# Cypher: a New Application Level Internet Protocol

Kushal Sultania and V Sakthivel

December 17, 2023

# Cypher : A New Application Level Internet Protocol

Kushal Sultania
*School of Computer Science and Engineering,*
*Vellore Institute of Technology,*
Chennai,
600127,
Tamilnadu,
India
Email: kushalsultania2@gmail.com

Sakthivel V
*School of Computer Science and Engineering,*
*Vellore Institute of Technology,*
Chennai,
600127,
Tamilnadu,
India
Email: sakthivel.v@vit.ac.in
(Corresponding author)

**Abstract : The Internet Protocol (IP) is a critical component of computer networks, playing a crucial role in data exchange between computers. Despite the development of several IP protocols, these protocols have limitations such as asynchronous implementation, a lack of connection stabilization features, and rigid request and response structures. This research paper aims to introduce a new IP protocol that addresses these limitations by creating and implementing an open and semi-rigid stateful protocol. The goal is to provide improved performance, ease of use, and flexibility in data exchange over networks. This work involves researching and analyzing existing IP protocols, designing and testing the new protocol, and evaluating its performance and functionality. The results of this research will contribute to the advancement of computer networks and data exchange technology.**

***Index Terms*—stateful, stateless, Web 4, efficient protocols, high speed data transfer, internet protocol, Cypher, multithreaded server**

## I. INTRODUCTION

We all know the importance of the internet in our daily lives. The whole idea about how the internet develops our capabilities and how it helps everyone is well documented in [1] [2]. This happening is well understood that only the internet has provided that much accessibility to everyone that no one was able to provide. This does not mean that life of each person became easy, some people struggled to design the internet over the seas to create a generic working, stable and secure model that works for everyone and everywhere. In the 1960's the very first working model of the internet was made by Advanced Research Projects Agency Network(ARPANET). ARPANET allowed students and researchers to share research papers and development resources. Earlier it didn't have any standards of information exchange like a protocol and a schema or structure for safety compliance and stability and it also didn't have any reliability. Then in 1980's it was made using the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite, TCP/IP started to become standards of ARPANET. The implementation stack for TCP/IP for 8 bit architectures can be studied from [1] [3]. User Datagram Protocol was not used for the purpose as it does not provide any control over data packets, packets get lost if any unusual event occurs. If packet loss occurs, the listener of the message was on its own, also the UDP was very insecure due to its limitations. Request for Comments (RFC) defines the compliance of all networking or hardware level protocols so that integrity of all devices on the internet can be maintained and it defines the standard of all the protocols and everyone must comply with it. UDP documentation by RFC is available at [1] [4]. Detailed documentation of TCP/IP compliance is available at [1] [5]. These were the protocols that were at transport level. Transport level protocols are responsible for managing structures of packets and ensuring the communication between the peers and making sure that the messages reach the destination.

As years passed, demand for connectivity increased and the need for applications was high. Thus the need for protocols at a higher level led to development of application level protocols. Hypertext Transfer Protocol/HTTP Secure (HTTP/HTTPS) are part of application level protocols as they don't need to care about lower level parts such as hardware level and network level protocols that are implemented so that hardware works fine and there is no collision in their network. The problem was these protocols were still not efficient as they could be, they were stateless and very much bloated. Stateless protocols do not stay connected for a long time, as the request is served they close automatically.

So, each time a request is made and served :

- Socket is created.
- Connection is established
- Request is made
- Request is served by a response
- Response is received
- Connection is closed
- Socket is deleted

So, for making n requests we need to execute/do these seven steps each time which sums up to 7n repetitive tasks. This is a very big overhead for computers and networks. This thing needs to be reduced.

This paper focus to implement prototype of an application level internet protocol that solves the following problems :

- Security
- Connection Stabilization
- Stateful connections
- Reducing network traffic
- Enable high speed data transfers over networks
- Multithreaded implementation
- Efficient data transfers

• Improving the request terminating characters or request delimiters

## II. TCP/IP The Legacy

TCP/IP suite is the most used internet protocol suite under the hood without knowing it, even if we use any custom protocol like HTTP/HTTPS, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) or any other known protocol it uses sockets. Sockets are gateways to computer networks through which data comes in or goes out to the target specified. Sockets have many attributes associated with it, the most common are : sender IP, target IP, timeout limit, receive buffer size, transmission buffer size. It establishes connection between peers by a 3-way handshake discussed in [3] [2] and detailed research and info is available at [1] [6]. Also the data is not transmitted as a whole, it is done in small chunks called packets, each packet having its own attributes discussed in [3] [3]. The properties of the packet are responsible for the packet reaching its destination before its lifetime finishes. It also defines the lifetime of the packet and what to do after lifetime exceeds.

### *A. 3-way Handshake*

The TCP 3-way handshake is a process used in a Transmission Control Protocol (TCP) network to establish a reliable connection between two devices before data transmission can begin. Complete reference for 3-way handshake is available at [1] [4]. Handshake involves three steps :

• SYN (Synchronize): The initiating device, also known as the client, sends a synchronization request (SYN) to the receiving device, also known as the server. This request includes the initial sequence number (ISN) that the client will use for the data transmission. The ISN is a random number chosen by the client to identify the individual segments of data that it will send to the server.

