Journal of Theoretical and Applied Information Technology
30th September 2025. Vol.103. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ADAPTATION OF LOSS RECOVERY MECHANISMS FOR IMPROVING SCALABILITY AND QUALITY OF SERVICES IN IOT NETWORKS

ABOURRICHE SAMIRA[1], ZYANE ABDELLAH[2], GHAMMAZ ABDELILLAH[3]

[1,2] LAPSSII Laboratory, EST SAFI, Cadi Ayyad University, Safi, Morocco.

[3]L.S.E.E.T, FST-G Marrakech, Cadi Ayyad University,Marrakech, Morocco.

E-mail: [1]s.abourriche.ced@uca.ac.ma, [2]a.zyane@uca.ac.ma,[3] a.ghammaz@uca.ma

ABSTRACT

Over the past decade, the Internet has undergone significant transformations in various sectors such as healthcare, transportation, and the environment. These developments are built upon the TCP/IP model protocols, a four-layer architecture where the transport layer plays a crucial role in enabling Internet communication and managing Quality of Service (QoS). This evolution spans from the original ARPANET design to the emergence of Web 3.0, including Web 1.0 and Web 2.0. It has enabled the development of new communication paradigms such as the Internet of Things (IoT), which seeks to connect physical objects and devices across diverse domains within a unified infrastructure, supporting real-time monitoring and control [1].As a result, IoT has become a leading technology in multiple sectors, with connected devices projected to increase from 30.7 trillion in 2020 to 75.4 trillion by 2025 [2]. This massive number of connected things introduces numerous challenges, particularly in terms of QoS and scalability, which present obstacles for IoT and, more specifically, Machine-to-Machine (M2M) networks.The main objective of this paper is to provide a comprehensive contribution toward implementing mechanisms for autonomous scalability management and Quality of Service (QoS). The approach integrates Transport Layer loss recovery protocols from the TCP/IP model within the middleware layer of IoT networks.. The outcome of this work proposes a new architecture for the existing IoTScal approach, incorporating additional components to simulate loss recovery mechanisms. This enhancement boosts the success rate of e-Health traffic from 95% to 99.99%, delivering a clear and significant improvement over the existing reference approach.

Keywords: *Quality Of Service,Internet Of Things,Scalability,M2M Networks,Loss Recovery Mechanisms*

1. INTRODUCTION

Since its creation in the 1960s, the Internet has relied on a standard and universal model that defines a set of communication protocols for network interaction.This model is TCP/IP where TCP and IP play an important roles.In this model, the Transport Layer plays a vital role in Internet communications. It guarantees end-to-end communication between applications and provides essential services like flow control, congestion control, and reliable transfer through loss recovery mechanisms in order to satisfy functional and non-functional requirements. Internet known over time a huge progress,evolving beyond human connections to include objects,giving birth to the IoT.Coined by Kevin Ashton in 1999 Co-founder of MIT's Automatic Identification Center-[1],IoT connects anything, anywhere, any time, enhancing daily life with minimal human intervention.Moreover, IoT is a collection of objects equipped with a number of sensors and actuators that allows to collect and transfer information about the surroundings via Internet in order to understand and control it [2],leading to applications in areas like smart cities, industries, and E Health. The connected devices of the IoT, overall, and Machine-to-Machine communications, specifically, provide significant opportunities but also present challenges due to the large number of users and their connected devices [3]. This requires the system to adapt to various requests and the vast amount of information exchanged among all involved parts (users, applications...), while ensuring a maintained acceptable level of QoS.The number of works that suggest a solution to the problem of scalability in

IoT networks is limited.Among the most famous approaches dealing the scalability issue are:NDN(Name Data Networking) that allows to enhance the scale of IoT-Cloud [4],DINAS(DIstributed NAming Service) that represents a novel approach to naming and service discovery in resource limited wireless sensor networks [5] ,SDNSNA(Secure DNS name auto-configuration) which was designed to automatically assign domain names for IPv6-based IoT applications within the DNS system [6]. In contrast , the different proposed methods aim to tackle the scalability challenge by focusing on how to name, protect, and authenticate the variety of devices in an IoT system. Still, none of these methods seem to offer a solution that can effectively handle all incoming requests from identified users or devices without impacting specific quality measurements, such as Round-Trip Time (RTT).To solve the problem mentioned previously ,the proposed approach at the work [7] seeks to satisfy the maximum number of incoming requests as well as to respect the QoS metrics.Inside this solution named IoTScal-H ,a part of the traffic are rejected :5% for E-health ,50% for industrial and 80% for transportation.Our contribution is centred on the integration of loss recovery mechanisms derived from the simple Automatic Repeat reQuest (ARQ) family into this approach.By incorporating these ARQ mechanisms,our solution can identify and retransmit lost data in order to guarantee the reliability of the overall IoT system.Our work focuses on improving scalability and QoS in IoT networks by the integration of these mechanisms at the Transport Layer ,assuming this layer as the core communication framewrok.While our approach is designed to improve scalability in specific IoT domains—such as e-health, industrial systems, and transportation—it also makes a fundamental assumption: the reliability of the underlying physical and link layers is not the focus of this study. Instead, we concentrate on the Transport Layer's ability to mitigate data loss and preserve QoS indicators such as Round-Trip Time (RTT). The scope of our work is limited to simulation-based evaluation and does not cover large-scale real-world deployments.The structure of this document is as follows: the second section will provide a detailed overview of the Transport Layer, emphasizing its importance for proper data routing and the loss recovery mechanisms, particularly those from the ARQ family. The third section will present the tests and results related to the impact of these mechanisms on QoS. The fourth section will discuss our upcoming contribution, which aims to enhance the IoTScal-H approach by integrating loss recovery mechanisms to improve scalability and QoS. Finally, the sixth section will conclude the paper.

2. TRANSPORT LAYER,GENERALITIES AND PROTOCOL MECHANISMS.

2.1 General Description of The Transport Layer

Situated between the network and application layers ,the transport layer is a vital part of internet communications, acting as a bridge between lower TCP/IP layers and upper layers. It provides end-to-end transport services between applications, offering two types: non-connection-oriented and unreliable and reliable and connection-oriented. The layer also provides a logical communication channel between applications on different hosts using transport protocols like TCP(Transmission Control Protocol) and UDP(User Datagram Protocol). It enhances reliability, confidentiality, and data integrity, providing uninsured network layer functionalities.

