**Retransmission handling**

One of the main objectives of this exercise was to consider a smart way to handle ARQ(Automatic repeat request). In other words, a way to check on the sink or the source, whether or not all messages were received on the sink. If a message was lost, the sink was to inform the source that one/multiple message(s) had to be retransmitted. The process of picking the right protocol for retransmission, involved considering many different tradeoffs. This section will go through the tradeoffs and explain in detail why and how the retransmission/ARQ is handled.

Figure x shows the final design from all the considerations and the following subsections will describe how we came up with this design.

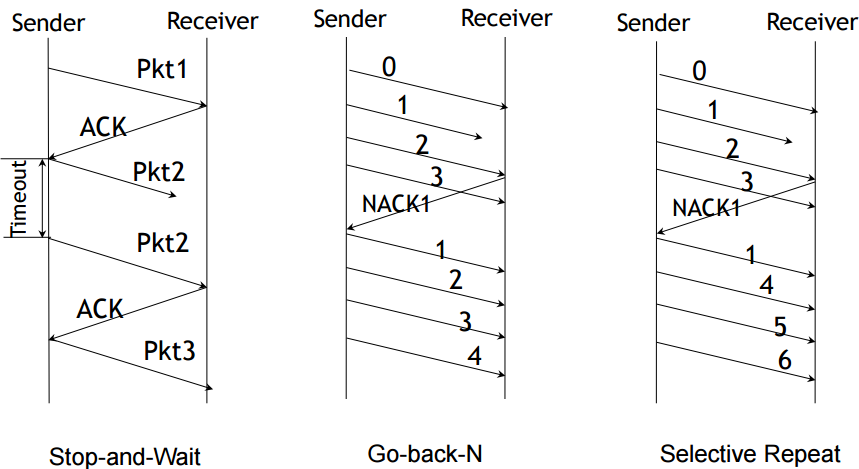


**ACK and NACK(bitmap)**

One of the first tradeoffs to consider when working with retransmission was positive acknowledgement vs only sending NACKs. In order to check the message sequence, a message counter was added to every data message such that the sink was able to know which messages were lost. The following table shows the pros and cons of using ACKs.

|  |  |
| --- | --- |
| PROs of ACKs | CONs of ACKs |
| More agility as the protocol can check whether the link between the sink and source has been lost.(This means it can stop sending data if no one receives the data) | Less energy efficient, as more messages are sent and received |
| In some cases less memory usage, as acknowledged messages can be removed from memory.(Depends on the implementation) | More collisions to avoid |
|  | Higher congestion |
|  | Higher complexity as a timer is needed on the source for every acknowledgement |

Even though positive acknowledgements would make the protocol more agile and would make it possible to check that the link between the source and the sink was lost. We chose to go with only sending NACKs and using the Selective repeat approach as seen on figure x.



This choice minimizes the amount of sent messages and maximizes the energy efficiency. One of the drawbacks of this decision is the fact that we have to send link requests once in a while to make sure that the link between the source and the sink is still accessible.

An improvement that could be made to the protocol, would be to start recording the messages when a lost message was discovered and then, instead of sending a NACK on the lost message, multiple NACKs and ACKs could be sent in a bitmap in a single message after N messages were received or after a given transmit window. This would reduce the amount of NACKs even further and optimize the energy efficiency even more. The problems of using a technique like the sliding window is that the retransmission will be delayed by at least the time of the sliding window and also the complexity of communicating the lost messages is increased.

**Storing retransmission data on relay node(End-to-end vs link-layer retransmission)**

As described in section x we implemented sleeping on the relay node to maximize the energy efficiency of the protocol. This means that in order to know that the data is available from the relay, we would need to have synchronization between the relay and the sink node to know whether or not the relay is sleeping. The synchronization would add a lot of complexity to the protocol. Instead we chose to broadcast the NACK and use it as a new indicator of the link quality between the nodes. So within our retransmission message in addition to the message counter, we also added the RSSI and LQI values so that we could get new gradients for the sensor network.  
The relay will always listen for retransmissions(if it is awake) and reply the retransmissions to the source node. By doing so, we can have a new indicator of the link quality through the relay and the link directly to the sink node. This implementation also reduces the amount of link requests necessary from the source node, as a retransmission can cause the source node to change tactic.  
This also means that we do not store any data on the relay node, and the protocol will always do an end-to-end retransmission in order to fetch the lost data messages.

**When to ask for retransmissions?**

If this protocol was to run on a binary symmetric channel, any time for retransmission would be good. But as we’re working with fading channels we have to consider: “when is a good time for retransmission?”.  
A probing protocol could be implemented to check if a message could be acknowledged by the receiver and then go to normal mode to send the message. But as we only send messages every 200ms, we actually know that there is a window of time after receiving a message before another message is sent (Assuming no one else is using this channel). The simple approach we use is: send a retransmission message after a data message is received. This has multiple drawbacks as we cannot assume that no one else is using the channels. Also if the link between the sink and the source node is lost and no messages are received, we will not send any retransmissions. Another problem is that if we lose more messages than we receive, we will never be able to catch up with the retransmissions.  
This could be solved by having a process that sends retransmissions with an interval that is shorter than the 200ms.

**Max number of retransmissions and size of retransmission buffer?**

The last consideration to be made was how many retransmissions to be allowed before discarding the message from the retransmission buffer. This is a very domain specific consideration and has a lot to do with the design principle of adaptive fidelity and accuracy. Some applications can live with imprecise and approximations of values, while other applications have critical requirements when it comes to precision and accuracy. Some applications only need multiple messages when an event happens while other applications need a steady flow of data.  
In this protocol we are sensing temperature data and are doing so quite often, which means the protocol could look at earlier values and check if enough information was gathered about the temperature in order to discard the information. Right now it is implemented in such a way that a retransmission counter increments for every retransmission and it is easy to remove the message from the buffer if it raised above a given threshold.

**Sleeping on sink and source node and the impact of retransmissions**

Currently the protocol does not include sleeping on the source and sink nodes. But as we can see from the calculations a lot of power could be saved by sleeping instead of idle listening. If sleeping on source and sink nodes were implemented the retransmissions would have to make the nodes stay awake. This would make the consideration of going through the delay a lot more favorable than it is with the current implementation of the protocol.