FLOW CONTROL PROTOCOLS

Stop N Wait, Go Back N, Selective Repeat

NOVEMBER 28, 2020

JADAVPUR UNIVERSITY

Computer Networks Lab Report

Title: "Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control."

Computer Networks Lab Assignment-2

Name: Rahul Kundu

Class: BCSE-III

Roll No: 001810501085

Problem statement:

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

Submission date: **28/11/2020**

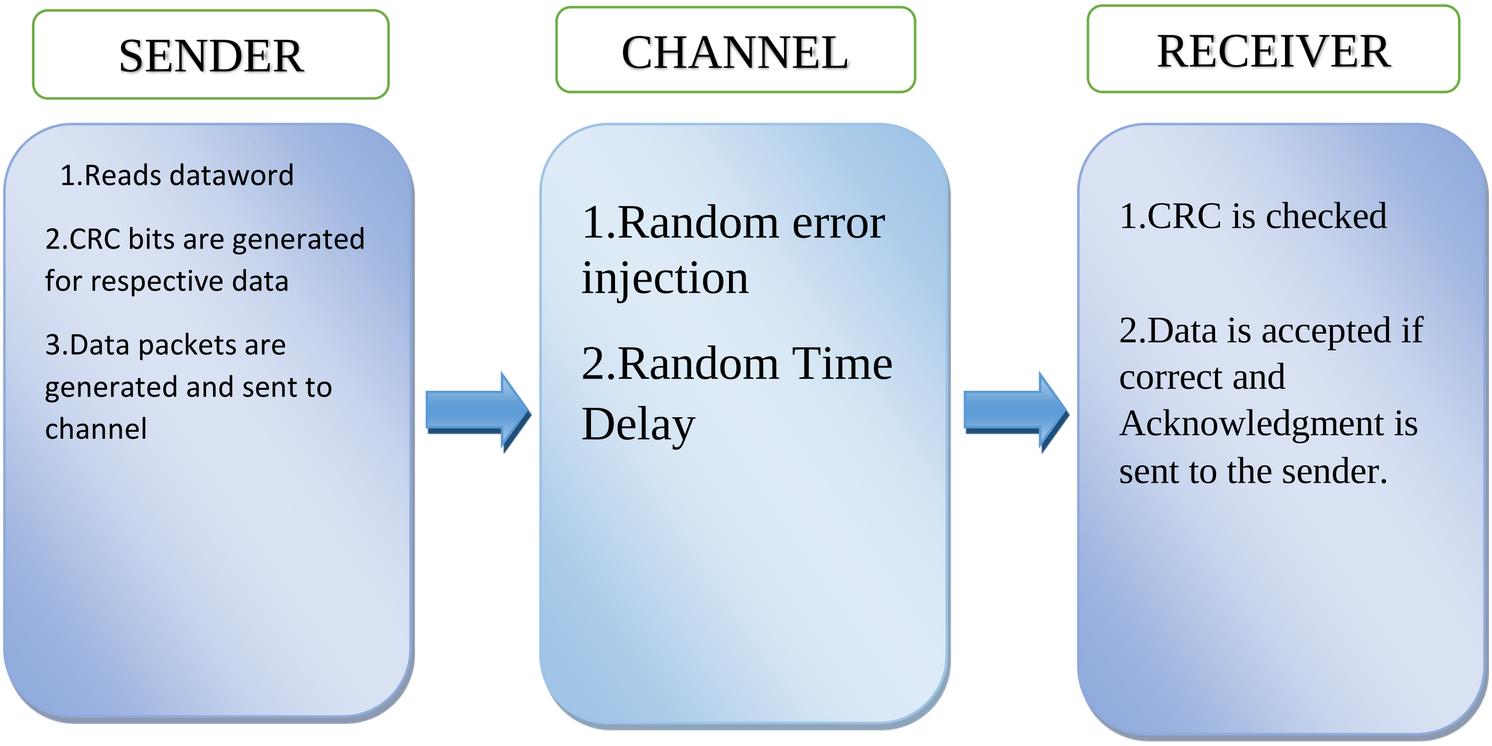
Submission deadline: **28/11/2020**

Design: In this assignment we are implementing **Flow Control Protocols** in **Data Link Layer**.

