Rechnernetze WiSe 2024/25
(Prof. Dr. Martin Becke)
INTERNET-DRAFT 1.1

RNP Group 01
HAW Hamburg
September 2025

Network Messaging Protocol

RNP Group 01

Abstract

This document outlines the Network Messaging Protocol (NMP), designed for message passing within a small network.
Users form a partially meshed topology that allows sending, receiving, and forwarding of text messages. Communication is ensured by the message structures and format defined in this specification.
NMP operates on the application layer as an intradomain routing protocol that uses distance vector routing to manage the user topology.
(Fischer, Leon)

Changelog

Version 1.0 [2024-12-15] released initial version **(RNP Group 01)**.

Version 1.1 [2025-01-04] [RFC 01] Edited 3.5 Checksum, added Changelog **(Fischer, Leon)**.

Contents

1. Introduction

1.1. Contributors

Contributors are indicated in bold and placed in brackets following a topic, paragraph, sentence, graphic, or table.
Example: **(Küppers, Tom)** or **(Group Diagrams)**

RNP Group 01

- Althiab, Hadi
- German, Michael
- Mjawaz, Yousef
- Shahrour, Samer
- Fischer, Leon
- Konoplev, Stepan
- Lööck, Kjell
- Treffenfeldt Montoya, Valentina
- Haag, Thomas
- Henske, Lars
- Rathje, Torben
- Küppers, Tom
- Kohrs, Janice

Group Diagrams

- Althiab, Hadi
- German, Michael
- Mjawaz, Yousef
- Shahrour, Samer

Group Chair

- Lööck, Kjell
- Treffenfeldt Montoya, Valentina

1.2. Intended use of the protocol

This protocol serves as a framework for messaging software designed for use in small networks with a single broadcast domain. It provides all necessary specifications for implementing the features in any programming language.
(Fischer, Leon)

1.3. Terminology

The words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "OPTIONAL" are used as defined in BCP 14, RFC 2119 [1]
(Fischer, Leon)

1.4. Glossary

ID	name	description
00	network messaging protocol	The specification that defines the chat tool
01	chat tool	The application that implements the network messaging protocol
02	environment	The local network in the HAW in which the chat tool MUST work
03	user	An instance of the chat tool running on a node in the environment
04	node	A workstation in the environment with a predefined IP
05	topology	The partially meshed network structure that is created by connecting users
06	reachability	Relation between two users. User A is reachable from user B, if B has a routing table entry for A.
07	connection	Relation between two users. User A is connected to user B, if A is reachable from B with a hop count of 1 and B is reachable from A with a hop count of 1.
08	activity	A user is active if the user has at least one connection.

Table 1: Glossary

(Shahrour, Samer; Fischer, Leon)

2. Requirements

The following requirements ensure a working implementation of the protocol:

Feature Requirements

- **RF-01** A user MUST be able to generate a list of all active (reachable) users of the current topology at any time.
- **RF-02** A user MUST be able to send text messages to any selected (reachable) user.
- **RF-03** A user MUST be able to receive text messages from any user.
- **RF-04** A user SHOULD be sure that a message sent reaches the selected user.
- **RF-05** A user MUST be able to establish a connection to join the network.
- **RF-06** A user MUST be able to go offline, terminating all their network connections.

Technical Requirements

- **RT-01** Connections between users MUST be implemented using TCP.
- **RT-02** The addressing of messages MUST be implemented with IPv4.
- **RT-03** The network environment MUST be a partially meshed topology within a broadcast domain.
- **RT-04** The target IP for establishing a connection of a new user SHOULD be specified.
- **RT-05** User connections MUST be point-to-point.
- **RT-06** A user MUST be able to act as both server and client.
- **RT-07** A new user MUST be able to start without a specified target IP.
- **RT-08** Routing MUST be implemented using Distance Vector Routing.
- **RT-09** Split Horizon SHOULD be implemented to prevent count-to-infinity problems.
- **RT-10** TTL SHOULD be implemented to prevent (endless) routing loops.
- **RT-11** The maximum number of users MUST be 16.
- **RT-12** Multiple users MAY be able to operate simultaneously on a single node, each uniquely identifiable within the network.
- **RT-13** All sent messages MUST be in JSON format.