• SYN-ACK (Synchronize-Acknowledgment): The server responds to the client's SYN request with a SYN-ACK message, which includes its own ISN. The SYN-ACK message acknowledges that the server has received the client's SYN request and is prepared to receive data.

• ACK (Acknowledgment): The client then sends an acknowledgment (ACK) message to the server to confirm that it has received the SYN-ACK and that the connection is established. The ACK message includes the client's ISN + 1, indicating that it has received the server's Initial Sequence Number (ISN) and is ready to receive the first segment of data. Refer Fig 2.1 for visualization of SYN/ACK

More details about TCP/IP is available in [1] [6].TCP/IP also ensures that the connection is established in an efficient manner and it also prevents the existence of half open connections. Open connections come into existence if peer A requests for connection to peer B and b does not accept that request and peer A don't close the connection on its own or if two peers are fully connected but on of the peer coles the connection but the other peer do not know about it and stay half alive without knowing it. This was addressed by TCP/IP by using 3-way handshake, it prevents half open connections by half open connection detection and then closing it.

### *B. TCP/IP Packets*

Data is always transmitted in very small units called packets. A TCP packet is a unit of data that is transmitted between two devices over a network using the TCP protocol. It is responsible for ensuring that data is transmitted reliably and efficiently between the devices. TCP packet consists of two parts: a header and the data. The header contains information about the data and the transmission, while the data is the actual information being transmitted. Refer Fig 2.2 for TCP/IP packet structure.

Packets are in general kept very small for minimizing packet loss penalty; the complete specification is given in [1] [5]. Effect of the packet loss penalty can be studied from [1] [7]. In short, if a large packet is lost then regenerating that packet is very costly as compared to regenerating a smaller packet and sending, regenerating a large packet is time consuming and it increases network ping. Also it can lead to network congestion and congestion at the particular node which is regenerating the packet. The header of a TCP packet includes the following information :

- Source and destination port numbers: These identify the applications on the sending and receiving devices that are transmitting and receiving the data.
- Sequence number and acknowledgment number: The sequence number is used to keep track of the order of the data segments, while the acknowledgment number is used to confirm the receipt of data by the receiving device.
- Flags: These indicate specific actions, such as the SYN or ACK flags in the 3-way handshake.
- Window size: This field specifies the amount of data that the receiver is willing to accept from the sender.
- Checksum: This is used to verify the integrity of the data in the packet.

All these things together guide the path of packet, nature and time to live (TTL). TTL is based on the timeout of the other socket to which one socket is connected.

## III. Application Level Protocols

The evolution of application level protocols has been shaped by the development of the Internet and the increasing number of connected devices. As new types of applications were developed and new technologies emerged, the need for new application level protocols emerged.

The demands that led to development of application are :