Those three protocols are:

1. Stop N Wait ARQ
2. Go Back N ARQ
3. Selective Repeat ARQ

STRUCTURE DIAGRAM:



SENDER

Here redundancy bits are generated for every dataword

↓

The datawords are converted into packets

↓

Each packet consist of [preamble + start of frame delimiter + data + CRC]

↓

If the time elapsed is more than the TIMEOUT or RTT > TIMEOUT then it is considered that negative acknowledgement is received and the data is retransmitted

↓

If there is no timeout, it is checked that if there is a negative acknowledgement received or not

↓

If a negative acknowledgement is received, the resend the previous frame else send the next frame

CHANNEL

Both bit and burst errors are injected randomly

Random time delay is also introduced

RECEIVER

Random Time Delay

Data and CRC bits are retrieved after unpacking the Received Data

Data validation is done by CRC Error Control Protocol

If received data is correct then Acknowledgement is sent

If received data is not correct then negative Acknowledgement is sent

USED DATA PACKET FORMAT:

I tried to implement standard Frame Format of Ethernet 802.3 by importing inbuilt module named “struct” in Python.

Ethernet 802.3 Standard Frame Format:

PREMBLE - 7 Bytes
SFD [Start of frame delimiter] - 1 Byte: 10101011
Destination Address - 6 Bytes
Source Address - > 6 Bytes
Length -> 2 Bytes
Data -> Max 1500 Bytes
Cyclic Redundancy Check - 4 Bytes

PREAMBLE	SFD	DESTINATION ADDRESS	SOURCE ADDRESS	LENGTH	DATA	CRC
7 Bytes	1 Byte	6 Bytes	6 Bytes	2 Bytes	46 - 1500 Bytes	4 Bytes

IEEE 802.3 ETHERNET Frame Format

Definition:

- Unsigned long long (Q) and network endian (‘!’) is used to create packets
- Each Data Segment’s length is 8 bytes [‘!QQQQQ’]
- I’ve used total 5 segments (instead of 7 segments as I was facing difficulties while sending the packets)
- I’ve used total 40 bytes in my design

Limitations of My Frame Format:

As I’ve used ‘Q’ (unsigned long long) for data segment, my Frame Format is limited to 40 bytes. I got “Memory Limit Exceeded” error while implementing Ethernet 802.3 Standard Frame Format.

ESTABLISHMENT OF CONNECTION:

I've implemented sockets for connection using the python module socket.

Code snippets are as follows for sender side as well as receiver side:

Sender:

```
# Establishing Connection
s = socket.socket()
host = socket.gethostname()
ip = socket.gethostbyname(host)
port = 8000
s.bind((host, port))
name = '<----------------------------WELCOME TO SENDER STATION------------------------->'
# Receiver End Connection
s.listen(1)
print("\nConnecting........................\n")
# get the socket object and address info from the receiver end
conn, addr = s.accept()
print("Connected to ---->", addr[0], "(", addr[1], ")")
# buffer size = 1518
s_name = conn.recv(1518)
s_name = s_name.decode()
print("\n", s_name)
conn.send(name.encode())
```

Receiver:

```
# Establishing Connection
s = socket.socket()
host1 = socket.gethostname()
ip = socket.gethostbyname(host1)
host = str(ip)
port = 8000
name = '<------------------------------WELCOME TO RECEIVER STATION-------------------------->'
# Connect to remote ADDR, and then wraps the connection in an SSL channel
s.connect((host, port))
s.send(name.encode())
s_name = s.recv(1518)
s_name = s_name.decode()
print(s_name)
# Sender end Connection
f = s.recv(1518)
f = f.decode()
f = int(f)
```

WHAT'S IN CHANNEL?

