Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
An Institute of National Importance
www.iiitdm.ac.in
COM302T Computer Networks-Lecture Notes

Instructor
N.Sadagopan
Scribe:
P.Renjith

Flow Control

Objective: In this lecture, we shall present flow control mechanisms using which one can synchronize a fast sender and a slow receiver. We shall be discussing stop and wait and sliding window protocols, and their performance analysis.

Introduction: In earlier lectures, we have discussed network topologies using which one can interface two or more systems. Subsequently, we looked at encoding schemes using which we have learnt how to convert bits into signals. Further, a signal, on travel may get corrupted due to transmission medium and to handle such scenarios, error detection and correction schemes were discussed in detail. Having seen rudimentary concepts required for data transmission, it is now natural to ask: how do synchronize the sender and receiver. What if the network speed of the sender is 1000Mbps and the receiver is 500Mbps ? Should we employ a retransmission policy if in case the receiver detects an error in the file sent by the sender or should we go for multiple bit error detection and correction scheme ? If retransmission is the order of the day, then what would be the expected number of retransmissions ? We shall investigate these questions in detail and learning outcomes from this study will help us to identify a suitable flow control design.

Stop and Wait Protocol

Consider a scenario where the sender wishes to transmit a file of size 1MB. As per this protocol, the given file is split into frames (a small group of bytes) of size 64 bytes. For each frame, CRC computation is done at the server before transmission. Similarly, on receiving the frame, the receiver does CRC check to ascertain whether there is an error in transmission. This process would be done for a total of 2^{14} frames. During this process, some of the frames may require retransmissions. If the frame received is in error which is detected by CRC, then the receiver sends NACK to the sender. If the frame received is without error, then the receiver sends ACK to the sender. We shall explain the behaviour of stop and wait protocol in detail, and highlight the issues involved while the frame is in transmission.

Strategy 1 The protocol behaviour is given as follows; (i) transmit a frame and wait for an acknowledgement from the receiver. (ii) if the acknowlegement is positive, then transmit the next frame. (iii) Otherwise, retransmit the previous frame. At any point of time, exactly one frame is in transmission. The sender and receiver can experience the following scenarios while the frame or acknowledgement in transmission. Figure 1, illustrates all seven possible scenarios.

Sender

1. Frame sent is dropped along the way.
2. For each frame, after sending the frame, the sender maintains a timer within which it expects a positive acknowledgement (ACK) and a negative acknowlegement (NACK). The timer may expire (time out) and the sender may not see ACK or NACK within the time out. i.e., ACK/NACK sent by the receiver drops along the way.

3. ACK/NACK reaches the sender late, i.e., delayed ACK or delayed NACK. This is also considered as time out.

Receiver
1. Receiver is expecting a frame, but it does not see anything in the receiver buffer. Frame sent by the sender is dropped along the way. Receiver sends a negative acknowledgement and asks for a retransmission of the frame.
2. Receiver transmits ACK if there is no error in the frame received or transmits NACK if there is an error in the frame received.

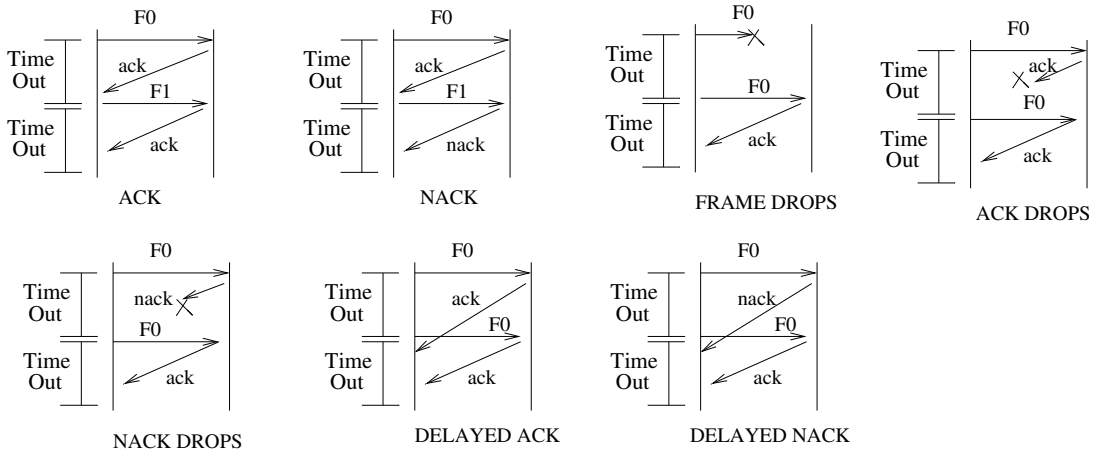

Fig. 1. Stop and Wait - Possible Scenarios

Questions:

1. Will the above scheme work fine always ? Will the sender and receiver correctly interpret the frames received ?

2. If ACK/NACK/Frame drops, will the sender and receiver synchronize correctly for the next frame ?

Remarks:

1. In Figure 2 (fig a and fig b), we see that the frame $F1$ is never received by the receiver. The sender is unaware of this scenario and continue to send subsequent frames. This synchronization issue happens as there is no label attached with ACK/NACK. In Fig a of Figure 2, the delayed ACK of the first $F0$ is considered by the sender as the ACK of the second $F0$ and due to which the sender sends the next frame $F1$. This issue can be resolved if ACK signal from the receiver contains the label of the frame that the receiver is expecting next from the sender.

