Preetha Datta
Vidur Singh
Prof. Mahavir Jhawar
Computer Networks
4 December 2018

Stop and Wait Protocol

End Semester Project Report

Abstract

This project implements the stop-and-wait protocol using sockets and threading. The stop and wait protocol is a special case of the Go-back-N protocol , with window size = 1. Through this project we have illustrated and then tackled the possible errors that can happen through erroneous channels during stop-and-wait protocol communication.

Introduction

The Stop-and-Wait protocol is a technique that is used to provide reliability. In this protocol, one frame(in our case, on bit) is sent at a time. The sender does not send any more frames, until it receives an acknowledgement from the receiver for the same. If the acknowledgement fails to arrive within a given time frame, the sender then re-sends the entire frame. This condition is known as a timeout. On the receiver's side, it sends an acknowledgement each time it receives a frame.

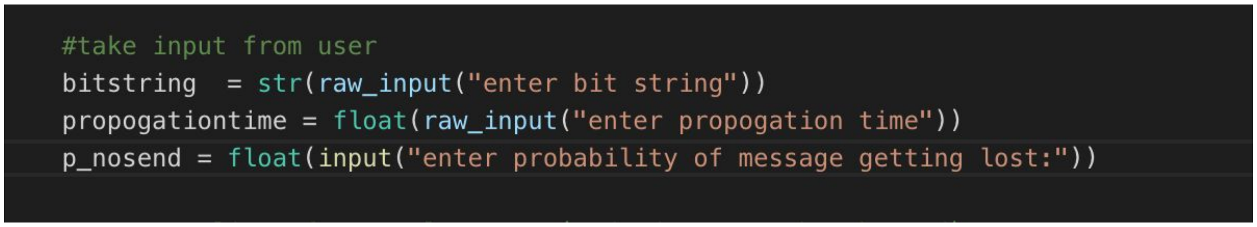
Issues in Communication

The Stop and Wait protocol ensures reliable communication. In essence, the protocol gives insurance against a noisy (or otherwise disturbed) channel that might cause a packet to drop while being sent from the sender side of the program. On the corollary, a noisy channel also might cause the acknowledgement sent back from the receiver's side to get lost. The Stop and Wait protocol ensures communication in both of these cases.

Execution

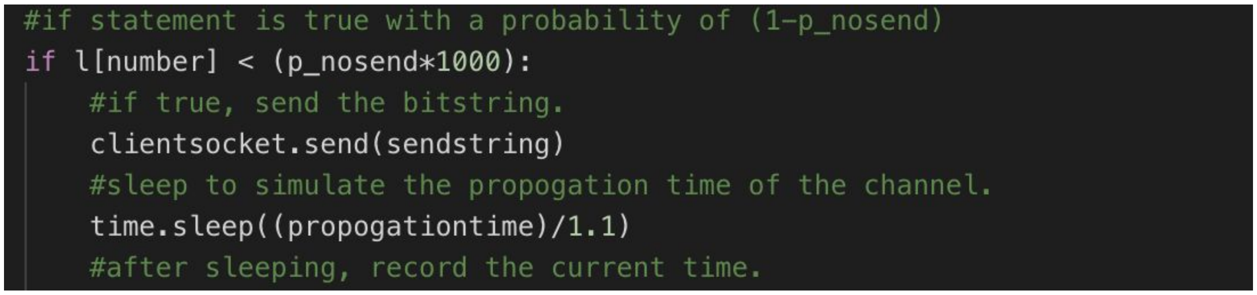
Our execution of the protocol simulates a package drop in a noisy channel. Each side of the program asks the user for a probability of the package getting dropped based on which we then forcibly ensure a package drop.

The sender side of the program asks for a bitstring input from the user (which is to be transmitted across the channel), the probability of package getting dropped and the propagation time.