I've implemented random error injection function, random delay, data packet maker with padding, CRC checker

Error Injection:

```
# Error Injection
def injectError(L):
    t = random.randint(0, 7)
    size = len(L)
    if t == 1:
        i = random.randint(0, size - 1)
        if L[i] == '0':
            L = L[:i] + '1' + L[i + 1:]
        else:
            L = L[:i] + '0' + L[i + 1:]
    elif t == 2:
        print("Random Error Injected\n")
        n = random.randint(0, size - 1)
        for x in range(n):
            i = random.randint(0, size - 1)
            if L[i] == '0':
                L = L[:i] + '1' + L[i + 1:]
            else:
                L = L[:i] + '0' + L[i + 1:]
    else:
        print("No Error Injected")
    return L
```

CRC Checker:

```
def CRCcheck(data, polynomial):
    l = len(polynomial)
    codeword = data + '0' * (l - 1)
    remainder = CRC(codeword, polynomial)
    flag = 1
    for i in range(0, len(remainder)):
        if remainder[i] == '1':
            flag = 0
            break

    if flag == 1:
        print(">> Remainder : " + remainder)
        return 0
    else:
        print(">> Remainder : " + remainder)
        return 1
```

Implementations of Protocols:

I’ve implemented these flow control protocols in python. The code snippet are as follows.

STOP N WAIT ARQ:

1) The idea of stop-and wait arq is pretty straightforward , after transmitting one frame the sender waits for an acknowledgement before transmitting the next frame

2) If the acknowledgement does not arrive after certain period of time (RTT), the sender times out and retransmits the original frame.

Sender:

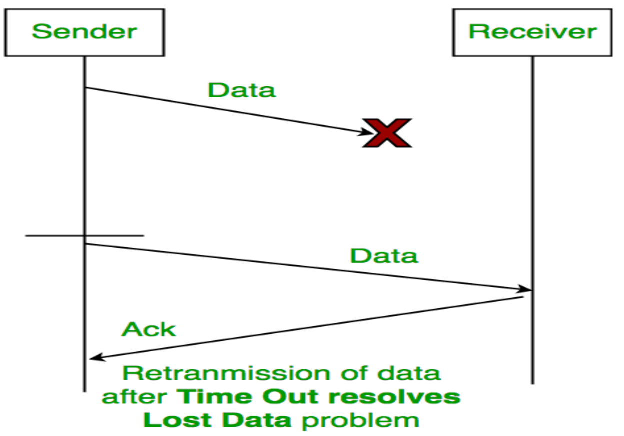
```
while True:
    counter = 0
    while counter < f:
        # send the dataframe
        conn.send(message[counter])
        # set timeout
        conn.settimeout(3)
        try:
            ack = conn.recv(1518)
            ack = ack.decode()
        except socket.timeout:
            print("Timeout!")
            ack = "1"
        if ack != "1":
            print("||------------->Ack received!")
            counter = counter + 1
            time.sleep(1)
        else:
            print("Resending!")
            time.sleep(1)

    # Closing Connection
    conn.close()
    print("Connection Closed")
    break
```

Receiver:

```
while True:
    counter = 0
    message = []
    polynomial = '1011'
    while counter < f:
        # receive the data packet
        temp = s.recv(1518)
        temp = unpacker(temp)
        # Injecting error
        temp = injectError(temp)
        # Data Validation
        error = CRCcheck(temp, polynomial)
        # random time delay
        a = random.randint(0, 4)
        time.sleep(a)
        print("Time: ", a)
        # 0 for error and 1 for no error
        e = str(error)
        # if timeout happens nothing is being sent
        if a < 3:
            s.send(e.encode())
        else:
            error = 1
        if a < 3:
            if e != "1":
                print("\n>>", temp[:-3], "\n")
                counter = counter + 1
    # Closing Connection
    s.close()
    break
```

