A Reliable Transfer File Protocol for Multi-hop Underwater Acoustic Networks

1st Vincenzo Cimino
Dept. of Information Engineering
University of Padova
Padua, Italy
vincenzo.cimino@unipd.it

2nd Filippo Campagnaro
Dept. of Information Engineering
University of Padova and *SubSeaPulse SRL*
Padua, Italy
filippo.campagnaro@unipd.it

3rd Michele Zorzi
Dept. of Information Engineering
University of Padua
and *SubSeaPulse SRL*
Padua, Italy
michele.zorzi@unipd.it

***Abstract*—This work addresses reliable data transfer in underwater acoustic networks. We develop a protocol based on the Trivial File Transfer Protocol (TFTP) [1], a UDP-based file transfer protocol. First, we present the Underwater Trivial File Transfer Protocol (UW-TFTP), a variant specifically adapted for underwater acoustic environments. Building upon this, we introduce an optimized version, UW-TFTP-Buffer (UW-TFTP-B), which introduces an out-of-order packet buffer at the receiver side. This enhancement improves resilience to packet loss while preserving the original Go-Back-N strategy to avoid the complexity of packet reordering on the transmitter side. We compare the performance of both versions in a multi-hop scenario, using emulated acoustic modems, in terms of the time required for successful file transfer under increasing packet error rates, thus demonstrating the effectiveness of our optimized approach in challenging underwater communication conditions. Finally, we evaluate the two protocols in a end-to-end scenario with two actual low-cost acoustic modems in a small water volume.**

***Index Terms*—Underwater Acoustic Networks, DESERT Simulation Framework, File Transfer Protocol.**

I. Introduction

Reliable data communication in underwater networks is crucial for a wide range of applications [2], including the transmission of multimedia content (such as images and videos for environmental monitoring) [3], underwater exploration, surveillance [4], and inspection of underwater pipelines and cables [5]. However, achieving robust and efficient communication in underwater environments remains a major challenge due to the inherent limitations of the physical medium [6]. The underwater wireless channel is recognized as one of the most challenging communication environments. The strong attenuation of electromagnetic signals makes radio-frequency transmissions impractical for most underwater scenarios [7]. Although optical communication [8] typically offers higher data rates and low latency, it is limited to short distances (usually a few tens of meters), requires highly accurate alignment, and is significantly affected by ambient light conditions, which can introduce noise and degrade signal quality. Acoustic communication, in contrast, offers much longer transmission ranges, often extending to several kilometers, and is currently the most used technology for underwater wireless networks. Despite this, it offers limited communication performance due to limited bandwidth and long propagation delays. Additionally, communication can be frequently disrupted by multipath effects, shadow zones, and ambient noise originating from ships, weather conditions, and marine life. These factors pose significant challenges to achieving reliable end-to-end data delivery, especially in multi-hop networks.

Traditional Internet protocols are not directly applicable to underwater acoustic networks. However, most underwater applications rely on datagram-based transport layers that follow principles similar to the User Datagram Protocol (UDP). These UDP-inspired protocols are often preferred over implementations that mimic the Transmission Control Protocol (TCP) due to their simplicity, low overhead, and connectionless nature, making them better suited for the constrained underwater acoustic channel. In contrast, the congestion control and acknowledgment mechanisms inherent in TCP-based approaches, which are designed for low-latency and high-bandwidth environments, perform poorly in underwater environments due to long delays and high packet loss. However, basic datagram protocols lack reliability, packet ordering, and congestion control, making them insufficient for scenarios where end-to-end data delivery guarantees are required. To overcome this, reliability mechanisms can be introduced at the application layer, enabling those protocols to maintain the efficiency and simplicity of datagram-based communication while ensuring end-to-end reliable data transmission.

