Simon Dereje Woldearegay (UGR-0952-14)

1. **How does the buffer size affect the frequency and timing of message passing?**

In Go, the buffer size of a channel determines the number of messages that can be held in the channel before the sender is blocked. A larger buffer allows the producer to send multiple messages without waiting for the consumer to process them, thus increasing the frequency of message passing. Conversely, with a smaller buffer size (or an unbuffered channel), the producer might have to wait until the consumer processes a message before it can send the next one, slowing down the frequency and creating a more synchronous flow. Therefore, the buffer size directly impacts the timing and frequency of message transmission.

2. **What happens when the buffer is full?**

When a buffered channel in Go is full, any additional send operation will block until there is space available in the buffer. This means that the producer goroutine will pause and wait until the consumer consumes at least one message from the channel, freeing up space. If the consumer is slow or not consuming messages at all, the producer will be halted until consumption resumes.

3. **How does RabbitMQ handle load balancing between multiple consumers?**

RabbitMQ employs a round-robin dispatch method for load balancing between multiple consumers of a queue. When messages are published to a queue, RabbitMQ distributes them among the available consumers in a round-robin fashion. This ensures that each consumer receives an approximately equal share of the workload. RabbitMQ also uses acknowledgments (`ack`) to ensure that a message is only considered processed once a consumer explicitly acknowledges it. If a consumer fails before sending an acknowledgment, RabbitMQ will re-queue the message and deliver it to another available consumer, enhancing reliability and load distribution.

4. **What happens when a consumer disconnects?**

When a consumer disconnects unexpectedly, RabbitMQ stops delivering messages to it and re-queues any unacknowledged messages that the consumer might have been processing. These messages are then redistributed to other active consumers in the queue. This ensures that no message is lost due to consumer failure, as the message will be retried with another consumer. If there are no other consumers, the message remains in the queue until a new consumer is available to process it.

5. **How does NATS handle different subjects?**

NATS uses a publish-subscribe model with subjects acting as channels or topics for message organization. A subject in NATS is a string identifier that allows publishers to send messages to a specific subject, and subscribers can listen to messages from specific subjects. NATS also supports wildcard subjects (e.g., `updates.*`), which allow subscribers to listen to a range of related topics. This flexible approach to subject naming helps in categorizing messages effectively.

6. **What advantages does this give in message organization?**

The use of subjects in NATS provides several advantages:

- **Scalability**: Publishers and subscribers can scale independently, as they are loosely coupled by the subject name.
- **Flexible Filtering**: With wildcard subjects, NATS allows subscribers to receive only the messages relevant to their interests, reducing unnecessary message processing.
- **Enhanced Organization**: Subjects provide a clear and logical way to organize messages based on different categories or types, making it easier to manage and process data in distributed systems.