# CS164 Notes on *"Verification of Link-Level Protocols" by D. Knuth*, published in BIT vol. 21 (1981), pp. 31-36.

## 1. Background

Recall that Alice's data-link controller needs to send a never-ending sequence frames

$$M\,[0],\ M\,[1],\ M\,[2],\ M\,[3]\ \ldots$$

to Bob's data-link controller reliably, across a noisy channel.

Based on our previous discussions, we know that Bob must send back some form of *acknowledgement* to Alice, telling her whether he has received a particular frame, so she can decide if she needs to send Bob another copy. Moreover, since frames and/or acknowledgements could be lost, they must agree on some method to identify which specific frame they are talking about, rather than assuming they will always agree on what is "this frame" or "the last frame". Otherwise, Alice and Bob might get confused in a way that causes Bob to miss a particular frame (if Alice thinks he received it when he did not, and never sends it again), or to accept the same frame multiple times (if Alice thinks he missed it when he did not, and sends another copy of the same frame anyway).

The easiest way to handle this problem would be to tag each frame, $M\,[j]$, with its actual sequence number, $j \in \boldsymbol{I}^+$. However, this naïve approach is guaranteed to fail eventually because the space available in the header of the frame is limited, whereas the space occupied by an actual sequence number, $j$, is $O\,(\log_2 j)$, which keeps growing forever. For example, if the frame header reserved 8 bits for storing the tag, Alice could send only 256 frames before the tag field overflows. If we increased the tag field to 16 bits, Alice could send a lot more frames (i.e., 65,536) before the problem occurred. However, if Alice tries to send 100 frames per second, the tag field will overflow in less than 11 minutes.

Clearly, we need a method for tagging frames that allows Alice to keep transmitting forever without running out of space in the (fixed size!) tag field. We already saw one such method in the Alternating Bit Protocol[1], which uses a 1-bit tag but forbids Alice from sending more than one frame before she must stop and wait for some feedback about the fate of "this frame" from Bob. Since this stop-and-wait strategy would be exceedingly wasteful if the round-trip time between Alice and Bob were much larger than the transmit time for a frame, there is a need for Alice to send multiple frames to Bob without waiting for a response from Bob, and at the same time using only a small number of bits to represent the tag field.

## 2. Detailed Description of the Problem

### *2.1 Behavior of Alice*

- Alice has one local variable, $A$, signifying that she knows Bob has correctly received (at least!) the first $A$ consecutive frames that she has sent since the

[1] Bartlett et al., "A Note on Reliable Full-Duplex Transmission Over Half-Duplex Links", *Communications of the ACM*, May 1969, pp. 260-261.

beginning of time. Note that the numbering of the frames starts at zero, so this is equivalent to saying that Bob must have received all of the frames:

$M\,[0], M\,[1], M\,[2], \ldots, M\,[A–1]$

- Alice can execute either one of two procedures whenever she likes:
    - **a1**: If she is not already in the middle of transmitting another frame, she can begin transmitting frame $M\,[j]$, for any frame whose sequence number falls within the range $j \in \{A, \ldots, A+k–1\}$. Thus, in contrast to the Alternating Bit protocol (where she is only allowed to send the *first* unacknowledged frame), she is free to choose *any* frame from among the next $k$ unacknowledged frames.
    - **a2**: If there are any incoming acknowledgements waiting, she can receive the first one, and use the value, $b$, that it contains as the new value for her local variable, $A$, i.e., $A \leftarrow b$.

### *2.2 Behavior of Bob*

- Bob has one local variable, $B$, signifying that he has correctly received (and decided to keep!) the first $B$ consecutive frames that Alice has sent since the beginning of time—and possibly also some additional messages, which are either not consecutive with the first $B$ or he has simply chosen not to reveal this extra information yet. Again, since the numbering of the frames starts at zero, this is equivalent to saying that Bob must have received all of the frames:

  $M\,[0], M\,[1], M\,[2], \ldots, M\,[B–1]$

- Bob can execute either one of two procedures whenever he likes:
    - **b1**: If he is not already in the middle of transmitting a previous acknowledgement to Alice, he can send a new acknowledgment to Alice containing the current value, $b$, of his local variable $B$.
    - **b2**: If there are any incoming frames waiting, and he is not in the middle of receiving an earlier frame, he can decide to begin receiving the first incoming frame, which is $M\,[i]$ say, and thereafter to decide whether or not to keep $M\,[i]$ or simply throw it away. Once Bob finishes with the reception of this frame, he computes a new value, $b$, for his local variable, $B$, i.e., $B \leftarrow b$. Note that the value $b$ must be greater than or equal to the current value of $B$. Furthermore, Bob must have received and decided to keep all of the frames:

      $M\,[0], M\,[1], M\,[2], \ldots, M\,[b–1]$

      However, $b$ does need not be related to the sequence number of the incoming frame, $i$, since he may have chosen this time to reveal the "secret" that he previously received and kept some extra frames.

