

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Wireless Underwater Broadband and Long Range Communications using Underwater Drones as Data Mules

Leonel Gaspar da Costa Soares

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Manuel Alberto Pereira Ricardo (PhD)

Second Supervisor: Filipe Borges Teixeira (MSc)

July 28, 2017

Resumo

A utilização de veículos subaquáticos autónomos (AUVs) exige transmissão de grandes quantidades de informação entre estes equipamentos e as respetivas estações de monitorização. As comunicações subaquáticas estão associadas à transmissão de sinais acústicos, óticos e de radiofrequência (RF). Para implementações com utilização de modems acústicos, é possível fornecer comunicações a longo alcance (alguns km), mas com baixo débito de transferência de dados, tornando o processo moroso; por outro lado, as comunicações óticas exigem um alinhamento rigoroso do emissor e receptor para a transmissão de informação. As comunicações RF, por outro lado, estão associadas a elevados débitos, mas apenas a curto alcance (dezenas de centímetros até alguns metros), não existindo, por isso uma solução capaz de fornecer um acesso de banda larga para grandes distâncias debaixo de água.

Nesta tese propõe-se aumentar o débito e o alcance das comunicações sem fios subaquáticas aproveitando os altos débitos associados às transmissões RF e expandindo o seu alcance através do uso de pequenos drones subaquáticos - *data mules* - capazes de transportar informação entre dois nós, criando uma ligação entre ambos. Para tal, serão aplicados protocolos de comunicações adequados para redes com restrições de conectividade, denominadas Redes Tolerantes a Atrasos (*Delay Tolerant Networks* - DTN).

As aplicações da plataforma DTN escolhida foram testadas em situações experimentais onde a transferência de dados entre dois elementos ocorre com movimento de uma *data mule* (veículo capaz de transportar informação entre dois nós para criar um link entre ambos). As estimativas iniciais para débito resultante deste processo foram na ordem de 3 Mbit/s para uma distância de 20 m, um valor 96 vezes superior ao valor máximo de débito do conjunto de modems acústicos disponíveis comercialmente (31.2 kbit/s). As aplicações foram avaliadas e, tendo em conta as limitações encontradas, introduziram-se alterações no código fonte de forma a obter sincronização bidireccional de ficheiros. Em seguida, foram planeados testes em meio subaquático e avaliadas as alterações na aplicação, foi possível obter sincronismo de ficheiros unidireccional e bidireccional. Desta forma, prova-se que a utilização de *data mules* em alternativa aos modems acústicos permite aumentar o débito, uma vez que foram obtidos experimentalmente valores da ordem de 1.56 Mbit/s e 519 kbit/s, 50 e 16 vezes superior ao débito acústico máximo esperado. A extrapolação desses valores para comunicações de elevado alcance (1000 m) é demonstrada graficamente e também apresenta ganhos consideráveis em relação às comunicações acústicas.

Abstract

The use of autonomous underwater vehicles (AUVs) requires transmission of large amounts of information between these devices and their monitoring stations. Underwater communications are associated with the transmission of acoustic, optical and radiofrequency (RF) signals. For deployments using acoustic modems, it is possible to provide long-range (some km) communications, but with low data throughput, making the process time-consuming; on the other hand, optical communications require a strict alignment of the sender and receiver for the transmission of information. RF communications, on the other hand, are associated with high speeds, but only at short range (from tens of centimeters up to a few meters), so there is no solution capable of providing broadband access for long distances underwater.

In this thesis we proposed to increase the throughput and range of underwater wireless communications by taking advantage of the high rates associated with RF transmissions and expanding their range through the use of small underwater drones, named data mules, which are vehicles that are able to carry information between two nodes, creating a link between them. To this end, appropriate communications protocols are applied to networks with connectivity constraints, called Delay Tolerant Networks (DTN).

The applications of the DTN platform chosen were tested in experimental setup where the transfer of data between two elements occurs with movement of a data mule. The initial estimation for throughput of this process was in the order of 3 Mbit/s for distance of 20 m, a value 96 times higher than the maximum value of the set of commercially available (31.2 kbit/s) acoustic modems. The applications were evaluated taking into account the limitations found, and changes were introduced in the source code in order to obtain bidirectional synchronization of files. These changes were evaluated in underwater environment, proving that it was possible to obtain unidirectional and bidirectional file synchronism.

Thus, it is proven that the use of data mules as an alternative to acoustic modems increases the throughput, since values in the order of 1.56 Mbit/s and 519 kbit/s, 50 and 16 times higher to the maximum acoustic throughput using an acoustic modem available on the market. Extrapolation of these values to high distance communications (1000 m) are graphically demonstrated and also present considerable gains relative to acoustic communications.

Agradecimentos

Gostaria de agradecer ao Professor Manuel Ricardo, por me aceitar gentilmente no INESC TEC CTM e orientar a minha tese. Agradeço também ao Eng. Filipe Borges Teixeira pelo aconselhamento e orientação prática, ao Doutor Rui Campos por suas sugestões e motivação, e todo o INESC TEC CTM pela ajuda especialmente Carlos Leocádio, Pedro Silva, Eduardo Almeida, Mário Lopes, André Coelho, Tiago Oliveira, Hélder Fontes e Filipe Ribeiro.

Expresso também a minha gratidão ao Professor Nuno Cruz e ao Professor Aníbal Matos por me permitirem realizar experiências na Unidade de Robótica. Finalmente, agradeço aos meus pais, irmã e sobrinho por todo o apoio.

Leonel Soares

“Water sustains all”

Thales of Miletus

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Objectives	2
1.4	Contributions	2
1.5	Structure	3
2	State-of-the-Art	5
2.1	Underwater wireless communications	5
2.1.1	Acoustic wireless transmission	5
2.1.2	Optical wireless transmission	7
2.1.3	Radiofrequency wireless transmission	8
2.2	Delay Tolerant Networks	11
2.2.1	Application scenarios	11
2.2.2	Architecture	11
2.2.3	Message sequence diagrams of bundle protocol	16
2.2.4	Bundle Protocol Implementations	17
2.2.5	Applications over Bundle Protocols implementations	19
3	IBR-DTN applications	21
3.1	Hardware specifications	21
3.2	Software specifications	23
3.3	IBR-DTN configuration and API	23
3.4	dtnd	24
3.5	dtnping	24
3.6	dtntrecv	25
3.7	dtntsend	27
3.8	dtntinbox and dtntoutbox	31
3.9	dtntstream	36
3.10	dtnttracepath	37
3.11	dtntconvert	38
3.12	dtnttrigger	38
3.13	dtntunnel	38
3.14	Conclusions	38
4	Synchronization application	41

5	Experimental planning and testbed design	47
5.1	Experimental evaluation	47
5.2	Study of movement conditions for favorable binary rate transfer	49
5.2.1	Metrics to be considered in the evaluation	52
5.2.2	Analysis of results	52
6	Preformance results	53
6.1	Single-hop performance comparison of dtnsend and iperf	55
6.2	Store and forward with the use of data mule in underwater environment	56
6.3	Unidirectional synchronization with proposed application	58
6.4	Bidirectional synchronization of folders with proposed application	60
6.5	Update of files in remote host when file is changed in local host	62
7	Conclusions and future work	63
A	Code used for file synchronization	67
	References	101

List of Figures

2.1	Examples of acoustic modems	6
2.2	Attenuation of sound underwater [1]	7
2.3	Examples of optical modems	8
2.4	Attenuation of RF underwater [2]	10
2.5	Architecture of bundle protocol [3]	13
2.6	Store and forward [3]	14
2.7	DTN graph abstraction [4]	15
2.8	Store and forward mechanism [4]	16
2.9	Message sequence diagram between two nodes, with a data mule	17
3.1	Alix3d3 board [5]	22
3.2	Mikrotik routerboard r52n-M [6]	22
3.3	Example of the IBR-DTN API use	24
3.4	Output of dtnd invocation	24
3.5	Example of the dtnping use	25
3.6	Testbed for simulation of datamule between sender and receiver nodes	26
3.7	Example of the dtnrecv usage	26
3.8	Example of the dtnsend use	27
3.9	Movement of data mule (node B) simulated between nodes A and C in test 1	28
3.10	Number of bundles generated by fragmentation vs. file size	30
3.11	Wireshark capture in transferring file 4.9 Mbytes with dtnsend	30
3.12	Wireshark capture in transferring file 49.7 Mbytes between nodes A and B	30
3.13	Wireshark capture in transferring file 49.7 Mbytes between nodes B and C	30
3.14	Example of the dtnoutbox use	31
3.15	Example of the dtninbox use	31
3.16	Procedure applied in testing one way folder transfer using dtninbox and dtnoutbox simultaneously	33
3.17	Results obtained in wireshark reading one way folder transfer using dtninbox and dtnoutbox simultaneously (packets blue have IP source 10.0.0.1 and packets red have IP source 10.0.0.2)	33
3.18	Procedure applied in testing bidirectional file transfer with dtninbox and dtnoutbox	34
3.19	Results obtained in wireshark reading with bidirectional file transfer experiment with dtninbox and dtnoutbox (packets blue have IP source 10.0.0.1 and packets red have IP source 10.0.0.2)	35
3.20	Example of the dtnstream use - receiver	36
3.21	Example of the dtnstream use - sender	36
3.22	Whireshark capture of streaming between nodes A and B	37
3.23	Wireshark capture of streaming between node B and C	37

3.24	Example of the dtnttracepath use	38
4.1	Algorithm used in file sharing application	42
4.2	Sequence diagram file synchronization application	44
4.3	Interaction between application and API	45
5.1	Experimental scenario considered for the application test in underwater environment	47
5.2	Simulation of useful bitrate and minimal docking time variations with distance .	48
5.3	Total delay versus distance	49
5.4	Useful bitrates vs distance (path velocity 1 m/s)	50
5.5	Useful bitrates vs velocity (path distance 1000 m)	51
5.6	Efficiency for Go-back-N estimated for different values of FER	52
6.1	Cylinder used in experiments	53
6.2	Testbed for data mule experiments between sender and receiver nodes	54
6.3	Testbed for data mule experiments between sender and receiver nodes in INESC TEC	54
6.4	Bit rate extrapolation for different distances with store and forward experimenta- tion results	57
6.5	Bit rate extrapolation for different distances with unidirectional synchronization results	60
6.6	Bit rate extrapolation for different distances with unidirectional synchronization results	61

List of Tables

2.1	Characteristics of different acoustic modems	7
2.2	Characteristics of different optical modems	8
2.3	Attenuation values of RF in water for different frequencies	10
2.4	Implementations for DTN	18
3.1	Caracteristics of ALIX3D3 Boards	21
3.2	RouterBOARD r52n-M Specifications	22
3.3	IP configuration of Ethernet and Wi-Fi addresses used in the tests	25
3.4	Number of fragments obtained in each file	29
3.5	Time between start of node detection and start of bundle transmission	29
3.6	Average time between start of node detection and start of bundle transmission	29
3.7	Average time of final delivery of bundles from node B to node C, compared with time to copy files from A to C	31
6.1	Time required for transfer of bundles between nodes A and B	55
6.2	Time required for transfer of bundles between nodes A and B	56
6.3	Time required for transfer of 49.7 Mbytes file between nodes A and B	56
6.4	Time required for transfer of 49.7 Mbytes file between nodes B and C	57
6.5	Time required for transfer of bundles between nodes A and B	59
6.6	Time required for transfer of bundles between nodes B and C	59
6.7	Time required for both folders present the same content	60

Abbreviations

AMS	Asynchronous Message Service
AUV	Autonomous Underwater Vehicles
CCDS	Consultative Committee for Space Data Systems
CGR	Contact Graph Routing
CFDP	CCSDS File Delivery Protocol
CLA	Convergence Layer Adapter
DTN	Delay Tolerant Networks
DTN2	Delay Tolerant Networks version 2
DTN-RI	Delay Tolerant Networks Reference Implementation
DTNRG	Delay Tolerant Networks Research Group
DTLSR	Delay Tolerant Link State Routing
ECL	External Convergence Layer
FER	Frame Error Ratio
ION	Interplanetary Overlay Network
JPL	Jet Propulsion Laboratory
LPT	Licklider Transmission Protocol
NTP	Network Time Protocol
PoE	Power over Ethernet
RF	Radio frequency
ROV	Remotely Operated Vehicles
RTEMS	Real-Time Executive for Multiprocessor Systems
TCP	Transmission Control Protocol
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Context

The need of monitoring aqueous environments and necessity of reliable communications between or with underwater vehicles, together with growing use of Autonomous Underwater Vehicles (AUV) in underwater surveillance and inspection missions [7] has brought interest in extensive research on underwater wireless communications [8]. Acoustic communications are used for long range, in order of some kilometers, but allows only low bitrates [8], moreover, modems applied in these systems have a high energy consumption, which limits the AUV endurance. On the other hand, optical signals demand precise alignment of the emitter and receiver and are affected by water turbidity [8]. Also, any obstacle to optical path between sender and receiver will compromise the transmission. Radio frequency, however, has brought high bitrate transmissions, but only in small ranges, typically some centimeters to some meters [2].

Set of specifications for wireless local area networks implied in norm IEEE 802.11 standard, already used for aerial transmissions can also be used for cost effective, high bandwidth, short range communications. The vehicles operating in underwater missions are now equipped with strong precision sensors, designed for extensive gathering of water's properties (such as pH, temperature, pollutant concentration) and are equipped with good resolution cameras for high quality video recording, which produce high amount of data during a mission to be transferred to shore. Underwater communications are also strongly prone to high bit error, long delays and intermittent connectivity, making conversational protocols, such as Transmission Control Protocol (TCP), unable to effectively work in data transmission [3]: for these situations it may be more suitable to apply protocols designed for Delay Tolerant Networks (DTN), which are implemented for transmission scenarios with these characteristics.

1.2 Motivation

Autonomous Underwater vehicles demand for transmission of large amounts of data, at long ranges. Acoustic transmissions can be used in long distance but only provides small data rates,

causing the process of files reception to take too much time.

Radiofrequency underwater communications have shown possibility of transmitting information with fast data rates, such as 100 Mbit/s, but only in close range transfers [9]. In order to extend the range of these communications, where no end-to-end communication path exists between eventual sender and receiver, the use of data mules -workers or vehicles scheduled to pass through the monitored area periodically and carry the information from sender to receiver - was tested, in order to bridge the gaps between the nodes [10].

Recently, DTN have gain considerable attention as an information gathering technology in challenging environments, like underwater medium, where difficulties such as continuous delay, disruption and disconnect may frequently occur [11]. The Bundle Protocol function in DTN is to store messages in the bundle layer of the protocol while connection is unavailable. When the connection is reestablished, DTN can restart forwarding the stored messages again to destination nodes without assuming end-to-end connectivity to them [11]. Implementation of DTN applications in underwater vehicles may allow more effective data transmission in many sorts of operations, since the environment in which they move is in line with DTN platforms purpose.

Considering this set of advantages, the proposed solution improves communications between AUVs and the docking stations where the data is uploaded to shore, allowing wide range underwater transmission of large data, and thus can represent an alternative to acoustic technologies.

1.3 Objectives

This thesis aims to study and evaluate the use of data mules capable of transporting data across networks with tolerance to delay (DTN) between a transmitter and an underwater receiver, taking advantage of high transfer rates at close range. Experiment procedure was carried with implementation of a file synchronization application that can tolerate high delays in the delivery of information. The application will be tested in the underwater environment using the large tank (dimensions of 6 meters long, 5 meters wide and 2 meters depth) available at INESC TEC and airtight cylinders, and compared with theoretical results for this scenario.

1.4 Contributions

The main contributions of this thesis are the following:

Extending range in broadband underwater communication with the use of data mules

Data mule movement was applied in this dissertation for transferring data between distant elements in underwater medium, in order to extend the range of RF communications. File transfer was quantified and bit rates estimated for different experimental situations, and later compared with acoustic results in similar conditions. Bit rates obtained were 3 Mbit/s, 1 Mbit/s and 500 kbit/s, 96, 32 and 16 times greater than maximum value of acoustic modems presented in this thesis (31.2 kbit/s). Also, extrapolation was made in order to estimate communication throughput at different

distances, and for 5000 m, bit rate estimated was still superior to maximum value of acoustic modem presented in this thesis.

Adapting an IBR-DTN application for bidirectional synchronization of files Underwater communications are associated with loss of connectivity, which can be compensated with DTN software. After installing IBR-DTN for this effect, some applications were tested and new code was created for synchronization of files. The application was tested for the proposed effect and results compared with present solutions.

1.5 Structure

The present document includes seven chapters. Chapter 2 explains underwater communications problems and technologies implemented in these processes. Also, this chapter presents a description of some concepts associated with delay tolerant networks, as well as the protocols included. Chapter 3 presents applications used in this thesis, with results of tests applied and in chapter 4, we explain development of a file synchronization application.

Chapter 5 presents experimental scenario and problem considered in this thesis with planning of proposed solution. Chapter 6 includes performance results obtained in planned testbed and finally chapter 7 indicates summary of conclusions and future work regarding the objectives of this thesis.

Chapter 2

State-of-the-Art

In spite of the interest in reliable underwater communication [8], the aquatic medium consists in regions where mobile devices find a diverse number of restrictions [12] in terms of technology and network (topology and protocols). Signal propagation may not be totally effective in transmission between sender and receiver, due to the signal (mechanical or electromagnetic waves) nature, which determines the propagation together with water's physical characteristics. Also, communicating links between nodes can be obstructed underwater by medium turbulence, and devices may have considerable power limitations.

2.1 Underwater wireless communications

The main transmission technologies implied within underwater communications are based on acoustic, optical and RF signal transmission. Acoustic communications are the most used in this context [13] [2] [14]. In the following sections, these technologies will be briefly described.

2.1.1 Acoustic wireless transmission

Sound waves propagate in fluids, like air and water, as a disturbance of pressure level. The speed of sound in water is approximately 4 to 5 times the value seen in air (about 1500 m/s on water relative to 340 m/s in air). This propagation speed is however dependent on water's conditions of temperature, pressure and salinity, which also can influence the direction of motion of sound waves or make it propagate through longer distances [15] [16].

Acoustic transmission is best supported for lower frequencies, and bandwidth available for communication is extremely limited, causing low data rates. These systems can operate in frequencies ranges between 10 and 15 kHz [17], which results in a total bandwidth of 5 kHz. The bandwidth is not negligible with respect to central frequency, which makes the system ultra-wideband [15].

Sound propagation in water is favorable to introduce big sensitivity in these systems to acoustic "noise" from other sources [15]. Some other limits can be found in shallow water, like reflection and attenuation, poor performance in shadow water and sensitivity to environmental characteristics



(a) LinkQuest UWM1000 [20]



(b) EvoLogics S2CR 48/78 [21]

Figure 2.1: Examples of acoustic modems

[15]. Also, sound waves can be affected by multi-path, Doppler effect [18] [19] and spreading loss. Real-time response synchronization and multiple access protocols are affected by high latency for long range, which is associated to low speed propagation.

Figures 2.1a and 2.1b show two different examples of acoustic modems available on the market.

Pressure associated with a spherical wave propagating in water can be expressed by[22]:

$$P(R,t) = \frac{p_0}{R} \exp(-\gamma R) \exp(j\omega(t - \frac{R}{c})) \text{ (Pa)} \quad (2.1)$$

Here, attenuation is expressed as a parameter γ (expressed in Np/m). The exponential decreases of pressure gives a loss expressed in dB proportional to the propagation range: this is commonly expressed by an attenuation coefficient α , which is quantified in decibels per meter, (dB/m), and can be related with γ by the following expression :

$$\alpha = 20 \gamma \log e \quad (2.2)$$

or, by approximation:

$$\alpha = 8.686 \gamma \quad (2.3)$$

Figure 2.2 shows attenuation dependency with frequency in saltwater and freshwater. Graphic shows that attenuation grows with frequency.

Some features associated with commercial acoustic modems [15] can be found in table 2.1.

This table show that the fastest acoustic modem can only transmit at 31.2 kbit/s, with a 1000 m range. It can also be seen that the modem with highest range (10 km) has a data bitrate of 5 kbits/s, which is not suitable for transmission of extensive files that require faster data exchange, like video or audio files [15]. Also, it must be refereed that these instruments also present considerable weight, ranging from 4.1 to 21 kg, which reduce the endurance of the AUVs.

Acoustic modems can have interest when used in particular scenarios, namely in transmission of basic control signals over extended distances (> 1 km), although wireless transmission is not effective with this equipment if high data rates are required, even at short distances [15].

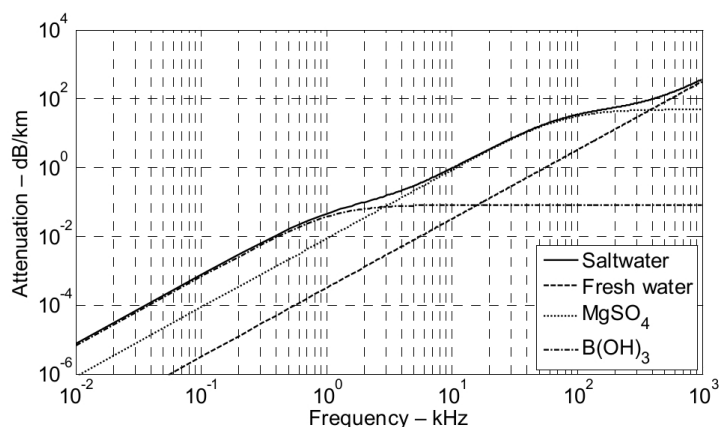


Figure 2.2: Attenuation of sound underwater [1]

Table 2.1: Characteristics of different acoustic modems

Model	Distance (m)	Rate kbit/s	Operating frequency (kHz)	Power (W)	Maximum depth (m)	Weight (kg)
LinkQuest UWM1000	350	9.6 to 19.2	26.77 to 44.62	2	200	4.2
LinkQuest UWM2000	1500	9.6 to 19.2	26.77 to 44.62	8	4000	4.8
LinkQuest UWM3000	5000	2.5 to 5	7.5 to 12.5	12	7000	4.1
LinkQuest UWM4000	4000	4.8 to 9.6	12.75 to 21.25	7	7000	7.6
LinkQuest UWM10000	10000	2.5 to 5	7.5 to 12.5	40	7000	21
EvoLogics S2CR 48/78	1000	31.2	48 to 78	60	2000	2.25 to 6.5
EvoLogics S2CR 42/65	100	31.2	42 to 65	60	2000	2.3 to 6.5
EvoLogics S2CR 18/34	3500	13.9	18 to 34	80	2000	2.4 to 6.5
EvoLogics S2CR 7/17	8000	6.9	7 to 17	80	6000	4.7 to 7.78

2.1.2 Optical wireless transmission

Optical communications use light to transmit information. Light also consists in the oscillation of an electromagnetic field, so it also presents similar characteristics to RF waves, but with higher frequencies [15].

Visible light frequencies show the lowest attenuation in the electromagnetic spectrum. Wavelengths close to 470 nm are generally attenuated the least [15], and this value depends on water's chemical and biological compositions, since absorption and scattering are determined by this conditioning.

The relatively low attenuation combined with high speed of electromagnetic waves makes possible to obtain high data rate transmissions in underwater wireless communications with optical systems, however operations are strongly depend on line-of-sight, which may not be possible in underwater environment [15].

Recent improvements in LED technology have enabled the development of power efficient optical transmitters at low cost that offer fast switching speeds, high level of light intensity, high

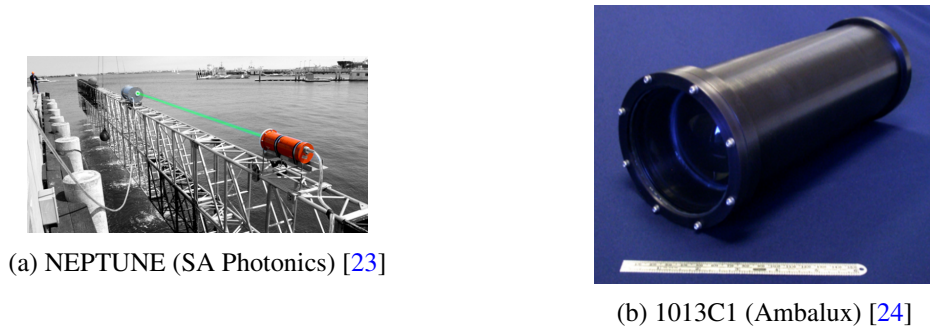


Figure 2.3: Examples of optical modems

efficiency, and favorable wavelengths for underwater transmission. Quality of transmission can be improved even further with the use of lasers since they provide a much better collimated light beam. However it is not always considered, since lasers are very susceptible to misalignment [15]. Some characteristics of optical modems can be seen in table 2.2.