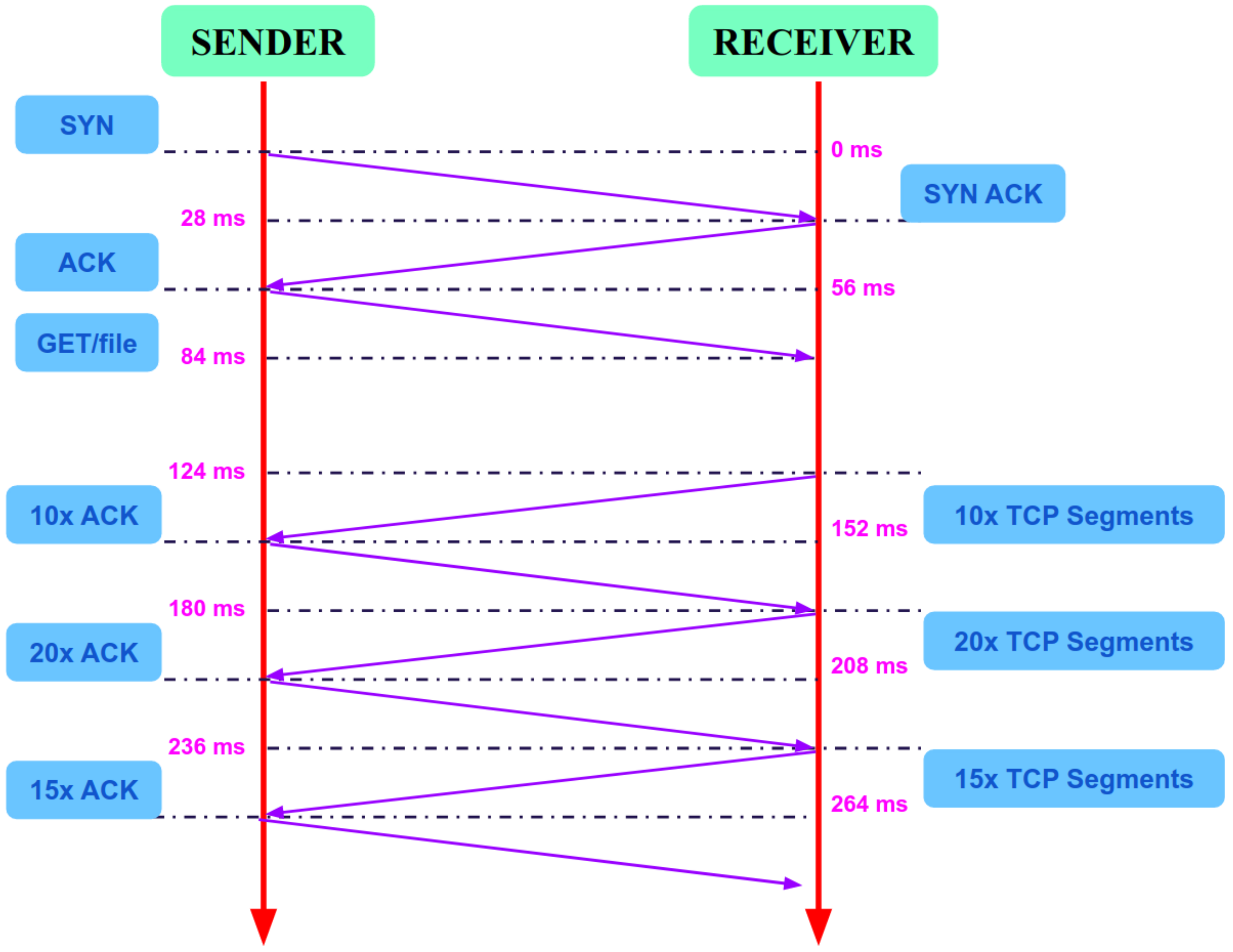

Fig. 1. Message Sharing and 3-way Handshake using SYN/ACK

- The growth of the Internet
- The need for standardization
- The need for security
- The need for new functionality

These requirements were led by the obsession of being connected to the internet every time. A new term Internet of Things (IoT) was introduced in 1999 by Kevin Ashton, this revolutionary idea was one of the technologies that was demanding all four of these requirements, so indirectly we can say that IoT led to massive development of the internet and protocols. How IoT is affected by choice of protocols and how IoT is still not standardized due to lack of standard protocols can be studied in [1] [6]. One reason for IoT not being stabilized is because of choice of stateless protocol for implementation. Since most applications of IoT relies on continuous being connected to the data source or the control source and high speed continuous or discrete data sharing and for that connection being stateless is not efficient, it needs to be stateful so that making new connections and sharing the context do not waste time.There is no purpose of a protocol if it is not being used for any application else it should not exist. On top of the transport layer there comes application level, the level which programmers in general use to interact with other machines. As this level is on top of the transport layer, programmers need not to worry about packet level interactions and how the connections are made and how the data will reach or has the data reached. For every networking task we rely on application level protocols and each application has its own specific demands and requirements, they can have their own custom or specific communication protocol to interact with other machines or they can use the generic application which is now being used everywhere which is HTTP/HTTPS. For visualization of layers of internet protocols refer Fig 3.1

## A. *Example of Application Level Protocol : HTTP/HTTPS*

An application protocol developed by Tim Berners-Lee and his team between 1989-1991. It was one of the first stable protocols which was then adapted to create the internet. The compliance specifications for HTTP are available at [1] [8]. Since there were only few options available at the moment http was chosen as standard protocol for data exchange for everything (data stored in memory). After some years, HTTP Secure was invented as a logical extension of HTTP, the main concern of HTTPS was to wrap the data being shared to protect the data being intercepted by someone else. Compliance with HTTPS is available at [1] [9]. The idea behind HTTPS is to wrap the HTTP socket with a Transport Level Security(TLS) encryption layer. The detailed research over SSL can be seen at [1] [1]. The idea behind TLS is to encrypt the data using a seed value which is in general a large sequence of characters, this key is shared to the client by the server every time a connection is made, and then the client also sends the key to the server to establish the two way encrypted connection, this process is called key exchange. Key exchange takes place in three steps: the server sends a certificate to the client and this certificate includes the server's public key. Then the client generates a random key and sends it to the server by encrypting it using the server's public key. Then the server decrypts the message using its private key. As this process completes, the socket is wrapped by a security layer, no one can intercept the data being shared, but if someone intercepts the connection while it was exchanging the keys, then there is a possibility of the connection being intercepted. Detailed Research on comparisons of exchange algorithm performance is available at [1] [10].

## IV. Problems With HTTP and its Logical Extensions

All the protocols have some virtues and deficiencies, no one is perfect, and the definition of perfect just changes with time. But the problem is HTTP hasn't overcome its one of deficiencies with the time and evolution of computers, some of them are :

On one hand HTTP solved many problems but on the other side it also started to get more buggy as the time passed, this is well researched in [1] [11]. Http was very slow and it was increasing latency, also Google deployed one temporary protocol that was like HTTP it was called SPDY, their aim was to reduce latency. One reason for HTTP becoming slow was high bloatedness. At that time and now also there is much data that HTTP response carries which is not even required. Google researched and then figured out how to reduce redundant data being sent in HTTP response, they reduced it and then it became fine, but with increasing demand of services and growth of the internet we need to further reduce it.

HTTP/HTTPS never provide any security mechanism for ensuring that the message reaches the other node when there occurs some network error or there is some error in network speed. Most existing protocols just leave everything to the user/programmer. This is a major issue which makes HTTP/HTTPS even more frustrating. There can be very heavy consequences which can occur if the correct response does not reach on the correct time.

Another problem of HTTP is its ambiguous End of File(EOF)/End of Request(EOR) character. HTTP uses \r\n as end of request as specified in [1] [8]. This seems not to be proper it is not appropriate to keep a combination of characters as end of a request the 0th character ie chr(0) of American Standard Code for Information Interchange(ASCII) is a more appropriate EOR character as it is also treated as EOF for files on the computers and also end of string in programs (C programs).

