DOS: Distributed On-demand Scheduling for High Performance MAC in Underwater Acoustic Networks

Yibo Zhu, Son N. Le, Zheng Peng, and Jun-Hong Cui
Department of Computer Science & Engineering, University of Connecticut, Storrs, CT, USA
Email: {yibo.zhu, sonle, zhengpeng, jcui}@engr.uconn.edu

UCONN CSE Technical Report: UbiNet-TR13-07
Last Update: July 2013

Abstract—**In underwater acoustic networks (UANs), due to the unique characteristics of acoustic modems such as long preambles and extremely low transmission rates, contentions are usually costly. As a result, both random access and handshake based MAC protocols do not perform as well as expected. A collision-free approach is therefore considered to more likely achieve a better performance. Following this principle, we propose a collision-free scheduling-based MAC protocol, called DOS, for UANs. DOS is a cluster-based protocol, where each cluster head generates on-demand and collision-free schedule for its cluster members independently using local information. Through scheduling, DOS can guarantee collision-free transmissions of both control and data packets. Compared with existing collision-free scheduling MAC protocols, DOS is distributed and on-demand, i.e., it schedules according to nodes' dynamic transmission requests. Further, DOS does not require CDMA or power adjustment for collision resolution. Extensive simulation shows that DOS far outperforms random access and handshake based MAC protocols, and achieves a comparable throughput to a centralized scheduling algorithm.**

I. Introduction

Due to the wide range of applications, such as undersea seismic monitoring, submarine detection, and undersea resource exploration, underwater acoustic network (UAN) has drawn greatly increasing attention in the past decade. To make UAN feasible, however, there are grand challenges to face, especially those introduced by the unique characteristics of underwater communications, such as long propagation delays, and low data transmission rates [1]–[5].

One critical problem we want to focus in this paper is to provide high performance MAC (medium access control) in UANs. In the literature, extensive studies have been conducted for underwater MAC. Random access-based MAC protocols ALOHA and Slotted ALOHA are explored in the underwater environment [6]. To achieve a better performance, some random access-based MAC protocols designed specifically for UANs such as UWAN-MAC [7] and ALOHA-AN [8] leverage short control packets to alleviate collisions. However, these solutions do not work well in multi-hop UANs because they cannot handle the hidden terminal problem.

To provide valid solutions for multi-hop UANs, some handshake-based MAC protocols have been proposed. FAMA [9] elongates the transmission delays of RTS/CTS packets to make MACA [10] work in networks characterized by long propagation delays, such as UANs. However, it is energy-consuming to transmit such long control packets. To improve the energy efficiency, Slotted-FAMA [11] introduces slot scheme to FAMA so that RTS/CTS packets are short again while still able to avoid collisions in UANs. MACA-MN [12], another MACA-based protocol, initiates multiple parallel transmissions with a single handshake to improve the channel utilization.

With the aforementioned MAC protocols, UANs systems are expected to achieve a high performance. However, according to some recent observations, long preambles and extremely low transmission rates introduced by real acoustic modems [13]–[15], make contentions costly, and thus both random access and handshake-based MAC are not as efficient as expected [16].

Specifically, packet transmission delay is much longer than expected due to these modem characteristics, and accordingly collision probabilities of random access-based MAC protocols become very high even for short packets. The performance of handshake-based MAC also degrades because the contentions by "short" control packets become costly. One basic assumption of handshake-based MAC is that control packets are short and their collision probabilities are much lower than regular data packets. Therefore, handshake using short control packets has a much lower collision probability than directly transmitting data packets in a random access manner. However, due to the said acoustic modem characteristics, this assumption does not hold in UANs and handshake becomes inefficient.

Since contentions are costly in UANs, a **collision-free** MAC is promising to achieve a high performance [16]. In addition, traffic rate of each node may be time-varying, so an **on-demand** MAC is desired to satisfy nodes' dynamic traffic requests. Besides, due to long propagation delays, a centralized MAC protocol usually takes a long time to collect the global topology and transmission requests from all nodes and notify them of the schedule, so a **distributed** solution is preferred. To make this MAC protocol compatible with most existing acoustic modems, we do **not require dynamic power adjustment or CDMA** to achieve these goals. Because available bandwidth for real acoustic modems is already extremely narrow, CDMA, whose spreading factor is at least 2, significantly elongates the transmission delays and degrades the system performance [17]. As for dynamic power adjustment, acoustic modems usually support a limited number of predefined power levels which may not match the value selected by the MAC protocol.

In this paper, we present a **D**istributed **O**n-demand **S**cheduling based MAC protocol, DOS, which is able to schedule collision-free transmissions considering spatio-temporal uncertainty [18]. In DOS, nodes are grouped to one-hop clusters. A cluster head collaborates with its neighboring cluster heads to collect the up-to-date schedule of all cluster members' two-hop neighbors in a distributed manner. With the collected information, the cluster head then can independently schedule for its cluster members. DOS ensures collision-free transmissions of both control and data packets, and it does not require special techniques like CDMA to eliminate collisions.

The remainder of this paper is organized as follows. Section II presents the network model and preliminaries. Section III gives an overview of DOS. Section IV introduces a method for finding a collision-free transmission slot. Based on this method, Section V describes how DOS performs distributed on-demand and collision-free scheduling. After that, Section VI gives a discussion, and Section VII evaluates DOS through simulation. Finally, Section VIII summaries the related work, followed by conclusion remarks and directions for future work in Section IX.

II. Network Model and Preliminaries

We target to design a collision-free MAC protocol for multi-hop UANs consisting of homogeneous nodes in horizontal underwater channel environments. The modems used by underwater nodes are characterized by long preamble and extremely low transmission rate issues, which are the general features of current real acoustic modems.

In DOS, all nodes are synchronized via existing synchronization algorithms [19], which can achieve microsecond accuracy. Usually, propagation delays and transmission delays are on the order of seconds in UANs, so it is insignificant to use guard time [20] of centiseconds to cover the synchronization error and clock drift. This allows for a relatively long interval between re-synchronizations, so the synchronization cost is not high in DOS.

We assume a stationary network topology and leave topology management for future work. Since DOS is a cluster-based protocol, it is desirable for DOS to borrow cluster's capability of topology management [21].

After being grouped into one-hop clusters [22], every node v maintains a set $\mathbf{C}_v^H$ of its **cluster heads** [1], and a cluster head v_H knows the set $\mathbf{M}_{v_H}$ of its cluster members including v_H itself. After nodes broadcast their ids for neighbor discovery[2], each node v knows all one-hop neighbors $\mathbf{N}_v$. Specifically, for v_H, $\mathbf{M}_{v_H} = \mathbf{N}_{v_H} + \{v_H\}$. By broadcasting $\mathbf{N}_v$ again, each node v can obtain the set $\mathbf{N}_v^2$ of nodes within its two hops, namely, two-hop neighbors. Then, the set $\mathbf{H}_v$ of v's hidden terminals is $\mathbf{N}_v^2 - \mathbf{N}_v$. During these procedures, node v can measure its propagation delay $T_{v,v'}^P$ to each node $v' \in \mathbf{N}_v$ through time-stamps in overheard packets. In DOS, there are three types of control packets, request packet (REQ), notification packet (NTF), and schedule update packet (SU). A REQ and NTF sent from a node v_i to node v_j are denoted by P_{v_i,v_j}^{req} and P_{v_i,v_j}^{ntf} respectively. A SU packet generated by a cluster head v_i and destined to its neighboring cluster head v_j is denoted by P_{v_i,v_j}^{su}.

[1]A node may locate in overlapped area of multiple clusters, and it has multiple cluster heads.

[2]To guarantee the success, this procedure may repeat for several times.

Another frequently used term in this paper is **slot**, which is categorized into three types, transmission (TX) slot, reception (RX) slot, and interference (IF) slot. A slot s can be represented by a 4-tuple (src, dst, t_b, T_{dur}), which are the sender address, receiver address, beginning time, and duration of s, respectively. For an easier description, we introduce another field t_e, which is the end time of s. Specifically, $s.t_e = s.t_b + s.T_{dur}$. A TX slot s of node v indicates that v is $s.src$ and transmits during interval $[s.t_b, s.t_e]$. A RX slot s of node v means that v is $s.dst$ and receives during interval $[s.t_b, s.t_e]$. An IF slot s of v implies that $v \neq s.src$ & $v \neq s.dst$ and v overhears this packet during interval $[s.t_b, s.t_e]$. We also use notations $\mathbf{S}_{tx}^v$, $\mathbf{S}_{rx}^v$, and $\mathbf{S}_{if}^v$ to denote the sets of node v's TX slots, RX slots, and IF slots respectively.

In addition, we introduce notation $\mathbf{C}_{v_s,v_r}$, which is the conflict slot list for transmissions from node v_s to v_r. In $\mathbf{C}_{v_s,v_r}$, a conflict slot could be a TX slot, a RX slot, a IF slot, or a merged one of overlapped slots of these three types. To be collision-free, v_s's transmission to v_r cannot overlap with any slot in $\mathbf{C}_{v_s,v_r}$.

III. Protocol Overview

In this section, we give an overview of DOS, which performs distributed on-demand scheduling and guarantees collision-free transmissions.

In ad hoc networks, including UANs, a collision is either caused by a one-hop neighbor or a hidden terminal of the sender. As a result, if a node knows the up-to-date schedules of its two-hop neighbors, it can find a collision-free schedule. Based on this fact, in Section IV we design an algorithm for finding a collision-free TX slot considering temporal-spatial reuse in UANs. With this algorithm, DOS can schedule collision-free TX slots for all control packets and data packets.

By using the aforementioned algorithm, we can schedule collision-free TX slots as long as the update-to-date schedule of the sender's two-hop neighbors is available. Now the most significant challenge is to obtain such information in a distributed manner while ensuring on-demand schedule. Our solution DOS leverages one-hop cluster architecture to perform distributed on-demand scheduling.

A. REQ/NTF Exchange

In DOS, each cluster head acts as a local scheduler for its cluster members. In Fig. 1, for example, cluster head v_1 generates schedule for its member v_2. Specifically, before each schedule update, v_1 collects v_2's requests of data transmissions via P_{v_2,v_1}^{req}, and notifies the allocated TX slots to v_2 via P_{v_1,v_2}^{ntf} after its update. With this REQ/NTF exchange, each node can obtain on-demand schedule each time when its cluster head updates schedule. Note that a cluster head also schedules for itself but without such explicit REQ/NTF exchanges. Also note that a node having multiple cluster heads can request TX slots from any cluster head.

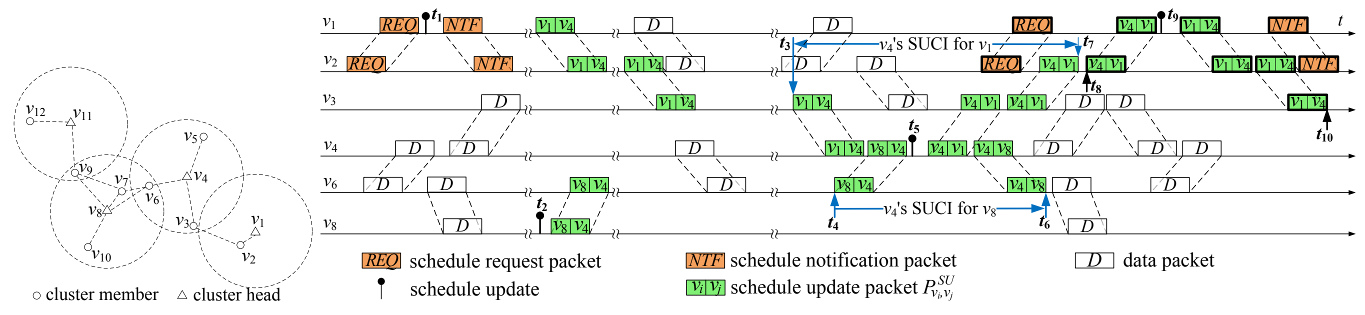

Fig. 1. (a) Topology of a clustered network; (b) An example of schedule. For clarity, we only mark the REQ/NTF interactions between v_1 and v_2, and do not mark IF slots.

