções inesperadas dos anúncios de rede vindos da Manta. Estas falhas podem ser devido à distancia que o veículo se encontrava e/ou ao facto destas mensagens serem transmitidas por UDP, sendo que não existe garantia de entrega.

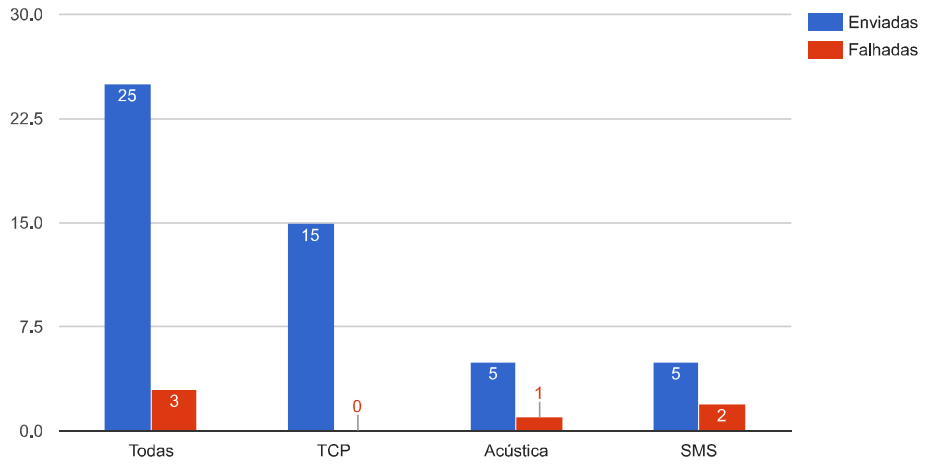


Figura 4.7: Cenário 2 - Estatísticas

Neste cenário, a comunicação por TCP teve um desempenho perfeito. As duas falhas de GSM foram, nomeadamente, as duas primeiras mensagens a tentarem serem transmitidas. Mais uma vez analisando a Figura 4.6 vemos que nos instantes anteriores a essas mensagens o veículo se encontrava debaixo de água. O algoritmo aplicado no `CommManager` verifica o sinal de ligação GSM antes de optar por este meio, pelo que podemos assumir que já havia cobertura de rede. No entanto, não é verificado se o módulo de hardware já se ligou a alguma antena e está pronto a enviar informação. Os resultados mostram que isto pode ser uma melhoria a considerar.

4.2.3 Cenário 3: Descarregar dados durante uma missão - `DataStore`

O 3 exercício foi uma demonstração da aplicabilidade do novo sistema de comunicação num cenário de missão. Muitas vezes, durante as missões do LSTS, são recolhidos dados (sensoriais) que habitualmente apenas são avaliados no final da operação. Uma maneira de fornecer mais

Figura 4.8: Cenário 3 - Trajeto

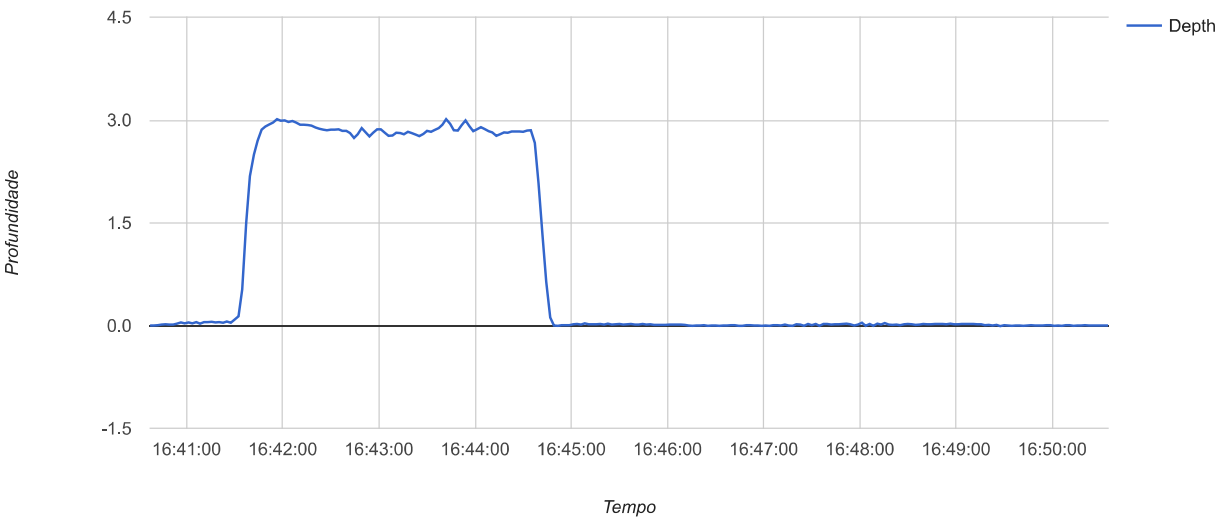


Figura 4.9: Cenário 3 - Profundidade

informação aos operadores em tempo real é dar acesso a estes dados recolhidos periodicamente.