The goal of this paper is to develop and evaluate a complete and optimized application layer protocol that enables reliable file transmission over Underwater Acoustic Networks (UANs). Our protocol operates independently of the underlying link-layer and routing protocols, assuming an end-to-end communication path, and handling retransmissions and data ordering entirely at the application layer. This modularity makes it well suited for deployment in multi-hop networks. Although TCP tunneling can be used for data transfer [9], for instance using protocols like SCP or SSH [10], the initial handshake and encryption overhead associated with these mechanisms is generally too high for underwater acoustic communications. For this reason, protocols requiring persistent secure connections are generally unsuitable for such settings. Our approach is based on a lightweight UDP-based protocol, called the Trivial File Transfer Protocol (TFTP) [1], which we adapted for UANs and then extended with additional mechanisms to ensure reliability and efficiency in challenging conditions.

In particular, we introduce an optimization to the Go-Back-N (GBN) Automatic Repeat Request (ARQ) strategy at the receiver side to improve robustness in multi-hop scenarios.

The remainder of the paper is organized as follows. Section II-A provides an overview of the basic TFTP mechanisms, while Section II-B outlines the modifications made to adapt the protocol for UANs. The general multi-hop network setup and the protocol stack employed are presented in Section II-C. Section III presents an analysis of the tests results and evaluates the performance of the adapted protocol. Finally, Section IV provides our concluding remarks and outlines future works.

II. Methods

A. Trivial Transfer File Protocol

The TFTP is a lightweight protocol designed for file transfers over the Internet. It is implemented over UDP and Internet Protocol (IP) (version 4), and is designed to be small and easy to implement. This protocol does not handle packet reordering, but relies on a GBN algorithm, where the sender can transmit multiple packets within a window without having to wait for individual packet acknowledgments. If a timeout occurs before an acknowledgment is received for the oldest packet in the window, all the packets in the window are retransmitted.

TFTP supports several types of packets. Each packet includes a 2-byte header known as the opcode (opc), a numerical value from 1 to 5 that identifies the type of packet sent or received. The protocol defines five distinct packet types: Read Request (RRQ) (1), Write Request (WRQ) (2), Data (DATA) (3), Acknowledgment (ACK) (4), and Error (ERROR) (5). According to the naming convention used by the protocol, we refer to DATA packets as blocks. A transfer is initiated when a sending node, referred to in the rest of the paper as client, issues a request to either write to or read from a remote host, referred to as the server from here onward, and receives a positive response. A request packet must include the opc and two strings: the name of the file to be read or written, and the file transfer mode. The latter can be specified as "netascii," "octet," or "mail." In this work, we focus only on the "octet" mode, which is used for binary file transfer. Request packets may also include optional parameters, each consisting of a key and a value separated by a null byte, with each parameter delimited by an additional null byte. Some of the possible options are listed below.

- Windowsize, the size of the window in blocks. It specifies the number of consecutive blocks that can be sent before waiting for an ACK of the last transmitted block. This option must be followed by an ASCII string representing the number of blocks in the windows. Valid values range from 1 (that corresponds to stop and wait ARQ) to 65535, inclusive.
- Blocksize, the size of the payload of a DATA packet. It specifies the size in bytes of the actual data in each block. This option must be followed by an ASCII string representing the block size. Valid values range from 8 to 65464, inclusive. Note that the upper bound is determined by the maximum UDP datagram size minus the UDP/IP header overhead (68 bytes) and the 4 bytes protocol header. A block consists of 2 bytes for the opc, 2 bytes for the sequence number of the block and n bytes for the payload (Fig. 1a). The block sequence numbers start at one and increment by one for each new block.
- Timeout, the number of seconds to wait before retransmitting. This value defines how long the client has to wait for the next ACK before retransmitting the last window. It must be followed by an ASCII string representing the timeout value. Valid values range between 1 and 255.

(a) Data packet. (b) Acknowledgment packet.

Fig. 1: TFTP packet structure.

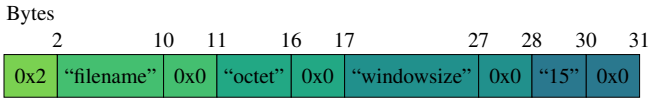

Fig. 2: Example TFTP WRQ packet for transmitting the binary file "filename" using a window size of 15 blocks.