One problem is that it is stateless, each request is treated as a different request from another entity. This is because HTTP creates a new connection each time we make a request. It is like creating a new road each time we go to a place and then destroying it when we come back, this is a highly unoptimised approach and this also increases network load as a new request is made and all the key exchanges take place again and again which leads to network congestion. Thus this makes HTTP not suitable for high speed data streaming purposes.

One major problem with HTTP is its implementation, nowadays multi core CPUs are available but majority HTTP implementations are still using the single threaded asynchronous approach. This prevents the software from using the true potential of the machine. As a result the latency also increases.

As the statelessness makes the connection unstable, the protocol becomes more vulnerable to cyber attacks like Man-in-the-Middle (MitM) attacks, more vulnerabilities about HTTP can be studied in [1] [12]. A MitM attack is when an attacker intercepts communication between two parties, acting as a relay and eavesdropping on the communication.

There is a limited number of status codes, it doesn't allow programmers to create new custom application specific status codes for easy debugging and convenience. Programmes must have more freedom, if we would be able to create new status codes, it would allow us to create more complex applications and error handling in them easier.

HTTP/HTTPS and most of the remaining protocols lack plug and play implementations. The only thing it provides is a RFC for defining the packet and buffer for standardization, developers have to follow this implementation for integrity of application worldwide. No internal features are provided by the paradigm itself, we need built in features to reduce developer overhead of implementing each and every feature externally, also each feature should be implemented as extension in plug and play manner just like vim extensions and allow the user to create new features and include them during runtime.

### *A. **Compliance with Web-4 Concept***

From the time the internet was invented (1991), we have evolved at a rigorous rate. We have seen the concept of Intranet (the one only ARPANET) evolved into the Internet. From this transition to till date we have seen generations of the internet. We are currently between the 2nd and 3rd generation of it, yet we are not changing old methods of data exchanges even in the case of exponential growth of number of users in the past few years.

The generations of Internet are listed below :

**Generation-1 (Web1.X) :** The first generation was the first one that provided connectivity between places, it was mainly used for research and laboratory works, it was none other than ARPANET. Detailed study on arpanet evolution is available at [1] [13]. When the Internet was born, only static pages were hosted on a server and those pages were nothing except static html files into which all data was written. This form of internet was called Web1.0.

**Generation-2 (Web2.X) :** The demand of users increased, the demand of storing data on remote machines and then accessing it from anywhere was shaking the internet and developers. Then the internet pages with dynamic functionality and server-client to and from date transfers were implemented. The databases were designed to store client's data that enabled the transition from Web1.X to Web2.0. These days we are majorly using Web2.X in our daily life but Web3.X also exists parallelly. Study more at [1] [14].

**Generation-3 (Web3.X) :** This thing is something which not everyone knows, it existed in a type of parallel internet. It extends the concept of Web2.0 to the next level for ensuring data integrity and data security. Instead of storing data on a single centralized database, it distributes data on many distributed databases, all data units are stored in a ledger. A ledger has 3 parts : address - stores the unique identifier for the legger and is permanently associated with the ledger and the data stored on that ledger at the instance; data - the data; pointer to next ledger - this is the part that makes it so secure and decentralized, if the data in the the ledger is changed the address of the same ledger changes and the ledger chain breaks if any unauthorized transaction or operation takes place. Hence at the broken point of the chain we can detect the problem by checking the logs. This is mainly used in transaction systems. Study more at [1] [14].

**Generation-4 (Web4.x) :** This is the future of the web, it is an upcoming web, currently it does not exist but making transitions in small steps. It is still in a concept and visualization stage. Read more about Web4 at [1] [15]. One of the concepts of Web4 is high connectivity and totally cloud based computation. In this web version all computers/machines are stored in server/cloud and we can access our computer from anywhere we want to. Each single file, each single os directory, each single user info and login credentials will be stored in that machine only, we would only have a mini chip attached with I/O devices and the chip will be continuously in connection with our computer through the server. In the present there is an option in the computer boot menu for network booting, that allows to run Operating System (OS) stored on the server on the current machine and the machine directly communicates to the server and every action is reflected in the server in realtime. This type of OS is called server OS, server OS is the good to go option for features if we want to become totally digitized and want all things online. Web 4.0 and server OS also aim to make computers reachable to everyone, using server OS we can use high performance hardware on a normal computer virtually only the I/O is provided and received to the user computer. All the keystrokes will be sent to the server and each pixel rendered by the server will be received parallelly. For this a high connectivity devices and paradigms will be required. Read details in [1] [14].

Since the Web4.0 concept is so reliant on network technologies, we need to create new paradigms for making this beautiful technology come to existence. We need to have multithreaded server implementations instead of single threaded and asynchronous as they wont be able to serve millions of requests requesting live and parallel data. Second, they will need an implementation to be completely stateful which means the server has full context of the connection made by the client for serving the client efficiently without any undesirable delay in delivery and picking up of responses. Thus we need new protocols to be implemented for the future technologies.

## V. Proposed System

In this section, we will discuss Cypher's workflow, implementation, algorithms and major components with their functionalities. If anyone wants to contribute or wants to collaborate, the source code is available at https://github.com/P-Y-R-O-B-O-T/CYPHER_PROTOCOL .