(RNP Group 01)

3. Specification

3.1. Routing Protocol and Routed Protocol

- In the network, the **Routing Protocol** defines the rules for exchanging routing information between nodes, ensuring reachability and updating routes dynamically. This is implemented using Distance Vector Routing (DVR), which periodically shares routing tables to maintain network topology.

- The **Routed Protocol** refers to the actual data (e.g., text messages) that is transmitted between nodes following the paths established by the routing protocol. These routed messages are forwarded based on the routing table entries.

Thus, the Routing Protocol enables route discovery and maintenance, while the Routed Protocol delivers user-specific payloads using the established routes.
(Küppers, Tom)

3.2. Connection

3.2.1. TCP and IPv4

The Network Messaging Protocol uses **TCP** and **IPv4** to ensure reliable communication in a partially meshed network.

- **TCP**, as defined in STD 7, Transmission Control Protocol [2], provides ordered, error-checked data transmission, ensuring that all messages, including text and routing updates, arrive intact and in the correct order. The protocol operates on the **standard port: 8080**.
  After each message, the connection is closed, enabling a bidirectional communication pattern by establishing a new connection for subsequent messages.

- **IPv4**, as defined in Internet Protocol [3], assigns unique 32-bit addresses to nodes for routing. Connections are initiated by sending **routing information** to a target address, dynamically integrating new users into the network. In the implementation, the four octets of an IPv4 address are converted into the variable type: long. For example, the IPv4 address 192.168.0.1 is converted using the formula:

$$\text{long\_value} = (\text{octet}_1 \cdot 256^3) + (\text{octet}_2 \cdot 256^2) + (\text{octet}_3 \cdot 256^1) + (\text{octet}_4 \cdot 256^0)$$

Applying the formula to the IPv4 address 192.168.0.1:

$$\text{long\_value} = (192 \cdot 256^3) + (168 \cdot 256^2) + (0 \cdot 256^1) + (1 \cdot 256^0) = 3,232,235,521$$

This long_value represents the IPv4 address in a efficient format.

(Küppers, Tom)

3.2.2. Handling Connections

Connections MUST be handled as follows: Every time a message needs to be sent, a new socket connection MUST be created. When the message is sent, the socket connection MUST be closed.
To Consider: When sending an ACK or NACK message, the socket connection used to receive the TEXT message MUST be closed. A new socket connection MUST be established to send the ACK or NACK.
(Althiab, Hadi; Mjawaz, Yousef; Shahrour, Samer)

3.2.3. Establishing a connection

Following requirement **RT-04** ("The target IP for establishing a connection of a new user should be specified"), User1 joins the topology by sending his ROUTING_INFORMATION (with only himself as a destination) to the target IP of User2.
This is sufficient to establish a connection, as User2 will then propagate the connectivity to User1 by including User1 in its own ROUTING_INFORMATION, which is subsequently shared with all other users.
Thus, no specific connection message is required.

A user MUST always be able to establish a new connection to any other user (except themselves) by sending their ROUTING_INFORMATION to the desired destination.
(Fischer, Leon)

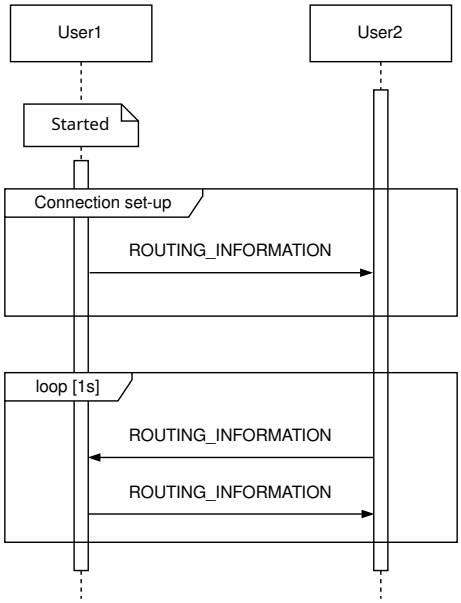

Figure 1: Establishing a connection **(Group Diagrams)**

3.2.4. Disconnection

There are two types of disconnection:

1. A user intentionally disconnects by terminating the client.
2. A user unintentionally disconnects due to an error.

No differentiation is required for handling disconnections, as both types result in the same outcome. As illustrated in the example (Figure 2): The disconnected User1 will fail to send its ROUTING_INFORMATION within the required time frame. After the timeout, User2 will remove User1 from its routing table, informing all other users that User1 is no longer reachable through User2.
(Fischer, Leon)

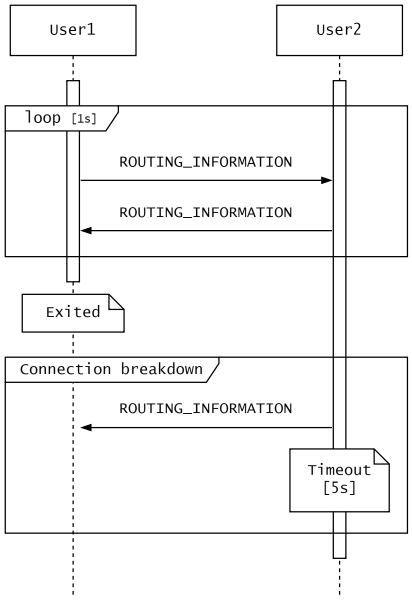

Figure 2: Disconnecting **(Group Diagrams)**

3.3. Messaging

3.3.1. Message structure

A message MUST consist of two fields: **header** and **payload**. The header has a fixed size and structure to enable efficient parsing, while the payload's size and content vary depending on the message type and user-provided data.

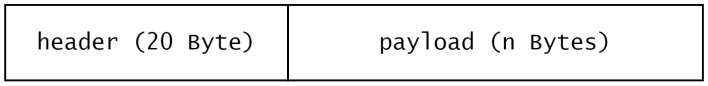

Figure 3: message structure

(Fischer, Leon)

3.3.2. Header structure

All message types share the same header structure as shown in this table:

Parameter	Data type	Description	Size in Byte
msg_type	u8	The type of message sent: 0 = TEXT, 1 = ROUTING_INFORMATION, 2 = ACKNOWLEDGE, 3 = NOT_ACKNOWLEDGE	1
ttl	u8	The message's time to live start = 16, end = 0	1
size	u16	Size of the message body (only the size of the TEXT field)	2
sender_ip	u32	IPv4 address of the User, who originally sent the message	4
destination_ip	u32	IPv4 address of the User, who should unpack the message's body	4
checksum	long	A number used to check the integrity of the data	8

Table 2: header structure **(Haag, Thomas)**

3.3.3. Message types

TEXT: The text that a user sends to another user is referred to as a text message. A text message requires a response: either ACKNOWLEDGE or NOT_ACKNOWLEDGE. Each text message has a unique ID, allowing the sender to match it with incoming responses. Until a valid response is received, the corresponding MSG_ID must remain blocked. The payload consists of two fields:
(Fischer, Leon)

1. **TEXT**
   contains the actual chat message from the user.
   size: char[size]
   The maximum size of the char-Array is 256 Characters and MUST be checked before sending the message. The size SHOULD also be checked after receiving a text-message.

2. **MSG_ID**
   Message ID; local ids from 0. . . 255.
   MSG_ID must be forwarded with the message unchanged and must be included in the Acknowledge or Not Acknowledge of said Message.

   (Kohrs, Janice; Rathje, Torben)

   The use of the MSG_ID is left to the Implementer of the Protocol. Possible Implementations:

   - TCP Send Window:
     - No combination with IP: u8 allows for 255 unacknowledged messages.
     - With combination with IP: allows for 255 unacknowledged messages per active user. Max $16 \times 255 = 4.080$ unacknowledged messages.
   - Ignored: Client has to wait (maybe indefinitely).
     - No combination with IP: Only 1 message sent.
     - With combination with IP: Only 1 per reachable user.

