

SDRT: A Reliable Data Transport Protocol for Underwater Sensor Networks

Peng Xie, Jun-Hong Cui

xp@engr.uconn.edu, jcui@cse.uconn.edu

UConn CSE Technical Report: UbiNet-TR06-03

Last Update: February 2006

Abstract

In this paper, we investigate the reliable data transport problem in underwater sensor networks. Underwater sensor networks are significantly different from terrestrial sensor networks in two aspects: acoustic channels are used for communication and most sensor nodes are mobile due to water current. These distinctions feature underwater sensor networks with low available bandwidth, large propagation delay, highly dynamic topology, and high error probability, which pose many new challenges for reliable data transport in underwater sensor networks. In this paper, we propose a protocol, called segmented data reliable transport (SDRT), to achieve reliable data transfer in underwater sensor network scenarios. SDRT is essentially a hybrid approach of ARQ and FEC. It adopts efficient erasure codes, random forward-error correction codes transferring encoded packets block by block and hop by hop. Compared with traditional reliable data transport protocols, SDRT can reduce the total number of transmitted packets significantly, improve channel utilization, and simplify protocol management. In addition, we develop a mathematic model to estimate the expected number of packets actually needed in both of the single-receiver and multiple-receiver cases. Based on this model, we can set the block size appropriately to enable SDRT to address the mobile nodes. We conduct simulations to evaluate our model and SDRT. The results show that our model can closely predict the number of packets actually needed, and SDRT is energy efficient, and can achieve high channel utilization.

I. INTRODUCTION

Sensor networks have been envisioned as powerful solutions for many applications, such as monitoring, surveillance, measurement, control and health care, etc [15], [22], [17], [26], [11]. Recently, applying sensor networks in aquatic environments (i.e., building underwater sensor networks) has received growing interests [27], [19], [18], [1]. Compared with traditional techniques used in underwater activities such as scientific exploration and commercial exploitation, underwater sensor networks empower us to monitor or detect phenomena more accurately and timely in extended areas.

As in terrestrial sensor networks, for mission critical applications, reliable data transport is demanded in underwater sensor networks. To design a good reliable data transport protocol, a natural concern is the energy efficiency

issue since, similar to terrestrial sensor nodes, nodes in underwater sensor networks are usually also powered by batteries. Maybe it is even more difficult to recharge or replace batteries in harsh aquatic environments. Besides energy consumption, there are several other important issues to consider due to the significant distinctions between underwater sensor networks and terrestrial sensor networks, which we describe as follows.

First, **acoustic channels are used for communication**. In underwater environments, radio does not work well due to extremely limited propagation distance, thus acoustic channels are employed. The propagation speed of acoustic signals in water is about $1.5 \times 10^3 \text{ m/s}$, five orders of magnitude lower than the radio propagation speed ($3 \times 10^8 \text{ m/s}$). In addition, the available bandwidth of underwater acoustic channels is limited and depends on both transmission range and frequency [8], [24], [5], [6]. According to [9], nearly no research and commercial system can exceed $40 \text{ km} \times \text{kbps}$ as the maximum attainable range \times rate product. Moreover, underwater acoustic channels are affected by many factors such as path loss, noise, multi-path, and Doppler spread. All these cause high error probability in acoustic channels.

Second, **underwater sensor nodes are passively mobile**. Empirical observation suggests that water current move at the speed of 3-6 kilometers per hour in a typical underwater condition. In an underwater sensor network, almost all the sensor nodes (except some fixed nodes equipped on surface-level buoys) are mobile due to water current¹. This kind of node mobility results in a highly dynamic network topology.

The above distinctions confer underwater sensor networks new characteristics: large propagation delay, low communication bandwidth, high error probability, and high topology dynamics. All these characteristics pose many new challenges for reliable data transport in underwater sensor networks.

A. Design Challenges

The new characteristics discussed above make many common wisdoms in reliable data transport undesirable for underwater sensor networks.

1) *End-to-End approach does not work well for underwater sensor networks*: Broadly speaking, there are two approaches for reliable data transport, namely end-to-end and hop-by-hop. Some recent studies [25], [10], [23] show that the end-to-end approach is infeasible for sensor networks. This conclusion still holds in underwater sensor networks, and is additionally justified by the high channel error probability and the low propagation speed of acoustic signals: the high channel error probability makes the probability of successfully transferring data from end to end almost approaching to 0, as results in too many retransmissions for a successful packet delivery; the low propagation speed of acoustic signals (1500 m/s) will cause very large end-to-end delay, or RTT, which introduces difficulty for the two ends to manage data transmission timely.

2) *Excessive feedbacks from the receiver are not desirable for underwater sensor networks*: In order to transfer data reliably, many protocols use ARQ (Automatic Repeat Request), in which the receiver needs to send feedbacks

¹In this paper, we mainly discuss “mobile” underwater sensor networks, where sensors are not fixed, and can passively float with water current. This feature is common in a wide range of aquatic applications, such as estuary monitoring and submarine detection.

to the sender, requesting for retransmission if packets are lost. The feedbacks from the receiver is useful for the sender to well control the number of packets to send as many as the receiver needs. However, excessive feedbacks from the receiver is not desirable for underwater sensor networks.

First, many feedbacks cause low channel utilization in underwater sensor networks. Though the bandwidth of acoustic channels is relatively low compared with that of RF channels, the propagation speed of sound (1500 m/s) is even lower compared with that of radio. Thus, even for hop-by-hop communication, RTT is relatively large. If the sender uses feedbacks from the receiver to pace its sending rate, the utilization of communication channels will be very low.

Second, the protocols featured with many feedbacks are not energy efficient in underwater sensor networks. In such protocols, lost feedbacks from the receiver not only degrade channel utilization, but also make the sender to re-transmit packets that are already successfully received. Additionally, in densely deployed sensor networks, there may be multiple sensor nodes in the transmission range of a sender, and they can overhear the transmission of the same packets from the sender. In this scenario, if a receiver requests for the retransmission of lost packets, other receivers have to consume energy to overhear the duplicate packets. Furthermore, the communication of feedbacks also consumes energy. Moreover, the requirement for feedbacks from receivers possibly causes the contention for medium, which is another source of energy waste [28].

Third, in the protocols that require many feedbacks, senders and receivers are overloaded with the feedback-related duties. The protocols have to address many problems resulted from excessive feedbacks such as timeout problem and synchronization between transmissions of data packets and feedbacks. In such protocols, both the senders and receivers have to keep track of lost packets. Sometimes each sender has to keep a timer for each packet per receiver. The problems become more complicated in underwater sensor networks due to the fact that the distance between senders and receivers plays a significant role on propagation delay.

In short, excessive feedbacks cause some problems that are more serious in underwater sensor networks, thus are undesirable for underwater sensor networks.

3) Pure feedback-free protocols are not energy efficient: Some reliable data transport protocols resort to Forward-Error-Correcting (FEC) to overcome the problems caused by excessive feedbacks. In a FEC approach, the sender keeps sending encoded packets, and the receiver keeps receiving encoded packets. At the receiver side, lost packets are ignored, and original data packets can be reconstructed after enough number of encoded packets successfully received. The management of feedback-free protocols using FEC is usually simple in that encoding and decoding are the only additional overhead introduced to the sender and receiver respectively. However, FEC essentially exploits redundancy for reliability. In other words, additional energy is wasted to achieve reliable data transport. In sensor networks, due to energy constraints, pure FEC protocols may not be a good choice.

4) Very large bulk data transmission is not suitable in underwater sensor networks: In the target underwater sensor networks, most nodes are mobile, therefore, there is no long-lasting fixed neighborhood, i.e., the communication time between any pair of sender and receiver is limited. Moreover, the bandwidth of communication channels

is relatively low, and the propagation delay is very high. These factors restrict the sender from sending very large bulk data one time.

B. SDRT Overview and Contributions

Summarizing the above discussions, we want to design a reliable data transport protocol for underwater sensor networks, with the goals of energy efficiency, high channel utilization and simple protocol management. In this paper, we propose a protocol called, Segmented Data Reliable Transport (SDRT). It is essentially a hybrid approach, exploring both FEC and ARQ.