B. Neighboring Cluster Head

Since a cluster head generates schedule for its members, each cluster head knows all members' up-to-date schedule. Then, to obtain the up-to-date schedule of all cluster members' two-hop neighbors, a cluster head v_H just needs to receive newly generated schedule from every cluster head v_i that $\mathbf{M}_{v_i} \cap \bigcup_{v \in \mathbf{M}_{v_H}} \mathbf{N}^2_v \neq \varnothing$. Such a cluster head v_i is a **n**eighboring **c**luster **h**ead (NCH) of v_H. For example, in Fig. 1(a), v_4 and v_8 are NCHs to each other because they share a common member v_6. v_1 and v_4 are NCHs to each other as well because v_2 and v_3 are one-hop neighbors. Also, v_4 and v_{11} are NCHs mutually. This is because v_9 in cluster v_{11} has a two-hop neighbor v_6 in cluster v_4. These three examples actually show the three types of NCHs, which are 2-hop, 3-hop, and 4-hop NCHs.

We have designed an algorithm in Section V-A1 to obtain a cluster head v_H's NCH set $\mathbf{C}^{CH}_{v_H}$ and paths $\mathbf{P}_{v_H,v_i}$ from v_H to NCH $v_i \in \mathbf{C}^{CH}_{v_H}$. Along path $\mathbf{P}_{v_H,v_i}$, v_H can send its new schedule to its NCH v_i via $P^{su}_{v_H,v_i}$. In path $\mathbf{P}_{v_H,v_i}$, there are n_{v_H,v_i} nodes v^{v_H,v_i}_k $(1 \leq k \leq n_{v_H,v_i})$, among which v_H and v_i are the first and last one respectively. We also define the k^{th} hop as the transmission from node v^{v_H,v_i}_k to v^{v_H,v_i}_{k+1}, and the corresponding propagation delay is T^{v_H,v_i}_k $(1 \leq k \leq n_{v_H,v_i} - 1)$.

Since nodes in a path belong to 2/3 clusters and every cluster head is only eligible to arrange TX slots of its own members. To forward $P^{su}_{v_H,v_i}$ without collisions, v_H and v_i collaboratively schedule each hop v^{v_H,v_i}_k's forwarding time-point t^{v_H,v_i}_k during their schedule update. Specifically, v_H schedules the first $n_{v_H,v_i} - 2$ hops, which are under v_H's control except for the 3^{rd} hop in the 4-hop case, and v_i always schedules the last hop, which is under v_i's control. For the case of 4-hop path, like the path $v_4 \to v_6 \to v_7 \to v_9 \to v_{11}$ in Fig. 1, the 3^{rd} hop is neither controlled by v_H nor v_i. We define this hop as a ferry hop, which pre-reserves slots with a fixed period when initializing DOS. v_H can pick a reserved one when scheduling a 4-hop path, and then v_H can control the 3^{rd} hop. We will detail ferry hop in Section V.

C. Schedule Update Conflict Interval

To conduct a collision-free scheduling, there is another crucial requirement: a cluster head v_H cannot update before receiving all up-to-date schedule from NCHs having generated schedule after v_H's last update. Otherwise, its new schedule may conflict with these NCHs'. Before introducing a solution to this issue, we first define a term **s**chedule **u**pdate **c**onflict **i**nterval (SUCI). Specifically, for a cluster head v_H and its NCH v_i, v_i's SUCI for v_H, represented by $I^C_{v_i,v_H}$, is the interval between the time-point $I^C_{v_i,v_H}.t_b$ when $v^{v_H,v_i}_{n_{v_H,v_i}-1}$ starts to forward $P^{su}_{v_H,v_i}$ and the time-point $I^C_{v_i,v_H}.t_e$ when $v^{v_i,v_H}_{n_{v_i,v_H}-1}$ receives the entire $P^{su}_{v_i,v_H}$. In Fig 1, for example, $I^C_{v_4,v_1}$ is the interval $[t_3, t_7]$, i.e., $I^C_{v_4,v_1}.t_b$ is t_3 and $I^C_{v_4,v_1}.t_e$ is t_7. Still in this figure, for instance, $I^C_{v_4,v_8}$ is the interval $[t_4, t_6]$ and $I^C_{v_1,v_4}$ is the interval $[t_8, t_{10}]$. When a cluster head v_H searches for a time-point to update schedule, the time-point should be outside of $I^C_{v_i,v_H}$ for all $v_i \in \mathbf{C}^{CH}_{v_H}$. The definition of SUCI yields $I^C_{v_i,v_H}.t_b < t^u_{v_i} < I^C_{v_i,v_H}.t_e$, where is $t^u_{v_i}$ is the time of v_i's coming schedule update. Also note that $t^u_{v_H}$ is out of $[I^C_{v_i,v_H}.t_b, I^C_{v_i,v_H}.t_e]$, so $t^u_{v_H} < I^C_{v_i,v_H}.t_b \Leftrightarrow t^u_{v_H} < t^u_{v_i}$ and $t^u_{v_H} > I^C_{v_i,v_H}.t_e \Leftrightarrow t^u_{v_H} > t^u_{v_i}$.

D. Distributed Schedule Update

After introducing NCH and SUCI, now we can give a big picture of DOS. At the time $t'^u_{v_H}$ when v_H performs its j^{th} schedule update, v_H determines the time $t^u_{v_H}$ of the $j+1^{th}$ update and schedule all control packets related to its $j+1^{th}$ schedule update. These control packets include $P^{req}_{v_m,v_H}$ for each $v_m \in \mathbf{N}_{v_H}$ before $t^u_{v_H}$, $P^{ntf}_{v_H,v_m}$ for each $v_m \in \mathbf{N}_{v_H}$ after $t^u_{v_H}$, $P^{su}_{v_i,v_H}$ from $v^{v_i,v_H}_{n_{v_i,v_H}-1}$ to v_H for each NCH $v_i \in \mathbf{C}^{CH}_{v_H}$ before $t^u_{v_H}$, and $P^{su}_{v_H,v_i}$ from v^{v_H,v_i}_k to v^{v_H,v_i}_{k+1} $(1 \leq k \leq n_{v_H,v_i} - 2)$ for each NCH $v_i \in \mathbf{C}^{CH}_{v_H}$ after $t^u_{v_H}$. In Fig. 1, for instance, at t_1, v_1 decides its next schedule update time t_9 and schedules all packet transmissions marked with bold lines. The schedule of these packets also yields $I^C_{v_H,v_i}$ for each $v_i \in \mathbf{C}^{CH}_{v_H}$. For example, in Fig. 1 $I^C_{v_1,v_4}$ is $[t_8, t_{10}]$ for v_H's update at t_9.

After scheduling these packets, v_H schedules data transmissions. Note that a data TX slot scheduled at $t'^u_{v_H}$ may be later than $t^u_{v_H}$. When it is the time, scheduled during the $j-1^{th}$ update, for v_H to forward packet $P^{su}_{v_H,v_i}$, in this packet v_H includes $t^u_{v_H}$, $I^C_{v_H,v_i}$, and all related newly scheduled TX slots, in which each slot s satisfies $s.t_e \geq t^u_{v_i}$ and $s.src \in \bigcup_{v_m \in \mathbf{M}_{v_i}} N^2_{v_m}$, in $P^{su}_{v_H,v_i}$, and then sends it out. $P^{su}_{v_H,v_i}$ carries $\mathbf{P}_{v_H,v_i}$ and the time $t^{v_H,v_i}_k (1 \leq k \leq \mathtt{min}(2,\, n_{v_H,v_i} - 2))$ when v^{v_H,v_i}_k starts to forward. With this information, v^{v_H,v_i}_k find out the next hop and when to forward. As stated before, the last hop is scheduled by v_i to forward, and then $P^{su}_{v_H,v_i}$ can finally reach v_i.

In the $P^{req}_{v_m,v_H}$ scheduled by v_H's j^{th} update at $t'^{u}_{v_H}$, v_m includes data transmission requests, and then v_H will allocate slots accordingly during its $j+1^{th}$ update at $t^u_{v_H}$. In the $P^{ntf}_{v_H,v_m}$ scheduled at $t'^{u}_{v_H}$, v_H includes all TX slots allocated for v_m during its $j+1^{th}$ update at $t^u_{v_H}$, the time when v_m perform its $j+2^{th}$ schedule update, and the time when v_m forwards the SU generated after $t^u_{v_H}$ to v_H if any.

At $t'^{u}_{v_H}$, v_H has obtained up-to-date schedule of nodes in $\bigcup_{v_m \in \mathbf{M}_{v_H}} N^2_{v_m}$ and $I^C_{v_i,v_H}$ for all $v_i \in \mathbf{C}^{CH}_{v_H}$. Then, v_H can use the method of finding collision-free slot in Section IV to schedule transmissions of all control and data packets. In this way, it can guarantee collision-free.

In a schedule update at $t'^{u}_{v_H}$, v_H first determines a qualified time-point $t^u_{v_H}$ for its next schedule update based on the following **three rules**: $\underline{\mathcal{R}_1.}$ for each v_i that $I^C_{v_i,v_H}.t_e \leq t^u_{v_H}$, v_H is able to find a slot in interval $[I^C_{v_i,v_H}.t_e, t^u_{v_H}]$ for $v^{v_i,v_H}_{n_{v_i,v_H}-1}$ to transmit $P^{su}_{v_i,v_H}$ to v_H; $\underline{\mathcal{R}_2.}$ for each v_i that $I^C_{v_i,v_H}.t_b \geq t^u_{v_H}$, v_H is able to find slots for the first $\min(2,\ n_{v_H,v_i}-2)$ hops of $\mathbf{P}_{v_H,v_i}$ and pick a ferry slot if applicable to forward $P^{su}_{v_H,v_i}$ in interval $[t^u_{v_H}, I^C_{v_i,v_H}.t_b - T^{tx}_{v_H,v_i}]$ where $T^{tx}_{v_H,v_i}$ is $P^{su}_{v_H,v_i}$'s transmission delay; $\underline{\mathcal{R}_3.}$ v_H is able to find slots for each $v_m \in \mathbf{M}_{v_H} - \{v_H\}$ to send a REQ packet to it. With these rules, v_H can determine its schedule update times independently and does not cause schedule update conflicts.

When determining $t^u_{v_H}$, $\mathcal{R}_1$ secures that v_H can receive up-to-date schedule from NCHs before its next update at $t^u_{v_H}$, and $\mathcal{R}_2$ ensures that the NCHs updating later than $t^u_{v_H}$ can get v_H's new schedule too. After $t^u_{v_H}$ is fixed, v_H can schedule $t^{v_H,v_i}_k (1 \leq k \leq \min(2,\ n_{v_H,v_i}-2))$ for each NCH v_i updating earlier than $t^u_{v_H}$. Then, v_H can guarantee that its new schedule generated at $t^u_{v_H}$ can be delivered to a cluster member of v_i. Later on, guaranteed by $\mathcal{R}_1$, v_i can receive this packet before its next schedule update.

In next section, we will describe how DOS works in detail.