From a global point of view: if there are 3 different implementations in the Network, one using TCP Send Window with combination, one using TCP Send Window without combination, and one ignoring the IDs and not combining with IP, The Network would have Max of $4.080 + 255 + 1 = 4.336$ globally unacknowledged messages.

size: u8

(Shahrour, Samer)

ROUTING_INFORMATION: This message type ensures the implementation of distance vector routing. Every user must periodically send his ROUTING_INFORMATION to each of their direct neighbors.
Is serves as a heartbeat, propagating the user's availability to others. If the ROUTING_INFORMATION from a direct neighbor is not received within the defined time frame, that neighbor must be removed from the routing table.
The payload contains the routing table of the user with the following columns:
(Fischer, Leon)

1. **DESTINATION**
   The target host to which the data packet should be delivered
   size: u32
2. **GATEWAY**
   The IP address of the next router along the path to the destination
   size: u32
3. **HOP_COUNT**
   The number of hops the packet must pass through to reach the destination
   size: u8

(Kohrs, Janice; Rathje, Torben)

ACKNOWLEDGE: This message type is one of two possible responses to a text message. It is the positive feedback that the destination user has received the text message by the source user. The user can safely reuse the MSG_ID after receiving ACKNOWLEDGE.
The payload contains the following field:
(Fischer, Leon)

1. **MSG_ID**
   The message ID of the text message whose arrival has been acknowledged
   size: u8

(Kohrs, Janice; Rathje, Torben)

NOT_ACKNOWLEDGE: This message type is one of two possible responses to a text message. It is the negative feedback that

- the destination is not reachable
- ttl expired
- crc failed

The user can safely reuse the MSG_ID after receiving NOT_ACKNOWLEDGE.
The payload consists of two fields:
(Fischer, Leon)

1. **MSG_ID**
   The message ID of the text message whose arrival has been NOT acknowledged
   size: u8

2. **ERROR_CODE**
   type of error: 0: not reachable; 1: ttl expired; 2 crc failed;
   3 msg format invalid; 4 payload size exceeded; 5 other
   size: u8

   (Kohrs, Janice; Rathje, Torben)

3.3.4. Sending a Text Message

User2 should be able to forward Messages with a different destination. After checking if the destination of the message is reachable, User2 sends the message to the listed Gateway in the routing table, in this case User4, who is the destination.

All message types, excluding ROUTING_INFORMATION, are forwarded in the same way.

Because all Gateways are direct connections, this will not result in new connections being made. Because a message may need to be forwarded multiple times, this process happens recursively until the message reaches its destination.
(Group Diagramms)