The target sensor networks are relatively dense, with nodes working in low-power transmission range (less than 100 m). The available data rate in such networks is expected to be between 10kbps to 50kbps. SDRT assumes that receivers can detect corrupted packets. This can be done by adding some redundant information in each packet. When a packet is totally lost or unrecoverable from corruption, this packet is treated as lost. In this paper, we are interested in reconstructing lost packets, not error-correction within packets.

In SDRT, the data source first groups data packets into blocks. The data packets are delivered from the source to the destination block by block, and hop-by-hop. An intermediate node encodes each data block using random forward-error correction codes and pumps encoded packets into its channel. When a receiver receives the encoded packets, it decodes and reconstructs the original block. After the reconstruction is done, the receiver encodes the block again and relays the block to the nodes in next hop. For each relay of a block, the sender keeps pumping encoded packets until receiving a positive feedback from next hop.

Our contributions can be summarized as follows:

- We design a reliable data transport protocol, SDRT, which can well satisfy the harsh requirements in underwater sensor networks: efficient random forward-error correction codes enable SDRT to improve channel utilization significantly and relieve both senders and receivers of handling excessive feedbacks as in ARQ-based approaches; more important, SDRT reduces the number of actually sent packets compared with traditional data transport protocols.
- Our SDRT protocol also helps to address the dynamic network topology problem due to sensor node passive mobility. By setting the appropriate block size, SDRT controls the transmission time of each block, therefore, loosens the requirements for the underlying routing protocol to select next hop. The central issue for setting block size is to estimate the number of packets actually transmitted. In this paper, we develop a mathematical model to enable the sender to estimate the expected number of actually needed packets and thus, set the block size appropriately.
- We conduct simulations to evaluate our model and SDRT. The results show that our model can closely estimate the expected number of packets transmitted, and SDRT is energy efficient, and can achieve high channel utilization.

The rest of this paper is organized as follows. We first give a brief review of random forward-error correction codes

in Section II and present our SDRT protocol in Section III. After that, we develop a mathematical model to estimate the expected number of packets transmitted and the block size in Section IV-A. We then conduct simulations to evaluate our model and SDRT, and present the results in Section V. Finally we discuss several seminal proposals on reliable data transport in sensor networks in Section VI. We conclude our paper in Section VII.

II. RANDOM FORWARD-ERROR CORRECTION CODES

Before introducing our codes, we first give some definitions. We call original n messages as data packets, all other messages are called check packets. In this paper, we use packet to refer to a data packet or a check packet. XOR operation is denoted as \oplus . Let P_1 and P_2 be two packets of the same size, then $P_1 \oplus P_2$ is the result of bitwise-XOR'ing packets P_1 and P_2 .

Random forward-error codes are one type of erasure codes, which typically encode a set of the original n packets into a set of N encoded packets, where $N \geq n$, i.e., $l = N - n$ redundant packets are generated. In order to reconstruct n original packets, the receiver has to receive a certain number (larger than n) of encoded messages. The stretch factor, defined as $\frac{N}{n}$, is used to measure the redundancy of erasure codes.

Some practical erasure codes are based on Reed-Solomon [20]. However, Reed-Solomon codes are not suitable for sensor networks due to the limited computation capability of the sensor nodes. Reed-Solomon codes are efficient for small values of n and l . The encoding time is proportional to $n \times l$ and the decoding time is $n \times l'$, where l' the number of redundant packets received. Moreover, the encoding and decoding algorithms require field operations. The computation overhead on the sensor nodes is too high to decode and encode packets using Reed-Solomon codes.

Tornado codes [14][13][4] are preferred in many network applications[4][3][2] in that they can be easily and fast encoded and decoded. The encoding and decoding of Tornado codes only involve XOR operations which are suitable for the limited computation capability of sensor nodes. The encoding and decoding algorithms of Tornado codes are faster than other coding methods such as Reed-Solomon codes. However, Tornado codes use a multi-layer bipartite graph to encode and decode the packets, which results in high computation and communication overhead.

LT codes [12] are very similar to Tornado codes since they have the same encoding and decoding methods. The difference between LT codes and Tornado codes is that they have different degree distributions used in encoding algorithm. Moreover, LT codes use only a two-level bipartite graph in encoding and decoding algorithms. Additionally, LT codes are not systematic codes whereas Tornado codes are.

Both Tornado codes [14] [13] [4] and LT codes [12] are two types of light weight erasure codes in that both encoding and decoding algorithms have low computation requirement, which is suitable for sensor nodes. However, they are inapplicable to sensor networks for the following reasons. First of all, Tornado codes and LT codes are designed for the large number of data packets. This is not true in sensor networks, especially for mobile networks where transmission time between the two nodes are very limited because of the node mobility. Second, both Tornado codes and LT codes generate a bipartite graph. The degree distributions used in Tornado codes and LT codes result

in high-degree nodes in the graph. The large degree of nodes in the graph causes large overhead for each packet. For example, In Tornado Z [14][13], the degree of node in graph can be 200. This is unacceptable in sensor networks. We expect that packet size in sensor networks are relatively limited, therefore, the degree of nodes in the graph has to be very limited.

Inspired by Tornado codes and LT codes, we propose random forward-error correction codes for underwater sensor networks. Random forward-error correction codes have the same encoding and decoding methods as Tornado codes and LT codes. Random forward-error correction codes use one two-layer bipartite as in LT codes. In this sense, random forward-error correction codes are simplified Tornado codes. However, the degree distribution used in encoding algorithm is different from Tornado codes and LT codes. We now brief the encoding and decoding algorithms of random forward-error correction codes as follows.

a) Encoding: The encoding algorithm in random forward error-correction codes is equivalent to constructing a bipartite graph where the nodes in the first level are only randomly connected to the nodes in the second level. In the graph, the nodes in the first level denote original data packets, and the nodes in the second level are the XORs of all their neighbors (nodes) in the first level. The randomly ordered packets including both data packets and check packets constitute the final encoded packets.

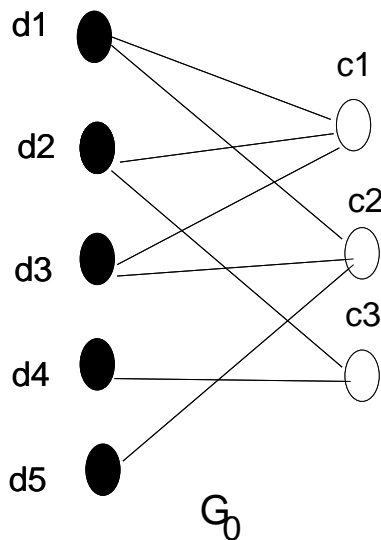


Fig. 1. An example of random forward error correction codes

An example of random forward-error codes is shown in Fig. 1. In this figure, d_1, d_2, \dots, d_5 denote the original data packets. $c_1 = d_1 \oplus d_2 \oplus d_3$, $c_2 = d_1 \oplus d_3 \oplus d_5$, $c_3 = d_2 \oplus d_4$ are the check packets.

In a bipartite graph, an edge is an *edge of degree i* on the left (right) if it is adjacent to a node of degree i on the right (left). For example, in the graph shown in Fig. 1, edges adjacent to node d_1 are edge of degree 2 on the left since node d_1 is a node with degree 2 on the right. A bipartite graph is determined by two edge-degree vectors, namely, left degree vector $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_M)$ and right degree vector $\rho = (\rho_1, \rho_2, \dots, \rho_M)$, where λ_i is the fraction of edges with the degree i on the left and ρ_i the fraction of edges with the degree of i on the right. Thus

for Fig. 1, $\lambda = (\frac{2}{8}, \frac{6}{8})$ and $\rho = (0, \frac{2}{8}, \frac{6}{8})$. Given these two vectors, we can generate the bipartite graph by a simple method suggested in [14].

b) Decoding: The encoding process of random forward error correction codes is to construct a graph from left to right. While, the decoding process of random forward error correction codes is to reconstruct the graph from the right level to the left level.

In each decoding step, for a node on the right side, if all but one of its neighbors are known, then this lost packet can be recovered by XOR all the related packets. For example, in Fig. 1, suppose packet d_1 is lost, we can reconstruct d_1 by $d_2 \oplus d_3 \oplus c_1$. For a graph, its *decoding graph* consists of the nodes on the left side that are lost and not decoded yet, all nodes on the right side and all the edges between them. One decoding step is equivalent to finding a node of degree 1 on the right side in the *decoding graph*, and removing it, its left neighbor and all edges adjacent to its neighbor from this *decoding graph*. This procedure is repeated until all the lost packets are reconstructed or no more node of degree 1 on the right side.