Timeout:



Characteristics of Stop and Wait ARQ:

- It uses link between sender and receiver as half duplex link
- Throughput = 1 Data packet/frame per RTT
- If Bandwidth*Delay product is very high, then stop and wait protocol is not so useful. The sender has to keep waiting for acknowledgements before sending the processed next packet.
- It is an example for "**Closed Loop OR connection oriented** " protocols
- It is an special category of SWP where its window size is 1
- Irrespective of number of packets sender is having stop and wait protocol requires only 2 sequence numbers 0 and 1

Pros and Cons:

The Stop and Wait ARQ solves main three problems, but may cause big performance issues as sender always waits for acknowledgement even if it has next packet ready to send. Consider a situation where you have a high bandwidth connection and propagation delay is also high (you are connected to some server in some other country though a high speed connection). To solve this problem, we can send more than one packet at a time with a larger sequence numbers. We will be discussing these protocols in next articles.

So Stop and Wait ARQ may work fine where propagation delay is very less for example LAN connections, but performs badly for distant connections like satellite connection.

GO BACK N ARQ:

1) Here ‘n’ denotes the size of window, in another way go-back-n means it can send ‘n’ numbers of frames before receiving an acknowledgement.

2) The window of frames is numbered as 0,1,2,…………,n

3) If the acknowledgement is not received within the agreed upon time period, all frames in the current window are transmitted.

Sender:

```
while True:
    'In go-back-n n frames are transmitted before receiving an acknowledgement'
    # First dataframes are transmitted before receiving an ack
    for j in range(0, n):
        time.sleep(1)
        conn.send(message[j])
        print("||------------------->Sending.....")
        time.sleep(1)

    i = n
    counter = 0  # acknowledgement counter
    TIMEOUT = 3

    switch = 0
    hpass = 0
    holder = message[0]
    windowEdge = n - 1
    while counter < f:
        # receive ack signals
        # time1 = time.time()
        conn.settimeout(3)
        try:
            ack = conn.recv(1518)
            ack = ack.decode()
        except socket.timeout:
            print("||------------------->Timeout Error")
            ack = "1"
        if i < f:
            switch = 0
        else:
            switch = 1
```

```
    # only ack left
    if switch == 1:
        if ack != "1":
            print("\n||------------------->ACK ", counter)
            print("\n")
            time.sleep(1)
            a = message[counter]
            holder = a
            counter = counter + 1
            windowEdge = windowEdge + 1
            if windowEdge >= f:
                windowEdge = f - 1
        else:
            # for nack
            print("\n||------------------->NACK NO", counter)
            print("||------------------->Resending the frame ", counter, "\n")
    if switch == 0:
        # transmit the next dataframe
        if ack != "1":
            print("\n||------------------->ACK NO ", counter)
            print("||------------------->Sending the next frame...", (counter + n), "\n")
            time.sleep(1)
            holder = message[counter]
            i = i + 1
            counter = counter + 1
            windowEdge = windowEdge + 1
            if windowEdge >= f:
                windowEdge = f - 1
        else:
            print("\n||------------------->NACK", counter)
            print("||------------------->Resending the frame ", counter, "\n")
```

```
    # how many frames need to be Sent
    if ack != "1":
        conn.send(holder)
    else:
        a1 = windowEdge - (n - 1)
        a2 = windowEdge + 1
        for a in range(a1, a2):
            time.sleep(1)
            conn.send(message[a])
            print("||------------------->Sending")
            time.sleep(1)
```

```
# ___________________________________________________________________#
conn.close()
print("<----------------------------------CONNECTION CLOSED------------------------------------->")
break
```