Optical systems may provide a better solution with higher bitrate transfer when compared to acoustic modems in underwater communications, but range is significantly smaller. Some systems may even achieve data rates in order of Mbit/s. Line-of-sight requirement and considerable loss of performance due to scattering however, make these systems impossible to apply in some cases [15].

Table 2.2: Characteristics of different optical modems

Model	Distance (m)	Rate kbit/s	Operating frequency (kHz)	Depth (m)
AQUAmodem 500	250	25 to 100 bits/s	27 to 31 kHz	200
AQUAmodem Op1	1	19.2 kbits/s	610 to 575 THz	3000
NEPTUNE (SA Photonics)	10-200	10 - 250 Mbit/s	532 nm / 486 nm	N/A
1013C1 (Ambalux)	up to 40	10 Mbit/s	N/A	60

Figures 2.3a and 2.3b show two different examples of optical modems. AQUAmodem Op1 still presents considerable weight, reaching 4.2 kg.

2.1.3 Radiofrequency wireless transmission

The limitations of acoustic and optical systems can be overcome with Radio-Frequency implementations. RF has the advantage of not being affected by turbidity, and can operate in non-line-of-sight, allowing high bandwidth to be obtained at close range without the effect of acoustic noise. However, RF wave propagation suffers strong attenuation underwater, especially if the 2.4 GHz ISM band is used. Both theoretical and experimental studies of IEEE 802.11 underwater networks based on RF only achieve ranges in the order of a few centimeters, bringing up the need to employ lower frequencies to obtain higher ranges. Experimental results have shown also that by using

frequencies under GHz, RF attenuation can be progressively reduced, with ranges achieving 5 m and throughput exceeding 550 kbit/s in freshwater [9].

Propagation of RF waves in water is slower than what occurs in the air, and generally attenuation of the signal will be stronger, due to dissolved salts and other matter. The propagation of RF waves can be characterized by a propagation constant, γ , given by [13]:

$$\gamma = \sqrt{j\omega\mu(\sigma + j\omega\epsilon)} = \alpha + j\beta \quad (2.4)$$

where σ is the conductivity of the medium expressed S/m, $\mu = \mu_r \cdot \mu_0$ is the permeability of the medium in N/A and $\epsilon = \epsilon_r \cdot \epsilon_0$ is the permittivity of the medium in F/m.

The propagation constant is a complex value and can be expressed by an attenuation factor α , and a phase factor, β [2]:

$$\alpha = \omega\sqrt{\mu\epsilon} \left[\frac{1}{2} \left(\sqrt{1 + \left(\frac{\sigma}{\omega\epsilon} \right)^2} - 1 \right) \right]^{\frac{1}{2}} \quad (Np/m) \quad (2.5)$$

$$\beta = \omega\sqrt{\mu\epsilon} \left[\frac{1}{2} \left(\sqrt{1 + \left(\frac{\sigma}{\omega\epsilon} \right)^2} + 1 \right) \right]^{\frac{1}{2}} \quad (rad/m) \quad (2.6)$$

The real part of the permittivity ϵ is dependent of the complex frequency, and is commonly described with the Debye model [2]:

$$\epsilon = \epsilon_\infty + \left[\frac{\epsilon_s - \epsilon_\infty}{1 + j \left(\frac{f}{f_{ref}} \right)} \right] \quad (F/m) \quad (2.7)$$

Where ϵ_s and ϵ_∞ are the real relative permittivity at low and high frequencies, respectively in F/m, f_{ref} is the relaxation frequency in Hz and ϵ_0 is the dielectric permittivity of the free space in F/m.

The wavelength λ is calculated according to and the velocity is determined by [2]:

$$\lambda = \frac{2\pi}{\beta} \quad (m) \quad (2.8)$$

$$v = \frac{\omega}{\beta} \quad (m/s) \quad (2.9)$$

Received power can be calculated by:

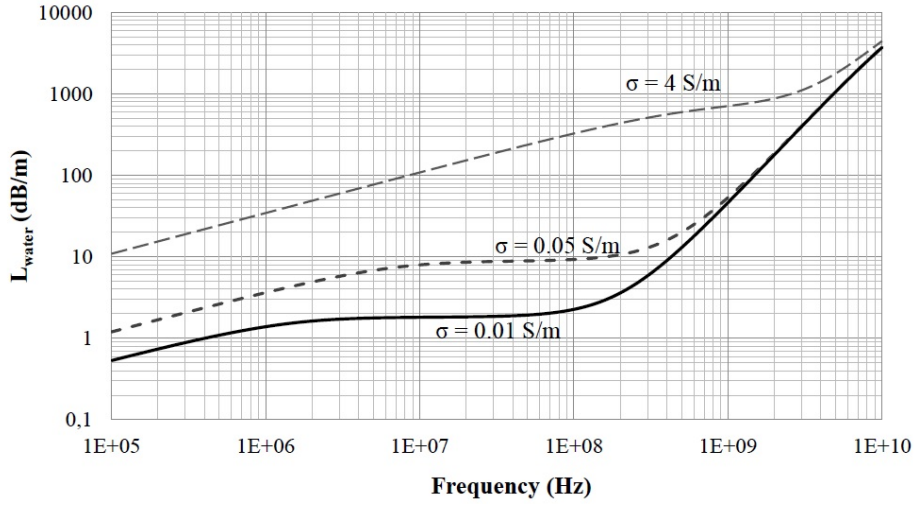


Figure 2.4: Attenuation of RF underwater [2]

$$P_{rx}(dBm) = P_{tx} + G_{tx} + G_{rx} - L_{FSPL} - L_{water} \quad (2.10)$$

where P_{tx} represents the transmission power in dBm , G_{tx} and G_{rx} are, respectively, the gains of the transmitter and receiver antennas in dB , L_{FSPL} is the free space path loss, and L_{water} is the attenuation of the RF waves in the water, which depends on the conductivity and operating frequency of the water.

Figure 2.4 shows attenuation of RF waves for different values of conductivity. It can be easily concluded that attenuation is strongly dependent of conductivity and increases with frequency. The plot also shows that, for freshwater, with typical conductivity values of $\sigma = 0.01$ S/m, the attenuation L_{water} is 269 dB/m for 2.4 GHz and 28 dB/m for 700 MHz; in sea water, presenting typical conductivity values of $\sigma = 4$ S/m, the attenuation L_{water} is much higher, raising to 1000 dB/m for 2.4 GHz and 700 dB/m at 700 MHz. The conductivity limit for freshwater is $\sigma = 0.05$ S/m and represents the worst case scenario for freshwater [2]. Frequencies between 10 and 100 MHz can extend range in RF underwater communications, allowing relative low attenuation and sufficient bandwidth for broadband communication.

Table 2.3: Attenuation values of RF in water for different frequencies

Frequency	Propagation velocity (m/s)	Attenuation (dB/m)
768 MHz	3.33×10^7	28
2.462 GHz	3.35×10^7	269
5.240 GHz	3.43×10^7	1161

The attenuation for different frequencies [25] can be expressed in the table 2.3. These values show that propagation speed does not suffer strong variations with frequency, in spite of the lower frequencies showing less attenuation. In RF it is possible to obtain bitrates as high as 100 Mbit/s

[9] in underwater communications, which makes this technology interesting for high bandwidth applications.

2.2 Delay Tolerant Networks

The growing use of AUVs in underwater surveillance and inspection missions has also brought interest in underwater wireless communications [26]. However, the challenge associated with introduction of internet communicating services in areas with strong connectivity limitations, such as the sub-aquatic medium (high delay in data transfer, high bit error rate and no guarantee of end-to-end path between nodes) has captured interest from developers and general scientific community in research of technologies to overcome these setbacks.

2.2.1 Application scenarios

The first use of Delay Tolerant Networks (RFC 4838 [27]) was applied in Space programs, but soon applications of these systems became extendable to terrestrial communications. Some present applications of these networks can be found in [3]:

- Space Agencies - interplanetary communication and space monitoring.
- Military and intelligence organizations - mobile ad-hoc networks (MANET) for wireless communication and monitoring, search and rescue communications, cargo tracking, UAV communications and control.
- Commercial use - preform cargo and vehicle tracking, in-store and in-warehouse asset tracking, culture monitoring.
- Public service and safety - security and disaster communication, search and rescue communications, smart electric power networks.
- Personal use - communicating in wilderness and urban areas, and environmental monitoring.
- Engineering and scientific research - to develop academic networks projects.

2.2.2 Architecture

Traditional data networks are modeled by a connected graph [4], assuming at least one end-to-end connection exists between any eventual communicating pair of nodes. Links between the network elements are treated as bidirectional, providing symmetric data flow with low delay and bit error rate. Also, it is common that nodes remain functional most of the time [4]. This network traffic is also associated with packet flow in which it is not common for the information units to be stored for a long time before reaching destination. In these systems, buffer size implemented is relatively small, and optimized for keeping a low packet drop rate due to buffer overload. The Internet is a global switching packet network based on traffic with these features, and relies on

protocols, like TCP/IP that were developed for these structures. The TCP protocol is commonly said to be a conversational protocol because a complete one-way message involves many sources-to-destination signaling round trips [3].

While internet technologies began to show expansion of wireless transmission, space agencies tried to extend the communicating ranges of satellites, shuttles and several probes installed in outer space, and it was clear that the equipment used would be subject to severe ambient conditions, causing great connectivity restrictions. Communicating devices in these scenarios demand tolerance of delay in transmission, and must deal with high bit error rate and useful information loss, and there even may not exist an end-to-end path between nodes: the network protocols must account for these conditions. Each node must operate independently, as most of the network elements are often unavailable to operate [4].

Arriving data or data that needs retransmission due to previous failure may have to be queued until next contact becomes available, which may take unpredictable time. Bandwidth asymmetry together with unpredictable delay makes the design of transmission/retransmission timers harder. However, users need to be continually informed of the delays and transmission errors, and the need of interactiveness at this level consumes bandwidth and power resources. Also, the possibility of the network nodes suffering buffer's overflow may lead to the loss of critical information. Multiple transmission can ensure timely delivery, but may cause battery depletion, and shutdown of critical network elements may occur. Message duplication increases delivery ratio and decreases delivery delay, however it is associated with high power consumption and network nodes become more prone to buffer overflow. Replicating the message to high number of nodes increases the probability that the information finds adequate path to the destination within cutoff time, however the repeating of message to nodes that are unavailable to accommodate new information leads to waste of resources: It is generally agreed that obtaining both high delivery rates and low delays simultaneously is difficult, if not impossible – network resources must be carefully managed in order to obtain desired purpose [4].

The problem associated with communications in areas with these connectivity adversities is directly associated with the concept of Intermittently Connected Network (Challenged Network) which is an infrastructure-less wireless network that supports the proper functionality of one or several wireless applications operating in stressful environments, where excessive delay and unguaranteed existence of end-to-end path(s) between any arbitrary source-destination pair, result from highly repetitive link disruption [4]. Delay and disruption tolerant network (DTN), however, consists in a network of smaller networks, which allow communication between intermittent nodes by isolating delay and disruption with store-and-forward technique. These systems support compatibility with other networks, and can accommodate many kinds of wireless technology, such as radiofrequency (RF), optical signals, ultra-wide band (UWB) and acoustic [3]. In severe environments, packet loss may be very high due to intermittent connectivity. If a packet cannot be immediately forward, it will be usually dropped (discarded) and the TCP protocol allows a slower retransmission in this case. DTNs overcome the problem of intermittent connectivity and long or variable delays with the so called store-and-forward process. This switching method consists in

storing information in on node, which moves toward another node and forwards the information to him. The second node stores information and forwards it when finds another node of interest and so the store and forward method is repeated, describing a path between source and destination, until adequate transfer of data is accomplished [4].

The message switching implied with store and forward implementation is based on an architecture known as bundle protocol, represented in figure 2.5. This structure ties together the lower-level protocols, so applications can communicate between themselves, regardless of their lower-level protocol in conditions of high and variable delay - it is possible to achieve isolation between conversational protocols and delay with bundle layer. Bundles (RFC5050 [28]) are identified by creation timestamp, source EID fragment offset (in case its fragmented) and payload size. These units are stored and forwarded between nodes by bundle protocol agents, and can be converted to Application Data Units (ADU) at the bundle layer (RFC4838 [27]). A single bundle protocol is used in a DTN [3], concerning three main entities: Application Agent, Bundle Protocol Agent (BPA) and Convergence Layer Adapter (RFC5050).

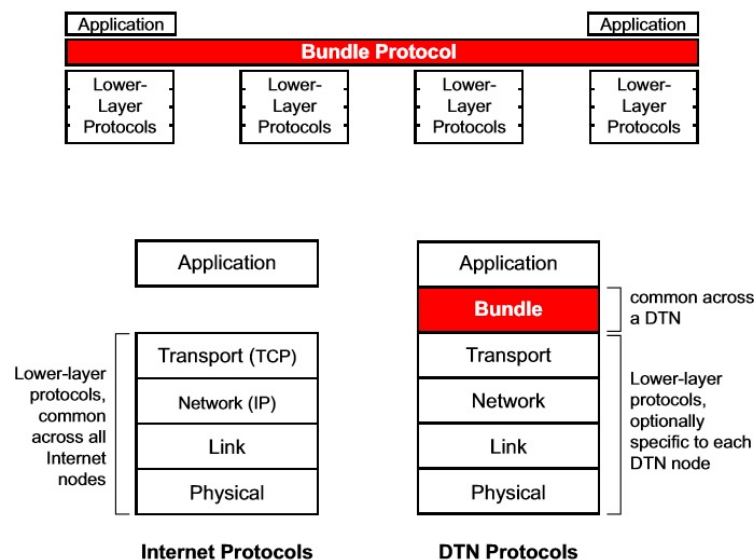


Figure 2.5: Architecture of bundle protocol [3]

Store and forward, represented in figure 2.6, depends on messages being stored indefinitely (persistent storage). DTN routers need persistent storage for handling their packet queues, because the communicating link may not be available for their next hop, as also there can be the case where one node may be much faster than the other in eventual commutating pair. Also, a message may need to be retransmitted if an error occurs, or if the upstream node declines to accept the forwarding of message. Network nodes have knowledge of message sizes, by moving a whole message in a single transfer (message switching technique): this is also helpful in adjusting the requirements for intermediate storage space and retransmission bandwidth [3].

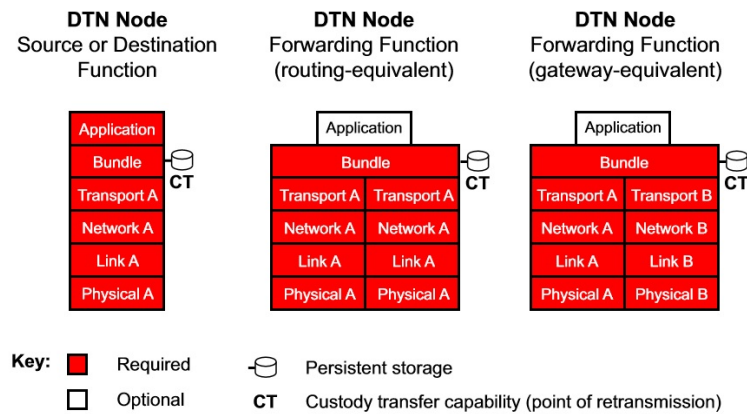


Figure 2.6: Store and forward [3]

Data delivery is achieved in DTN with nodes having storage capability augmented and being equipped with relatively large buffer size, enabling the system to preform the store-and forward process. Unknown network capability impels nodes to launch multiple message copies to increase message delivery probability, in a process called flooding [4]. However, this mechanism causes rapid buffer overflow, and therefore the drop rate of information generally increases. Buffer size is for this reason treated as an important resource to be managed and analyzed for effective transmission.

Nodes are identified in DTN's by Endpoint Identifiers (EIDs) (maximum length of 1024 bytes) [4] that can be treated as Uniform Resource Identifier (URI). Each EID can identify a single node or a group of nodes, in latter case with support to multicast.

Communication in DTNs is subject to various time dependent constrains and is characterized by begin and end instants, directions and endpoints and, most importantly link capacities and delays. Also, communication in these networks is not always bidirectional. Therefore, usually DTNs are represented by abstract graphs [4], as can be seen in figure 2.7.

In TCP protocol, data transfer service is reliable, and only end nodes are responsible for acknowledging the reception of error-free packets, or requesting retransmission of lost or corrupted data. This process works, because most of the nodes are available in traditional internet infrastructures. In DTNs however, network constrains previously referred make this process impossible. Storage of packets for long periods of time and retransmission of data made necessary to introduce a mechanism in bundle protocol called custody transfer, to convey the responsibility of retransmission of a bundle (or fragment) that has not yet reached final destination, to another node.

The node that currently holds custody of the bundle is called the bundle custodian, and ultimately works as the bundle source. Custody may be transferred to other intermediary nodes, however the next hop chosen must be closer to the bundle destination, demonstrating favorable storage capability and enough power [4].

Store and Forward mechanism is illustrated in figure 2.8 can be explained in the following

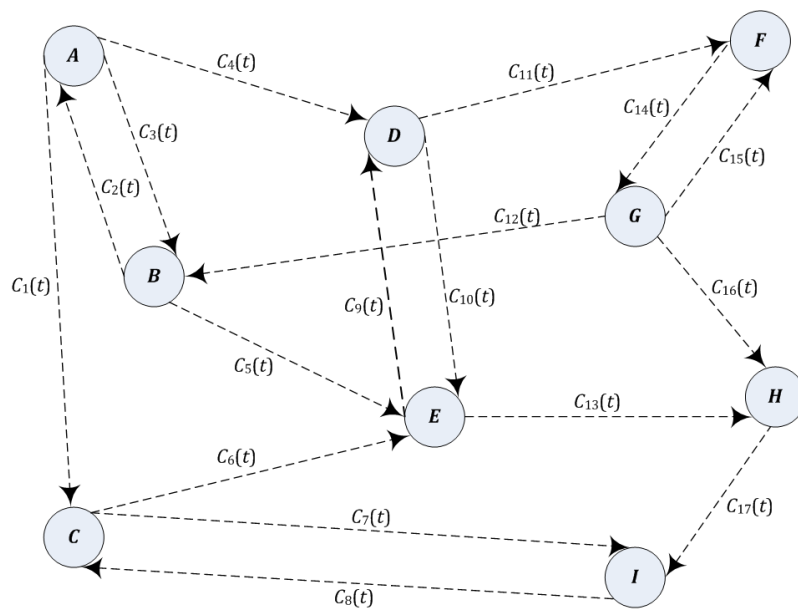


Figure 2.7: DTN graph abstraction [4]

example [4]: source S encounters a node I1. S initially transfers a bundle to I1, which in turn receives and stores it in its persistent storage space. This occurs with S transmitting a special request bundle to I1, asking to take custody of the bundle, and starts a time-out timer. If there is no reply from I1 before the timer expires, S then retransmits the bundle again, followed by another request, however if I1 accepts custody, it will transmit an acknowledgment bundle back to S. After receiving the ACK, S deletes the bundle from its buffer and ends the process successfully. This mechanism is continually repeated until the bundle finds his last destination. Generally, an additional option exists to enable confirmation from destination of the correct bundle arrival [4].

Bundle protocol can not determine if a received bundle is corrupted [4]. Also, custody transfer may not be confirmed correctly if communication between receiver and source suddenly becomes unavailable. There is also the possibility that node congestion arises from the presence of malicious elements or other factors that make buffer space unavailable and bundle drop arises.

The storing of bundles demands resources that must be freed after some time, when elements are considered invalid: for this effect, the timestamp of bundles is evaluated together with corresponding lifetime, and in case the element is considered expired, it can be deleted in order to free buffers and memory. Correct treatment of data transfer demands adequate synchronization between nodes in these networks, which can be very difficult to obtain with DTNs connectivity restrictions and strong delay.

Problems may arise, for in restricted network conditions nodes may not present correct time value: in some cases, a node may be chosen as a time reference if the system displays a reliable clock signal and periodically informs the other elements about the current time. The solution may also involve the introduction of a gateway element node, working as a Network Time Protocol (NTP) client, obtaining time information from a server outside the DTN restrictions - the other

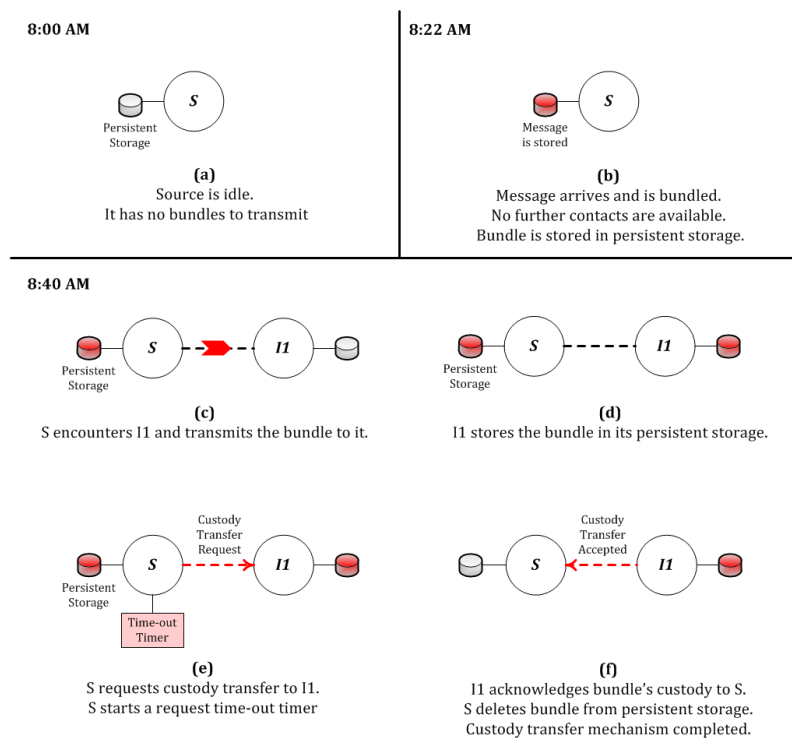


Figure 2.8: Store and forward mechanism [4]

nodes may obtain the right clock value from these gateways. However, these synchronization processes are conditioned by connectivity between nodes, which in DTN context is not always guaranteed.

2.2.3 Message sequence diagrams of bundle protocol

Store and forward mechanism can be expressed by a message sequence diagrams. For a scenario with three nodes, having EID's `dtm://node1.dtm`, `dtm://node2.dtm` and `dtm://node3.dtm`, if `dtm://node1.dtm` wants to send a bundle to `dtm://node3.dtm`, with `dtm://node2.dtm` moving between them as a data mule node to carry the bundle, the expected exchange of messages will be as represented in figure 2.9.

Sending node `dtm://node1.dtm` receives an "Hello message" from a custodian node `dtm://node2.dtm`, and then sends a "forwarding-bundle request", containing information about the destination EID of the bundle, to the custodian node for obtaining permission to send the bundle [29].

The custodian node then sends a "forwarding-bundle response", which confirms or declines acceptance of the bundle custody. In case of custody acceptance, the sending node `dtm://node1.dtm` sends a copy of the bundle to the custodian node `dtm://node2.dtm`. Otherwise, custody can be requested to another node [29].

After receiving the bundle, the custody node `dtm://node2.dtm` sends an acknowledgement (ACK) message to the sending node, acknowledging that it has received the bundle. The sending node

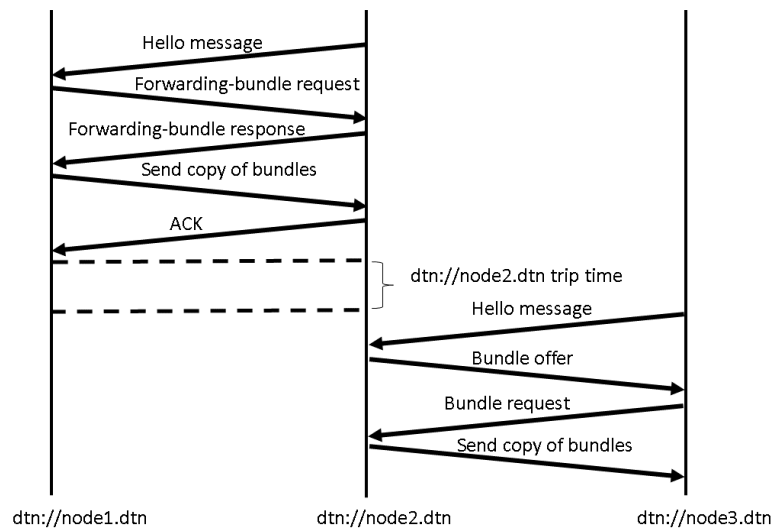


Figure 2.9: Message sequence diagram between two nodes, with a data mule

never repeats transmission of a bundle that has the same identifier as the one contained in the ACK message. The custody node can now request custody to another node, in a similar way.

