Pipelined Transmission MAC for String Underwater Acoustic Networks

UCONN CSE Technical Report: UbiNet-TR13-01
(Last Update: Dec. 2012)

Son N. Le, Yibo Zhu and Jun-Hong Cui
Computer Science and Engineering Department
University of Connecticut
Storrs, CT 06269
Email: {son,yibo.zhu,jcui}@engr.uconn.edu

Zaihan Jiang
Acoustic Division
US Naval Research Laboratory
Washington, DC 20375

***Abstract*—Despite effort spent over past years, an efficient MAC protocol design is still a challenging issue in underwater acoustic networks (UANs) mostly due to two factors: first, the long propagation delay of the acoustic signal in underwater environment; and second, the high variability of the environment. In this paper, we introduce a new MAC protocol for string UANs called Pipelined Transmission MAC (PTMAC). This protocol is essentially scheduling-based which pipelines data transmission to optimize network performance. The main contributions of the paper are: (1) design and implement a pilot version of PTMAC; (2) measure the performance of the protocol in the real open sea environment; and (3) propose a distributed method to improve the performance of the protocol through picking the appropriate maximum packet length. With this work, we introduce a working protocol for applications such as oil pipe monitoring, sea environment surveillance; as well as provide a baseline for performance comparison for future MAC protocols.**

I. Introduction

Underwater acoustic networks (UANs) [1]–[3] are often referred to as a set of technologies that can enable a wide range of promising applications, for example, tsunami forecasting or disaster prevention. Underwater communication is made possible with the use of acoustic signals; however, the propagation delay of acoustic signals is around five orders of magnitude longer than that of radio waves. Additionally, underwater communication suffers severely from environmental variability, which virtually lengthens the aggregate delay of sending a packet. In our past experiments, we observed that the quality of an underwater link can be intermittent; however, we did not have sufficient clues to pinpoint the reasons. Usually, the link became bad when there were dolphins around, or when the weather was unfavorable, such as after a rain or in rough sea. These unique characteristics of acoustic communication impose a great challenge on an efficient MAC protocol design.

By the time of this writing, there have been a number of MAC protocols proposed for UANs. Initially, the ALOHA mechanism is applied to UANs with two notable variants, ALOHA-AN [4] and UW-ALOHA [5]. ALOHA-AN alleviates collisions by having every sender notify its neighbors of the sending time using a short packet. UW-ALOHA introduces acknowledgment to improve reliability in environments with high packet error rates. However, in multi-hop scenarios, ALOHA-based solutions are inferior because they fail to handle the hidden terminal problem. Handshaking-based MAC protocols are devised [6]–[9] to deal with this problem. In these protocols, every node tries to schedule its channel occupation with its neighbors before it sends out data in order to avoid collisions during data transmission. However, in reality, handshaking may fail with high probability because of heavy collisions among control packets and intermittent link quality, which significantly degrades system performance. Although scheduling-based MAC protocols [10]–[12] guarantee collision-free transmissions, their applications are limited because the time allocation solutions are pre-determined based on the global network topology.

In this paper, we propose a scheduling-based MAC protocol, called **PTMAC**, short for **Pipelined Transmission MAC**, for string UANs. Such a topology can be found in applications such as monitoring oil pipes or submarine cables. The main idea of the protocol is transmission pipelining:

- **Slot-based operations:** A data packet is transferred within a time slot.
- **Parallel transmission:** Any two nodes that are three hops away from each other are able to send data simultaneously without collisions.
- **Implicit acknowledgment:** After a node N_i finishes sending a packet, if it overhears that same packet forwarded by the next node, N_{i+1} on the string, the packet will be considered as successfully received and N_i will proceed with a new packet; otherwise, it will retransmit the current one.

The main contributions of the paper are following. First, we design and implement a pilot version of PTMAC. Second, we evaluate the performance of the protocol in a real field test offshore New Jersey. The performance metrics consist of the end-to-end throughput and the end-to-end delay. These metrics can be used as a comparison baseline for future protocols. Third, based on the field test data and experience, we introduce a scalable strategy to choose the slot length which provides a sub-optimal performance. Since a time slot must be sufficiently long to transmit the largest packet possible, choosing an optimal slot length is equivalent to picking an appropriate maximum packet length.

The rest of this paper is divided into four parts. In Sec. II, we review some MAC protocols for UANs. Sec. III lays out the pilot design of the protocol, with data collected from a sea test. Sec. IV introduces a distributed algorithm to choose the suboptimal packet length, whose efficiency is demonstrated through simulation data. Sec. V concludes this paper.

II. Related Work

MAC protocols for UANs can be roughly divided into three classes: random access, handshaking-based and scheduling-based. So far, many random access MAC protocols have been proposed [4], [5], [13], [14] for UANs. Some protocols in this class adopt collision avoidance techniques to improve their performance. In ALOHA-AN [4], for example, before transmitting a data packet, a node first sends a short notification (NTF), which carries the size, sender identifier and receiver identifier of the data packet. Since every node is assumed to know propagation delays to its neighbors, it can determine when the destined node will receive the data packet after receiving the NTF; therefore, this scheme can significantly alleviate collisions in single-hop networks. However, since this class of protocols lacks the ability to handle the hidden terminal problem, their performance is substantially impaired in multi-hop networks. Compared with random access MAC protocols, PTMAC can also avoid the collisions caused by both neighbors and hidden terminals because of its scheduling scheme. Thus, PTMAC is expected to achieve good performance in multi-hop string UANs.

In multi-hop scenarios, handshaking-based MAC protocols are widely used [6]–[9], [15]–[17] because they are able to prevent collisions caused by neighbors and handle the hidden terminal problem. A notable member of this category is Slotted FAMA [8]. A node having data to send should first emit an RTS packet. All neighbors of the node upon receiving the packet can calculate when the data packet is transmitted. The receiver replies to this packet with a CTS packet, which implicitly notifies all hidden terminals of the sender the transmission of the corresponding data packet completes. In this way, an ideal handshake eliminates possible collisions during a data transmission.

However, handshaking-based MAC protocols work well providing short transfer delay[*] of RTS and CTS packets and low bit error rates (BER). In UANs' reality, even a short packet suffers from long transfer delay and high BERs usually occur in burst. As a result, a handshake may fail with a high probability, which not only wastes the transmission opportunity of the sender but also idles some of its neighbors, thus impairing the system performance. In particular, if the sender fails to decode a CTS packet, it will not transmit data packet in the next slot. Nodes that are able to decode the RTS or CTS packets will remain silent to avoid collisions during the slots specified by these packets, which also means they will not reply to incoming RTS packets; consequently, more handshakes are failed. Therefore, in practice, handshaking-based MAC protocols cannot work well in multi-hop networks, such as a string topology. In contrast, PTMAC does not rely on control packets to avoid collisions and a failed transmission does not affect other nodes' concurrent transmissions, which, we believe, is necessary for the protocol to operate well even under high BERs and long transfer delay.