Same as Tornado codes, random forward error-correction codes have encoding time and decoding time proportional to $N \ln(\frac{1}{\epsilon})P$, where P is the length of each packet, N is the number of encoded packets and ϵ is the decoding inefficiency factor, i.e., $(1 + \epsilon)n$ packets are needed for a receiver to reconstruct n packets [14].

From the encoding and decoding algorithms, we can see that the degree distribution in random forward-error correction codes is determined by the two vectors, λ and ρ . Random forward-error correction codes have no requirement on the degree distribution except the minimum degree and maximum degree. As proved in [14], to make the decoding of Tornado codes efficient, the first two elements of the left degree vector should be 0, i.e., $\lambda_1 = \lambda_2 = 0$. In other words, the left degrees of nodes are at least 3. This conclusion is drawn based on combinatorial arguments. It still holds in random forward-error correction codes. In random forward-error correction codes, the graph structure used in encoding is transmitted from the sender to the receiver. The graph structure is included in each packet. Therefore, the maximum degree, M has to be determined by the packet size and very limited. The inclusion of the graph structure in packets also allows the dynamic graph structure which eases the encoding algorithm since the encoding algorithm does not have to address the different graph structure caused by randomness.

III. PROTOCOL DESIGN

The key idea of the segmented data reliable transport (SDRT) protocol is to transfer encoded packets (using random forward-error correction codes), block by block and hop-by-hop. In order to reconstruct the original data packets, the receiver has to receive sufficient encoded packets. Because the node mobility in underwater environment results in short communication time between any pair of sender and receiver, the transmission time for the encoded packets is limited. Thus SDRT has to guarantee that the receiver can receive enough encoded packets in such a limited time interval. By setting the block size n (the number of original data packets in each block) appropriately, SDRT can control the transmission time and allow the receiver to be able to receive enough packets in order to reconstruct original block even in node motion.

A. Segmented Data Reliable Transport (SDRT)

In SDRT, a data source first groups data packets into blocks of size n , i.e., there are n data packets in each block. Then the source encodes these blocks of packets, and sends the encoded blocks into the network. The data packets are forwarded from the source to the destination block by block and each block is forwarded hop-by-hop.

In each hop-by-hop relay, the sender keeps sending the encoded packets until receiving a positive feedback from the receiver. While receiving packets, the receiver tries to reconstruct the original data packets. If the reconstruction is successful, it sends back a positive feedback. On reception of a feedback, the sender stops sending packets, while the receiver encodes the original data packets again and relays them to the next hop.

The operations performed on the sender and the receiver are presented in the following.

Sender

- 1) *Encodes a block using random forward-error correction codes.*
- 2) *Keeps pumping a stream of encoded packets (in a random order), until receiving a positive feedback from the receiver.*

Receiver

- 1) *Keeps receiving packets until it can reconstruct the original data packets, and sends a positive feedback to the sender.*²
- 2) *Encodes the reconstructed packets again and relay them to the next hop.*

From the above description, we can see that SDRT unloads the burden of the sender and the receiver by requiring only one feedback per block. The sender has no additional responsibility except encoding and injecting packets, and the receiver only needs to send one feedback after reconstructing the original packets.

1) Discussions: Multiple receivers In underwater sensor networks, it is common that there are multiple nodes in the transmission range of a sender. All these nodes can overhear the transmission of the same packets and they are potentially capable of relaying the block to the next hop. As pointed in [10], sometimes it is necessary for the underlying routing protocols to provide alternative paths to overcome link failures. Our SDRT protocol can be smoothly integrated with the routing protocols that support alternative paths to improve the network robustness. In fact, our later theoretical analysis and simulation results will show that SDRT performs even better in multiple-receiver case.

Energy consumption In SDRT, a sender keeps sending randomly ordered encoded packets into the network until receiving a positive feedback for the target block. On the one hand, this improves the channel utilization since the sender doesn't need to wait the feedback from the receiver to send subsequent packets. On the other hand, as shown in Section V, SDRT actually reduces the number of packets transmitted, therefore, saves more energy.

Recall that SDRT is a transport protocol not a routing protocol. The highly dynamic network topology issue in underwater sensor networks is a major concern of the underlying routing protocols. However, SDRT contributes

²In real implementation, the feedback can be sent several times to make sure that the sender receives the feedbacks.

to address this problem by setting appropriate block size such that the next hop has sufficient time to reconstruct the whole block even in motion. The block size n is determined by many factors such as the implementation of random forward-error correction codes, the mobile speed of nodes, the bandwidth of communication channels, the propagation speed and the distance between the sender and the receiver. We will present an approach to estimate the block size, n , in section IV-C.

IV. ANALYSIS

In this section, we present an estimation model for the expected number of packets transmitted for both single-receiver case and multiple-receiver cases. We also discuss how to estimate block size for SDRT. Since the encoding and decoding algorithms are randomized.

A. Mathematical Models for Random Forward-Error Correction Codes

We have the following assumptions for the random forward-error correction codes we use. First, we assume packet loss is independent. Second, we assume that the sender sends at most three rounds of encoded packets to guarantee the receiver successfully recovers the original data packets. This can be achieved by selecting appropriate left degree vectors and right degree vectors. Our model can be easily extended to random forward-error correction codes with more than 3 rounds.

We still use bipartite graphs to denote random forward-error correction codes. Let $G = \{V, E\}$ be the bipartite graph, where E is the set of edges and V is the set of nodes in the graph. $V = D \cup C$ and $D \cap C = \phi$, where D is the set of data packets and C is the set of check packets. The edges in G randomly connect the nodes in D and the nodes in C . For each node $v \in C$, we define a box, $b_v = \{u | u = v, \text{ or } uv \in E\}$. Thus, for each node $v \in C$, there is a corresponding box, which is the set of this node and its neighbors. There are $|C|$ totally boxes in graph G . If the left degree of node $v \in C$ is i , we say the degree of b_v is i and the capacity of b_v (i.e., the number of nodes in this box) is $i + 1$. We call the set of all the boxes with the same degree a *cluster*. The cluster of boxes with degree i is denoted as B_i .

For a sender-receiver pair, the receiving process at the receiver side is equivalent to filling packets into $|C|$ boxes. When a check packet is delivered successfully, it is equivalent to putting one packet into one box. If a data packet of degree j is received, it is equivalent to put j packets into j different boxes. For clarity, we refer the packets generated by a data packet or a check packet to *replicas*. The replicas generated by the same data packet are called *sibling replicas*.

We assume that sibling replicas are independent of each other. This assumption is reasonable to some extent since sibling replicas are independently and randomly put into different boxes. When a box of capacity k is filled with $k - 1$ or k replicas, we consider this box full, i.e., all the replicas in this box can be recovered. When the associated packet is reconstructed, all the replicas generated by this packet are considered to be recovered. Thus, in random forward-error correction codes, one full box can potentially cause other boxes full.

We still use λ and ρ to specify the edge-degree vectors, i.e., $\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_M\}$ is the left degree vector and $\rho = \{\rho_1, \rho_2, \dots, \rho_M\}$ is the right degree vector. We use p to denote the erasure probability, n to denote the number of data packets or block size, and $1 + \beta$ to denote the stretch factor. Then $|D| = n$, $|C| = n\beta$, and $N = |C| + |D| = (1 + \beta)n$. It is easily to prove the following lemmas.

Lemma 1: The average degree of data packets, $l = \frac{1}{\sum_j \frac{\lambda_j}{j}}$, and the average degree of check packets, $r = \frac{1}{\sum_j \frac{\rho_j}{j}}$.

Proof: Assume the total number of edges is E , by definition, the number of edges of degree i on the left is $E\lambda_i$, therefore, the number of data packets of degree i is $\frac{E\lambda_i}{i}$, the total number of data packets is $\sum_j \frac{E\lambda_j}{j}$. The average degree of data packets is calculated as:

$$l = \frac{1 \times E\lambda_1}{\sum_j \frac{E\lambda_j}{j}} + \frac{2 \times E\lambda_2}{\sum_j \frac{E\lambda_j}{j}} + \dots + \frac{M \times E\lambda_M}{\sum_j \frac{E\lambda_j}{j}} = \frac{E(\lambda_1 + \lambda_2 + \lambda_3 + \dots + \lambda_M)}{\sum_j \frac{E\lambda_j}{j}} = \frac{E}{\sum_j \frac{E\lambda_j}{j}} = \frac{1}{\sum_j \frac{\lambda_j}{j}}.$$

Similarly, the average degree of check packets is $r = \frac{1}{\sum_j \frac{\rho_j}{j}}$.