O `DataStore` permite agrupar dados em pacotes para serem enviados. Com os desenvolvimentos consequentes deste trabalho, o `DataStore` foi modificado para transmitir a informação pelo `CommManager` em vez de criar e despachar as mensagens internamente. Desta forma, os dados recolhidos puderam ser transmitidos pelos vários meios de comunicação disponíveis.

Para a realização deste teste foi então configurado no veículo a tarefa do `DataStore` para criar mensagens periódicas. O plano de ações foi similar ao do cenário anterior. Podemos observar nos gráficos abaixo o percurso do veículo, Figura 4.8, e a profundidade durante todo o

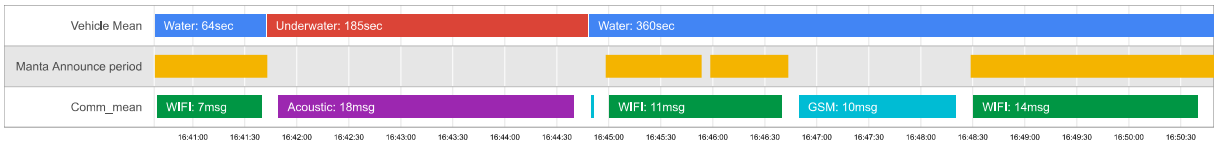


Figura 4.10: Cenário 3 - Mensagens

teste, Figura 4.9.

Para analisar o meio de transmissão escolhido podemos ter em conta dois fatores principais: o meio ambiente que se encontrava o veículo e os `Announce` vindos da Manta. Ao relacionar estes dois parâmetros temporalmente podemos prever o tipo de comunicação mais adequado a uma dada altura e comparar com o selecionado pelo CommManager. Na Figura 4.10 temos estas 3 informações compactadas para facilitar este processo.

Inicialmente o veículo estava à deriva na superfície da água e relativamente perto da Manta. Cerca de 1 minuto após o começo podemos ver o veículo a afundar até ao 3 metros de profundidade. Durante esses 3 minutos, as transmissões foram por via acústica, já que nenhum outro meio estaria disponível.

Um pouco antes das 16:45 o veículo começa a submergir. Ao chegar à superfície podemos ver que existe uma transmissão por SMS e não por WiFi. A visibilidade por WiFi para outros sistemas é relativa ao último `Announce` desse mesmo sistema, sendo que esta mensagem é transmitida periodicamente. Neste caso, quando o veículo chegou à superfície, o `DataStore` gerou uma mensagem para enviar antes do veículo receber o `Announce` da Manta, e assim sendo o veículo assumiu que ainda não existiria conexão por TCP. Nos momentos seguintes, essa mensagem foi recebida e as comunicações voltaram para WiFi como esperado.

Aproximadamente às 16:46:45 foi retirada a alimentação do módulo WiFi por 90 segundos, que corresponde ao tempo em que as mensagens foram despachadas por SMS.

De seguida voltou-se a ligar o modem à Manta e as comunicações por TCP ficaram novamente disponíveis.

O desempenho do `CommManager` a distribuir as mensagens pelos vários meios pode ser quantificado pelo gráfico da Figura 4.11. Houve um total de 61 tentativas de transmissão nos quais apenas 5 não tiveram sucesso, o que representa ±92% de eficácia. As falhas derivadas do TCP podem ser melhoradas ajustando a frequência dos `Announce` vindo dos outros sistemas, e expirando a sua validade mais cedo. Isto previne que o veículo tente enviar mensagens por TCP quando já não tem ligação.

4.2.4 Cenário 4: *Data muling*

Neste cenário pretendeu-se mostrar as capacidades do `CommManager` de gerir as retransmissões de mensagens não sucedidas, utilizando esta mais valia num cenário de *Data Muling* acústico.