[*]Transfer delay in this context includes transmission delay, propagation delay and processing delay.

In addition to the two above categories, scheduling-based protocols are also studied for multi-hop networks [10]–[12]. In this class of protocols, after deployed, every node is assigned fixed time slots to transmit data. For example, ST-MAC [10] is accompanied with a collision-free time slot allocation algorithm whose input is the network topology; the algorithm takes into account Rx-Rx, Rx-Tx, and Tx-Tx collisions. Similar to PTMAC, scheduling-based MAC protocols are not significantly affected by long transfer delay and high BERs occurring in burst; however, their time allocation algorithms rely on global topology information. Unlike these protocols, PTMAC determines the slot length in a distributed fashion, and thus it is robust to varying propagation delays.

III. Pipelined Transmission MAC Protocol

In this section, we first begin with an overview how pipelined transmission works in PTMAC and then proceed to describing technical details of the protocol. Note that in this paper, we do not assume time synchronization across the network.

A. Overview

One of our objectives in the design of PTMAC is to guarantee transmission reliability. This can be done using techniques such as retransmission and acknowledgment. However, since delays in acoustic networks are much more significant than those of radio networks, one way to improve performance is to minimize the number of control packets. In this design, we proposed the use of cumulative acknowledgments and implicit acknowledgments. Explicit acknowledgments are used only when needed.

PTMAC is designed for string networks on which, we assume that every two nodes that are at least three hops away are able to transmit data simultaneously without collisions. The operation of PTMAC is illustrated in Fig. 1. In this protocol, time is divided into slots, each of which is delimited by a dashed line between two black dots. This figure also shows that time slots are synchronized among nodes.

In order to facilitate our discussion, let N_k denote the k-th node on the string, S_i the i-th slot, and P_l the l-th packet.

Every node is restricted to limit any its data transmissions within one time slot. Take Fig. 1 as an example: N_1 starts first with P_1 in a slot and by the end of this slot, the packet is transferred to N_2. N_2, in turn, waits for the next time slot and forwards the data to N_3. By the end of this time slot, this packet not only reaches N_3 but also is overheard by N_1, which is an implicit indication to N_1 that P_1 has been received

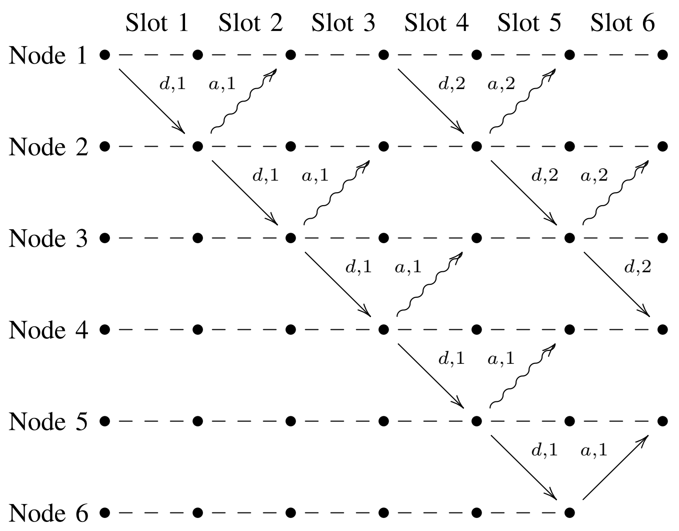


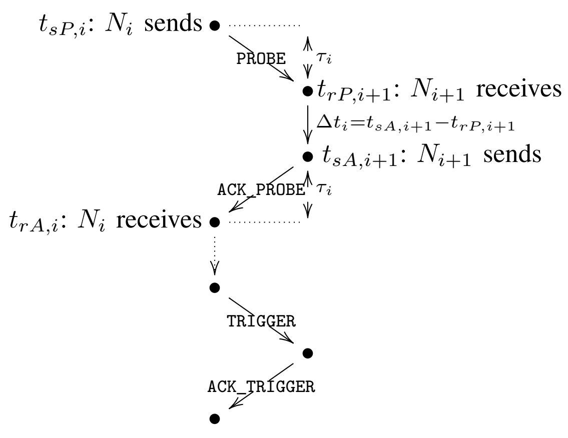
Fig. 1. **Pipeline operation.** A straight solid arrow represents a packet sent from one node to another, and a squiggly one denotes an overheard packet: (1) the arrow direction indicates where the corresponding packet comes from, and (2) the arrow label has two components: the corresponding packet type ('a' means acknowledgment and 'd' data) and packet sequence number.

by N_2 and that N_1 can proceed to sending P_2. This process repeats on successive nodes on the string, and note that while N_4 is forwarding P_1 to N_5, N_1 is sending P_2 to N_2. This concurrency is possible because N_3 is unable to hear the data packet from N_1 and neither is N_2 from N_4. Therefore, no collision occurs while N_2 is receiving P_2 from N_1 and N_3 overhears the implicit acknowledgment for P_1 from N_4.

From the above description, implicit acknowledgments are employed to minimize the exchange of control packets. This protocol however does not eliminate the use of explicit acknowledgments. In Fig. 1, when P_1 reaches the last node, N_6 in this example, this node needs to explicitly acknowledge the reception of P_1 for it does not forward the data any further. The more general case is when a node have already finished forwarding a particular packet, but the sender keeps transmitting the same packet, because for example it did not receive the implicit acknowledgment. In this case, an explicit acknowledgment is needed to tell the sender to move on to the next packet.

There are also situations that we use piggybacked implicit acknowledgments. A forwarder may accumulate more than one packet in its incoming queue because the quality of the next hop link is not as good as that of the previous one. When this node forwards a packet, it implicitly acknowledges the last packet that it has received successfully.

Before we have the network transmissions working as in Fig. 1, there are two major issues to solve. First, the network needs to determine a uniform slot length to use. Second, because transmissions occur on the slot basis, and a uniform slot length is adopted, the start time of the first slot must be synchronized on all nodes. In other words, each node needs to align its transmission timeline to match others. In the following sections, we are going to present how the protocol handles these issues.