Lemma 2: Given the left degree vector λ , right degree vector ρ and the number of data packets n , then $\beta = \frac{\sum_j \frac{\rho_j}{j}}{\sum_j \frac{\lambda_j}{j}}$.

Proof: By lemma 1, we know the average degree of data packets is $l = \frac{1}{\sum_j \frac{\lambda_j}{j}}$ and the average degree of check packets is $r = \frac{1}{\sum_j \frac{\rho_j}{j}}$. Since the number of edges emanating from data packets is equal to the number of edges entering check packets, we get, $n \times l = \beta \times n \times r$. Therefore, $\beta = \frac{l}{r} = \frac{\sum_j \frac{\rho_j}{j}}{\sum_j \frac{\lambda_j}{j}}$.

Lemma 3: The probability that a replica is put into a box in cluster B_i (called the ration of B_i), $q_i = \frac{\rho_i \times (i+1)}{\sum_j (\frac{\rho_j}{j} \times (j+1))}$.

Proof: For a check packet of degree i , the capacity of the corresponding boxes is $i + 1$. The number of boxes of degree i is $E \times \frac{\rho_i}{i}$. Therefore, the total capacity of all boxes is $\sum_j E \times \frac{\rho_j}{j} \times (j + 1)$. The capacity of the cluster with degree i is $E \times \frac{\rho_i}{i} \times (i + 1)$. For any replica, the probability that this replica is put into the cluster with degree i is $\frac{E \times \frac{\rho_i}{i} \times (i+1)}{\sum_j E \times \frac{\rho_j}{j} \times (j+1)}$

$$= \frac{\frac{\rho_i}{i} \times (i+1)}{\sum_j \frac{\rho_j}{j} \times (j+1)}.$$

Lemma 4: For a randomly selected packet, the probability that this packet is a check packet, $p_c = \frac{\beta}{\beta+1}$, the probability that this packet is a data packet, $p_d = 1 - p_c = \frac{1}{\beta+1}$, and the probability that this data packet has degree i is $p_d \times \frac{\lambda_i}{\sum_j \frac{\lambda_j}{j}}$.

Proof: Since the total number of packets is $(\beta + 1)n$, the number of check packets is βn . Therefore, for a randomly selected packet, the probability that this packet is a check packet, $p_c = \frac{\beta}{\beta+1}$ and the probability that this packet is a data packet, $p_d = 1 - \frac{\beta}{\beta+1} = \frac{1}{\beta+1}$. The number of data packets of degree i is $E \times \frac{\lambda_i}{i}$, where E is the total number of edges, the number of data packets is $\sum_j E \times \frac{\lambda_j}{j}$, therefore, the probability that a data packet has degree i is $p_d \times \frac{\lambda_i}{\sum_j \frac{\lambda_j}{j}}$.

From lemma 2, we can see that left degree vector λ , right degree vector ρ and the number of data packets determine random forward-error correction codes. In this rest of this paper, we use l to denote the average degree of data packets, q_i to denote the ration of B_i , p_d to denote the probability of data packets, and p_c to denote the probability of check packets. Then we can define a function $\Theta(x)$ to represent the number of replicas generated by x packets as follows $\Theta(x) = xp_d \times l + x \times p_c$.

B. The Estimation of The Expected Number of Packets Transmitted

In this subsection, we first present a model to estimate the expected number of packets actually needed by the receiver to reconstruct original data packets for the case of single receiver, then we extend the model for the case of multiple receivers.

1) *Single receiver*: Let $f(x)$ be the probability that the receiver reconstructs original data packets given that x packets has been sent out before. Let $d(1|x)$ be the probability that a successfully delivered packet is duplicate given that x packets has been sent out before and $r(1|x)$ be the probability that the receiver can reconstruct original data packets when a non-duplicate packet is successfully delivered given that x packets has been sent out before. Then, we have

$$\begin{aligned} f(x+1) &= pf(x) + (1-p)(d(1|x)f(x) \\ &\quad + (1-d(1|x))r(1|x)) \\ f(0) &= 0 \end{aligned} \tag{1}$$

Calculating $d(1|x)$: When there is no packet sent before, it is impossible that the new packet is duplicate. In the first round of transmission of the encoded packets, the only chance for the new packet is duplicate is that it has been recovered earlier. After the first round of transmission, the new packet can be received or recovered before. If we use $R(x)$ to denote the expected number of packets received or recovered by the receiver given that x packets are sent out before, then we have $d(1|x)$ and $R(x)$ as follows.

$$d(1|x) = \begin{cases} 0 & x = 0 \\ \frac{R(x)-x(1-p)}{N-x(1-p)} & 0 < x \leq N \\ \frac{R(x)}{N} & x > N \end{cases} \tag{2}$$

and

$$\begin{aligned} R(x+1) &= R(x) + (1-d(1|x)) \times (1-p) \\ &\quad \times (D(x,1) + 1) \\ R(0) &= 0 \end{aligned} \tag{3}$$

where $D(x,1)$ is the number of newly recovered packets when a non-duplicate packet is delivered given that x packets are sent out before, and it is evaluated as

$$D(x,1) = (N - R(x) - 1) \times \left(1 - \left(1 - \frac{1}{N - R(x) - 1}\right)^{\delta(x)}\right) \tag{4}$$

In Equation 4, $\delta(x)$ is the number of replicas newly recovered when a non-duplicate packet is delivered given that x packets are set out before, which is computed by algorithm *RecoveryReplica*($x,1$). The second term in the equation is the probability that a lost packets is recovered caused by the recovery of $\delta(x)$ replicas. Here we assume that each of these $\delta(x)$ replicas has the same probability to be associated with the lost packets. The algorithm *RecoveryReplica*($x,1$) is presented as follows.

In random forward-error correction codes, when a box is one replica short, then the lost replica in this box can be recovered. In Algorithm 1, we count the number of boxes with one replica short as the initial recovered replicas.

Algorithm 1 *RecoveryReplica(x,1)*

```

1:  $t_p = N - R(x) - 1$  { $t_p$ : number of lost packets}
2:  $t_r = \Theta(R(x)) + l$  { $t_r$ : number of received replicas}
3:  $l_r = \Theta(N) - t_r$  { $l_r$ : number of lost replicas}
4:  $r = 0$ 
5: repeat
6:   for all  $i$  such that  $\rho_i \neq 0$  do
7:      $l_{r,i} = l_r \times q_i$  { $l_{r,i}$ : number of lost replicas in  $B_i$ }
8:     if  $l_{r,i} \leq 1.4 \times |B_i|$  then
9:        $r_i = l_{r,i} \times (1 - \frac{1}{|B_i|})^{l_{r,i}-1}$  { $r_i$ : number of recovered replicas in  $B_i$ }
10:       $r = r + r_i$ 
11:       $l_r = l_r - r_i$ 
12:     end if
13:   end for
14:    $r_p = (1 - (1 - \frac{1}{t_p})^r) \times t_p$  { $r_p$ : number of recovered packets}
15:    $t_n = \Theta(r_p) - r$  { $t_n$ : number of extra recovered replicas}
16:    $l_r = l_r - t_n$ 
17:    $t_p = t_p - r_p$ 
18:    $r = r + t_n$ 
19: until  $t_n < 1$ 
20: return  $r$ 

```

Then we compute the number of extra replicas recovered due to these recovered replicas, we repeat these steps until no more replica can be recovered. The number of boxes with one replica short is very important for the recovery of lost packets. In Algorithm 1, we make a heuristic about the loop of recoveries, i.e., for each cluster, the loop of recoveries occurs only when averagely less than half of the boxes in the cluster are 2 replicas short. Since in such case, most boxes are likely to be one replica short. As shown in the algorithm, line 9 computes the number of boxes with one replica short for cluster B_i . After getting all the number of recovered replica, line 14 computes the number of packets recovered by these replicas. Then line 15 computes the number of extra replicas recovered. The procedure from line 5 to line 19 is repeated until there is no more extra replica recovered.