Fig. 2 shows an example of a WRQ. If the server supports option negotiation, it may respond with an Options Acknowledgment (OACK), containing the negotiated options.

After receiving the OACK, the client begins transmitting a window of consecutive blocks, then stops to wait for an ACK confirming the reception of the window. To acknowledge successful reception, the server sends an ACK containing the block number of the last block received in order (Fig. 1b). If the client does not receive the expected ACK within the timeout time, it retransmits the last sent window. When the server receives a block with a sequence number greater than the expected one but not in order, it sends an ACK to the client for the last block received in order. This behavior prompts the client to retransmit the entire window starting from the first unacknowledged block.

B. Underwater TFTP

To adapt TFTP for the underwater network scenario, we introduce several modifications and optimizations, resulting in the development of Underwater Trivial File Transfer Protocol (UW-TFTP). This adapted protocol does not use the traditional UDP/IP protocol stack but rather connects directly to underwater acoustic modems via TCP sockets. Given that the typical Maximum Transmission Unit (MTU) of these modems is limited to 64 bytes, considerable optimization is required for packet size, particularly for WRQ and RRQ packets. We reduced the opc field from 2 bytes to 1 byte, as it only needs to represent five possible packet types. The windowsize option now ranges from 8 to 255, while the blocksize option

ranges from 8 to 247. The upper bound on the blocksize account for the 3 bytes of the UW-TFTP header plus an estimate of 5 bytes for underwater network stack overhead (as we will see in Section II-C). To accommodate the long propagation delays of underwater acoustic communications, we also introduce a multiplier for timeout values. The timeout field in packets still range from 1 to 255, but the actual received timeout duration has to be scaled accordingly. Since all these parameters can be represented within a single byte, we transmit them as binary numbers rather than ASCII strings, significantly reducing packet size. Additionally, we replaced option string keys with single-character identifiers: "W" for windowsize, "B" for blocksize, and "T" for timeout. Given our focus on binary file transfers, the mode string is now optional, with "octet" mode being the default one. Finally, we removed null bytes between option keys and values, further reducing overhead, and set a maximum limit of 45 characters for filenames. Fig. 3 illustrates an example of WRQ packet structure in UW-TFTP.

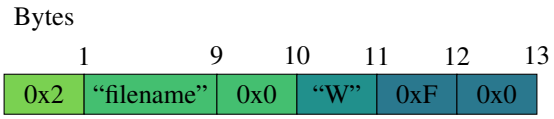

Fig. 3: Example UW-TFTP WRQ packet for transmitting the binary file "filename" using a window size of 15 blocks.

In order to reduce duplicate packet retransmissions caused by missing blocks at the server side, we propose an optimization to UW-TFTP that introduces an out-of-order blocks buffer at the receiver. This improved version will be referred to as UW-TFTP-Buffer (UW-TFTP-B) in the the rest of this paper. In the original version of the protocol, when consecutive blocks are missing, the server responds with an ACK for the last in-order block. Instead of discarding subsequent out-of-order blocks within the window, our approach is to store them in a buffer (like in Selective Repeat (SR) ARQ). This allows the server to retrieve missing blocks locally if lost in the next transmission windows, avoiding unnecessary retransmissions from the client. In fact this mechanism significantly reduces redundant data transmission, especially in environments prone to burst errors, such as the UANs considered in this work, where minimizing retransmissions is critical due to the long propagation delays.

C. Multi-hop network scenario

In this work, we evaluate UW-TFTP and compare its performance with the optimized version. Both versions of the UW-TFTP protocol are implemented in Go and are publicly available in a GitHub repository [11]. We conducted our tests on a linear multi-hop network consisting of three nodes. Specifically, the server is located 2 km from an intermediate relay node, which is 3 km away from the client, as shown in Fig 4.