After some travel time, when the custodian node reaches the receiver node `dtn://node3.dtn`, with EID matching the bundle destination EID, receiving node sends a “Hello message” to custodian node. This one returns a “bundle offer”, containing the bundle identifiers of the bundles destined to the EID of the receiving node.

A “bundle request” is returned by receiving node, containing the bundle identifiers of elements in the “bundle offer” that it does not have. Finally, custodian node sends the requested bundles to receiving node `dtn://node3.dtn`. For adequate reception of data, the receiver generally must set DTN application before sender emits message, in order to process bundles right upon reception.

2.2.4 Bundle Protocol Implementations

The current most sophisticated DTN implementations supported by Delay Tolerant Networks Research Group (DTNRG) are Delay Tolerant Networking version 2 (DTN2) and Interplanetary Overlay Network (ION) [30]. The Bundle Protocol reference is currently DTN2 implementation (the original prototype implementation). On the other hand, ION is not just an implementation of Bundle Protocol, since it also implements the Licklider Transmission Protocol (LTP) as well as Consultative Committee for Space Data Systems (CCSDS) File Delivery Protocol (CFDP) and Asynchronous Message Service (AMS). Comparing both implementations, DTN2 is taken as more flexible and flexible to experiment with Bundle Protocol [30].

DTN Reference Implementation (DTN-RI) was developed in Dublin Ireland, by Trinity College. It was designed for Linux, Solaris, Win32 (through Cygwin environment), Linux on Personal Digital Assistant (PDA)-ARM, FreeBSD, and Mac OS X. DTN-RI was initially developed in C++, and included a discrete event simulator for testing and prototyping. DTN-RI latter evolved into

current DTN reference implementation - DTN2. This platform is written in C++ and tested on Linux (x86 or x64) and Mac OS X (PPC and x386). DTN2 supports tablebased routing, Bonjour, Probabilistic Routing Protocol using History of Encounters and Transitivity (ProPHET), Delay Tolerant Link State Routing (DTLSR), and epidemic routing and also external routing via Extensible Markup Language, (XML) messaging [31].

The DTN2 implementation was created as a platform where researchers can validate the protocol design, and do experiments to show that the DTN protocol is working as expected or suitable for a given application. Besides implementing the bundle protocol, DTN2 also provides a number of routing mechanisms to direct the forwarding of bundles to their intended destinations. The reference implementation includes a number of Convergence Layer Adapter (CLAs) that interface between the bundle protocol and the transport.

The convergence layer manages the interfacing with particular underlying protocol, network or device and presents a consistent interface to the bundle layer. Complexity of the convergence layer is mostly determined by the underlying protocol it adapts: a TCP/IP convergence layer used for internet, for example, might only have to add message boundaries to TCP streams, whereas a convergence layer for some network where no reliable transport protocol exists may be much more complex (implementation of reliability, fragmentation, flow control, and other features) in order to obtain reliable delivery to the bundle layer. DTN2 includes a special convergence layer called External Convergence Layer (ECL), which allows the implementation of CLAs outside the DTN2 code base. CLAs implemented with this interface exchange XML messages with ECL via a TCP/IP socket.

ION – Interplanetary Network (ION-IPN) project required implementation for space of Jet Propulsion Laboratory (JPL), which led to ION development at the University of Ohio. ION is a software implementation of the bundle protocol stack and capable of routing for space environment. ION provides an implementation in the C language for the Bundle Protocol, the LTP, the Consultative Committee for Space Data Systems (CCSDS) protocols CFDP and Asynchronous Message Service AMS, and the Contact Graph Routing (CGR) algorithm, which is considered to be the most suitable for space contacts. ION is tested on Linux, Mac OS X, FreeBSD, Solaris, Real-Time Executive for Multiprocessor Systems RTEMS, and VxWorks. Includes supports for TCP, UDP, and LTP as convergence layers. Saratoga space protocol is also supported on ION [31].

Currently, other known DTN implementations are available, and some of them are referred in table 2.4.

Table 2.4: Implementations for DTN

Name	Source code	Operating Systems
Bytewalla [31]	Java	Android
IBR-DTN [31]	C++	OpenWRT
POSTELLATION [31]	C	Linux, Windows

2.2.5 Applications over Bundle Protocols implementations

Some applications over bundle protocols can be found, such as BPTAP (IP over DTN implementation) [32], HTTP over DTN (HTTP-DTN) [33], e-mail [34] and peer-to-peer [35]. However, some improvements can be made in the existing platforms and also developments of new solutions.

The interest of using embedded systems on drones imposed the need of using light operating systems such as OpenWRT. This software is suited for networking embedded platforms and provides great resources for support off-the-shelf wireless routers [36]. Attending to the advantages of applying this operating system, IBR-DTN was also considered an interesting platform: the software was developed in order to obtain an implementation of the bundle protocol in low performance devices and also in standard Linux computers [37].

The IBR-DTN software provides not only the daemon and tools to set up a DTN on embedded hardware or standard PC's, but also includes features like TCP and UDP convergence layer [38] [39], IP neighbor discovery (IPND) and epidemic routing support, besides standard bundle protocol. [40]. A fully functional Bundle Protocol implementation of IBR-DTN for Android is also available. IBR-DTN can run on smartphones and is compatible with other Bundle protocol implementations, such as DTN2 or ION [41]. Some applications included in this software, such as `dtnd`, `dtntping`, `dtntsend`, `dtntreceiv`, `dtntinbox`, `dtntoutbox`, are useful in bundle transfer customization and will be briefly explained in the following chapter.

The background process running in every DTN node is called DTN daemon [42]. In IBR-DTN case, this element implements an architecture presenting the following entities:

- **Event switch:** allows modules to register themselves and supports raising of events. Represents the core module of IBR-DTN, enabling the interaction between the different modules.
- **Bundle storage:** before transmission, the bundles can be stored (store and forward) or retrieved after reception from the bundle storage. This process may use RAM memory, SQL database or persistent file system for consistent element storing.
- **Connection Manager:** provides interaction between the transport layer (TCP, UDP, HTTP) and the bundle layer. This mechanism is supported by the convergence layer.
- **Discovery Agent:** implementation of the DTN node awareness, supporting various discovery plug-ins. Supports two compatible plug-ins: the IP-Discovery frames compatible to DTN2 and DTN IP Neighbor Discovery.
- **API Server:** socket based Application Programming Interface (API) that comes from the TCP convergence layer. This interface could be a Unix domain socket or TCP based socket.
- **Wall Clock:** reads the clock of the host to determine the global time. The timestamp provided by the wall clock is used as a reference for determining when a bundle should expire. This process allows time synchronization between the nodes.

- Base Router: implements different routing protocols as plug-ins of IBR-DTN. The base router module communicates with the discovery agent module through events, determining when a DTN node has been discovered.

Chapter 3

IBR-DTN applications

The following chapter presents the hardware and software choices in order to support DTN applications, and also results and conclusions of tests carried with these applications.

3.1 Hardware specifications

This thesis considers the creation of three underwater nodes representing the AUV, the data mule and the receiver. The hardware must be capable of being integrated in airtight cylinders and communicate using IEEE 802.11n standard. Wireless network interfaces used for this project was the RouterBOARD R52n-M (represented in figure 3.2) in the 2.4 GHz band, using IEEE 802.11n standard [15]. These wireless cards were assembled in the system board alix3d3 from PC Engines (represented in figure 3.1), and carefully inserted in airtight cylinders manually created at INESC TEC CTM unit, which in turn will act as emitter (AUV), receiver and drone (data mule). Previous work made at INESC TEC proved that PC Engines ALIX3D3 are a robust and reliable solution for research. These elements allow wide range of applications, as in industrial development, firewall or routers implementations. Presenting a 500 MHz AMD Geode LX800 processor and 256 Mbyte DDR RAM, ALIX3D3 allows full range 32 bit x86 application. In this thesis, these boards were powered by passive Power over Ethernet (PoE), which also includes LAN port for communicating with other devices. The two USB ports were used, either for connecting a keyboard when needed or inserting the Operating System in a Pen Flash Drive. Also, compact size of this device makes it suitable for being enclosed in the acrylic cylinders created at INESC TEC. Specifications associated with ALIX3D3 are presented in table 3.1. The nodes can be autonomous, with the use of batteries suitable for the effect.

Table 3.1: Characteristics of ALIX3D3 Boards

Parameter	Value
CPU	500 MHz AMD Geode LX800
RAM	256 Mbyte DDR DRAM
Power	DC jack or passive POE, min. 7 V to max 20 V
Expansion	2 miniPCI slots, LPC bus

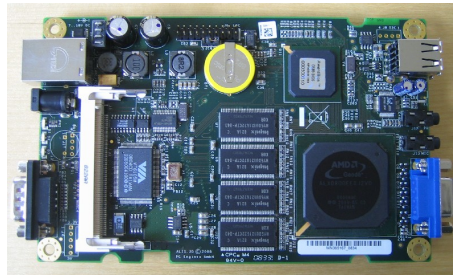


Figure 3.1: Alix3d3 board [5]



Figure 3.2: MikroTik routerboard r52n-M [6]

The Routerboard r52n-M presents high performance in 802.11a/b/g/n standards, in both 2.4 and 5 GHz bands, this card also presents sturdy MMCX connectors for use of external antennas and adds extra durability. Physical rates supported are about 300 Mbit/s and is possible to obtain up to 200 Mbit/s actual user throughput, uplink and downlink, with MIMO technology. Also, power transmission output can be up to 23 dBm. Atheros AR9220 chipset present in this board, and fully compatible drivers are available and continually optimized for Linux operating systems, such as ath9k driver. More relevant characteristics of this wireless card are summarized in 3.2.

Table 3.2: RouterBOARD r52n-M Specifications

Parameter	Value
Chipset	Atheros AR9220
Standard	Dual band IEEE 802.11a/b/g/n standard
Output power	up to 23 dBm
Antenna Connectors	Two MMCX Antenna Connectors
Operating temperatures	-50° C to +60° C
Performance	up to 300 Mbit/s physical data rates

3.2 Software specifications

The software used in the dissertation will be mainly the two following platforms:

- Debian - this operating system presents easy installation, stability, strong security and very good packing system avoiding conflict of software. Also provides compatibility with many hardware architectures and kernels. Packages are very well integrated in Debian, allowing easy upgrades and most drivers available are open source;

In initial stage of dissertation, OpenWRT was used, and for the effect an image was compiled from source code obtained from github repositories, including all software packages needed. However, connection losses were found when transferring files of size 100 Mbyte, with available version ath9k in Chaos Calmer version. Trunk version of the software showed unstable initialization of IBR-DTN daemon in many situation with ad-hoc network configuration. Also, some features of IBR-DTN were not available in OpenWRT, and found in Debian. However, IBR-DTN was studied in a first stage with OpenWRT and features regarding synchronization of nodes and connectivity were tested.

- IBR-DTN - this software provides a bundle protocol suitable for embedded devices and is compatible with Debian and OpenWRT [43]. More description of this package can be found in chapter 2.2.4.

IBR-DTN defines a set of applications used for communication and file sharing. This chapter describes some of the functional characteristics and tests performed with these tools.

3.3 IBR-DTN configuration and API

After installation, IBR-DTN must be configured in order to associate the daemon with the adequate network interface, and also for choosing the node as a time reference announcer or a slave depending on foreign time message. The suitable configuration may be verified with the following command:

```
telnet ip 4556
```

which should respond with a message including the host EID.

IBR-DTN API

User can get access to the API of IBR-DTN and obtain the information about neighbor list with the following commands: (represented in figure 3.3):

```
telnet localhost 4550
```

```
protocol extended
```

```
neighbor list
```

```

root@OpenWrt:~# telnet localhost 4550
IBR-DTN 1.0.1 (build 294b543) API 1.0
protocol extended
200 SWITCHED TO EXTENDED
neighbor list
200 NEIGHBOR LIST
dtn://node1.dtn
dtn://node2.dtn
dtn://node3.dtn

```

Figure 3.3: Example of the IBR-DTN API use

3.4 dtnd

This application is used start the IBR-DTN daemon.

Example: (represented in figure 3.4)

dtnd

```

root@OpenWrt:~# dtnd
Tue Jan 31 23:10:56 2017 INFO NativeDaemon: IBR-DTN daemon 1.0.1 (build 294b543)
Tue Jan 31 23:10:56 2017 INFO Configuration: Using default settings. Call with -
-help for options.
Tue Jan 31 23:10:56 2017 INFO BundleCore: Local node name: dtn://OpenWrt
Tue Jan 31 23:10:56 2017 INFO BundleCore: Forwarding of bundles enabled.
Tue Jan 31 23:10:56 2017 INFO NativeDaemon: using bundle storage in memory-only
mode
Tue Jan 31 23:10:56 2017 INFO NativeDaemon: API initialized using tcp socket: lo
opback:4550
Tue Jan 31 23:10:56 2017 INFO DiscoveryAgent: listen to [ff02::142]:4551
Tue Jan 31 23:10:56 2017 INFO DiscoveryAgent: listen to [224.0.0.142]:4551
Tue Jan 31 23:10:56 2017 INFO NativeDaemon: Using default routing extensions

```

Figure 3.4: Output of dtnd invocation

3.5 dtnping

This application is used to ping another dtn instance, and can be used to verify the presence of nodes carrying IBR-DTN daemon in the network. Example (represented in figure 3.5):

dtnping dtn://node3.dtn/echo

This command sends a request to `dtn://node3.dtn`, which in case of the communication is established, sends a echo reply to the requesting node.

The IBR-DTN applications were tested in this thesis with intention of characterizing their functionality and limitations in this platform. Connectivity loss in this stage was simulated with iptable rules to block traffic between nodes in scheduled scripts. The three Alix3d3 nodes used were connected to an ethernet switch (node A: IP=192.168.1.1 netmask=255.255.255.0; node B: IP=192.168.1.2, netmask=255.255.255.0; node C: IP=192.168.1.3, netmask=255.255.255.0).

```

root@OpenWrt:~# dtntping dtn://node3.dtn/echo
ECHO dtn://node3.dtn/echo 64 bytes of data.
64 bytes from dtn://node3.dtn/echo: seq=1 ttl=30 time=12.04 ms
64 bytes from dtn://node3.dtn/echo: seq=2 ttl=30 time=12.94 ms
64 bytes from dtn://node3.dtn/echo: seq=3 ttl=30 time=13.91 ms
64 bytes from dtn://node3.dtn/echo: seq=4 ttl=30 time=11.66 ms
64 bytes from dtn://node3.dtn/echo: seq=5 ttl=30 time=14.73 ms
64 bytes from dtn://node3.dtn/echo: seq=6 ttl=30 time=14.89 ms
64 bytes from dtn://node3.dtn/echo: seq=7 ttl=30 time=14.25 ms
64 bytes from dtn://node3.dtn/echo: seq=8 ttl=30 time=11.72 ms
64 bytes from dtn://node3.dtn/echo: seq=9 ttl=30 time=16.05 ms
64 bytes from dtn://node3.dtn/echo: seq=10 ttl=30 time=17.70 ms
^C
--- dtn://node3.dtn/echo echo statistics ---
10 bundles transmitted, 10 received, 0% bundle loss, time 9.58 s
rtt min/avg/max = 11.66/13.99/17.70 ms

```

Figure 3.5: Example of the dtntping use

Elements were synchronized with NTP according to the time imposed by INESC TEC PC (IP= 192.168.1.10, netmask = 255.255.255.0), once this network allowed direct access to all three nodes and synchronization in this case avoid choosing a master and slave nodes. The experimentation testbed is represented in figure 3.6.

In all preformed 6 testes, the Wi-Fi network used was named "teste", in ad-hoc mode. The channel selected was 36 (f=5.18 GHz) and bandwidth 20 MHz (mode 802.11n) for low interference and high throughput. Power output in wireless board was 23 dBm, corresponding to maximum value on the hardware.

Node A was set with IP= 10.0.0.1 netmask = 255.255.255.0, node B IP= 10.0.0.2 netmask = 255.255.255.0 and node C IP= 10.0.0.3 netmask = 255.255.255.0.

The EID nodes of the elements were: node A dtn://node1.dtn; node B dtn://node2.dtn and node C, dtn://node3.dtn. Routing configuration selected in IBR-DTN was set to flooding, considering the advantages and simplicity in transmitting data between nodes with this configuration in small networks. Table 3.3 shows different addresses and EID chosen for each node.

Table 3.3: IP configuration of Ethernet and Wi-Fi addresses used in the tests

Node	IP (Ethernet)	IP (Wi-Fi)	EID
A	192.168.1.1	10.0.0.1	dtn://node1.dtn
B	192.168.1.2	10.0.0.2	dtn://node2.dtn
C	192.168.1.3	10.0.0.3	dtn://node3.dtn

3.6 dtnrecv

This application is used to receive bundles, and must be executed before these elements are sent in the network [42]. On reception, it displays the information received in the standard output.

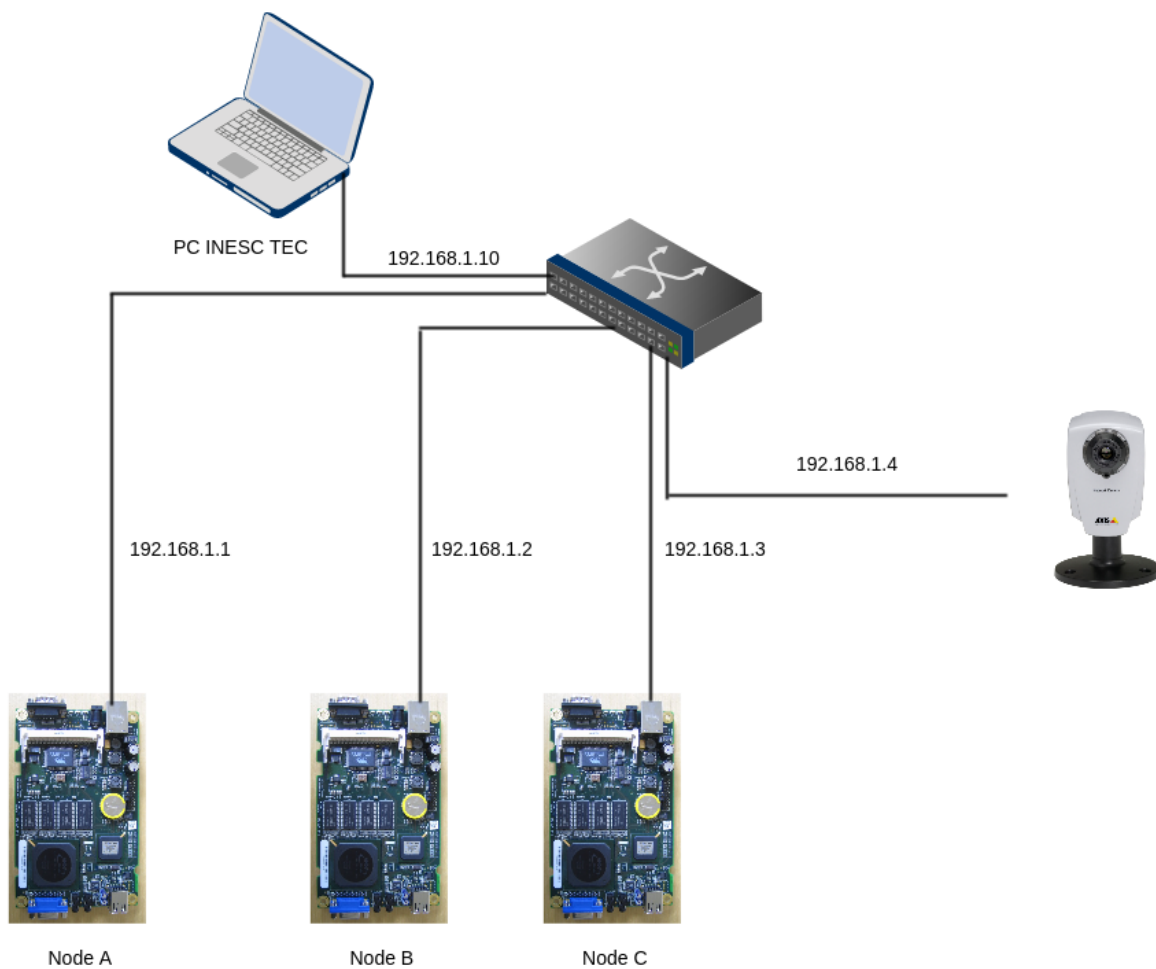


Figure 3.6: Testbed for simulation of datamule between sender and receiver nodes

Apart from this waiting process, the bundle is received by the destination node automatically, upon discovery instant of the communicating pair. [42].

Example (represented in figure 3.7):

dtmrecv -name dtmReceiver

```
root@OpenWrt:~# dtmrecv --name dtmReceiver
Hello!
```

Figure 3.7: Example of the dtmrecv usage

3.7 dtnsend

This application is used to send bundles generated from files to a chosen destination [42]. The dtn daemon access is provided by IBR-DTN API and injects the bundles in the DTN.

Example: (represented in figure 3.8)

```
dtnsend dtn://node3.dtn/dtnReceiver
```

```
root@OpenWrt:/# echo Hello! > message
root@OpenWrt:/# dtnsend dtn://node3.dtn/dtnReceiver message
Transfer file "message" to dtn://node3.dtn/dtnReceiver
```

Figure 3.8: Example of the dtnsend use

These applications were tested in the following procedures, presented in test 1 and 2. The application dtnsend was tested with four files, sending data from node A (dtn://node1.dtn) to node C (dtn://node3.dtn) with the following commands:

In node A:

```
dtnsend dtn://node3.dtn/dtnReceiver <filename>
```

The waiting process was applied in node C without direct connectivity between A and C, with the command:

```
dtnrecv -name dtnReceiver > <filename>
```

where <filename> was the chosen name for destination of receive archive.

Direct connectivity between nodes A and C is avoided in the procedure, and data between these two elements is carried by node B, assuming function of data mule.

Store and forward and fragment test

- **Objective**

The purpose of this test was to verify the store and forward mechanism, as well as fragment creation from imposed maximum size in configuration file. The fragmentation size imposed was 500 kbytes. Also, this experiment intended to determine the time interval between node detection and file sending.

- **Procedure**

The beginning of each procedure consisted in blocking traffic in nodes A and C. Next, IBR-DTN daemon would be initialized in three nodes, together with packet capture with tcpdump. Traffic in node A was unblocked, and dtnsend application was then applied in this node for transfer the file with destiny to node C. Here, the dtnrecv application was applied for adequately receiving the file. In node B, the access to the API for verification of received bundles allowed the confirmation of accepted fragments, and after sending the file, node A would be blocked again. Finally, node C would be unblocked for receiving the file. Traffic was blocked with following command:

```
iptables -A INPUT -i adhoc -j REJECT
```

```
iptables -A OUTPUT -o adhoc j REJECT
```

Consequently, unblocking of traffic was imposed with the instruction:

```
iptables -F
```

The movement simulated in this test is represented in figure 3.9. Four files were tested, with ten repetitions of this procedure for each archive.

