Computer Networks
Link initialization

Saad Mneimneh
Computer Science
Hunter College of CUNY
New York

- hello
- hello/ack
- ack

three-way handshake

1 Introduction

All ARQ algorithms assume some sort of initialization of the DLC link. We did not explicitly emphasize this fact before to keep the illustration simple. However, for the ARQ to operate correctly, we require that initially:

- no frames are traveling on the link
- SN and RN are both equal to 0 at both ends of the link

One might think that this problem of initialization is trivial: simply initialize the DLC at each end of the link when the link is first physically placed, and then use an ARQ algorithm forever. Unfortunately, the link can fail.

- good news (link view): the ARQ algorithm guarantees that frames not received correctly before the failure would be received after the failure.
- bad news (network view): when a link fails for a long period of time, it becomes necessary for the transport and/or network layers to take over and set up alternative paths for packets that were not delivered. When the failed link eventually returns to operation, the higher layers restore the path through the link, thus causing the DLCs at both ends to possibly deliver duplicate packets.

Therefore, both DLCs should view the link as being segmented into an alternation of up and down periods. The DLCs must then be properly initialized at the beginning of each up period. At the end of an up period, however, there may be frames in

the process of being communicated that are not received. These frames will not be received upon the start of another up period because the DLCs will be reinitialized. But that's ok.

There is a problem in the definition of an up period. In particular, the DLCs at opposite ends of the link must agree at any given instant whether the link is up or down. But we have seen that such an agreement is theoretically impossible to achieve. Therefore, instead of resolving this issue in the abstract, we present simple protocols for initializing and disconnecting the link simple to ARQ in concept.

1.1 Master-slave link initialization

With the Master-slave protocol, one DLC, say A, is in charge of determining whether the link is up or down. A then informs B at the other end of the link whether the link has changed from down to up or up to down. The sequence of messages that go from A to B is simply an alternating sequence of initialize and disconnect messages, as seen in the figure below. For each such message, B responds with the appropriate ack.

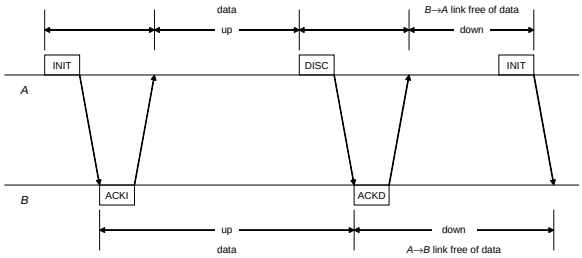

Figure 1: Master-slave link initialization

Conceptually, this can be interpreted as a stop and wait algorithm modulo 2. An initialize message (denoted by INIT) corresponds to a frame with $SN = 1$, and a disconnect message (denoted by DISC) corresponds to a frame with $SN = 0$. The corresponding acks of B are denoted by ACKI and ACKD, respectively. ACKI corresponds to $RN = 0$ (i.e. acknowledging the INIT message $SN = 1$), and ACKD corresponds to $RN = 1$ (i.e. acknowledging the DISC message $SN = 0$).

Note that the decisions to initialize and disconnect (made at A) are not part of the DLC, and should be viewed as coming from a higher layer, possibly by making measurements of the link. The decisions to send DISC or INIT messages are part of the DLC, however, and rely on having already received an ack for the previous message. For instance, there is no way for A to abort the initialization before receiving ACKI from B. Naturally, A can start to disconnect as soon as ACKI is received, but must continue to send INIT until receiving ACKI.

1.2 Balanced link initialization

It is more desirable if both A and B can initialize/disconnect the link because either one (at some higher layer) can be monitoring the link. At the very least, there is a conceptual simplification in not having to decide which one is master. The essential idea of a balanced initialization protocol is to use two master-slave protocols with A playing the master for one, and B playing the master for the other.