The physical and channel layers of the network are emulated using the EvoLogics S2C D-MAC Emulator [12],

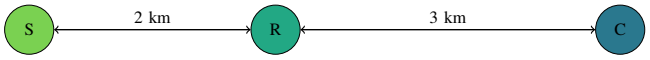

Fig. 4: Multi-hop network topology. S represents the server, R the relay node and C the client.

which enables the use of simulated S2C modems [13] over a simulated acoustic channel. It is worth noting that we connect to the emulator over the Internet, which introduces a variable delay that would not occur with actual modems. Between the modems and the UW-TFTP application we insert a full protocol stack (Fig. 5) implemented using the DESERT Underwater framework for underwater network simulation and experimentation [14], freely available at [15]. This stack provides an application layer module called `uwApplication`, which listens to a specific port for incoming and outgoing packets to and from external applications. The UW-TFTP client is configured to send windows of size 13 blocks (denoted in the following as W_s), each with a blocksize (B_s) of 56 bytes plus a 3-byte header. An additional 5-byte header is added by the DESERT protocol stack, resulting in a total packet size (P_s) of 64 bytes, which matches the MTU supported by the S2C modems used in instant message mode. This protocol stack also includes a custom datagram-based module called `uwUDP`, which mimics the behavior of UDP, and a static routing module (`uwStaticRouting`) coupled with `uwIP` that mimics IPv4 addressing by assigning addresses from 1 to 254 to nodes to enable underwater routing.

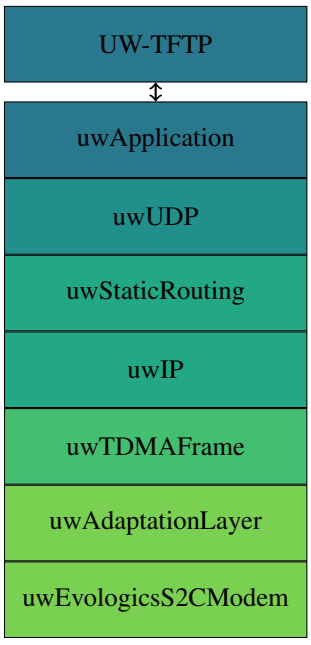

Fig. 5: Protocol stack used in the DESERT Framework.

At the Media Access Control (MAC) layer, we use a super Time Division Multiple Access (TDMA) protocol called `uwTDMAFrame` [16], which is configured to maximize utiliza-

tion of the available bitrate, computed as

$$R_b = \frac{P_s \cdot 8}{t_{blk}} = 394 \text{ bps}, \tag{1}$$

where t_{blk} is the transmission duration of a block and is equal to 1.3 s. Specifically, we divide a superframe of duration t_f into equal length time slots, each corresponding to the transmission duration of a single block, t_{blk}. We allocate W_s consecutive slots plus one guard slot to the client node, allowing it to transmit an entire window of blocks. Similarly, the relay node uses the same number of slots to forward the blocks to the server. Then, both server and relay are assigned one slot each, plus one guard slot, to send an ACK back to the client. In general, in a network with N nodes there are $(W_s + 1)(N - 1)$ slots to forward the window to the server and $2(N-1)$ slots to send the ACK back to the client. Thus, the total number of slots in a superframe is given by $(W_s+3)(N-1)$. In our scenario with three nodes, the overall slot allocation is shown in Fig. 6.

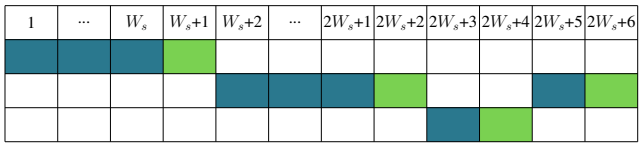

Fig. 6: Superframe slot allocation. From top to bottom, each row represents respectively: slot numbers, client, relay and server node. In blue are depicted transmission slots and in green guard ones.