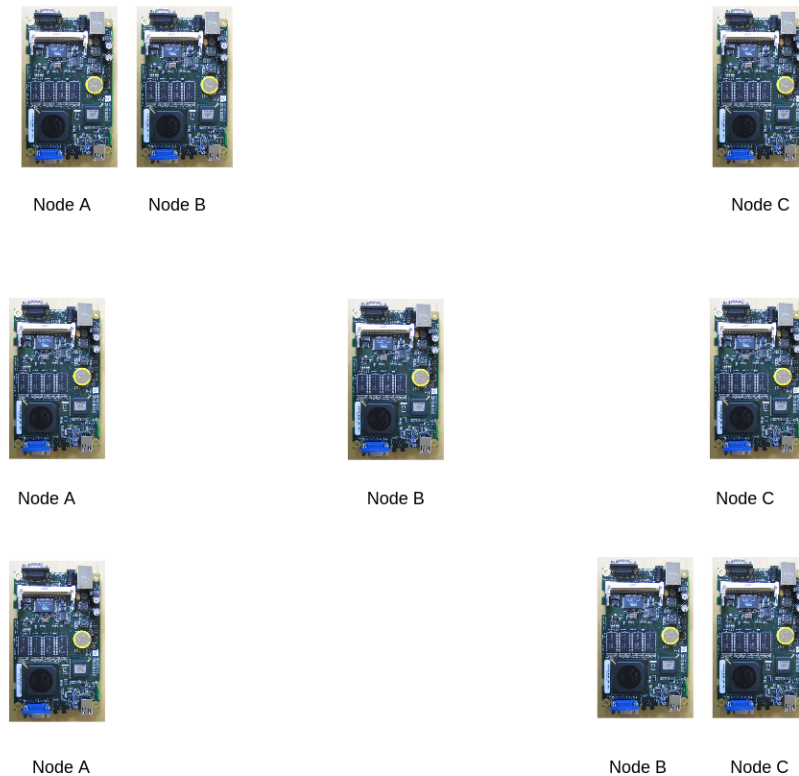


Figure 3.9: Movement of data mule (node B) simulated between nodes A and C in test 1

- **Results and conclusions**

Experiments with dtnsend application indicated problems in sending files of size bigger than 50 Mbytes due to processing difficulties with limited hardware RAM memory, and so this test procedure was carried with small archives in order not to interfere with daemon functioning. From verification with md5 checksum integrity, all files were correctly received in node C. The API messages regarding number of bundles received in node B demonstrated correct fragmentation of elements, in the first stage, when the number of bundles stored demonstrated division of files in 500 kbytes bundles. When node C was unblocked, API in node C informed these bundles to be deleted. Table 3.4 shows the number of fragments obtained for each file, and figure 3.10 plots the number of bundles generated against file size.

Wireshark logs analysis also showed a first stage of bundle transfer, between nodes A and B, and a final stage, between nodes B and C. In node A logs, a peak of transfer was registered,

Table 3.4: Number of fragments obtained in each file

File size (Mbytes)	Bundle fragments
4.8	10
9.6	20
19.3	39
49.7	100

representing bundle transfer from A to B. In node B two peeks were recorded, corresponding to transfer of bundles from node A to B and from B to C, as for in node C a peek was registered corresponding to transfer of bundles from B to C.

Also, these bundle transfer occurrences were registered in conformance to node availability, regarding the predicted contacts between elements. Results expressed in table 3.5 correspond to time interval between start of node detection, regarding nodes B and C, and start of bundle transmission for files of size 4.9, 9.6, 19.3 and 49.7 Mbytes. Table 3.6 shows average time between node detection (node B detecting node C) and start of file exchange, corresponding to the average difference in start of bundle transmission and first packet exchange (showed in figure 3.11). Transfer of bundles between nodes A and B is represented in figure 3.12, and final delivery of elements from B to C is represented in figure 3.13.

Table 3.5: Time between start of node detection and start of bundle transmission

	4.9 Mbytes	9.6 Mbytes	19.3 Mbytes	49.7 Mbytes
Experiment	Time (s)	Time (s)	Time (s)	Time (s)
1	0.02	3.158	0.013	0.86
2	2.318	2.448	0.03	0.023
3	0.02	0.02	0.02	0.808
4	0.02	3.65	0.02	0.02
5	0.244	0.02	0.197	1.63
6	2.118	0.03	0.03	0.02
7	3.175	0.02	1.63	2.22
8	2.554	0.06	0.02	0.022
9	0.019	0.02	0.02	2.197
10	1.839	0.77	0.02	0.026

Table 3.6: Average time between start of node detection and start of bundle transmission

File size (Mbytes)	Average time between node detection and file sending (s)	Standard deviation (s)
4.8	1.23	1.27
9.6	1.02	1.47
19.3	0.2	0.51
49.7	0.78	0.924

Results suggest average time is low in our system, without introduction of high delays.

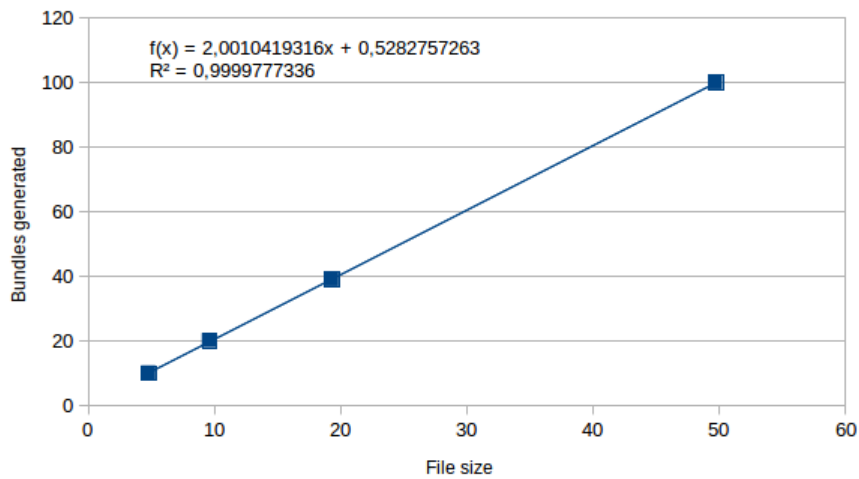


Figure 3.10: Number of bundles generated by fragmentation vs. file size

18	72.662384	10.0.0.2	10.0.0.3	UDP	100 9999 → 9999 Len=58
19	72.662764	10.0.0.3	10.0.0.2	UDP	91 9999 → 9999 Len=49
20	72.670902	10.0.0.2	10.0.0.3	TCP	74 48211 → 4556 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
21	72.670983	10.0.0.3	10.0.0.2	TCP	74 4556 → 48211 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 ...
22	72.671877	10.0.0.2	10.0.0.3	TCP	66 48211 → 4556 [ACK] Seq=1 Ack=1 Win=29216 Len=0 TSva...
23	72.672934	10.0.0.2	10.0.0.3	TCPCl	90 Contact Header
24	72.673044	10.0.0.3	10.0.0.2	TCPCl	90 Contact Header
25	72.673165	10.0.0.3	10.0.0.2	TCP	66 4556 → 48211 [ACK] Seq=25 Ack=25 Win=28960 Len=0 TSV...
26	72.674164	10.0.0.2	10.0.0.3	TCP	66 48211 → 4556 [ACK] Seq=25 Ack=25 Win=29216 Len=0 TSV...
27	72.681150	10.0.0.2	10.0.0.3	TCP	1514 [TCP segment of a reassembled PDU]

Figure 3.11: Wireshark capture in transferring file 4.9 Mbytes with dtmsend

21	41.028063	10.0.0.1	10.0.0.2	TCP	74 55106 → 4556 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
22	41.028143	10.0.0.2	10.0.0.1	TCP	74 4556 → 55106 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 ...
23	41.029382	10.0.0.1	10.0.0.2	TCP	66 55106 → 4556 [ACK] Seq=1 Ack=1 Win=29216 Len=0 TSva...
24	41.030542	10.0.0.2	10.0.0.1	TCPCl	90 Contact Header
25	41.031543	10.0.0.1	10.0.0.2	TCP	66 55106 → 4556 [ACK] Seq=1 Ack=25 Win=29216 Len=0 TSva...
26	41.033937	10.0.0.1	10.0.0.2	TCPCl	90 Contact Header
27	41.033974	10.0.0.2	10.0.0.1	TCP	66 4556 → 55106 [ACK] Seq=25 Ack=25 Win=28960 Len=0 TSV...
28	41.044692	10.0.0.1	10.0.0.2	TCP	1514 [TCP segment of a reassembled PDU]
29	41.044773	10.0.0.2	10.0.0.1	TCP	66 4556 → 55106 [ACK] Seq=25 Ack=1473 Win=31872 Len=0 T...
30	41.044845	10.0.0.1	10.0.0.2	TCP	1514 [TCP segment of a reassembled PDU]

Figure 3.12: Wireshark capture in transferring file 49.7 Mbytes between nodes A and B

42297	76.884621	10.0.0.2	10.0.0.3	TCP	74 33378 → 4556 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA...
42298	76.885546	10.0.0.3	10.0.0.2	TCP	74 4556 → 33378 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 ...
42299	76.885620	10.0.0.2	10.0.0.3	TCP	66 33378 → 4556 [ACK] Seq=1 Ack=1 Win=29216 Len=0 TSva...
42300	76.886694	10.0.0.2	10.0.0.3	TCPCl	90 Contact Header
42301	76.887721	10.0.0.3	10.0.0.2	TCPCl	90 Contact Header
42302	76.887780	10.0.0.2	10.0.0.3	TCP	66 33378 → 4556 [ACK] Seq=25 Ack=25 Win=29216 Len=0 TSV...
42303	76.887825	10.0.0.3	10.0.0.2	TCP	66 4556 → 33378 [ACK] Seq=25 Ack=25 Win=28960 Len=0 TSV...
42304	76.896695	10.0.0.2	10.0.0.3	TCP	1514 [TCP segment of a reassembled PDU]
42305	76.896871	10.0.0.2	10.0.0.3	TCP	1514 [TCP segment of a reassembled PDU]
42306	76.896945	10.0.0.2	10.0.0.3	TCPCl	1514 [Bundle TCPCl Segment][TCP segment of a reassembled ...

Figure 3.13: Wireshark capture in transferring file 49.7 Mbytes between nodes B and C

In experiment 1, for file of 4.9 Mbytes, figure 3.11 shows time since node discovery between nodes B and C and start of bundle exchange can be determined by difference between bundle emission at 72.68 s and first packet exchange at 72.66 s, resulting in the first value of table 3.5 of

0.02 s. Other values of table were determined in similar way. Table 3.7 shows time to copy a file from node A to C, compared with value of time for final delivery of archive from data mule B to node C.

Table 3.7: Average time of final delivery of bundles from node B to node C, compared with time to copy files from A to C

File size (Mbytes)	Time for copying file from node A to node C (s)	Final transfer from node B to node C (s)
4.8	2	3
9.6	4	4
19.3	9	7
49.7	22	18

3.8 dtinbox and dtnoutbox

The applications dtinbox and dtnoutbox allow the creation of a automatic process of sending bundles. The user can automatically send the files created in a selected outbox folder to a destiny inbox folder.

```
root@OpenWrt:/outBoxFolder/outboxFolder# echo content > file2
files sent: file1
files sent: file1
```

Figure 3.14: Example of the dtnoutbox use

```
root@OpenWrt:/inboxFolder# dtinbox inboxReceiver inboxFolder/
received bundle: [539193016.1] dtn://node2.dtn/outboxSender
received bundle: [539193018.1] dtn://node2.dtn/outboxSender
received bundle: [539193018.6] dtn://node2.dtn/outboxSender

received bundle: [539193061.1] dtn://node2.dtn/outboxSender
received bundle: [539193063.1] dtn://node2.dtn/outboxSender
received bundle: [539193063.6] dtn://node2.dtn/outboxSender
_
```

Figure 3.15: Example of the dtinbox use

Both applications dtinbox and dtnoutbox were tested in experiments presented in following tests.

One way folder transfer using dtnoutbox and dtininbox

- **Objective**

Verify the possibility of transferring files with dtininbox and dtnoutbox applications, using alix3d3 as data mule between two nodes.

- **Procedure**

The experiment started with the creation of folders outboxFolder in node A, and inboxFolder in node C. Files of size 4.9, 9.6, 19.3 and 49.7 Mbytes were placed in outboxFolder. Nodes A and C were blocked and the daemon was started in the three elements. The application dtnoutbox would be started in node A and in node C, the application dtininbox would set a process for reception of bundles. The following command was used in node A for this effect: `dtnoutbox outboxSender outboxFolder dtn://node3.dtn/inboxReceiver` while in node C, the following command was applied: `dtininbox inboxReceiver inboxFolder/`. Also, in the three nodes, `tcpdump` capture would be started for analyzing traffic in the procedure.

Two scripts were created in nodes A and C for simulation of data mule movement, node B, between them, imposing blocking and unblocking of traffic in synchronized fashion. The scripts included a cycle with initial blocking of node C and open traffic in node A, for simulation of docking time for 20 s between A and B, followed by blocking node A, for simulation of trip time of 20 s in node B, and ending with unblocking of node C for delivering bundles received from node A for 20 s. Following this procedure, return of node B to A was simulated with blocking node C for more 20 s. Cron service was set in nodes A and C so that both scripts would start simultaneously.

- **Results and conclusions**

In this experiment, correct transfer of files was confirmed with md5 checksum verification. For total transfer of files, three trips were needed in each of the three repetitions of the procedure.

For a total trip time of 80 s, total bit rate transfer can be quantified in the following determination:

$$R_{bit} = \frac{\text{Total size of data}}{\text{Total trip time}} = \frac{83.4 \times 8}{80 \times 3} = 2.78 \text{ Mbit/s.}$$

This estimation shows the possibility of obtaining high bit rates in data transferring, when gain is compared to values of acoustic transmission.

One way folder transfer using dtininbox and dtnoutbox simultaneously

- **Objective**

Experiment the effect of using dtininbox and dtnoutbox together, applied in the same folder for obtaining file transfer in selected paths.

- **Procedure**

The test was initialized with creation of folder TesteA in node A, containing files, of size 4.9, 9.6, 19.3 and 49.7 Mbytes. In node B, folder TesteB was created, represented in figure 3.16.

In both nodes, a tcpdump reading would be started, and the daemon was then initialized: in node A with TesteA defined with application dtinibox as inboxfolder and defined with application dtnoutbox as outboxfolder; in node B the folder TesteB was defined with application dtinibox as inboxfolder and as outboxfolder with application dtnoutbox.



Figure 3.16: Procedure applied in testing one way folder transfer using dtinibox and dtnoutbox simultaneously

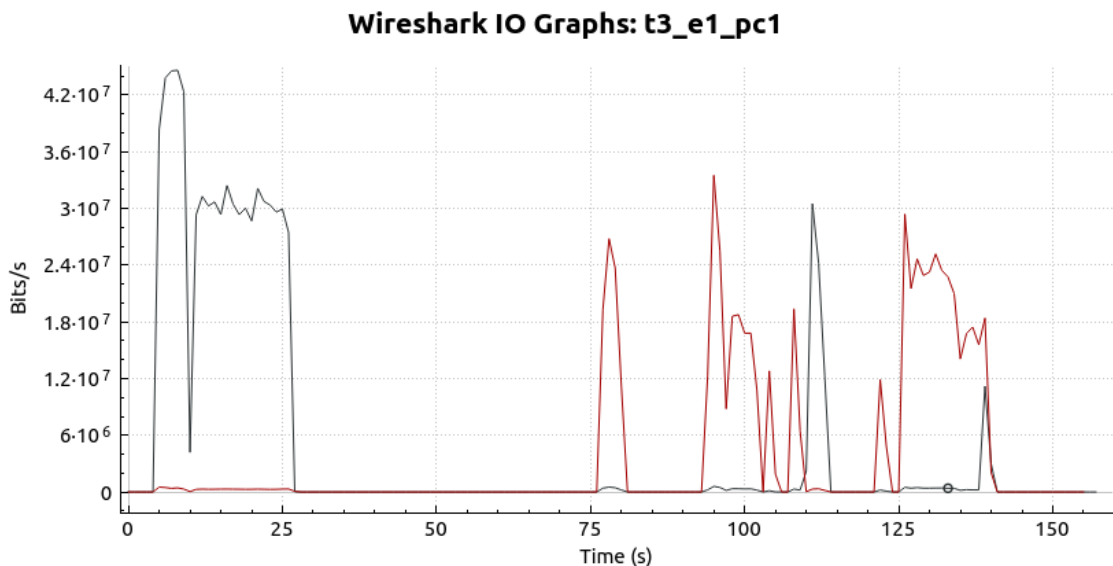


Figure 3.17: Results obtained in wireshark reading one way folder transfer using dtinibox and dtnoutbox simultaneously (packets blue have IP source 10.0.0.1 and packets red have IP source 10.0.0.2)

- **Results and conclusions**

In the end of each procedure, both nodes obtained equal files in folders TesteA and TesteB. However, its possible to conclude from wireshark log observation, showed in figure 3.17 that after node B receives the files, these are sent back to node A, resulting in unnecessary exchange of files already present in elements.

Bidirectional file transfer experiment with dtininbox and dtnoutbox

- **Objective**

Testing the use of dtininbox and dtnoutbox for file sharing in two folders with different initial content.

- **Procedure**

The test was initialized with creation of a folder named TesteA in node A, containing files, of size 4.9, 9.6 Mbytes. In node B, it was created a folder named TesteB containing files of size 19.3 and 49.7 Mbytes, represented in figure 3.18, and daemon was initialized in each element.



Figure 3.18: Procedure applied in testing bidirectional file transfer with dtininbox and dtnoutbox

In node A folder TesteA was defined with application dtininbox as inboxfolder and as outboxfolder with application dtnoutbox, as in node B TesteB was defined with application dtininbox as inboxfolder and as outboxfolder with application dtnoutbox.

- **Results and conclusions**

Similar to previous observations, both nodes obtained equal files in folders TesteA and TesteB in the end of each experiment. Whireshark graph regarding this experiment is represented in figure 3.19.

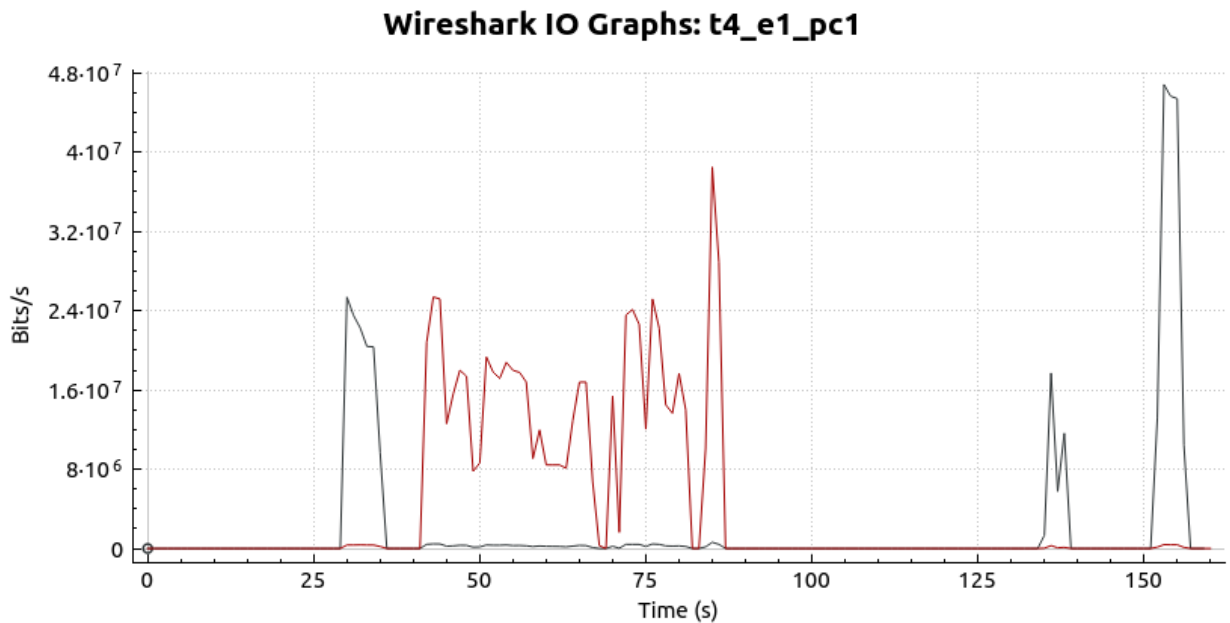


Figure 3.19: Results obtained in wireshark reading with bidirectional file transfer experiment with dtinbox and dtnoutbox (packets blue have IP source 10.0.0.1 and packets red have IP source 10.0.0.2)

Bidirectional file transfer experiment with dtinbox and dtnoutbox with the use of data mule

- **Objective**

Verify the use of dtinbox and dtnoutbox together for file sharing in DTN nodes, with the intervention of a data mule for bundle transport.

- **Procedure**

After experimenting the possibility of sharing files, folder named TesteA in node A was again filled with content similar to beginning of test 4, containing two files, of size 4.9 and 9.6 Mbytes. In node C, it was created a folder named TesteC containing two files of size 19.3 and 49.7 Mbytes.

Cron service was set to start scripts used in test 2 simultaneously, in order to simulate data mule travel of node B between nodes A and C.

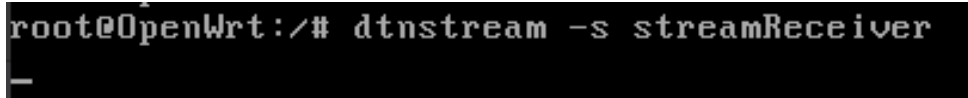
- **Results and conclusions**

In this experiment, in spite of exchange of bundles verified with wireshark, applications could not create the files in nodes A and C. Results suggest that in high disruption situations, convergence layer may fail bundle delivery, and so in future experiments docking time and travel period was raised.

3.9 dtstream

Streaming multimedia files is also possible with the IBR-DTN daemon usage, by invoking the application dtstream. Other node (in this case dtn://node2.dtn) can set the stream with the following command (represented in figure 3.20):

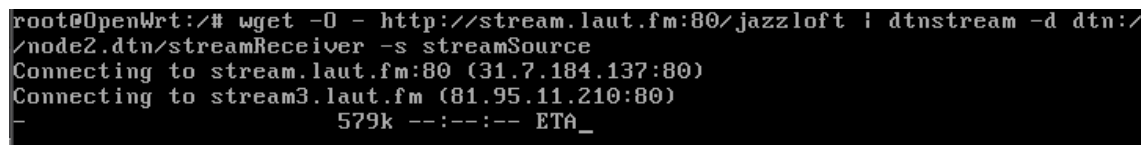
```
dtstream -s dtstreamReceiver
```



```
root@OpenWrt:/# dtstream -s streamReceiver
```

Figure 3.20: Example of the dtstream use - receiver

The other element can send the stream with the following command (represented in figure 3.21):



```
root@OpenWrt:/# wget -O - http://stream.laut.fm:80/jazzloft | dtstream -d dtn://node2.dtn/streamReceiver -s streamSource
Connecting to stream.laut.fm:80 (31.7.184.137:80)
Connecting to stream3.laut.fm (81.95.11.210:80)
- 579k --:--:-- ETA_
```

Figure 3.21: Example of the dtstream use - sender

The dtstream application was tested with procedures described in the following two tests.

Dtstream application for online radio streaming with use of data mule

- **Objective**

Test the use of a data mule between two nodes for transferring bundles with dtstream in streaming radio.

- **Procedure**

Verification of dtstream operation was carried with transfer of online radio music between nodes A (sender) and C (receiver). Node B would assume function of data mule, with scripts in nodes A and C similar to the ones applied in test 2. However, in this experiment, the docking time was raised for 60 seconds and fragmentation was set to 2000 kbytes in each node. The radio used for streaming was <http://stream.laut.fm:80/jazzloft>.

- **Results and conclusions**

In this experiment, it was not possible to obtain continuous display of radio in node C. However, the receiver could record the stream in a mp3 file, which showed correct display after collection of stream. Also, the reception of bundles in node C was confirmed with daemon messages. Wireshark capture of streaming with DTN is represented in figures 3.22 and 3.23.

