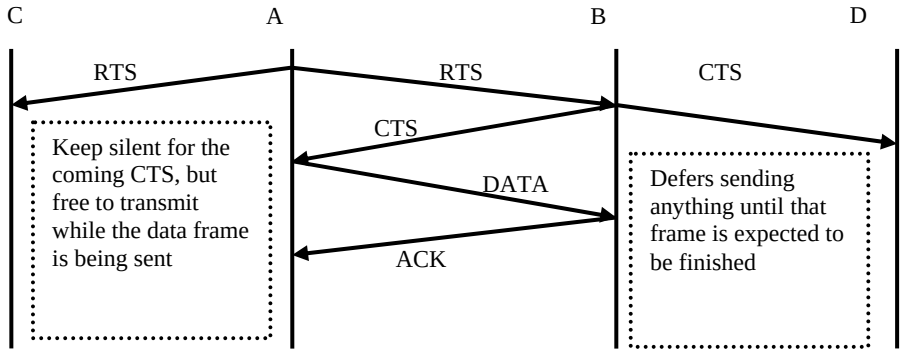
MAC Protocol Implementation on Atmel AVR for Underwater Communication

- Report 3-

Protocol Implementation

Based on the survey, for small packet size in a small network, Aloha can bring more throughput. because there is no much collision in small network, senders can transmit whenever they have data to be sent. Senders should resend the data only if there is no ACK back. The waiting time to resending must be random, otherwise the same packets will collide over and over. At the receiver end, it sends an ACK to the sender if the received packet is error free. This aloha protocol layer has been finished.

In addition, because in some cases where the collision increases and more data to send, the MACA protocol will be a potential better choice. So I also implemented an improve MACA(RTS/CTS) protocol to make a comparison. This improved MACA will send back an ACK after the correct data, which can reduce the waiting time of retransmission. The basic idea is listed below.



ALOHA protocol:

if databuffer is not empty
compose and send a packet;
listening in the channel for a period (period = wait)
if received a packet in a period(time<wait)
if the packet's destination matches with the its device ID,
if it is an ACK
Succeed. Check the databuffer
if not empty, compose and send next packet
set 'wait' to round trip time
else if it is a DATA
if CRC is right
put data to RX databuffer
compose and send an ACK packet to this node
else CRC is wrong
drop the packet, set 'wait' to round trip time
else if there is an error
drop the packet, set 'wait' to round trip time
else if the packet's destination doesn't match with its device ID
if it is an ACK

do nothing
else if it is a DATA
keep silent for the ACK to be sent back
set 'wait' to one way time

else if period ends (time==wait)
send the same DATA packet to the same node
set 'wait' to a random time+RTT

MACA (RTS/CTS) protocol:

if databuffer is not empty
compose and send a packet;
listening in a period (period = wait)
if received a packet in a period(time<wait)
if the packet's destination matches with the its device ID,
if it is an ACK
Succeed. Check the databuffer
if not empty, compose and send next packet
set 'wait' to round trip time
set status=STATUS_WAIT_CTS;
else if it is a DATA
if CRC is right
put the data to RX databuffer
compose and send an ACK packet to this node
else CRC is wrong
drop the packet, set 'wait' to RTT
else if it is a RTS
compose and send a CTS to this node
else if it is a CTS
compose and send a DATA packet to this node
set status=STATUS_WAIT_ACK;
else
set 'wait' to round trip time
else if the packet's destination doesn't match with its device ID
if it is an ACK
do nothing
else if it is a DATA
keep silent for the ACK to be sent back
set 'wait' to one way time
else if it is a RTS
Keep silent for the coming CTS, but free to transmit while the data frame is being sent, set 'wait' to one way time
else if it is a CTS
Keep silent for the coming DATA. Defers sending anything until that DATA is finished by some other node.
set 'wait' to (one way time + data transmission time)

else if period ends (at time==wait)
if status == STATUS_WAIT_CTS
send the same RTS packet to the same node
set 'wait' to a random time+RTT
else if status == STATUS_WAIT_ACK
send the same DATA packet to the same node
set 'wait' to a random time+RTT

Packet Format

ALOHA:

Data Packet

ID (4 bits)	Dest (4 bits)	Source (4 bits)	Length (4 bits)	Data (8*n bits)	CRC (4 bits)

ACK

ID (4 bits)	Dest (4 bits)	Source (4 bits)

MACA(RTS/CTS):

Data Packet

ID (4 bits)	Dest (4 bits)	Source (4 bits)	Data (8*n bits)	CRC (4 bits)

RTS/CTS

ID (4 bits)	Dest (4 bits)	Source (4 bits)	Length (4 bits)

ACK

ID (4 bits)	Dest (4 bits)	Source (4 bits)

For the aloha, the data packet has a ID filed, which tells the packet type (Data or ACK), a Dest filed, which tells the packet destination, a source filed, which tells where the packet is from, a length filed, which tells the data length, and a CRC field, which is used to check if the data packet is correct. The ACK packet is pretty simple, which only contains ID, Dest and Source.