$t_{sP,i}$: N_i sends
PROBE
τ_i
$t_{rP,i+1}$: N_{i+1} receives
$\Delta t_i = t_{sA,i+1} - t_{rP,i+1}$
$t_{sA,i+1}$: N_{i+1} sends
ACK_PROBE
$t_{rA,i}$: N_i receives
TRIGGER
ACK_TRIGGER

Fig. 2. Slot length estimation

B. Slot Length Estimation

Slot length is estimated on a per-hop basis, starting from one end of the string network and spreading to the next node until the other end is reached. In this section, we assume that the packet size is predetermined and uniform in the network; therefore, the slot length must be sufficient for any node to transmit a packet to its neighbors. If we let τ_i be the time needed to transmit a packet on the link between N_i and N_{i+1}, the appropriate slot length will be $\max_{k=1}^{n}\{\tau_i\}$, where n is the number of nodes.

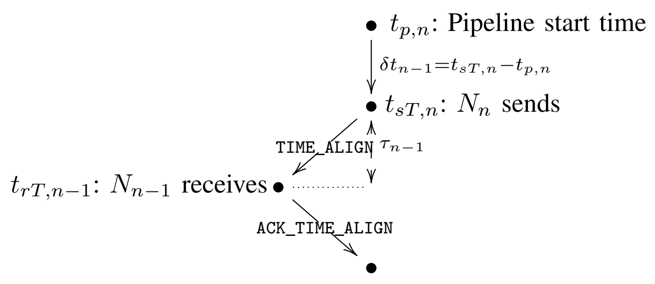
In our protocol, this estimation phase is implemented recursively. Let us assume that N_{i-1} has calculated the slot length of the network segment from the first node to itself, $\max_{k=1}^{i-1}\{\tau_k\}$. N_{i-1} then informs N_i of this value, so that N_i can start its estimation as illustrated in Fig. 2. In this estimation, four packet types are involved: PROBE, ACK_PROBE, TRIGGER and ACK_TRIGGER. N_i first sends a PROBE to N_{i+1} and record the event time as $t_{sP,i}$. N_{i+1} after receiving the packet, acknowledges it with an ACK_PROBE. PROBE and ACK_PROBE packets are zero padded so that their sizes match the predetermined value to guarantee that the aggregate delays of transferring a PROBE and an ACK_PROBE are close. In addition to this, the ACK_PROBE carries the latency, Δt_i from when the corresponding PROBE is received to when it is acknowledged to improve estimation accuracy.

After N_i receives the ACK_PROBE, it calculates its τ_i as follows

$$\tau_i = \frac{t_{rA,i} - t_{sP,i} - \Delta t_i}{2}, \tag{1}$$

where $t_{rA,i}$ is the time when the ACK_PROBE arrives at N_i . In order to mitigate the effect of randomness, this PROBE-ACK_PROBE exchange is run several times on each hop to obtain the average value.

Upon completion of this estimation process, N_i wraps $\max\{\max_{k=1}^{i-1}\{\tau_k\}, \tau_i\} = \max_{k=1}^{i}\{\tau_k\}$ in a TRIGGER packet and sends to the next node on the string. This node recursively repeats the above procedure. The last node in this chain will



$t_{p,n}$: Pipeline start time
$\delta t_{n-1}=t_{sT,n}-t_{p,n}$
$t_{sT,n}$: N_n sends
TIME_ALIGN
τ_{n-1}
$t_{rT,n-1}$: N_{n-1} receives
ACK_TIME_ALIGN

Fig. 3. Timeline alignment

have the slot length to use in the network, $\max_{k=1}^{n}\{\tau_i\}$. A TRIGGER packet is much shorter than a PROBE or ACK_PROBE; it carries only $\max_{k=1}^{i}\{\tau_k\}$. N_i acknowledges the TRIGGER packet with an ACK_TRIGGER packet. If N_i does not receive the ACK_TRIGGER within $2\tau_i$ units of time, it resends the TRIGGER packet. Note that the TRIGGER packet can also be implicitly acknowledged by N_i overhearing a PROBE packet from N_{i+1}.

C. Timeline Alignment

Slot length estimation is initiated by N_1 and ends with N_n having the slot length, $\tau = \max_{k=1}^{n}\{\tau_k\}$. However, this information is not known to other nodes. In this section, we introduce a mechanism to propagate this information to the rest of the network which in the same time, aligns nodes' timelines. This procedure is illustrated in Fig. 3.

N_n, after receiveing a TRIGGER packet, picks a starting time for its pipeline transmission, $t_{p,n}$. Theoretically, whether or not $t_{p,n}$ is behind the current time makes no difference; however, in this protocol, we picked some point after the current time to make room for processing such as initializing timers. This choice also simplifies time arithmetic. Starting from $t_{p,n}$, N_n updates its slot count every τ units of time.

After the starting time is picked, N_n sends a TIME_ALIGN packet to N_{n-1}, which carries the slot length τ, its sending token, time difference from $t_{p,n}$ to when this packet is sent. If we denote the time when this TIME_ALIGN is sent as $t_{sT,n}$, the time difference is

$$\delta t_{n-1} = t_{sT,n} - t_{p,n}. \tag{2}$$

Same as PROBE or ACK_PROBE packets, a TIME_ALIGN packet is zero padded so that its size equals the predetermined value. This padding is crucial because the transfer of a TIME_ALIGN packet takes as long as that of a PROBE packet. Therefore, N_{n-1} is able to estimate the pipeline start time using

$$t_{p,n-1} = t_{rT,n-1} - \delta t_{n-1} - \tau_{n-1}, \tag{3}$$

where $t_{rT,n-1}$ is the time when N_{n-1} receives the TIME_ALIGN packet and τ_{n-1} is the aggregate delay of transferring a packet of the predetermined size from N_n to N_{n-1} or vice versa. This value is known to N_{n-1} from the slot length estimation phase as described in Sec. III-B.

Similar to the PROBE-ACK_PROBE exchange in Sec. III-B, multiple TIME_ALIGN packets are sent in each hop to improve estimation accuracy. After N_{n-1} receives the last TIME_ALIGN packet from N_n, it follows the same procedure as N_n does to help N_{n-2} determine the pipeline starting time and so on. N_n considers a TIME_ALIGN packet to be successfully received if it receives an ACK_TIME_ALIGN packet from N_{n-1}, which acknowledges the TIME_ALIGN packet. Different from an ACK_PROBE packet, an ACK_TIME_ALIGN packet need not be zero padded; it carries the sequence number of the corresponding TIME_ALIGN packet.

After a node finishes estimating the pipeline starting time, it counts time slots in a circular progression fashion on three numbers, 1, 2, and 3. Namely, if we let SI be the index of the current time slot, and SI_{+1} be that of the next one, SI_{+1} can be written as

$$SI_{+1} = \begin{cases} SI+1, & SI = 1, 2 \\ 1, & SI = 3 \end{cases}. \tag{4}$$

As a side note, N_i can always know the current time slot index using the following equation

$$SI = \left\lceil \frac{t_{now,i} - t_{p,i}}{\tau} \right\rceil \bmod 3 + 1, \tag{5}$$

where $t_{now,i}$ is the current time in N_i's timeline.