2. Since the sender sends exactly one frame at a time, to send a sequence of frames: $F0, F1, F2, F3, ...$, we can use a 1-bit label, thus the transmitting sequence would be $F0, F1, F0, F1, F0, F1, ...$ The third frame $F0$ corresponds to $F2$ and the fourth frame $F1$ corresponds to $F3$ and so on.

3. With the above modification, the modified snapshots of Fig a and Fig b is shown in Fig c and Fig d, respectively. An example sequence is given in Fig e.

4. It is important to note that both the sender and receiver maintain a window of size one in the stop and wait protocol. Sender clears the window and brings in the next frame when it receives an acknowledgement from the receiver for the frame sent, and similarly, the receiver

clears the window when it receives the expected frame without error. Thus, at any time of the protocol functioning, the number of outstanding frames is one.

5. Since ACKs are labeled and exactly one frame is in transmission, there is no need to label NACK. On seeing NACK, the sender knows that the previous frame sent is either dropped or in error, and performs a retransmission of the frame. Moreover, the above scheme works fine even if there is a label associated with NACK. That is, NACK 1 to the sender indicates that the frame $F1$ sent before is in error and asks for a retransmission.

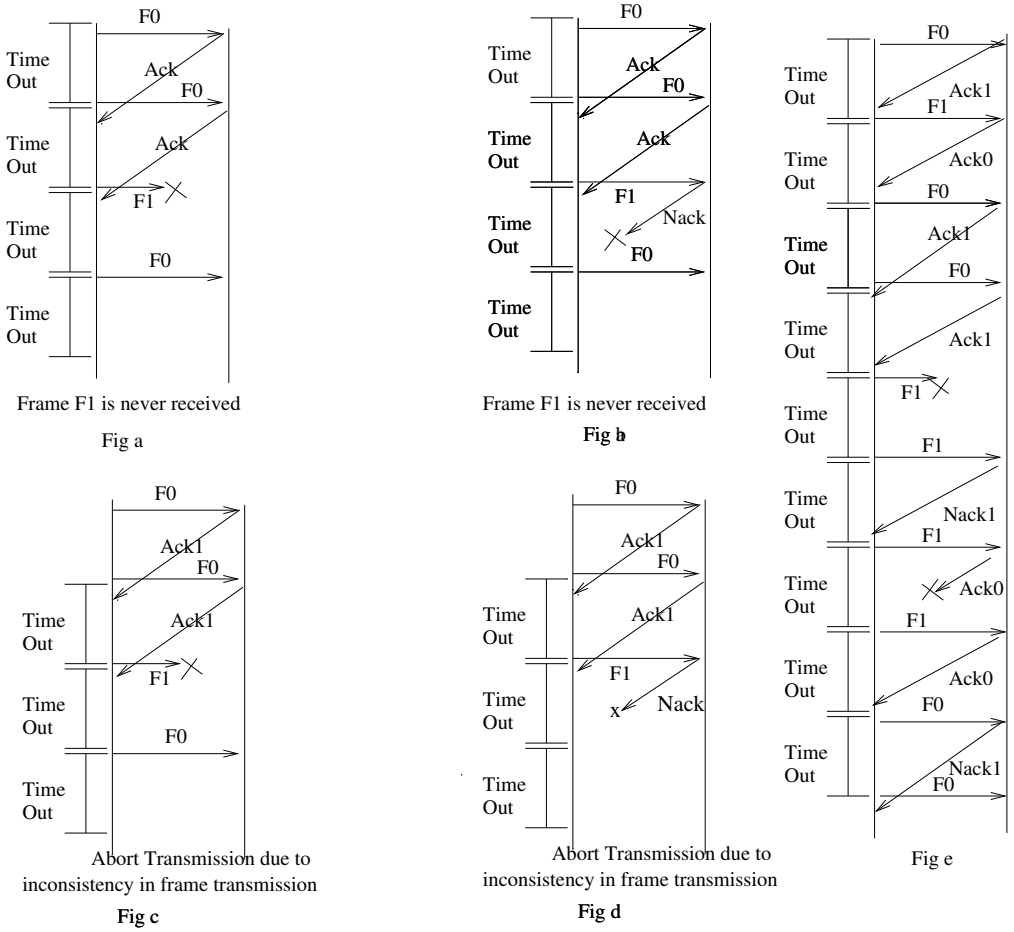

Fig. 2. Stop and Wait - ACK with labels

Performance of Stop and Wait Protocol

We shall analyze the performance of stop and wait protocol and discuss possible directions for improving the link utilization. Assuming there is no frame error/delayed ack/delayed nack, then

Utilization=$\dfrac{t^f_{trans}}{total\ \ time}$

where total time = $t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc}$. Note that t^f_{trans} and t^a_{trans} refer to the transmission time of a frame and an acknowledgement, respectively. t_{prop} denotes the propagation delay and t_{proc} denotes the overhead computation time such as CRC computation at the

sender, CRC check at the receiver, the time to update the sender window, etc.