- UDP protocol [8] :is one of the transport protocol.It's unreliable and non connection-oriented protocol with multiplexing/demultiplexing service for transport data between application and networks layers ,ensuring also a fast delivery service.That's why many applications like DNS use it.In addition,many applications are better suited to UDP for the following reasons : Firstly,UDP isn't based to a congestion control mechanism and consequently,the data transmission doesn't take a lot of time .Furthermore,the real time applications which often require minimal debit,doesn't wish to delay segment transmission excessively and can tolerate data loss.Secondly ,UDP doesn't allow to do a three way handshake between sender and receiver.Third reason ,UDP gives no idea about the connection status between systems and finally ,UDP has a segment of a minimum size equal 8 bytes at the header level shown in Figure 1.Among applications using UDP,we find;NFS,SNMP,RIP,DNS as well as some recent applications like «Internet phone» and «streaming multimedia».

Journal of Theoretical and Applied Information Technology
30th September 2025. Vol.103. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

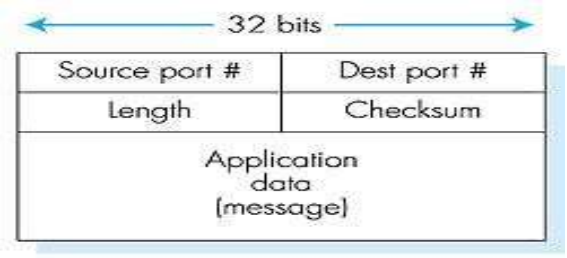

Figure 1.UDP segment structure

- TCP protocol [9] :is a reliable transport protocol,connection-oriented and it relies to many mechanisms (Errors detection,cumulative acknowledgements,sequence number and Timer).Moreover ,TCP runs only in the end systems and provides a connection in bidirectional mode(full-duplex) whose data can flow in both directions ,respecting a minimal segment size (MSS) ,knowing that the receiver and the sender has a buffer to manage the flow control.TCP is connection oriented means that before an application process begins to send data to the other,the both must do a three way handshake .TCP implements a reliable data transfer service under unreliable IP service.It's based on the following mechanisms :cumulative acknowledgements,segment pipeline,retransmissions triggered by timer expiration or by receipt of a duplicate acquittal.TCP provides also a flow control service to the applications for prevent the possibility of the sender overflowing the receiver buffer without forgetting congestion control.The below picture Figure 2 shows the TCP segment structure.

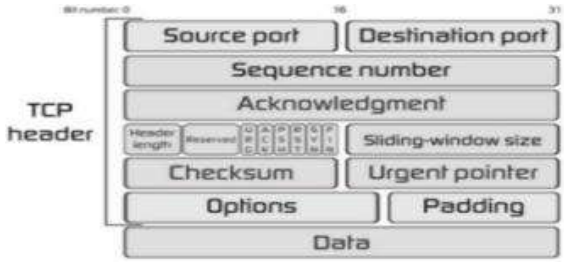

Figure 2.TCP segment structure

2.2 Protocol Mechanisms and Functions

- Connection Establishment Mechanism [10]: Also known as the signaling mechanism, this is a three-phase process used by TCP before data transmission. It operates in client/server mode, where the client initiates the connection and the server acknowledges it, establishing communication.
- Data Segmentation and Sequencing: TCP involves segmenting and sequencing data during transmission. The sender cuts data from its send buffer into segments, each with a maximum size defined by the MSS, and assigns sequence numbers to ensure reliability. These segments are encapsulated in datagrams at the network layer and stored in the receiver's buffer.
- Flow Control [11]: TCP provides flow control, preventing the sender from overwhelming the receiver's buffer. It ensures speed matching, adjusting the sending rate to the receiver's capacity by limiting transmission based on available buffer space.
- Congestion Control [12]: TCP uses the AIMD algorithm to control congestion by increasing bandwidth gradually and reducing it when congestion occurs. This mechanism adjusts the sender's transmission rate by increasing bandwidth when the network is clear and reducing it when packet loss or congestion is detected.
- Transfer Reliability [13]: Reliable data transfer is supported by protocols at the Data-link, Transport, or Application layers. These protocols create a reliable channel, ensuring data transmission without errors or loss. TCP achieves this through a reliable data transfer protocol (RDT) and a system of acknowledgements and sequence numbers, allowing the sender and receiver to confirm successful data delivery.

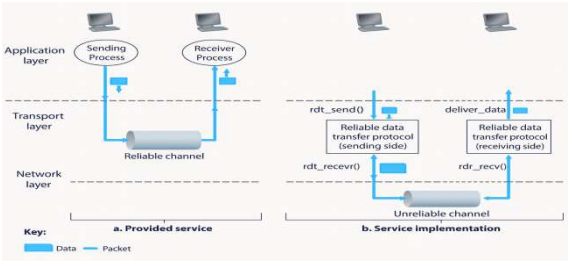

Figure 3.Reliable Data Transfer

3. PRESENTATION OF LOSS RECOVERY MECHANISMS

A reliable data protocol (rdt) is crucial for IT networks, ensuring data delivery to recipients. Designing a dependable transfer protocol involves using complex protocols, resulting in a faultless and reliable transfer protocol, unlike unreliable ones.

- Rdt 1.0 [13]: is a protocol that ensures reliable and orderly data transfer without error bits or packet loss. The sender sends data in the underlying channel, while the receiver reads it at the end. No feedback is required from the receiver side.
- Rdt 2.0 and its secondaires [13]:into this protocol ,the channel is considered orderly and unreliable ,i.e ,it can damage bits of the packet without any loss of the packet when it propagates.The errors are detected through error control.If this control is correct ,the receiver explicitly indicates the correct receipt of the packet by sending a positif ACK,otherwise,the receiver explicitly indicates the incorrect reception of the packet through a negative ACK (NACK) and the sender will retransmit the wrong packet.The derivatives of this protocol are : rdt 2.1 which allows to add alternating sequence number (0 or 1) as well as rdt 2.2 which has the same rdt2.1 functionalities but it uses only ACKs.
- Rdt 3.0 [13]:In this protocol, the channel can lose packets (data or ACKs). Even with included mechanisms like checksums, sequence numbers, positive ACKs, and retransmissions, these may not be enough. To recover lost packets, a tool called a timer needs to be integrated.

3.1 Classification

To ensure transfer reliability, there are two main classes of mechanisms:

- Reactive Mechanisms (ARQ Simple): Also known as Automatic Repeat reQuest (ARQ), these mechanisms ensure orderly packet transmission without loss or error through retransmissions. Types include Stop and Go, Stop and Wait, Go-Back-N, and Selective Repeat.
- Proactive Mechanisms (ARQ Hybrid): ARQ Hybrid mechanisms add redundancy packets to allow receivers to recover lost packets without requiring retransmissions. They include HARQ type I, HARQ type II, and HARQ type III.