IV. Finding a Collision-free Transmission Slot

In this section, based on the assumption that we have update-to-date schedules of nodes in $\mathbf{N}^2_{v_s}$, we design an algorithm to find all collision-free candidate TX slots for a transmission from v_s to v_r. Further, based on this algorithm, we design two algorithms for locating the first and the last qualified collision-free transmission respectively to meet the requirements of DOS. For a better understanding on the algorithm, we begin the description with an example.

A. An Example

As shown in $Part\ A$ of Fig. 2, v_1 has all two-hop neighbors' up-to-date TX slots stored $\mathbf{S}_{tx}$. To get a collision free slot for v_1's transmission to v_2, the sufficient condition is that this new slot does not overlap with v_1's TX/RX slots, v_2's TX/RX/IF slots or v_3's RX slots. To satisfy this condition, we conduct three step conversions from $Part\ A$ to $Part\ D$ in Fig. 2 to get a collision-free slot.

First, for each TX slot $s \in \mathbf{S}_{tx}$, as marked in $Part\ A$, by considering propagation delays, we can find the corresponding RX slot is $[s.t_b + T^P_{s.src,s.dst}, s.t_e + T^P_{s.src,s.dst}]$ at receiver $s.dst$, and IF slots are $[s.t_b + T^P_{s.src,v}, s.t_e + T^P_{s.src,v}]$ for all $v \in \mathbf{N}_{s.src} - \{s.dst\}$, i.e., all $s.src$'s neighbors except for $s.dst$. These slots are marked in $Part\ B$.

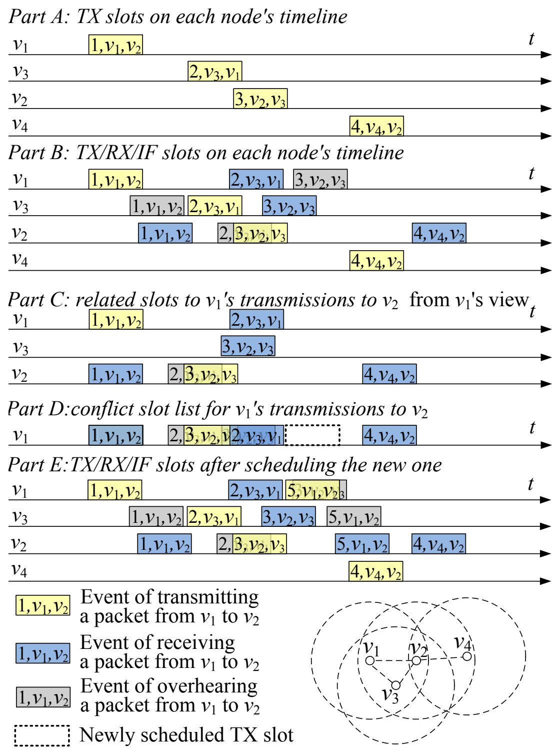
Part A: TX slots on each node's timeline
Part B: TX/RX/IF slots on each node's timeline
Part C: related slots to v1's transmissions to v2 from v1's view
Part D:conflict slot list for v1's transmissions to v2
Part E:TX/RX/IF slots after scheduling the new one
Event of transmitting a packet from v1 to v2
Event of receiving a packet from v1 to v2
Event of overhearing a packet from v1 to v2
Newly scheduled TX slot

Fig. 2. Finding collision-free slots for v_1's transmissions to v_2 considering temporal-spatial reuse

With the slots in $Part\ B$, v_1 is able to check if its new TX slot causes any collision. Specifically, the new TX slot s should meet the following requirements: 1) it does not overlap with any v_1's TX/RX slot; 2) at the receiver v_2, s's corresponding reception interval $[s.t_b + T^P_{v_1,v_2}, s.t_e + T^P_{v_1,v_2}]$ does not overlap with any TX/RX/IF slot of v_2; 3) at each node $v \in \mathbf{N}_{v_1} - \{v_2\}$, s's corresponding IF slot $[s.t_b + T^P_{v_1,v}, s.t_e + T^P_{v_1,v}]$ does not overlap with any of v's RX slots.

However, the method above is still not straightforward for v_1 to identify collision-free slots. To facilitate this procedure, we further convert all related slots to v_1's view, where the related slots are sender v_1's TX/RX slots, receiver v_2's TX/RX/IF slots, and RX slots of all node $v \in \mathbf{N}_{v_1} - \{v_2\}$. Basically, for any slot s of these related slots, we can convert it to $[s.t_b - T^P_{v_1,v}, s.t_e - T^P_{v_1,v}]$ from v_1's view. As long as v_1's new TX slot does not overlap with $[s.t_b - T^P_{v_1,v}, s.t_e - T^P_{v_1,v}]$, the corresponding RX/IF slot will not overlap with slot $[s.t_b, s.t_e]$ at v. Through this conversion, we can obtain the slots shown in $Part\ C$. Essentially, if v_1's new slot does not overlap with any of these slots, it is collision-free.

To more efficiently find a collision-free slot, we merge overlapped slots in $Part\ C$ and get $Part\ D$ in Fig. 2. Since any newly scheduled slot of v_1's transmission to v_2 cannot conflict (overlap) with any of those slots in $Part\ D$, we call the list as conflict slot list for v_1's transmission to v_2, i.e.,

$\mathbf{C}_{v_1,v_2}$. Then, for a slot of duration T_{dur} to be scheduled, any gap between two successive conflict slots in $\mathbf{C}_{v_1,v_2}$ and longer than T_{dur} could be a candidate. For example, we can choose the dotted slot in $Part\ D$ as v_1's new TX slot to v_2. After showing all involved TX/RX/IF slots in $Part\ E$ considering propagation delays, we can see this new TX slot does not cause any collision. In addition, the packet sent in the new slot propagates in channel in parallel with v_2's packet to v_3, which indicates that our algorithm can utilize temporal-spatial diversity to increase the channel utilization.

B. Detailed Description

Through the example above, we have given a big picture of the algorithm for finding all collision-free candidate TX slots. Now, we formally introduce it as Alg. 1.

The input of Alg. 1 are sender v_s, receiver v_r, and all upcoming TX slots $\mathbf{S}_{tx}$ of all nodes in $\mathbf{N}^2_{v_s}$. The output is a conflict slot list $\mathbf{C}_{v_s,v_r}$ for v_s's transmission to v_r. Essentially, this algorithm finishes the procedures for converting transmission slots, like that in $Part\ A$ of Fig. 2, to a conflict slot list, like that in $Part\ D$ of Fig. 2.

Algorithm 1 Form conflict slot list for an unicast

```
Function Name: formCListUcast()
Input: $v_s, v_r, \mathbf{S}_{tx}$
Output: $\mathbf{C}_{v_s,v_r}$
 1: for each node $v \in \mathbf{N}^2_{v_s}$ do
 2:     $\mathbf{S}^v_{tx} \leftarrow \varnothing$, $\mathbf{S}^v_{rx} \leftarrow \varnothing$, $\mathbf{S}^v_{if} \leftarrow \varnothing$;
 3: end for
 4: for each TX slot $s \in \mathbf{S}_{tx}$ do
 5:     Update $\mathbf{S}^v_{tx}, \mathbf{S}^v_{rx}, \mathbf{S}^v_{if}$ for all $v \in \mathbf{N}^2_{v_s} \cap \mathbf{N}_{s.dst}$ by $s$
        using Alg. 2;
 6: end for
 7: Insert all slots in $\mathbf{S}^{v_s}_{tx} \cup \mathbf{S}^{v_s}_{rx}$ into $\mathbf{C}_{v_s,v_r}$ sorted by field
    $t_{start}$ and merge overlapped slots;
 8: for each slot $s \in \mathbf{S}^{v_r}_{tx} \cup \mathbf{S}^{v_r}_{rx} \cup \mathbf{S}^{v_r}_{if}$ do
 9:     $s' \leftarrow s$,  $s'.t_b \leftarrow s'.t_b - T^P_{v_s,v_r}$;
10:     Insert $s'$ into $\mathbf{C}_{v_s,v_r}$ sorted by field $t_b$ and merge
        overlapped slots;
11: end for
12: for each slot $s \in \mathbf{S}^v_{if}$ where $v \in \mathbf{N}_{v_s} - \{v_r\}$ do
13:     $s' \leftarrow s$,  $s'.t_b \leftarrow s'.t_b - T^P_{v_s,v}$;
14:     Insert $s'$ into $\mathbf{C}_{v_s,v_r}$ sorted by field $t_b$ and merge
        overlapped slots;
15: end for
```

In lines 1-6, each TX slot s in $\mathbf{S}_{tx}$ generates the corresponding TX/RX/IF slots of node $v \in \mathbf{N}^2_{v_s} \cap \mathbf{N}_{s.dst}$, which are stored in $\mathbf{S}^v_{tx}$, $\mathbf{S}^v_{rx}$ and $\mathbf{S}^v_{if}$. The concrete conversion method is illustrated in Alg. 2 and will be introduced later. This step completes the conversion from $Part\ A$ to $Part\ B$ in Fig. 2.

The remaining lines in Alg. 1 accomplish the conversion from $Part\ B$ to $Part\ D$ in Fig. 2. Specifically, line 7 merges v_s's TX/RX slots into the conflict slot list $\mathbf{C}_{v_s,v_r}$. Then, lines 8-11 convert all TX/RX/IF slots to v_s's view by shifting these slot by $-T^P_{v_s,v_r}$. Lines 12-15 shift all RX slots of all node $v \in \mathbf{N}_{v_s} - \{v_r\}$ to v_s's view. Also, lines 8-15 merge these converted slots and insert them into $\mathbf{C}_{v_s,v_r}$. Then, a TX slot is collision-free if it is within any gap between two successive slots in $\mathbf{C}_{v_s,v_r}$.

As mentioned above, Alg. 2 helps accomplish the conversion from $Part\ A$ to $Part\ B$ in Fig. 2. It takes a TX slot s as input and calculates all incurred TX/RX/IF slots. Specifically, at any $v \in \mathbf{N}_{s.src}$, slot s incurs either a RX or IF slot after the corresponding propagation delay $T^P_{s.src,v}$. These slots are used to update input $\mathbf{S}^v_{tx}, \mathbf{S}^v_{rx}, \mathbf{S}^v_{if}$, which are also output.

Algorithm 2 Update affected slots by a TX slot

```
Function Name: updateAffSlotsByTxSlot()
Input: $s$, $\mathbf{S}^v_{tx}, \mathbf{S}^v_{rx}, \mathbf{S}^v_{if}$ for all $v \in \mathbf{N}_{s.src}$
Output: $\mathbf{S}^v_{tx}, \mathbf{S}^v_{rx}, \mathbf{S}^v_{if}$ for all $v \in \mathbf{N}_{s.src}$
 1: if $s.src \in \mathbf{N}^2_{v_s}$ then
 2:     $\mathbf{S}^{s.src}_{tx} \leftarrow \mathbf{S}^{s.src}_{tx} \cup \{s\}$
 3: end if
 4: for each node $v \in \mathbf{N}^2_{v_s} \cap \mathbf{N}_{s.dst}$ do
 5:     $s' \leftarrow s$,  $s'.t_b \leftarrow s.t_b + T^P_{s.src,v}$;
 6:     if $s.dst$ is a broadcast address or $s.dst$ is $v$ then
 7:         $\mathbf{S}^{s.dst}_{rx} \leftarrow \mathbf{S}^{s.dst}_{rx} \cup \{s'\}$;
 8:     else
 9:         $\mathbf{S}^v_{if} \leftarrow \mathbf{S}^v_{if} \cup \{s'\}$;
10:     end if
11: end for
```

Till now, we have introduced Alg. 1. Based on it, we are able to find out the first/last available TX slot considering spatial-temporal reuse with Alg. 3 and Alg. 4 respectively. These two algorithms are frequently used in DOS.