As described in Sec. III-A, a node is only allowed to transmit data within one slot. This mechanism is implemented with the notion of token mentioned earlier in this section. A node occupies a time slot only when its token matches the slot index. As the initiator of the time alignment phase, N_n is free to choose it token, which is either 1, 2 or 3. N_{n-1} when receiving a TIME_ALIGN packet from N_n, computes its token in an inverse circular progression. Let T_n be the token of N_n, and T_{n-1} be that of N_{n-1}, we have

$$T_{n-1} = \begin{cases} T_n - 1, & T_n = 2, 3 \\ 3, & T_n = 1 \end{cases}. \tag{6}$$

D. Data Transmission Phase

In the data transmission phase, N_i sends at the beginning of a time slot if its sending token matches the time slot index. If the packet is intended for one of its neighbors, N_k, N_k sends an explicit acknowledgment to N_i. Otherwise, N_k forwards the packet, which implicitly tells N_i to proceed to the next packet. If N_k receives a duplicate packet, it either sends an explicit acknowledgment back if it has no data to send or piggybacks it on an outgoing data packet.

E. Field Test Results

We implemented PTMAC in Aqua-Net [5], a framework for underwater networking, as a plugin at the MAC layer and tested it in a sea experiment organized by the Naval Research Laboratory in early September of 2012. The test site was about 100 miles offshore Delaware, on the New Jersey Shelf as depicted in Fig. 4.

Each network node is a standalone system, equipped with the following instruments:

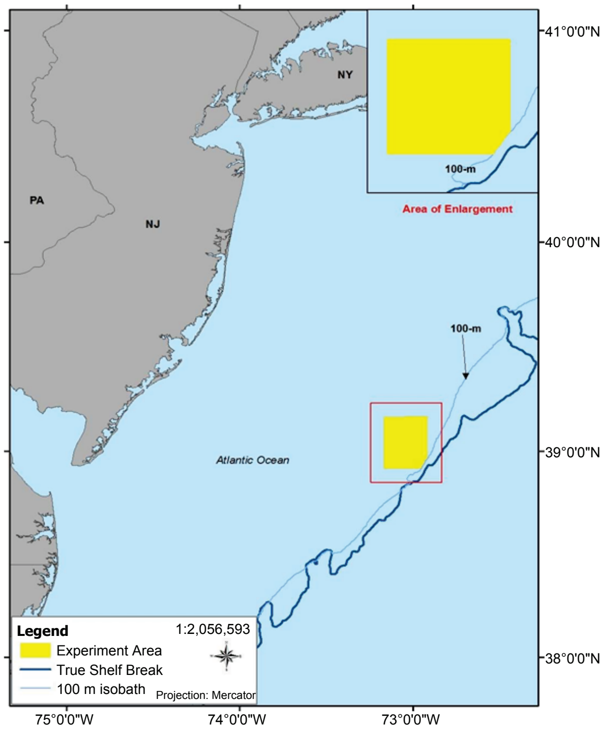


Fig. 4. Test site area

TABLE I
DATA SET PARAMETERS

	Topology	Packet size (bytes)	Data rate (bps)	Packet rate (packets/s)
Set 1	8-4-7-6-5	500	600	0.015
Set 2	8-4-7-6-5	200	300	0.015
Set 3	8-4-7-6-5	500	600	0.02
Set 4	5-10-7-4-3-1	200	300	0.005
Set 5	5-10-7-4-8-2-9-3-1	200	300	0.005

- a Gumstix Verdex Pro XM4 microcomputer [18] running Embedded Linux which hosts the protocol stack,
- a GPS receiver to track its position,
- an RF modem to monitor the node from our boat, and
- an ATM-885 MF modems [19] from Benthos.

Due to time limitation, we were only able to collect 5 sets of data for PTMAC whose parameters are given in Table I. In each experiment, there is only one sender and one receiver; the left most node on the topology column is the sender and the right most one is the receiver. Because of varying environmental conditions, we could not have sent at a fixed data rate in all the tests because lower data rates gave us higher reliability. At the sender, packets are generated so that the inter-packet intervals follow the Poisson distribution whose parameter, λ, is listed in the packet rate column.

From the protocol log files, we collect the following metrics: end-to-end (e2e) throughput, average and the standard deviation of the e2e delay, and efficiency. The metrics are calculated as follows:

- The e2e throughput of the network is the total *non-duplicate* data received at the sink divided by the time from when the first packet is sent by the source node to when the last packet reaches the sink.
- The e2e delay of a packet is calculated as the time difference from when it is first sent from the source to when it is first first received at the sink.
- Efficiency is the number of distinct packets received by the sink divided by maximum retransmissions that a node in the network makes. The node that retransmits the most is the bottleneck.

Ideally, there are no retransmissions; however, in practice, they occurs due for many reasons: bad link quality caused by environment variability, multipath effects caused by environment geometry that impairs the pipelined transmissions, etc. The result is listed in Table II.

The first three set of data was conducted under favorable environmental conditions on September 7 and 8, compared to the latter two sets. In the first set, with the packet size of 500 bytes and the packet rate of 0.015 packets/s; therefore, in average we have 60.00 bits generated every second. For this set, the end-to-end throughput is 50.90 bps which is close to the sending rate. The throughput of the second set, 23.32 bps is even closer to the data rate, 24 bps; the average e2e delay is also lower than the previous case. This result was mostly accounted for by the lower data rate, which guaranteed that a packet was delivered more reliably. The third set of data was obtained with same parameters as the first one, except for the sending rate. The data rate in this case was 80 bps, and the throughput was about half this value, 43.86 bps. This fact was because the sea became rougher, which is easily spotted by the high average e2e delay. We can also see this by looking at the efficiency column: efficiency of Set 3 is much lower than that of Set 1 and Set 2.

In the afternoon of September 8, we deployed all the 9 nodes that we had, forming a string topology 1-3-9-2-8-4-7-10-5. After that, we tested each link individually. Although the segment consisting of 4-7-10-5 was stable, communication on nodes 2, 8 and 9 was intermittent; there were many CRC errors. So we skipped them and ran Set 4 on 6 nodes 1-3-4-7-10-5. The e2e delay of this set is two times longer than that of Set 2. On this day, it was before a storm, and the weather was bad, which was reflected by the low efficiency of $18/70$. Although we sent data at a very low rate, reliability was still not guaranteed. Set 5 was run after Set 4 on all the nodes despite unfavorable weather. What we noticed was that during the test, the links 8-2 and 10-7 were the worst, and they limit the performance.

