National Institute of Technology, Raipur

Department of Computer Science & Engineering

Transmission Control Protocol Over User Datagram Protocol

A Term Project on Network Programming

GitHub Project Link:
https://github.com/Vipul-Bajaj/TCP-Over-UDP

Submitted By:

Roll no: 14115085	Name: Shubham Gupta
Roll no: 14115901	Name: Aishwarya Srivastava
Roll no: 14115902	Name: Ekta Agrawal
Roll no: 14115904	Name: Vipul Bajaj

Abstract

Implementation of the functionality of TCP using UDP. We have simulated features of TCP like flow control, message reliability, congestion control etc. using UDP.

In computer networking, the User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths. The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. Also, it performs very limited error checking.

UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system. Applications use datagram sockets to establish host-to-host communications. An application binds a socket to its endpoint of data transmission, which is a combination of an IP address and a service port. A port is a software structure that is identified by the port number, a 16 bit integer value, allowing for port numbers between 0 and 65535. Port 0 is reserved, but is a permissible source port value if the sending process does not expect messages in response.

UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload. If transmission reliability is desired, it must be implemented in the user's application. Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Some applications, such as TFTP, may add rudimentary reliability mechanisms into the application layer as needed. Most often, UDP applications do not employ reliability mechanisms and may even be hindered by them. Streaming media, real-time multiplayer games and voice over IP (VoIP) are examples of applications that often use UDP. In these particular applications, loss of packets is not usually a fatal problem.

We have tried to solve this problem of unreliability in this connectionless protocol for data transfer by adding flow control and error control mechanisms**[1]** similar to that present in TCP. Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment. Error control in the data link layer is based on automatic repeat request, which is the retransmission of data. We have implemented Stop and Wait and Go-Back-N ARQ Protocols for adding reliability to UDP.

List of Figures

Figure 1: Design of the Stop-and-Wait ARQ Protocol..8
Figure 2: Design of Go-Back-N ARQ..9

Table of Contents

Abstract..2

List of Figures..3

Introduction...5

Detailed Description..7

1. Stop-and-Wait Automatic Repeat Request...7
2. Go-Back-N Automatic Repeat Request 8

Conclusion 10

References…………………………………………………………………………...11

Introduction

UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP. UDP uses concepts common to the transport layer**[2]**.

Connectionless Services

UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP.

Flow and Error Control **(Problem)**

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages. There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded. The lack of flow control and error control means that the process using UDP should provide these mechanisms.

Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.

Queuing

We have talked about ports without discussing the actual implementation of them. In UDP, queues are associated with ports.

At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process. Note that even if a process wants to communicate with multiple processes, it obtains only one port number and eventually one outgoing and one incoming queue. The queues opened by the client are, in most cases, identified by ephemeral port numbers. The queues function as long as the process is running. When the process terminates, the queues are destroyed.

The client process can send messages to the outgoing queue by using the source port number specified in the request. UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system can ask the client process to wait before sending any more messages.

When a message arrives for a client, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the server. All the incoming messages for one particular client program, whether coming from the same or a different server, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the server.

At the server site, the mechanism of creating queues is different. In its simplest form, a server asks for incoming and outgoing queues, using its well-known port, when it starts running. The queues remain open as long as the server is running.

When a message arrives for a server, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the client. All the incoming messages for one particular server, whether coming from the same or a different client, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the client.

When a server wants to respond to a client, it sends messages to the outgoing queue, using the source port number specified in the request. UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system asks the server to wait before sending any more messages.

(Project Outline)

User Datagram Protocol (UDP) is designed as light weight protocol with no support for handshaking, ordering and error recovery etc. Hence, UDP is a preferred protocol where reliability can be traded-off for the end-to-end delay experienced by the users. This makes UDP suitable for delay sensitive traffic such as video and voice applications. In the literature, various models have been presented for Transmission Control Protocol (TCP), a dominant transport protocol for the Internet. There does not exist any model for the UDP throughput calculation to the best of our knowledge. In this project we have presented an algorithmic model for the reliable UDP in the presence of TCP and implemented with Protocols for flow and error control.

There are 2 versions of the Sender and Receiver,each with increasing levels of data reliability and throughput.