Algorithm 3 Find the first available TX slot

```
Function Name: findFtTxSlot()
Input: $v_s, v_r, T_{dur}, t_e, t_l, \mathbf{S}_{tx}$
Output: $t_{tx}$
 1: Obtain $\mathbf{C}_{v_s,v_r}$ by running Alg. 1;
 2: Find the first gap $[t^G_b, t^G_e]$ between two successive slots in
    $\mathbf{C}_{v_s,v_r}$ such that $\mathtt{min}(t^G_e, t_l) - \mathtt{max}(t^G_b, t_e) > T_{dur}$.
 3: if It is able to find such a slot then
 4:     $t_{tx} \leftarrow \mathtt{max}(t^G_b, t_e)$;
 5: else
 6:     $t_{tx} \leftarrow$ INVALID_TIME;
 7: end if
```

Alg. 3 is to schedule a collision-free TX slot of T_{dur} from v_s to v_r, and the obtained slot should be no early than t_e and no later than t_l. In line 2, we traverse each gap in sequence to check if it is larger than T_{dur}. If it does, we are able to schedule a TX slot of duration T_{dur} within this qualified gap. The start time t_{tx} of this slot is determined by lines 3-7. If this procedure fails, t_{tx} is set to an error code.

Alg. 4 is similar to Alg. 3 except that Alg. 4 is to find the last qualified slot. In line 2, we still traverse the gaps between every two successive slots but in a decreasing order, and try to find the last gap which is longer than T_{dur} and within interval $[t_e, t_l]$. After finding such a gap, lines 3-7 can decide t_{tx}.

Algorithm 4 Find the last available TX slot

```
Function Name: findLtTxSlot()
Input: $v_s, v_r, T_{dur}, t_e, t_l, \mathbf{S}_{tx}$
Output: $t_{tx}$
1: Obtain $\mathbf{C}_{v_s,v_r}$ by running Alg. 1;
2: Find the last gap $[t_b^G, t_e^G]$ between two successive slots in $\mathbf{C}_{v_s,v_r}$ such that $\mathtt{min}(t_e^G, t_l) - \mathtt{max}(t_b^G, t_e) > T_{dur}$.
3: if It is able to find such a slot then
4:     $t_{tx} \leftarrow \mathtt{min}(t_e^G, t_l) - T_{dur}$;
5: else
6:     $t_{tx} \leftarrow$ INVALID_TIME;
7: end if
```

V. Distributed On-demand Scheduling

In this section we describe how DOS performs collision-free scheduling in a distributed on-demand manner. DOS has two phases, initialization phase and regular scheduling phase. In the initialization phase, nodes are clustered. Then, each cluster heads discovers its NCHs and the corresponding paths, pre-reserves ferry slots if applicable, performs the first schedule update, and delivers new schedule to its NCHs. After that, DOS enters its regular scheduling phase, during which all transmissions are collision-free.

A. Initialization Phase

After nodes are clustered, each node v knows $\mathbf{N}_v$, $\mathbf{N}_v^2$, $\mathbf{H}_v$, and $\mathbf{C}_v^H$. If v is a cluster head, it also knows $\mathbf{M}_v$. Then, every cluster head identifies its NCHs and the corresponding paths as stated in Section V-A1. After that, as described in Section V-A2, every cluster head pre-reserves ferry slots if applicable. Then, each cluster head conduct its first schedule update and sends related new schedule to its NCHs as stated in Section V-A3. In this phase, there are still collisions. To guarantee that every packet can be correctly delivered in this phase, a sender randomly accesses the channel and re-transmit a packet with exponential back-off until it gets the corresponding acknowledgement.

1) Identifying Neighboring Cluster Heads and Paths: It is easy to prove that **a cluster head and its NCH are 2-4 hops away**. If l_{v_1,v_2} is the hops of the shortest path between v_1 and v_2, then our goal is to prove $2 \leq l_{v_H,v_i} \leq 4$ for any cluster head v_H and its NCH v_i. On one side, one-hop cluster architecture implies that any two cluster heads cannot be one-hop neighbors, i.e., they are at least 2 hops away. Therefore, $2 \leq l_{v_H,v_i}$. On the other side, for a node v_1 and its two-hop neighbor v_2, which are in cluster v_H and v_i respectively, their distance $l_{v_1,v_2} \leq 2$. Also we have $l_{v_H,v_1} \leq 1$ and $l_{v_2,v_i} \leq 1$, where the equality holds when v_1 is v_H and v_2 is v_i respectively. Then $l_{v_H,v_i} \leq l_{v_H,v_1} + l_{v_1,v_2} + l_{v_2,v_i} \leq 4$. Combining l_{v_H,v_i}'s lower bound and upper bound yields $2 \leq l_{v_i,v_j} \leq 4$.

Because NCHs are at least 2 hops apart, a path is required to pass SU packets among them. To this end, we propose Alg. 5 to identify a cluster head's NCHs and the corresponding paths. In Alg. 5, lines 2-5 find out all 2-hop NCHs of v_H and the corresponding paths. Specifically, a 2-hop NCH must be a hidden terminal of v_H. Therefore, all cluster heads in $\mathbf{H}_{v_H}$

Algorithm 5 Identify NCHs and the corresponding paths

```
Function Name: identifyNCHsPaths()
Input: $v_H$, $\mathbf{M}_{v_H}$, $\mathbf{N}_v$ and $\mathbf{H}_v$ for all $v \in \mathbf{M}_{v_H}$
Output: $\mathbf{N}_{v_H}^{CH}$, $\mathbf{P}_{v_H,v}$ for all $v \in \mathbf{N}_{v_H}^{CH}$
1: $\mathbf{N}_{v_H}^{CH} \leftarrow \varnothing$
2: for each cluster head $v \in \mathbf{H}_{v_H}$ do
3:     $\mathbf{N}_{v_H}^{CH} \leftarrow \mathbf{N}_{v_H}^{CH} \cup \{v\}$;
4:     $\mathbf{P}_{v_H,v} \leftarrow \{v_H \rightarrow G_1 \rightarrow v\}$, where $G_1 \in \mathbf{N}_{v_H} \cap \mathbf{N}_v$;
5: end for
6: for each cluster head $v \in \bigcup_{v_m \in \mathbf{N}_{v_H}} \mathbf{H}_{v_m} - \mathbf{N}_{v_H}^{CH}$ do
7:     $\mathbf{N}_{v_H}^{CH} \leftarrow \mathbf{N}_{v_H}^{CH} \cup \{v\}$;
8:     $\mathbf{P}_{v_H,v} \leftarrow \{v_H \rightarrow G_1 \rightarrow G_2 \rightarrow v\}$, where $G_1 \in \mathbf{N}_{v_H} \cap \mathbf{H}_v$ and $G_2 \in \mathbf{N}_{G_1} \cap \mathbf{N}_v$;
9: end for
10: for each cluster head $v \in \bigcup_{v_n \in \bigcup_{v_m \in \mathbf{N}_{v_H}} \mathbf{H}_{v_m}} \mathbf{C}_{v_n}^H - \mathbf{N}_{v_H}^{CH}$ do
11:     $\mathbf{N}_{v_H}^{CH} \leftarrow \mathbf{N}_{v_H}^{CH} \cup \{v\}$;
12:     $\mathbf{P}_{v_H,v} \leftarrow \{v_H \rightarrow G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow v\}$, where $G_2 \in \mathbf{H}_{v_H} \cap \mathbf{H}_v$, $G_1 \in \mathbf{N}_{v_H} \cap \mathbf{N}_{G_2}$, and $G_3 \in \mathbf{N}_v \cap \mathbf{N}_{G_2}$;
13: end for
```

are v_H's 2-hop NCHs, and the path from v_H to a 2-hop NCH v is $v_H \rightarrow G_1 \rightarrow v$, where G_1 is a common one-hop neighbor of cluster v and v_H. For example, in Fig. 1(a), v_4 is a 2-hop NCH of v_8, and a path from v_8 to v_4 is $v_8 \rightarrow v_6 \rightarrow v_4$.

Lines 6-9 identify all 3-hop NCHs of v_H and the corresponding paths. Note that a 3-hop NCH v must be a hidden terminal of v_m where $v_m \in \mathbf{N}_{v_H}$. Based on this clue we can find all 3-hop NCHs. Also note that we need to exclude all 2-hop NCHs, which are stored in $\mathbf{N}_{v_H}^{CH}$, from this case. Path $\mathbf{P}_{v_H,v}$ is discovered as $v_H \rightarrow G_1 \rightarrow G_2 \rightarrow v$. In the path, G_1 is one of v_H's neighbor and a hidden terminal of v, i.e., $G_1 \in \mathbf{N}_{v_H} \cap \mathbf{H}_v$. To make the path connected, G_2 must be a common one-hop neighbor of G_1 and v, i.e., $G_2 \in \mathbf{N}_{G_1} \cap \mathbf{N}_v$. In Fig. 1(a), for example, v_4 is a 3-hop NCH of v_1 and a path from v_1 to v_4 is $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$.

Lines 10-13 give all 4-hop NCHs of v_H and the corresponding paths. A 4-hop NCH must be a cluster head of a node which has at least one hidden terminal in $\mathbf{M}_{v_H}$. Similarly, we need to exclude 2/3-hop NCHs stored in $N_{v_H}^{CH}$. Line 12 only shows the relationship in topology among these 5 nodes in $\mathbf{P}_{v_H,v}$. However, because v_H does not know $\mathbf{H}_v$, this relationship cannot directly give $\mathbf{P}_{v_H,v}$. To calculate $\mathbf{P}_{v_H,v}$, we can first search for a G_1 such that $G_1 \in \mathbf{N}_{v_H}$ and $v \in \bigcup_{v' \in \mathbf{H}_{G_1}} \mathbf{N}_{v'}^{CH}$, i.e., v is a cluster head of one of G_1's hidden terminals. Then, G_3 is this hidden terminal of G_1. Further, G_2 can be any common neighbor of G_1 and G_3, i.e., $G_2 \in \mathbf{N}_{G_1} \cap \mathbf{N}_{G_3}$. In Fig. 1(a), for example, v_{11} is a 4-hop NCH of v_4, and a path from v_4 to v_{11} is $v_4 \rightarrow v_6 \rightarrow v_7 \rightarrow v_9 \rightarrow v_{11}$.

2) Scheduling Ferry Slots: In a path $\mathbf{P}_{v_H,v_i}$, the first $\mathtt{min}(2, n_{v_H,v_i} - 1)$ hops are always under v_H's control while v_i is always able to schedule the last hop. For the 2/3-hop cases, v_H and v_i can collaboratively schedule the whole path. As for a 4-hop path, nevertheless, neither v_H nor v_i can schedule the 3^{rd} hop, assumed to be $v_s \rightarrow v_r$. To make the

3rd hop under control so that $P^{su}_{v_H,v_i}$ can go through $\mathbf{P}_{v_H,v_i}$ as scheduled, we introduce **ferry slot**. Basically, we pre-reserves TX slots from v_s to v_r with a fixed period T_{period}. With these reserved slots, just like a ferry service, v_s periodically offers a chance to 'ship' $P^{su}_{v_H,v_i}$ to its next hop v_r. Inspired by this analog, we call v_s, the 3rd node in a 4-hop path, as a ferry node, and the pre-reserved slots for v_s's transmissions to v_r are referred to as ferry slots.