Calculating $r(1|x)$: We compute $r(1|x)$ as follows.

$$r(1|x) = \begin{cases} 0 & x = 0 \\ r(1|x-1) + (1 - r(1|x-1))V(x, 1) & \text{otherwise} \end{cases} \quad (5)$$

where $V(x, 1)$ is a function to compute the success probability (i.e., the probability that the original data packets

can be reconstructed) when a non-duplicate packet is delivered given that exact x packets are sent out before, and it is evaluated as

$$V(x, 1) = \sum_j p_j \times \omega(x, j) \quad (6)$$

where p_j is the probability that a packet has degree j , and the check packets can be considered as packets with degree 1. The probability, p_j can be easily computed using Lemma 4. $\omega(x, j)$ is the success probability when the newly added packet has degree j given that x packets have been sent out. It is computed by the algorithm *Recovery*(x, j) as follows.

In the algorithm, the success probability, $\omega(x, j)$ is the product of the success probabilities of all the clusters. Each cluster has two states, namely *recovered*, and *done*. If a cluster is in state *done*, it means that all the boxes in this cluster are full. If a cluster is in state *recovered*, then some lost replicas in this cluster are recovered. If the number of lost replicas is smaller than the number of boxes in a cluster, we assume this cluster is full and the success probability for this cluster is 1. After a series of recoveries, if each box in a cluster is less than 1 replica short, the success probability is the probability that these lost replicas are located in different boxes, which is essentially the bin and ball problem [16], i.e., given m balls and n bins, the m balls are randomly thrown into n bins, the probability that all these balls are in different bins $P_r \leq e^{-\frac{m(m-1)}{2n}}$. Line 21 computes the number of recovered replicas for the clusters whose shortage of replicas per box is less than 2. Line 30 computes the average number of recovered packets caused by the r recovered replicas. Line 32 computes the number of newly recovered replicas due to these recovered packets. The loop of recoveries stops when there is no extra replica recovered.

Now, we can evaluate the probability that a receiver recovers the original data packets when the sender sends out exactly x encoded packets. Let $P(x)$ denote this probability, and it is computed as

$$P(x) = f(x) - f(x - 1) \quad (7)$$

Then, the average number of packets a sender needs to send can be evaluated as

$$E_1 = \sum_{i=0}^{\infty} i \times P(i) \quad (8)$$

2) *Multiple Receivers*: Let R be the number of receivers. When a sender sends one packet, we assume that this packet is transmitted to R receivers independently.

For one receiver, we use random variable X to denote the number of successfully received packets given that k packets have been sent out by the sender. We can present X as $X = \sum_j X_j$, where random variable X_j has value 1 if a packet is successfully delivered, and 0 otherwise. Further, we have notations $P_r(X_j = 1) = 1 - p$ and $P_r(X_j = 0) = p$. Then, X has a binomial distribution, with expected value as $\mu = k(1 - p)$ and standard deviation as $\delta = \sqrt{k \times (1 - p) \times p}$.

For the multiple-receiver case, R receivers are equivalent to R trails of X with the same binomial distribution. If any one receiver can reconstruct the original data packets, thus sending a positive feedback, the sender then stops

Algorithm 2 *Recovery*(x,j)

```

1:  $t_p = N - R(x) - 1$  { $t_p$ : number of lost packets}
2:  $t_r = \Theta(R(x))$  { $t_r$ : number of received replicas}
3:  $l_r = \Theta(N) - t_r - d$  { $l_r$ : number of lost replicas}
4: repeat
5:    $r = 0$ 
6:   for all  $i$  such that  $\rho_i \neq 0$  do
7:      $l_{r,i} = l_r \times q_i$  { $l_{r,i}$ : number of lost replicas in  $B_i$ }
8:     if  $l_{r,i} \leq |B_i|$  then
9:       if  $B_i$  is not marked recovered or done then
10:         $p_i = 1$ 
11:         $r = r + l_{r,i}$ 
12:        mark  $B_i$  done
13:       end if
14:       if  $B_i$  is recovered then
15:         $p_i = e^{\frac{-l_{r,i}(l_{r,i}-1)}{2|B_i|}}$ 
16:         $r = r + l_{r,i}$ 
17:        mark  $B_i$  done
18:       end if
19:       end if
20:       if  $|B_i| < l_{r,i} \leq 2 \times |B_i|$  then
21:         $r = r + l_{r,i} \times (1 - \frac{1}{|B_i|})^{l_{r,i}}$ 
22:         $p_i = 0$ 
23:        mark  $B_i$  recovered
24:       end if
25:       if  $l_{r,i} > 3 \times |B_i|$  then
26:         $p_i = 0$ 
27:       end if
28:     end for
29:      $l_r = l_r - r$ 
30:      $r_p = (1 - (1 - \frac{1}{t_p})^r) \times t_p$  { $r_p$ : number of recovered packets}
31:      $t_p = t_p - r_p$ 
32:      $t_n = \Theta(r_p) - r$  { $t_n$ : number of extra recovered replicas}
33:      $l_r = l_r - t_n$ 
34:   until  $t_n < 1$ 
35:   return  $\prod p_i$ 

```

sending packets. In other words, if we can find the expected maximum value among these R trials, we can utilize the results for one receiver.

We evaluate the expected maximum value of these R trials by the Chebyshev's Inequality [16], i.e.

$$Pr[|X - \mu| \geq t\delta] \leq \frac{1}{t^2} \quad (9)$$

Let $R \times Pr[|X - \mu| \geq t\delta] \leq \frac{R}{t^2} \leq 0.1$, and we get $t \geq \sqrt{10} \times R$. It means that the times that in R trials, X falls in the range, $[\mu + t\delta, k]$ is 0.1. In other words, in R trails, X has deviation less than $t\delta$ for $R - 0.1$ times. We estimate the possible maximum value of X in R trails is $\mu + t\delta$. Therefore, the expected maximum value of X in R trails is $\mu + \frac{t\delta}{2}$.

In the case of R receivers, sending n packets has the same effect as sending $\frac{\mu + \frac{t\delta}{2}}{p}$ in the case of one receiver, i.e., $\frac{\mu + \frac{t\delta}{2}}{p} = E_1$. Substituting μ , δ and t with R , p and n , we get $\frac{k(1-p) + \frac{\sqrt{10R \times kp(1-p)}}{2}}{p} = E_1$. Solving this equation, we obtain k , i.e., E_R as follows

$$E_R = \frac{(\sqrt{Rp} + 4pE_1 - \sqrt{Rp})^2}{4p} \quad (10)$$

From Equation 10, we can easily prove that $E_r \leq E_1$. that is, the sender actually sends fewer packets when there are more than one receivers.

Using the Equation 8 and Equation 10, the sender can determine the block size accordingly. Let R be the number of receivers.

C. Block Size Estimation

As we state earlier, in order to address the node mobility issue in underwater sensor networks, SDRT limits the transmission time by grouping the original data packets into blocks. The block size, n , i.e., the number of data packets is determined by many factors such as the stretch factor, the confidence factor, the mobile speed of the nodes, the bandwidth of communication channels and the propagation speed of sound. We use $1 + \beta$ to denote the stretch factor, bw to denote the bandwidth of communication channels, V to be the propagation speed of acoustic signals. Further, we denote the possible minimum time interval for the communication between a sender and a receiver (supported by the underlying routing protocol) as T_r . Then the block size, n , must satisfy the following inequity:

$$\frac{\gamma \times E_R \times L}{bw} < T_r \quad (11)$$

where L is the packet size and γ is confidence factor. Note that n is actually one important parameter in computing E_R .

If we have $n = 1000$, $\beta = 0.6$, $bw = 50 \text{ kbps}$, $V = 1500 \text{ m/s}$, the erasure probability $p = 0.5$, the packet size $L = 40 \text{ bytes}$, the left degree vector $\lambda = \{0, 0, 0, 1\}$ and the right degree vector $\rho = \{0, 0, 0, 0, \frac{1}{8}, 0, \frac{7}{8}\}$, in the case of single receiver, the average number of packets needed is 4185. If $\gamma = 1$, then the transmission time is less than 26.2 seconds. In other words, a block size of 1000 requires that the minimum time interval for the communication between a sender and a receiver is at least 26.2 seconds.