### *A. Overview*

Cypher's main workflow and functionality depends on creating a new connection, encrypting and decrypting message to avoid interception from an outsider (currently this feature made temporary just to server on local network, instead we can wrap socket using SSL/TLS support provided by the language libraries that will enable more dynamic encryption), connection stabilization that provides relaxation to programmer because now the programmer is set free from managing connections, whenever a request is made it ensures that the request reaches the destination without any compromise, minimisation of errors : Cypher handles all the errors on its own the mechanism is that much capable that it can handle almost all of the errors that occur on itself internally.

### *B. Implementation*

Cypher utilizes encryption algorithms to encrypt and decrypt the message at nodes so that they are not intercepted in between the nodes or in the path that they follow to reach the destination. Currently for a temporary basis it uses Advanced Encryption Standard (AES)-128 bit algorithm for the security purposes which is hard coded into its server and client and can be changed by the user. AES is one of the strongest encryption algorithms and is used as a standard in many mechanisms that require high security. Detailed study on AES algorithms can be seen in [1] [16]. Average time required to crack an AES 128 bit message or password is roughly around 1 billion years using brute force where we check all the combinations of keys. Here we are using AES 128 bit encryption on both sides (server and client), there are 2 passwords on both client and server one is for encryption and one for the decryption. The encryption key in the server is the decryption key in the client, and the decryption key in the server is the encryption key in the client. Hence there is 2 way encryption AES-128 + AES-128 which still is not equivalent to AES 256 but still very close to AES-256 bit.

Cypher gives connection stabilization and to ensure efficient data transfers. It uses sockets in a stateful manner, the sockets are provided with a timeout limit, that limit defines the time up to which the connection is required to be stateful. If that limit is crossed in the server the client is disconnected or the connection/socket is closed. If the timeout on the client side is reached then the connection from the client side is closed resulting in the server also closing the connection from the other side as implemented in TCP/IP mechanism by default as discussed in [3] [2], after that if the client wants to make a new request then the connection is reestablished internally without any external intervention and process is continued as it was supposed to be continued. The parameter timeout allows us to implement this feature and we can also change its value to change the nature of Cypher's behavior. This is the solution to the problem described in [5] in the fourth point.

JSON format is used for interchange on information and for that purpose, we utilize internal json parsing libraries which enable efficient conversion from json to native data structures and vice versa. The reason behind using JSON is very simple : it is easy to parse, compact in nature, easy to read format, self describing also parsing is faster than Extensible Markup Language (XML). XML was a very popular format for data interchange until JSON arrived, See more detailed research over XML vs JSON in [1] [17].

Another thing to note is the EOR character used by Cypher. It uses the terminal character (chr(0)) used by low level languages like C/Cpp to terminate strings and sequences, this is also the character which is used for terminating communication between devices in computers. This character makes more sense than the \r\n\r\n and uses less space. This is the solution to the problem described in [5] in the third point.

Ensurance of message reaching its destination. This is one of the most important features of this mechanism. To ensure that the message reaches its destination it utilizes the default nature of TCP/IP as seen in [3]. Once the request is made, the request is sent to the server the server processes the request and sends the response, now this is the step where all the magic happens, now the client listens to the server for receiving the server, if there is any problem receiving the request the connection is closed by client and a new connection is established the request is made again on the new connection and the loop goes until the request is received properly. This is the solution to the problem described in [5] in the second point.

When it comes to performance the utilization of system resources is very important. Here we implemented the protocol in a multi threaded manner, this enables to serve multiple clients parallelly without any errors. This approach allows the use of all system resources to maximize performance. This is the solution to the problem mentioned in [5] in the fifth point. This allows the live stream without minimal delay, if the systems are implemented in a more distributed and load balanced manner then we could achieve even more minimized delay, detailed research at [1] [18].

Now coming to the algorithmic explanation. Up To now clearly seen the theoretical overview of the implementation. The algorithms will give a clear view of the implementation. Before jumping to the algorithms, first discussing the controlling parameters for the server and the client nature.

The server has the following parameters which can control the nature of the server : encryption_key - encrypts the response, decryption_key - decrypts the request from the client (NOTE : both decryption and encryption keys are static and being used to prevent intervention until SSL/TLS support is added to Cypher), request_handler - this is a method passed by user that handles the requests, this function is responsible for all requests being server as it has all the logic for serving the requests, recv_buffer : parameter is responsible deciding the size of chunk that will be received in one recv() call, transmission_buffer : it is responsible for deciding the size of chunk that will be transmitted to the other node in one send() call and the last one timeout : this specifies the number of seconds the connection will be stateful if it doesn't receive any response from the other node.

The client has the following parameters which can control the nature of the server : encryption_key - encrypts the response, decryption_key - decrypts the request from the client (NOTE : both decryption and encryption keys are static and being used to prevent intervention until SSL/TLS support is added to Cypher), response_handler : this handles all the response received from the server this has all the logic for data processing, offline_signal_processor : this is a very important parameter which is actually a user defined function which receives all the signals which are generated when the client faces any disconnection, all the actions which should be taken when client goes offline or faces some problems are taken by this function, online_signal_processor : user defined function is responsible for taking all the decisions when the client comes online state from offline state, recv_buffer : parameter is responsible deciding the size of chunk that will be received in one recv() call, transmission_buffer : it is responsible for deciding the size of chunk that will be transmitted to the other node in one send() call and the last one timeout : this is the limit up to which one recv() call can listen for response for receiving the data, if this time threshold passes the connection is re-established and the request is made again.