22	70.052685	10.0.0.1	10.0.0.2	TCP	66	43237 → 4556 [ACK] Seq=25 Ack=25 Win=29216 Len=0 TSV...
23	70.059375	10.0.0.1	10.0.0.2	TCP	1514	[TCP segment of a reassembled PDU]
24	70.060807	10.0.0.1	10.0.0.2	TCP	1514	[TCP segment of a reassembled PDU]
25	70.060861	10.0.0.2	10.0.0.1	TCP	66	4556 → 43237 [ACK] Seq=25 Ack=2921 Win=34752 Len=0 T...
26	70.061765	10.0.0.1	10.0.0.2	TCPCPL	1514	[Bundle TCPCPL Segment][TCP segment of a reassembled ...
27	70.062377	10.0.0.1	10.0.0.2	TCP	842	[TCP segment of a reassembled PDU]
28	70.062432	10.0.0.2	10.0.0.1	TCP	66	4556 → 43237 [ACK] Seq=25 Ack=5145 Win=40544 Len=0 T...
29	70.062490	10.0.0.1	10.0.0.2	TCP	1514	[TCP segment of a reassembled PDU]

Figure 3.22: Whireshark capture of streaming between nodes A and B

8729	311.966919	10.0.0.2	10.0.0.3	TCPCPL	1514	[Bundle TCPCPL Segment][TCP segment of a reassembled ...
8730	311.967048	10.0.0.2	10.0.0.3	TCP	1514	[TCP segment of a reassembled PDU]
8731	311.967106	10.0.0.2	10.0.0.3	TCP	1514	[TCP segment of a reassembled PDU]
8732	311.967170	10.0.0.2	10.0.0.3	TCPCPL	1514	[Bundle TCPCPL Segment][TCP segment of a reassembled ...
8733	311.967384	10.0.0.2	10.0.0.3	TCP	1514	[TCP segment of a reassembled PDU]
8734	311.967551	10.0.0.2	10.0.0.3	TCP	1514	[TCP segment of a reassembled PDU]
8735	311.967791	10.0.0.3	10.0.0.2	TCPCPL	70	TCPL ACK Segment(s)
8736	311.968310	10.0.0.3	10.0.0.2	TCP	66	4556 → 52383 [ACK] Seq=1405 Ack=1495010 Win=139776 L...

Figure 3.23: Wireshark capture of streaming between node B and C

Video transmission with dtntstream application and the use of data mule

- **Objective**

Determining the adequate conditions for video transmission in DTN network with the use of data mule.

- **Procedure**

Verification of dtntstream operation was carried with transfer of video between nodes A (sender) and C (receiver). Node B would assume function of data mule, with scripts in nodes A and C similar to the ones applied in previous test.

- **Results and conclusions**

Video display transfer between two nodes was first experimented (nodes A - sender, node C-receiver), and although node C could record a video transmitted by A with dtntstream application, VLC player could not display the images, due to low capacity of graphic processing with Alix3d3. In the experiment with the use of data mule, reception of bundles in node B was confirmed with daemon messages. Also, the experiment was conducted with recording of the received video stream, which could later be displayed on INESC TEC PC.

3.10 dtnttracepath

A dtn-version of tracepath is available (represented in figure 3.24).

The following commands can be used in `dtm://node1.dtm` with the application to trace `dtm://node1.dtm` and `dtm://node2.dtm`:

```
dtmtracepath dtm://node1.dtm/echo
dtmtracepath dtm://node2.dtm/echo
```

```
root@OpenWrt:~# dtmtracepath dtm://node1.dtm/echo
TRACE TO dtm://node1.dtm/echo
 1: dtm://node1.dtm          delivery    3.61 ms
 2: dtm://node1.dtm/echo    ECHO       5.38 ms

root@OpenWrt:~# dtmtracepath dtm://node2.dtm/echo
TRACE TO dtm://node2.dtm/echo
 1: dtm://node2.dtm/echo    ECHO       7.76 ms
 2: dtm://node2.dtm        delivery    9.69 ms
```

Figure 3.24: Example of the `dtmtracepath` use

3.11 dtmconvert

This application can create bundles in standard output.

Command for this purpose is:

```
cat content | dtmconvert -c -s dtm://host1/echo
```

3.12 dtmtrigger

This application triggers a script upon received bundles.

Command for this effect can be:

```
dtmtrigger triggerReceiver bash.
/triggerScript
```

3.13 dtmtunnel

Creating a `dtm-tunnel` is also possible with this application, through which IP packets can be routed.

For this, the following command can be used:

```
dtmtunnel dtm://host1/tunnel
```

Also included in IBR-DTN implementations is a Distributed Hash Tables (DHT) for naming in BitTorrent file sharing system, and is compatible with other DTN implementations [44].

3.14 Conclusions

These tests could validate the possibility of obtaining high bit rates in data transfer between two nodes with the use of a data mule to carry bundles. Application `dtmsend` was first tested, showing store and forward in transport of bundles with a data mule, from a sender node to a receiver, and

correct number of bundles generated with the chosen fragmentation size. Also, this experiment showed limit of file size in sending archives for adequate daemon operation, when files of size bigger than 50 Mbytes were used, and so tests were made with archives smaller. Regarding the code associated with dtinbox and dtoutbox, the later application includes a cycle for verification of new files, comparing a calculated hash with previous values recorded in a list for previous sent files. If the hash is not found in the list of these elements, the archive will be sent, regardless if present or not in destination folder. New hash value is attributed to files when recorded, even if no change in content occurs. After bundle is sent, no guarantee of reception exists. These characteristics make undesirable occurrence of repetition in archive exchange with no guarantee of file reception, and overall file synchronization is not possible.

For dtstream application, tests made reveal good functioning in radio transmission, although reproduction was not possible, still audio recording was achieved. Video transmission was also possible, although for small recording time of five to six minutes.

File sharing between AUV or Remotely Operated Vehicles - ROV can be useful for synchronization of content between elements in situations of transfer sensor data or video samples in a practical and automatic process, without need of complex solutions for archive transfer. The next stage of this dissertation consisted in development and evaluation of an application in underwater environment.

Chapter 4

Synchronization application

The applications `dtnoutbox` and `dtinbox` are not sufficient for file synchronization purpose, as showed in the functional tests. These elements do not guarantee bundle delivery to destiny, once `dtnoutbox` sends bundles only when hash value of file is not present in a sent hash list without confirmation of reception or repetition, in case file is not received, and altogether unnecessary file transfer may occur- file is sent even if already present in destiny folder. Analyzing `dtnoutbox` code, the decision of file sending is based on the presence of archive reference in the list of files to send. In case this list is not empty, files contained are grouped in the next bundle to send. However, the files to send list is filled with archive reference according to a condition that only includes elements with hash not present in sent hashes list: if a file changes hash value, transfer will occur even if already present in destination folder, and if a file does not change hash value, transfer will not occur even if the destiny folder did not receive it in a previous transfer.

Hash value changes with last modification date of file, even if content remains unchanged, explaining the situation when new archive detected in `dtnoutbox` is automatically sent to destination folder each time it reaches reception, due to new hash value associated.

To avoid this situation, correcting unnecessary data transfer and introducing guarantee of file delivery, an application was developed in order to create a list with file names and md5 checksum value: the code creates a vector with elements present during cycle and compares with files detected in list received from the other pair and only files with different combination of name and checksum value are exchanged. Diagram explaining the algorithm is represented in figure 4.1.

Bundles are sent only with files not common to nodes, in case the list of local host has files not present in node pair, until a list is received with reference to files in remote pair confirming reception, in a way to guarantee delivery. When both list have same archives, transfer stops and only the lists of files in elements are exchanged.

Application starts with creation of adequate TCP stream, initialization of client and connection to the server. Next, a cycle starts clearing string vectors used for data storing. Files present in chosen folder are then verified, and reception of the list of files present in remote host is also confirmed and recorded.

Each file combination of name and md5 checksum value is added to a list of current files, with

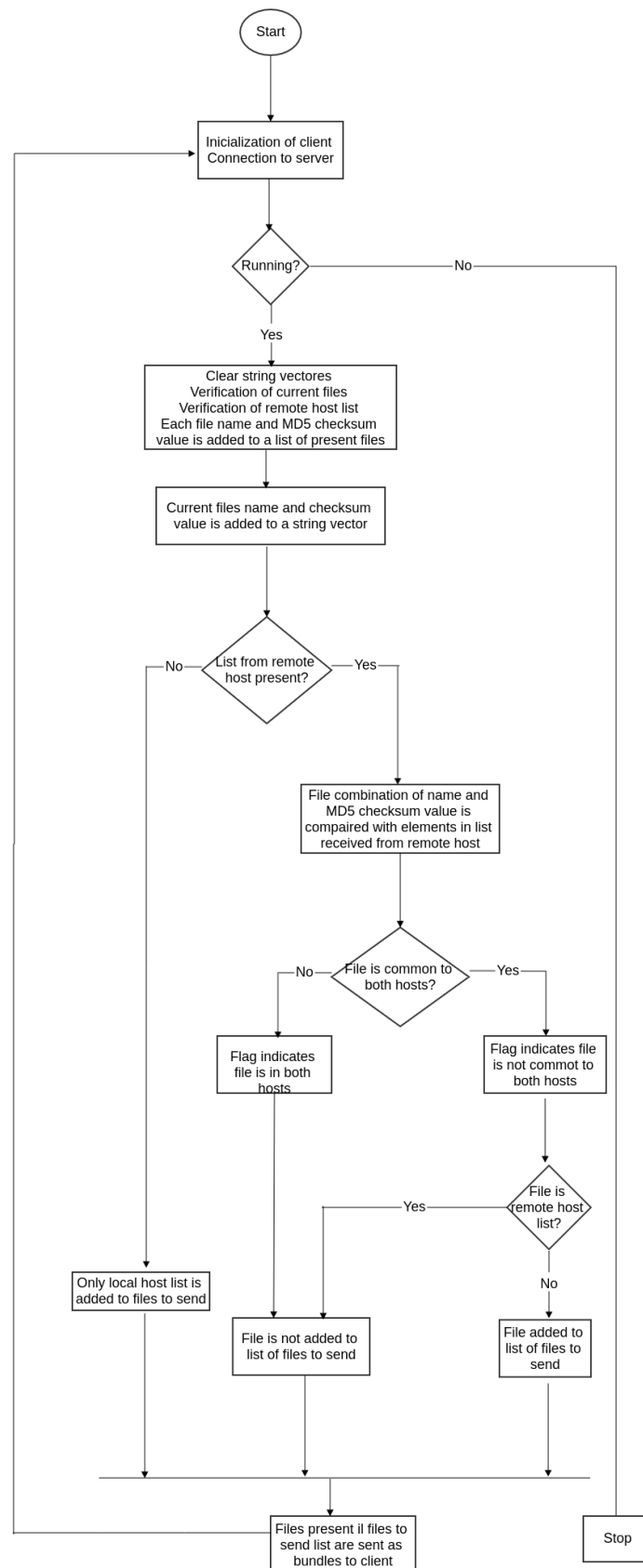


Figure 4.1: Algorithm used in file sharing application

name attributed by the destination of the folder. For example, if the application is configured with destination *dtn://node2.dtn/inboxReceiver*, the name of the file list will be *node2.dtn* - this allows each list to have a different name. In this step, a condition imposes the list name of files received from pair is not written in the list of files, in order to avoid not obtaining the list of the pair; also the name of the list is not written.

Current files name and checksum value is recorded in a string vector in the following stage, and the list received from remote node is opened, in case the presence of the list was detected previously. Each file data is compared with information placed in vector of current archives: if equal elements are detected, the name and md5 checksum value associated is stored in a different string vector, with common elements. Observed files are again updated and each file presence in the common files vector previously created is verified: in case the element is present, meaning already in remote host destination folder, the archive will not be added to the list of files to send. Finally, the archives included in files to send list are added to bundles, and destination EID of elements is established. Bundles are sent to client, and flushed before cycle restarts.

Archives created with the list of files in each node present extension ".dtn" and present format of common text files. Each line in these lists contains the following representation for each file:

<date of last modification> <MD5 checksum value> <filename> The date of last modification and name of archives can be obtained from functions included in library available in IBR-DTN, as for MD5 checksum value is obtained by a function created by research of this algorithm. Each line can be read by the receiving node and data from files included in foreign host target folder is compared to local host information.

List of files must take into account some precautions for correct file exchange: local host can never transmit list received from foreign node, to avoid interfering with list creation, hence conditions are imposed to avoid sending files with extension "dtn" and name different from local host list. Also, before receiving a list of files from the remote node, local host only sends his own list, to avoid unnecessary file transfer. The lists are named according to application destiny, imposing different names, to avoid confusion between archives. The sequence diagram associated with this application is presented in figure 4.2

Node B approaches node A in stage I), and file list is delivered from A to B. Travel of node B then occurs from A to C, and delivery of the file list received from A occurs in stage II). Node C then verifies files in this list, and in case the folder defined in the application includes archives not present in target folder in node A, bundles with files not present in node A are delivered to node B. Once the elements are received, node B travels again to node A and delivers received archives in stage III), together with file list of node C. Now, node A can verify files present in list received from node C target folder, and sends files that are not included. These archives are delivered together with list of files from node A to node B. Data mule travels again from node A to C, delivering missing files and both folders reach equal content in stage IV).

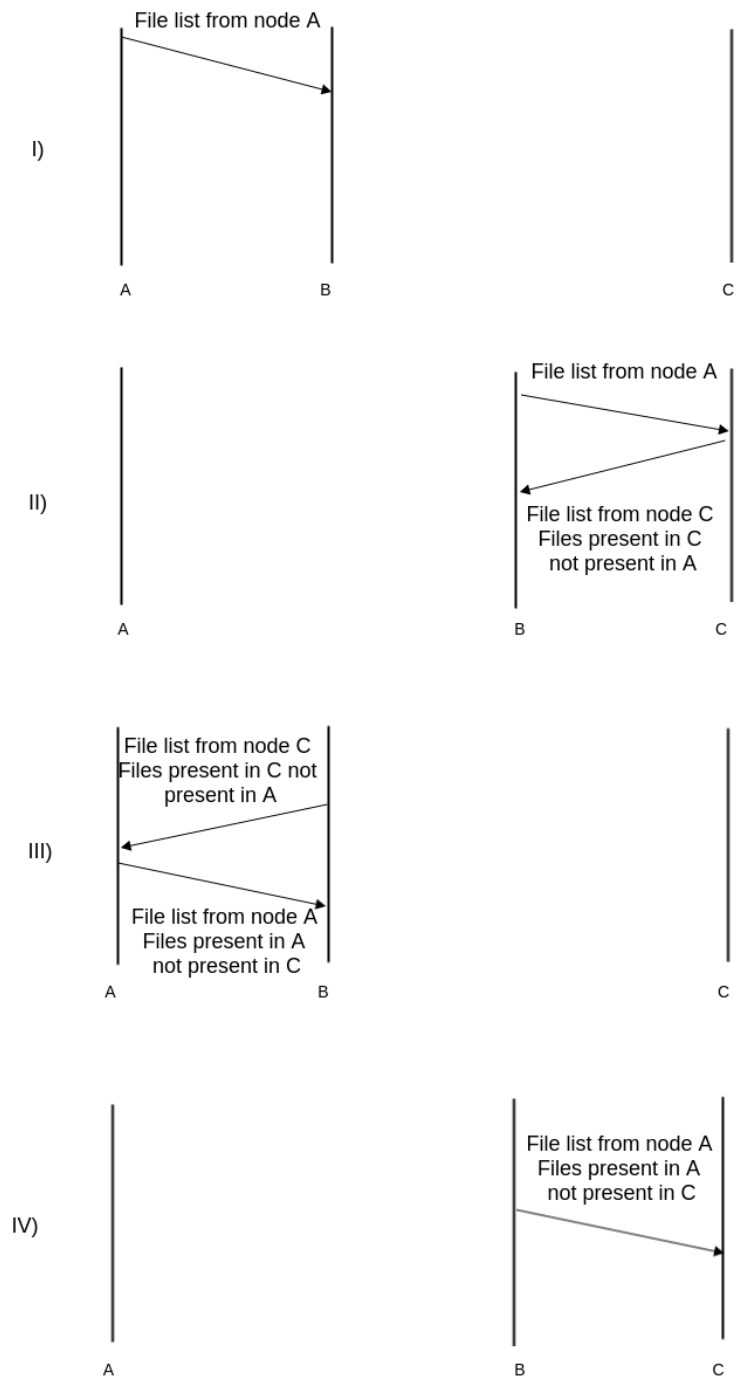


Figure 4.2: Sequence diagram file synchronization application

Application use of IBR-DTN libraries can access API functions, which in turn interact with daemon for sending bundles, as represented in figure 4.3.

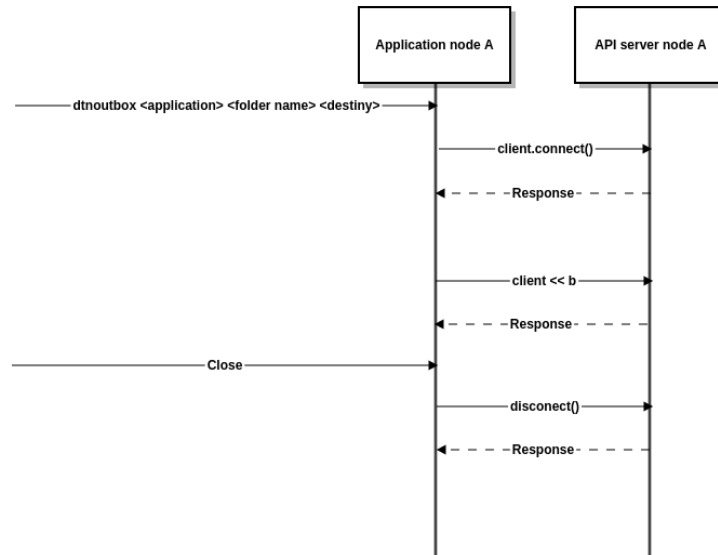


Figure 4.3: Interaction between application and API

This application was developed in C++ with the use of libraries included in IBR-DTN platform. Source code for IBR-DTN applications was obtained from github repository [45], and necessary libraries for compiling were also installed. Code used in this application is presented in appendix A.

Chapter 5

Experimental planning and testbed design

This chapter presents the description of the scenario considered and the experimental evaluation taken place in a freshwater tank at INESC TEC. Also, a study on the movement conditions for favorable binary rate transfer is presented.

5.1 Experimental evaluation

For the proposed scenario, we tested data transfer between source and destination nodes, using an underwater data mule to deliver data from source to destination, as represented in figure 5.1. These elements were tested in a laboratory environment with a fresh water tank at INESC TEC Robotics facilities, presenting dimensions of 6 meters long, 5 meters wide and 2 meters depth.

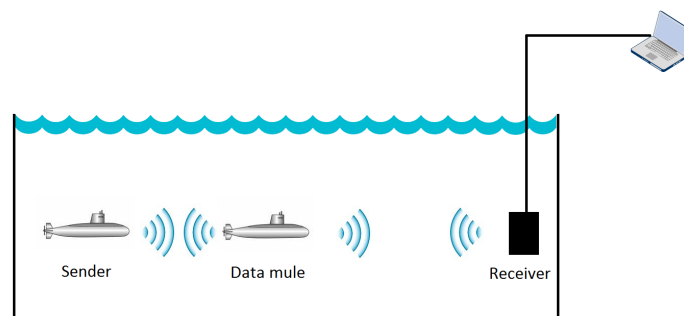


Figure 5.1: Experimental scenario considered for the application test in underwater environment

Data rate transfer in our system can be predicted, assuming ideal conditions, for different velocities of carriers and 100 Mbit/s file transfer rate between close nodes. Proximity between drones will also be evaluated in order to minimize their traveled distance.

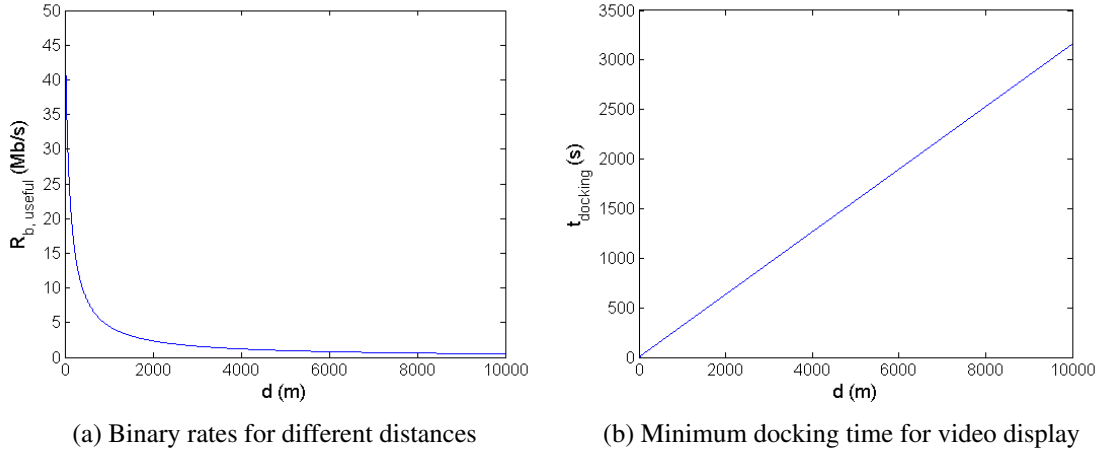


Figure 5.2: Simulation of useful bitrate and minimal docking time variations with distance

Useful throughput $R_{b,useful}$ can be determined considering transfer throughput between elements of R_b in Mbit/s, velocity v in m/s, docking time $t_{docking}$ in s and a distance path d (in meters) with the following expression:

$$R_{b,useful} = \frac{R_b \times t_{docking}}{2 \times t_{docking} + 2 \times \frac{d}{v}} \quad (\text{Mbit/s}) \quad (5.1)$$

Considering $R_b = 100$ Mbit/s, $t_{docking} = 100$ s, and $v = 1$ m/s, $R_{b,useful}$ will be:

$$R_{b,useful} = \frac{10^4}{200 + 2 \times d} \quad (\text{Mbit/s}) \quad (5.2)$$

The graphical representation of $R_{b,useful}$ versus distance in figure 5.2a shows that for distances as high as 10000 m, useful throughput is 0.5 Mbit/s, which is still superior to most acoustic rates available (31.2 kbits/s).

Minimal docking time can be obtained by rearranging 5.5:

$$t_{docking} = \frac{2 \times \frac{d}{v}}{\frac{R_b}{R_{useful}} - 2} \quad (\text{s}) \quad (5.3)$$

For an effective video streaming, considering video H.264/AVC High Profile 12 Mbps encoding [46], minimum docking time can be determined, considering transfer throughput between elements of 100 Mbit/s and velocity of 1 m/s for different transmission distances in the following expression:

$$t_{docking} = \frac{24 \times d}{76}$$

The graphical representation of docking time versus distance is represented in figure 5.2b, showing linear behavior.

Total delay of video transmission can be determined with the following expression:

$$\text{Delay} = 2 \times d/v + 2 \times t_{docking} \quad (\text{s}) \quad (5.4)$$

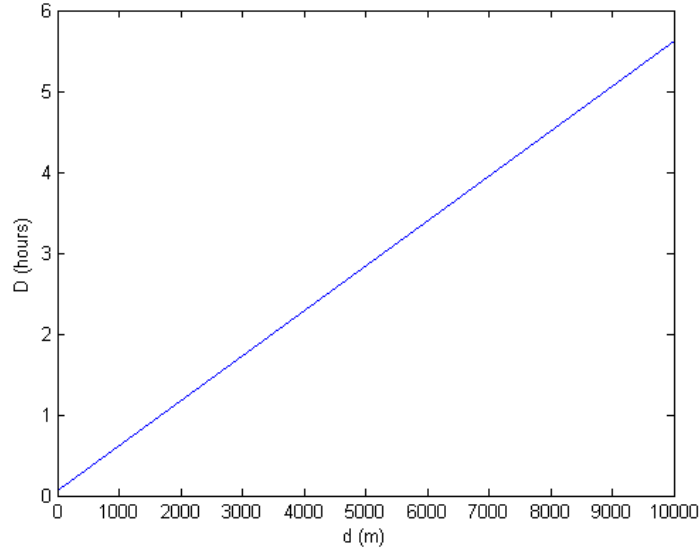


Figure 5.3: Total delay versus distance

The graphical representation of delay versus distance, considering a docking time of 100 s is represented in figure 5.3, showing linear behavior.

5.2 Study of movement conditions for favorable binary rate transfer

In order to determine favorable conditions for data transfer in our system, equation 5.5 can be derived in order to both docking time (equation 5.6) and velocity (equation 5.7).

$$R_{b,useful} = \frac{R_b \times t_{docking}}{2 \times t_{docking} + 2 \times \frac{d}{v}} \quad (5.5)$$

$$\frac{dR_{b,useful}}{dt_{docking}} = \frac{\frac{d}{v} \times R_b}{2 \times \left(t_{docking} + \frac{d}{v} \right)^2} \quad (5.6)$$

$$\frac{dR_{b,useful}}{dv} = \frac{\frac{d}{v^2} \times R_b \times t_{docking}}{2 \times \left(t_{docking} + \frac{d}{v} \right)^2} \quad (5.7)$$

Comparing equations 5.6 and 5.7, derivative functions show that in similar conditions, raising docking time might raise useful bitrate faster than increasing velocity. This effect leads, however, to higher delay of data reception.