In Set 4 and Set 5, node 10 was reset when it was redeployed. Since the microcomputer does not have a battery to maintain system time, when it is reset, the time is reverted back to some default value. On top of that, we forgot to synchronize its time with others. However, from the log files and what we observed in the field, the protocol was able to deal with this time mismatch, which again confirm that this protocol does not require prior time synchronization.

In this field experiment, besides PTMAC, we also imple-

TABLE II
EXPERIMENT RESULT

	End-to-end throughput (bps)	Average e2e delay (s)	Stddev e2e delay (s)	Efficiency
Set 1	50.90	222.16	92.69	38/79
Set 2	23.32	94.83	31.08	45/87
Set 3	43.86	519.30	250.43	95/211
Set 4	10.37	192.25	93.35	18/70
Set 5	6.69	2658.25	1078.92	48/334

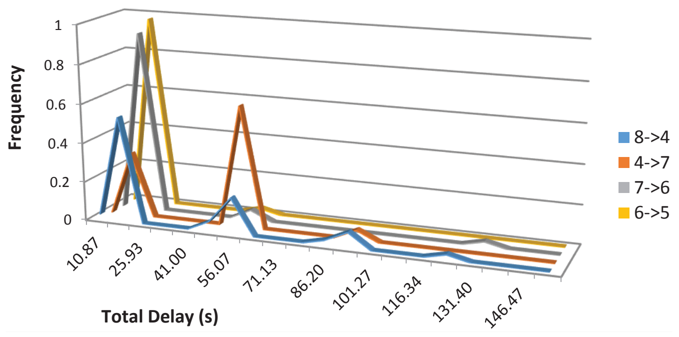


Fig. 5. Delay distribution for Set 1

mented and tested Slotted FAMA [8]. In the same settings as Set 2, the e2e throughput was only 6.67 bps, which is about 3.50 times lower than that of PTMAC. With the settings of Set 5, the e2e throughput of Slotted FAMA was 2.11 bps, 3.17 times lower than PTMAC. Note that in tests involved Slotted FAMA, every network node's local time was synchronized with its GPS.

Besides these metrics, we also process the trace files to obtain per-hop delay distributions and visualize them in Fig. 5-Fig. 9 for Set 1-Set 5, respectively. From these figures, a good link is the one whose distribution has a short tail, for example the link between 5 and 6 in Fig. 5 and Fig. 6. The time gap between two consecutive peaks in these graphs equals 3 time slots because as presented in Sec. III-A, a node, after sending, needs to wait for 3 time slots for another turn. As we mention earlier in this section, the links 8-2 and 10-7 were bad in Set 5, which is shown in Fig. 9 that their tails are heavier than others'.

Another remark that we draw from these distributions is that underwater links may vary greatly, even though they are close by. For example, the link from 8 to 4 and 4-7 in Fig. 5. This means that a preset packet length for the whole network may not be a good idea. In other words, the protocol should also handle the choice of packet size.

IV. Adaptive Maximum Packet Length

As we can see from the real field test data, link quality of two successive links can be completely different. Moreover, according to what we found from the trace files, link quality varies with time. Therefore, a fixed maximum packet length even initially considering environmental conditions may not give us the best performance if the network is to operate for

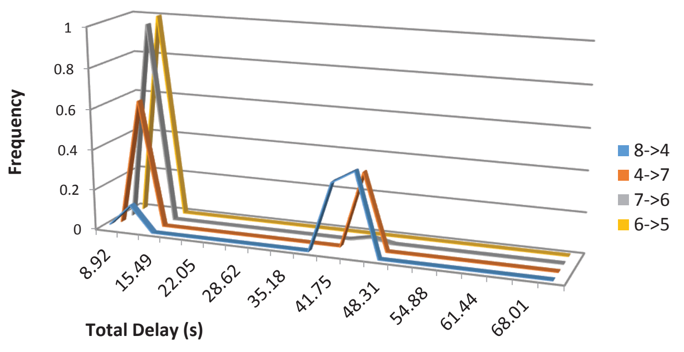


Fig. 6. Delay distribution for Set 2

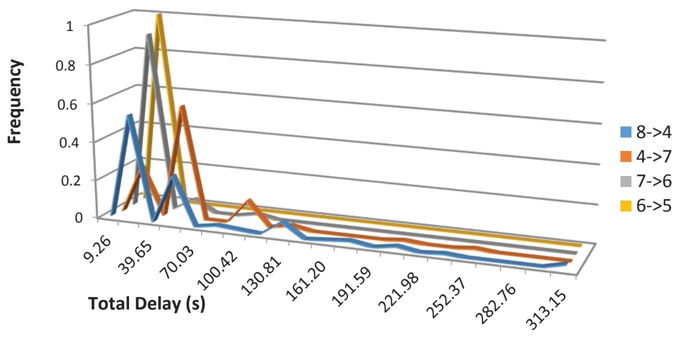


Fig. 7. Delay distribution for Set 3

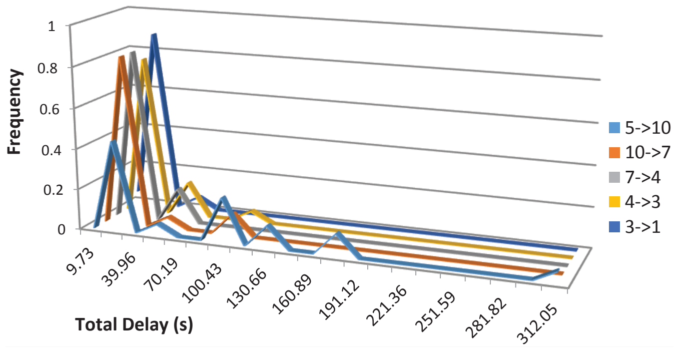


Fig. 8. Delay distribution for Set 4

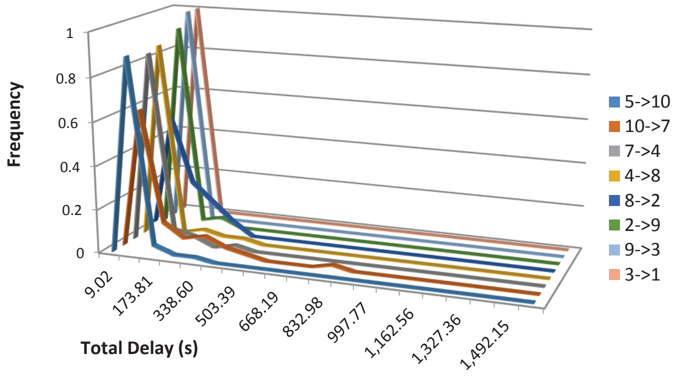


Fig. 9. Delay distribution for Set 5

a long time. In this section, we consider an adaptive method to select the value.

