

**EAST WEST UNIVERSITY**

**Course Title:** Data Communications

**Course Code:** CSE350

**Section No:** 3

**Project Report**

**Submitted To:**

Lutfun Nahar Lota

Lecturer,

Department of Computer Science and Engineering,

East West University.

**Submitted By:**

Fatema Nihar ID: 2015-3-60-048

Shakera Chowdhury ID: 2015-3-60-051

Shaykh Siddique ID: 2016-1-60-053

Atia As Samia ID: 2015-3-60-049

Md.Shohag ID: 2016-1-60-047

Jannatui Ferdous Sorna ID: 2016-1-60-029

***Abstract-* This paper presents the stop and wait protocol of data link control protocol. The main regions are about a simulator application of data transfer using stop and wait protocol.**

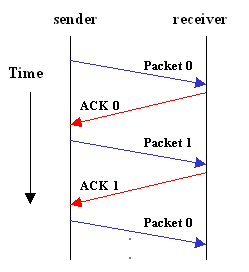
**Introduction:**

In data communications flow controls is the process of managing the rate of data transmission between to nodes to prevent a fast sender from overwhelming a slow receiver. Flow control ensures the sending entity does not overwhelm the receiving entity with data, sending and receiving rate same. It also prevents buffer overflow. In the absence of flow control the receiver buffer may fill up and overflowed. Stop and wait is one kind of flow control. It is simplest form and we can send one frame at a time. Source transmits frame and waits for ACKs before sending next frame. Destinations receives frames and replies with ACKs.

Stop and wait, sometimes known as positive acknowledgement with retransmission is the fundamental technique to provide reliable transfer under unreliable packet delivery system.

**Protocol Description:** This protocol works in two states.

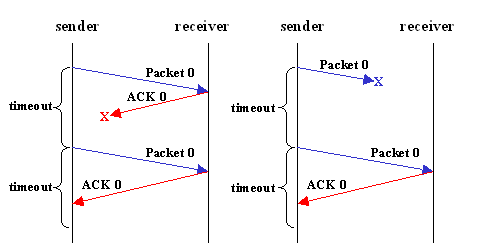
* Normal operation
* Timeout



**Normal operation-** After transmitting one packet, the sender waits for an acknowledgment (ACK) from the receiver before transmitting the next one. In this way, the sender can recognize that the previous packet is transmitted successfully and we could say stop-n-wait guarantees reliable transfer between nodes. To support this feature, the sender keeps a record of each packet it sends.

Also, to avoid confusion caused by delayed or duplicated ACKs, stop-n-wait sends each packets with unique sequence numbers and receives that numbers in each ACKs.

**Timeout-** If the sender does not receive ACK for previous sent packet after a certain period of time, the sender times out and retransmits that packet again. There are two cases when the sender does not receive ACK, one is when the ACK is lost and the other is when the frame itself is not transmitted.



To support this feature, the sender keeps timer per each packet.

**Implementation Details:**

The application is developed in C++ language.

All data stored in object with link list.

**Receiver Side**

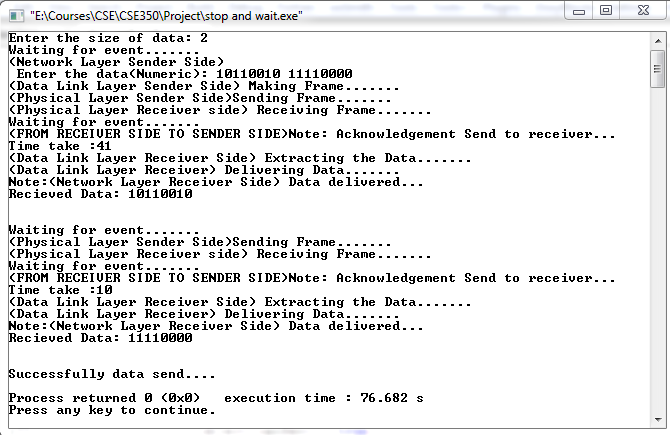
**Sender Side**

* It will start a new event and input all data from keyboard. Then make a frame with data and in physical layer it will send data to receiver side.
* Physical layer of receiver side will receive that frame and will send acknowledgment to the sender side. Receiver side will extract that data and delivered.
* If any acknowledgment failed to sender side then, the sender side will resend that frame.

**Input:**

Size of data: 2

Data 1: 10110010 `Data 2: 11110000



In the first event it received data 10110010 and in second event it received data 1111000.