V. PERFORMANCE EVALUATION

In this section, we evaluate the accuracy of SDRT estimation model proposed in Section IV-A and the performance of SDRT by simulations.

A. Metrics

We define the following metric to measure the performance of SDRT.

1) Goodput: This metric is define as

$$\theta = \frac{\# \text{ of original data packets}}{\text{time to send the total packets}}. \quad (12)$$

2) Sending inefficiency: This metric is defined as

$$\alpha = \frac{\# \text{ of packets sent}}{\# \text{ of original data packets}}. \quad (13)$$

We use inefficiency α is used to measure the efficiency of communication. The smaller the sending inefficiency, the fewer packets are sent into the network, thus, less energy consumption.

B. Accuracy of the Estimation Model

In Section IV-A, we develop a mathematical model to estimate the average number of packets needed for the receiver(s) to reconstruct the original data packets given the erasure probability, random forward-error correction code construction, and the number of receivers. In this subsection, we compare the value calculated by our models with the average value from the simulations.

We have conducted a large number of simulations to evaluate our model under different configurations of random forward-error correction codes and various parameter settings. Due to the space limit, we only show the results for three scenarios in Fig. 2, Fig. 3, and Fig. 4. The mean and standard deviation shown in these figures are based on 100 simulation trials. And in these three sets of simulations, the number of data packets is 1000 and the stretch factor is 1.6.

Fig. 2 shows the results for random forward-error correction codes with uniform degree distribution, e.g., all the left nodes in the bipartite graph have the same degree and so do the nodes on the right side. In this set of simulations, there is only one receiver. We set $\lambda = \{0, 0, 1\}$ and $\rho = \{0, 0, 0, 0, 1\}$. We vary the erasure probabilities from 0.01 to 0.5. From Fig. 2, we can see that the computed value is very close to the average value from the experiments.

Fig. 3 plots the results for random forward-error correction codes with non-uniform degree distribution, $\lambda = (0, 0, 0, 1)$ and $\rho = (0, 0, 0, 0, \frac{1}{8}, 0, \frac{7}{8})$. Again, there is one receiver. From this figure, we can see that though the degree distribution slightly affects the accuracy of our model, the computed value is still close to the average value obtained from the simulations.

Fig. 4 illustrates the accuracy of our model in multiple-receiver case. In this set of simulations, the degree vectors $\lambda = \{0, 0, 0, 1\}$, $\rho = (0, 0, 0, 0, \frac{1}{8}, 0, \frac{7}{8})$, and the number of receivers is 3. This figure tells us that the expected value calculated by Equation 10 could approximate the average number of packets with an error less than 10%.

It is worth pointing out that the simulation results for uniform degree vectors are relatively higher than what we predicts based on our model. This figure conforms the conclusion that Random forward-error correction codes generated by random regular graphs are not optimal [14]. Comparing Fig. 2 and Fig. 3, we can see that different vectors slightly affect the performance of SDRT, but the effect is not significant.

The implementations of random forward-error correction codes in Fig. 3 and Fig. 4 are same except the number of receivers. Comparing these two figures, we can see that actual number of packets sent decreases as the number of receivers increases. This result agrees with Equation 8 and Equation 10. Therefore, SDRT performs better when the underlying routing protocol supports alternative paths.

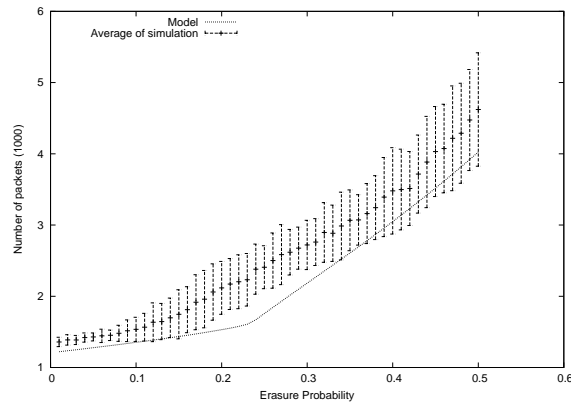


Fig. 2. The computed value vs. simulation results for random forward-error correction codes with uniform degree distribution for single receiver

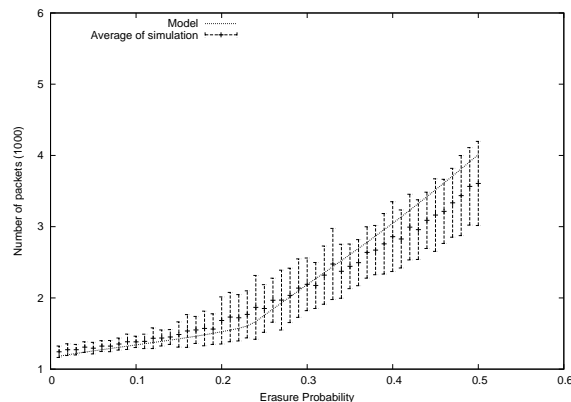


Fig. 3. The computed value vs. simulation result for random forward-error correction codes with non-uniform degree distribution for single receiver

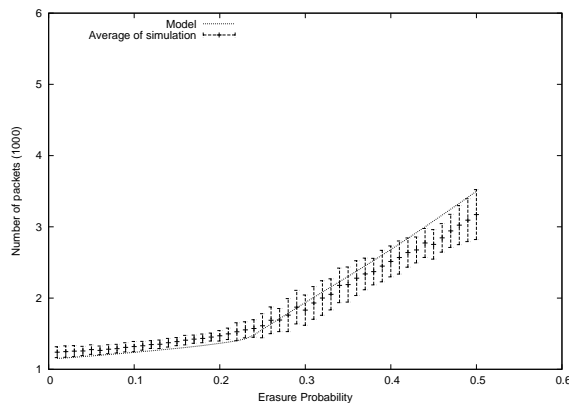


Fig. 4. The computed value vs. simulation result for random forward-error correction codes with non-uniform degree distribution for 3 receivers

C. Goodput Improvement

In this set of simulations, we compare the throughput of SDRT with naive ARQ approach and accumulative ARQ approach in the case of single receiver.

In the naive ARQ approach, the sender sends one packet and waits for the ACK. If the sender does not receive an ACK in RTT, the sender retransmits the packet. If the sender receives an ACK, the sender then sends the next packet.

In the accumulative ARQ, the sender adopts a window to control its transmission, the window size is set to $\frac{1}{p}$, where p is the error probability, i.e., we expect that there is one lost packet in a window. The receiver acknowledges the latest in-order packet in its buffer whenever a new packet arrives. From these ACKs, the sender infers the lost packet, retransmits the packet and slides the sending window accordingly. Since we assume the sensor nodes use half duplex communication. They can not send and receive data at the same time. In the accumulative ARQ, the sender first sends a window of packets and waits for the ACKs. The receiver sends back ACKs in a burst for the packets of the current sending window.

We set the packet size to $40B$ and the bandwidth to $10kbps$. We check the goodputs of naive ARQ and accumulative ARQ under two scenarios, i.e., the distances between the sender and the receiver are 60 meters and 90 meters. The random forward-error correction codes with non-uniform degree distribution, $\lambda = (0, 0, 0, 1)$ and $\rho = (0, 0, 0, 0, \frac{1}{8}, 0, \frac{7}{8})$. We conduct 100 trials and calculate the average of the results as shown in Fig. 5 and 6. Note these two figures are the same except the scale.