Sender and Receiver 1 uses Stop and Wait ARQ Protocol.

Sender and Receiver 2 uses Go-Back-N ARQ Protocol.

Detailed Description

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism. A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgement packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming.

We have implemented the following two approaches to make it reliable like TCP.[3]

1. *Stop-and-Wait Automatic Repeat Request*

To detect and correct corrupted frames, we need to add redundancy bits to our data frame. When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded. The received frame could be the correct one, or a duplicate, or a frame out of order. The solution is to number the frames. When the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated.

The completed and lost frames need to be resent in this protocol. If the receiver does not respond when there is an error, how can the sender know which frame to resend? To remedy this problem, the sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted. Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number. The ACK frame for this protocol has a sequence number field. In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.

In Stop-and-Wait ARQ we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic. In Stop-and-Wait ARQ the acknowledgment number always announces in modul0-2 arithmetic the sequence number of the next frame expected.

Design

Figure 1 shows the design of the Stop-and-Wait ARQ Protocol. The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call Sn (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1). The receiver has a control variable, which we call Rn (receiver, next frame expected), that holds the number of the next frame expected. When a frame is sent, the value of Sn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of Rn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site. Variable Sn points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged; Rn points to the slot that matches the sequence number of the expected frame.

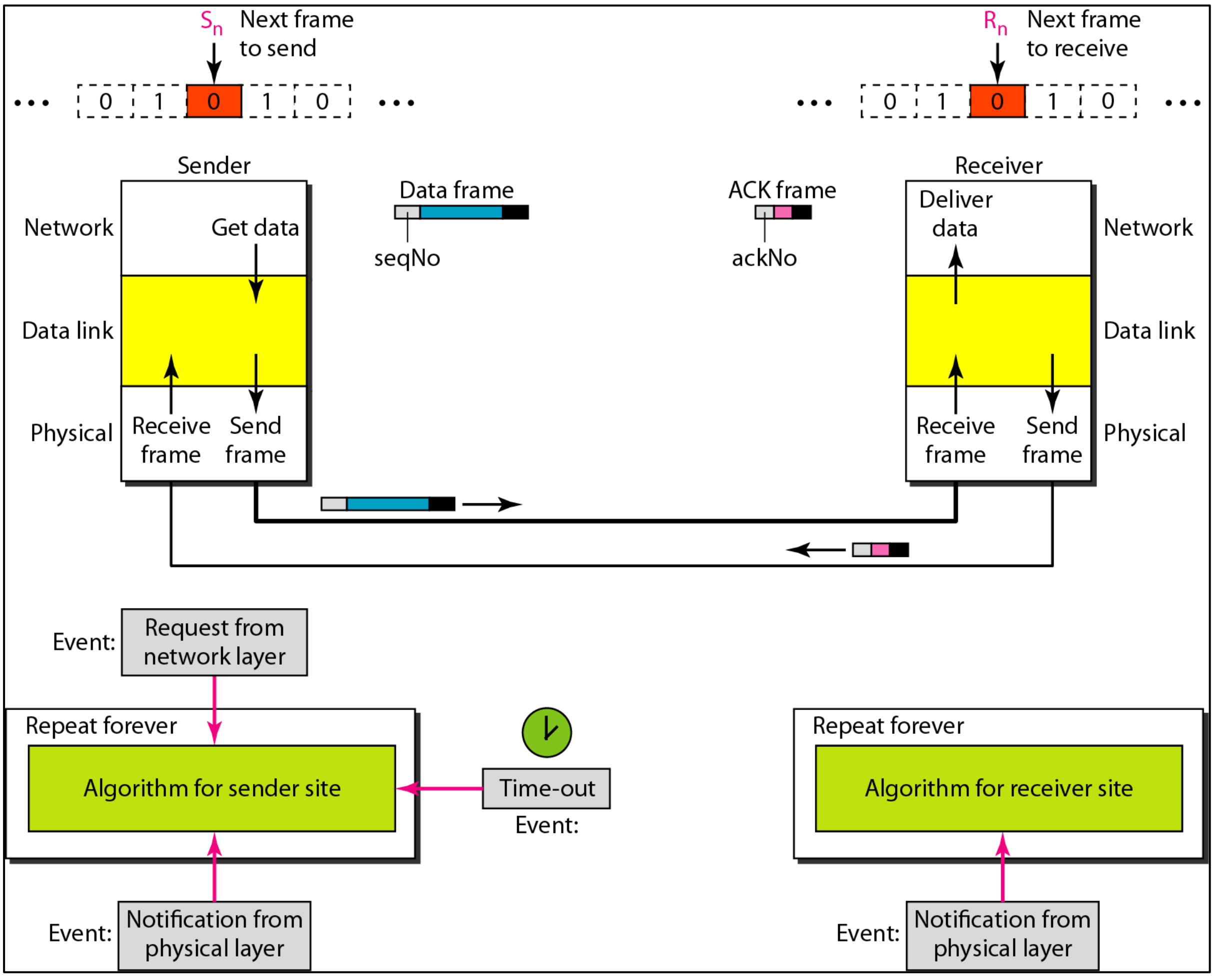

***Figure 1:** Design of the Stop-and-Wait ARQ Protocol*

2. *Go-Back-N Automatic Repeat Request*

In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive. In the Go-Back-N Protocol, the sequence numbers are modulo1!" where m is the size of the sequence number field in bits. In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver.

The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is 2m – 1. The window itself is an abstraction; three variables define its size and location at any time. We call these variables Sf(send window, the first outstanding frame), Sn (send window, the next frame to be sent), and S size (send window, size). The variable Sf defines the sequence number of the first (oldest) outstanding frame. The variable Sn holds the sequence number that will be assigned to the next frame to be sent. Finally, the variable S size defines the size of the window, which is fixed in our protocol.

Timers

Although there can be a timer for each frame that is sent, in our protocol we use only one.

The reason is that the timer for the first outstanding frame always expires first; we send all outstanding frames when this timer expires.

Acknowledgment

The receiver sends a positive acknowledgment if a frame has arrived safe and sound and in order. If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting. The silence of the receiver causes the timer of the unacknowledged frame at the sender site to expire. This, in turn, causes the sender to go back and resend all frames, beginning with the one with the expired timer. The receiver does not have to acknowledge each frame received. It can send one cumulative acknowledgment for several frames.

Resending a Frame

When the timer expires, the sender resends all outstanding frames. For example, suppose the sender has already sent frame 6, but the timer for frame 3 expires. This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4, 5, and 6 again. That is why the protocol is called Go-Back-NARQ.

