## Step 1: Initial Observation

Questions to Answer:

**1. What did you observe first that indicated there was a problem?**

These were the following issues that were indicative of a problem:

- Messages started duplicating *AND*
    - Disconnecting at random
    - Appearing out of order

**2. Can you consistently reproduce the issue, or does it seem random?**

While it seems mostly random, I can reproduce the issues by:

- Navigating between screens
- Sending multiple messages
- Leaving the app running idly

**3. What specific behaviors seem "random" about the real-time features?**

- Messages will unreliably send twice
- Sometimes the socket reconnects instantly, other times it'll get stuck in a loop
- Listeners fire unreliably; sometimes once, sometimes multiple times
- The "Chat Connected" log appears more than once at random

**4. When does the issue occur (immediately, after time, after specific actions)?**

These issues don't appear immediately. The user usually has to:

- Navigate to the chat screen more than once
- Send multiple messages
- Attempt to reconnect
- When the component unmounts/remounts (by refresh usually)

## Step 2: Environment and Context Analysis

Questions to Answer:

**1. Does this happen on all platforms (web, iOS, Android) or specific ones?**

I tested this on the **web** and **iOS** and the behavior was consistent across all of them.

**2. Does it happen with multiple users or just single user testing?**

The bugs are local so it happens with a single user testing.

**3. Are there any console errors or warnings when the issue occurs?**

***Yes***. These were the following console errors that appeared during testing:

- Socket reconnection warnings
- Multiple “connected” logs appearing
- Random occurrences of “transport close” or “disconnected” messages
- Logs relating to duplicate listeners

**4. What is the network environment (WiFi, cellular, corporate network)?**

I was only able to test this application with **WiFi**, but I don’t think that there is any correlation between the network environment and these issues because the bugs are internal, *AKA* within the code itself, not network-based.

## Step 3: Connection State Investigation

Questions to Answer:

**1. What does the browser's Network tab show for WebSocket connections?**

Multiple WebSocket connections appear over time instead of a single persistent one. Some of these connections remain stuck on “pending” even after disconnecting.

**2. Are there multiple socket connections being created?**

***Yes***. `Connect()` is called inside a hook that runs multiple times without any pre-existing logic established to check if a socket already exists.

**3. How does the connection state change over time?**

- Connection state ping-pongs between connected and disconnected
- Multiple reconnection attempts stack up in the log
- Duplicate “connect” handlers are firing all at once
    - In addition, event listeners begin to accumulate, which amplifies the behavior

**4. Are there any patterns to when connections fail?**

These seem to be the events that trigger a connection failure:

- Navigating between screens
    - When chat remounts, for example
- Reconnect events firing
    - `setupEventListeners` is called again and again
- Sending more than one message
    - Invokes more listeners

## Step 4: Memory and Resource Analysis

Questions to Answer:

**1. Is memory usage increasing over time?**

***Yes***. MemoryMonitor shows steady JS growth even when the app is idle, which I think is because of the listeners rapid firing and the sockets going stale.

**2. Are event listeners accumulating?**

***Yes***. The `eventListenerCount` function runs every time:

- `initalizeChat` is called
- Whenever a socket reconnects
- Whenever a component mounts/remmounts

This correlates with one of the expected discoveries, **memory leaks**.

**3. How many Socket.io connections are active?**

Way more than there should be, *AKA* more than one at a time. This is because old sockets are never cleaned up and the reconnection logic keeps recreating new listeners.

**4. Are there JavaScript errors in the console?**

Not always. But some of the warnings and repeated console logs are pretty indicative of something going wrong.

Such as:

- Multiple “chat connected logs”
- Events firing more than once
- Reconnection loops

## Step 5: Code Review and Pattern Identification

Questions to Answer:

**1. Are socket connections being properly cleaned up?**

***No***. The `disconnect()` function never sets the socket to `null`, which is mandatory if you wish to remove stale socket connections. Also, the hooks create new sockets without checking for an existing one.

**2. Are event listeners being removed when components unmount?**

***No***. Every listener registration lacks any cleanup. The problematic hook doesn't even have a cleanup function to begin with. Additionally, the `off()` function doesn't update the internal listener tracking, further contributing to the chaos.

**3. Is there logic that runs multiple times when it should run once?**

Yes. These are my findings:

- `intializeChat` runs every time the effect is called
- The socket service reattaches listeners on reconnect
- Component navigation recreates the hook instances
- The listeners begin to accumulate with no limit or cut off

**4. Are there race conditions in connection/reconnection logic?**

***Yes***. When a reconnect occurs, `setupEventListeners` is called again which is adding duplicate listeners before the previous ones can finish firing.

## Step 6: Systematic Testing of Hypotheses

Questions to Answer:

**1. If you suspect duplicate listeners, how can you test this?**

This is how I would test for duplicate listeners:

- Watch the `eventListenerCount` in the `useConnectionMonitor` output
- Count how many times "chat connected" or "new_messages" logs in the console
- Trigger the screen mounting/unmounting via refresh or changing screen views and watch the listener count increase.

**2. If you suspect connection cleanup issues, what evidence would confirm this?**

If I suspected connection cleanup issues, the following evidence would be essential in order to to make a declarative confirmation:

- Multiple WebSocket connections appearing in the network tab
- The socket object persists even when disconnected
- Reconnection attempts keep piling up even after the screen is unmounted
- Memory continues to rise because the stale sockets aren't being killed

**3. How can you test your fixes without introducing new problems?**

If I was testing my fixes and wishing to avoid introducing any new problems to the project, I would:

- Add cleanup functions to effects and observe whether `eventListenerCount` stabilizes
- Ensure `socket = null` after disconnect and confirm only one active connection persists
- Add logs that'll verify how many times `connect()` is being called

After that, I would observe the memory usage to see if it stabilizes or rises