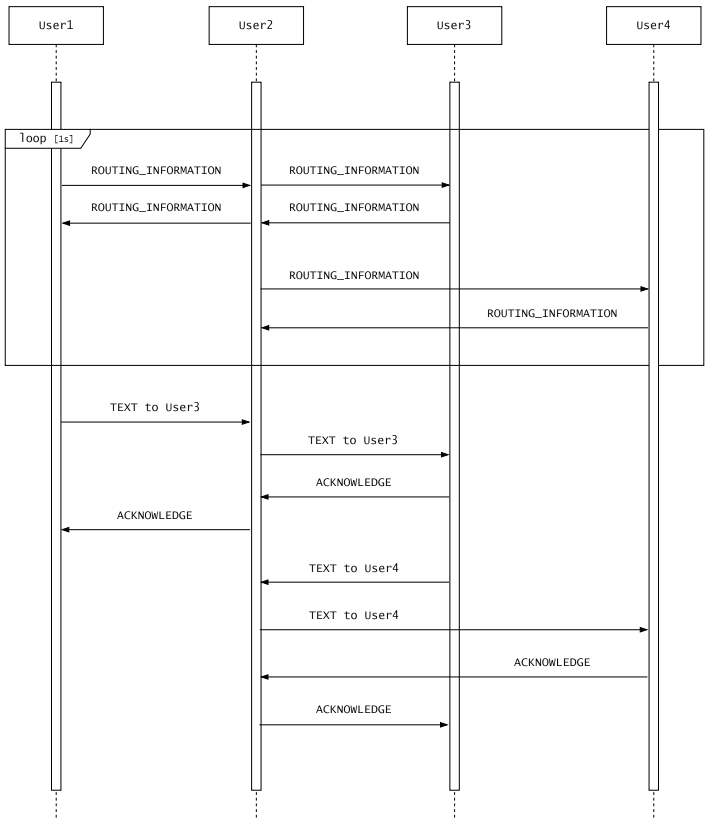

Figure 4: message forwarding **(Group Diagramms)**

3.3.5. Standard format for sending messages over network

JSON (JavaScript Object Notation) MUST be used as the standard format for sending and receiving messages over the network in this protocol.

COMPACT_JSON: This message format ensures that the JSON is transmitted in a compact, serialized format. All unnecessary whitespace, indentation, and line breaks MUST be removed to minimize payload size.
The JSON structure is defined in RFC 8259[6] and MUST be used as follows:

1. **HEADER**
   type: object
   Fields:
   - **MSG_TYPE**
     type: u8
   - **TTL**
     type: u8
   - **SIZE**
     type: u16
   - **SENDER_IP**
     type: u32
   - **DESTINATION_IP**
     type: u32
   - **CHECKSUM**
     type: u32
2. **BODY**
   type: object
   Structure depends on the MSG_TYPE field. The following definitions apply:
   - **TEXT Message (MSG_TYPE = 0)**
     Fields:
     - **MSG_ID**
       type: u8
     - **TEXT**
       type: char[size]
   - **ROUTING_INFORMATION (MSG_TYPE = 1)**
     Fields:
     - **ROUTING_TABLE**
       A list of routing entries. Each entry contains:
       - **DESTINATION**
         type: u32
       - **GATEWAY**
         type: u32
       - **HOP_COUNT**
         type: u8
   - **ACKNOWLEDGE (MSG_TYPE = 2)**
     Fields:
     - **MSG_ID**
       type: u8

- **NOT_ACKNOWLEDGE (MSG_TYPE = 3)**
  Fields:
  - **MSG_ID**
    type: u8
  - **ERROR_CODE**
    type: u8
    - 0: Not reachable
    - 1: TTL expired
    - 2: CRC failed
    - 3: Invalid message format
    - 4: Payload size exceeded
    - 5: Other

3. **EXAMPLES**
   Example of a compact JSON message for each MSG_TYPE:

   - **TEXT Message:** {"HEADER":{"MSG_TYPE":0,"TTL":16,"SIZE":61,
     "SENDER_IP":3232235521,"DESTINATION_IP":3232235522,"CHECKSUM":2309737967},
     "BODY":{"MSG_ID":0,"TEXT":"Hallo, dies ist eine Beispielnachricht."}}
   - **ROUTING_INFORMATION:** {"HEADER":{"MSG_TYPE":1,"TTL":1,"SIZE":39,
     "SENDER_IP":3232235521,"DESTINATION_IP":3232235522,"CHECKSUM":0},
     "BODY":{"ROUTING_TABLE":[
     {"DESTINATION":3232236032,"GATEWAY":3232236034,"HOP_COUNT":2},
     {"DESTINATION":167772160,"GATEWAY":3232236035,"HOP_COUNT":3},
     {"DESTINATION":2886729728,"GATEWAY":3232236036,"HOP_COUNT":1}]}}
   - **ACK:** {"HEADER":{"MSG_TYPE":2,"TTL":16,"SIZE":1,"SENDER_IP":3232235522,
     "DESTINATION_IP":3232235521,"CHECKSUM":439041101},"BODY":{"MSG_ID":0}}
   - **NACK:** {"HEADER":{"MSG_TYPE":3,"TTL":16,"SIZE":2,"SENDER_IP":3232235522,
     "DESTINATION_IP":3232235521,"CHECKSUM":439041101},
     "BODY":{"MSG_ID":0,"ERROR_CODE":0}}

IMPLEMENTATION NOTE The compact JSON serialization MUST be enforced during implementation. The following guidelines SHOULD be adhered to:

- Use a serialization method that removes all whitespace, indentation, and line breaks.
- Ensure that the message payload does not exceed the specified SIZE field.
- Validate the CHECKSUM to ensure data integrity.