From Fig. 5 and Fig 6, we can see that goodput of SDRT is two orders of magnitude higher than that of the naive ARQ and the accumulative ARQ. The naive ARQ performs poorly for two reasons, one is that the sender has low goodput, the another is attributed to the duplicate packets caused by lost ACK. The accumulative ARQ improves the naive ARQ approach in that it improves the goodput and is robust against lost ACKs. Even though the sending window enables the sender to improve the goodput significantly compared with the naive ARQ, its effect is still limited due to the large RTT and channel error which are the major factors affecting goodput. It is worthy

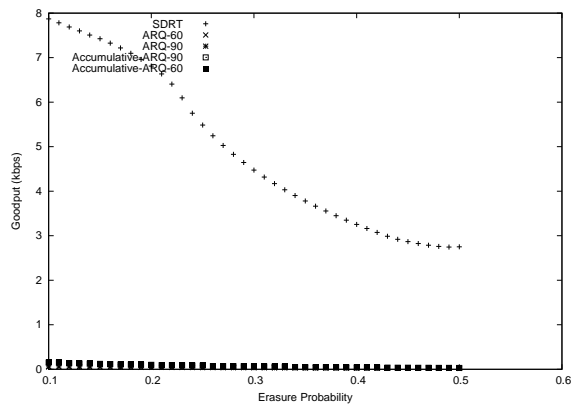


Fig. 5. goodput

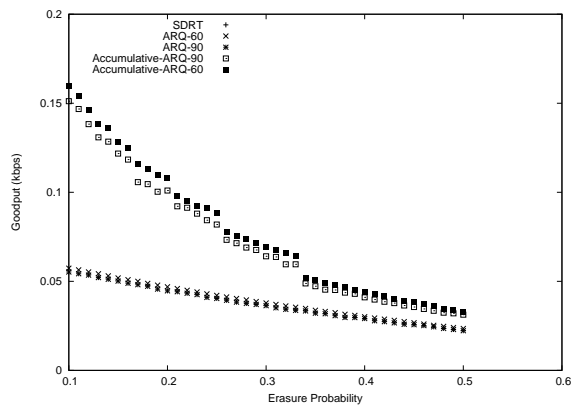


Fig. 6. goodput

noting that distance between the sender and the receiver has no effect on goodput, whereas, it plays a significant role in both the naive ARQ and the accumulative ARQ in terms of goodput.

D. Effect of Random Forward-Error Correction Codes

In this set of simulations, we evaluate the effect of Random forward-error correction codes in SDRT. We compare the performance of four protocols from four paradigms, namely, SDRT, data carousel approach, naive ARQ approach and accumulative ARQ approach. We conduct simulations in both single-receiver and three-receiver scenarios.

For the naive ARQ approach, data packets are not encoded at all. The sender sends a packet and waits for ACK. After some pre-defined time period, if the sender does not receive ACK yet, the sender retransmits the packet. We assume that the data packets and ACK packets have the same erasure probability. In the case of multiple receivers, all the potential receivers can overhear the packet and they can detect lost packets independently. If a node detects a packet loss, this node sends back a negative ACK, otherwise, sends back a positive ACK. The sender keeps retransmitting a packet until receiving positive ACKs from all the receivers and then repeats the procedures for subsequent packets. When we measure the sending inefficiency, we count the number of ACK packets and data packets actually sent as the total number of packets. This approach is actually PSFQ proposed in [25].

The naive ARQ approach is vulnerable to the ACK loss, but the accumulative ARQ approach is more robust against the ACK loss. In the accumulative ARQ approach, data packets are not encoded either. The sender sends a sending window of packets and slides the sending window based on the ACKs from the receiver. The receiver always acknowledges the latest in-order packet whenever a new packet arrives. From the ACKs, the sender can infer which packet is lost and retransmit it. In the case of multiple receivers, each receiver acknowledges the latest in-order packet whenever it receives a packet. The sender uses the lowest sequence number acknowledged as the lower bound for the sending window. The sender retransmits all the lost packets until all the receivers receive all the packets in a sending window. When we measure the sending inefficiency, we count the number of ACK packets and data packets actually sent as the total number of packets. The accumulative ARQ approach is more efficient than naive ARQ approach in that a few lost ACKs can not cause the retransmission of data packets.

For the data carousel approach, data packets are not encoded, either. However, there is no need for feedback in this approach. The sender keeps sending (a block of) data packets in a random order until one of the receivers receives these data packets successfully.

In these simulations, the number of data packets is set to 1000. For SDRT, the block size is set to 1000 packets and stretch factor is 1.6, i.e., there are 1000 data packets and 600 check packets in each block. We use the random forward-error correction code with the left degree vector $(0, 0, 0, 1)$ and the right degree vector $(0, 0, 0, 0, \frac{1}{8}, 0, \frac{7}{8})$.

In the first experiment, the erasure probability is fixed at 0.3 and we vary the network size by changing the number of hops from 1 to 20. We measure the sending inefficiency of all the three approaches under both the cases of single-receiver and three-receiver. The results are shown in Fig. 7. From this figure, we can see that the sending inefficiency of these algorithms are relatively stable. This is attributed to the fact that all these algorithms are based on hop-by-hop forwarding. Among the three approaches, SDRT has the least sending inefficiency, therefore, fewest packets are sent, in all cases. In addition, SDRT and the data carousel approach perform better in the case of three-receiver than in the case of single-receiver. However, this conclusion does not hold in naive ARQ approach (PSFQ) and accumulative ARQ approach: when there are three receivers, the sender actually sends more packets. Multiple receivers send more ACKs for the retransmission of a packet. Moreover, lost ACKs cause the sender to send duplicate packets. The accumulative ARQ approach shows improvements on the naive ARQ approach. Thus, both the naive ARQ approach and accumulative ARQ approach perform worse when the underlying routing protocol supports alternative paths.

To compare the performance of these approaches under different erasure probabilities, in the second experiment, we fix the hop number to 20 and vary the erasure probabilities from 0.1 to 0.5. We check both the single-receiver case and three-receiver case. The results are shown in Fig. 8. From this figure, we observe that when the erasure probability increases, the sending inefficiency also increases. SDRT outperforms other approaches under all erasure probabilities. In the case of single receiver, SDRT reduces inefficiency by half compared with naive ARQ approach and accumulative ARQ approach and even more for the case of three-receiver. From Fig. 8, we can also observe the effect of check packets. The benefit of using check packets is more significant when the erasure probability

increases. Once again, Fig. 8 demonstrates that more receivers actually reduce the sending inefficiency for SDRT and the data carousel approach. But for the ARQ approach, they cause more packets to be sent.

To summarize, random forward-error correction codes reduce the number of packets under various erasure probabilities, thus, reducing energy expenditure.

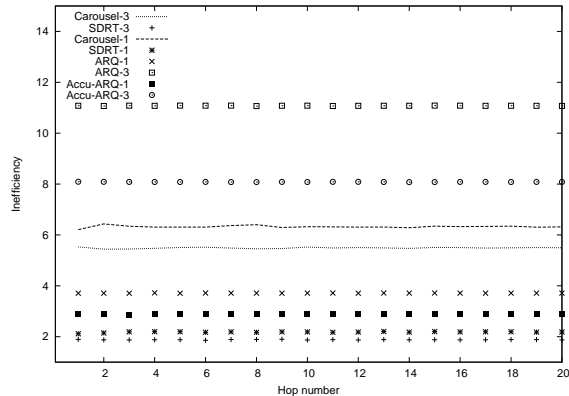


Fig. 7. Inefficiency vs. the network size

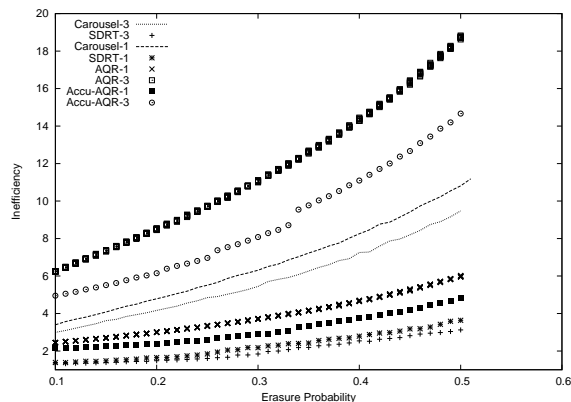


Fig. 8. Inefficiency vs. the erasure probability

E. Summary

From our results and analysis, we can conclude that in SDRT, even though the sender has to send some redundant check packets, the total number of packets that are actually needed are still much smaller than in those approaches (naive ARQ, accumulative ARQ and data carousel) without check packets. The benefit of check packets is even more significant in the case of multiple-receiver, which is typical in sensor networks.

VI. RELATED WORK

In the literature, there are several reliable transport protocols proposed for terrestrial sensor networks [25], [21], [23], [10], [29]. However, due to the significant distinctions between underwater sensor networks and terrestrial

sensor networks, these protocols are unsuitable for underwater sensor networks. We review and discuss each of the protocols as follows.

