**Investigation Report:**

**Telegram Message Failures on the App Respond.io**

1. **Reproduction Steps**

* **Goal:**

To create a clear set of steps to consistently reproduce the issue and investigate its root cause.

* **Approach:**
  + **Customer Information Review:**

Begin by gathering customer-provided details about the issue. Ask customers to describe when the failure occurs (time, message type, device, etc.). Commonly reported data points include:

- Time and frequency of failure

- Message type (text, media, file)

- Any noticeable patterns, such as connection loss during message transmission

* + **Review Historical Data and CloudWatch Logs:**

Access AWS CloudWatch logs to review historical failures of the Telegram integration. Focus on:

- Frequency of failures across different times and days

- Any patterns linked to specific customers, message types, or server conditions

* + **Check Message Types & Formats:**

Cross-reference the types of messages being sent—such as plain text, media (images, videos), or files—and note whether certain types of messages fail more frequently than others.

Investigate any issues related to message size limits, media formats, or invalid characters that could trigger API errors.

* + **Simulate Under Similar Conditions:**

Attempt to simulate the same conditions in a controlled environment:

- Recreate message sending using the Telegram API with various message types and sizes

- Test during similar traffic periods (peak vs. low traffic) to see if the failures correlate to server load.

- Monitor real-time logs during these simulations to detect any abnormal errors or warnings.

1. **Understanding the Intermittency of the Issue**

* **Goal**

To determine why the issue occurs intermittently, rather than consistently.

* **Approach**:
  + Pattern Analysis:

Review error logs and look for commonalities among failed messages. For example:

- Does the failure happen more frequently during peak hours (morning, afternoon)?

- Is the issue location-based? For instance, are users in a specific region affected more?

- Is the problem more likely with certain message types or larger payloads, such as media attachments?

* + Hypothesis Testing:

Based on initial findings, form and test hypotheses:

- Time-Based Failures: Check if message failures correlate with high-traffic periods or when server loads are increased.

- Telegram API Rate Limits: Review Telegram’s API rate limits to see if the issue is linked to hitting these thresholds. When too many messages are sent within a specific window, Telegram may throttle or reject API requests.

- Specific Message Types: Analyze if media or larger file messages are more likely to fail compared to smaller, text-based messages.

* + CloudWatch Log Review for External Factors:

Investigate AWS CloudWatch logs for network disruptions or API rate-limit errors that might be causing the intermittent failures. Look for:

- Network timeout errors

- API call rate limits exceeded

- Service interruptions with the Telegram API or dependencies on other third-party services

**3. Data Collection from AWS CloudWatch**

* **Goal:**

Extract relevant data from AWS CloudWatch logs to narrow down the issue.

* **Approach:**
  + Collect Error and Warning Logs:

Focus on error and warning logs tied to the Telegram API. Common error codes or messages related to message failures might include:

- API rate limits exceeded

- Network connectivity issues

- Invalid message format errors

* + Time-Specific Logs:

Isolate logs during the exact time frames where failures were reported by customers. Compare these logs to successful message transmissions, looking for any anomalies.

* + Correlate Logs with Customer Messages:

For each reported failure, match the log data to the corresponding customer message. This helps in identifying specific patterns, such as failure with larger media files or messages sent at peak traffic hours.

* + Identify Repeating Error Patterns:

Analyze the CloudWatch logs for recurring error codes or failure messages that occur during reported incidents. This can help pinpoint if the issue is related to network failures, API limits, or external service disruptions.

**4. Root Cause Identification**

* **Goal:**

Determine the exact cause of the intermittent failures in sending messages through the Telegram channel.

* **Approach:**
  + Log Analysis with CloudWatch Insights:

Use AWS CloudWatch Insights to filter and query large sets of logs for specific error codes or patterns tied to Telegram API failures. Look for patterns that are common in failure cases (e.g., rate limiting, API timeout).

* + Investigate Telegram API Usage:

Work closely with the development team to ensure the Telegram API is correctly integrated and configured. Review the following:

- API authentication and request limits

- Error handling and retry logic

- Message size restrictions or media format compliance

* + Load and Stress Testing:

Simulate high-load conditions on the platform to see if message failures occur at specific traffic thresholds. By stress testing, it’s possible to identify whether the platform becomes unstable under high demand, leading to intermittent message failures.

* + Third-Party Service Check:

Investigate third-party services or dependencies, such as AWS services or external API gateways, that might be involved. Check for outages or delays that could indirectly cause failures.

**Findings and Root Cause Analysis**

1. **Brief Investigation Report:**

* **Reproduction Steps:** We successfully reproduced the issue by simulating large traffic loads and sending multiple media files via Telegram API. Failures occurred primarily during peak load times.
* **Patterns Discovered:** The issue was more frequent with media files and when API rate limits were reached. Text-based messages had a higher success rate. Failures occurred during times of high traffic or API request spikes.
* **Key Findings:**Logs showed repeated "API Rate Limit Exceeded" errors during failures, along with occasional network timeouts from Telegram's servers.

**2. Detailed Root Cause Analysis:**

* **Root Cause:**The intermittent message failures are primarily due to Telegram API rate limits being exceeded during high traffic times. Media messages, especially larger files, are more prone to failure. Additionally, occasional network connectivity issues between AWS and Telegram servers contributed to some message drops.
* **Technical Details:**CloudWatch logs show repeated error codes related to rate limiting (e.g., "429 Too Many Requests"), along with network timeouts. Specific timestamps correlate to high-traffic periods where multiple users attempted to send media-heavy messages simultaneously.

**3. Recommendations for Prevention:**

* Telegram API Rate Limit Handling:

- Implement more robust error handling and retry logic when API rate limits are exceeded.

- Consider batching messages or spreading out API requests over time to avoid  
triggering rate limits.

* Enhanced Monitoring and Alerts:

- Set up real-time monitoring and alerts for rate-limit errors, network failures, or high traffic periods in AWS CloudWatch.

- Use tools like AWS CloudWatch Alarms to proactively notify teams when the error rate increases.

* System Optimization for High Traffic:

- Conduct regular load testing to ensure the system can handle peak loads and simulate various scenarios (e.g., media-heavy messages).

- Consider adding caching mechanisms or alternative message delivery systems for high-volume periods.

* Improved API Integration:

- Work with the development team to optimize API usage, ensuring that all requests follow Telegram’s best practices for message size, format, and rate limit thresholds.

**Conclusion:**

This detailed investigation, root cause identification, and recommended actions can help address the issue and ensure customer satisfaction by improving message delivery stability across Telegram.