3.2 Reactive Mechanisms (ARQ Simple)

The ARQ protocol, developed by Van Duüren in 1943, is a loss/error recovery algorithm designed for reliable data transmission. Commissioned in the Netherlands in 1947, it uses positive and negative acknowledgements along with a timer to ensure efficient transmission. The performance of ARQ depends on the reliability of the round-trip channel and the Round-Trip Time (RTT) between sender and receiver, making it a widely used solution in IT networks. Key elements of ARQ include CRC, positive and negative acknowledgements, a timer, and sequence numbers. Figure 4 illustrates the basic elements of the ARQ protocol.

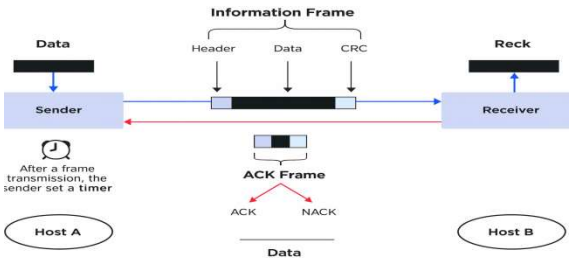

Figure 4.Basics elements of an ARQ protocol

There are a various types of simple ARQ mechanisms, these include:

- Stop and Go: The most basic ARQ form, where a packet is sent and the sender waits for either a positive acknowledgement (ACK) or a negative acknowledgement (NACK) from the receiver, who checks packet integrity using checksum calculations.
- Stop and Wait: In this mechanism, the transmitter sends a packet and waits for an acknowledgement before sending the next one. This can limit hardware capabilities and bandwidth. To allow multiple packets to be transmitted without waiting for each acknowledgement, the sequence number range must be expanded, and buffers added.

To allow the transmitter to send multiple packets in-flight without waiting for each acknowledgement and to optimize bandwidth usage, the sequence number range must be expanded, and buffers must be added at both the sender and receiver. These adjustments depend on how the transfer protocol handles lost and corrupted packets. The simple ARQ family includes two key pipeline protocols:

- Go-Back-N (GBN): The sender can transmit multiple packets, but the total number must not exceed a specified maximum N. The receiver acknowledges all packets up to N and generates duplicate acknowledgements for missing packets, only acknowledging the expected sequence numbers.
- Selective Repeat (SR): This protocol minimizes unnecessary retransmissions by only requesting retransmission of lost or corrupted packets.The receiver acknowledges and buffers correctly received packets, improving efficiency by ensuring that only lost or corrupted packets are retransmitted.

3.3 Proactive Mechanisms (ARQ Hybrid)

- HARQ type I:known as Chase Combining (CC), is a fundamental form of HARQ, combining ARQ with FEC encoding, requiring small receiver memory and suitable for low-noise channels.
- HARQ type II:Incremental Redundancy (IR) is a HARQ that improves decoding chances and outperforms Chase Combining, particularly for fast-fading channels, if initial transmission errors persist.
- HARQ type III: is an extension of HARQ II, encoding each code word with independent information and parity bits, requiring larger receiver memory than CC for decodable individual or combined transmissions.

3.4 Principles of Simples ARQ Mechanisms

In the previous section ,an overview of reactive mechanisms (Basic ARQ) was made.Indeed ,the Basic ARQ family, consisting of four mechanisms: Stop and Go, Stop and Wait, Go Back N, and Selective Repeat, ensures orderly service transmission without packet loss or error by detecting errors and retransmissions, each described in pseudo code for easy understanding.

3.4.1 Stop and go mechanism

The simplest form of the simple ARQ family operates as follows:

- Principle: A packet is sent only after receiving an acknowledgement (ACK) for the previous one.
- Assumptions: No errors occur on acknowledgements, and no packets are lost.
- Description
  - Sender (A)
    - Sends a packet to the receiver and waits for either an ACK or a NACK.
    - Sends the next packet only after receiving an ACK.
  - Receiver (B)
    - Upon receiving the packet, the receiver checks its integrity by verifying the packet's correctness, usually through a checksum or similar error-detection mechanism.. If the packet is error-free:
      - B sends an ACK; otherwise,
      - B sends a NACK.
- Errors processed: Corrupted data.
- Stop and Go Algorithm

The two algorithms below illustrate the detailed pseudo code of the ARQ Stop and Go protocol.The first algorithm depicted in Table 1 represents the sender's process, detailing how it sends a packet and waits for an acknowledgement (ACK) or negative acknowledgement (NACK). The Table 2 outlines the receiver's process, demonstrating how it checks the integrity of each packet and responds accordingly by sending either an ACK or a NACK. These algorithms ensure reliable data transmission by handling errors and retransmissions in an orderly manner.

Table 1: Stop and Go sending procedure.

Stop and Go sending procedure
Packet Type : int type int sum Message payload Emission (Message data) Packet pdata, pack pdata.payload ← data pdata.sum ← calculate_sum (pdata) repeat send_packet(pdata) wait_packet (pack) until pack.type = ACK

Table 2: Stop and Go receiving procedure.

```
Stop and Go receiving procedure
Packet Type :
        int type
        int sum
        Message payload
Reception (Message data)
        Packet pdata, pack
        Boolean exit ← FALSE
        repeat
                wait_paquet(pdata)
                if check_sum(pdata) = TRUE then
                        data ← pdata.payload
                        pack.type ← ACK
                        exit ← TRUE
                else
                        pack.type ← NACK
                        send_packet(pack)
        until exit = TRUE
```

3.4.2 Stop and wait mechanism

This mechanism is an improvement over the Stop and Go protocol, featuring the following characteristics:

- Principle: It combines the 'Stop and Go' technique, requiring the sender to retransmit a packet if the desired acknowledgement (ACK) is not received within a specified time-out.
- Assumptions: The system assumes no multiplexing, user data segmentation, and follows a modulo 2 approach for sequence number calculation, ensuring no errors or losses on acknowledgements.
- Description
  - Sender (A):A field is added to the packet to contain a sequence number (0 or 1). If there is data to send, the sender creates a packet with a specific sequence number and transmits it. The sender then waits for an acknowledgement for a specified time:
    - If an ACK with the correct sequence number is received before the time-out expires, the sender sends the next packet with the following sequence number.
    - If no ACK is received, or if the wrong ACK is received, the sender retransmits the same packet.
  - Receiver (B):The receiver also maintains an expected sequence number for incoming packets. Upon receiving a packet, the receiver checks its integrity:
    - If the packet is valid and its sequence number matches the expected one, the receiver extracts the data and sends an ACK with that sequence number.
    - If the sequence number doesn't match or the packet is corrupted, the receiver rejects the packet and either sends an ACK for the last correctly received packet or does not send an ACK at all, making the sender believe the packet was lost
- Advantages
  - Simplicity in transmission and reception procedures.
  - Low memory cost since only one packet needs to be stored at both the sender and receiver.
- Disadvantages
  - The sender stays inactive until it receives an acknowledgement, which may hinder performance.
- Errors Processed:Corrupted Data and lost Data
- Stop and Wait Algorithm