Wan et al. designed PSFQ (Pump Slowly and Fetch Quickly) in [25], which employs the hop-by-hop approach. In this protocol, a sender sends data packet to its immediate neighbors at very slow rate. When the receiver detect a packet loss, it has to fetch the lost packet quickly. The rate of fetching lost packet is 5 times as fast as the rate of transmitting data. However, PSFQ employs ARQ mechanism in the transmission from hop to hop, it suffers the problems inherited in ARQ paradigms in the case of underwater sensor networks as we discussed in Section I-A.

ESRT (Event to Sink Reliable Transport) is a transport protocol for homogeneous sensor networks, where each node has the same sending rate [21]. This protocol achieves the reliable detection of an event by controlling the sending rate of sources. When the sink detects the congestion in the network, it commands all sources to reduce the sending rate by broadcasting a control message. ESRT uses congestion avoidance techniques to improve reliability instead of providing reliable data transport, which is different from the problem we are investigating. Furthermore, the assumptions such as homogeneous sensor networks and instant feedback from the sink are not applicable to the case of underwater sensor networks.

RMST (Reliable Multi-Segment Transport) and RBC (Reliable Bursty Convergecast) are two other recently proposed reliable data transport protocols for terrestrial sensor networks [23], [29]. RMST [23] is designed for the directed diffusion paradigm [7]. In the proposed architecture, reliable data transfer schemes are implemented at both transport and MAC levels, and ARQ mechanism is adopted. In RBC [29], a window-less block ACK is used to improve the channel utilization, and a differentiated contention control mechanism is employed to reduce the end-to-end delay. Both RMST and RBC have the problems inherited in ARQ paradigm when applied in underwater sensor networks.

In [10], Kim et al. evaluate several methods to improve the reliability of data transport, namely erasure code, retransmission and route fix. All the methods are implemented and evaluated in a real testbed of Mica2Dot. The results shows that each of these methods are effective to overcome some failures. However, the erasure codes evaluated in this work is Reed-Solomon codes [20]. They suggest to look up a table instead to compute for field operation. However, the table can be very large.

VII. CONCLUSIONS AND FUTURE WORK

The unique characteristics of underwater sensor networks pose many new challenges for reliable data transport. To address these issues, we have proposed segmented data reliable transport (SDRT) protocol. In SDRT, data packets are segmented into blocks and encoded using random forward-error correction codes. The application of random forward-error correction codes reduces the number of packets transmitted in networks and relieves both of the sender and receivers the burden of lost packet management, therefore, SDRT is energy efficient and light weight. The appropriate block size enables SDRT to address the highly dynamic network topology problem effectively.

Setting appropriate block size depends on the estimation of the number of packets needed to be sent. We develop

a model to estimate the number of packets actually needed for data recovery. We evaluate the the accuracy of the model and the performance of SDRT by simulations. The results show that our model accurately estimates the number of packets actually sent and SDRT has much higher energy efficiency compared with other types of approaches.

Future Work We plan our future work in two directions. 1) The most energy efficient reliable data transport is to retransmit packets only when necessary, i.e., selective repeat. The current version of SDRT uses “cumulative” ACK. We believe that it is worth investigating how selective repeat help to further save energy expenditure; 2) In this paper, we focus on the design of reliable data transport protocol. Theoretically, it is very interesting and important to optimize random forward-error correction codes under a limited number of data packets and a limited degree of check packets. The approaches proposed in [14] are not applicable to underwater sensor networks since all those approaches are asymptotic behaviors and assumes that the number of data packets is very large and degree of the graph used in encoding and decoding algorithm can be very large.

REFERENCES

- [1] I. F. Akyildiz, D. Pompili, and T. Melodia. Challenges for Efficient Communication in Underwater Acoustic Sensor Networks. *ACM SIGBED Review*, Vol. 1(1), July 2004.
- [2] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery Over Adaptive Overlay Networks. In *ACM SIGCOMM'02*, Pittsburgh,PA, August 2002.
- [3] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. In *INFOCOM'99*, New York, NY, March 1999.
- [4] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Approach to Reliable Distribution of Bulk Data. In *ACM SIGCOMM'98*, Vancouver, British Columbia, August 1998.
- [5] V. Capellano. Performance Improvement of a 50km Acoustic Transmission through Adaptive Equalization and Spatial Diversity. In *Oceans'97*, Nova Scotia, Canada, 1997.
- [6] V. Capellano, G. Loubet, and G. Jourdain. Adaptive Multichannel Equalizer for Underwater Communications. In *Oceans'96*, Ft. Lauderdale, FL, USA, 1996.
- [7] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Roust Communication Paradigm for Sensor Networks. In *ACM International Conference on Mobile Computing and Networking (MOBICOM'00)*, Boston, Massachusetts, USA, August 2000.
- [8] A. Kaya and S. Yauchi. An Acoustic Communication System for Subsea Robot. In *Oceans'89*, volume Vol.3, pages 765–770, September 1989.
- [9] D. B. Kilfoyle and A. B. Baggeroer. The State of the Art in Underwater Acoustic Telemetry. *IEEE Journal of Oceanic Engineering*, OE-25(5):4–27, January 2000.
- [10] S. Kim, R. Fonseca, and D. Culler. Reliable Transfer on Wireless Sensor Networks. In *Frist IEEE International Conference on Sensor and Ad Hoc Communication and Networks (SECON'04)*, Santa Clara, CA, October 4-7 2004.
- [11] B. Krishnamachari, D. Estrin, and S. Wicker. Modeling Data-Centric Routing in Wireless Sensor Networks. In *IEEE INFOCOM 2002*, New York, USA, June 2002.
- [12] M. Luby. LT Codes. In *IEEE FOCS*, pages 271–282, November 2002.
- [13] M. Luby, M. Mitzenmacher, and A. Shokrollahi. Analysis of Random Processes via And-Or Tree Evaluation. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1998.

- [14] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical Loss-Resilient Codes. In *ACM STOC*, pages 150–159, 1997.
- [15] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA'02*, Atlanta, Georgia, USA, September 2002.
- [16] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 2000.
- [17] G. Pottie and W.J.Kaiser. Wireless Intergrated Network Sensors. *Communications of the ACM*, 43(5):551–8, May 2000.
- [18] J. Proakis, E.M.Sozer, J. A. Rice, and M.Stojanovic. Shallow Water Acoustic Networks. *IEEE Communications Magazines*, pages 114–119, November 2001.
- [19] J. G. Proakis, J. A. Rice, E. M. Sozer, and M. Stojanovic. *Shallow Water Acoustic Networks*. Ed. John Wiley and sons, 2001.
- [20] I. REED and G. Solomon. Polynomial Codes over certain finite fields. In *Journal of the Society for Industrial and Applied Mathematics*, pages 8:300–304, June 1960.
- [21] Y. Sankarasubramainam, Ö. B. Akan, and I. F. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *The 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, Annapolis, Maryland, USA, June 2003.
- [22] L. Schwiebert, S. K. S. Gupta, and J. Weinmann. Research Challenges in wireless Networks of Biomedical Sensors. In *ACM SIGMOBILE'01*, Rome, Italy, July 2001.
- [23] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *1st IEEE International Workshop on Sensor Net Protocols and Applications(SNPA)*, Anchorage, Alaska, USA, May 2003.
- [24] M. Suzuki, K. Nemoto, T. Tsuchiya, and T. Nakanishi. Digital Acoustic Telemetry of Color Video Information. In *Oceans'89*, pages 693–696, September 1989.
- [25] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. In *WSNA'02*, Atlanta, Georgia, USA, September 2002.
- [26] J. Warrior. Smart Sensor Networks of the Future. In *Sensor Magazine*, March 1997.
- [27] G. G. Xie and J. Gibson. A Networking Protocol for Underwater Acoustic Networks. In *Technical Report TR-CS-00-02*, Department of Computer Science, Naval Postgraduate School, December 2000.
- [28] W. Ye, J. Heidemann, and D. Estrin. Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM TRANSACTIONS ON NETWORKING*, Vol.12(3), June 2004.
- [29] H. Zhang, A. Arora, Y. ri Choi, and M. G. Gouda. Reliable Bursty Convergecast in Wireless Sensor Networks. In *Mobihoc'05*, Urbana-Champaign, Illinios, USA, May 25-27 2005.