If the channel is noisy, then the frame may get corrupted during the transmission, and due to which a frame may be retransmitted many times. The number of retransmissions is a random variable which is modeled as a 'geometric random variable'. Similarly, frame drops/delayed ACKs/delayed NACKs triggers time out and hence it is appropriate to look at the expected number of time outs during a transmission which is again a geometric random variable. Let p denote the probability that the frame is in error and q denote the probability of time out.

Let X be the random variable denoting the number of retransmissions of frames due to NACKs, then

$$E(X) = \sum_{i=1}^{\infty} i \cdot p^{i-1} \cdot (1-p).$$

i.e., a sequence of $(i-1)$ failure attempts followed a successful attempt. The first $(i-1)$ retransmissions of a frame results in error and the last transmission is without error.

$$E(X) = (1-p)(1+2p+3p^2+\ldots) = \tfrac{1}{1-p}.$$

On the similar line, let Y be the random variable denoting the number of time outs, then

$$E(Y) = \sum_{i=1}^{\infty} i \cdot q^{i-1} \cdot (1-q) = \tfrac{1}{1-q}.$$

Thus, the utilization incorporating the above expected number is given as follows;

$$\text{Utilization} = \frac{t^f_{trans}}{total \;\; time + (E(X)) \cdot (total \;\; time) + (E(Y)) \cdot (total \;\; time)}.$$

$$\text{Utilization} = \frac{t^f_{trans}}{(total \;\; time) \cdot (1 + \frac{1}{1-p} + \frac{1}{1-q})}$$

Observations:
1. From the above expressions, it is clear that to improve the utilization of stop and wait protocol, either the sender must transmit more frames instead of just one or must decrease t_{prop}. We shall now consider the former strategy and analyze its utilization.

2. Suppose, the sender window is increased to three, what should be the value of receiver window size so that there is no synchronization issue. In such a scenario, how many outstanding frames will be there in the transmission line and the receiver window.

3. When the receiver receives three frames, out of which, say the first frame and the third frame are in error and the second frame is without error, how do we modify ACK/NACK to implement this scenario.

4. For stop and wait, we have used 1-bit label, also known as sequence numbers. Can we still work with sequence numbers $\{0, 1\}$ to transmit three frames at a time. If this is not feasible, what is the minimum number of sequence bits required to avoid synchronization issues/overlapping of sender and receiver windows. By overlapping of window of frames, we mean, the labels used by the sender and the receiver has some overlap, due to which misinterpretation of frames may happen at the receiver.

Sliding Window Protocol - Generalization of Stop and Wait

We shall generalize stop and wait protocol by considering a flow control design with $SWS = 4$ and $RWS = 1$. Note that for stop and wait $SWS = 1$ and $RWS = 1$, and the number of sequence bits is two (0 and 1). Let us consider the following four schemes.

1. Scheme 1: To transmit a sequence of 16 frames, we use two labels, $\{0, 1\}$ alternately. i.e., $F_0, F_1, F_0, F_1, F_0, F_1, \ldots$. Figure 3 shows that this scheme fails due to synchronization issue. For the example given, the second frame F_1 and the third frame F_0 are dropped along the way. The receiver accepts the fourth frame F_1 thinking that it is actually the first F_1.

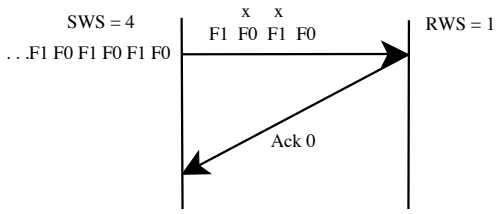

Fig. 3. Sliding Window protocol - Scheme 1

2. Scheme 2: To transmit a sequence of 16 frames, we use three labels, $\{0, 1, 2\}$ cyclically. i.e., $F_0, F_1, F_2, F_0, F_1, F_2, \ldots$. Figure 4 shows that this scheme also fails due to synchronization issue. The first three frames are dropped during the transmission and the receiver is unaware of this scenario. The receiver misinterprets the fourth frame as F_0; accepts and generates the acknowledgement signal as well.