Graphical representations of figure 5.4 show useful bitrate dependency with distance for different docking times (100, 200, 300 and 400 s).

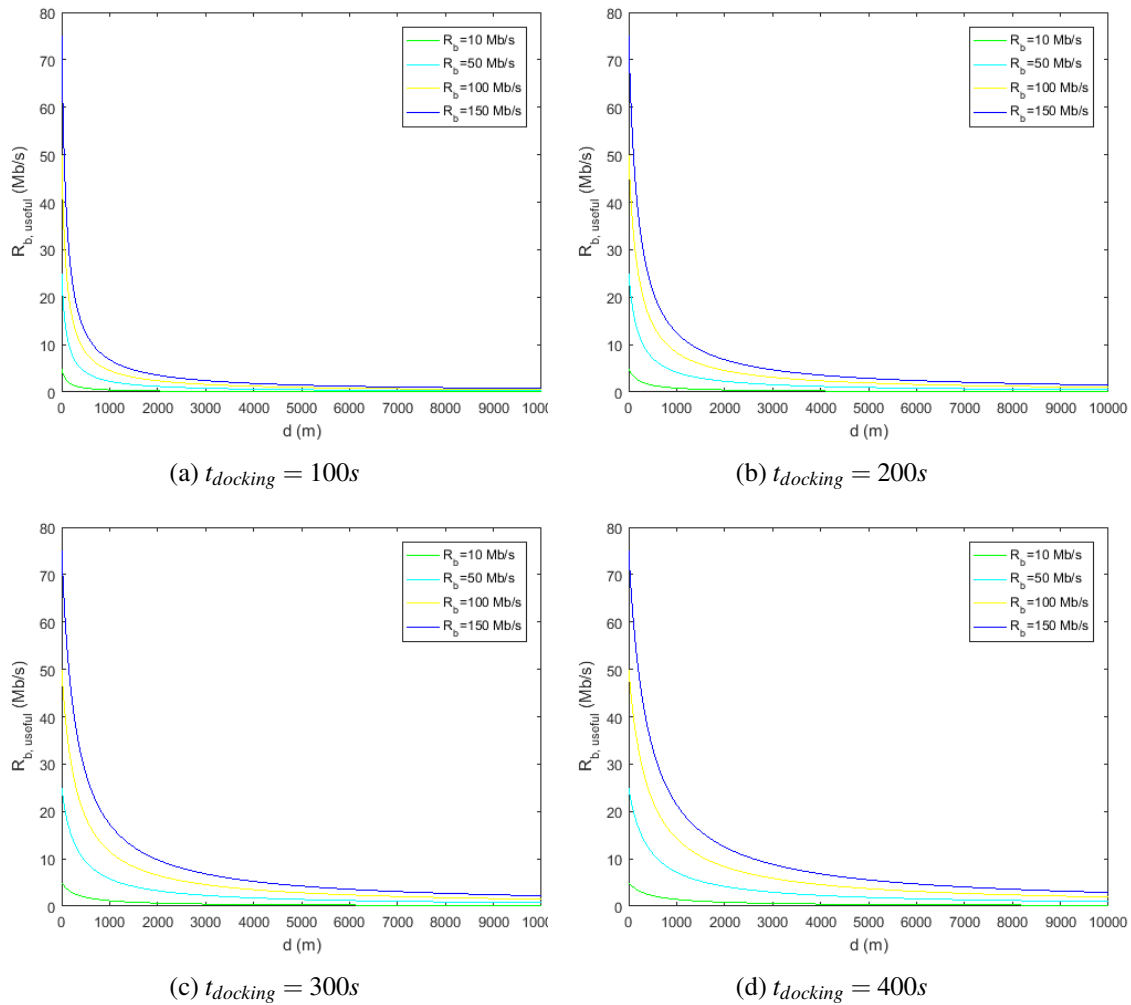


Figure 5.4: Useful bitrates vs distance (path velocity 1 m/s)

Figure 5.5 shows representation of bitrates dependency with velocity, for different docking times (100, 200, 300 and 400 s). Graphical representations show a strong decay in equivalent bit rate with distance path, however raising docking time and bit rate between nodes may decrease this effect. Also, an increase in velocity in this movement raises equivalent bit rate. Data correction can be applied to our scenario, with an Automatic Repeat reQuest (ARQ) method. Go-Back-N can present a more suitable result for networks limited by delay and restrictions of connectivity, by allowing transmission of new packets before acknowledgment of older ones.

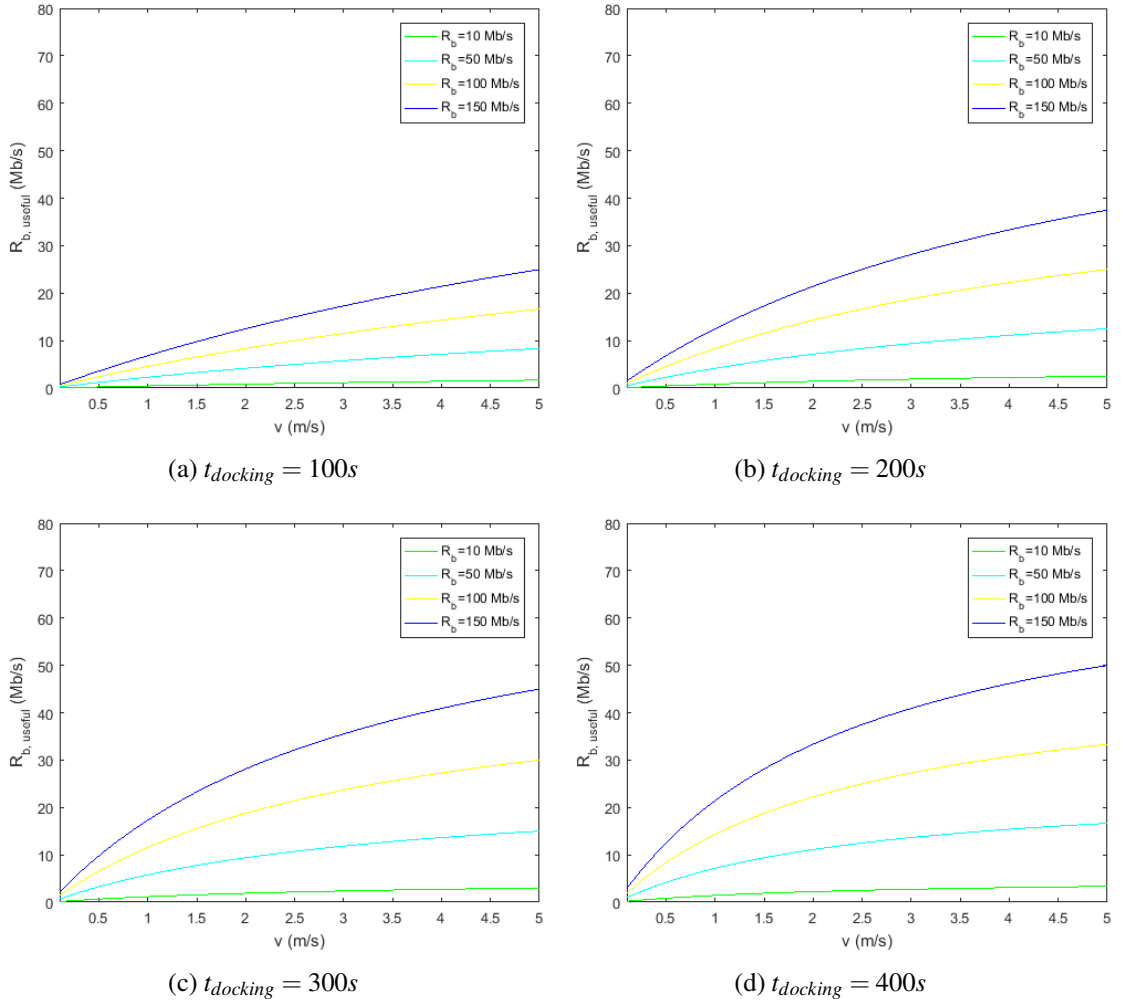


Figure 5.5: Useful bitrates vs velocity (path distance 1000 m)

Go-back-N

Assuming window size of sender packets $W > 2a + 1$, efficiency S of Go-back-N can be given by [47]:

$$S_{Go-Back-N} = \frac{1 - FER}{1 + 2 \cdot a \cdot FER} \quad (5.8)$$

The factor a can be determined with the following expression:

$$a = \frac{\text{Propagation time}}{\text{Transmission time}} \quad (5.9)$$

where propagation time can be determined simply dividing propagation distance d_p by velocity of propagation v_p , and transmission time is determined by dividing frame length L by bit rate R_b . If these parameters are substituted in equation 5.9, we obtain:

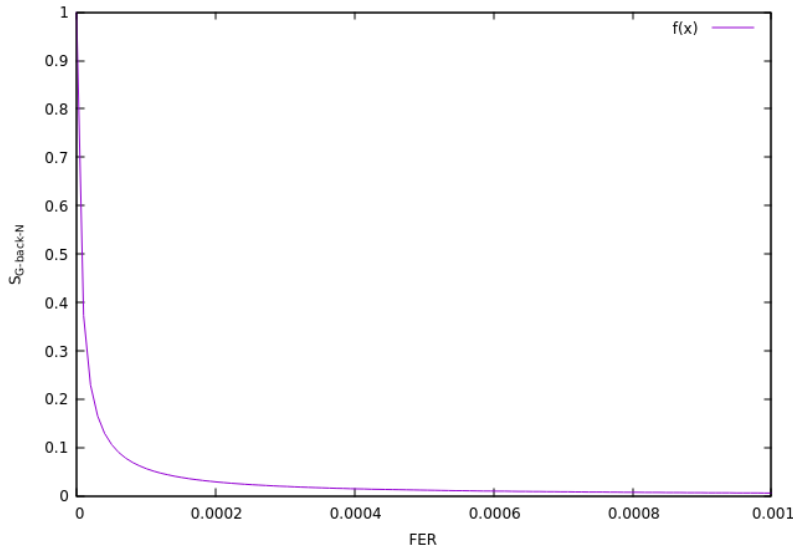


Figure 5.6: Efficiency for Go-back-N estimated for different values of FER

$$a = \frac{d_p \times R_b}{v_p \times L} \quad (5.10)$$

For a path distance, $d_p = 1000 \text{ m}$, $v_p = 1 \text{ m/s}$ and $L = 1500 \text{ bytes}$, a can be estimated by:

$$a = \frac{1000 \times 10^6}{1 \times 1500 \times 8} = 83333.3$$

For this value of a , efficiency of Go-back-N can be graphically plotted, for different values of Frame Error Ratio - FER, as represented in 5.6. Plot shows strong decay of efficiency with FER, however for values inferior to 2×10^{-6} efficiency is superior to 70 %.

5.2.1 Metrics to be considered in the evaluation

The implemented application will be evaluated in terms of different metrics, such as:

- Average data rate: maximum achievable throughput from the sender node to the receiver;
- Delay: time required for a packet to reach the destination, after transmission.
Conclusions from these observations will help in obtaining traffic optimization.

5.2.2 Analysis of results

The results obtained in this thesis will be compared with a similar solution using data rates and delays associated with acoustic transmission. All conclusions and further research will be summarized in the dissertation and an article concerning the achievements will be written.

Chapter 6

Performance results

After development of proposed solution, the applications was tested in an underwater tank at INESC TEC. The three Alix3d3 nodes used were connected to Ethernet switch, as in previous tests with address scheme 192.168.1.x. Elements were also synchronized with NTP according to the time imposed by INESC TEC PC, as previously.

The three elements were placed in different acrylic cylinders (represented in figure 6.1), and closed with two end-caps, each with two o-rings for protection and avoiding water inside.

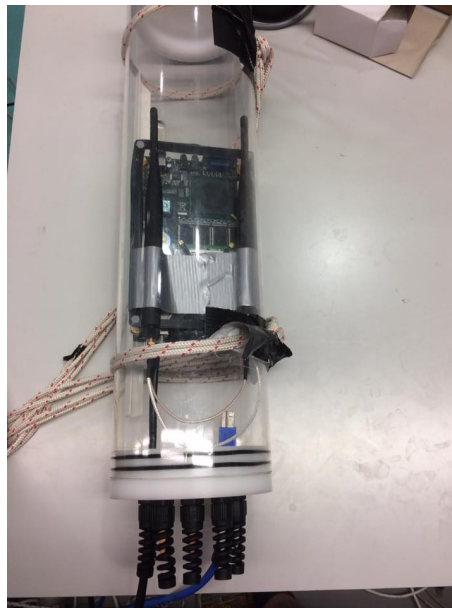


Figure 6.1: Cylinder used in experiments

Each Alix3d3 was fixed in acrylic boards and connected with two antennas, maintaining distance between these emitters greater than $\frac{\lambda}{2} = \frac{3 \times 10^8}{2 \times 2.4 \times 10^9} \approx 7$ cm in order to avoid destructive interference. The movement of data-mule cylinder was induced in experiments with ropes suspending the cylinders. Experimentation testbed is represented in figure 6.2. Each cylinder was powered using PoE.

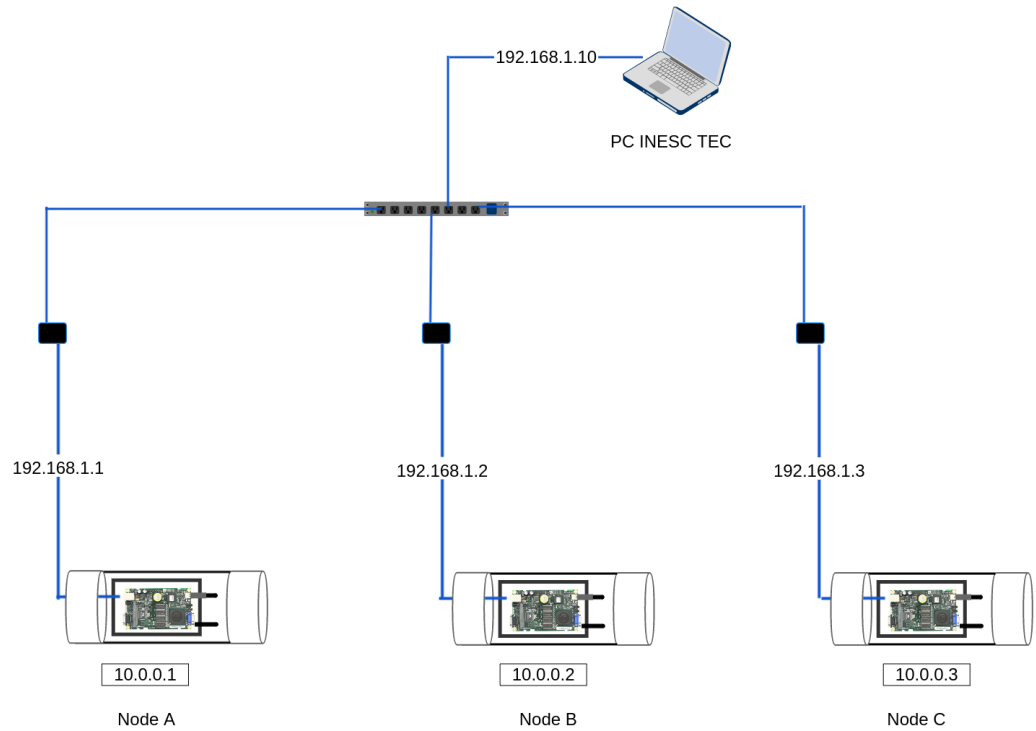


Figure 6.2: Testbed for data mule experiments between sender and receiver nodes



Figure 6.3: Testbed for data mule experiments between sender and receiver nodes in INESC TEC

The real scenario of the tests can be seen in figure 6.3. As defined in previous functional tests, the Wi-Fi network used was named "teste", in ad-hoc mode. The channel selected was 6 ($f=2.437$

GHz) and bandwidth 20 MHz (mode IEEE 802.11 n). This mode was chosen due to possibility of operating in 2.4 GHz or 5 GHz and also allowing to boost data rate with the use of MIMO. Power output in wireless board was 23 dBm. Address attributed to each element was defined in the same scheme 10.0.0.x as previously.

The EID nodes of the elements were: node A dtn://node1.dtn; node B, dtn://node2.dtn and node C, dtn://node3.dtn. Routing configuration selected in IBR-DTN was set to flooding.

6.1 Single-hop performance comparison of dtnsend and iperf

This test aimed at determining the transfer time of a single file with IBR-DTN application dtnsend, and comparing the value with the result obtained with iperf, i.e., without the DTN protocol stack. To perform this test, a file of 49.7 Mbytes was sent from node A to node B with dtnsend application. The procedure was made together with tcpdump capture in both nodes. Experiment was repeated ten times.

The application iperf was used with node B as server and node A as client. The file of 49.7 Mbytes was sent and average flow rate was measured in server. The experiment was also repeated ten times.

The results obtained with iperf application are presented in table 6.1.

Table 6.1: Time required for transfer of bundles between nodes A and B

Experiment	Average bit rate (Mbit/s)
1	25.3
2	36
3	37.1
4	36.9
5	36.2
6	25.5
7	27.3
8	36.6
9	37.2
10	36.5

The average flow rate of values obtained in iperf simulations here is 34.46 Mbits/s with standard deviation of 5.16 Mbit/s, which corresponds to a transfer of 49.7 Mbytes of $\frac{49.7 \times 8}{34.46} = 11.53$ s. Time associated with transferring bundles between nodes A and B is presented in table 6.2.

The experiments regarding transfer of 49.7 Mbytes file with dtnsend showed an average value of 15.7 s for transmission with standard deviation of 4.24 s, equivalent to a bit rate of 26.7 Mbit/s, with standard deviation of 5.94 Mbit/s. Transfer shows slightly lower value of transfer time for IBR-DTN application, as expected. Slower performance with IBR-DTN daemon relative to raw TCP results is associated with the fact of Bundle protocol running on top of TCP, so overhead associated with Bundles and user-space processing is added to overhead in common TCP.

Table 6.2: Time required for transfer of bundles between nodes A and B

Experiment	Time seconds
1	24
2	18
3	12
4	13
5	13
6	15
7	14
8	14
9	12
10	22

Difference between these values can also be quantified with percentage deviation, and the following value is obtained:

$$\frac{|34.46 - 26.7| \times 100}{26.7} \approx 29\% \quad (6.1)$$

6.2 Store and forward with the use of data mule in underwater environment

This experiment was made in order to obtain the value of transfer time for a file of 49.7 Mbytes between two nodes, with the use of a data mule. Also, this procedure intended to confirm store and forward mechanism and the fragmentation of bundles. The three cylinders containing nodes A, B and C were placed far apart, avoiding connectivity between nodes. Node B was brought close to node A, as in this node a file was sent with dtnsend application. Next, node B was again placed apart from node A and C, for about three minutes, and then it was placed next to node C, for file reception. Time for transfer of bundles between nodes is expressed in tables 6.3 and 6.4.

Table 6.3: Time required for transfer of 49.7 Mbytes file between nodes A and B

Experiment	Time (seconds)
1	19
2	18.5
3	16.6
4	15.8
5	21
6	19
7	16.95
8	18.6
9	19.6
10	18.6

Average transfer time between nodes A and B was 18.4 s with standard deviation of 1.53 s and transfer time between nodes B and C was 16.71 s with standard deviation of 7.6 s. The file transfer

Table 6.4: Time required for transfer of 49.7 Mbytes file between nodes B and C

Experiment	Time (seconds)
1	13
2	14.1
3	12.7
4	11.6
5	37.9
6	16.8
7	16
8	15
9	15
10	15

mechanism with DTN daemon was verified in underwater medium, as well as store and forward, by node B. This element carried data between nodes A and C, assuming the function of data mule, as tested in previous experimentations. For trip time of 3 minutes for data mule between nodes A and C, total bit rate associated with this transmission can be determined in following calculation:

$$R_b = \frac{\text{Total size of file transferred}}{2 \times T_{trip} + 2 \times T_{transfer}} = \frac{49.7}{2 \times 180 + 18.4 + 16.71} \times 8 = 1.006 \text{ Mbit/s}$$

Estimated value in this transfer is clearly superior to bit rates available when using acoustic communication, 32 times greater than maximum value presented for bit rate in acoustic modem (31.2 kbit/s). For a velocity of 0.5 and 1 m/s, communication at corresponding distances of 90 and 180 m respectively can present these gains.

Extrapolation bit rates for different values of distances is represented in plots of figure 6.4. Assuming a velocity value $v=1 \text{ m/s}$ $T_{trip} = d/v$ leads to:

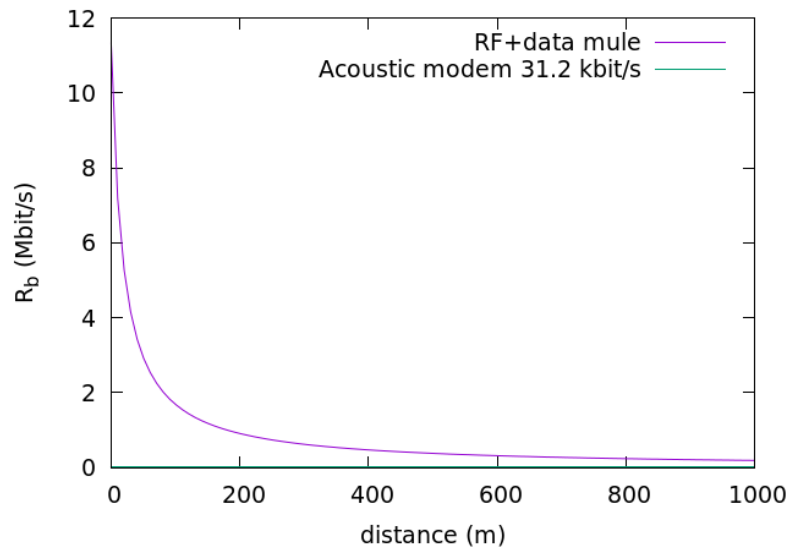


Figure 6.4: Bit rate extrapolation for different distances with store and forward experimentation results

$$R_b = \frac{\text{Total size of file transferred}}{2 \times T_{\text{trip}} + 2 \times T_{\text{transfer}}} = \frac{49.7}{2 \times d + 35.11} \times 8 \quad (6.2)$$

Estimation of bit rate, considering movement of data mule between sender and receiver with velocity 1 m/s, can be estimated using these results for different travel distances. Result obtained for distances 1000, 3000, 4000 and 5000 m are:

$$R_b = \frac{\text{Total size of file transferred}}{2 \times T_{\text{trip}} + 2 \times T_{\text{transfer}}} = \frac{49.7}{2 \times 1000 + 18.4 + 16.71} \times 8 = 195.37 \text{ kbit/s} \quad (6.3)$$

This value reveals throughput 6 times higher than the fastest acoustic modem considered, in longest available range.

$$R_b = \frac{\text{Total size of file transferred}}{2 \times T_{\text{trip}} + 2 \times T_{\text{transfer}}} = \frac{49.7}{2 \times 3000 + 18.4 + 16.71} \times 8 = 65.9 \text{ kbit/s} \quad (6.4)$$

$$R_b = \frac{\text{Total size of file transferred}}{2 \times T_{\text{trip}} + 2 \times T_{\text{transfer}}} = \frac{49.7}{2 \times 4000 + 18.4 + 16.71} \times 8 = 49.5 \text{ kbit/s} \quad (6.5)$$

$$R_b = \frac{\text{Total size of file transferred}}{2 \times T_{\text{trip}} + 2 \times T_{\text{transfer}}} = \frac{49.7}{2 \times 5000 + 18.4 + 16.71} \times 8 = 39.6 \text{ kbit/s} \quad (6.6)$$

Results in equation 6.6 show bit rate eight times higher than modem with highest range (5 kbit/s).

6.3 Unidirectional synchronization with proposed application

This experiment was performed in order to obtain the confirmation of file sharing capability with the proposed application. The test also allowed to determine transfer time of four files with sharing folder application.