(Henske, Lars; Rathje, Torben)

3.4. Routing

3.4.1. Routing Table

Every row of the routing table represents one route to a specific endpoint (destination). All available routes collectively form the user's routing table.
(Fischer, Leon)

The following technical requirements MUST be implemented:

- Max Hop count = 16
- Hop count > 16 or Hop count < 0 -> unreachable
- Only Entries with the lowest Hop count or equal Hop count but different Gateway
- Prioritize Longest Prefix Match

Destination	**Gateway**	**Hop count**
IPv4 address (u32)	IPv4 address (u32)	counter (u8)

Table 3: routing table **(Konoplev, Stepan)**

3.4.2. Distance Vector Routing

DVR: Distance Vector Routing, as defined in STD 56, RIP Version 2 [4]
DVR selects the optimal route based on the metric, which in this case is the number of hops (Hop Count). The route with the fewest hops is preferred. Routing tables are exchanged with neighboring nodes at a fixed interval of 1 second. Although periodic exchange of routing tables generates additional traffic, it ensures continuous monitoring of neighbors. This helps determine whether a neighbor is still active or not.

Split Horizon: Split Horizon is a method used in routing protocols to avoid problems like routing loops and the Count-to-Infinity problem. It ensures that routing information is not sent back to the router from which it was originally received. This prevents routing information from circulating in loops and improves network stability.

TTL: The Time to Live (TTL) indicates how many hops a packet can still make before being discarded. With each forwarding of the packet, the TTL value is decremented by 1. If the TTL value reaches 0 before the packet reaches its destination, it is discarded, and a NOT_ACKNOWLEDGE is sent back to the sender to inform them that the packet could not be forwarded due to the expired TTL.
For the purpose of this protocol's application in small networks, the TTL SHOULD be set to 16.
(Mjawaz, Yousef)

3.5. Checksum

CRC32: The checksum MUST be calculated solely based on the payload of messages of type TEXT. It MUST also be verified on the receiving side, using only the payload of messages of type TEXT.
If the checksum, calculated by the receiver, is different than the one in the header of the message, then the receiver MUST send a NOT_ACKNOWLEDGE message with error code 2 (see error cases).
For message types ROUTING_INFORMATION, ACKNOWLEDGE, and NOT_ACKNOWLEDGE, no checksum is calculated or verified.
CRC as defined in [5]
(Althiab, Hadi)

3.6. Error cases

Destination IP Not Found: The specified destination IP address could not be resolved or does not exist within the network. This may indicate a routing issue or an incorrect destination address.
Handling Send NACK with error Code 0 to source IP

TTL expired: Message has exceeded the maximum TTL.
Handling Send NACK with Error Code 1 to source IP

CRC Invalid: The message's CRC failed, indicating that the data has been corrupted during transmission.
Handling Send NACK with Error Code 2 to source IP

Invalid Payload Format: The received payload does not conform to the expected protocol format, such as missing required fields, incorrect data types, or an unrecognized message type.
Handling Send NACK with Error Code 3 to source IP
(Henske, Lars)