A. Optimal Hop-based Packet Size

To facilitate our discussion, let us consider a string network of n hops. Denote

- l as the packet length,
- $\tau(l)$ as the length of the slot in which a packet of size l can be transferred. In this work, $t(l)$ is estimated by a linear function: $t(l) = \alpha l + \beta$, where $\alpha, \beta > 0$.
- $p_i(l)$ be the probability that packet of size l gets through the i-th link. If we let γ_i be the bit error rate (BER) of L_i, $p_i(l)$ can be written as
$$p_i(l) = (1 - \gamma_i)^l, \tag{7}$$
- L_i as the i-th link,
- S as the chunk of data to send, and $|S|$ as its size.

We would like to minimize the time it takes to transfer S to from one end of the string to another. One can realize that S will be passed through all links before it reaches the destination. Therefore, one heuristic solution to the original problem is to minimize the average time that the chunk spends on each link:
$$l_i = \arg\min_l \frac{|S|}{l}\, t_i(l) = \arg\min_l \frac{t_i(l)}{l}, \tag{8}$$
where $t_i(l)$ is the average time needed to send a packet of size l over L_i.

Let R_l be the event that a packet of size l is received successfully. R_l is equivalent to the event that the packet reaches the next node, and its acknowledgment reaches the sender. Therefore, the probability of this event is
$$\Pr[R_l] = p_i^2(l). \tag{9}$$

Let X_i be the number of retransmissions for a packet of size l. Since the packet is retransmitted until it is successfully received, X_i follows a geometric distribution with parameter $p_i^2(l)$, meaning
$$\Pr[X_i = k] = \left(1 - p_i^2(l)\right)^{k-1} p_i^2(l). \tag{10}$$

Each retransmission will delay a packet for 3 time slots; thus, the average delay that a packet of size l experiences on L_i is
$$t_i(n) = \mathbb{E}[3\tau(l)\, X_1] = 3\tau(l)\, \mathbb{E}[X_1] = \frac{3(\alpha l + \beta)}{p_i^2(l)}, \tag{11}$$
where $\mathbb{E}[X]$ denotes the expected value of a random variable X. Hence, Eq. 8 becomes
$$l_i = \arg\min_l \frac{3(\alpha l + \beta)}{l p_i^2(n)} = \arg\min_l \left(\frac{\alpha}{p_i^2(l)} + \frac{\beta}{l p_i^2(l)}\right). \tag{12}$$
Using calculus techniques, we can prove that the value of l_i is
$$l_i = \frac{1}{2\alpha}\sqrt{\alpha^2 + \beta^2 + \frac{2\alpha\beta\,(1 + (1-\gamma_i)^2)}{1 - (1-\gamma_i)^2}} - \frac{\alpha + \beta}{2\alpha}. \tag{13}$$
A proof for this result is presented in the appendix.

B. PTMAC with Adaptive Packet Size

In Sec. IV-A, we have laid out the theoretical foundation to improve PTMAC. In this section, we describe how this is integrated into the existing protocol.

When the network is initially deployed, we assume that no environmental information is available. The protocol, therefore, initializes with a default packet size. After the network is in operation, a node N_i can collect the **packet error rate** (PER) the link L_i to the node N_{i+1} as follows:

- N_i keeps track of the number of retries for a particular data packet,
- N_{i+1}, as mentioned above, acknowledges the data packet upon receiving it. However in this version of the protocol, N_{i+1} adds the number of times it receive the same data packet in the acknowledgment packet.

Let the number of retransmissions and the number of successful receptions for the packet of with the sequence number k be $R_{rt}^{(k)}$ and $R_{sr}^{(k)}$ respectively. Using these two pieces of information, PER of L_i is calculated as
$$PER_i = \frac{\sum_k R_{rt}^{(k)}}{\sum_i R_{sr}^{(k)}}. \tag{14}$$

Let l be the current packet size throughout the string. Based on Eq. 7, the **bit error rate** (BER) of L_i is obtained by
$$\gamma_i = 1 - (1 - PER_i)^{1/l}. \tag{15}$$

Periodically, a new packet size is chosen so that the network adapts to environmental changes. The refreshment procedure is depicted in Fig. 10, which is initiated from N_1. Specifically, N_1 computes the best packet length l_1 for L_1 using Eq. 15 and Eq. 13 and passes this information in a `NEW_SLEN` packet to N_2. N_2, in turn, computes l_2, compares it with l_1 and sends the smaller value to N_3. As the last node, N_n has the shortest optimal length, $l = \min\{l_1, \ldots, l_{n-1}\}$.

N_n passes l to N_{n-1} in an `UPDATE_SLEN` packet. N_{n-1} passes on the value to N_{n-2} and so on. Eventually, N_1 receives l and initiates the new transmission pipeline as in Sec. III-B and Sec. III-C.

Note that both `NEW_SLEN` and `UPDATE_SLEN` packets control packets and are therefore short. Here we utilize implicit acknowledgment to reduce network traffic. A packet of these types is retransmitted if the corresponding forwarded packet is not overheard.

C. Simulation Results

In this section, we use simulations to verify the revised protocol introduced in Sec. IV-B. As described earlier, PTMAC is essentially a collision-prevention protocol, meaning that we need not simulate collisions. Therefore, the simulations, each consisting of packet exchange, packet loss and scheduling, are written in MATLAB.

Throughout these simulations, we assume a 10-hop network where the BER of a link is either $q_1 = 0.00001$ or

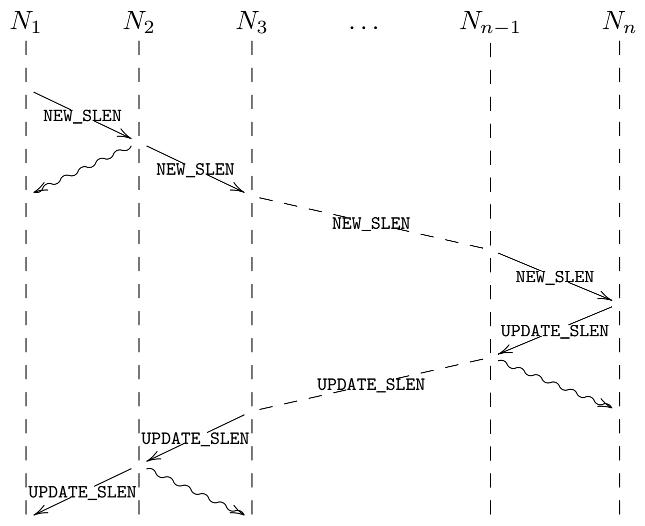


Fig. 10. Slot length update

$q_2 = 0.0002^\dagger$. In each simulation 1 MiB of data is transmitted from one end to another. The transmission is considered finished when the last data packet reaches the destination. For a particular packet size, we measure the average transmission length, which is the time it takes from transmission start to finished. In order to minimize effect of randomness, for each packet size, we simulate the transmission 1000 times.