The maximum achievable throughput on a single link is given by the ratio between the blocksize (B_s) and the transmission time of a packet (P_s/R_b). Given that there is one hop between the client and the server (two links), the maximum achievable throughput is given by

$$T_M = \frac{R_b \cdot B_s}{P_s \cdot (N-1)} = 172 \text{ bps}, \tag{2}$$

where N is the number of nodes in the network. Eventually, the throughput obtained by our protocol is

$$T = \frac{W_s \cdot B_s \cdot 8}{t_f} = 140 \text{ bps}, \tag{3}$$

which gives a channel utilization of

$$\eta\% = \frac{T}{T_M} \cdot 100 = 83\%. \tag{4}$$

As shown in Fig. 7, for values of W_s greater than 13, the throughput begins asymptotically reaching its maximum value, resulting in minimal improvements in channel utilization. Moreover, it increases the number of retransmitted packets in the presence of errors (e.g., lost packets). For this reason, we choose a windowsize (W_s) of 13, meaning a total of 32 slots. Multiplying the number of slots by the block duration (t_{blk}), we have a superframe (t_f) of length 41.6 s.

Finally, an adaptation layer and the S2C modem drivers are used to translate packets between the DESERT simulated environment and actual bitstreams, enabling communication with

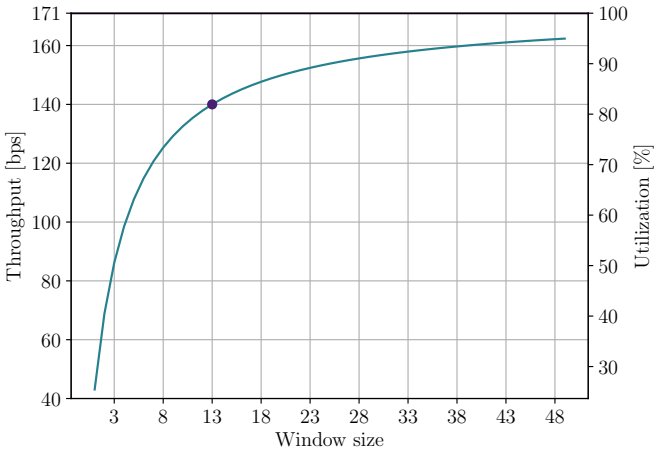

Fig. 7: Throughput (bps) and channel utilization (%) versus the window size. The purple dot represents the chosen windowsize.

the emulated modem interface. Table I provides a summary of the network configuration parameters.

TABLE I: Network settings

Parameter	Value
Number of nodes (N)	3
Window size (W_s)	13
Bitrate (R_b)	394 bps
Blocksize (B_s)	56 B
Packet size (P_s)	64 B
Transmission duration (t_{blk})	1.3 s
Frame duration (t_f)	41.6 s
Slots per frame	32

III. Results

In order to compare the performance of UW-TFTP and UW-TFTP-B, we set up a transmission scenario where a binary file of 10 kilobytes is sent from the client to the server in a multi-hop linear network, as shown in Fig. 4. Each transmission begins with a WRQ packet sent from the client, containing the filename of the file being transferred, a windowsize of 13 blocks, a blocksize of 56 bytes and a timeout of 50 s which include the whole frame duration plus a guard time. The server then confirms these options with an OACK packet. In addition, both the client and the server have a maximum retransmission count set to 10. In all the considered cases, the initial exchange lasts an entire frame (t_f), even though only 4 of the 32 available slots are actually used.

The results, averaged over ten runs, are presented in Fig. 8, with three different values of Packet Error Rate (PER) (5%, 25% and 50%) considered. Specifically, we measure the total time required to successfully transmit an entire file. To simulate actual underwater network conditions, we introduce packet errors, by randomly dropping blocks at the receiver side using a fixed uniform probability distribution. Additionally, we assume that the initial handshake never fails, since for very

large files this added delay becomes negligible compared to the overall transmission time. Although these simplifications do not reflect the highly variable nature of an actual underwater channel, they are useful for showing the difference in performance between the two versions of the protocol.