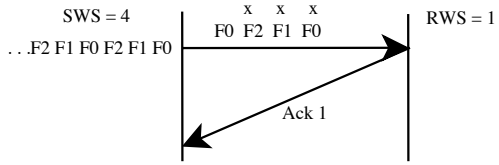

Fig. 4. Sliding Window protocol - Scheme 2

3. Scheme 3: To transmit a sequence of 16 frames, we use four labels, $\{0, 1, 2, 3\}$ cyclically. i.e., $F_0, F_1, F_2, F_3, F_0, F_1, F_2, F_3 \ldots$. Figure 5 shows that this scheme also fails due to synchronization issue.

4. Scheme 4: To transmit a sequence of 16 frames, we use five labels, $\{0, 1, 2, 3, 4\}$ cyclically. i.e., $F_0, F_1, F_2, F_3, F_4, F_0, F_1, F_2, F_3, F_4, \ldots$. Figure 6 shows that this scheme works fine and no synchronization issue.

In Figure 7, an illustration is given for the design SWS=4,RWS=1 with sequence numbers $\{0, 1, 2, 3, 4\}$, and SWS=3,RWS=1 with sequence numbers $\{0, 1, 2, 3\}$. It is important to highlight that there is no overlapping of frames between the sender window and the receiver window

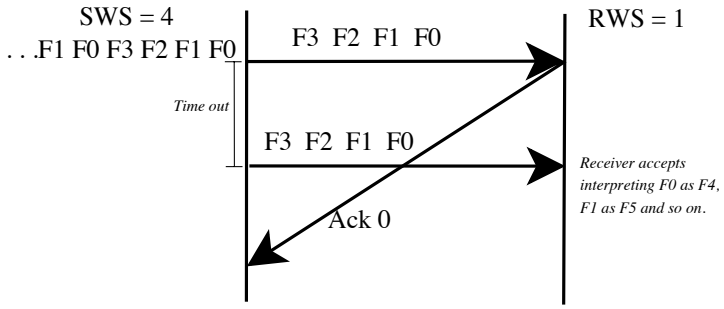

Fig. 5. Sliding Window protocol - Scheme 3

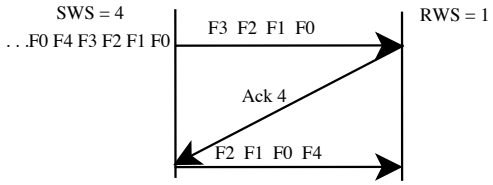

Fig. 6. Sliding Window protocol - Scheme 4

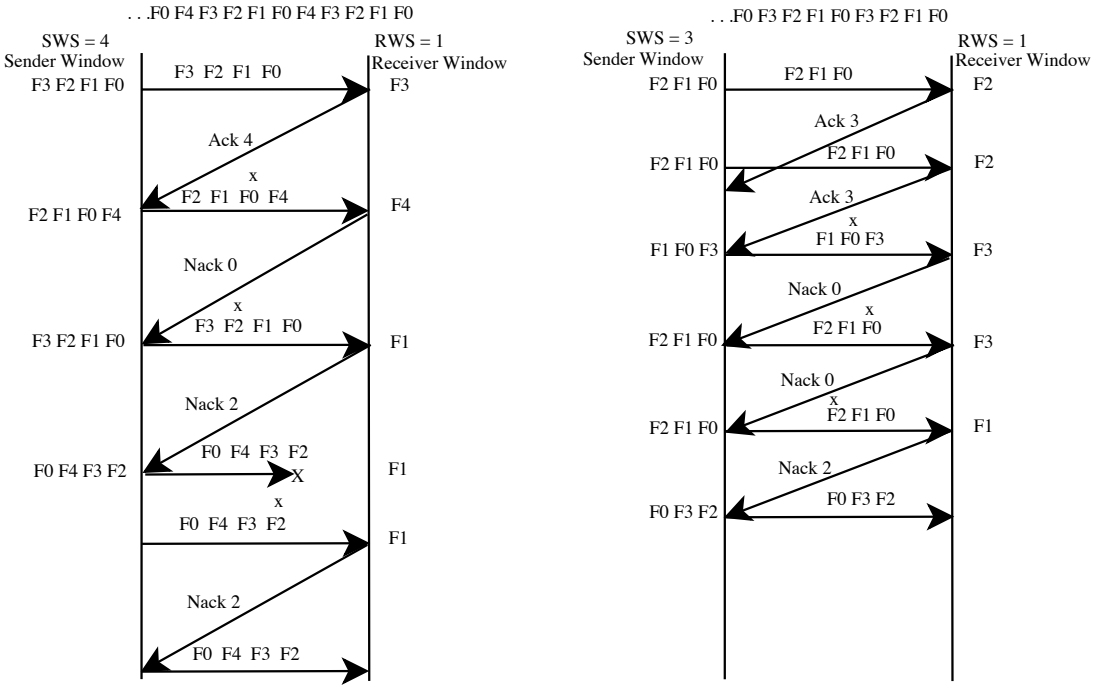

Fig. 7. Sliding Window protocol: SWS=4,RWS=1 and SWS=3,RWS=1

in both cases. In the first case, SWS+RWS=5 which is same as the number of distinct sequence numbers. In the other case, SWS+RWS=4 which is same as the number of distinct sequence numbers.