For simplicity, we assume that each INIT or DISC message from a master (on one side of the link) also contains a piggybacked ACKI or ACKD for the slave (on the other side of the link), but we regard the acks as being acted on first. ACKI and ACKD messages can also be sent as stand alone message upon receiving an INIT or a DISC message respectively. As the following figure shows, each side in its role as a slave acks each INIT and DISC message from the other side; also each side in its role as a master continues to transmit INIT and DISC messages until acked.

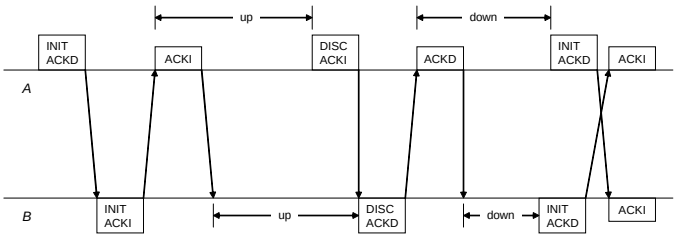

Figure 2: Balanced link initialization

The initialization part of this balanced protocol, i.e. sending INIT, waiting for INIT and ACKI from the other side, and responding to INIT with ACKI, is often called the **three-way handshake**. The reason for this terminology is that the ACKI from the other side is usually piggybacked on the INIT, making a three-message interchange. It is possible, however, that both sides start to initialize independently, and then all four messages must be sent separately as seen on the right side of Figure 2. In any case, either one can decide to initialize or disconnect the link.

- As a master: To initialize the link: send INIT+ACKD, wait for ACKI, send ACKI. To disconnect the link: send DISC+ACKI, wait for ACKD, send ACKD.
- As a slave: Upon receiving INIT+ACKD, send INIT+ACKI or simply ACKI if sent INIT+ACKD already as a master. Upon receiving DISC+ACKI, send DISC+ACKD or simply ACKD if sent DISC+ACKI already a as master.

Here's a state diagram for initializing the link in the form of event/action (ϵ means empty event or empty action):

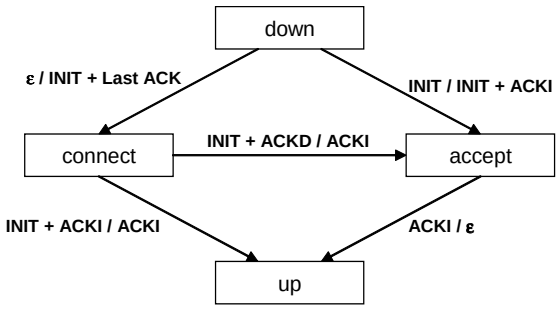

Figure 3: Initializing the link

From the above diagram, we can see that if the last ACKI was lost, A thinks the link is up but B thinks it is not up yet. In this case, B will eventually resent

INIT+ACKI. But what if A disconnects the link before B knows it went up? In this case, the last ack (ACKI) with DISC will tell (acks are acted upon first).

Here's a state diagram for disconnecting the link in the form of event/action (ϵ means empty event or empty action):

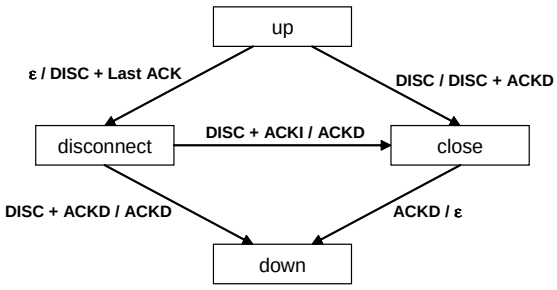

Figure 4: Initializing the link

From the above diagram, we can see that if the last ACKD was lost, A thinks the link is down but B thinks it is not down yet. In this case, B will eventually resend DISC+ACKD. But what if A initializes the link before B knows it went dow? In this case, the last ack (ACKD) with INIT will tell (acks are acted upon first).

References

Dimitri Bertsekas and Robert Gallager, Data Networks
Larry Peterson and Bruce Davie, Computer Networks: A Systems Approach