All ferry slots have the same duration T^{su}_{tx}, a predefined maximum transmission delay of SU packet. This is because their durations are unknown at the time of being reserved. Beside T_{period} and T^{su}_{tx}, a ferry slot F_{v_H,v_i} of path $\mathbf{P}_{v_H,v_i}$ contains 3 more fields: sender v_s, receiver v_r, and the beginning time t_{base} of the first instantiated ferry slot. With these fields, it is straightforward to generate F_{v_H,v_i}'s instantiated slots $\mathbf{S}^{t_b,t_e}_{v_H,v_i}$ located in interval $[t_b, t_e]$. Specifically, $\mathbf{S}^{t_b,t_e}_{v_H,v_j}$ is given as a set $\{$slot s | interval $[s.t_b, s.t_e]$ within $[t_b, t_e]$ where $s.t_b = F_{v_H,v_i}.t_{base} + n \times T_{period}$ (n is a natural number)$\}$. Note all 4-hop paths' ferry slots have the same T_{period}, so that their instances will not collide with each others as long as there is no conflict among their first instances within interval $[0, T_{period}]$.

One of v_s's cluster heads pre-reserves slots for the ferry hop $v_s \rightarrow v_r$. Specifically, v_H notifies one of v_s's cluster head v_k that $v_s \rightarrow v_r$ is the ferry hop of $\mathbf{P}_{v_H,v_i}$. Then, starting from a predefined time-point t_{ferry}, before which both clustering and NCH identifications have been done, each cluster head v_k starts to reserve ferry slots by using Alg. 6. After getting F_{v_H,v_i}, v_k notifies this to v_H, and then v_H can schedule the ferry hop by picking some instances of F_{v_H,v_i}.

Algorithm 6 Scheduling Ferry Slots

```
Function Name: scheduleFerrySlot()
Input: $\mathbf{N}^{CH}_{v_k}$, $\mathbf{F}_{v_k}$, $T_{period}$, $T^{su}_{tx}$
Output: $t_{base}$ of each ferry slot $F_{v_i,v_j} \in \mathbf{F}_{v_k}$
 1: Insert $v'.id < v_k.id$ for each $v' \in \mathbf{N}^{CH}_{v_k}$ into $\mathbf{N}^{CH}_{pre}$
 2: while Have not received scheduled ferry slots from all cluster heads in $\mathbf{N}^{CH}_{pre}$ do
 3:     wait for some delay and then check again
 4: end while
 5: Instantiate TX slots of each $f \in \mathbf{F}_{NCH}$ during $[0, T_{period}]$ and store them in $S_{tx}$
 6: for each $F_{v_i,v_j} \in \mathbf{F}_{v_k}$ do
 7:     $F_{v_i,v_j}.t_{base}$ ← findFirstTxSlot($F_{v_i,v_j}.v_s$, $F_{v_i,v_j}.v_r$, $T^{su}_{tx}$, 0, $T_{period}$, $\mathbf{S}_{tx}$)
 8:     Insert $F_{v_i,v_j}$'s slot during $[0, T_{period}]$ into $S_{tx}$
 9: end for
10: Notify each NCH the related ferry slots
```

To avoid conflicts among ferry slots, as stated in Alg. 6, v_k determines which NCHs have higher priorities to schedule ferry slots in line 1, and these NCHs are stored in $\mathbf{N}^{CH}_{pre}$. Then, in lines 2-4, v_k waits for these NCHs's notifications of their scheduled ferry slots. During this waiting period, all received scheduled ferry slots are stored in $\mathbf{F}_{NCH}$. Then, in line 5, v_k instantiates ferry slots in $\mathbf{F}_{NCH}$ during interval $[0, T_{period}]$. After that, in lines 6-9, v_k finds collision-free slots for each ferry slot. Finally, in line 10, v_k lets each NCH know the related newly scheduled ferry slots. Note that v_k still sends every NCH a ferry slot update packet even if there is no related ferry slot for a NCH, so that v_k's NCHs with larger ids will not be trapped by a dead loop in lines 2-4.

3) The First Schedule Update: Starting from a pre-defined time-point t_{sched}, before which all ferry slots have been scheduled, cluster heads start to perform their first schedule update. Similar to scheduling ferry slots, a cluster head cannot update schedule until it receives SU packets from all NCHs of smaller ids. This can avoid conflicts with NCHs' first schedule update. The scheduling algorithm is same to the one used in the regular scheduling phase, which will be introduced in Section V-B. All scheduled slots in the first schedule update are later than a predefined time-point t_{reg}, which is the starting time-point of the regular scheduling phase and before it cluster heads can get their first schedule update done. By splitting these two phases by a specific time-point t_{reg}, the scheduled slots will not collide with these random access transmissions in the initialization phase. In the first schedule update, a cluster head does not have cluster members' transmission requests, so it does not schedule any data transmission this time. After obtaining new schedule, a cluster head sends its NCHs the related new schedule along the corresponding paths.

B. Regular Scheduling Phase

After the initialization phase is done by t_{reg}, cluster heads start to perform regular scheduling, which is distributed and on-demand, and all transmissions are collision-free.

When a cluster head v_H's schedule update time $t'^{u}_{v_H}$ comes, v_H has obtained requested TX slots $\mathbf{S}^{req}_{v_m}$ for each $v_m \in \mathbf{M}_{v_H}$ from v_m's REQ packets. From the last schedule update, v_H knows $t'^{ntf}_{v_m}$ when to send a NTF packet to v_m after its update at $t'^{u}_{v_H}$. Via $P^{su}_{v_i,v_H}$, v_H extracts $I^C_{v_i,v_H}$ for each NCH $v_i \in \mathbf{N}^{CH}_{v_H}$. In addition to v_i's coming schedule update time $t^u_{v_i}$, v_H also gets v_i's new transmission schedule and inserts them into $\mathbf{S}_{tx}$. Because v_H does not know the durations of REQ, NTF, and SU packets, it uses their predefined maximum durations T^{req}_{tx}, T^{ntf}_{tx} and T^{su}_{tx} for scheduling.

For simplicity, we assume we have inserted instances of ferry slots both in $\mathbf{F}_{NCH}$ and $\mathbf{F}_{v_H}$ into $\mathbf{S}_{tx}$ before finding a collision-free slots in the algorithms in this section. Additionally, whenever a TX slot is scheduled, it is inserted into $\mathbf{S}_{tx}$ immediately by default even we do not explicitly specify in the algorithms. This can ensure that transmission slots to be scheduled do not conflict with this new slot.

Taking these aforementioned information as input, v_H runs Alg. 7 to generate new schedule. As described in Section III, it is critical to first determine $t^{su}_{v_H}$, subject to the three rules in Section III. To do this, in line 1, v_H forms a SUCI list $\mathbf{L}^C$ with $I^C_{v_i,v_H}$ for $v_i \in \mathbf{N}^{CH}_{v_H}$. In $\mathbf{L}^C$, overlapped SUCIs are merged and SUCIs are sorted by their beginning time-points. After that, in lines 2-10, v_H searches each gap $[t^G_b, t^G_e]$ between two successive intervals in $\mathbf{L}^C$ and looks for the first gap good for the next schedule update.

When searching in $[t^G_b, t^G_e]$, for each NCH v_i that $I^C_{v_i,v_H}.t_b \geq t^G_e$, v_H must be able to schedule $t^{v_H,v_i}_k (1 \leq k \leq n_{v_H,v_i} - 2)$ in the interval $[t^G_b, I^C_{v_i,v_H}.t_b]$ subject to

Algorithm 7 Generate new schedule

```
Function Name: genSchedule()
Input: $t'^{ntf}_{v_m}$ and $\mathbf{S}^{req}_{v_m}$ for all $v_m \in \mathbf{M}_{v_H}$; $I^C_{v_i,v_H}$ for all $v_i \in \mathbf{N}^{CH}_{v_H}$; $F_{v_H,v_i}$ for all $v_i \in \mathbf{N}^{CH}_{v_H}$ and $\mathbf{P}_{v_H,v_i}$ is 4-hop; $T^{su}_{tx}$, $T^{req}_{tx}$, $T^{ntf}_{tx}$
Output: $t^u_{v_H}$, $\mathbf{S}^{tx}_{v_m}$, $t^{ntf}_{v_m}$ and $t^{req}_{v_m}$ for all $v_m \in \mathbf{M}_{v_H}$, $t^{v_H,v_i}_k(0 \le k \le n_{v_H,v_i} - 2)$ and $t^{v_i,v_H}_{n_{v_i,v_H}-1}$ for all $v_i \in \mathbf{N}^{CH}_{v_H}$
1:  Form a SUCI list $\mathbf{L}^C$ by $I^C_{v_i,v_H}$ for $v_i \in \mathbf{N}^{CH}_{v_H}$ in an non-decreasing order of $I^C_{v_i,v_H}.t_b$, merge overlapped SUCIs.
2:  for each gap $[t^G_b, t^G_e]$ between each two successive SUCIs in $\mathbf{L}^C$ do
3:      $t^u_{est} \leftarrow \infty$ and run Alg. 8 to update $t^u_{est}$ and schedule non-last hops' forwarding time-points of $SU$ packets with deadline;
4:      $t^u_{v_H} \leftarrow t^u_{est} - T_{SU}$ and run Alg. 9 to schedule $t^{req}_{v_m}$ for each $v_m \in \mathbf{M}_{v_H}$ and $t^{v_i,v_H}_{n_{v_i,v_H}-1}$ for each $v_i \in \mathbf{N}^{CH}_{v_H}$
5:      if all slots searched in Algs. 8, 9 can be found then
6:          break;
7:      else
8:          reset all revised variables to their values before entering this loop;
9:      end if
10: end for
11: Tighten SUCIs by running Alg. 10;
12: For each $v_i$ that $I^C_{v_i,v_H}.t_b > t^u_{v_H}$, $t^{v_H,v_i}_k \leftarrow$ getLtTxSlot($v^{v_H,v_i}_k$, $v^{v_H,v_i}_{k+1}$, $T^{su}_{tx}$, $t^{v_H,v_i}_{k-1} + T^{su}_{tx} + T^{v_H,v_i}_{k-1}$, $\infty$, $\mathbf{S}_{tx}$) for $k = 1$ to $\min(n_{v_H,v_i} - 2, 2)$, where $t^{v_H,v_i}_0 = t^u_{v_H} + T_{SU}$. Get the first instance of $F_{v_H,v_i}$ in $[t^{v_H,v_i}_2 + T^{su}_{tx} + T^{v_H,v_i}_2, \infty]$ if $\mathbf{P}_{v_H,v_i}$ is 4-hop;
13: $t^{ntf}_{v_m} \leftarrow$ findFtTxSlot($v_H$, $v_m$, $T^{ntf}_{tx}$, $t^u_{v_H} + T_{SU}$, $\infty$, $\mathbf{S}_{tx}$) for each $v_m \in \mathbf{N}_{v_H}$;
14: For each $s \in \bigcup_{v_m \in \mathbf{M}_{v_H}} \mathbf{S}^{req}_{v_m}$, find a slot in $[t^{ntf}_{v_m} + T^P_{v_H,v_m} + T^{ntf}_{tx}, \infty]$ by calling findFtTxSlot(). Save the scheduled slots in $\mathbf{S}^{tx}_{v_m}$;
```

$t^{v_i,v_H}_k + T^{v_i,v_H}_k + T^{su}_{tx} \le t^{v_i,v_H}_{k+1} \le I^C_{v_i,v_H}.t_b$. This ensures that all v_H's NCHs which update later than $t^{su}_{v_H}$ can receive v_H's new schedule generated at $t^{su}_{v_H}$. These are implemented in Alg. 8. In Alg. 8, v_H always finds the last available slot for each hop, which defers the transmission of $P^{su}_{v_H,v_i}$ to the deadline $I^C_{v_i,v_H}.t_b$ instead of immediately after v_H's update. With this mechanism, v_H may use these scheduled slots to send new schedule to v_i if it can update again during $[t^u_{v_H}, t^{v_H,v_i}_1]$. In line 1 of Alg. 8, $t^{v_H,v_1}_1 \le t^G_b$ checks if this case is possible ($t^{v_H,v_1}_1 > t^G_b$ indicates it is possible), where t^{v_H,v_1}_1 stores the first hop forwarding time scheduled in the last schedule update. This can offer v_H a chance to schedule more transmissions and therefore improve the channel utilization. The latter update may limit the amount of allocated slots to fit the maximum size of SU packet. Fortunately, because most new schedule generated at $t^u_{v_H}$ has expired (earlier than $t^u_{v_i}$) in this case, the maximum size of SU packet is usually enough to carry slots related to v_i.