Observations:
1. Scheme 4 works fine because there is no overlap between the sender window and receiver window. Since there are five distinct sequence labels and at any point of time, the sender contains four distinct frames and the receiver window contains the largest indexed frame for which the acknowledgement is sent.

2. Also, the design in which $SWS = 3$, $RWS = 1$, and sequence labels $\{0, 1, 2, 3\}$ works without any synchronization issues. This is true as there is no overlap between the sender window and the receiver window.

3. In general, if $SWS + RWS \leq n$, where n represents the number of distinct sequence numbers, then there is no overlap between the sender window and the receiver, and hence the protocol design works fine.

How Sliding Window works:

Consider a scheme with $SWS = 4$ and $RWS = 1$, and the frames to be sent are $F_0, F_1, F_2, F_3, F_4, F_0, F_1, F_2, F_3, F_4, \ldots$. At each iteration, the sender places a window of four frames along the transmission link. As soon as the four frames reaches the receiver, say F_0, F_1, F_2, F_3, the receiver performs CRC check on each of them.

(i) Suppose, F_0 is delivered without error and F_1 is in error, then the receiver sends NACK 1 indicating to the sender that F_0 is perfect (positive acknowledgement for F_0) and asks for a retransmission of F_1. The sender retransmits all frames starting from F_1.

(ii) Since the receiver window is one, it can hold exactly one outstanding frame which is F_0 in this case. Since F_1 is in error, the receiver does not accept subsequent frames, namely F_2, F_3.

(iii) Whenever a frame F_i is in error, the receiver stops accepting frames starting from F_{i+1} and sends NACK i, the sender retransmits $F_i, F_{i+1}, \ldots$. The receiver stores the largest indexed frame without error at the window, i.e., F_{i-1}. NACK i is a positive acknowledgement for $F_0, F_1, \ldots, F_{i-1}$ and a negative acknowledgement for F_i.

(iv) The scenario for delayed ack/delayed nack/ack drop/nack drop is similar to stop and wait. Frame drop is similar to frame in error. Suppose in a sequence $F_0, F_1, F_2, \ldots$; F_0, F_1 reaches the receiver with no error, whereas F_2 is dropped along the way, then the receiver stores F_1 at the window and sends NACK 2.

(v) Although, this scheme improves the link utilization by sending more frames in each iteration, the overhead due to retransmission is high. In some cases, a frame without error may be retransmitted unnecessarily. For example, F_{i+1} may be without error and since F_i is in error, F_{i+1} is retransmitted unnecessarily until F_i is received with no error.

(vi) It is natural to ask, can we increase the receiver window size so that the number of outstanding frames can be increased. For example, if the receiver window size is two, then for the previous example, F_{i+1} can be stored at the receiver and when F_i is received with no error, it can be inserted at the appropriate location in the window.

(vii) Sliding window with $RWS = 1$ is called Go-back-n strategy in the literature, where n denotes the number of sequence numbers. The working principle of sliding window with $RWS = 2$ and $RWS \geq 3$ is little different from Go-back-n strategy due to the presence of outstanding frames. Sliding window with $RWS \geq 2$ is known as selective-reject sliding window in the literature.

Working principle of selective-reject:

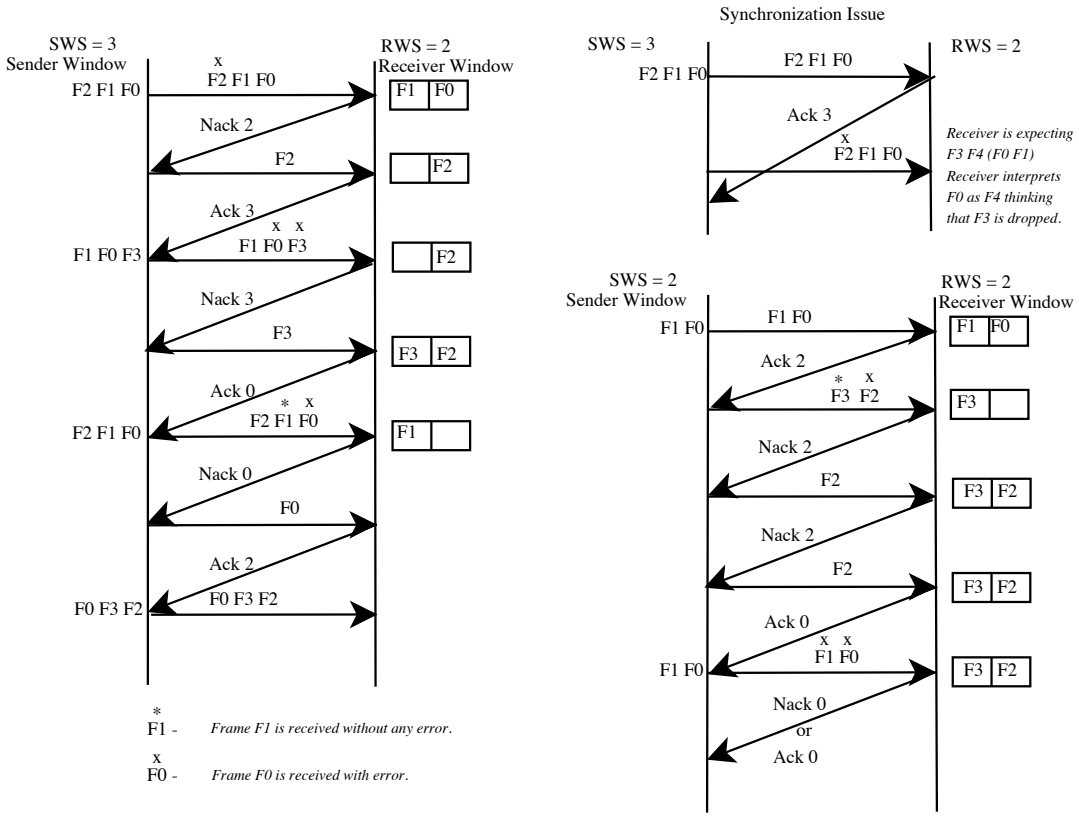

Fig. 8. Sliding Window Selective Reject protocol: SWS=3,RWS=2 and SWS=2,RWS=2

(a) Suppose the sender wishes to transmit $F_0, F_1, F_2, F_3, F_0, F_1, F_2, F_3 \ldots$ with SWS=3 and RWS=2. Although, the sender sends three frames in each iteration, the receiver will only look at the first two frames as the outstanding window capacity is two. Suppose, during the first iteration, out of F_0, F_1, F_2; F_0 is in error and F_1 is without error, then the receiver stores F_1 in the window and reserve the other cell in the window for F_0. Further, the receiver sends NACK 0 to the sender indicating that F_0 is in error. On seeing NACK 0, the sender transmits only F_0. Unlike, go-back-n strategy, in this scheme, the frame in error alone is retransmitted until it reaches the receiver without error. When the receiver receives F_0 without error, it generates a

cumulative acknowledgement ACK 2 indicating both F_0 and F_1 (which is already there in the window) are without errors.

(b) It is important to note that the sender is unaware of whether F_1 is in error or not until it sees ACK 2. On seeing ACK 2, the sender sends the next set of frames; F_2, F_3, F_0. If suppose, both F_2 and F_3 are in error, then the receiver sends NACK 2. F_0 will be discarded by the receiver irrespective of whether it is a true frame or a corrupted frame. Two slots in the receiver window are reserved for F_2 and F_3, and therefore, any other outstanding frames beyond F_3 will be discarded by the receiver.

(c) An illustration of sliding window selective-reject protocol is given in Figure 8. It is now appropriate to analyze the number of sequence numbers used and whether there are any synchronization issues. Note that when SWS=3 and RWS=2 the number of outstanding frames at the sender is 3 and at the receiver is 2, and hence, their sum is more than the sequence number labels which is $\{0, 1, 2, 3\}$. As per our previous observation, this creates a synchronization issue which is illustrated in Figure 8. In the example given, the ACK sent by the receiver at the end of the first iteration is a delayed ACK, however, the sender retransmits the first set of $F0, F1, F2$ after time out. Assuming $F2$ is dropped along the way, the receiver accepts $F0$ thinking that $F3$ is dropped along the way and $F0$ is actually $F4$. This can be avoided if we reduce the sender window size to two. An illustration for SWS=2 and RWS=2 is also shown in the figure. Here again, the scheme works fine as SWS+RWS=4, which is the number of sequence bits.

How do we determine SWS and sequence numbers ?

(1) SWS is dictated by the delay bandwidth product as DB product gives the estimate of the maximum number of bits along the transmission channel. For example, a 100Mbps link with t_{prop} = 10 micro sec can hold $100 \times 10^6 \times 10 \times 10^{-6} = 1000$ bits. If frame size is 100 bits, then the number of frames is 10. To improve the utilization, the sender has to keep the transmission channel busy during t_{prop} for which the sender is expected to send 10 frames in each iteration.

(2) For a Go-back-n strategy, one can work with $SWS = 10$ and $RWS = 1$. Thus, there will be 11 distinct frames, and to avoid overlapping of windows, we need 11 sequence numbers.

(3) For a selective reject strategy with $SWS = 10$ and $RWS = 2$, we need 12 sequence numbers. To reduce retransmission overhead and to exploit the fact that $SWS = 10$, the receiver can set the window size to 10. Thus, if $SWS = RWS = 10$, then the receiver can see 10 outstanding frames, and to avoid overlapping of windows, we need 20 sequence numbers.