There are mainly 5 algorithms that contribute to this approach. There are algorithms for accepting connections, deleting connections, server algo for maintaining connections alive, connecting client to server and for making requests to server. The first 3 algorithms are for the server and the last two are for the client.

**Algorithm1 (Accepting New Connections) :**

For storing all the connection objects a dictionary or a map is maintained as CLIENTS. A loop is running continuously server_mainloop accepts all the connections, as soon as the connection is accepted, a connection object is created and it is added to the dictionary in the format - <key> : <object> where

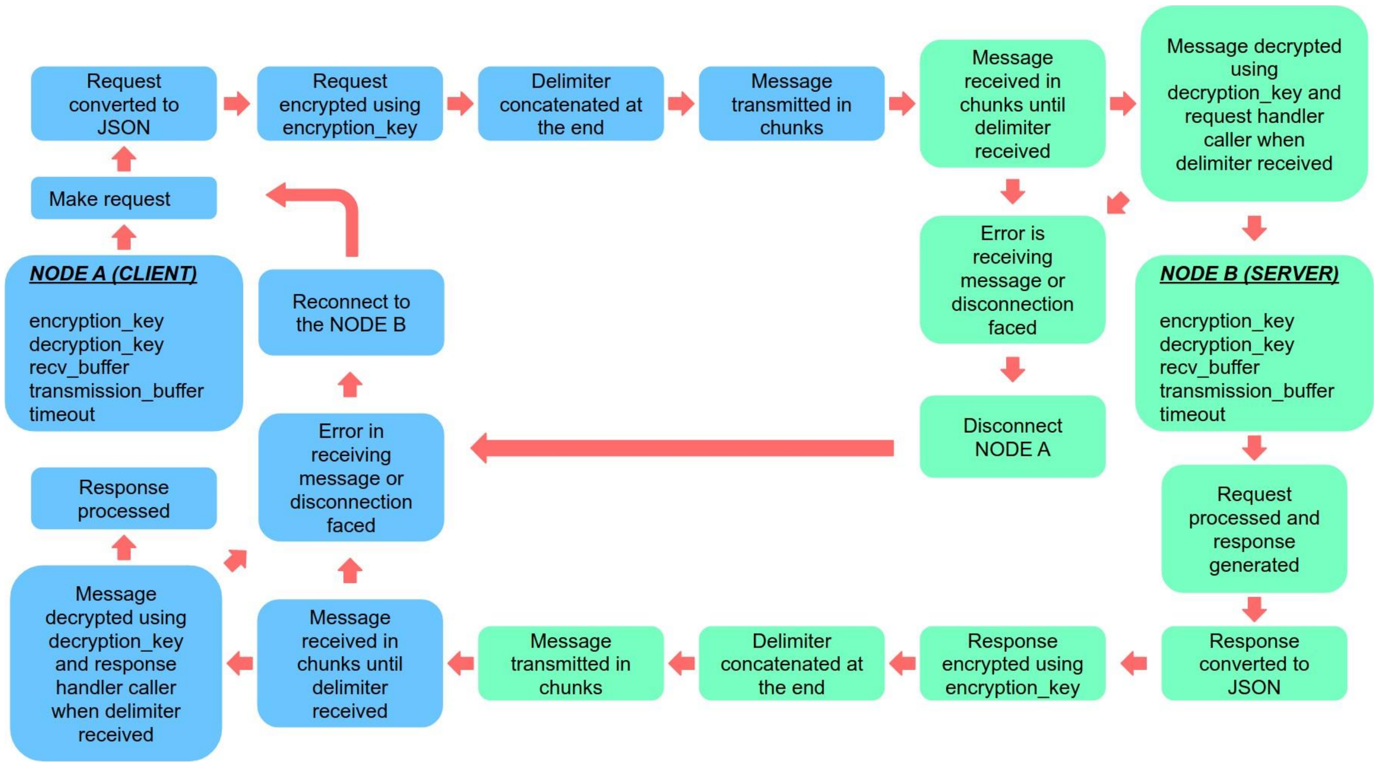
Request converted to JSON
Request encrypted using encryption_key
Delimiter concatenated at the end
Message transmitted in chunks
Message received in chunks until delimiter received
Message decrypted using decryption_key and request handler caller when delimiter received
Make request
NODE A (CLIENT)
encryption_key
decryption_key
recv_buffer
transmission_buffer
timeout
Reconnect to the NODE B
Error is receiving message or disconnection faced
NODE B (SERVER)
encryption_key
decryption_key
recv_buffer
transmission_buffer
timeout
Disconnect NODE A
Error in receiving message or disconnection faced
Response processed
Request processed and response generated
Message decrypted using decryption_key and response handler caller when delimiter received
Message received in chunks until delimiter received
Message transmitted in chunks
Delimiter concatenated at the end
Response encrypted using encryption_key
Response converted to JSON