Procedure was started by creating two folders: in node A, folder TesteA, and in node C folder TesteC. Files of 4.8, 9.6, 19.3 and 49.7 Mbytes were placed in folder TesteA. The application developed was applied in both folder, together with dtinibox. Cylinders were placed apart in the tank, and Node B was brought close to node A. After about 40 s, node B was again placed apart from both nodes, during about 3 minutes, and then, it was brought close to node C. This experiment was repeated ten times. Results obtained in this experiment are presented in tables 6.5 and 6.6.

Average transfer time between these two nodes is 37.51 s, with standard deviation of 11.1 s.

For transfer of files between these two nodes, average time is 30.5 s with standard deviation of 9.5 s. Considering trip time of 3 minutes, bit rate associated to this transmission can be determined

Table 6.5: Time required for transfer of bundles between nodes A and B

Experiment	Time (seconds)
1	48.1
2	44.8
3	55.3
4	52
5	29.2
6	34
7	33.7
8	22
9	31
10	25

Table 6.6: Time required for transfer of bundles between nodes B and C

Experiment	Time (seconds)
1	30
2	39.95
3	38.5
4	39.4
5	30.2
6	29.8
7	23
8	27
9	21.15
10	26

in the following calculation:

$$R_b = \frac{\text{Total file size}}{\text{Total trip time}} = \frac{83.4}{2 \times 180 + 37.51 + 30.5} \times 8 = 1.56 \text{ Mbit/s} \quad (6.7)$$

Also, this experiment shows the possibility of obtaining high gains, relative to acoustic transmission, up to 50 times greater than maximum value presented for bit rate in acoustic modem (31.12 kbit/s) This experiment demonstrated file sharing with created application, without repetition of file delivery, after the same content is present in both folders. The file identity was verified with md5 checksum value in all experiments.

Bit rate extrapolation for different values of distances is represented in plots of figure 6.5. Graphical plot was created, assuming a velocity of 1 m/s corresponding to a trip time $T_{trip} = d/v$

$$R_b = \frac{\text{Total file size}}{\text{Total trip time}} = \frac{83.4}{2 \times d + 68.01} \times 8 \quad (6.8)$$

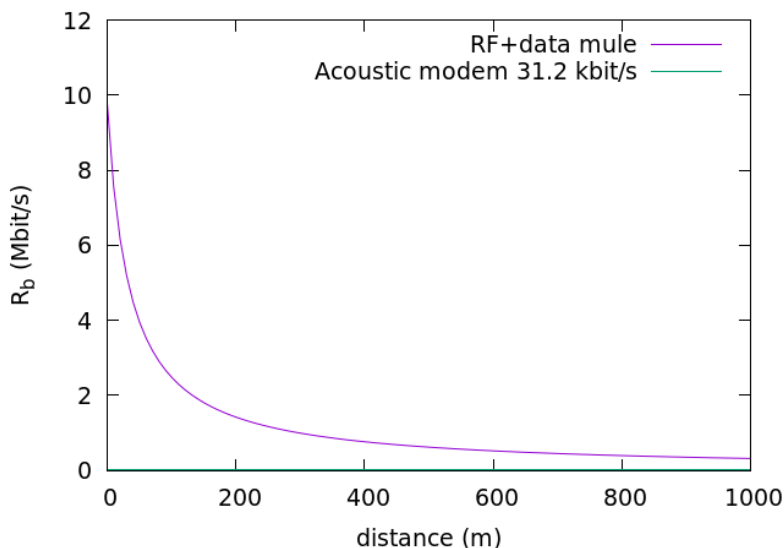


Figure 6.5: Bit rate extrapolation for different distances with unidirectional synchronization results

6.4 Bidirectional synchronization of folders with proposed application

The following experimental procedure was conducted to confirm archives sharing with developed application, starting with different contents in selected folders. This experiment was started with creation of two folders: TesteA, with file of 15 Mbytes in node A and TesteC with files of 19.3 in node C. The three nodes were placed apart in FEUP INESC TEC tank, and then node B was brought close to node A for about 20 s. Node B was again brought apart from both nodes for about 3 minutes. Finally node B was brought close to node C. Procedure was repeated until both folders TesteA and TesteC presented equal content. The experiment was repeated ten times.

Time required in each experiment for synchronization of folders is presented in table 6.7.

Table 6.7: Time required for both folders present the same content

Experiment	Time (minutes)
1	12.05
2	11.85
3	7.45
4	8.88
5	8.38
6	9.28
8	8.75
9	8.18
10	8.68

Average time for this file transfer is 8 and 49 s, with standard deviation of 90 s. Bit rate can be

estimated for this transmission with the following calculation:

$$R_b = \frac{15.0 + 19.3}{60 \times 8.82} \times 8 = 519 \text{ kbit/s} \quad (6.9)$$

Global bit rate in this situation can be approximated by the following expression:

$$R_b = \frac{\text{file size}}{4 \times \text{docking time} + 3 \times \text{trip time}} \quad (6.10)$$

In this case, bit rate can also be plotted against trip time, assuming a velocity of 1 m/s. Trip time is again given by d/v , leading to a global expression of:

$$R_b = \frac{34.4 \times 8}{80 + 3 \times d} \quad (6.11)$$

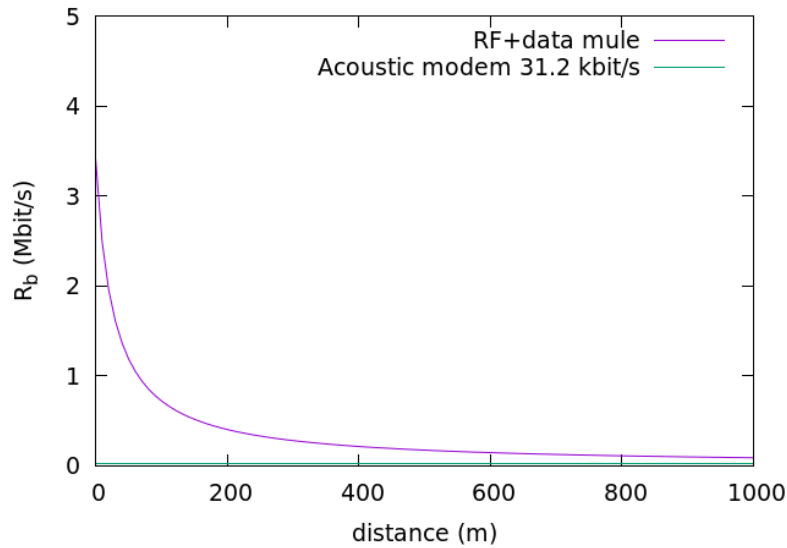


Figure 6.6: Bit rate extrapolation for different distances with unidirectional synchronization results

Graphical representation in figure 6.6 was created with equation 6.11. Comparing with acoustic transmission, this result still presents superior bit rate, in spite of longer time needed for bidirectional synchronization. Estimated bit rate is 17 times greater than maximum value presented for bit rate in acoustic modem (31.2 kbit/s). The results obtained in this experiment allowed to acknowledge that until both folders presented equal content, files detected missing in nodes were periodically sent, in a way to guarantee archive delivery, prevent bundle loss or avoid missed reception due to lifetime expiration. Also, files were confirmed with md5 checksum verification after each experiment.

6.5 Update of files in remote host when file is changed in local host

This experiment intended to determine if the application allowed file update in a remote folder, when changes in local node are made in that archive. First step in this experiment consisted in the creation of folders TesteA in node A, and TesteC in node C. Next, in node A, a cron job was created in order to copy the file syslog to folder TesteA every 10 minutes. Then, daemon was started in three nodes, with node B next to node A for about 20 s. Node B would then be brought far from both nodes for about 3 minutes. Finally, node B was brought close to node C for about 20 s and md5 checksum of files in folder TesteC was observed. This experiment showed, from changes in md5 checksum value of file syslog, possibility of updating files in remote host, from changes in archives made in local node.

Chapter 7

Conclusions and future work

Underwater communications depends on technologies based on optical, acoustic and radiofrequency signals. Optical transmission relies on the alignment between source and receiver and clear optical path between communicating nodes, as for acoustic modems present high transmission range, but low bit rates of data transfer. Radiofrequency allows high throughput in underwater medium, although transmission range is small. Dealing with underwater periods of loss of connectivity brought interest to include a Delay Tolerant Network solution, with suitable applications for file transfer and allow store and forward of information by the data mule between sender and receiver nodes.

In this thesis, we propose the use of small drone acting as data mules for increasing the communication range between sender producing a message and receiver of the message combined with high bit rates available in radiofrequency transmission, in order to obtain higher efficiency compared to acoustic modems.

In order to address the limitations of the IBR-DTN applications set, developed for DTN networks, we propose a synchronization application that is able to perform a bidirectional synchronization between an AUV and an underwater access point using a data mule to transport the data.

Results obtained in this thesis allowed to show possibility of improving bit rate transmission gain with the use of drones as data mules in underwater communications. Also, experimentation showed higher gains with respect to acoustic transmission when data mule movement was used for file delivery. For this effect, IBR-DTN was used as the DTN software platform, and made possible to obtain results for transmission in underwater communication, in situations with loss of connectivity. We also demonstrated achievement of longer distance in transmission with use of data mule movement, and developed a new application for synchronization of files with DTN platform, assuring delivery to receiver.

The study of IBR-DTN applications brought interest in improving functionality with file sharing purpose, and so an application was developed for sharing folders with DTN daemon synchronization. The proposed code also guarantees file delivery with verification in equality inside a list of files with data concerning all archives inside target folders, avoids unnecessary file exchange, and allows file update when archives are changed.

Results obtained outside water, allow to estimate a global bit rate of 2.78 Mbit/s for a simulated distance of 20 m. In the following tests, in underwater medium, bit rates of 1.006 Mbit/s were estimated for distances of 180 m, and global bit rate of 1.56 Mbit/s and 519 kbits/s were obtained in synchronization with the developed application. Obtained values are respectively 32, 50 and 17 times higher than acoustic modem transmission. For higher distances, such as 4000 m, bit rate 49.5 kbit/s was also estimated with proposed results, suggesting interesting gains of communication even at these distances.

The hardware chosen also was proved to be adequate in achieving the proposed objectives: alix3d3 and Routerboard R52n-M showed good performance in testbed conditions. Also, including hardware in acrylic cylinders presented excellent solution for drone movement emulation, avoiding water incomes and eventual mechanical collision effects on hardware - it was possible, with these elements, to obtain results without use of actual AUV.

IBR-DTN showed good performance in transferring files, and we verified previous conclusions regarding the use of this implementation in low capacity elements. The code could be easily compiled in Debian 8.0 for development. Research showed this platform can benefit with some code improvements for extended purpose. Development of original application for DTN might be useful for understanding IBR-DTN daemon source code, and also motivate development of new and versatile software for different scenarios. However, due to the limited RAM available on alix3d3 and the way IBR-DTN creates the bundles, the maximum file size to be transferred was limited to about 50 Mbytes. If suitable conditions are established and extensive files are transferred, bit rates can be even more expanded and gains even more attractive can be achieved for using data mules in combination with RF communications to transport large amounts of data underwater.

Future work

The proposed solution has brought interesting results in underwater communication, although improvements can be made with DTN implementations for these transmissions on future investigation.

- **Transferring large archives with IBR-DTN and the use of data mule:** by changing the way the bundles are created at IBR-DTN implementation, we could overcome the limitation of the file transfer to the available RAM on the system, improving the overall bitrate when using data mule.
- **Compatibility of IBR-DTN with other DTN platforms:** The effect of combining IBR-DTN with another software, as DTN2 or ION in underwater communications can be further studied, and performance can be evaluated. Functional characteristics of simultaneous operation with IBR-DTN and other DTN daemon can be interesting for evaluating performance and correct eventual disadvantages for better knowledge of DTN implementation.
- **Using extensive number of data mules for transferring bundles** The effect of using higher number of data mules in tests similar to those presented in these thesis can be interesting for obtaining higher efficiency of data exchange.

- **Effect of routing schemes on DTN underwater communication, with use of data mules**
Configuration of DTN routing patterns can also be investigated, with intervention of data mule for store and forward bundles. Bit rate obtained in similar transmissions with different routing options can be compared, and results may be improved with corrections in IBR-DTN core code.
- **Development of new DTN applications**
DTN platforms still require new applications, with improved usability and interface. New solutions may be applied not only in underwater transmission, but also in regions of low connectivity, or even between moving vehicles and individuals. Smartphones, tablets and other communicating devices may benefit from the development of interesting applications, that still require more compatibility regarding different operating systems.

Appendix A

Code used for file synchronization

```
/*
 * dtnoutbox.cpp
 *
 * Copyright (C) 2013 IBR, TU Braunschweig
 *
 * Written-by: Johannes Morgenroth <morgenroth@ibr.cs.tu-bs.de>
 *             David Goltzsche <goltzsch@ibr.cs.tu-
 *             bs.de>
 *
 * Licensed under the Apache License, Version 2.0 (the "License
 * ");
 * you may not use this file except in compliance with the
 * License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software
 * distributed under the License is distributed on an "AS IS"
 * BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
 * or implied.
 * See the License for the specific language governing
 * permissions and
 * limitations under the License.
 */
```

```

#include "md5.h" /* Calcular md5*/
#include "config.h"
#include <ibrdtn/api/Client.h>
#include <ibrcommon/net/socket.h>
#include <ibrcommon/thread/Mutex.h>
#include <ibrcommon/thread/MutexLock.h>
#include <ibrcommon/thread/SignalHandler.h>
#include <ibrdtn/data/PayloadBlock.h>
#include <ibrcommon/data/BLOB.h>
#include <ibrcommon/data/File.h>
#include <ibrcommon/appstreambuf.h>
#include <ibrcommon/Logger.h>

#include "io/TarUtils.h"
#include "io/ObservedFile.h"

#ifdef HAVE_LIBTFFS
#include "io/FatImageReader.h"
#endif

#include <stdlib.h>
#include <iostream>
#include <map>
#include <vector>
#include <sys/types.h>
#include <unistd.h>
#include <regex.h>
#include <getopt.h>
#include <vector> /* Incluir a biblioteca para utilizar vetores de
strings */
#include <ctime>
#include <cstdio>
#include <ctype.h> /* Biblioteca para calcular md5*/

typedef std::list<io::ObservedFile> filelist;
typedef std::set<io::ObservedFile> fileset;
typedef std::set<io::FileHash> hashlist;

```

```
// set this variable to false to stop the app
bool _running = true;

//global wait conditional
ibrcommon::Conditional _wait_cond;
bool _wait_abort = false;

const std::string TAG = "dtnoutbox";

class config {
public:
    config()
        : interval(5000), rounds(3), path("/"), regex_str("^\\.
            "),
          bundle_group(false), invert(false), quiet(false)
            , verbose(false), fat(false), enabled(true)
    {}

    //global conf values
    string name;
    string outbox;
    string destination;

    //optional paramters
    std::string workdir;
    std::size_t interval;
    std::size_t rounds;
    std::string path;
    std::string regex_str;
    regex_t regex;

    int bundle_group;
    int invert;
    int quiet;
    int verbose;
    int fat;
    int enabled;
};

typedef struct config config_t;
```

```

struct option long_options[] =
{
    {"destination", required_argument, 0, 'd'},
    {"help", no_argument, 0, 'h'},
    {"group", no_argument, 0, 'g'},
    {"workdir", required_argument, 0, 'w'},
    {"interval", required_argument, 0, 'i'},
    {"rounds", required_argument, 0, 'r'},
    {"path", required_argument, 0, 'p'},
    {"regex", required_argument, 0, 'R'},
    {"quiet", no_argument, 0, 'q'},
    {"verbose", no_argument, 0, 'v'},
    {0, 0, 0, 0}
};

void init_logger(config_t &conf)
{

    // logging options
    const unsigned char logopts = 0;

    // error filter
    const unsigned char logerr = ibrccommon::Logger::
        LOGGER_ERR | ibrccommon::Logger::LOGGER_CRIT;

    // logging filter , everything but debug, err and crit
    const unsigned char logstd = ibrccommon::Logger::
        LOGGER_ALL ^ (ibrccommon::Logger::LOGGER_DEBUG |
            logerr);

    // syslog filter , everything but DEBUG and NOTICE
    const unsigned char logsys = ibrccommon::Logger::
        LOGGER_ALL ^ (ibrccommon::Logger::LOGGER_DEBUG |
            ibrccommon::Logger::LOGGER_NOTICE);

    const unsigned char logall = ibrccommon::Logger::
        LOGGER_ALL;

    //log notice messages to cout, if quiet not configured
    if(!conf.quiet)

```



```

    {
        ibrccommon::Logger::addStream(std::cout, logsys,
            logopts);
        if (conf.verbose)
        {
            ibrccommon::Logger::addStream(std::cout,
                logall, logopts);
        }
    }

    // add logging to the cerr
    ibrccommon::Logger::addStream(std::cerr, logerr, logopts)
        ;
}

void print_help()
{
    std::cout << "—_dtnoutbox_(IBR-DTN)_—" << std::endl;
    std::cout << "Syntax: _dtnoutbox_[ options ]_<name>_<outbox
        >_<destination>" << std::endl;
    std::cout << "_<name>_The_application_name" <<
        std::endl;
#ifdef HAVE_LIBTFFS
    std::cout << "_<outbox>_Location_of_outgoing_
        files,_directory_or_vfat-image" << std::endl;
#else
    std::cout << "_<outbox>_Directory_of_outgoing_
        files" << std::endl;
#endif
    std::cout << "_<destination>_The_destination_EID_for_
        all_outgoing_files" << std::endl << std::endl;
    std::cout << "*_optional_parameters_" << std::endl;
    std::cout << "_-h|--help_ Display_this_text" <<
        std::endl;
    std::cout << "_-g|--group_Receiver_is_a_
        destination_group" << std::endl;
    std::cout << "_-w|--workdir_<dir>" << std::endl;
    std::cout << "Temporary_work_directory
        " << std::endl;
}

```

```

std::cout << "  -i|--interval<milliseconds>" << std::
    endl;
std::cout << "  Interval in milliseconds
    , in which<outbox> is scanned" << std::endl;
std::cout << "  for new/changed files .
    default: 5000" << std::endl;
std::cout << "  -r|--rounds<n>  Number of rounds of
    intervals , after which a unchanged" << std::endl;
std::cout << "  file is considered as
    written . default: 3" << std::endl;
#ifdef HAVE_LIBTFFS
std::cout << "  -p|--path<path> Path of outbox within
    vfat-image . default: /" << std::endl;
#endif
std::cout << "  -R|--regex<regex>" << std::endl;
std::cout << "  All files in<outbox>
    matching this regular expression" << std::endl;
std::cout << "  will be ignored . default
    : ^\\. " << std::endl;
std::cout << "  -I|--invert  Invert the regular
    expression defined with -R" << std::endl;
std::cout << "  -q|--quiet  Only print error
    messages" << std::endl;
std::cout << "  -v|--verbose  print more verbose info
    messages , only works without -q" << std::endl;

    _running = false; //stop this app, after printing help
}

void read_configuration(int argc , char** argv , config_t &conf)
{
    while(1)
    {
        /* getopt_long stores the option index here. */
        int option_index = 0;
        int c = getopt_long (argc , argv , "hw:i:r:p:R:
            qvIg" ,
                            long_options , &option_index);
        /* Detect the end of the options. */

```

```
    if (c == -1)
        break;

    switch (c)
    {
    case 0:
        /* If this option set a flag, do nothing
           else now. */
        if (long_options[option_index].flag !=
            0)
            break;
        printf ("option_%" , long_options[
            option_index].name);
        if (optarg)
            printf ("_with_arg%" , optarg);
        printf ("\n");
        break;

    case 'h':
        print_help();
        exit(EXIT_SUCCESS);
        break;

    case 'g':
        conf.bundle_group = true;
        break;

    case 'w':
        conf.workdir = std::string(optarg);
        break;

    case 'i':
        conf.interval = atoi(optarg);
        break;

    case 'r':
        conf.rounds = atoi(optarg);
        break;

    case 'p':
        conf.path = std::string(optarg);
        break;

    case 'R':
        conf.regex_str = std::string(optarg);
        break;
```

```

        case 'q':
            conf.quiet = true;
            break;
        case 'v':
            conf.verbose = true;
            break;
        case 'I':
            conf.invert = true;
            break;
        case '?':
            break;
        default:
            abort();
            break;
    }
}

// print help if not enough parameters are set
if ((argc - optind) < 3)
{
    print_help();
    exit(EXIT_FAILURE);
}

conf.name = std::string(argv[optind]);
conf.destination = std::string(argv[optind+2]);
conf.outbox = std::string(argv[optind+1]);

// compile regex, if set
if (conf.regex_str.length() > 0 && regcomp(&conf.regex,
    conf.regex_str.c_str(), 0))
{
    IBRCOMMON_LOGGER_TAG(TAG, error) << "ERROR: _
        invalid_regex:_" << optarg <<
        IBRCOMMON_LOGGER_ENDL;
    exit(-1);
}

// check outbox path for trailing slash

```

```

        if ( conf.outbox.at( conf.outbox.length() - 1 ) == '/' )
            conf.outbox = conf.outbox.substr( 0, conf.outbox.
                length() - 1 );
    }

    void sighandler_func( int signal )
    {
        IBRCOMMON_LOGGER_TAG( TAG, notice ) << "got_signal_" <<
            signal << IBRCOMMON_LOGGER_ENDL;
        switch ( signal )
        {
            case SIGTERM:
            case SIGINT:
            {
                // stop waiting and stop running, on SIGINT or
                SIGTERM
                ibrccommon::MutexLock l( _wait_cond );
                _running = false;
                _wait_cond.signal( true );
                break;
            }
#ifdef __WIN32__
            case SIGUSR1:
            {
                // stop waiting on SIGUSR1 -> "quickscan"
                ibrccommon::MutexLock l( _wait_cond );
                _wait_abort = true;
                _wait_cond.signal( true );
                break;
            }
#endif
            default:
                break;
        }
    }

    /*
     * main application method
     */
    int main( int argc, char** argv )

```

```
{  
  
    int contador=0;  
  
    int listHostBPresent=0;  
    string listLocalHost , listHostB ;  
    string fileData ;  
    string fileDataVector ;  
    string fileDataTime ;  
    char md5[MD5_LEN + 1];  
  
  
  
  
  
  
  
  
  
    std::vector<std::string> elementosLista ;  
    std::vector<std::string> elementosComuns ;  
    std::vector<std::string> elementosApagar ;  
    std::vector<std::string> foreignFileData ;  
    std::vector<std::string> elementosActList ;  
    std::vector<std::string> elementosActListB ;  
    std::vector<std::string> elementosEnviados ;  
  
    // catch process signals  
    ibrccommon::SignalHandler sighandler(sighandler_func);  
    sighandler.handle(SIGINT);  
    sighandler.handle(SIGTERM);  
#ifndef __WIN32__  
    sighandler.handle(SIGUSR1);  
#endif  
  
    // configuration object  
    config_t conf;  
  
    // read the configuration  
    read_configuration(argc , argv , conf);  
  
    init_logger(conf);
```

```
// initialize sighandler after possible exit call
sighandler.initialize();

// init working directory
if (conf.workdir.length() > 0)
{
    ibrcommon::File blob_path(conf.workdir);

    if (blob_path.exists())
    {
        ibrcommon::BLOB::changeProvider(new
            ibrcommon::FileBLOBProvider(blob_path
            ), true);
    }
}

std::string str;
str=conf.destination;
char seps[] = "/";/
const char *indicateList="List_to_";
char *namelistA;
const char *namelistB;
string pathListB;

namelistA = strtok( &str[0], seps );
while( namelistA != NULL )
{
    /* Do your thing */
    namelistA = strtok( NULL, seps );
    break;
}

listLocalHost=namelistA;

const char *raiz = "./";
string pathList= raiz+conf.outbox+seps+namelistA;
```

```

time_t date;
// backoff for reconnect
unsigned int backoff = 2;

ibrcommon::File outbox_file(conf.outbox);

// create new file lists
fileset new_files, prev_files, deleted_files,
    files_to_send, ficheiros_enviar;
filelist observed_files;
hashlist sent_hashes;

// observed root file
io::ObservedFile root(ibrcommon::File("/"));

#ifdef HAVE_LIBTFFS
    io::FatImageReader *imagereader = NULL;
#endif

    if (outbox_file.exists() && !outbox_file.isDirectory())
    {
#ifdef HAVE_LIBTFFS
        conf.fat = true;
        imagereader = new io::FatImageReader(conf.outbox
            );
        const io::FATFile fat_root(*imagereader, conf.
            path);
        root = io::ObservedFile(fat_root);
#else
        IBRCOMMON_LOGGER_TAG(TAG, error) << "ERROR: _image
            -file _provided, _but _this _tool _has _been _
            compiled _without _libtffs _support!" <<
            IBRCOMMON_LOGGER_ENDL;
        return -1;
#endif
    }
    else
    {
        if (!outbox_file.exists()) {