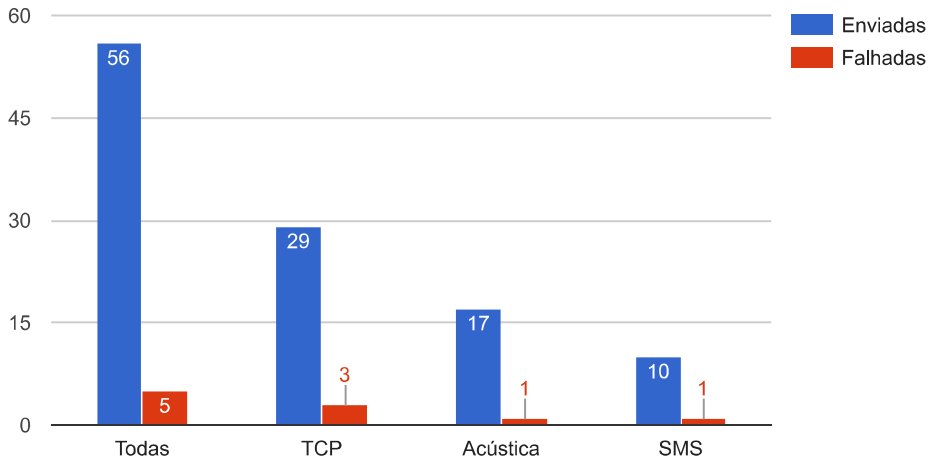


Figura 4.11: Cenário 3 - Estatísticas

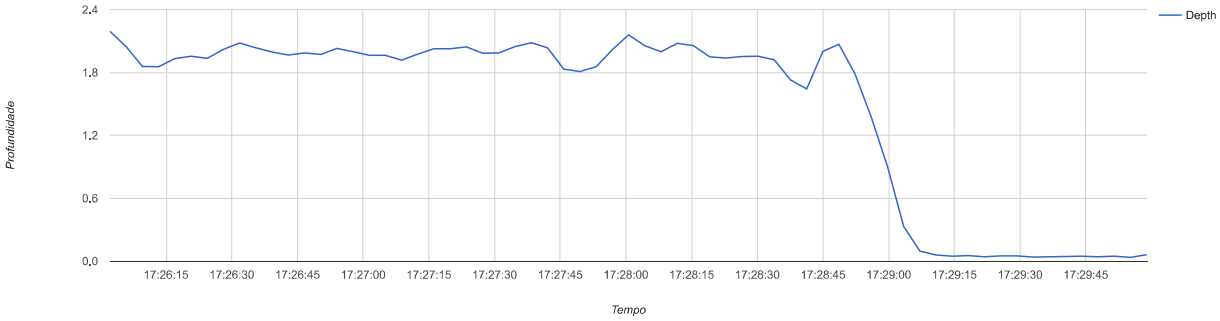
A Figura 4.12 é uma representação do cenário testado. Antes da realização do teste o veículo noptilus-2 foi enviado para o outro lado do pontão de operações, pontão esse que é feito em betão e por tal bloqueia a transmissão de som. Assim sendo, inicialmente os dois veículos encontravam-se afastados e sem comunicação direta. O veículo com as marcações a vermelho foi submergido à profundidade de 3 metros, mantendo-se na mesma posição por tempo indeterminado. (Na prática, e como se pode observar na Figura 4.12, este veículo não está parado mas sim aos círculos. Isto acontece porque os veículos são naturalmente flutuantes e precisam de se movimentar para manter uma posição debaixo de água. Em todo caso, iremo-nos referir a este veículo como se estivesse imóvel durante este subcapítulo).

Os procedimentos para o teste foram bastante simples. Foi injetada no veículo do lado Este do pontão uma mensagem acústica para abortar a execução de um plano, com validade suficiente para que este chegasse ao outro lado, tendo o veículo imóvel como destinatário.