(4) We often work with RTT while computing DB product in place of t_{prop} (which is RTT/2). Since the sender typically waits for at least one RTT before it updates the window, it is appropriate to keep the sender busy until one RTT. In such a case, for the above scenario, the DB product is $100 \times 10^6 \times 10 \times 10^{-6} \times 2 = 2000$ bits. If frame size is 100 bits, then the number of frames is 20. Thus, SWS=20. For Go-back-n strategy with SWS=20, RWS=1, the minimum number of sequence numbers required to avoid synchronization issue is 21. For selective-reject with SWS=20 and RWS=20, we need at least 40 sequence numbers to avoid synchronization/overlapping issues.

(5) It is important to note that, having $RWS > SWS$ will not have any impact on the pro-

tocol design, and hence, the behaviour of protocol for such schemes is same as $RWS = SWS$. Thus, protocol with SWS=2 and RWS=7 is as good as working with SWS=2 and RWS=2. In general, any protocol with $RWS \geq SWS$ and $SWS + RWS \leq n$ works fine without any synchronization/overlapping of window issues, where n denotes the number of distinct sequence numbers.

Utilization of Sliding Window Protocol

We shall analyze first the performance of Go-back-n protocol. Since Go-back-n strategy is similar to stop and wait, the analysis of this protocol is similar to stop and wait. The only difference lies in computing the retransmission overhead. For noise-free channel, the utilization is given as follows; assuming $SWS = n - 1$ and $RWS = 1$;

Utilization=$\frac{(n-1)t^f_{trans}}{total \;\; time}$

where total time = $(n-1)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc}$. Note that t^f_{trans} and t^a_{trans} refer to the transmission time of a frame and an acknowledgement, respectively. t_{prop} denotes the propagation delay and t_{proc} denotes the overhead computation time.

If the transmission channel is a noisy channel, then we need to compute the expected number of retransmissions to successfully transmit the entire sender window. Suppose, the window contains $F_0, F_1, \ldots, F_{n-1}$, we say the transmission is complete if all of $F_0, F_1, \ldots, F_{n-1}$ are received at the receiver. Note that each time, the sender sends a window of $(n-1)$ frames. For example, during

Iteration 1, it is $F_0, F_1, \ldots, F_{n-1}$

Iteration 2, it is $F_0, F_1, \ldots, F_{n-1}$ again due to NACK 0

Iteration 3, it is $F_1, \ldots, F_{n-1}, F_n$ due to NACK 1

Iteration 4, it is $F_{n-2}, F_{n-1}, F_n, \ldots$ due to NACK $(n-2)$

Iteration 5, it is $F_{n+1}, F_{n+2}, \ldots$ due to NACK $(n+1)$. At this time the initial $F_0, \ldots, F_{n-1}$ has reached the receiver safe.

Although, the frame sequence sent by the sender in each iteration is different depending on ACK/NACK signal, the number of frames transmitted is always $(n-1)$. Suppose Iteration k successfully transmits all of the first $(n-1)$ frames, then all other iterations starting from 2 to k is considered as retransmissions. The number of retransmissions is a random variable which is modeled using a geometric random variable. If p is the probability that a frame is in error, then the expectation is $\frac{1}{1-p}$. Similarly if q is the probability of time out, then the utilization is

$$\text{Utilization}= \frac{(n-1)t^f_{trans}}{(total \;\; time) \cdot (1 + \frac{1}{1-p} + \frac{1}{1-q})}$$

We shall now analyze the utilization of selective-reject sliding window protocol by considering SWS=$n-2$ and RWS=2. The objective is to successfully transmit $F_0, F_1, \ldots, F_{n-3}$ to the receiver. Since the receiver can store a frame ahead of the corrupted frame, the analysis depends on whether the frame ahead of the corrupted frame is in error or not.

Worst Case: Each frame in $F_0, F_1, \ldots, F_{n-3}$ is in error. Thus, the receiver first generates NACK 0 and waits until true F_0 is reached. The number of such retransmissions of F_0 is given by $\frac{1}{1-p}$ as it is modeled using a geometric random variable. Similarly, for each of $F_2, \ldots, F_{n-3}$, the number of retransmissions is $\frac{1}{1-p}$. Therefore,