Fig. 2. Workflow of Cypher Protocol

key is the tuple containing (<ip>, <port>) of the client that requested to establish the connection. The accepting loop runs until the SERVER_STATUS is set to false by the programmer or it runs infinitely, SERVER_STATUS is the flag variable for the server loop. There is a server_status variable which runs the connection accepting loop until that is set to true. A lock object LOCK is maintained for avoiding race conditions in a multithreaded environment. Server socket object is maintained as SERVER_SOCKET which accepts the connections. After the server loop exits the SERVER_SOCKET is closed and the server stops running.

```
define server_mainloop :
while SERVER_STATUS :
LOCK.acquire()
try :
CLIENTS[(<ip>, <port>)] <-
    SERVER_SOCKET.accept()
catch :
PRINT DEBUG TRACEBACK
LOCK.release()
SERVER_SOCKET.close()
```

Code Snippet 1. Connection Accepting Loop (Pseudocode/Algorithm)

**Algorithm 2 (Deleting Inactive connections) :**

As the connections become inactive, the connection objects put the (<ip>, <port>) tuple which is the address of the connection. Connection becomes inactive if it does not receive any request within the timeout limit TIMEOUT. A connection also becomes inactive if the connection is closed by the client side. Addresses of all connections are stored in a list or vector CLIENTS_TO_BE_DISCONNECTED. A loop also runs with the server_mainloop which is connection_object_destruction_loop which closes all the inactive connections by traversing through CLIENTS_TO_BE_DISCONNECTED, this loop runs until SERVER_STATUS become false and CLIENTS gets empty, both conditions should be true.

```
define
    connection_object_destruction_loop
     :
while SERVER_STATUS or CLIENTS != {} :
wait(1) //wait for 1 second
for connection_address in
    CONNECTIONS_TO_BE_DISCONNECTED :
destroy_connection_object(
    connection_address)
COLLECT GARBAGE
```

Code Snippet 2. Connection Object Destruction Loop (Pseudocode/Algorithm)

**Algorithm 3 (Maintaining Alive Connections) :**

This algorithm is implemented in connection object class, this plays the main role in implementation of Cypher as this provides the main functionality from the server side which is statefulness. There is a loop connection_loop running which listens to the requests that are coming from the client side. Two objects for encryption and decryption are also maintained - ENCRYPTION_OBJECT, DECRYPTION_OBJECT. There is also a method for processing requests. A server object is also maintained storing the reference of server as SERVER_OBJECT. One very important reference REQUEST_HANDLE_TRIGGER is maintained which is defined by the user and contains all the logic flow for handling the requests. This is the function that provides core server functionality. Also TRANSMISSION_BUFFER and RECV_BUFFERS that decide buffer sizes for transmission and receiving messages. A connection object also has a socket connection object CONNECTION which provides communication methods between the nodes. For each connection there is a CONNECTION_STATUS which ensures the statefulness of the connection until it is set to true. IP_PORT stores the address of the connection in the form (<ip>, <port>).

```
define process_request(client_resp) :
 decrypted_response <-
    DECRYPTION_OBJECT.decrypt(
    client_resp) // raise error if can
    't decrypt
 request_json <- convert_from_json(
    decrypted_response) // raise error
     if can't decrypt
 response_for_client <-
    REQUEST_HANDLE_OBJECT(request_json
    )
 response_json <- convert_to_json(
    responce_for_client)
 Client_resp[0] <- EMPTY STRING
 response_encrypted <-
    ENCRYPTION_OBJECT(responce_json)
 response <- bytes(responce_encrypted)+
    chr(0)
 for sub_bytes in response, step_size=
    TRANSMISSION_BUFFER :
 CONNECTION.send(sub_bytes)
 define connection_loop() :
 client_resp <- [EMPTY STRING]
 while CONNECTION_STATUS :
 try :
 temp_resp <- CONNECTION_RESP.recv(
    RECV_BUFFER)
 client_resp[0] += temp_resp
 if chr(0) in temp_resp :
 process_request(client_resp)
 elif temp_resp == EMPTY STRING : //
    there is some network problem
 break // breaking the loop will lead
    to closing and destruction of
    object
 except : // the connection is either
    closed by other side or there is
    some network error
 break //breaking will lead to closing
    and destruction of object
 SERVER_OBJECT.
    add_connection_to_be_destroyed(
    IP_PORT)
 //adding of IP_PORT to
    CLIENTS_TO_BE_DESTROYED will
    destroy the CONNECTION
```

Code Snippet 3. Maintaining Alive Connections (Pseudocode/Algorithm)