Receiver:

```
while True:
    counter = 0
    message = []
    polynomial = '1011'
    while counter < f:
        # receive the data packet
        temp = s.recv(1518)
        temp = unpacker(temp)
        # Injecting error
        temp = injectError(temp)
        # Data Validation
        error = CRCcheck(temp, polynomial)
        # random time delay
        a = random.randint(0, 4)
        time.sleep(a)
        print("Time Delay Encountered is: ", a)
        # 0 for error and 1 for no error
        e = str(error)
        # if timeout happens nothing is being sent
        if a < 3:
            s.send(e.encode())
        else:
            error = 1
        if a < 3:
            if e != "1":
                print("\n>>Partition Received is :", temp[:-3], "\n")
                counter = counter + 1
    # Closing Connection
    s.close()
    print("\n<--------------------Connection Closed--------------------->")
    break
```

Feedbacks/Acknowledgements

There are basically 2 types of feedbacks/acknowledgments:

1. **<u>Cumulative Ack</u>:** Here we use only one feedback for many data packets, because of this the main advantage we get is that the traffic is less. But it can also result in a huge drawback which is, if one acknowledgment is lost then it means that all the data packets transmitted are lost.
2. **<u>Independent Ack</u>:** Here every data packet gets acknowledged independently. Here the reliability is high, but the main drawback is high traffic.

Go-Back-N ARQ uses a cumulative acknowledgment technique, which means receiver starts an acknowledgment timer whenever it receives a data packet which is fixed & when it expires, it will transmit a cumulative acknowledgment for the number of data packets received in that interval of time out timer. Now, if the receiver has received N data packets, then the feedback/acknowledgment number is going to be N+1. The important point here is that the acknowledgment timer will not start after the expiry of the first-timer, but it will do so when the receiver has received a data packet. The thing which should be kept in mind is that the time out timer at the sender node must be greater than the acknowledgment timer.

SELECTIVE REPEAT REQUEST ARQ:

1) In selective repeat arq there is a window of size n similar to go-back-n
2) In this case, only the erroneous or lost frames are retransmitted, while correct frames are received and buffered.
3) The receiver while keeping track of sequence numbers, buffers the frames in memory and sends NACK for only frame which is missing or damaged.
4) The sender will transmit packet for which NACK is received.

Sender:

```
while True:
    'In selective repeat only the lost data frames are retransmitted'
    # First dataframes are transmitted before receiving an ack
    for j in range(0, n):
        time.sleep(1)
        # print(time.time())
        conn.send(message[j])
        print("||------------------->Sending")
        time.sleep(1)
    i = n
    counter = 0  # acknowledgement counter
    TIMEOUT = 3

    switch = 0
    hpass = 0
    holder = message[0]
    while counter < f:
        # receive ack signals
        # time1 = time.time()
        conn.settimeout(3)
        try:
            ack = conn.recv(1518)
            ack = ack.decode()
        except socket.timeout:
            print("||------------------->Timeout Error")
            ack = "1"

        if i < f:
            switch = 0
        else:
            switch = 1
```

```python
# only ack left
if switch == 1:
    if ack != "1":
        print("\n||------------------->ACK NO ", counter)
        print("\n")
        time.sleep(1)
        a = message[counter]
        holder = a
        counter = counter + 1
    else:
        # for nack
        print("\n||------------------->NACK NO ", counter)
        print("||------------------->Resending the frame ", counter, "\n")
        time.sleep(1)
        holder = message[counter]
if switch == 0:
    # transmit the next dataframe
    if ack != "1":
        print("\n||------------------->ACK NO ", counter)
        print("||------------------->Sending the next frame...", (counter + n), "\n")
        time.sleep(1)
        holder = message[i]
        i = i + 1
        counter = counter + 1
    else:
        print("\n||------------------->NACK NO ", counter)
        print("||------------------->Resending the data packet ", counter, "\n")
        time.sleep(1)
        holder = message[counter]
conn.send(holder)

conn.close()
```