Design

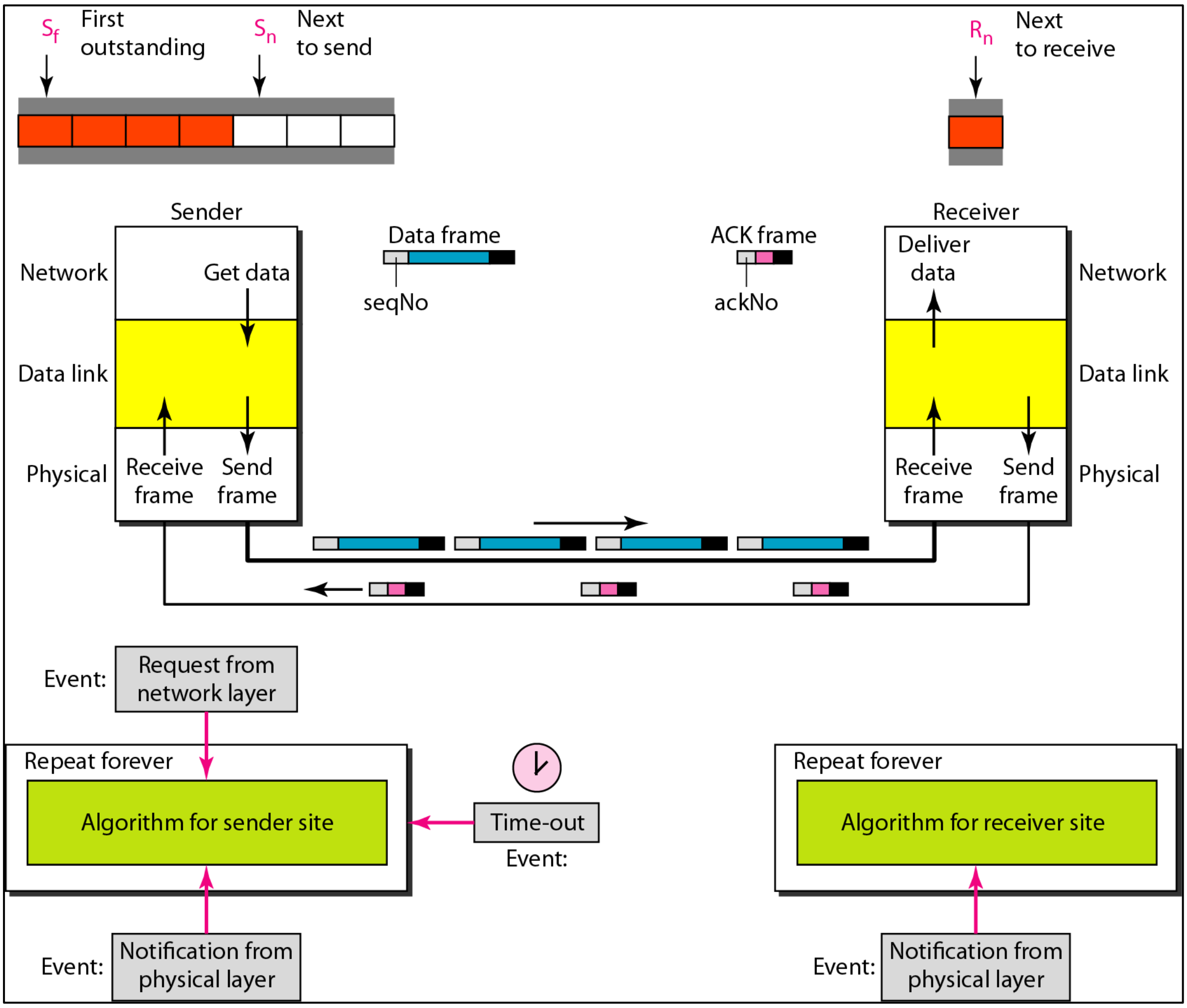

Figure 2: *Design of Go-Back-N ARQ*

Conclusion

The User Datagram Protocol offers only a minimal transport service, non-guaranteed datagram delivery and gives applications direct access to the datagram service of the IP layer. UDP is used by applications that do not require the level of service of TCP or that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP.

UDP is almost a null protocol; the only services it provides over IP are check summing of data and multiplexing by port number. Therefore, an application program running over UDP must deal directly with end-to-end communication problems that a connection-oriented protocol would have handled e.g., retransmission for reliable delivery, packetization and reassembly, flow control, congestion avoidance, etc., when these are required. The fairly complex coupling between IP and TCP will be mirrored in the coupling between UDP and many applications using UDP.

In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing[4].

We have used Stop and Wait and Go-Back N ARQ to make reliable UDP like TCP. We may find that there is a similarity between Go-Back-N ARQ and Stop-and-Wait ARQ. We can say that the Stop-and-Wait ARQ Protocol is actually a Go-Back-NARQ in which there are only two sequence numbers and the send window size is 1. In other words, m = 1, 2m - 1 = 1. In Go-Back-N ARQ, we said that the addition is modulo-2m; in Stop-and-Wait ARQ it is 2, which is the same as 2m when m = 1.

References

[1] A. O. F. Atya and J. Kuang, “RUFC: A flexible framework for reliable UDP with flow control,” *2013 8th Int. Conf. Internet Technol. Secur. Trans. ICITST 2013*, pp. 276–281, 2013.

[2] A. Tanenbaum and D. Wetherall, *COMPUTER NETWORKS*, Fifth. Prentice Hall, 2011.

[3] B. Forouzan and S. Fegan, *Data Communications and Networking*, Fourth Edition. Alan R. Apt, 2007.

[4] S. Thombre, “Modelling of UDP throughput,” *IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON*, pp. 482–487, 2017.