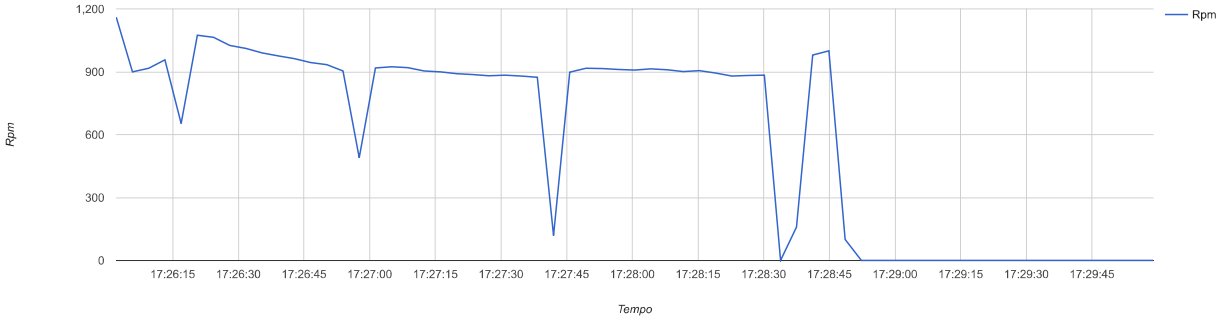
À medida que o veículo noptilus-2 se deslocava, tentava transmitir a mensagem acústica para o outro sistema. Não foi possível transmitir a mensagem com sucesso até os dois sistemas terem linha de visão. Na Figura 4.12 a marca representada por uma seta amarela identifica onde o veículo se encontrava às 17:28:47, altura no qual conseguiu transmitir a mensagem com sucesso. Tal como esperado, e verificável nos gráficos das Figuras 4.13a e 4.13b, o veículo noptilus-3 parou de realizar o seu plano (desligando os motores) e naturalmente submergiu.

Sem o mecanismo de gestão de mensagens não era prático enviar uma mensagem acústica oportunisticamente. Neste caso, tudo o que foi preciso foi introduzir uma mensagem com uma validade suficiente para que o veículo móvel se aproximasse do outro.

Figura 4.12: Cenário 4 - Trajetos



(a) Profundidade.



(b) RPM.

Figura 4.13: Cenário 4 - lauv-noptilus-3: Reacção ao pedido de “abort”

Capítulo 5

Conclusão

5.1 Apreciação

A uniformização das mensagens trouxe várias vantagens no panorama das comunicações. Todas as mensagens passaram a ter uma resposta com o estado de transmissão, dando mais *feedback* a quem pediu a transmissão. Todas as tarefas implementam um mecanismo de *buffering* juntamente com um controlo de validade, acrescentando robustez às comunicações. Para dar flexibilidade, tornou-se possível transmitir mensagens sem especificar o meio (comunicação via `ANY`) ou transmitir por todos os meios disponíveis (comunicação via `ALL`). As retransmissões automáticas vieram complementar as melhorias acima mencionadas, facilitando a realização de cenários como o de *Data Muling*.

Os resultados, em especial os do teste de campo em Leixões, demonstraram uma boa flexibilidade e fiabilidade, sem comprometer no desempenho ou na compatibilidade com versão do software mais antigas. A uniformização das mensagens para além de ter facilitado na transmissão de informação também solidificou as funcionalidades por todos os meios disponíveis. Além disso, os testes demonstram o potencial do roteamento adaptativo e *buffering* de mensagens, funcionalidades, alargando o leque de interação em rede por parte dos veículos.

Havia alguma relutância sobre o desempenho das comunicações por TCP no ambiente típico de operações com os veículos, devido ao atraso a estabelecer ligações e *overheads* em manter as mesmas. No entanto, as transmissões por TCP obtiveram muito bons resultados uma vez que, durante os testes, quase não tiveram falhas de transmissão (não expectáveis).

5.2 Trabalho Futuro

A nível de comunicações, o sistemas do LSTS são invulgarmente ricos em alternativas, o que faz abrir muito as possibilidades de melhorias. Ao longo do trabalho houve ideias que não foram desenvolvidas mas que se enquadravam no tema.

- **Algoritmo de escolha:** No sentido de melhorar o algoritmo de escolha, a idealização passa por considerar não só os parâmetros internos (do próprio sistema que quer transmitir), como também considerar os parâmetros do sistema de destino. Na prática isto significa disseminar alguma informação crucial por todos os sistemas, como por exemplo o meio que se encontra. Em exemplo, uma *base station* (Manta) saberia quando um veículo está debaixo de água, e facilmente decidiria enviar por esse meio.

  É de realçar que as comunicações neste tipo de rede de sistemas cooperativos são cada vez mais importantes tendo em vista a coordenação e desempenho dos mesmos. Apesar de todos os avanços, em termos de utilização das comunicações neste sentido ainda podemos dizer que está "verde". Esperamos que os desenvolvimentos apresentados nesta dissertação sejam um motivo para encorajar a propagação das comunicações entre sistemas.

