Inventory Management Protocol

Shreya Laheri
College of Computing & Informatics
Drexel University
Philadelphia, USA
sl3798@drexel.edu

Rahul Bolineni
College of Computing & Informatics
Drexel University
Philadelphia, USA
rb3432@drexel.edu

Apurva A. Deshpande
College of Computing & Informatics
Drexel University
Philadelphia, USA
aad368@drexel.edu

***Abstract*—The Inventory Management Protocol facilitates real-time communication between client systems and inventory management servers, enabling tasks such as item addition, quantity updates, and stock checks. With moderate complexity, the protocol ensures secure and efficient inventory operations in retail environments.**

***Index Terms*—inventory management protocol, quic, client, server, stock**

I. Service Description

The multi-user inventory management protocol facilitates collaborative and concurrent management of inventory across retail stores or distribution centers. Users, including store managers, and team members, interact with the central inventory management server through various client devices such as desktop website, and mobile applications.

Key features of the protocol include user authentication and role-based access control mechanisms. Users are required to authenticate themselves using credentials, ensuring secure access to inventory data and functionalities. Role-based access control mechanisms are implemented to define user permissions based on their roles and responsibilities within the organization. Access permissions are granularly configured to enable or restrict users' abilities to add, update, delete, or view inventory items, ensuring data integrity and confidentiality.

Additionally, the protocol supports concurrent access to inventory data by multiple users, allowing them to perform inventory operations simultaneously without conflicts. Transactions and locking mechanisms are employed to manage concurrent modifications, preventing data inconsistencies and ensuring data integrity across all users. Users can view real-time updates made by other users, providing a comprehensive and up-to-date view of inventory status and changes.

II. Message Definition (PDUs)

A. Query Inventory Message (QIM)

Description: Sent by the client to request information about the current inventory.
Fields:

- **Message Type (1 byte):** Type identifier for QIM.
- **Client ID (4 bytes):** Unique identifier for the client.
- **Timestamp (8 bytes):** Timestamp of the message.

Data Types:

- **Message Type:** Unsigned integer (uint8).
- **Client ID:** Unsigned integer (uint32).
- **Timestamp:** Unsigned long long integer (uint64).

B. Inventory Response Message (IRM)

Description: Sent by the server in response to a query inventory request, containing inventory information.
Fields and Data types:

- **Message Type (1 byte):** Type identifier for IRM.
- **Item ID (4 bytes):** Unique identifier for the item.
- **Item Name (variable length string):** Name of the item.
- **Quantity (4 bytes):** Current quantity of the item in inventory.

Data Types:

- **Message Type:** Unsigned integer (uint8).
- **Item ID:** Unsigned integer (uint32).
- **Quantity:** Unsigned integer (uint32).
- **Item Name:** Character array.

Here's the message definition section in a Python-style typedef:

```python
from typing import Tuple

# Define a tuple type for Query Inventory Message
QueryInventoryMessage = Tuple[int, int, int]

# Define a tuple type for Inventory Response Message
InventoryResponseMessage = Tuple[int, int, str, int]
```

Fig. 1. Python typedef

In this Python-style typedef:

- `QueryInventoryMessage` is defined as a tuple containing three integers: `messageType`, `clientID`, and `timestamp`.
- `InventoryResponseMessage` is defined as a tuple containing four elements: `messageType`, `itemID`, `itemName`, and `quantity`.

III. DFA

In order to create a DFA for the inventory management protocol that allows multi-user approach, below are the steps representing the protocol's stateful nature and the transitions

```
# Example usage
query_msg = QueryInventoryMessage(1, 123456, 1620400000)
print(query_msg)

response_msg = InventoryResponseMessage(2, 1001, 'Item A', 50)
print(response_msg)
```

Fig. 2. Python typedef Example

between different states based on incoming messages and events.

States involved in this protocol are as follows:

- **Idle:** This is the initial state where the protocol waits for a connection from a client. In this state, the server is not actively engaged in any communication.
- **Connected:** Once a client establishes a connection with the server by sending a connection request, the protocol transitions to the Connected state. At this point, the server is ready to receive messages from the client.
- **Authenticating:** Upon receiving an authentication request from the client, the protocol transitions to the Authenticating state. Here, the server verifies the client's credentials to authenticate its identity.
- **Authenticated:** If the authentication is successful, the protocol moves to the Authenticated state. In this state, the client has been successfully authenticated, and the server can proceed with handling inventory-related requests.
- **Inventory Querying:** When the client sends a request to query inventory information, the protocol transitions to the Inventory Querying state. Here, the server processes the request and retrieves the requested inventory data.
- **Inventory Response:** After successfully retrieving the inventory information, the protocol moves to the Inventory Response state. In this state, the server sends the inventory data to the client.
- **Error:** If an error occurs at any point during the protocol execution, such as authentication failure or data retrieval errors, the protocol transitions to the Error state. In this state, the protocol remains until the issue is resolved or the connection is terminated.

Now let's focus on all the inputs or input requests included in this protocol.