If v_H can find all slots in line 3 of Alg. 8, $\mathcal{R}_1$ can be satisfied. Meanwhile, if there is at least one path is scheduled, v_H is able to set $t^u_{v_H}$ to $t^u_{est} - T_{SU}$, where T_{SU} is the duration taken by schedule update and t^u_{est} is the earliest first hop schedule forwarding time-point scheduled in line 3. Otherwise, $t^u_{v_H}$ is still not determined and marked as ∞.

Algorithm 8 Schedule forwarding time-points of non-last hops for paths with deadline

```
Input: $t^G_b$, $t^G_e$, $I^C_{v_i,v_H}$ for each $v_i \in \mathbf{N}^{CH}_{v_H}$, $v^{v_H,v_i}_k$ for $1 \le k \le n_{v_H,v_i}$, $F_{v_H,v_i}$ for each $v_i \in \mathbf{N}^{CH}_{v_H}$ and $\mathbf{P}_{v_H,v_i}$ is 4-hop, $T^{su}_{tx}$, $t^u_{est}$
Output: $t^{v_H,v_i}_k(0 \le k \le n_{v_H,v_i} - 2)$ for each $v_i \in \mathbf{N}^{CH}_{v_H}$, $t^u_{est}$
1: for each $v_i \in \mathbf{N}^{CH}_{v_H}$ such that $t^{v_H,v_1}_1 \le t^G_b$ and $I^C_{v_i,v_H}.t_b \ge t^G_e$ do
2:     $t^{v_H,v_i}_{n_{v_H,v_i}-1} \leftarrow I^C_{v_i,v_H}.t_b$;
3:     find the last instance $f$ of $F_{v_H,v_i}$ in $[t^G_b, I^C_{v_i,v_H}.t_b]$ if $\mathbf{P}_{v_H,v_i}$ is 4-hop, and $t^{v_H,v_i}_3 \leftarrow f.t_{begin}$;
4:     $t^{v_H,v_i}_k \leftarrow$ findLtTxSlot($v^{v_H,v_i}_k$, $v^{v_H,v_i}_{k+1}$, $T^{su}_{tx}$, $t^G.t_b$, $t^{v_H,v_i}_{k+1} - T^{v_H,v_i}_k$, $\mathbf{S}_{tx}$) for each $k = n_{v_H,v_i} - 2$ to 1;
5:     $t^u_{est} \leftarrow$ min($t^u_{est}$, $t^{v_H,v_i}_1$);
6: end for
```

Then in line 4 of Alg. 7, v_H calls Alg. 9 to schedule slots for REQ packets and $t^{v_i,v_H}_{n_{v_i,v_H}-1}$, which are all supposed to be earlier than $t^u_{v_H}$. This will ensure $\mathcal{R}_2$ and $\mathcal{R}_3$. As described in Alg. 9, if $t^u_{v_H}$ has not been decided yet, i.e., it is ∞, v_H finds the first available slot earlier than t^G_e for these slots. Specifically, for a cluster member v_m's REQ packet, $t^{req}_{v_m}$ must be later than the time $t'^{ntf}_{v_m} + T^{ntf}_{tx} + T^P_{v_H,v_m}$ when v_m receives its NTF packet scheduled at $t'^u_{v_H}$. This is because $t^{req}_{v_m}$ is carried by the NTF transmitted at $t'^{ntf}_{v_m}$. Similarly, $t^{v_i,v_H}_{n_{v_i,v_H}-1}$ must be later than $t'^{ntf}_{v^{v_i,v_H}_{n_{v_i,v_H}-1}} + T^{ntf}_{tx} + T^P_{v_H,v^{v_i,v_H}_{n_{v_i,v_H}-1}}$. Additionally, $v^{v_i,v_H}_{n_{v_i,v_H}-1}$ cannot forward $P^{su}_{v_i,v_H}$ before $I^C_{v_i,v_H}.t_e$, the time when it receives $P^{su}_{v_i,v_H}$ from its previous hop. Therefore, the earliest time for $v^{v_i,v_H}_{n_{v_i,v_H}-1}$ to forward is the later one of these two time-points. Based on the obtained schedule, v_H can decide $t^u_{v_H}$ by setting it to the time when v_H receives the last packet transmitted in these scheduled slots. To improve the protocol performance, all scheduled slots in this case are recorded, and will be rescheduled to tighten SUCIs.

In the case that $t^u_{v_H}$ is known, as stated in lines 4-5 of Alg. 9, all slots should be earlier than $t^u_{v_H}$ minus the corresponding propagation delays. In this way, v_H can receive these packets by its next schedule update at $t^u_{v_H}$. The earliest scheduled time of each slot is same to the case that $t^u_{v_H}$ is unknown.

If v_H can find all these slots supposed to be scheduled in lines 3-4 of Alg. 7, the selected $t^u_{v_H}$ satisfies $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$, and v_H can advance to line 11. Otherwise, v_H jumps back to line 2 to check the next gap. Note that v_H must be able to obtain all targeted slots through lines 2-10. Specifically, assume the last interval in $\mathbf{L}^C$ is I^C_l, and then the last gap is $[I^C_l.t_e, \infty]$, which is long enough for v_H schedule all slots.

In line 11, for optimization purpose, v_H tightens SUCIs with Alg. 10, which will be introduced later. After that, v_H schedules non-last hop SU forwarding times of paths except for the ones having been scheduled in line 3. Then

Algorithm 9 Schedule packet transmissions before the next schedule update

```
Input: $t_b^G$, $t_e^G$, $I^C_{v_i,v_H}$ for each $v_i \in \mathbf{N}^{CH}_{v_H}$, $v_k^{v_H,v_i}$ for $1 \leq k \leq n_{v_H,v_i}$, $F_{v_H,v_i}$ for each $v_i \in \mathbf{N}^{CH}_{v_H}$ and $n_{v_H,v_i} = 4$, $t^u_{v_H}$
Output: $t^{req}_{v_m}$ for all $v_m \in \mathbf{M}_{v_H}$, $t^{v_i,v_H}_{n_{v_i,v_H}-1}$ for all $v_i \in \mathbf{N}^{CH}_{v_H}$
1: if $t^u_{v_H}$ is $\infty$ then
2:     for each $v_m \in \mathbf{N}_{v_H}$, $t^{req}_{v_m} \leftarrow$findFtTxSlot($v_m$, $v_H$, $T^{req}_{tx}$, $t^{ntf'}_{v_m} + T^{ntf}_{tx} + T^P_{v_H,v_m}$, $t^G_e$, $\mathbf{S}_{tx}$); Insert this slot to $\mathbf{Q}$; $t^u_{v_H} \leftarrow \min(t^u_{v_H}, t^{req}_{v_m} + T^{req}_{tx} + T^P_{v_m,v_H})$;
3:     for each NCH $v_i$ such that $I^C_{v_i,v_H}.t_e < I^C.t_e$, $t^{v_i,v_H}_{n_{v_i,v_H}-1} \leftarrow$ findFtTxSlot($v^{v_i,v_H}_{n_{v_i,v_H}-1}$, $v_H$, $T^{su}_{tx}$, $\max(I^C_{v_i,v_H}.t_e$, $t^{ntf'}_{v_m}$+$T^{ntf}_{tx}$+$T^P_{v_H,v^{v_i,v_H}_{n_{v_i,v_H}-1}})$, $t^G_e$, $\mathbf{S}_{tx}$);
       Add field $t_{ready} = I^C_{v_i,v_H}.t_e$ to this slot and insert it to $\mathbf{Q}$; $t^u_{v_H} \leftarrow \min(t^u_{v_H}$, $t^{v_i,v_H}_{n_{v_i,v_H}-1}$+$T^{su}_{tx}$+$T^{v_i,v_H}_{n_{v_i,v_H}-1})$;
4: else
5:     for each $v_m \in \mathbf{N}_{v_H}$, $t^{req}_{v_m} \leftarrow$findLtTxSlot($v_m$, $v_H$, $T^{req}_{tx}$, $t^{ntf'}_{v_m} + T^{ntf}_{tx} + T^P_{v_H,v_m}$, $t^u_{v_H} - T^P_{v_H,v_m}$, $\mathbf{S}_{tx}$);
6:     for each NCH $v_i$ such that $I^C_{v_i,v_H}.t_e < I^C.t_e$, $t^{v_i,v_H}_{n_{v_i,v_H}-1} \leftarrow$ findLtTxSlot($v^{v_i,v_H}_{n_{v_i,v_H}-1}$, $v_H$, $T^{su}_{tx}$, $\max(I^C_{v_i,v_H}.t_e$, $t^{ntf'}_{v_m}$+$T^{ntf}_{tx}$+$T^{v_i,v_H}_{n_{v_i,v_H}-1})$, $t^u_{v_H}$-$T^{v_i,v_H}_{n_{v_i,v_H}-1}$, $\mathbf{S}_{tx}$); ;
7: end if
```

in line 13, v_H schedules a NTF packet to each cluster member respectively. Because packets transmitted in the slots scheduled in lines 12 and 13 carry the schedule information generated by the schedule update at $t^u_{v_H}$, these slots must be no earlier than $t^u_{v_H} + T_{SU}$, the time when the coming schedule update is done. When scheduling SU forwarding times, each hop's earliest forwarding time is subject to $t^u_{v_H} + T_{SU} \leq t^{v_i,v_H}_k + T^{v_i,v_H}_k + T^{su}_{tx} \leq t^{v_i,v_H}_{k+1} (1 \leq k \leq n_{v_H,v_i} - 2)$.

Finally, in line 14, v_H finds collision free slots for each requested data transmission extracted from REQ packets.

Now let us come back to discuss Alg. 10. Recall that v_H always finds the first available slot in Alg. 9 if $t^u_{v_H}$ is unknown. Then, these slots can yield $t^u_{v_H}$. However, this scheme may cause long SUCIs, which severely degrade the system performance. For example, assume v_H has two NCHs v_1 and v_2, and it gets $t^{v_1,v_H}_{n_{v_1,v_H}-2} = 100$ and $t^{v_2,v_H}_{n_{v_2,v_H}-2} = 700$ after running Alg. 9 given $t^u_{v_H} = \infty$. Then the earliest time for $t^u_{v_H}$ is $700 + T^P_{n_{v_2,v_H}} + T^{su}_{tx}$. Because $t^u_{v_H} < I^C_{v_H,v_1}.t_e$, v_1 cannot update its schedule at least in $[100, 700]$ and all nodes in cluster v_1 may be idle during this interval, which significantly wastes system resources. In addition, it is good to let v_m to send its REQ packet as late as possible, so this REQ packet can include more transmission requests. Then, the scheduling overhead can be reduced as more TX slots are scheduled in one schedule update. Due to these benefits, Alg. 10 is designed to tighten SUCIs and defer REQ transmissions, i.e., reschedule the slots in $\mathbf{Q}$ given by Alg. 9.