Receiver:

```python
while True:
    buffer = []
    message = []

    polynomial = '1011'
    # create a buffer to store data temporarily
    for x in range(0, n):
        buffer.append('0')

    for j in range(0, n):
        time.sleep(1)
        temp = s.recv(1518)
        temp = unpacker(temp)
        buffer[j] = temp
        time.sleep(1)

    i = n
    counter = 0
    time1 = time.time()
    TIMEOUT = 3
    switch = 0
```

```python
while counter < f:
    # send ack for correct data frame and nack for incorrect data frame
    # 0 denotes no error 1 denotes error
    # error = random.randint(0,1)
    # Check CRC
    valid = injectError(buffer[counter % n])
    error = CRCcheck(valid, polynomial)
    # ----> 1 represent error and 0 represent no error <----#
    e = str(error)
    # send the ack

    # random time delay <-------------------------------------->
    a = random.randint(0, 4)
    time.sleep(a)
    print("Time: ", a)
    # <------------------------------------------------------------>
    # if timeout happens nothing is being sent
    if a < 3:
        s.send(e.encode())
    else:
        error = 1

    # collect the message from buffer
    if a < 3:
        if error != 1:
            message.append(valid[:-3])
```

```python
        if i < f:
            switch = 0
        else:
            switch = 1
        # receiver
        temp1 = s.recv(1518)
        temp1 = unpacker(temp1)
        # print(temp1)
        # validate remaining ack
        if switch == 1:
            if error != 1:
                counter = counter + 1
            else:
                buffer[counter % n] = temp1

        if switch == 0:
            # receive the dataframe
            if error != 1:
                # reorganize the buffer
                x = 0
                for x in range(0, n - 1):
                    buffer[x] = buffer[x + 1]
                buffer[n - 1] = temp1
                i = i + 1
                counter = counter + 1

            else:
                # correct the values in buffer
                buffer[counter % n] = temp1
    s.close()
    print("<----------------------------------CONNECTION CLOSED------------------------------------>")
    break
```

Acknowledgements:

- In this flow control protocol it also has similar sliding window as GO-BACK-N
- This protocol is the improved version of GO-BACK-N as, Here the sender only retransmits data for the wrong frames and the correct ones are buffered

Measurement Parameters:

Delay is nearly 0.000000s in socket connection for bandwidth of 1518 bytes

Size of each data packet is = 40 Bytes

Size of the bandwidth = 1518 Bytes

RESULT AND ANALYSIS:

NAME	Packet per Time	RTT	Utilization Percentage	COMMENTS	EFFICIENCY OF FLOW CONTROL
STOP N WAIT	1	0.03-0.01	40/1518= 0.0263	One frame is send at a time which is certainly not an efficient method to send data over network	Poor
GO BACK N	n	0.07(for ack-0, n =4)	4*40/1518= 0.1054(for n = 4)	Resending the whole window is unnecessary as for only a single data frame error it has to transmit several frames again and again which is time and memory inefficient.	Better Than Stop N Wait
SELECTIVE REPEAT	n	0.07(for ack-0,n = 4)	4*40/1518= 0.1054(for n = 4)	This protocol is the improved version of GO-BACK-N as, Here the sender only retransmits data for the wrong frames and the correct ones are buffered	Mostly Used and better than both of the above protocols

Comments:

What did I learn from it?

During the making of this assignment I got to know about 3 different Flow Control Mechanisms clearly

Was it too hard? (Explain why?) Too easy? (Explain why?)

Considering the prerequisites of the assignment it was neither so hard nor too easy.

One Major drawback of my design is that, I was unable to make the connections multi sender and multi receiver.

But the most difficult part for me was implementing IPC (Inter Process Communication).

I initially tried to use message queue instead of using socket (Worst Option for Learning), but I wasn't able to completely run those codes or establish the connections properly.

Later on I moved to socket as it was my last option.

I encountered MLE many times during the implementation of Ethernet Standard Frame 802.3.

Ultimately I was able to design the assignment.