The Stop and Wait algorithm will be demonstrated through pseudo-code , with one representing the transmitting side detailed in Table 3 and the other illustrating the receiving station shown in Table 4

Journal of Theoretical and Applied Information Technology
30th September 2025. Vol.103. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 3: Stop and Wait sending procedure.

```
Stop and Wait sending procedure
Packet Type :
        int type
        int sum
        Message payload
Emission (Message data)
        Packet pdata,
        pack int numSeq ← 0
        Boolean exit ← FALSE
        pdata.numPack ← numSeq
        pdata.payload ← data
        pdata.sum ← calculate_sum(pdata)
        repeat
                send_packet(pdata)
                wait(time)
                if receive_packet(pack) = FALSE then
                        Continue
                else
                        // Is this a reliable ACK?
                        if pack.numPack = numSeq then
                                exit ← TRUE
                                numSeq ← increment(numSeq)
                        else
                                exit ← FALSE
        until exit = TRUE
```

Table 4: Stop and Wait receiving procedure.

```
Stop and Wait receiving procedure
  Packet
     Type :
        int type
        int sum
        Message payload
Reception (Message data)
        Packet pdata,pack
        int expected_numSeq ← 0
        Boolean exit← FALSE
        repeat
                receive_packet(pdata)
                if check_sum(pdata) = FALSE or pdata.numPack =/=
                expected_numSeq
                then
                        Continue
                // New packet or copy ?
                data ← pdata.payload
                exit ← TRUE
                pack.numPack ← expected_numSeq Send_packet(pack)
                expected_numSeq ← increment(expected_numSeq)
        until exit = TRUE
```

3.4.3 Go back n mechanism

Go Back N is a sliding window protocol, commonly used for error and loss control in both the Link and Transport Layers. This ARQ mechanism is characterized by the following:

- Principle:The transmitter can send multiple packets within a window without waiting for an acknowledgement (ACK) for each individual packet, unlike Stop and Wait.
- Assumptions
  - The sequence number calculation is based on the number of packets in the window.
  - No errors or losses occur on acknowledgements.
- Remarks

The sender manages a window that consists of four parts (illustrated in Figure. 5):

  - [0-base-1]: Packets that have already been transmitted and acknowledged.
  - [base, nextseqnum-1]: Packets that have been transmitted but not yet acknowledged.
  - [nextseqnum, base+N-1]: These are the packets that the sender is permitted to transmit immediately without waiting for an acknowledgement from the receiver.
  - numseq > base+N: Untransmitted and unacknowledged packets within the window.
  - Base: The sequence number of the oldest packet that has been sent but has not yet received an acknowledgement.
  - Nextseqnum: The smallest unused sequence number.

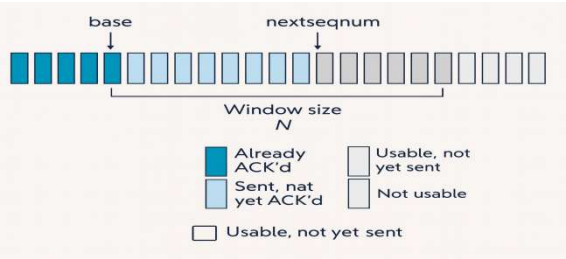

Figure 5. Go Back N sender's window[14]

- Description
  - Sender (A): The GBN sender handles three types of events:
    - Invocation from the top:
      - If there is data to send, A first checks if the window is full (i.e., there are N unacknowledged packets).

- If the window is not full, A creates a packet, sends it, starts the timer, and updates its variables.
- If the window is full, A refuses new data, indicating the window is full

▪ Receipt of an ACK:

- An ACK for a packet with sequence number N is considered cumulative, meaning that all packets with sequence numbers less than or equal to N have been received correctly.
- If base = nextseqnum, A stops the timer.
- Otherwise, A starts the timer.

▪ Timer expiration:

- If a time-out occurs, the sender (A) retransmits all packets that have been sent but have not yet been acknowledged.

o Receiver (B): The GBN receiver operates as follows:

▪ If a packet with sequence number N is received in order and without errors, B sends an ACK for the packet and delivers the packet data to the top layer.

▪ If the packet is out of order or corrupted, B rejects all subsequent packets and retransmits an ACK for the last correctly received packet.

- Errors Processed: Corrupted and lost data.
- Advantages
  - o GBN reduces the waiting time between transmissions seen in Stop and Wait by allowing continuous transmission within a window (anticipation window).
  - o It improves transmission efficiency and optimizes link usage.
- Disadvantages
  - o GBN can lead to unnecessary retransmissions for out-of-sequence packets, increasing processing and bandwidth usage.
- Go Back N algorithm

The Go Back N algorithm will be illustrated through pseudo-code showing in Table 5 and 6, with one version depicting the sender's operations and the other showing the receiver's processes.

Table 5: Go Back N sending procedure.

```
Go Back N sending procedure
Packet Type :
        int type
        int sum
        Message payload
Emission (Message data)
        Packet pack
        Packet[MAX] tab_packet //Using a Packet type array to store packets to send
        int base ← 0 //The sequence number of the oldest unacknowledged packet
        int nextseqnum ← 0 //the smallest sequence not in use
        int numSeq ← 0
        int N ← 2^m-1          //The size of the window
        Boolean exit ← FALSE
        int i ← 0
        repeat nextseqnum <= base + N then
                tab_packet[numSeq].payload ← data
                tab_packet[numSeq].numPack ← numSeq
                tab_packet[numSeq].sum ←
                        calculate_sum(tab_paquet[numSeq])
                send_packet(tab_paquet[nextseqnum])
                nextseqnum ← nextseqnum + 1
                numSeq ← increment(numSeq)
        end repeat
        wait(time)
        if received_packet(pack) = TRUE then
                if pack.numPack >= base and pack.numPack <= base + N then
                        base ← increment(pack.numPack)
                        if base = nextseqnum then
                                exit ← TRUE numSeq ← 0
                        else
                                continue
                else
                        i← base
                repeat  i=/= nextseqnum
                        send_packet(tab_paquet[i]) i← increment(i)
                end repeat
exit ← FALSE
until exit = TRUE
```

Table 6: Go Back N receiving procedure

Go Back N receiving procedure

```
Packet Type :
        int type
        int sum
        Message payload
Reception(Message data)
        Packet pdata ,pack
        int numseq_attendu ← 0
        Boolean ← FALSE
repeat
        while expected_numSeq <= 2^m-1 //The same value of M used
        at the sender
        receive_packet(pdata)
        if check_sum(pdata) = TRUE then
                if (pdata.numPack = expected_numseq) then
                        data ← pdata.payload
                        pack.numPack ← expected_numSeq
                        expected_numSeq ←
                        increment(expected_numSeq)
                        send_packet(pack) //This ACK will be
                        transmitted in two cases : If packet contains
                        errors or it's hors sequence
end repeat
exit ← TRUE
Expected_numSeq← 0
until exit = TRUE
```