For MACA protocol, the data and ACK packets are similar as Aloha. The difference is the RTS and CTS packet have a length filed, which is used to notify the destination the data size and notify the other nodes how long they should keep silent.

The CRC in the packet is a Polynomial Code to make sure there is no data error in the packet, otherwise, the sender will send that packet again. Here I use generator $G(x)=x^4+x+1$ ->(10011).

I didn't use error correction mechanism in the datalink layer, only focusing on the estimation of protocol performance at this initial implementation. However, the convolutional code could be used in the datalink layer as the future work.

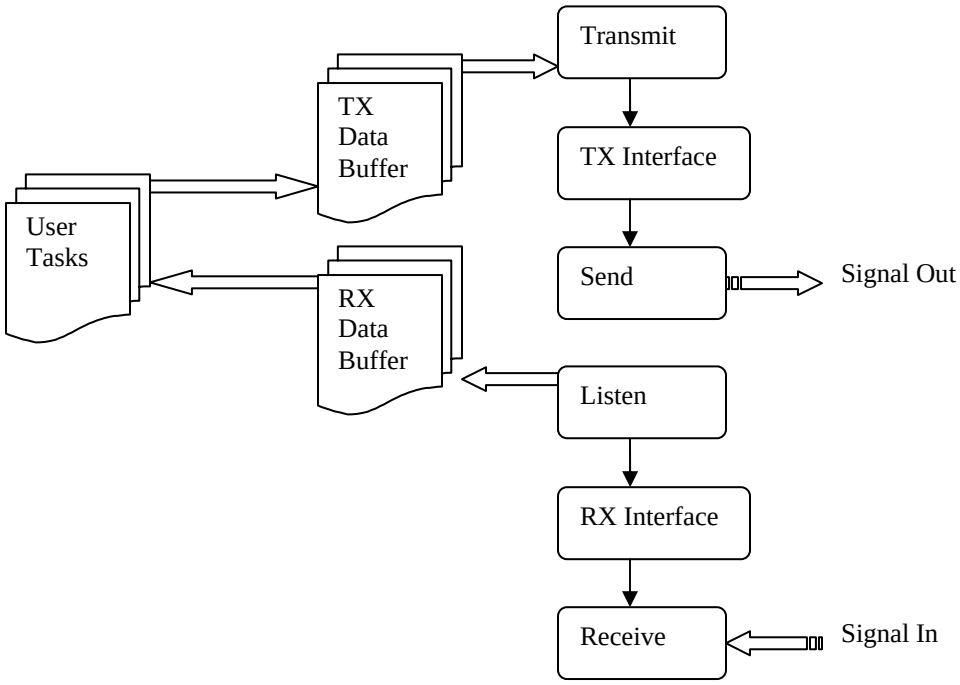
System Implementation

```
While(true){
        if(Databuffer!=Empty)
                Transmit();
        if(TaskReadytoRun)
                Task();
        if(RequestforISR)
                ISRtask();
        ListenChannel();
…
}

Interrupt ISR();
…
```

The operating system or even some schedulers are too complex and expensive for this device. I use a Round-Robin scheduling for the protocol implementation, with interrupt enabled. The

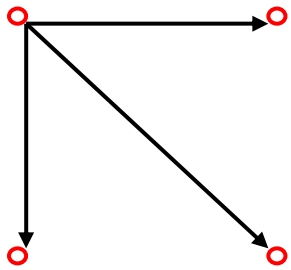
application tasks could be serviced in the loop or be configured as an ISR. They can put the data into TX data buffer. The transmitting task will check if there is any data to send in the data buffer. The listening task will keep checking the channel if there is a coming packet. If there is a coming packet, the listen task will end earlier, otherwise it will return at the end of the specified period. The transceiver device is a half- duplex system as is usually the case in many acoustic nodes, which means listening while transmitting is not possible. So when it is transmitting, it shouldn't be interrupted by any events so the interrupt is disabled during that time and incoming packets will be lost due to this half-duplex mode.



Transmit and listen are protocol layer functions which will call TX & RX separately. The TX and RX are the interfaces and they will call physical layer functions to send or receive the signals.

Performance Measurement

To estimate the protocol performance in this small network, I will focus on the throughput. As we see on the right, I will build a simple grid network and each node will unicast the packets to all the other three nodes sequentially. In the end, I will collect the successfully received packets from all four nodes within a period, and then get the number of received packets per minute as the throughput.



Experiments

The experiments will be finished in three steps.

The first experiment is to test whether the system could work using two nodes. One works as sender, the other is a receiver. This could be done indoors. But the problem is that the distance between nodes for the indoor test can't be long enough, so the delay will be very short. If we use LEDs or the small display as status indicators, we can't tell the protocol works or not since the status will change very fast. My solution is to use UART and a PC. By setting up a communication with UART, the transceiver can talk to the HyperTerminal. We can output the RX/TX status to the computer to verify if the system works. This first step has been done.

The second experiment is to make four nodes work together and test if the protocol works properly with collisions. This also could be done indoors. By using UART, we observe how the status changes, how the packets are transferred between nodes, and eventually fix the potential problems in the protocol.

The third step is to test the performance of the protocols in open water, and collect the results.