**Code:**

1. #include<bits/stdc++.h>
2. #include<windows.h>
3. #include<time.h>
4. #define sfi(n) scanf("%d", &n)
5. **#define sf scanf**
6. #define pf printf
7. #define bl printf("\n")
8. #define repe(i, n, test) for(int i=n; i<=test; i++)
9. #define rep(i, n, test) for(int i=n; i<test; i++)
10. **#define EPS 1e-7**
11. #define MAX 1000005
12. #define Index 100
13. #define sleepTime 500
14. using namespace std;


18. struct Data {
19. int dataFrame, dataRecieved, extractedDatavar, data[Index], \*Sn, \*Rn, frame, deliverDat, turn = true;
20. **bool input = true, Event = true, canSent = true, Extrachk = true, Extrachk1 = true;**
21. }object;

24. //Functions of Sender-Side
25. **void waitForEvent() {**
26. pf("Waiting for event");
27. rep(i, 0, 7){
28. Sleep(sleepTime);
29. cout << ".";
30. **}bl;**
31. }

34. void getData(int sizeOfData) {
35. **cout << "(Network Layer Sender Side) \n Enter the data(Numeric): ";**
36. rep(i, 0, sizeOfData){
37. cin >> object.data[i];
38. }
39. object.data;
40. **cout << "(Data Link Layer Sender Side) Making Frame";**
41. rep(i, 0, 7){
42. Sleep(sleepTime);
43. cout << ".";
44. }bl;
45. **}**
46. void makeFrame() {
47. if (object.Extrachk == true) {
48. object.Sn = object.data;
49. object.Extrachk = false;
50. **}**
52. object.frame = \*(object.Sn);

55. **}**
56. int sendFrame() {
57. cout << "(Physical Layer Sender Side)Sending Frame";
58. rep(i, 0, 7){
59. Sleep(sleepTime);
60. **cout << ".";**
61. }bl;
63. object.dataFrame = object.frame;
64. object.turn = false;
66. return object.dataFrame;
67. }
69. void ackFrame() {
70. **cout << "(FROM RECEIVER SIDE TO SENDER SIDE)Note: Acknowledgement Send to receiver..." << endl;**
71. }
73. //Functions of Reciver-Side
74. int recieveFrame() {
75. **if (object.Extrachk1 == true) {**
76. object.Rn = object.data;
77. object.Extrachk1 = false;
78. }
79. cout << "(Physical Layer Receiver side) Receiving Frame";
80. **rep(i, 0, 7){**
81. Sleep(sleepTime);
82. cout << ".";
83. }bl;
85. **if (\*(object.Sn) == \*(object.Rn)) {**
86. object.dataRecieved = object.dataFrame;
87. }
88. object.Sn++;
89. object.Rn++;
91. return object.dataRecieved;
92. }
94. int extractData() {
95. **cout << "(Data Link Layer Receiver Side) Extracting the Data";**
96. rep(i, 0, 7){
97. Sleep(sleepTime);
98. cout << ".";
99. }bl;
100. **object.extractedDatavar = object.dataRecieved;**
102. return object.extractedDatavar;
103. }
105. **int deliverData() {**
107. if (object.turn == false) {
108. object.turn == true;
109. }
110. **object.deliverDat = object.extractedDatavar;**
111. cout << "(Data Link Layer Receiver) Delivering Data";
112. rep(i, 0, 7){
113. Sleep(sleepTime);
114. cout << ".";
115. **}bl;**
116. cout << "Note:(Network Layer Receiver Side) Data delivered..." << endl;
117. cout << "Recieved Data: " << object.deliverDat<<endl<<endl<<endl;
118. return object.deliverDat;
119. }
121. int main() {
122. time\_t start=0, end=0;
123. double difference;
124. int data\_size;
125. **pf("Enter the size of data: ");**
126. sfi(data\_size);
127. object.canSent = true;
128. rep(i, 0, data\_size){
129. //Sender-side code
130. **while (object.turn == true and object.canSent == true) {**
131. waitForEvent();
132. time(&start);
133. if (object.Event == true){
134. if (object.input == true){
135. **getData(data\_size);**
136. object.input = false;
137. }
138. makeFrame();
139. sendFrame();
140. **object.canSent = false;**
141. }
143. break;
144. }
146. //Reciever side code
147. while (object.turn == false) {
148. if (object.Event == true) {
149. recieveFrame();
150. **waitForEvent();**
151. //Reciever to Sender code
152. ackFrame();
153. time(&end);
154. difference = difftime(end,start);
155. **cout << "Time take :" << difference << endl;**
156. object.canSent = true;
158. extractData();
159. deliverData();
160. **object.turn = true;**
161. break;
162. }
163. }
165. **}**
166. pf("Successfully data send....**\n**");
167. }