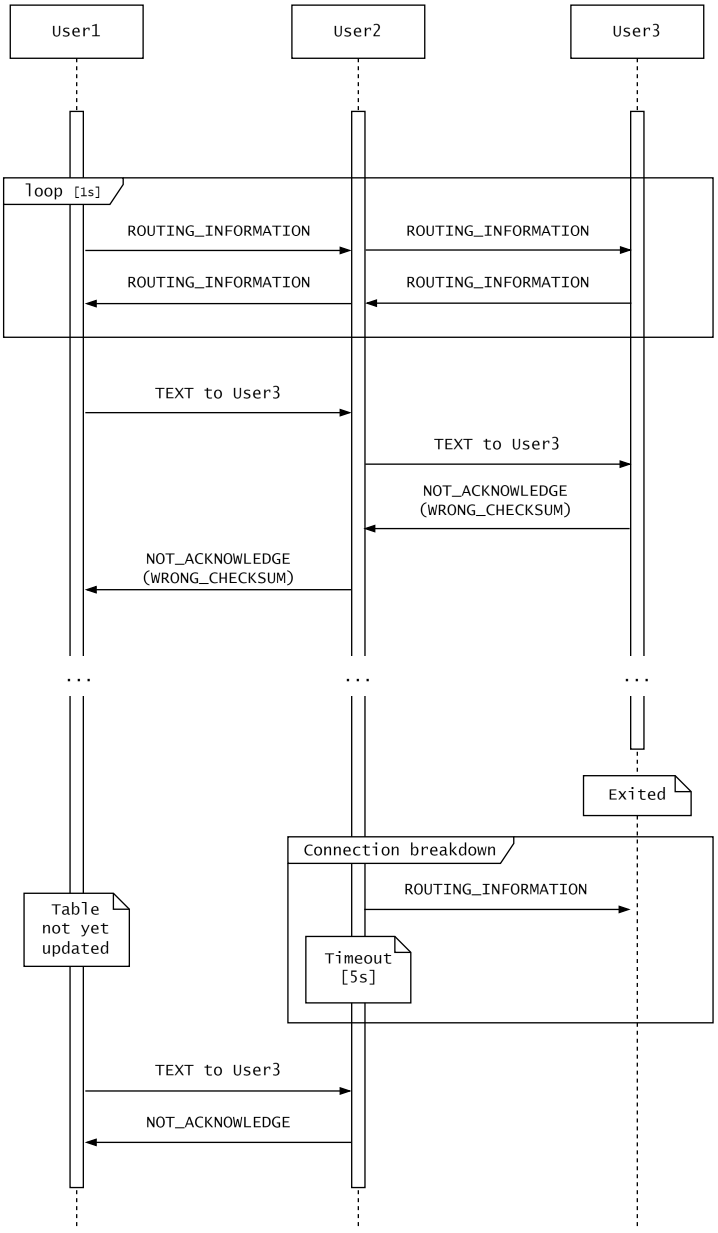

Figure 5: Error Cases **(Group Diagrams)**

4. Known Limitations

4.1. No Response (No ACK/NACK)

In cases where a connection exists, but a node does not receive an ACK/NACK response for a sent message, the system operates on a best-effort basis, waiting indefinitely for a response or until the disconnection is detected via the routing table. This can result in prolonged or endless waiting periods.

(Henske, Lars)

4.2. Acknowledge of a Disconnect

Nodes are not immediately notified when connected nodes disconnect, whether intentionally or unintentionally. The disconnection is only detected after a predetermined time interval during the routing table update. The greater the number of hops between two disconnected nodes, the longer it takes to detect the disconnection. As a result, nodes may attempt to send messages to offline nodes.

(Henske, Lars)

5. Sources

References

[1] S. Bradner, ‘‘Key words for use in RFCs to Indicate Requirement Levels,’’ RFC 2119, March 1997. Available: https://www.rfc-editor.org/rfc/rfc2119

[2] J. Postel et al., ‘‘Transmission Control Protocol,’’ RFC 9293, STD 7, August 2022. Available: https://www.ietf.org/rfc/rfc9293.html

[3] J. Postel, ‘‘Internet Protocol,’’ RFC 791, September 1981. Available: https://www.rfc-editor.org/rfc/rfc791

[4] RFC Editor, ‘‘Routing Information Protocol Version 2,’’ RFC 2453, 1998. Available: https://www.rfc-editor.org/rfc/rfc2453

[5] S. Floyd et al., ‘‘CRC-Based Framing for TCP,’’ RFC 3385, September 2002. Available: https://www.rfc-editor.org/rfc/rfc3385

[6] T. Bray, ‘‘The JavaScript Object Notation (JSON) Data Interchange Format’’ RFC 8259, December 2017. Available: https://datatracker.ietf.org/doc/html/rfc8259