3.4.4 Selective repeat mechanism

This mechanism represents a specific implementation of the ARQ protocol and utilizes a sliding window at both the sender and receiver. Its features are as follows:

- Principle: The Selective Repeat (SR) protocol optimizes retransmissions by sending only the lost or corrupted packets to the receiver. The receiver acknowledges and buffers all received packets, passing the data to the application layer.
- Assumptions
  - The sequence number calculation depends on the window size, ensuring that the sender and receiver correctly manage and track packets within the defined window range.
  - No errors or losses in acknowledgements.
- Remarks: Both sender and receiver operate with a window of the same size, as illustrated in the accompanying diagram showing the window views at both ends.
- Description:
  - Sender (A) : The SR sender (A) addresses three types of events:
    - Data received from above: The sender checks if the data's sequence number is within the window. If so, the packet is sent. Otherwise, data is rejected, indicating a full window.
    - Timer expiration: Each packet has its own timer. If a packet's timer expires, only that packet is retransmitted.
    - Receiving an ACK: On receiving an ACK, the sender checks if it falls within the window. If it does, the sender marks the associated packet as acknowledged and adjusts the window base if needed.
  - Receiver: The SR receiver (B) responds to three types of events:
    - Reception of a packet: If the packet's sequence number is within the expected range, the data is delivered to the application layer, and the window is advanced.Out-of-order packets are buffered, and a selective acknowledgement (ACK) is sent to indicate which packets have been correctly received.
    - Acknowledgement of previously received packets: An ACK is sent for packets within the previous range to ensure all data is correctly processed.
    - Ignoring non-relevant packets: Packets outside the expected sequence are ignored.
- Errors Processed: Corrupted and lost data.
- Advantages and Disadvantages: The Selective Repeat protocol is more efficient than other ARQ mechanisms by reducing unnecessary retransmissions. However, it is more complex to implement
- Selective Repeat algorithm

The pseudo-code outlines the operations of the Selective Repeat (SR) protocol, providing a clear representation of how the receiver and sender

handle data transfer and error recovery. This detailed explanation is broken into two sections, each showing their respective roles in Table 7 and Table 8.

Table 7: Selective Repeat receiving procedure

```
Selective Repeat receiving procedure
Type Packet:
        Integer numPack
        Integer sum
        Message payload
Reception (Message data)
        Packet pack, pdata
        Packet[MAX] packet_array
        Integer rcv_base ← 0
        Integer N ← 2^m - 1
        Integer i ← 0
        Boolean exit ← FALSE
Repeat
        while rcv_base <= N do
        receive_packet(pdata)
              if check_sum(pdata) = TRUE then
                 if pdata.numPack >= rcv_base and pdata.numPack <= rcv_base
        + N then
                  if pdata.numPack = rcv_base then
                           pack.numPack ← rcv_base
                 send_packet(pack)
                 data ← pdata.payload
                           rcv_base ← rcv_base + 1
                        for j = 0 to i do
                           if packet_array[j].numPack = rcv_base then
                              rcv_base ← packet_array[j].numPack + 1
                              data ← packet_array[j].payload
                           else
                              break // Exit the loop
                           end if
                        end for
                     else
                        packet_array[i] ← pdata
                        i ← i + 1
                        pack.numPack ← packet_array[i].numPack
                        send_packet(pack)
                     end if
                  else
                     if pdata.numPack >= rcv_base - N and pdata.numPack <=
        rcv_base - 1 then
                        pack.numPack ← pdata.numPack
                        send_packet(pack)
                     else
                        continue
                     end if
                  end if
               end if
            end while
            exit ← TRUE
            rcv_base ← 0
        until exit = TRUE
```

Table 8: Selective Repeat receiving procedure

```
Selective Repeat sending procedure
Type Packet:
        Integer numPack
        Integer sum
        Message payload
Emission (Message data)
        Packet pack
        Packet[MAX] packet_array
        Integer base ← 0
        Integer nextseqnum ← 0
        Integer numSeq ← 0
        Integer i ← 0
        Integer m ← 0
        Integer N ← 2^m - 1
        Packet[N] received_packets_array
        Boolean exit ← FALSE
        while nextseqnum <= base + N do
           packet_array[nextseqnum].payload ← data
           packet_array[nextseqnum].numPack ← numSeq
           packet_array[nextseqnum].sum ←
        calculate_sum(packet_array[nextseqnum])
           send_packet(packet_array[nextseqnum])
           nextseqnum ← nextseqnum + 1
           numSeq ← increment(numSeq) // Increment based on modulo N
        end while
        //Label1
        repeat
           wait(time)
           if receive_packet(pack) = TRUE then
              if pack.numPack >= base and pack.numPack <= base + N then
                 if pack.numPack = base then
                    if base = N then
                       exit ← TRUE
                       numSeq ← 0
                    else
                       base ← pack.numPack + 1
                    end if
                    for j = 0 to i do
                       if received_packets_array[j].numPack = base then
                          if packet_array[j].numPack = N then
                             exit ← TRUE
                             numSeq ← 0
                          else
                             base ← received_packets_array[j].numPack + 1
                          end if
                       else
                          go to Label_1
                       end if
                    end for
                 else
                    received_packets_array[i] ← pack
                    i ← i + 1
                 end if
              else
                 i ← base
                 send_packet(packet_array[i])
                 exit ← FALSE
              end if
           end if
        until exit = TRUE
```

Journal of Theoretical and Applied Information Technology
30th September 2025. Vol.103. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.4.5 Illustrations of loss recovery mechanisms

The figures below figure 6,figure 7,figure 8 and figure 9 explains how different loss recovery mechanisms handle packet loss and ensure reliable data transmission. Stop-and-Wait ARQ waits for an acknowledgement before sending the next packet, ensuring reliability but reducing efficiency. Go-Back-N ARQ improves throughput by allowing multiple packets to be sent without waiting for acknowledgements, but if a packet is lost, all subsequent packets are retransmitted. Selective Repeat ARQ retransmits only the lost packet, which is more efficient but requires handling out-of-order packets, making it more complex. Each mechanism balances reliability, complexity, and efficiency in data transmission.