Let $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{10}\}$ denote the BER vector in the network. We realize that two permutations of the same γ give us very close transmission length, especially when the number of runs increases. An example is given in Table III. In this table, $\gamma^{(1)} = \{q_1, q_2, q_2, q_1, q_1, q_2, q_2, q_1, q_1\}$ and $\gamma^{(2)} = \{q_2, q_2, q_2, q_2, q_1, q_1, q_1, q_1, q_1\}$.

Based on this remark, it is sufficient to consider unordered BER vectors. For a 10-hop network, there are 11 such vectors. We denote an unordered vector as $\gamma^{(k,10-k)}$, which indicates there are k components equal to q_2 and $10-k$ components equals to q_1. We do not consider homogeneous cases, $\gamma^{(0,10)}$ and $\gamma^{(10,0)}$, because our algorithm will give the optimal value. Therefore, there are 9 vectors left and we compare the following strategies on them:

- $l_{min} = \min_{1\le i\le 10}\{l_i\}$,
- $l_{max} = \max_{1\le i\le 10}\{l_i\}$,
- l_{opt}, the optimal packet length.

The actual optimal packet length is obtained by searching for a value between l_{min} and l_{max} that minimizes the average transmission length. From Table IV, we can see that the absolute difference between the optimal transmission length and our sub-optimal solution decreases when the number of bad links increase. In the case with one bad link, the transmission length difference is about 5000 seconds, which is around 6% the optimal value. Picking the maximum packet length is definitely not a good choice, because it doubles the transmission length. In homogenious cases, $\gamma^{(0,10)}$ and $\gamma^{(10,0)}$, the optimal packet length is identical to that by our protocol. Transmission length for $\gamma^{(0,10)}$ and $\gamma^{(10,0)}$ is 41049.22 s and 81791.66 s, respectively. The optimal packet length is 154 B for $\gamma^{(0,10)}$ and 890 B for $\gamma^{(10,0)}$.

†With a BER equal to q_1, a 400 byte packet is sucessfully delivered with probability 0.97. The probability drops to 0.53 if the BER equals q_2.

TABLE III
TRANSMISSION LENGTH OF PERMUTATED BER VECTORS

Packet size	$\gamma^{(1)}$	$\gamma^{(2)}$
154 B	80559.73 s	80592.47 s
246 B	76779.89 s	76779.89 s
338 B	80322.41 s	80300.01 s
430 B	87303.17 s	87666.13 s
522 B	97315.21 s	97373.67 s
614 B	109619.79 s	109857.24 s
706 B	124570.68 s	124666.21 s
798 B	141718.89 s	141906.59 s
890 B	163056.92 s	162460.06 s

TABLE IV
TRANSMISSION LENGTH OF DIFFERENT APPROACHES

BER vector	l_{min} = 154 B[1]	l_{max} = 890 B[2]	l_{opt}[3]
$\gamma^{(1,9)}$	79517.86 s	152957.48 s	246 B, 75098.03 s
$\gamma^{(2,8)}$	79481.56 s	152649.54 s	246 B, 75959.57 s
$\gamma^{(3,7)}$	80407.33 s	160615.21 s	246 B, 76291.62 s
$\gamma^{(4,6)}$	80593.56 s	162954.90 s	246 B, 76755.18 s
$\gamma^{(5,5)}$	80918.32 s	165205.35 s	246 B, 77029.72 s
$\gamma^{(6,4)}$	81062.06 s	166750.69 s	223 B, 77304.81 s
$\gamma^{(7,3)}$	81196.48 s	168820.18 s	223 B, 77552.11 s
$\gamma^{(8,2)}$	81399.95 s	170285.96 s	223 B, 77758.23 s
$\gamma^{(9,1)}$	81660.69 s	171625.69 s	223 B, 77998.37 s

[1] This column corresponds to the approach which uses the minimum optimal per-hop packet length throughout the network.
[2] This column corresponds to the use of the maximum optimal per-hop packet length.
[3] This column lists both the global optimal packet lengths and respective packet lengths.

V. Conclusions

In this paper, we introduce PTMAC which utilizes pipelined transmissions to improve performance of a string network. A pilot version was tested in a real sea test in 2012 and the result can be used as a comparison baseline for the development of future protocol. Also based on this data, we introduce an approach to improve the protocol performance by choosing the appropriate maximum packet length. The solution provided in this paper is distributed and scalable; however, it is suboptimal. Therefore, one direction for future studies is to improve the efficiency of the algorithm. The other direction is to study the fairness issue when multiple senders are present in the network.

Acknowledgment

The authors would like to thank Dr. Jeffrey Schindall from the Naval Research Lab for his support during the sea experiment and Dr. Lance Fiondella from the University of Connecticut for his review of the manuscript.

References

[1] I. F. Akyildiz, D. Pompili, and T. Melodia, “Underwater acoustic sensor networks: Research challenges,” *Ad Hoc Networks*, vol. 3, no. 3, pp. 257–279, May 2005.

[2] J. Cui, J. Kong, M. Gerla, and S. Zhou, "The challenges of building scalable mobile underwater wireless sensor networks for aquatic applications," *IEEE Network*, vol. 20, no. 3, pp. 12–18, May 2006.

[3] J. Partan, J. Kurose, and B. N. Levine, "A survey of practical issues in underwater networks," in *Proc. ACM WUWNet*, 2006, pp. 1–8.

[4] N. Chirdchoo, W.-S. Soh, and K. C. Chua, "Aloha-based MAC Protocols with Collision Avoidance for Underwater Acoustic Networks," in *Proc. IEEE INFOCOM*, April 2007, pp. 2271–2275.

[5] Z. Peng, Z. Zhou, J.-H. Cui, and Z. J. Shi, "Aqua-Net: An Underwater Sensor Network Architecture: Design, Implementation, and Initial Testing," in *Proc. IEEE OCEANS*, 2009.

[6] Chirdchoo, N., 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. VTC*, 2008.

[7] Z. Peng, Y. Zhu, Z. Zhou, Z. Guo, and J.-H. Cui, "COPE-MAC: A Contention-based Medium Access Control Protocol with Parallel Reservation for Underwater Acoustic Networks," in *Proc. IEEE OCEANS*, 2010.

[8] M. Molins and M. Stojanovic, "Slotted FAMA: A MAC protocol for underwater acoustic networks," in *Proc. IEEE OCEANS*, 2006.

[9] P. Xie and J.-H. Cui, "R-MAC: An Energy-Efficient MAC Protocol for Underwater Sensor Networks," in *Proc. WASA*, 2007.

[10] 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. IEEE INFOCOM*, 2009, pp. 1827–1835.