Because slots in $\mathbf{Q}$ will be rescheduled and will not be used, in line 1 of Alg. 10, v_H removes these slots from $\mathbf{S}_{tx}$. For clarity, we assume the n slots in $\mathbf{Q}$ are $s'_k (1 \leq k \leq n)$

Algorithm 10 Tighten SUCIs and defer REQ transmissions

```
Input: $\mathbf{Q}$, $t^{ntf'}_{v_m}$ for each $v_m \in \mathbf{M}_{v_H}$, $\mathbf{S}_{tx}$
Output: new $s.t_b$ of each $s$ in $\mathbf{Q}$, $t^u_{v_H}$
1: Remove each slot in $\mathbf{Q}$ from $\mathbf{S}_{tx}$
2: for each $s \in \mathbf{Q}$ in an decreasing order of $s.t_e$ do
3:     if $s$ is a slot for a $REP$ packet then
4:         $t_{est} \leftarrow t^{ntf\ '}_{s.src} + T^{ntf}_{tx} + T^P_{v_H,s.src}$;
5:     else
6:         $t_{est} \leftarrow \max(t^{ntf\ '}_{s.src} + T^{ntf}_{tx} + T^P_{v_H,s.src}, s.t_{ready})$;
7:     end if
8:     $t_{lst} \leftarrow t^u_{v_H} - T^P_{s.src,v_H}$;
9:     $s.t_b \leftarrow$ findLtTxSlot($s.src$, $s.dst$, $s.T_{dur}$, $t_{est}$, $t_{lst}$, $\mathbf{S}_{tx}$);
10: end for
```

after being sorted and they are $s_k (1 \leq k \leq n)$ after being rescheduled. Due to long propagation delays, a slot's new schedule may affect other pending slots' candidate schedule, so we cannot reschedule these slots arbitrarily. To make sure the success of Alg. 10, v_H first reschedules s'_n. Note that s'_n is given by findFtTxSlot() and s_n is by findLtTxSlot(), so s_n is no earlier than s'_n. Also note that s'_n is later than s'_{n-1} and does not affect s'_{n-1}, so s_n does not affect s'_{n-1} either. As a result, when rescheduling s'_{n-1}, s_{n-1} is same to s'_{n-1} or later. This is similar for the rest $n-2$ slots. Therefore, Alg. 10 guarantees $s_k.t_b \geq s'_k.t_b (1 \leq k \leq n)$, which indicates that Alg. 10 can tighten SUCIs and defer REQ transmissions.

VI. Discussion

In this section, we discuss some advanced features of DOS and how it deals with packet errors caused by acoustic channels other than collisions.

A. Supporting Multicast, Broadcast

In addition to conventionally supported unicast, both Alg. 3 and Alg. 4 are able to schedule a multicast or broadcast slot with a minor extension to Alg. 1. Specifically, for a unicast/multicast/broadcast of node v_s, every node in $\mathbf{R}$ is a receiver. Particularly, $\mathbf{R} = \mathbf{N}_{v_s}$ for broadcast. To avoid collisions at these receivers, we need to repeat lines 8-11 in Alg. 1 for each $v_r \in \mathbf{R}$. Then, for each $v \in \mathbf{N}_{v_s} - \mathbf{R}$, repeat lines 12-15.

Algorithm 11 Form conflict slot list for a broadcast

```
Function Name: formCListBcast()
Input: $v_s, \mathbf{R}, \mathbf{S}_{tx}$
Output: $\mathbf{C}_{v_s,Bcast}$
1: Lines 1-7 are same as that in Alg. 1;
8: for each slot $s \in \bigcup_{v_r \in \mathbf{R}} \mathbf{S}^{v_r}_{tx} \cup \mathbf{S}^{v_r}_{rx} \cup \mathbf{S}^{v_r}_{if}$ do
9:     $s' \leftarrow s$, $s'.t_{start} \leftarrow s'.t_{start} - T_{v_s,v_r}$;
10:    Insert $s'$ into $\mathbf{C}_{v_s,\mathbf{R}}$ sorted by field $t_{start}$ and merge overlapped slots;
11: end for
```

B. Handling packet errors caused by channel environments

In order to obtain collision-free schedule, each cluster head needs to receive up-to-date schedule from all NCHs. It would be a disaster if losing any SU packet. Although our scheduling algorithm can guarantee no collision with SU packets, a packet loss may still occur due to a poor channel condition.

To solve this problem, beside SU forwardings scheduled during each update, in the initialization phase every cluster head preserves slots for SU forwardings to its NCHs with a fixed period, which is similar to ferry slots, and periodically sends NCHs its up-to-date schedule via these preserved slots. The period is predefined and much longer than the average interval between two successive schedule updates, so this mechanism does not introduce too much overhead. With this mechanism, even if a SU packet is lost, the collision-free schedule can be guaranteed again after receiving a periodical SU from the same NCH. In addition, as an improvement, we can significantly reduce the packet loss rate of SU packets by applying forward error correction (FEC) scheme.

C. Potential to Handle Topology Changes

In underwater environment, nodes usually either passively or actively move, so it is important to handle topology changes. Although DOS does not consider dynamic topology, it still has the potential to handle that. Recall that DOS adopts one-hop cluster architecture and many distributed clustering algorithms like [23] are capable to deal with topology changes. If DOS can seamlessly integrate one of these clustering algorithms, it would be able to handle dynamic topology too. Because we focus on how to perform collision-free scheduling in this paper, we will leave this topic as our future work.

VII. Performance Evaluation

In this section, we evaluate the performance of DOS via simulations. We compare DOS with benchmark protocols ST-MAC [24], Slotted-FAMA [11], and ALOHA, which are a centralized scheduling-based, a handshake-based, and a random access-based MAC respectively. Particularly, we did necessary revisions to ST-MAC so that it can schedule on packet level[3]. This revision makes ST-MAC more practical. In addition, even ST-MAC admits that its strong assumption of priori traffic load, so it also has a variant which schedules without such priori knowledge [24]. We name the variant as ST-MAC w/o T. We evaluate both ST-MAC and ST-MAC w/o T for the comparison. The simulation platform is Aqua-Sim [25], an NS-2 based simulator for UANs.

A. Simulation Settings

Unless otherwise specified, there are 80 nodes randomly deployed in a multi-hop network in an area of $6000m \times 6000m \times 50m$, and each node randomly picks one neighbor as its receiver. In such a horizontal shallow water area, the effective modem transmission rate is set to $667bps$ and each packet contains a preamble of $1.5s$ according to our measurement of Teledyne Benthos Modem [16]. All nodes are configured with the same transmission range, 1100 meters, which is set according to our experiment experience. Each sender generates data packets of size $600B$ following an independent and identical Poisson process with the same traffic generation rate of $0.06pkt/s$. Due to space limitations, only the results of nodal throughput, the most important metric of the low bandwidth characterized networks, are presented to quantitatively show the performance.

B. Nodal Throughput with Varying Traffic Generation Rate

We change traffic generation rate λ from 0.01 to $0.15pkt/s$ to evaluate the performance of the protocols, and the results are shown in Fig. 3. From Fig. 3, we can see that DOS, ST-MAC, and ST-MAC w/o T achieve a much higher throughput than both ALOHA and Slotted-FAMA. This is because the former there protocols are collision-free and can eliminate the impact of costly contentions. The benefit of being collision-free becomes more obvious when the traffic is heavy, i.e., when λ is large.

As shown in Fig. 3, with a large λ, there are heavier traffic which correspondingly incur more collisions. Due to lacking of collision avoidance capability, ALOHA's throughput gradually drops to 0 as λ increases. As for Slotted-FAMA, its handshake scheme limits the traffic injected to the channel and can effectively reduce collisions among control packets. However, because control packets are not short in time anymore, handshake becomes inefficient. As a result, Slotted-FAMA's throughput reaches its upper bound and becomes stable after exceeding a small λ.

From Fig. 3, we can see ST-MAC far outperforms DOS as λ increases. This is because ST-MAC is a centralized solution and its design does not consider the overhead to collect nodes' traffic load and global topology, and the cost to let each node knows its schedule. Accordingly, the evaluation of ST-MAC does not consider these overhead either. Compared with ST-MAC, DOS is distributed and all the aforementioned cost is counted in the simulation.

When comparing DOS with ST-MAC w/o T, from Fig. 3, we can see that DOS achieves a better performance. This is because DOS schedules according to nodes' transmission requests and thus can guarantee a good performance while ST-MAC w/o T just wastes more than a half of slots due to no information of nodes' traffic load.

Another issue of ST-MAC and ST-MAC w/o T is that they sacrifice fairness to a higher throughput. Specifically, when allocating slots, they always first satisfy all requests of the link with the heaviest traffic load. Although this strategy can reduce time slot fragments and thus improve throughput, it impairs the fairness. Especially, when λ is large, a node may transmit for the whole simulation time while nodes around its receiver starve. Compared with these two protocols, DOS does not have this issue because T_{tx}^{req} can limits the maximum requested slots and all members can be assigned slots in each schedule update. For a fair comparison, we changed ST-MAC to ST-MAC w/ F in which all nodes having traffic can get a

[3]ST-MAC implicitly assumes that a packet can be of any the transmission delay, and it schedules in bit-level or even signal symbol level. In practice, however, all packets, even a 1-bit data packet, need to carry a MAC header and be embedded in a frame containing preamble and other necessary supplements. Also, it need to consider the cost of packet reassembling. Thus, it is impractical to schedule in unit of bit.

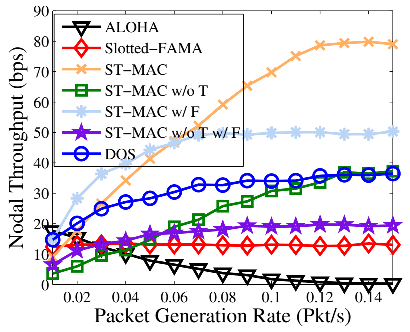

Fig. 3. Nodal throughput with varying traffic generation rate

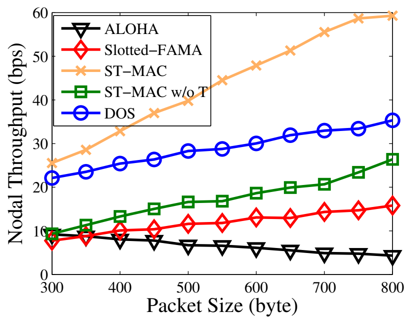

Fig. 4. Nodal throughput with varying packet size

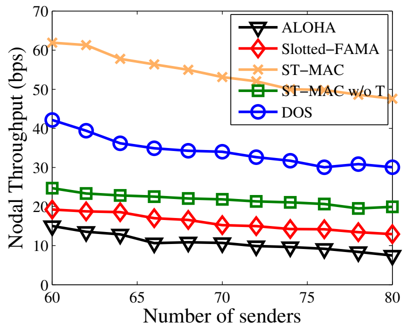

Fig. 5. Nodal throughput with varying number of senders

chance to transmit during the entire simulation time. A similar version for ST-MAC w/o T is ST-MAC w/o T w/ F.

From Fig. 3, we can observe an interesting phenomenon cased by fairness is that ST-MAC's throughput is lower than ST-MAC w/ F when λ is low. This is because ST-MAC assumes each node has all traffic ready before generating schedule while ST-MAC does not consider traffic arrivals. In practice, packets arrive as time elapses. Then, with a small λ, due to the greedy slot allocation strategy, in ST-MAC a node may not have packets in queue to send when its TX slots comes while other nodes having packets ready cannot transmit. In ST-MAC w/ F, more nodes get chance to transmit and this diversity helps increase the throughput when λ is small. However, as λ increases, this diversity causes more time slot fragments and thus the throughput of ST-MAC w/ F is much lower than ST-MAC. This explanation is also applicable to ST-MAC w/o T and ST-MAC w/o T w/ F.