With a PER of 5%, the performance of the two versions is nearly identical, with both achieving an average transmission time of approximately 900 s (15 minutes). For reference, in the absence of errors, the average time required to successfully transmit the file is about 11 minutes. This lower bound is determined by the nature of the GBN protocol, which causes the client to remain idle after sending a packet window until it receives an ACK from the server. As PER increases, the performance of UW-TFTP worsens significantly. For a 25% PER, the average transmission time goes up to 2100 s (35 minutes), representing a 133% increase compared to the 5% error scenario. At 50% PER, we have a quadratic increase over the previous case, with an average transmission time exceeding 4000 s (more than one hour), marking a 344% increase from the baseline.

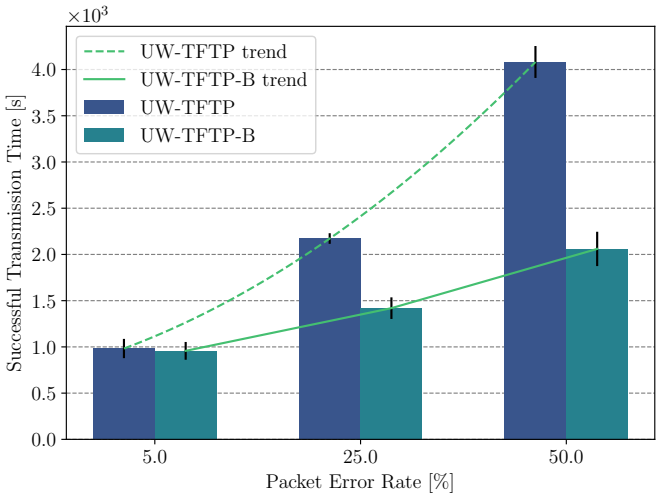

Fig. 8: Transmission time (in seconds) for a 10 kB file against the packet error rate. The green lines indicate the trend of transmission time relative to the PER (in percentage).

In contrast, UW-TFTP-B shows a much more resilient behavior with increasing PER. In particular, the transmission time increases approximately linearly as PER increases. At 25% PER, the average transmission time is 1400 s (23 minutes), which is 33% lower than that of UW-TFTP and only 55% higher than in the case with 5% PER. Finally, at 50% PER, the optimized version completes the transmission, on average, in 2000 s (33 minutes), achieving a 50% reduction in transmission time compared to the basic version. It also shows a slight reduction in time compared to the unoptimized version with 25% PER. Furthermore, it shows an increase of 120% from the 5% PER case.

A. *Water tank test*

To validate the protocol with actual modems, we set up an experimental testbed in a water tank measuring $68 \times 62 \times 52$ cm^3. In this experiment, two acoustic transducers are placed approximately 30 cm apart, and are both operating at a transmission frequency of 40 kHz. These transducers are connected to two low-cost, software-defined acoustic modems known as Subsea acoustic Modem (SuM) [17], as shown in Fig. 9.

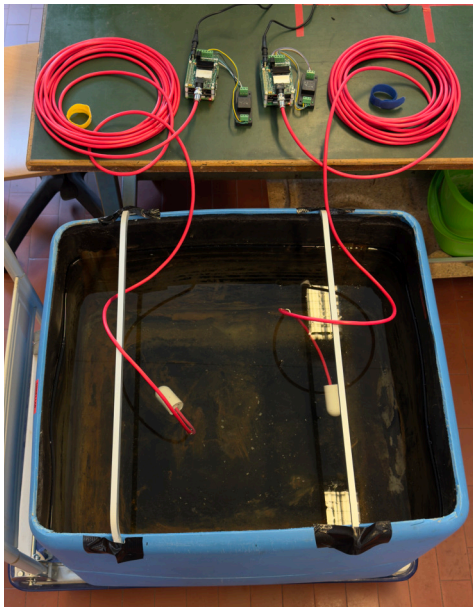

Fig. 9: Water tank setup. The acoustic modem on the left represents the server, while the client is positioned on the right. Both modems are connected to transducers placed inside the water tank.