- **Anúncio de meios de comunicação:** Uma desvantagem de usar mapas estáticos, como no caso de comunicações acústicas, é não ter qualquer informação se esse sistema está disponível ou não. Ou seja, se tentar transmitir para um sistema que esteja mapeado mas desligado, não existe maneira de saber, e por isso o `CommManager` vai tentar retransmitir até a validade expirar. Partindo da ideia implementada para a tarefa `TCPOnDemand` com as mensagens `Announce`, seria interessante os sistemas também se anunciarem periodicamente por via acústica. Consequentemente seria possível fazer uma verificação de visibilidade acústica antes deste meio ser selecionado para transmissão.

- **Atualização do Neptus:** Apesar do trabalho ter sido desenvolvido maioritariamente no DUNE, este componente da toolchain não funciona sem o resto dos componentes. Com isso em vista, é indiscutível a necessidade de atualizar o Neptus para reconhecer a nova API de comunicação e desenvolver uma ferramenta para criar mensagens `TransmissionRequest` e mostrar o seu estado (`TransmissionStatus`).

- **Melhorias no uso de GSM:** Uma das restrições que no futuro se pode colmatar, é o limite de tamanho ao enviar uma mensagem por GSM. Ainda que este protocolo imponha um limite permanente numa transmissão singular, é possível dividir um pacote em fragmentos e fazer múltiplas transmissões. Desta forma, o meio de comunicação GSM teria menos entraves a sua utilização.

5.3 Valorização pessoal

O trabalho desenvolvido na toolchain, e principalmente no DUNE, foi sem dúvida envolvente. Houve um grande estudo do software e de conceitos até que fosse possível começar a implementação. O trabalho realizado pelo LSTS é realmente multidisciplinar, e isso foi perfeitamente observável durante este percurso de realização da dissertação. A experiência de trabalhar e fazer parte do laboratório superou, sem qualquer dúvida, as expectativas, não só em aprendizagem como no desenvolvimento a nível pessoal.

Para além do trabalho feito em laboratório, foi muito enriquecedor a participação em várias idas ao porto de Leixões no âmbito de testes. A experiência de operar veículos reais a executar código parcialmente desenvolvido e descrito nesta tese, foi recompensadora, tal como saber que o trabalho implementado pode ser realmente usado, no futuro, por vários sistemas autónomos.

Bibliografia

[1] R. Martins, "Disruption/delay tolerant networking with low-bandwidth underwater acoustic modems", *2010 IEEE/OES Autonomous Underwater Vehicles*, 2010.

[2] K. Fall, "A delay-tolerant network architecture for challenged internets", *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, 2003.

[3] A. Rumpold, "Transmission Protocols for Delay-Tolerant Networks", *Chair for Network Architectures and Services, Department of Computer Science, Technische Universität München*, 2011.

[4] J. Pinto, M. Ribeiro, P. Gonçalves and J. Sousa, "Web enabling ocean vehicles under intermittent communications", *OCEANS 2016 MTS/IEEE Monterey*, 2016.

[5] Y. Bayrakdar, J. Pinto, J. Pereira and J. Teixeira, "Routing protocol for AUV communications: Report from field tests", *OCEANS 2017 - Aberdeen*, 2017.

[6] R. Martins, P. Dias, E. Marques, J. Pinto, J. Sousa and F. Pereira, "IMC: A communication protocol for networked vehicles and sensors", *OCEANS 2009-EUROPE*, 2009.

[7] A. Ferreira, J. Pinto, P. Dias and J. Borges de Sousa, "The LSTS software toolchain for persistent maritime operations applied through vehicular ad-hoc networks", *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017.

[8] R. Martins, J. Borges de Sousa, R. Caldas, C. Petrioli and J. Potter, "SUNRISE project: Porto university testbed", *2014 Underwater Communications and Networking (UComms)*, 2014.

[9] J. Pinto, P. Dias, R. Martins, J. Fortuna, E. Marques and J. Sousa, "The LSTS toolchain for networked vehicle systems", *2013 MTS/IEEE OCEANS - Bergen*, 2013.

[10] A. Zolich, D. Palma, K. Kansanen, K. Fjørtoft, J. Sousa, K. Johansson, Y. Jiang, H. Dong and T. Johansen, "Survey on Communication and Networks for Autonomous Marine Systems", *Journal of Intelligent & Robotic Systems*, 2018.