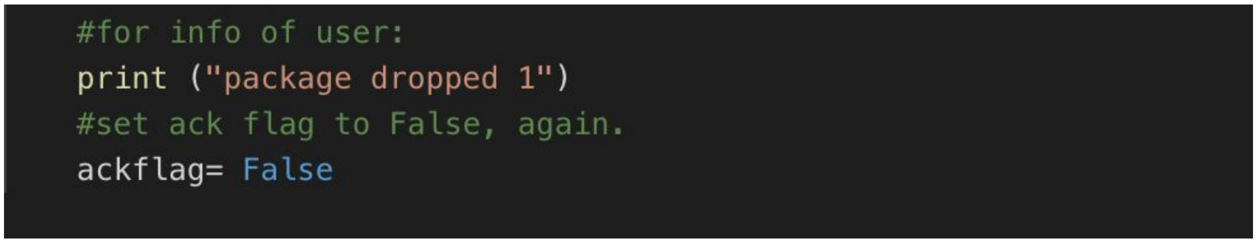
```
#take input from user
bitstring  = str(raw_input("enter bit string"))
propogationtime = float(raw_input("enter propogation time"))
p_nosend = float(input("enter probability of message getting lost:"))
```

We use sockets in order to send given bitstring from the sender to the receiver, with a success rate entered by the user. If true, the packet is sent to the receiver’s side. We then induce a time.sleep(), which is meant to simulate the propagation time.



```
#if statement is true with a probability of (1-p_nosend)
if l[number] < (p_nosend*1000):
    #if true, send the bitstring.
    clientsocket.send(sendstring)
    #sleep to simulate the propogation time of the channel.
    time.sleep((propogationtime)/1.1)
    #after sleeping, record the current time.
```

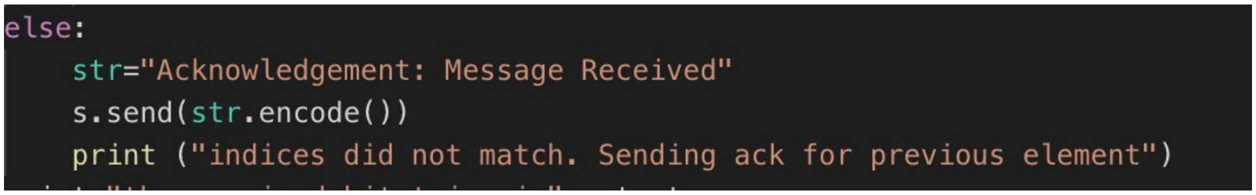
The other case is when the frame is dropped by the channel. In this scenario, we again induce a sleep, in order to simulate the package being sent. As far as the sender "knows", it has sent the package, but it does not know that the frame got lost due to the noisy channel. After waiting for the given time period, the sender does not receive an acknowledgement (since, receiver never received the packet).



```
    #for info of user:
    print ("package dropped 1")
    #set ack flag to False, again.
    ackflag= False
```

After waiting, the sender attempts to send the same packet again, with the same probability and repeats until the package is sent successfully, ie it receives an acknowledgement for it.

For the receiver's side, an acknowledgement is sent every time a frame, or bit, is received, albeit with a success rate given by the user. Therefore, some acknowledgements get lost in the channel. Therefore, since sender does not receive acknowledgement, it times out and sends the same frame again. The receiver, however had already successfully gotten the frame. In this case, since the indices do not match - the acknowledgement for the previous frame is sent.



```
else:
    str="Acknowledgement: Message Received"
    s.send(str.encode())
    print ("indices did not match. Sending ack for previous element")
```

Conclusion

We have implemented the stop-and-wait protocol through this project - and while we realise that it is indeed a reliable method of communication (it ensures that the sent frame is received and acknowledged reliably), we also realised that it is extremely inefficient. The constant waiting, after sending a single frame makes the protocol slow and therefore, we realise that the Go-Back-N protocol or the Selective Repeat protocol is a much faster way to ensure reliable communication.