- **Connection Request:** It initiates the connection process between the client and server. Upon receiving a connection request from a client, the protocol transitions to the Connected state.
- **Authentication Request:** When the client sends an authentication request to the server, indicating its intention to authenticate, the protocol moves to the Authenticating state.
- **Authentication Response:** Once the server sends an authentication response to indicate successful authentication, this input transitions the protocol to the Authenticated state.
- **Inventory Query:** Here, client requests server for inventory information. Upon receiving such a request, the protocol transitions to the Inventory Querying state.
- **Inventory Response:** After processing the inventory query request and retrieving the inventory information, the server sends an inventory response to the client.
- **Error:** Any unexpected error condition encountered during protocol execution triggers a transition to the Error state.

As we have dived deep into the states, now let's focus more on the continuous transition that will be occuring while the protocol is up and running.

- **Idle to Connected:** Once a connection request from the client is received, the protocol transitions from the Idle state to the Connected state, indicating that a connection has been established.
- **Connected to Authenticating:** When the server receives an authentication request from the client, it transitions to the Authenticating state to verify the client's credentials.
- **Authenticating to Authenticated:** If the authentication process is successful, the protocol moves from the Authenticating state to the Authenticated state, indicating that the client has been successfully authenticated.
- **Authenticated to Session Establishment:** Once the user/client is authentication, a session is established for that particular user for a specific period of time, with the help of login credentials specific for the user.
- **Session Establishment to Inventory Querying:** Here, the protocol transitions from the Authenticated state to the Querying Inventory state, indicating that the server is processing the inventory query.
- **Inventory Querying to Inventory Response:** After successfully retrieving the inventory information, the protocol transitions from Inventory Querying state to the Inventory Response state to send the inventory data to the client.
- **Error Handling:** If an error occurs at any stage of the protocol execution, the protocol transitions to the Error state, where it remains until the issue is resolved or the connection is terminated.

The details included in this DFA illustrates the sequential flow of the multi-user inventory management protocol, outlining the various states, their inputs, and transitions that govern the communication between the client and server. By defining the protocol's behavior in this manner, we ensure that the protocol operates in a deterministic and predictable manner, facilitating reliable communication and error handling.

IV. Extensibility

In order to ensure the extensibility, we need to design the protocol in a way that allows for easy integration of new

features, updates, and extensions without disrupting existing functionality. Here's how we can achieve good extensibility.

- **Version Negotiation:** Including some mechanism or for protocol version negotiation during session establishment state allows client and server to communicate their supported protocol versions and select the most appropriate version for compatibility. Mechanisms such as, Protocol Version Negotiation Message or including transport parameters in the initial connection setup phase to convey information about supported protocol versions, etc.
- **Flexible Handshake Protocol:** A flexible handshake protocol allowing negotiation of additional parameters and options enables future extensions to introduce new handshake messages or options without requiring modifications to the core protocol.

V. Security implications

When using QUIC (Quick UDP Internet Connections) to create the inventory management protocol, the following are several security implications we will consider.

- **Encryption:** QUIC provides built-in encryption using Transport Layer Security (TLS) 1.3, ensuring that all data exchanged between the client and server is encrypted. This protects against eavesdropping and ensures the confidentiality of inventory data.
- **Authentication:** Implement strong authentication mechanisms to verify the identity of both clients and servers. This can include mutual TLS authentication using client and server certificates, ensuring that only authorized entities can access the inventory management system.
- **Authorization:** Enforce access control policies to restrict access to sensitive inventory data based on user roles and permissions. This ensures that users can only perform actions that they are authorized to do, preventing unauthorized access or modification of inventory information.
- **Protection Against DoS Attacks:** QUIC includes features such as connection migration and stateless retries to mitigate Denial of Service (DoS) attacks. Implement rate limiting and other protective measures at the application layer to further safeguard against DoS attacks targeting the inventory management system.
- **Data Integrity:** QUIC provides mechanisms for data integrity verification, ensuring that data exchanged between the client and server has not been tampered with during transmission. Implement checksums or message authentication codes (MACs) to verify the integrity of inventory data.
- **Session Resumption:** QUIC supports session resumption to improve performance by allowing clients to resume previous sessions without reestablishing a new connection. Ensure that session resumption does not compromise security by properly managing session tickets and keys.
- **Secure Key Management:** Implement secure key management practices to protect cryptographic keys used for encryption and authentication. This includes securely storing keys, rotating keys periodically, and protecting against key leakage or theft.
- **Secure Configuration:** Ensure that QUIC is configured securely, following best practices and guidelines provided by the QUIC protocol specification and relevant security standards.

Conclusion

In conclusion, the adoption of QUIC for developing the Inventory Management Protocol will support various authentication mechanisms, including username/password authentication and token-based authentication. Role-based access control mechanisms can be implemented to restrict access to inventory information based on user roles and permissions. It will thus provide numerous benefits, including improved performance, enhanced security, and better reliability. However, it also presents unique challenges, particularly in terms of firewall inspection and security implications. By implementing appropriate security measures such as encryption, authentication, access control we can mitigate these challenges and ensure the secure and efficient management of inventory data.

References

[1] https://techcommunity.microsoft.com/t5/networking-blog/what-s-quic/ba-p/2683367
[2] https://en.wikipedia.org/wiki/QUIC
[3] R. Lychev, S. Jero, A. Boldyreva and C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses," 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015, pp. 214-231, doi: 10.1109/SP.2015.21. keywords: Protocols;Servers;Encryption;IP networks;Public key
[4]
[5]
[6]
[7]
[8]