### *3.2 Behavior of the Channel between Alice and Bob*

As far as Alice and Bob are concerned, the channel between them looks like a pair of FIFO queues. For the *frame queue,* Bob deletes (i.e., reads) incoming frames from the front, starting with the oldest first, while Alice inserts (i.e., writes) outgoing frames to the end, with the newest last. Similarly, for the *acknowledgement queue*, Alice deletes (i.e., reads) incoming acknowledgements from the front, while Bob inserts (i.e., writes) outgoing acknowledgments to the end. At any time, there might be multiple frames and/or acknowledgments in transit between Alice and Bob, either because the propagation delay over the link is very large (e.g., the data is being relayed through a satellite), or because data-link control *programs* at Alice and Bob are simply so slow that data from the link is piling up in an input buffer faster than they can read it. We really don't care what is responsible for the delay as long as the channel behaves like "a piece of wire" i.e., data can be lost or damaged in transit, but never reordered.

Let us now define some notation to make it easier to describe what we would see in the frame queue and acknowledgement queue if we took a snapshot of the system. First, we will use the variable $i$ as an index to the acknowledgement queue, so the contents of the acknowledgment queue (from front to back) would be described as:

$$b\,[1],\ b\,[2],\ \ldots,\ b\,[i],\ \ldots,\ b\,[n]$$

where $n$ represents the total number of acknowledgements that have been sent by Bob and not yet received by Alice.

Similarly, we would like to use the variable $j$ as an index into the frame queue, but the situation here is more complicated because we must be careful not to confuse the *sequence number* for a frame with its *current position* in the frame queue. In particular, the sequence number for the second frame in the queue is not necessarily one greater than the sequence number for the first frame in the queue because Alice has the freedom to pick any of the first $k$ unacknowledged frames for her next transmission using procedure **a1**. Thus, we will define $s\,[j]$ to be the sequence number of the frame currently at position $j$ in the frame queue, so the contents of the frame queue (from front to back) may be described as:

$$M\,[s\,[1]],\ M\,[s\,[2]],\ \ldots,\ M\,[s\,[j]]\ ,\ \ldots,\ M\,[s\,[r]]$$

where $r$ represents the total number of frames that have been sent by Alice and not yet received by Bob.

## 3. Invariant Properties of the Combined System

What is especially interesting about the material in this section is that we will derive some non-trivial global properties, which are always true for the combined system (consisting of Alice, Bob and their intermediate channel) even though neither Alice nor Bob by themselves has access to enough information about the global system to check whether those properties are really true. Moreover, once we have established those global properties, it will become very easy to find the minimum space requirement for tagging frames when Alice and Bob can send and/or receive frames out of order.

### *3.1 The contents of the acknowledgment queue is always non-decreasing.*

More generally, the following condition is always true:

$$\boxed{A \le b\,[1] \le b\,[2] \le \ldots \le b\,[n] \le B \qquad (3.1)}$$

The proof of Invariant 3.1 is by induction on the total number of times that the above condition has been changed by either Alice or Bob. The base case is established at the beginning of time, when Alice and Bob initialize $A = B = 0$, and the acknowledgement queue starts out empty (i.e., $n = 0$). Thereafter, there are exactly three ways in which the above condition can be changed:

1. Alice executes procedure **a2**, which removes $b\,[1]$ from the front of the acknowledgment queue and sets $A \leftarrow b\,[1]$. As a result, we must subtract one from the queue length, $n$, and from the queue position indices for each of the remaining acknowledgements.
2. Bob executes procedure **b1**, which inserts an $(n+1)^{st}$ entry at the end of the acknowledgement queue that carries the same value as $B$.
3. Bob executes procedure **b2**, which causes him to set $B \leftarrow b$, where the new value satisfies $B \le b$, after he has removed the first frame, $M\,[s\,[1]]$, from the front of the frame queue.

Clearly each of these changes preserves the truth of the Invariant, assuming it was true previously.

QED.