The protocol stack has the same structure as depicted in Fig. 5, with the only difference being the physical layer, which now integrates the SuM modem drivers. These modems offer a TCP socket interface, allowing for a local connection over Wi-Fi. The transmission duration for each block is 1.5 s, and based on the rationale provided in Sec.II-C, a window size of 13 is used, resulting in a total number of TDMA slots of 18 and a total superframe duration of 27 s. Note that, unlike in the previous scenario, two guard slots are used instead of one to better handle multi-path effects in this confined environment.

The results, averaged over ten runs, are shown in Fig. 10. Specifically, we present the time difference between UW-TFTP and UW-TFTP-B, along with the corresponding PER computed at the application layer as the ratio of packets successfully received by the server and those sent by the client. Although the unoptimized version of the protocol has a slightly lower average PER than the optimized one (13% versus 15%, respectively), it takes, on average, 949 s (15 minutes) to successfully transmit a 10 kilobytes file, compared to 713 s (12 minutes) for UW-TFTP-B.

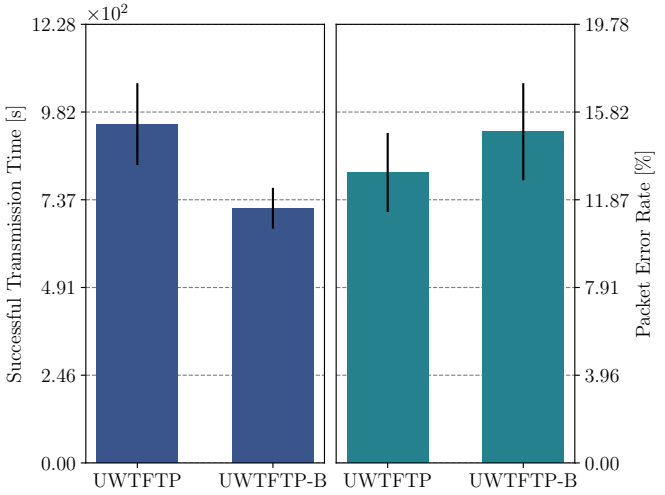

Fig. 10: Comparison of the two protocol versions in the water tank test. The left subplot shows the transmission time (in seconds) for a 10 kB file, while the right subplot shows the corresponding PER (in percentage).

IV. Conclusions and future works

In this work, we presented a lightweight, UDP-based file transfer protocol known as the Trivial File Transfer Protocol (TFTP) and its adapted version for Underwater Acoustic Networks (UANs), called the Underwater Trivial File Transfer Protocol (UW-TFTP). UW-TFTP is an application layer protocol that provides reliable end-to-end file transmission over UANs, independently of the underlying data-link and routing layers. By implementing retransmission logic and loss recovery at the application layer, it ensures robust performance in both constrained environments and multi-hop network scenarios. Our focus was on evaluating and improving the performance of UW-TFTP, which is built upon a Go-Back-N (GBN) Automatic Repeat Request (ARQ) strategy to maintain in-order delivery and avoid the complexity of packet reordering. While GBN simplifies protocol design, it suffers from inefficiencies under high packet error rates, due to its retransmission behavior that discards potentially useful out-of-order packets. To address this limitation, we developed a new version of the protocol, called UW-TFTP-Buffer (UW-TFTP-B), which introduced a buffer for out-of-order blocks at the receiver side. This buffer temporarily stores correctly received packets that arrive out of order and makes them available for reuse during subsequent retransmissions. Our experimental evaluation showed that this optimization leads to substantial reductions in transmission time, particularly under high packet error rates. However, this approach still lead to the retransmission of multiple duplicate packets in the event of packet loss. Specifically, the sender transmits a window of packets and then waits for a cumulative Acknowledgment (ACK) from the receiver, which contains the sequence number of the last correctly received in-order packet. After receiving this ACK, the sender transmits a new window of packets starting from the next in-order packet. We plan to further extend the functionality of UW-TFTP by including Selective Repeat (SR) retransmissions. In this new version of the protocol, the sequence number in the acknowledgment will be replaced by a binary string, where each bit corresponds to a packet in the window. This string allows the sender to identify which packets were successfully received and which were lost. Upon receiving such an ACK, the sender will retransmit only the lost packets, and additionally send new in-order packets.