Utilization= $\frac{(n-2)t^f_{trans}}{X + Y + Z + W}$

Where $X = (n-2)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc}$
$Y = (n-2) \cdot (\frac{1}{1-p} - 1) \cdot (t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$
$Z = (n-2) \cdot (\frac{1}{1-q}) \cdot ((n-2)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$
$W = (n-3)((n-2)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$
Note: X denotes the overhead due to transmission of the first set of $(n-2)$ frames, Y and Z denotes the overhead due to retransmissions and time out, respectively. Finally, W denotes the overhead due to transmission of $(n-2)$ frames at the end of every unsuccessful attempt. That is, after a sequence of NACK 0, when we get ACK 0 from the receiver, the sender updates the window and sends the next $(n-2)$ frames. Similarly, after a sequence of NACK 1, when we get ACK 2 from the receiver, the sender updates the window and sends the next $(n-2)$ frames. This process happens for $(n-3)$ times for each of F_1 to F_{n-2}. Note that in the expression Y we subtract 1 from the expected number as the last retransmission transmits $(n-2)$ frames which is counted as part of W. The expression Z counts the expected number of time outs that happen due to frame drops whenever the sender sends a window of $(n-2)$ frames. All other time outs due to delayed ACK/NACK, etc. will be counted as part of Y. That is, after a sequence of NACK 0, and when the sender sees a ACK 2, it sends the next set of $(n-2)$ frames, and during this transmission the entire window may be dropped leading to time out scenario. This may happen whenever the sender sends a window of $(n-2)$ frames after a sequence of NACKs. For example,
Iteration 1; the sender sends $F_0, F_1, \ldots, F_{n-3}$
Iteration 2; F_0 is sent as it received NACK 0
... NACK 0 again.
... NACK 0 again.
The above sequence of NACKs is counted as part of Y.
Iteration 15; sends $F_1, \ldots, F_{n-2}$ on seeing ACK 1. The transmission time is counted as part of W. However, during transmission, all of them can be dropped and due to which the sender times out which triggers retransmission of $F_1, \ldots, F_{n-2}$ again. This may happen repeatedly which is counted as part of Z.
Iteration 16; F_1 is sent as it received NACK 1. Again, this expected number is counted by Y.

Best Case: In this case, every second frame is without error. For example, Iteration 1; the sender sends $F_0, F_1, \ldots, F_{n-3}$. F_0 is in error and F_1 is without error.
Iteration 2; F_1 is stored at the receiver. Only F_0 is sent by the sender as it received NACK 0
... NACK 0 again.
... NACK 0 again.
Iteration 10; After a sequence of NACK 0, the receiver sends ACK 2 as a cumulative acknowledgement for F_0 and F_1. The sequence of NACKs is counted as part of Y.
Iteration 11; the sender sends $F_2, F_3, \ldots, F_{n-1}$. Now, F_3 is safe and F_2 is in error. F_3 is stored at the receiver window and the receiver generates NACK 2. During this transmission, the entire window of frames $(F_2, F_3, \ldots, F_{n-1})$ may be dropped and time out happens which is counted by Z. If there is no time out, then the time for $F_2, F_3, \ldots, F_{n-1}$ is counted as part of W.

Since, the error is for every second frame, the above sequence happens for $\frac{n-2}{2}$ times. Therefore,
Utilization= $\dfrac{(n-2)t^f_{trans}}{X + Y + Z + W}$
Where $X = (n-2)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc}$
$Y = \frac{(n-2)}{2} \cdot (\frac{1}{1-p} - 1) \cdot (t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$
$Z = \frac{(n-2)}{2} \cdot (\frac{1}{1-q}) \cdot ((n-2)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

$W = (\frac{(n-2)}{2} - 1)((n-2)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

If receiver window size is k, then the receiver can hold k outstanding frames. Worst case happens when every transmitted frame in $F_0, \ldots, F_{n-k}$ is in error. The analysis of this case is similar to the previous one. Note that, $SWS = n - k$ and $RWS = k$ so that the number of sequence numbers used is n.

Utilization= $\frac{(n-k)t^f_{trans}}{X + Y + Z + W}$

Where $X = (n-k)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc}$

$Y = (n-k) \cdot (\frac{1}{1-p} - 1) \cdot (t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

$Z = (n-k) \cdot (\frac{1}{1-q}) \cdot ((n-k)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

$W = (n-k-1)((n-k)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

For the best case, we need to consider the scenario in which every k^{th} frame is corrupted by the link and $F_0, \ldots, F_{k-1}$ are safe and it is stored in the receiver window.

Utilization= $\frac{(n-k)t^f_{trans}}{X + Y + Z + W}$

Where $X = (n-k)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc}$

$Y = \frac{(n-k)}{k} \cdot (\frac{1}{1-p} - 1) \cdot (t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

$Z = \frac{(n-k)}{k} \cdot (\frac{1}{1-q}) \cdot ((n-k)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

$W = (\frac{(n-k)}{k} - 1)((n-k)t^f_{trans} + t_{prop} + t^a_{trans} + t_{prop} + t_{proc})$

Summary:

In this lecture, we have addressed a fundamental question: how do we synchronize a fast sender and a slow receiver ? In an attempt to answer this question, we first designed a basic protocol, namely, stop and wait protocol using which the sender can send exactly one frame at a time and must wait for an acknowledgement before transmiting the next frame. This scheme worked fine and flow control objective is also met, however, the link utilization is poor as we used a very small portion of DB product during the transmission. Subsequently, to increase the link utilization, we considered sliding window protocol which sends a window of frames during each iteration. While the window of frames in transmission, a part of the window may get corrupted leading to a retransmission of the window or the frame in error. Accordingly, we designed two sliding window protocols; Go-back-n and selective-reject. The utilization of each of the protocol scheme is also analyzed in detail.