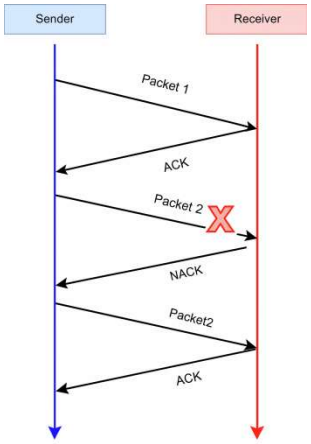

Figure 6.Principle of Stop and Go mechanism

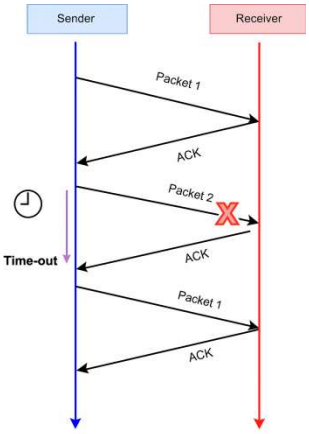

Figure 7.Principle of Stop and Wait mechanism

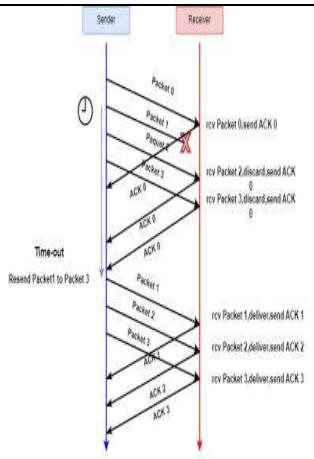

Figure 8.Principle of GBN mechanism

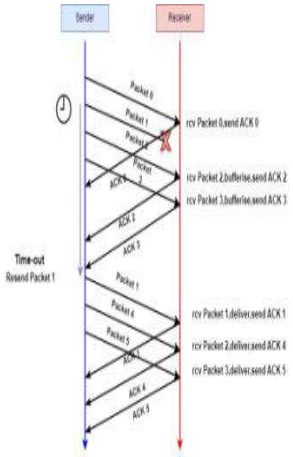

Figure 9.Principle of Selective Repeat mechanism

4. IMPLEMENTATION OF LOSS RECOVERY MECHANISMS,TESTS AND RESULTS

4.1 Implementation of Simple ARQ Loss Recovery Mechanisms

The implementation of simple ARQ loss recovery mechanisms is structured as a modular software system, designed to encapsulate the sender and receiver behaviours within discrete units. This modular approach aligns with principles of modular programming, which organizes code into manageable, logical units, facilitating code reuse, maintenance, and enhancement. The overall design is monolithic, meaning it is developed as a single cohesive unit but consists of several interconnected modules.

4.1.1 The core modules in the implementation

- udp.h and udp.c: These modules define and implement the functions necessary to

emulate a UDP socket. They handle the low-level details of socket communication, providing the foundational layer for packet transmission and reception.

- libktf_util.h and libktf_util.c: These utility modules offer additional functionalities such as checksum calculation, sequence number incrementation, and timer management. They support the core ARQ mechanisms by ensuring packet integrity, proper sequencing, and timeout handling.
- libktf.h and libktf.c: This pair of modules provides an API for communication between the application layer and other software components. They offer a standardized interface for accessing and interacting with the ARQ mechanisms.
- libktf_internal.h: This header file includes various data structures required for the internal operations of the ARQ mechanisms. It defines structures for packets, sequence numbers, timers, and other essential components.
- libktf_X.h and libktf_X.c: These modules are specific to each ARQ mechanism (where X represents the specific mechanism, such as Stop and Go). They contain the functions for transmitting and receiving packets according to the rules of each ARQ protocol. For example, for the Stop and Go mechanism, the modules are named libktf_stopgo.h and libktf_stopgo.c, containing functions tailored to the Stop and Go protocol's operations.

4.1.2 Detailed module functions

- udp.h and udp.c: Implement the creation, binding, sending, and receiving of UDP packets. They ensure that packets are transmitted and received over a network connection, providing basic network communication services.
- libktf_util.h and libktf_util.c: Offer utility functions such as:
  - **Checksum Calculation**: Verifies the integrity of transmitted data.
  - **Sequence Number Incrementation**: Manages the ordering of packets.
  - **Timer Management**: Handles retransmission time-outs to ensure reliable data delivery.
- libktf.h and libktf.c: Provide high-level API functions that facilitate interaction between the application and ARQ modules, abstracting the details of the ARQ mechanisms from the application code.
- libktf_internal.h: Defines the internal data structures used throughout the ARQ modules, ensuring consistent and efficient data management.
- libktf_X.h and libktf_X.c: Implement the specific ARQ protocol logic:
  - **Transmission Functions**: Handle packet creation, sending, and managing sequence numbers based on the ARQ protocol rules.
  - **Reception Functions**: Manage incoming packets, handle acknowledgements, and process retransmissions as needed according to the specific ARQ mechanism.

4.2 Tests and Results

To validate the efficiency and impact of the implemented ARQ mechanisms on throughput and transmission time, tests were conducted in a controlled environment at LAAS-CNRS, where network disruptions were simulated. The results of these tests are presented as graphs and curves, which illustrate the performance metrics of each mechanism under varying network conditions. These visualizations provide insights into how well each ARQ protocol handles disruptions and their effects on overall network performance.

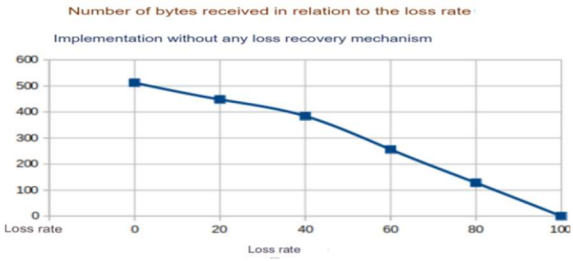

Figure 10 .The efficiency of an implementation without any ARQ mechanisms

Journal of Theoretical and Applied Information Technology
30th September 2025. Vol.103. No.18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

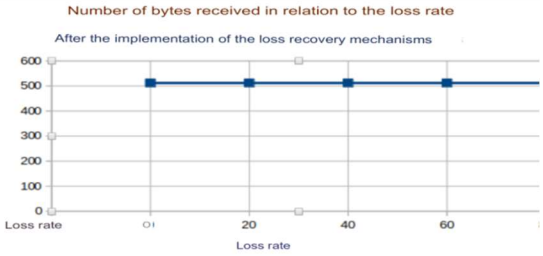

Figure 11 .The efficiency of an implementation with an ARQ mechanisms

Based on the graphs above figure 10 and figure 11,we note that the number of received bytes without any ARQ mechanisms decreases as long as the loss rate increase despite the existence of the UDP Transport Protocol ,on the other hand,the number of bytes keeps the original value after using the loss recovery mechanisms regardless of the loss rate applied.Indeed,The presence of a mechanism for recovery losses in a communication channel ,contributes to the reliability of data transfers. Moreover,and through the graphs below figure 12,figure 13 and figure 14,we can observe that as long as the loss rate increases ,the debit decreases and the delay increases after testing all ARQ mechanisms implemented .