```



```

        ibrcommon::File::createDirectory(
            outbox_file);
    }
    root = io::ObservedFile(outbox_file);
}

IBRCOMMON_LOGGER_TAG(TAG, info) << "—_dtnoutbox_—" <<
    IBRCOMMON_LOGGER_ENDL;

// loop, if no stop if requested
while (_running)
{
    try
    {
        // Create a stream to the server using
        // TCP.
        ibrcommon::vaddress addr("localhost",
            4550);
        ibrcommon::socketstream conn(new
            ibrcommon::tcpsocket(addr));

        // Initiate a client for synchronous
        // receiving
        dtn::api::Client client(conf.name, conn,
            dtn::api::Client::MODE_SENDONLY);

        // Connect to the server. Actually, this
        // function initiate the
        // stream protocol by starting the
        // thread and sending the contact header
        .
        client.connect();

        // reset backoff if connected
        backoff = 2;

        // check the connection
        while (_running)
        {

```

```

// get all files
fileset current_files;
root.findFiles(current_files);
elementosApagar.clear();
// determine deleted files
fileset deleted_files;
std::set_difference(prev_files.
    begin(), prev_files.end(),
    current_files.begin(),
    current_files.end(),

                    std::inserter(
                        deleted_files, deleted_files.
                        begin()));
// for (fileset::const_iterator
    iter = deleted_files.begin(); iter != deleted_files.end(); ++
    iter){
//         const io::ObservedFile &
    of = (*iter);
//         elementosApagar.push_back(of.
    getFile().getBasename().c_str());
//     }

    elementosActList.clear();
    elementosComuns.clear();
    elementosActListB.clear();
    std::ofstream outfile(pathList.
        c_str(), std::ios::out | std::
        ios::trunc);
    for (fileset::const_iterator
        iter = current_files.begin();
        iter != current_files.end();
        ++iter)
    {

```

```
const io::ObservedFile &
    of = (*iter);

std::size_t found;
found=of.getFile().
    getBasename().find(".
    dtn");

if(found!=std::string::
    npos&&(strcmp(
    listLocalHost.c_str()
    , of.getFile().
    getBasename().c_str()
    )))
{

    namelistB=of.
        getFile().
        getBasename()
        .c_str();
    listHostB=of.
        getFile().
        getBasename()
        ;
    pathListB=raiz+
        conf.outbox+
        seps+
        namelistB;
        listHostBPresent
            =1;
}

found=of.getFile().
    getBasename().find(".
    dtn");
if(found==std::string::
    npos)
```

```

{
fileData . clear () ;
fileDataVector . clear () ;
stringstream ss ;
ss << of . getFile () .
    lastmodify () ;
string timestamp = ss .
    str () ;
fileData . append (
    timestamp ) ;
fileData . append ("_") ;
fileData . append (of .
    getFile () . getBasename
    () . c_str ()) ;
fileDataVector . append (of
    . getFile () .
    getBasename () . c_str ()
    ) ;

char buff [20] ;

char *filePath = new
    char [of . getFile () .
        getPath () . length () +
        1] ;
strcpy (filePath , of .
    getFile () . getPath () .
    c_str ()) ;

if (!CalcFileMD5 (
    filePath , md5))
puts ("Error_occured !") ;

fileData . append ("_") ;
fileDataVector . append ("_") ;

fileData . append (md5) ;

```

// do stuff

```

        fileDataVector.append(
            md5);

        if (std::find(
            elementosActList.
            begin(),
            elementosActList.end
            (), fileDataVector)
            == elementosActList.
            end())
            elementosActList.
            push_back(
                fileDataVector);

        outfile <<fileData <<endl;
        }
    }

    if (!deleted_files.empty()) {
    for (std::vector<string>::const_iterator i = elementosApagar.
        begin(); i != elementosApagar.end(); ++i)
        outfile <<"Delete:_"<< *i<<endl;
    }

    outfile.close();
    outfile <<std::flush;

    std::ifstream infile;
    string fileOnHostB;
    if (listHostBPresent) {
    infile.open(pathListB.c_str());
    while (!infile.eof())
    {
    getline(infile, fileOnHostB);
    elementosActListB.push_back(
        fileOnHostB);
    }
    }

```

```

std::size_t found;
found=fileOnHostB.find("Delete")
;
if(found!=std::string::npos)
{
    string elemento_delete=
        fileOnHostB.substr(0,
            fileOnHostB.find('␣'
                ));
    string
        pathElementoEleminar=
            raiz+conf.outbox+
            seps+elemento_delete;
    remove(
        pathElementoEleminar.
            c_str());
}
if(!fileOnHostB.empty()){
fileDataVector=fileOnHostB.
    substr(11, fileOnHostB.find("
    \n", 0));
if (std::find(elementosActList.
    begin(), elementosActList.end
    (), fileDataVector) !=
    elementosActList.end())
    {
        elementosComuns.
            push_back(
                fileDataVector);
    }
}
}
}

```

*// remove deleted files from
observation*

```
for (fileset::const_iterator
    iter = deleted_files.begin();
    iter != deleted_files.end();
    ++iter)
{
    const io::ObservedFile &
        deletedFile = (*iter)
        ;

    // remove references in
    // the sent_hashes
    for (hashlist::iterator
        hash_it = sent_hashes
            .begin(); hash_it !=
            sent_hashes.end(); /*
            blank */) {
        if ((*hash_it).
            getPath() ==
            deletedFile.
            getFile().
            getPath()) {
            sent_hashes
                .
                erase
                (
                hash_it
                ++);
        } else {
            ++
                hash_it
                ;
        }
    }
}

// remove from observed
// files
observed_files.remove(
    deletedFile);

// output
```

```

IBRCOMMON_LOGGER_TAG(TAG
, info) << "file_
removed:_" <<
deletedFile.getFile()
.getBasename() <<
IBRCOMMON_LOGGER_ENDL
;
}

```

```

// determine new files
fileset new_files;
std::set_difference(
    current_files.begin(),
    current_files.end(),
    prev_files.begin(),
    prev_files.end(), std::
    inserter(new_files, new_files
    .begin()));

```

```

// add new files to observation
for (fileset::const_iterator
    iter = new_files.begin();
    iter != new_files.end(); ++
    iter)
{
    const io::ObservedFile &
        of = (*iter);

    int reg_ret = regexec(&
        conf.regex, of.
        getFile().getBasename
        ().c_str(), 0, NULL,
        0);
    if (!reg_ret && !conf.
        invert)

```



```

        continue;
    if (reg_ret && conf.
        invert)
        continue;

    // print error message,
    // if regex error occurs
    if (reg_ret && reg_ret
        != REG_NOMATCH)
    {
        char
            msgbuf
            [100];

        regerror
            (
                reg_ret
                ,&
                conf.
                regex
                ,
                msgbuf
                ,
                sizeof
                (
                    msgbuf
                )
            );
        IBRCOMMON_LOGGER_TAG
            (TAG,
            info)
            << "
            ERROR
            :_
            regex
            _
            match
            _
            failed
            _:_ "
            <<
    }

```

```

        std::
        string
        (
        msgbuf
        ) <<
        IBRCOMMON_LOGGER_ENDL
        ;
    }

    // add new file to the
    // observed set
    observed_files.push_back
    (of);

    // log output
    IBRCOMMON_LOGGER_TAG(TAG
    , info) << "file_
    found:_ " << of.
    getFile().getBasename
    () <<
    IBRCOMMON_LOGGER_ENDL
    ;

}

// store current files for the
// next round
prev_files.clear();
prev_files.insert(current_files.
begin(), current_files.end())
;

IBRCOMMON_LOGGER_TAG(TAG, notice
)
    << "file_
    statistics:_ "

```

```

        <<
            observed_files
            .size() << "
            observed, "
        << deleted_files
            .size() << "
            deleted, "
        << new_files.
            size() << "
            new"
        <<
            IBRCOMMON_LOGGER_ENDL
            ;

// find files to send, create
// std::list
files_to_send.clear();
ficheiros_enviar.clear();
IBRCOMMON_LOGGER_TAG(TAG, notice
) << "updating_observed_files
:" << IBRCOMMON_LOGGER_ENDL;
for (filelist::iterator iter =
observed_files.begin(); iter
!= observed_files.end(); ++
iter)
{
    io::ObservedFile &of =
        (*iter);

// tick and update all
// files
of.update();

fileData.clear();
fileData.append(of.
getFile().getBasename
().c_str());
char buff[20];

```

```

// do stuff

char *filePath = new
    char[of.getFile().
        getPath().length() +
        1];
strcpy(filePath, of.
    getFile().getPath().
    c_str());

if (!CalcFileMD5(
    filePath, md5))
    puts("Error_occured!");

printf("timestamp:_%d",
    of.getFile().
    lastmodify());
stringstream ss;
ss << of.getFile().
    lastmodify();
string timestamp = ss.
    str();

fileData.append("_");

fileData.append(md5);

int comun=0;

if (std::find(elementosComuns.begin(),
    elementosComuns.end(), fileData.c_str
    ()) != elementosComuns.end())
{
    comun=1;
}
else{
    comun=0;
}

```

```
}
```

```
if (
    listHostBPresent
){
```

```
if (of.
    getStableCounter
    () > conf.
    rounds ||
    strcmp(of.
    getFile().
    getBasename()
    .c_str(),
    listLocalHost
    .c_str()))
```

```
{
```

```
if (strcmp(of.getFile().
    getBasename().c_str()
    , listHostB.c_str()))
{
```

```
if (!comun){
```

```
    sent_hashes
```

```
    .
```

```
    insert
```

```
    (of.
```

```
    getHash
```

```
    ());
```

```
files_to_send
```

```
    .
```

```
    insert
```

```
    (*
```

```
    iter)
```

```
    ;
```

```
}
```

```

    }
}

if (!listHostBPresent){

    if (!strcmp(of.
        getFile().
        getBasename()
        .c_str(),
        listLocalHost
        .c_str())){
        if ((!
            comun
        )){
            sent_hashes
            .
            insert
            (of.
            getHash
            ());
            files_to_send
            .
            insert
            (*
            iter)
            ;
        }
    }
}

IBRCOMMON_LOGGER_TAG(TAG
, notice)
<< "\t"
<< of.getFile().
getBasename()
<< ":\t"
<< of.
getStableCounter
()

```

```

        <<
            IBRCOMMON_LOGGER_ENDL
        ;
    }

    if (!files_to_send.empty())
    {
        std::stringstream ss;
        for (fileset::
            const_iterator it =
            files_to_send.begin()
            ; it != files_to_send
            .end(); ++it) {
            ss << (*it).
                getFile().
                getBasename()
                << "_";
            ficheiros_enviar
                .insert(*it);

            IBRCOMMON_LOGGER_TAG("
                dtnoutbox",info) << "
                files_sent:_" << ss.
                str() <<
                IBRCOMMON_LOGGER_ENDL
            ;

            try {
                // create a blob
                ibrccommon::BLOB
                    ::Reference
                    blob =
                    ibrccommon::
                    BLOB::create
                    ();

                // write files
                into BLOB
                while it is
                locked

```

```
{
    ibrcommon
        ::
        BLOB
        ::
        iostream

        stream
        =
        blob .
        iostream
        ();
    io ::
        TarUtils
        ::
        write
        (*
        stream
        ,
        root ,

        ficheiros_enviar
        );
}
// create a new
// bundle
dtn :: data :: EID
    destination =
    EID(conf .
    destination);

// create a new
// bundle
dtn :: data ::
    Bundle b;

// set
// destination
b.destination =
    destination;
```



```
// add payload  
block using  
the blob  
b.push_back(blob  
);  
  
// set  
destination  
address to  
non-singleton  
, if  
configured  
if (conf.  
bundle_group)  
    b.set(  
        dtn::  
        data  
        ::  
        PrimaryBlock  
        ::  
        DESTINATION_IS_SIN  
        ,  
        false  
    );  
  
// send the  
bundle  
client << b;  
client.flush();  
  
} catch (const  
    ibrccommon::  
    IOException &  
    e) {  
        IBRCOMMON_LOGGER_TAG  
        (TAG,  
        error  
        ) <<
```

```

        " send
        _
        failed
        :_"
        << e.
        what
        () <<

        IBRCOMMON_LOGGER_ENDL
        ;
    }
    ss.clear();
    ficheiros_enviar.clear()
    ;
}

// wait defined seconds
ibrcommon::MutexLock l(
    _wait_cond);
IBRCOMMON_LOGGER_TAG(TAG, notice
) << conf.interval <<"_ms_
wait" <<
IBRCOMMON_LOGGER_ENDL;

while (!_wait_abort && _running)
{
    _wait_cond.wait(conf.
        interval);
}
_wait_abort = false;
}

// clean up regex
regfree(&conf.regex);

// close the client connection
client.close();

```

```
        // close the connection
        conn.close();
    }
    catch (const ibrccommon::socket_exception&)
    {
        if (_running)
        {
            IBRCOMMON_LOGGER_TAG(TAG, error)
                << "Connection_to_bundle_
                daemon_failed._Retry_in_" <<
                backoff << "_seconds." <<
                IBRCOMMON_LOGGER_ENDL;
            ibrccommon::Thread::sleep(backoff
                * 1000);

            // if backoff < 10 minutes
            if (backoff < 600)
            {
                // set a new backoff
                backoff = backoff * 2;
            }
        }
    }
    catch (const ibrccommon::IOException&)
    {
        if (_running)
        {
            IBRCOMMON_LOGGER_TAG(TAG, error)
                << "Connection_to_bundle_
                daemon_failed._Retry_in_" <<
                backoff << "_seconds." <<
                IBRCOMMON_LOGGER_ENDL;
            ibrccommon::Thread::sleep(backoff
                * 1000);

            // if backoff < 10 minutes
            if (backoff < 600)
            {
                // set a new backoff
```

```

        backoff = backoff * 2;
    }
}
}
    catch (const std::exception&) { };
}

    // clear observed files
    observed_files.clear();

#ifdef HAVE_LIBTFFS
    // clean-up
    if (imagereader != NULL) delete imagereader;
#endif

    return (EXIT_SUCCESS);
}

int CalcFileMD5(char *file, char *md5_sum)
{
    #define MD5SUM_CMD_FMT "md5sum_%s." STR(PATH_LEN) "s_2>/dev/
    null"
    char aux[PATH_LEN + sizeof (MD5SUM_CMD_FMT)];
    sprintf(aux, MD5SUM_CMD_FMT, file);
    #undef MD5SUM_CMD_FMT

    FILE *pointer = popen(aux, "r");
    if (pointer == NULL) return 0;

    int i, ch;
    for (i = 0; i < MD5_LEN && isxdigit(ch = fgetc(pointer)); i
        ++) {
        *md5_sum++ = ch;
    }

    *md5_sum = '\0';
    pclose(pointer);
    return i == MD5_LEN;
}

```

}

References

- [1] J. M. Hovem. Modeling and measurement methods for acoustic waves and for acoustic microdevices. 2013.
- [2] F. B. Teixeira, P. F., L. Pessoa, R. Campos, and M. P. Ricardo. Evaluation of IEEE 802.11 underwater networks operating at 700 MHz, 2.4 GHz and 5 GHz. *Proceedings of the International Conference on Underwater Networks and Systems*, pages 103–114, November 2014.
- [3] Forrest Warthman. Delay- and disruption-tolerant networks: A tutorial, July 2012. Warthman Associates.
- [4] M. J. Khabbaz, C. M. Assi, W. F., and F. Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges. *IEEE Communications Surveys and Tutorials*, pages 607–640, May 2011.
- [5] PC Engines Alix3D3 Accessed: 20/04/2017 URL: <https://www.pcengines.ch/alix3d3.htm>.
- [6] Mikrotik Accessed: 26/04/2017 URL: <https://routerboard.com/R52NM>.
- [7] N. Faidhon, J. Agung N., and S. Unang. Unexar – mini auv design and measurement. *Proceedings of the International Conference on Control, Electronics, Renewable Energy, and Communications*, pages 203–207, November 2015.
- [8] S. I. Inácio, M. R. Pereira, H. M. Santos, L. M. Pessoa, F. B. Teixeira, M. J. Lopes, O. Aboderin, and H. M. Salgado. Dipole antenna for underwater radio communications. *Proceedings of the Underwater Communications and Networking Conference (UComms)*, pages 103–114, October 2016.
- [9] F. B. Teixeira, J. Santos, L. Pessoa, M. Pereira, R. Campos, and M. P. Ricardo. Evaluation of IEEE 802.11 underwater networks at VHF and UHF frequency bands using software defined radios. *Proceedings of the 10th International Conference on Underwater Networks and Systems*, pages 103–114, October 2015.
- [10] K. Wong, T. Wan, and W. Ang. A survey on current status of disruption tolerant network support for multicast. *2016 3rd International Conference On Computer And Information Sciences (ICCOINS)*, pages 276–281, December 2016.
- [11] A. Fujihara and H. Miwa. On the use of congestion information for rerouting in the disaster evacuation guidance using opportunistic communication. *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, pages 563–568, September 2013.

- [12] F. Raimondi, M. Trapanese, V. Franzitta, A. Viola, and A. Colucci. A innovative semi-immersible usv (si-usv) drone for marine and lakes operations with instrumental telemetry and acoustic data acquisition capability. *Proceedings of OCEANS 2015 - Genova*, pages 1–10, September 2015.
- [13] S. I. Inácio, H. M. Santos, L. M. Pessoa, F. B. Teixeira, M. J. Lopes, O. Aboderin, and H. M. Salgado. Antenna design for underwater radio communications. *Proceedings of OCEANS 2016*, pages 103–114, April 2016.
- [14] R. Su, R. Venkatesan, and C. Li. Acoustic propagation properties of underwater communication channels. *2012 IEEE*, pages 5015–5019, November 2012.
- [15] P. M. Freitas. Evaluation of Wi-Fi underwater networks in freshwater, July 2014. Master’s Thesis FEUP, University of Porto.
- [16] H. Brundage. Designing a Wireless Underwater Optical Communication System. Master’s thesis, Massachusetts Institute of Technology, 2010.
- [17] M. Stojanovic. Underwater acoustic communications: Design considerations on the physical layer. *Proceedings of the Fifth Annual Conference on Wireless on Demand Network Systems and Services*, January 2008.
- [18] L. Liu, Y. Zhang, P. Zhang, L. Zhou, J. Li, J. Jin, and J. Zhang. PN sequence based doppler and channel estimation for underwater acoustic OFDM communication. *Proceedings of the IEEE International Conference on Signal Processing, Communications and Computing (IC-SPCC)*, pages 103–114, November 2016.
- [19] P. Walree. Propagation and scattering effects in underwater acoustic communication channels. *IEEE JOURNAL OF OCEAN ICENGINEERING, VOL. 38, NO. 4, OCTOBER 2013*, pages 892–897, October 2013.
- [20] LinkQuest Inc. Accessed: 25/11/2016 URL : http://www.link-quest.com/html/pic_uwm1000.htm.
- [21] SUBSEA 20/20. Accessed: 26/11/2016 URL : <http://www.subsea2020.com>.
- [22] X. Lurton. An introduction to underwater acoustics principles and applications, 2010. Springer.
- [23] SA Photonics. Accessed: 26/11/2016 URL: <http://www.saphotonics.com/high-bandwidth-optical-communications/underwater/>.
- [24] Ambalux. Accessed: 26/11/2016 URL: http://www.ambalux.com/gdresources/media/AMB_1013_Brochure.pdf.
- [25] J. P. Santos. Evaluation of IEEE 802.11a/g/p transceiver for SDR, July 2015. Master’s thesis, FEUP, University of Porto.
- [26] F. B. Teixeira, R. Campos, and M. P. Ricardo. IEEE 802.11 rate adaptation algorithms in underwater environment. *Proceedings of the 10th International Conference on Underwater Networks and Systems*, pages 103–114, October 2015.
- [27] RFC 4838 Accessed : 5/10/2016 URL: <https://tools.ietf.org/html/rfc4838>.

- [28] RFC 5050 Accessed : 5/10/2016 URL: <https://tools.ietf.org/html/rfc5050>.
- [29] K. Okamoto and K. Takami. Routing based on information about the routes of fixed-route traveling nodes and on destination areas aimed at reducing the load on the dtn. *Future Internet*, pages 82–89, April 2016.
- [30] A. Papalambrou, A. G. Voyiatzis, D. N. Serpanos, and P. Soufrilas. Monitoring of a DTN2 network. *Internet Communications*, pages 103–114, March 2011.
- [31] U. E., R. V., S. S., and S. Udupa. Delay tolerant network for space. *Proceedings of the 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 103–114, April 2016.
- [32] P. Tsao and S. Nguyen. Bptap: A new approach toward IP over DTN. *Aerospace Conference, 2012 IEEE*, pages 607–640, April 2012.
- [33] HTTP-DTN. Accessed : 25/11/2016 URL: <https://sourceforge.net/projects/http-dtn/>.
- [34] A. Wibowo E. Husni. E-mail system for Delay Tolerant Network. *Proceedings of the International Conference on System Engineering and Technology*, pages 103–114, September 2012.
- [35] P. Gilbert, V. Ramasubramanian, and D. Terry P. Stuedi. Peer-to-peer Data Replication meets Delay Tolerant Networking. *2011 31st International Conference on Distributed Computing Systems*, pages 103–114, September 2012.
- [36] S. Dutt, D. Habibi, and I. Ahmad. A low cost atheros system-on-chip and OpenWrt based testbed for 802.11 WLAN research. *Proceedings of the TENCON 2012 - 2012 IEEE Region 10 Conference*, pages 82–89, January 2013.
- [37] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf. IBR-DTN: an efficient implementation for embedded systems. *CHANTS '08 Proceedings of the third ACM workshop on Challenged networks*, pages 117–120, September 2008.
- [38] A. Galati, T. Bourchas, S. Siby, and S. Mangold. System architecture for delay tolerant media distribution for rural south africa. *Proceedings of the WiNTECH' 14, 7 September, 2014 Maui Hawaii USA*, pages 9–15, September 2014.
- [39] A. Galati, T. Bourchas, S. Siby, S. Frey, M. Olivares, and S. Mangold. Mobile-enabled delay tolerant networking in rural developing regions. *Proceedings of the 2014 IEEE Global Humanitarian Technology Conference*, pages 699–706, September 2014.
- [40] J. Morgenroth, T. Pögel, and L. Wolf. Live-streaming in delay tolerant networks. *Proceedings of the CHANTS'11, September 23, 2011, Las Vegas, Nevada, USA.*, pages 82–89, September 2011.
- [41] J. Morgenroth, S. Schildtl, and L. Wolf. A bundle protocol implementation for android devices. *Proceedings of the MobiCom'12, August 22-26, 2012, Istanbul, Turkey*, pages 443–445, August 2012.
- [42] Shyam Balasubramanian. Store-and-forward networking solutions with autonomous aerial vehicles, January 2014. Indhoven University of Technology, Thales Nederland B.V.

- [43] IBR-DTN Accessed: 6/12/2016 URL: <https://github.com/ibrdtn>.
- [44] J. Morgenroth, S. Schildtl and W. Pottner, and T. Lorentzen L. Wolf. Free-riding the bittorrent dht to improve dtn connectivity. *Proceedings of the CHANTS'12, August 22, 2012, Istanbul, Turkey*, pages 9–15, August 2012.
- [45] IBR-DTN repository Accessed: 10/04/2017 URL: <https://github.com/ibrdtn/ibrdtn>.
- [46] G. Sullivan, P. Topiwala, and A. Luthra. The h.264/avc advanced video coding standard: Overview and introduction to the fidelity range extensions. *Proceedings of the SPIE Conference on Applications of Digital Image Processing XXVII Special Session on Advances in the New Emerging Standard: H.264/AVC, August, 2004*, pages 203–207, August 2004.
- [47] William Stallings. *Data and computer communications*. 2007.