From the discussions above, we can see, although there is a large gap between DOS and the original ST-MAC, which is a centralized algorithm, DOS can outperform ST-MAC when ST-MAC considers some reasonable constraints, like fairness. These comparisons justify that DOS is able to achieve a really high performance as a distributed on-demand solution.

C. Nodal Throughput with Varying Packet Size

Varying packet size from $300B$ to $800B$ gives the results in Fig. 4. Again, benefiting from the collision-free feature, both ST-MAC and DOS provide a much high throughput than others. From Fig. 4, we can also observe that all protocols except for ALOHA achieve a higher throughput as packets become longer. For DOS, this is because more data bytes can be delivered with the same amount of scheduled packets and thus the relative scheduling overhead is reduced. Similarly, Slott-FAMA's relative overhead also decreases because more data can be delivered after one successful handshake with longer data packets. In ST-MAC and ST-MAC w/o T, longer packets help reduce the time slot fragments and thus improve the channel utilization. As for ALOHA, due to lacking of collision avoidance capability, longer packet size renders heavier collisions and thus its throughput drops.

D. Nodal Throughput with Varying Number of Senders

Changing the number of senders from 60 to 80 generates the results in Fig. 5. From Fig. 5, we can see that collision-free MAC protocols, DOS, ST-MAC, and ST-MAC w/o T, outperform the other two. This set of results confirm again that being collision-free is a desired feature for MAC protocols to achieve a high performance in UANs. Also, we can see that the throughput of all protocols goes down as there are more senders. For DOS, ST-MAC and ST-MAC w/o T, this is because there are more time fragments with more senders. For Slotted-FAMA and ALOHA, this is because more senders cause heavier collisions.

VIII. Related Work

In the literature, an underwater MAC protocol usually employs scheduling or utilizes cluster architecture to prevent collisions.

There are several scheduling-based MAC protocols which guarantee collision-free transmissions [17], [18], [24], [26], [27]. Among them, ST-MAC [24] constructs a conflict graph based on the global network topology. Then, taking nodes' traffic load as input, ST-MAC generates collision-free schedule with the conflict graph, and it significantly improves the network performance. However, ST-MAC is a centralized scheduling algorithm. To construct the conflict graph, it requires the global topology information, which is expensive to collect in UANs due to long propagation delays and low transmission rates. In addition, the assumption of priori traffic load is too strong because usually this is unknown at the time of deployment. Moreover, to achieve a higher network throughput, ST-MAC always allocates slots in a batch to the links with the most traffic. This strategy significantly impairs the fairness of the system and make some nodes starve. ISTLS and its variants [18] can also provide collision-free schedule, but faces similar problems.

STUMP [17] is collision free too. However, similar to TDMA, each node's schedule is fixed during the whole network lifetime. With such a strategy, the channel utilization would greatly decrease if nodes' traffic loads are significantly heterogeneous. In [27], MAC scheduling are modeled as a mixed integer linear programming problem considering all kinds of possible conflicts so that the throughput can be maximized. Nevertheless, this solution is for one-hop networks only.

Compared with the aforementioned scheduling algorithms, DOS is a distributed algorithm and can generate collision-free schedule on-demand. It does not require to know nodes' traffic load in advance or any global information. Moreover, it can be used for both single-hop and multi-hop UANs.

There are also some MAC protocols utilizing cluster architectures to eliminate collisions [28], [29]. In [28], for example, adjacent nodes are grouped into clusters, and use TDMA within cluster while CDMA for inter-cluster communications. Inter-cluster communications occur through nodes belonging to multiple clusters, and these nodes are required to equip with multi-user receiver systems so that they can receive packets with different CDMA codes from multiple clusters simultaneously. However, as explored in [17], although CDMA can helps resolve collisions, it may impair the system performance in low bandwidth characterized UANs because packet transmission delays are significantly elongated by the spreading factor. Besides, existing commercial acoustic modems usually equipment single-user receiver systems. Multi-user receiver systems would increase the system complexity and cost.

Different from these cluster-based protocols, DOS can universally schedule intra-cluster and inter-cluster communications. DOS does not require CDMA modulation or multi-user receiver systems, and it still guarantees collision-free transmissions through pure scheduling.

IX. Conclusion

In this paper, first we propose an algorithm to find out collision-free TX slots. Then, based on this algorithm, we propose DOS, a distributed on-demand scheduling for MAC in UANs. DOS guarantees collision-free transmissions for both data and control packets, so that it can avoid costly contentions caused by unique acoustic modem characteristics. DOS is designed to work in a distributed way to avoid collecting expensive information, such as global topology and nodes' traffic load. Also, it can adapt to nodes' dynamic transmission requests and offer on-demand scheduling. DOS guarantees collision-free transmissions solely through scheduling, requiring no special modem capabilities such as CDMA, and therefore the protocol can more likely work with most contemporary modems. Simulation results show that DOS can far outperforms Slotted-FAMA and ALOHA, and achieve a comparable performance to ST-MAC, which is a centralized algorithm and its performance evaluation does not count any overhead because of its ideal assumptions.

Future Work To make DOS more practical, we plan to smoothly integrate DOS with a clustering algorithm capable to manage topology changes. In addition, we will explore the cross-layer design approach to couple DOS with cluster-based routing.

References

[1] E. M. Sozer, M. Stojanovic, and J. G. Proakis, "Underwater Acoustic Networks," *IEEE J. of Oceanic Engr.*, vol. 25, no. 1, pp. 72–83, 2000.

[2] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater Acoustic Sensor Networks: Research Challenges," *Ad Hoc Networks*, vol. 3, no. 3, pp. 257–281, 2005.

[3] J.-H. Cui, J. Kong, M. Gerla, and S. Zhou, "Challenges: Building Scalable Mobile Underwater Wireless Sensor Networks for Aquatic Applications," *IEEE Network*, vol. 20, no. 3, pp. 12–18, 2006.

[4] J. Partan, J. Kurose, and B. N. Levine, "A Survey of Practical Issues in Underwater Networks," in *Proc. of ACM WUWNet*, 2006, pp. 11–24.

[5] L. Liu, S. Zhou, and J.-H. Cui, "Prospects and Problems of Wireless Communication for Underwater Sensor Networks," *Wiley Wireless Communications and Mobile Computing*, vol. 8, no. 8, pp. 977–994, 2008.

[6] L. F. M. Vieira, J. Kong, U. Lee, and M. Gerla, "Analysis of Aloha Protocols for Underwater Acoustic Sensor Networks," in *Poster abstract in ACM WUWNet*, 2006, pp. 1–2.

[7] M. K. Park and V. Rodoplu, "UWAN-MAC: An Energy-Efficient MAC Protocol for Underwater Acoustic Wireless Sensor Networks," *IEEE J. of Oceanic Engr.*, vol. 32, no. 3, pp. 710–720, 2007.

[8] N. Chirdchoo, W.-S. Soh, and K. C. Chua, "Aloha-based MAC protocols with collision avoidance for underwater acoustic networks," in *Proc. of INFOCOM*, 2007, pp. 2271–2275.

[9] C. L. Fullmer and J. Garcia-Luna-Aceves, "Floor Acquisition Multiple Access (FAMA) for Packet-Radio Networks," in *Proc. of ACM SIGCOMM*, 1995, pp. 262 – 273.

[10] P. Karn, "MACA - a New Channel Access Method for Packet Radio," in *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, 1990, pp. 134–140.

[11] M. Molins and M. Stojanovic, "Slotted FAMA: a MAC Protocol for Underwater Acoustic Networks," in *Proc. of IEEE/MTS OCEANS*, 2006, pp. 1–7.

[12] N. Chirdchoo, W.-S. Soh, and K. C. Chua, "MACA-MN: A MACA-based MAC protocol for underwater acoustic networks with packet train for multiple neighbors," in *Proc. of IEEE VTC*, 2008, pp. 46–50.

[13] Teledyne-Benthos, "Acoustic Telemetry Modems: User's Manual," 2006.

[14] H. Yan, L. Wan, S. Zhou, Z. Shi, J.-H. Cui, J. Huang, and H. Zhou, "DSP based Receiver Implementation for OFDM Acoustic Modems," *Elsevier J. on Phys. Comm.*, vol. 5, no. 1, pp. 22–32, 2012.

[15] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball, "The WHOI Micro-Modem: An Acoustic Communications and Navigation System for Multiple Platforms," in *Proc. of IEEE/MTS Oceans*, 2005, pp. 1086 – 1092.

[16] Y. Zhu, Z. Jiang, Z. Peng, M. Zuba, J.-H. Cui, and H. Chen, "Toward Practical MAC Design for Underwater Acoustic Networks," in *Proc. of IEEE INFOCOM*, 2013, pp. 683–691.

[17] K. B. K. II, P. Djukic, and P. Mohapatra, "Stump: Exploiting position diversity in the staggered tdma underwater mac protocol," in *Proc. of IEEE INFOCOM*, 2009, pp. 2961–2965.

[18] J. Ma and W. Lou, "Interference-aware Spatio-Temporal Link Scheduling for Long Delay Underwater Sensor Networks," in *Proc. of IEEE SECON*, 2011, pp. 1–9.

[19] F. Lu, D. Mirza, and C. Schurgers, "D-sync: Doppler-based time synchronization for mobile underwater sensor networks," in *Proc. of ACM WUWNet*, 2010, pp. 3:1–3:8.

[20] J. Yackoski and C.-C. Shen, "UW-FLASHR: Achieving High Channel Utilization in a Time-based Acoustic MAC Protocol," in *Proc. of ACM WUWNet*, 2008, pp. 59–66.

[21] L. Bao and J. J. Garcia-Luna-Aceves, "Topology management in ad hoc networks," in *Proc. of the ACM MobiHoc*, 2003, pp. 129–140.

[22] O. Younis and S. Fahmy, "Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Trans. on Mobile Computing*, vol. 3, no. 4, pp. 366–379, 2004.

[23] B. An and S. Papavassiliou, "A mobility-based clustering approach to support mobility management and multicast routing in mobile ad-hoc wireless networks," *Int. J. Netw. Manag.*, vol. 11, no. 6, pp. 387–395, Nov. 2001.

[24] C.-C. Hsu, K.-F. Lai, C.-F. Chou, and K. C.-J. Lin, "ST-MAC: Spatial-Temporal MAC Schedulging for Underwater Sensor Networks," in *Proc. of IEEE INFOCOM*, 2009, pp. 1827–1835.

[25] P. Xie, Z. Zhou, Z. Peng, H. Yan, T. Hu, J.-H. Cui, Z. Shi, Y. Fei, and S. Zhou, "Aqua-Sim: An NS-2 Based Simulator for Underwater Sensor Networks," in *Proc. of IEEE/MTS OCEANS*, 2009, pp. 1 – 7.

[26] K. Kredo and P. Mohapatra, "Distributed scheduling and routing in underwater wireless networks," in *Proc. of IEEE GLOBECOM*, 2010, pp. 1–6.

[27] Y. Guan, C.-C. Sheng, and J. yackoski, "MAC Scheduling for High Throughput Underwater Acoustic Networks," in *Proc. of IEEE WCNC*, 2011, pp. 1–6.

[28] F. Salva-Garau and M. Stojanovic, "Multi-cluster protocol for ad hoc mobile underwater acoustic networks," in *Proc. of IEEE/MTS OCEANS*, vol. 1, 2003, pp. 91–98.

[29] K. B. Kredo, II and P. Mohapatra, "A hybrid medium access control protocol for underwater wireless networks," in *Proc. of ACM WUWNet*, 2007, pp. 33–40.