V. Acknowledgments

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE0000001-program “RESTART”),

References

[1] K. R. Sollins, “The TFTP Protocol (Revision 2).” RFC 1350, July 1992. [Online]. Available: https://www.rfc-editor.org/info/rfc1350, [Accessed: May 9, 2025].

[2] I. F. Akyildiz, D. Pompili, and T. Melodia, “Underwater acoustic sensor networks: Research challenges,” *Ad Hoc Networks*, vol. 3, pp. 257–279, May 2005.

[3] N. L. Guinasso *et al.*, “Observing and forecasting coastal currents: Texas automated buoy system (TABS),” in *MTS/IEEE Oceans 2001. An Ocean Odyssey. Conference Proceedings*, vol. 2, pp. 1318–1322, 2001.

[4] K. Casey, A. Lim, and G. Dozier, “A sensor network architecture for tsunami detection and response,” *International Journal of Distributed Sensor Networks*, vol. 4, pp. 27–42, January 2008.

[5] W. Akram and A. Casavola, “A visual control scheme for AUV underwater pipeline tracking,” in *IEEE International Conference on Autonomous Systems (ICAS)*, 2021.

[6] M. Stojanovic, “On the relationship between capacity and distance in an underwater acoustic communication channel,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 11, pp. 34–43, October 2007.

[7] S. M. Maher, Z. M. Ali, H. H. Mahmoud, S. O. Abdellatif, and M. M. Abdellatif, “Performance of RF underwater communications operating at 433 MHz and 2.4 GHz,” in *International Conference on Innovative Trends in Computer Engineering (ITCE)*, pp. 334–339, 2019.

[8] G. S. Spagnolo, L. Cozzella, and F. Leccese, “Underwater optical wireless communications: Overview,” *Sensors*, vol. 20, p. 2261, April 2020.

[9] M. Hashimoto, G. Hasegawa, and M. Murata, “Experimental evaluation of SCTP tunneling for energy-efficient TCP data transfer over a WLAN,” in *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 833–838, 2014.

[10] K. R. Sollins, “The Secure Shell (SSH) Transport Layer Protocol.” RFC 4253, January 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4253, [Accessed: May 9, 2025].

[11] V. Cimino, “UW-TFTP,” 2025. [Online]. Available: https://github.com/ciminov/uw-tftp.git, [Accessed: May 9, 2025].

[12] Evologics GmbH, “S2C D-MAC emulator,” 2025. [Online]. Available: https://www.evologics.com/emulator, [Accessed: May 13, 2025].

[13] Evologics GmbH, “S2C technology,” 2025. [Online]. Available: https://www.evologics.com/emulator, [Accessed: May 13, 2025].

[14] F. Campagnaro *et al.*, “The DESERT Underwater framework v2: Improved capabilities and extension tools,” in *IEEE UComms*, (Lerici, Italy), Aug. 2016.

[15] SIGNET Group, “DESERT underwater,” 2023. [Online]. Available: https://github.com/signetlabdei/desert_underwater.git, [Accessed: May 9, 2025].

[16] F. Campagnaro, P. Casari, M. Zorzi, and R. Diamant, “Optimal transmission scheduling in small multimodal underwater networks,” *IEEE Wireless Communications Letters*, vol. 8, pp. 368–371, October 2019.

[17] F. Campagnaro, R. Francescon, E. Coccolo, A. Montanari, and M. Zorzi, “A software-defined underwater acoustic modem for everyone: Design and evaluation,” *IEEE Internet of Things Magazine*, vol. 6, pp. 102–107, March 2023.