[11] K. Kredo II and P. Djukic, "STUMP: Exploiting Position Diversity in the Staggered TDMA Underwater MAC Protocol," in *Proc. IEEE INFOCOM*, 2009.

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

[13] A. Syed, W. Ye, and J. Heidemann, "T-Lohi: A New Class of MAC Protocol for Underwater Acoustic Sensor Networks," in *Proc. IEEE INFOCOM*, 2008.

[14] D. Pompili, T. Melodia, and I. F. Akyildiz, "A CDMA-Based Medium Access Control for Underwater Acoustic Sensor Networks," *IEEE Tran Wireless Comm*, vol. 8, no. 4, pp. 1899–1909, April 2009.

[15] B. Peleato and M. Stojanovic, "Distance Aware Collision Avoidance Protocol for Ad-Hoc Underwater Acoustic Sensor Networks," vol. 11, no. 12, 2007, pp. 1025–1027.

[16] X. Guo, M. R. Frater, and M. J. Ryan, "Design of a Propagation-Delay-Tolerant MAC Protocol for Underwater Acoustic Sensor Networks," *IEEE J Ocean Eng*, pp. 170–180, April 2009.

[17] Y. Noh, P. Wang, U. Lee, D. Torres, and M. Gerla, "DOTS: A propagation Delay-aware Opportunistic MAC Protocol for Underwater Sensor Networks," in *Proc. ICNP*, 2010.

[18] Gumstix Inc., "Verdex Pro XM4 specification," https://www.gumstix.com/store/product_info.php?products_id=209.

[19] Teledyne Benthos, "ATM-885 modem specification," http://www.benthos.com/acoustic-telesonar-modem-subsea-ATM885.asp.

APPENDIX
OPTIMAL HOP-BASED PACKET SIZE

Let $g(l)$ denote the objective function in Eq. 12:

$$g(l) := \frac{\alpha}{p_i^2(l)} + \frac{\beta}{l p_i^2(l)}. \tag{16}$$

Since $g(l)$ is a discrete function, its derivative is calculated as

$$\Delta g(l) = g(l+1) - g(l) = \frac{A_1(l) + A_2(l)}{B(l)}, \tag{17}$$

where

$$\begin{aligned} A_1(l) &= \alpha\, l(l+1)\left(p_i^2(l) - p_i^2(l+1)\right) \\ A_2(l) &= \beta\left(l p_i^2(l) - (l+1) p_i^2(l+1)\right) \\ B(l) &= l\,(l+1)\, p_i^2(l)\, p_i^2(l+1). \end{aligned}$$

Notice that since $A_1(l) > 0$ and $B(l) > 0$, the sign of $\Delta g(l)$ is determined by that of $A_2(l)$. Let us consider two cases of $A_2(l)$.

Case one: when $A_2(l) \geq 0$, we have $\Delta g(l) \geq 0$. This means that $g(l+1) \geq g(l)$ or in other words, $g(l)$ is an increasing function. In this case, l must satisfy

$$\begin{aligned} & l\, p_i^2(n) - (l-1)\, p_i^2(l+1) \geq 0 \\ \Leftrightarrow\ & l\, p_i^2(l) \geq (l+1)\, p_i^2(l+1) \\ \Leftrightarrow\ & \frac{l}{l+1} \geq \frac{p_i^2(l+1)}{p_i^2(l)} = \frac{(1-\gamma_i)^{2(l+1)}}{(1-\gamma_i)^{2l}} = (1-\gamma_i)^2 \\ \Leftrightarrow\ & l \geq \frac{(1-\gamma_i)^2}{1-(1-\gamma_i)^2}. \end{aligned} \tag{18}$$

Case two: when $A_2(l) < 0$, we immediately have

$$l < \frac{(1-\gamma_i)^2}{1-(1-\gamma_i)^2}. \tag{19}$$

In this case, we need to find a value of l such that $A_1(l) + A_2(l) = 0$:

$$\begin{aligned} & \alpha\, l\,(l+1)\left((1-\gamma_i)^{2l} - (1-\gamma_i)^{2l+2}\right) \\ & \qquad + \beta\left(l\,(1-\gamma_i)^{2l} - (l+1)(1-\gamma_i)^{2l+2}\right) = 0 \\ \Leftrightarrow\ & (1-\gamma_i)^{2l}\left(\alpha\, l\,(l+1)\left(1-(1-\gamma_i)^2\right)\right. \\ & \qquad \left. + \beta\left(l - (l+1)(1-\gamma_i)^2\right)\right) = 0 \\ \Leftrightarrow\ & \alpha\, l\,(l+1)\left(1-(1-\gamma_i)^2\right) + \beta\left(l - (l+1)(1-\gamma_i)^2\right) = 0 \\ \Leftrightarrow\ & \alpha\, l^2 + (\alpha+\beta)\, l - \frac{\beta\,(1-\gamma_i)^2}{1-(1-\gamma_i)^2} = 0 \end{aligned} \tag{20}$$

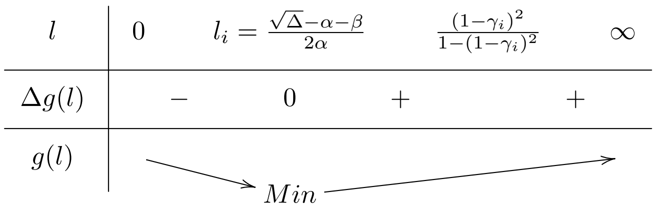
The above equation is quadratic with respect to l, whose determinant is

$$\begin{aligned} \Delta &= (\alpha+\beta)^2 + \frac{4\alpha\beta\,(1-\gamma_i)^2}{1-(1-\gamma_i)^2} \\ &= \alpha^2 + \beta^2 + 2\alpha\beta\,\frac{1+(1-\gamma_i)^2}{1-(1-\gamma_i)^2} \\ &> \alpha^2 + \beta^2 + 2\alpha\beta = (\alpha+\beta)^2 > 0 \end{aligned} \tag{21}$$

This means that Eq. 20 has two roots; however, note that l is positive since it represents the packet size. Thus, the appropriate root is

$$l_i = \frac{\sqrt{\Delta} - \alpha - \beta}{2\alpha} \tag{22}$$

One can also easily confirm that l_i satisfies Eq. 19. Therefore we can combine the two above cases as in Fig. 11, from which we see that l_i minimizes the value of $g(l)$.



l
0
$l_i = \frac{\sqrt{\Delta}-\alpha-\beta}{2\alpha}$
$\frac{(1-\gamma_i)^2}{1-(1-\gamma_i)^2}$
∞
$\Delta g(l)$
$-$
0
$+$
$+$
$g(l)$
Min

Fig. 11. Minimum of $g(l)$