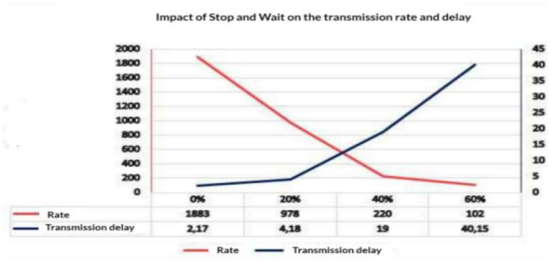

Figure 12 .Impact of Stop and Wait on the transmission rate and delay

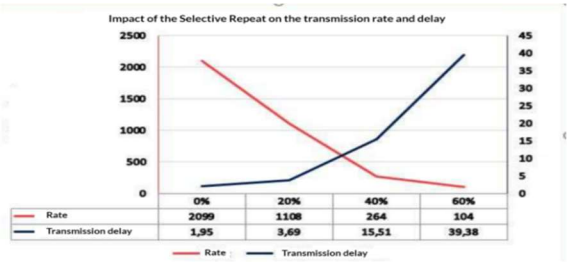

Figure 13 .Impact of GBN on the transmission rate and delay

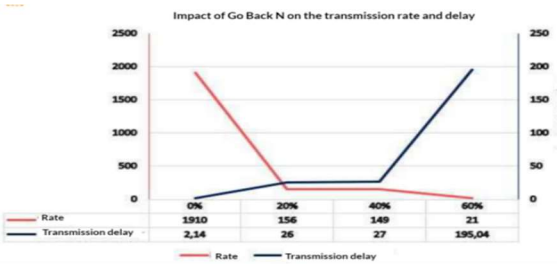

Figure 14 .Impact of Selective Repeat on the transmission rate and delay

The paragraph explains the comparison between three Automatic Repeat reQuest (ARQ) mechanisms based on their throughput performance.

- Selective Repeat (SR)
  - Achieves the highest throughput among the three mechanisms due to its ability to retransmit only the lost packets.
  - Ideal for applications that cannot tolerate delays and require high reliability in data transmission.
  - SR is more efficient but also more complex to implement than the other mechanisms. It offers a more selective and precise approach to packet loss recovery, minimizing wasted bandwidth.
- Stop-and-Wait (SW):
  - Occupies the second rank in terms of throughput performance.
  - Simpler in its implementation compared to SR, making it suitable for applications that require reliable communication but can tolerate moderate performance.
  - SW waits for an acknowledgement after each packet, which slows down the process but reduces the complexity compared to SR.
- Go-Back-N (GBN):
  - Comes in last regarding throughput due to its less efficient handling of errors. When a packet is lost or corrupted, GBN retransmits the lost packet along

with all subsequent packets, even if they were correctly received.

- Suitable for applications where delays are tolerable and reliability is still a priority.
- GBN sacrifices efficiency to ensure reliable data transmission, making it more appropriate for systems where occasional retransmissions are acceptable.

5. DISCUSSION

The section emphasizes the importance of loss recovery protocols in computer networks for reliable data transfer, a fundamental aspect of Quality of Service (QoS). Experimental results from a two-virtual-machine local network testbed illustrate the performance of various loss recovery mechanisms—including Stop-and-Go, Stop-and-Wait, Go-Back-N (GBN), and Selective Repeat (SR)—across a range of operating scenarios.These tests offer insights into how these mechanisms affect reliability within a basic client/server setup, with relevance extending beyond traditional client/server applications to include radio-based systems. The Internet of Things (IoT) is expected to transform inter-object communication by 2025 , yet it faces significant challenges—one of the most critical being scalability. In this context, scalability refers to a device's capacity to adapt to environmental changes while continuing to meet user requirements [16]—. Our findings validate our initial goal of improving scalability and Quality of Service (QoS) in IoT networks through the integration of loss recovery mechanisms from the ARQ family into the IoTScal solution. By evaluating key metrics such as Round-Trip Time (RTT) and success rate under varying traffic loads, we demonstrate that Transport Layer recovery techniques can deliver significant performance gains. This work offers a novel contribution, as most existing IoT scalability strategies focus on aspects like naming or security, often overlooking the potential of loss recovery to manage traffic surges while maintaining QoS. The proposed approach is designed to maximize the number of requests the oneM2M platform can process while maintaining key QoS metrics such as RTT and resource availability. It incorporates an autonomous computing framework based on the MAPE-K loop to reduce human intervention, operating at the middleware layer of the ETSI M2M standard. By combining scalability-focused and QoS-focused mechanisms, the IoTScal-H [7] strategy efficiently manages diverse traffic types—including E-Health, industrial, and transportation—ensuring compliance with SLAs. This is achieved through the seamless integration of middleware components, autonomous management, and cloud-enabled load balancing, which together sustain platform performance and middleware efficiency.

To ensure all requests are handled effectively without loss or rejection, the implementation of the simple ARQ mechanisms will enhance scalability and optimize delay and throughput, giving rise to a new IoTScal solution called IoTScal-LR. This solution will integrate reactive loss recovery mechanisms—Stop and Wait, Go Back N, and Selective Repeat—into the Middle-ware layer of the IoT architecture defined by the ETSI standard. These mechanisms, which are derived from TCP protocol functionalities, will be adapted to comply with the ETSI architecture standards and rules. Each mechanism is tailored to meet the specific QoS requirements of different types of traffic: Selective Repeat for E-Health, Go Back N for Transportation, and Stop and Wait for Industrial applications.This work also builds upon and extends existing research on IoT scalability solutions, including NDN [4], DINAS [5], SDNSNA [6], and the IoTScal-H approach [7]. While these studies address important aspects such as naming, service discovery, and request handling, they do not directly integrate Transport Layer loss recovery mechanisms into scalability-oriented frameworks. The proposed IoTScal-LR model addresses this gap by incorporating ARQ-based mechanisms, thereby establishing a valuable connection to previous work while contributing a novel perspective.

Strengths: The approach demonstrates measurable improvements in RTT and success rate, offers a novel scalability enhancement focused on the Transport Layer, applies adaptive ARQ mapping to traffic categories, and achieves middleware-level integration without requiring changes to lower layers.

Limitations: The evaluation is restricted to simulations and small-scale testbeds, without examining energy consumption or security implications, and retransmission overhead may become significant under extreme traffic loads.