***3.2 The contents of the frame queue is "almost" non-decreasing in the sense that later sequence numbers can never back up to* k *or more below the largest previous value.***

More generally, let *j_max* represent the largest frame sequence number ever to reach Bob up to the current time.[2] Then if we augment the list of queued frame sequence numbers by defining $s\,[0] \equiv j\_max$, and $s\,[r+1] \equiv A$, the following condition is always true:

$$\boxed{s\,[j] < s\,[j'] + k, \text{ for all } 0 \le j < j' \le r+1 \qquad (3.2)}$$

In other words, if the frame with sequence number $s\,[j']$ is currently at position $j'$ in the frame queue, then Alice has never transmitted another frame ahead of it with a sequence number $j'+k$ or larger. In particular, if such an earlier frame (with a sequence number that is "too large") existed, then there are only two possibilities for its current location. Either the "too large" frame has already reached Bob, in which case its value would be included in *j_max*, or it is still in the frame queue at some earlier position $j < j'$. However, since we have defined *j_max* to be the sequence number for position zero in the augmented frame queue, it should be clear that both of these possibilities are part of the above condition.

Once again, the proof of Invariant 3.2 is by induction on the total number of times that the augmented frame queue has been changed by either Alice or Bob. The base case is established at the beginning of time, when Alice and Bob initialize $A = j\_max = 0$, and the frame queue starts out empty (i.e., $r = 0$). Thereafter, there are exactly two ways in which the above condition can be changed:

[2] Note that we need to refer to the value *j_max* as part of the proof. However since Bob has no particular use for this information, he may not choose to keep track of it and hence we did not define it as another local variable for Bob in section 2.2.

1. Alice executes procedure **a1**, which inserts an $(r+1)^{st}$ entry at the end of the frame queue. Note that Alice may choose any value between $A$ and $A+k-1$ for the sequence number for the newly-inserted frame, which is both large enough to preserve the inequality when the new frame acts as $j'$ relative to some earlier frame, and also small enough when it acts as $j$ relative to the last value $A$.
2. Bob executes procedure **b2**, which causes him to remove frame $M\ [s\ [1]]$ from the front of the frame queue. In addition, we must subtract one from the queue length, $r$, and from the queue position indices for each of the remaining entries. Note that removing the first entry from the frame queue will clearly trigger a reevaluation of *j_max*, which makes this change equivalent to replacing the first *two* entries from the head of the augmented frame queue with a single entry, equal to the maximum of the two. Thus since the upper limit condition on $s\ [j]$ was individually true for both values beforehand, it is also true for their maximum.

QED.

### *3.3 If frame* **M [s [j]]** *is currently in the frame queue, then its sequence number,* **s [j],** *must be "close" to* **B.**

More specifically, the following condition is always true:

$$\boxed{B - k \le s\ [j] < B + k \qquad\qquad (3.3)}$$

The proof of Invariant 3.3 can be done in the following way. First, by evaluating Invariant 3.2 with $j' = r+1$, we see immediately that

$$s\ [j] < A + k$$

Thus, the right-hand inequality from Invariant 3.3 is established by using the result $A \le B$, which is obtained from Invariant 3.1.

Next, we recognize that since $j\_max \equiv s\ [0]$ is part of the definition of the augmented frame queue, we see immediately from Invariant 3.2 that

$$j\_max < s\ [j] + k$$

We also recognize from the definition of $B$ that all frames up to and including at least $M\ [B-1]$ must have reached Bob, and hence that

$$B - 1 \le j\_max$$

By combining these last two results, we see that $B - 1 \le s\ [j] + k$, or $B - k < s\ [j]$, establishes the left-hand inequality from Invariant 3.3.

QED.

### *3.4 If acknowledgement* **b [i]** *is currently in the acknowledgement queue, then its value must be "close" to* **A.**

More specifically, the following condition is always true:

$$\boxed{A \le b\ [i] \le A + k \qquad\qquad (3.4)}$$

The proof of Invariant 3.4 is very easy. The left-hand inequality from Invariant 3.4 is just a special case of Invariant 3.1. Thus, to prove the right-hand inequality we just need to continue the derivation in section 3.3 for a few more steps. In particular, by combining the above result that $B - 1 \leq j\_max$ with the fact that $j\_max < A + k$ from Invariant 3.2, we see immediately that $B - 1 < A + k$, or $B \leq A + k$. The final step is to use Invariant 3.1 to show that $b\,[i] \leq B$.

QED.

## 4. Main Result: Sequence Numbers Modulo *m* May be Used as Tags

Now consider what happens to the correctness of the system if Alice and Bob don't include *actual* sequence numbers (i.e., non-negative integers of potentially unlimited size) in Alice's frames or Bob's acknowledgements. Instead, they include only its *remainder*, after dividing the actual sequence number by some constant, $m$. Under what conditions on $m$ (as a function of Alice's out-of-order sending limit, $k$) can we be sure that Alice and Bob can never misinterpret the other party's reference to frame $j$, even though the corresponding message only contains a tag value of $t \equiv j$ modulo $m$, rather than its actual sequence number, $j$?

