MAC Protocol Implementation on Atmel AVR for Underwater Communication

- Report 4-

(updated 4/19)

Problem Summary

There are lots of problems and hurdles to face during the implementation. Below are some major problems as well as their corresponding solutions I proposed.

P1: Debugging Instrument

The protocol implemented on this transceiver is used for underwater communication. Before testing it in the real world, we have to make sure the protocol is properly implemented. At the initial stage, there is no need to use transducers to set up the communicate, but some wires would work. Connecting the two nodes through wire could help to verify if communication works under the protocol. But the problem is the data is substantially fast using wire than acoustic underwater communication, which makes the status change very fast so it is hard to observe the current status on the LEDs or even the small display.

My solution is to set up a UART communication with PC, and output the status information to the HyperTerminal on PC through UART. More than that, in the way I could debug the system conveniently with all necessary information printed out. However, the debugging is still non-trivial.

P2: Starvation

There is a scenario, when node A is sending a packet node B, but the received packet has some error by checking CRC field. Previously my design was to let B do nothing but wait for a new RTT time, because B knows A will send again if A doesn't receive an ACK and in the way it will keep the channel clear. Although in some case, this will bring more through put, however this will lead to a starvation for B if there are more nodes trying to send data to B and such CRC error happens often. B will never have the chance to send its data. The solution is not to reset B's RTT waiting time if it is not an ACK (Same for other wrong packet). The corresponding code is modified as follows:

```
……
if received a packet in a period(time<=RTT)
        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 RTT
            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
else if there is an error
drop the packet
else if the packet's destination doesn't match with its device ID
do nothing(no RTT reset)

P3: Flexible Length Packet Receiving

In my packet, there is no the stop-bit field. In my design, I have to tell the RX function how many bytes it should receive next time. This could reduce the overhead, but for the receiver, it is hard to find when it is the end of a packet since packet and ACK have different size. A node could receive either a packet or an ACK at any time. My solution is to receive fixed size data at the beginning of any receiving process. Because the packet header is the same length as an ACK, I set the fixed size to be an ACK size. Based on the received information, I decode the header and check if it is an ACK or a data packet. If that is a data packet, then I use the length information contained in the header to tell RX function how many more bytes are still needed.

P4: CRC Consideration

At the beginning, I used a 4 bit CRC to check the data error, which turns out not to be effective for long data and also the extra half-byte style is not common in communication. I changed to a CRC-8 standard ($x^8 + x^2 + x + 1$). This will add one more bytes after the data field.

P5: Hardware Limitation

This transducer uses Atmega168 microcontroller, which only has 1K RAM and 16K flash. This limits the system design have to be small. Because receive, transmit functions and some decoding algorithm are used, no complex scheduler and application is allowed, otherwise "The contents of the HEX file doesn't fit in the selected device". Also, because I would like to make a comparison between Aloha and MACA, I have to reuse and modify some function to make them fit into the microcontroller together. A better solution is to use different optimization level, for example, for Goertzel algorithm, which is a critical path, I use –O3 to increase the speed of the algorithm but use –Os on the other files to reduce the final size. Below are some results, as we can see, with same optimization level, the executable can't be programmed to the Atmega168.

Text	data	bss	total	
14986	302	141	15429	(different optimization levels)
16380	338	109	16827	(same optimization level)

P6: Random Number Generator

In the aloha protocol, if a packet is dropped or destroyed, the sender should wait a random amount of time and sends it again. The waiting time has to be random, otherwise the same packet will collide over and over with other nodes. But do we really need a random number generator function like rand() to create those number? My solution is to ADC or Timer Counter. Timer counter register contains a changing number and the least significant bits of ADC are random numbers.

P7: No Response Problem

In some cases, when there are more than two nodes, the collision number increases and there may be congestion happening to some nodes, so the network will be stuck at that point. My solution is to use a MAX_RETRY_NUMBER. Once this number is reached, then either give up this node and switch to next one or make other smart choices. In my test, I use the former one.