Opportunities: The middleware positioning allows integration without disrupting lower protocol layers, and the adaptive ARQ selection is rare in IoT middleware research. Moreover, combining cloud-based load balancing with loss recovery

presents a promising hybrid strategy for addressing both scalability and QoS challenges.

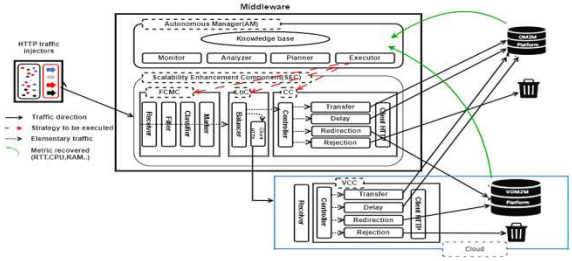

Figure 15 . The architecture of the IoTScal-H solution[7]

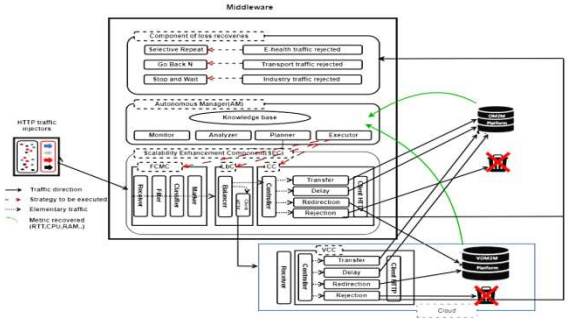

Figure 16 . The new architecture IoTScal-LR solution

6. CONCLUSION

In this paper ,we have discussed about Transport layer and its role to guarantee an end-to-end transport service between different applications.Also,we have presented the protocol mechanisms and functions integrated within the Transport layer such as congestion control,transport reliability etc ...Moreover,a presentation of loss recovery mechanisms was addressed and especially those related to the ARQ simples family which named reactive mechanisms.These mechanisms included the Stop and Go,Stop and Wait ,Go Back N and Selective Repeat. Each mechanism can be used for a specific type of applications in order to insure the transfer reliability and consequently satisfying a part of the QoS.But with the appearance of the new paradigms such as IoT ,this future technology offers new applications and services linking the physical and virtual worlds,which leads to increase the number of problems and challenges that represent a major obstacle for implementation and adoption of the IoT in our daily life.Among the most important challenges,we found the scalability.To overcome this issue ,a proposed approach named IoTScal approach aims to satisfy a large number of requests into IoT systems without impacting QoS.Furthermore ,the reliability of transfer which represents one of the criteria of the QoS and through the addition of the loss recovery mechanisms treated in this paper, will make it possible to improve even more the QoS within the IoT networks.By integrating these mechanisms, the proposed IoTScal-LR solution aims to further improve QoS by ensuring reliable data transfer while maintaining scalability. This contribution is particularly relevant in the current IoT landscape, where the rapid growth of connected devices intensifies network load and heightens the risk of data loss. The findings provide a foundation for designing middleware-level solutions capable of balancing high request throughput with reliability in diverse IoT domains such as E-Health, Industry, and TransportationAs this work presents a proposal, the declared purpose is addressed through the design of the IoTScal-LR approach, which integrates ARQ-based loss recovery mechanisms to enhance scalability and QoS. While the concept is fully defined and aligned with the stated objectives, the complete validation will be carried out in future work through implementation and experimental evaluation.In summary, this study contributes: (1) a Transport Layer–oriented approach to IoT scalability, (2) adaptive ARQ mapping to domain-specific traffic types, and (3) a middleware-level strategy that integrates seamlessly into ETSI IoT architectures

REFERENCES:

[1] Madakam, R. Ramaswamy, and S. Tripathi, “Internet of things (IoT): A literature review,” J. Comput. Commun., vol. 03, no. 05, pp. 164–173, 2015

[2] Pawan Kumar , Satvir Singh , Lavish Kansal,”A Comprehensive Review on Internet of Things (IoT)”,ISBN: 978-93-91355-11-1,pp 222-227.

[3] L. Farhan, R. Kharel, O. Kaiwartya, M. Quiroz-Castellanos, A. Alissa, and M.Abdulsalam, “A concise review on internet of things (IoT) -problems, challenges and opportunities,” in 2018 11th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP), 2018.

[4] S. Han and H. Woo, “NDN-based Pub / Sub System for Scalable IoT Cloud,” 2016 IEEE International Conference on Cloud Computing

Technology and Science (CloudCom), pp. 488–491, Dec 2016.

[5] M. Amoretti, O. Alphand, G. Ferrari, F. Rousseau, and A. Duda, "DINAS: A lightweight and efficient distributed naming service for all-IP wireless sensor networks," IEEE Internet Things J., vol. 4, no. 3, pp. 670–684, 2017.

[6] Lee, K., Kang, H., Jeong, J. P., Kim, H., & Park, J.-S. (2016). Secure DNS name auto configuration for IPv6 internet-of-things devices. 2016 International Conference on Information and Communication Technology Convergence (ICTC). doi:10.1109/ictc.2016.7763534

[7] A. Zyane, M. N. Bahiri, and A. Ghammaz, "IoTScal-H : Hybrid monitoring solution based on cloud computing for autonomic middleware-level scalability management within IoT systems and different SLA traffic requirements," Int. J. Commun. Syst., vol. 33, no. 14, 2020

[8] J. F. Kurose and K. W. Ross, Computer Networking : A Top-down Approach. Boston, Pearson, 2017. p224

[9] J. F. Kurose and K. W. Ross, Computer Networking : A Top-down Approach. Boston, Pearson, 2017. p240

[10] J. F. Kurose and K. W. Ross, Computer Networking : A Top-down Approach. Boston, Pearson, 2017. p272

[11] J. F. Kurose and K. W. Ross, Computer Networking : A Top-down Approach. Boston, Pearson, 2017. p292

[12] Lorincz, Josip, et al. "A Comprehensive Overview of TCP Congestion Control in 5G Networks: Research Challenges and Future Perspectives." Sensors, vol. 21, no. 13, 1 Jan. 2021, p. 4510, www.mdpi.com/1424-8220/21/13/4510/htm, 10.3390/s21134510. Accessed 25 Aug. 2021.

[13] J. F. Kurose and K. W. Ross, Computer Networking : A Top-down Approach. Boston, Pearson, 2017. p 244-252.

[14] J. F. Kurose and K. W. Ross, Computer Networking : A Top-down Approach. Boston, Pearson, 2017. p256-265.

[15] Manna, P., & Das, R. (2021). Scalability in Internet of Things: Techniques, Challenges and Solutions. International Journal for Research in Engineering Application & Management (IJREAM), 07(Issue-01), 2454–9150. https://doi.org/10.35291/2454- 9150.2021.0175.