### *4.1 Alice executes* **a2***, and must understand the acknowledgement sent by Bob.*

First, suppose the value of the acknowledgement that Alice reads from the front of the queue carries only the tag $x \equiv b\,[1]$ modulo $m$, rather than the complete sequence number, $b\,[1]$. How can she reliably pick out Bob's intended value, $b\,[1]$, out of the (infinite) set of the possible interpretations for a tag value of $x$, i.e.,

$$x, (x + m), (x + 2m), \ldots, (b\,[1] - m), b\,[1], (b\,[1] + m), \ldots$$

The key here is that, because of Invariant 3.4, Alice knows *even before she reads it*, that the sequence number $b\,[1]$ must fall within a very narrow range:

$$A \leq b\,[1] \leq A + k$$

Thus, if only one of the possible interpretations for the tag value $x$ falls within this range, then Bob's intended value is obvious! But since the possible range for $b\,[1]$ is an interval of $k + 1$ consecutive values, whereas the different interpretations are separated by at least $m$, Alice will be left with only one possible interpretation to choose from we require

$$\boxed{m \geq k + 1 \qquad\qquad (4.1)}$$

### *4.2 Bob executes* **b2***, and must identify the sequence number of the frame sent by Alice.*

Now, suppose the value of the frame that Bob reads from the front of the queue carries only the tag $x \equiv s\,[1]$ modulo $m$, rather than the complete sequence number, $s\,[1]$. This time, we can use Invariant 3.3 to show that *even before he reads it*, Bob knows that the sequence number for the first frame, $s\,[1]$, must fall within a (slightly less-narrow) range:

$$B - k \leq s\,[1] < B + k$$

Thus, using a similar argument to the one in section 4.1, Alice's intended sequence number will be obvious to Bob as long as we are sure that only one of the possible interpretations will fit within this range. This time, however, Bob's range contains *2k* consecutive values, so Bob will be left with only one possible interpretation to choose from if we require

$$\boxed{m \geq 2k \qquad\qquad (4.2)}$$

In the case where Bob is attempting to receive frames from Alice, there is another complication we need to consider. Remember that procedure b2 gives Bob the flexibility to *receive frames out of order*, and/or to *withhold information about received frames from Alice,* even though a lazy/inefficient (but still correct) implementation of Bob's data-link controller might decide to accept frames only if they arrive in numerical order, or more generally only if they are among the first $l$ as-yet unreceived frames. As a result, as long as Bob can determine that the sequence number for the incoming frame is *not* in the "useful" (i.e., worth keeping) range

$$B \leq s\,[1] < B + l - 1$$

he does not need to know whether the incoming frame is "useless" because it is too old (i.e., $s\,[1] < B$, so Bob has already received it) or too new (i.e., $s\,[1] > B + l - 1$, so it falls beyond the end of Bob's resequencing buffer of size $l$). Let's see how this relaxation of the problem changes the requirements on $m$.

Clearly, Bob needs to examine each incoming frame, whose actual sequence number could be any value in the range

$$B - k,\ \ B + k + 1,\ \ldots,\ \ B + k - 1,$$

and correctly identify the sequence numbers for every "useful" frame, i.e., those whose sequence numbers in the range

$$B,\ B + 1,\ \ldots,\ B + l - 1$$

which must be kept. The remaining sequence numbers are classified as "useless" and the associated frames are simply discarded. The key to solving this problem is to recognize that if Alice sends the sequence numbers modulo $m$, then

$$B - m,\ B + 1 - m,\ \ldots,\ B + l - 1 - m$$

represents the most-recent previous range of sequence numbers that shares its tags with the "useful" range, and

$$B + m,\ B + 1 + m,\ \ldots,\ B + l - 1 + m$$

represents the earliest future range of sequence numbers that shares its tags with the "useful" range. Thus, if a failure occurs then one of two situations must have happened:

1. The most-recent previous range of sequence numbers that shares its tags with the "useful" range is too close, and overlaps some valid sequence numbers for "too old" frames. But this cannot happen if the largest item from the previous tag range is smaller than the smallest valid sequence number, i.e.,

$$B + l - 1 - m < B - k,$$

or

$$k + l \leq m.$$

2. The earliest future range of sequence number that shares its tags with the "useful" range is too close, and overlaps some valid sequence numbers for "too new" frames. But this cannot happen if the smallest item from the next tag range is greater than the largest valid sequence number, i.e.,

$$B + m > B + k - 1$$

or

$$m \geq k.$$

Since the first situation generates a requirement on k that is stronger than both the second situation and the earlier requirement from section (4.1), our final result is that the sequence number modulus must satisfy

$$\boxed{m \geq k + l \qquad (4.3)}$$

where

- $k$ represents the limit of Alice's ability to send frames out of order, $k \geq 1$, and
- $l$ represents the limit of Bob's ability to receive frames out of order, $1 \leq l \leq k$.