# Alexander Einshøj - Assignment 3 - 2301
Alternating-Bit-Protocol(ARQ)

Introduction

This paper describes an implementation of the Alternating-Bit-Protocol(ABP)

Technical Background

In order to implement ABP, knowledge about certain aspects of the transport layer is essential.

Transport layer

According to the OSI model, there are 7 layers, of which only 5 are in actual use(Presentation and Session is not). The transport layer communicates with both the application layer from above, and the network layer below. The task of the transport layer is to reliably transfer data between points on a network.

Alternating-Bit-Protocol

The alternating-bit-protocol, also known as stop-and-wait ARQ(Automatic Repeat-Request) only sends one frame at a time, and waits for an ACK from the receiver before sending any more packages. If an ACK is not received within a reasonable time, then the sender provides the same frame again until it is ACK'ed by the receiver.

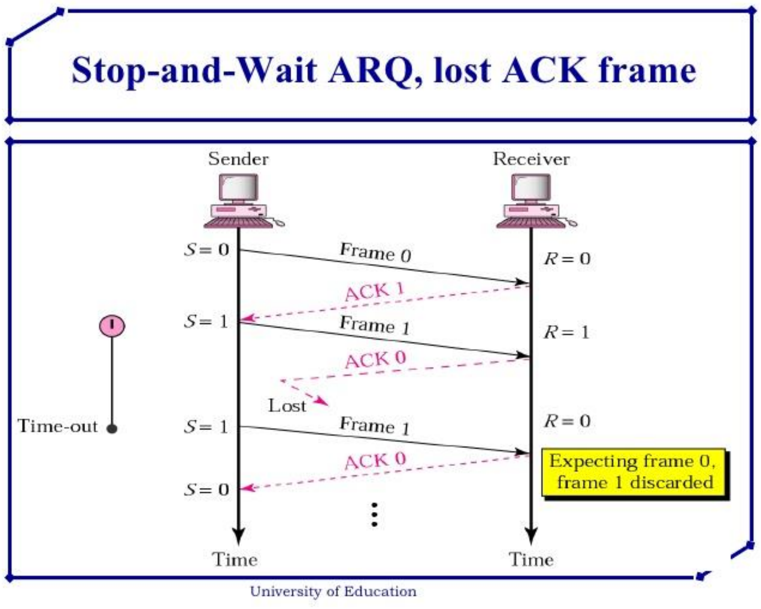

Figure 1, Stop-and-wait ARQ

Figure one shows how a transport package is acknowledged, and also what happens when encountering a loss.

ACK & SEQ

When the transport layer transfers packages accross the network, it is necessary for it to know whether the correct packet has arrived. Since both parts of a transfer could be a host, it's also important to know what package you are expecting to receive. This is why the use of ACK's and SEQ's are vital. An ACK, or an acknowledgement, is in the alternating-bit-protocol a bit indicating the package that's been sent, has been received in the other end.

A SEQ(Sequence number), is an ACK, also a bit being sent with the packet, but indicates what package is being sent, instead of what is expected.

Queue

The queue is in this implementation a linked list that handles packages sent from the application layer.

Design

For this assignment, one were to choose to either implement Go-Back-N or Stop-And-Wait ARQ, and the author chose the latter.

There are several situations one has to be aware of when implementing Stop-And-Wait ARQ. What do you do if a packet is lost? What happens if an ACK never reaches the sender? How do you know if the receiver got the correct data? When do you change the ACK or SEQ of a packet? How do you know you got the correct data?

All of this needs to be handled by your implementation.

Implementation

The code was written in C.

Transport tasks

There are really not too many variables to keep in mind while implementing the alternating-bit-protocol, but as seen from the design section, it's essential to know exactly what you're doing and when. When the application layer sends down a package to the transport layer, there is not much to consider. All you have to know is whether there is any current elements in the queue and if you are still waiting for an ACK from the receiver. If there is an existing queue with elements in it, there is nothing else to do than to add the current package to the queue.

If there is no queue, it means all packages sent are received and ACK'ed, and we can safely proceed with sending the current package to the receiver.

When a package is received from the network layer, there are multiple options:
1. An ACK is received, in which case we do pop the first item of the queue if there is one.
2. The wrong ACK is received, which means the receiver has not gotten the last package, so it needs to be resent.
3. A normal data package is received, whereafter we negate the layer of the ack(and also the next package) and send it up to the application layer
4. The ACK never reaches the sender, and it needs to be retransmitted.

Discussion and Results

The current implementation does not fulfill all the requirements set for the assignment. However, it is but a short stretch until it's fully implemented.

Checksum has not yet been implemented, but doing it would be rather easy. Checksum is something you retrieve after using a hashing algorithm on the data being sent, such as md5. This way you always know whether the data received is the correct data, because hashing the same data will always have the same result, which means you can check on both ends whether it's a match.

The implementation does not fully work with packet loss either, although it shouldn't be more than a minor bug fix at this point, as most situations are accounted for.

Figure 1 shows the fundamental idea of how to handle packet loss, which is how the author has chosen to implement the feature.

The Stop-and-wait protocol is an ARQ, alongside Go-Back-N and Selective repeat, which all are defined by different structures and capabilities. While stop-and-wait has a window size of 1 at sender and receiver at any given time, Go-Back-N has a window size of N. This difference sounds immediately quite subtle, but the consequences of this are huge. If the window size is defined by N, multiple packages can be sent at once before requiring an ack to proceed, which means more data can be transferred at the time. When dealing with packet loss, the number of packets sent would be minimal by choosing Go-Back-N.

Stop-and-wait is very rarely used in practice, because how slow and sadly unreliable it is. Since it is known as the alternating-bit-protocol, there are only two different sequence numbers, as opposed to other ARQ' s, where it could be much more.

An important issue to raise, is whether the current implementation is considered reliable data transfer or not. The fundamental issue with alternating-bit-protocol is that it can never be fully reliable, as mentioned above. Further on, checksum isn' t implemented, thus the program won' t be able to correct the data. I have not yet been able to figure out the issue my implementation has with packet loss, but the situations where the bug occurs, has been narrowed down quite a bit. It seems as if the loss rate is very high, the implementation won' t handle more than one bidirectional package, but with a low loss-rate it handles a lot more. This is only the case if the data is bidirectional, and I can only assume that the issue has grounds in a situation I have not yet considered or discovered.

This is also only an issue in the case of multiple messages(2 or more), which tells us that something happens around the time the first ACK is sent back, I presume.

If the program is run with no loss and no corruption, bidirectional or not, it should handle about 1000-2000 messages before the program terminates. It always terminates with the same message, that the data received does not match the expected data. That is interesting, as corruption is turned completely off. My assumption is that at some point, either the ACK or SEQ of the package or layer is wrong, in some special case that always occurs after a certain amount of packages.

References

http://image.slidesharecdn.com/lec-1819-100419044113-phpapp02/95/flow-error-control-20-728.jpg?cb=1271652132