**Algorithm 4 and 5 (Connecting To Server and Making Request) :**

This algorithm is implemented on the client side for establishing connection with the server. On the client side, the variables CONNECTED as a flag to determine whether it is connected or not, CYPHER_STATUS to determine whether the user wants to stay connected to the server. The functions for signalizing the user about internal events like disconnection and reconnection as signalize_offline and signalize_online are used, whenever the client goes offline; signalize_offline is called, and whenever the client transitions from offline to online state the method signalize_online is called. CONNECTION is the socket connection that provides the methods to interact with the server. TIMEOUT is being used to set a threshold for socket recv() timeout limit. Along with all these the objects ENCRYPTION_OBJECT and DECRYPTION_OBJECT for encryption and decryption, RECV_BUFFER and TRANSMISSION_BUFFER variables are used for specifying the buffer sizes for receiving and transmission functions. The method connect is used to connect to the client and make_request is used to request a server. IP and PORT of the server are accepted from the user.

```
define connect() :
 if CONNECTED :
 CONNECTION.close(); del CONNECTION
 signalize_offline()
 while not CONNECTED :
 if CYPHER_STATUS :
 try :
 CONNECTION <- socket.socket()
 CONNECTION.connect(IP, PORT)
 CONNECTED <- True
 except :
 PASS //DO NOTHING
 else : break // break if user do not
    want to continue connection
    process
 wait(1) //wait for 1 second
 if CONNECTED :
 signalize_online()
 define make_request(path, operation,
    data, metadata) :
 data_ = {} //initialize a map or
    dictionary
 data_[``PATH''] <- path ; data_[``
    OPERATION''] <- operation
```

```
data_[``DATA''] <- data ; data_[``
    METADATA''] <- metadata
data_json <- convert_to_json(data_)
data_encrypted <- ENCRYPTION_OBJECT.
    encrypt(data_json)
request <- bytes(data_encrypted)+chr
    (0)
while CYPHER_STATUS :
try :
for sub_bytes in request, step_size=
    TRANSMISSION_BUFFER :
CONNECTION.send(sub_bytes)
except : connect() ; continue
server_resp <- [EMPTY STRING] ;
    error_occured_at_recieving <-
    False
while chr(0) not in server_resp[0] :
try :
temp_resp <- CONNECTION.recv(
    RECV_BUFFER)
if temp_resp != EMPTY STRING :
server_resp[0] += temp_resp
if temp_resp == EMPTY_STRING :
error_occured_at_recieving <- True ;
    break
except :
error_occured_at_recieving <- True ;
    break
if error_occured_at_recieving :
connect() ; continue
handle_responce(server_resp)
break
```

Code Snippet 4. Connecting To Server

## VI. Conclusion

We presented a new mechanism of data interchange, Cypher Protocol, to make computer networking easier and more efficient than it was in the past. Cypher connects to a node statefully in the network without giving any errors, instead it returns the status of types of errors that occur during establishment of new connections and lets the user decide what to do. It also ensures that the full potential of the TCP model is utilized, TCP ensures that the message reaches the destination properly, on top of that Cypher ensures that the message reaches the destination in the correct format without worrying about network issues and network quality. This mechanism does not give up to ensure a message is sent to the destination unless the user or developer tells or signalizes to do so. Using this approach we are able to stabilize the connection and the developer is more free to focus on logic rather than the connection management. Cypher also enables us to unlock the future possibilities for implementing the beautiful technology called Web4. The results demonstrate that Cypher can be used anywhere and very easily, it can also be used to create new frameworks for networking, especially IoT frameworks.

## References

[1] "Privacy Vulnerabilities in Encrypted HTTP Streams | SpringerLink." https://link.springer.com/chapter/10.1007/11767831_1 (accessed Feb. 24, 2023).

[2] "(PDF) Concept and Dimensions of Web 4.0." https://www.researchgate.net/publication/321366810_Concept_and_Dimensions_of_Web_40 (accessed Feb. 24, 2023).

[3] "(PDF) A Comparison Study on Key Exchange-Authentication protocol." https://www.researchgate.net/publication/46286685_A_Comparison_Study_on_Key_Exchange-Authentication_protocol (accessed Feb. 24, 2023).

[4] "'From ARPANET to Internet: A history of ARPA-sponsored computer network' by Janet Ellen Abbate." https://repository.upenn.edu/dissertations/AAI9503730/ (accessed Feb. 24, 2023).

[5] "A JSON/HTTP communication protocol to support the development of distributed cyber-physical systems | IEEE Conference Publication | IEEE Xplore." https://ieeexplore.ieee.org/document/8472084 (accessed Feb. 24, 2023).

[6] P. Wehner, C. Piberger, and D. Gohringer, "Using JSON to manage communication between services in the Internet of Things," May 2014, pp. 1–4. doi: 10.1109/ReCoSoC.2014.6861361

[7] T. He, J. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A stateless protocol for real-time communication in sensor networks," presented at the Proceedings - International Conference on Distributed Computing Systems, Jun. 2003, pp. 46–55. doi: 10.1109/ICDCS.2003.1203451.

[8] T. J. Hacker, B. D. Noble, and B. D. Athey, "The Effects of Systemic Packet Loss on Aggregate TCP Flows," Dec. 2002, pp. 7–7. doi: 10.1109/SC.2002.10029.

[9] "(PDF) A Survey on Application Layer Protocols for the Internet of Things." https://www.researchgate.net/publication/299535653_A_Survey_on_Application_Layer_Protocols_for_the_Internet_of_Things (accessed Feb. 24, 2023).

[10] N. Su, Y. Zhang, and M. Li, "Research on Data Encryption Standard Based on AES Algorithm in Internet of Things Environment," in 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Mar. 2019, pp. 2071–2075. doi: 10.1109/ITNEC.2019.8729488.

[11] A. Shieh, A. C. Myers, and E. G. Sirer, "A stateless approach to connection-oriented protocols," ACM Trans. Comput. Syst., vol. 26, no. 3, p. 8:1-8:50, Sep. 2008, doi: 10.1145/1394441.1394444

[12] E. Rescorla, "HTTP Over TLS," Internet Engineering Task Force, Request for Comments RFC 2818, May 2000. doi: 10.17487/RFC2818

[13] M. A. Abdillahi, U. Dossetov, and A. Saqib, "Performance evaluation of HTTP/2 in Modern Web and mobile Devices," American Journal of Engineering Research, 2017.