RxJS Fault-Tolerant Error Handling

PROJECT MEMBERS

Hartmut Sinterhauf

Micheal McMullen

PROJECT SPONSOR

Professor Tian Zhao

1. INTRODUCTION

Currently there is a need for better tools to work with streaming applications in the context of IOT systems. Collecting data from multiple IOT data sources and creating meaningful insights from them can be a challenge and the current JavaScript tools for doing so don't provide some of the key functionality necessary to do so effectively all in one module. Also, concern over the lack of clear debugger tools has been expressed by our sponsor.

RxJS is a JavaScript module that implements much of the originally envisioned functionality of our project, including the splitting and combining of streams. It also allows for the transformation of streams using functional programming operators such as map, filter, reduce. RxJS Observers and Observables to implement both the Observer Pattern, where nodes keep a list of dependents and update them when state changes are made, as well as the Iterator Pattern, which abstracts the ability of an objects items to be traversed without revealing it's representation. An observer subscribes to an observable. The observable emits streams of data which the observer listens and reacts to. Observable can execute 3 types of values: next, error, complete. The next notification sends a value such as a numbers, string, or object. The error value sends a JavaScript error or exception. The complete does not send a value but rather halts the Observable.

The only requirement left to implement from our original project scope is fault-tolerance. Currently the extent of RxJS's error handling is to shutdown an Observable when an 'error' signal is received. This prevents the Observable from calling 'next' and processing more data. Fault-Tolerance in this case would be to capture errors based on user defined criteria and handle them where they occur. A default value, tagged as an error, could be propagated down stream until the error is resolved for a more graceful degradation of the system. This prevents dependents from coming to a hard fail when only one dependency is in an error state.

We are using the paper “Fault-Tolerant Distributed Reactive Programming” as a guide for the considerations to factor into our implementation. This paper will help us understand how to implement eventual consistency as well as how to cope with faults from different devices.

This fault-tolerant addition to RxJS would allow for the rapid prototyping of stream based IOT applications for JavaScript developers with clear and informative debugging statements related to the status of streams and calculations based on those streams.

2. GLOSSARY

KPI :

Key Performance Indicator. A performance indicator used to evaluate the success of an organization.

Window-based KPI :

A KPI which is gathered over a period of time. An example could be the average flow rate of over the last hour time window.

Point-based KPI :

A KPI which is based on a specific point in time. An example could be the maximum value found in a stream of data.

Functional composition :

The mechanism whereby simple functions are connected to build more complicated ones.

Reactive programming :

Asynchronous programming paradigm concerned with data streams and the propagation of change. Streams of data are actively observed by the consumers rather than the stream source telling the consumer new data is available. This related to the 'Hollywood Principle' of "don't call us, we'll call you".

MQTT (MQ Telemetry Transport) :

Lightweight publish-subscribe messaging protocol for small sensors and mobile devices, optimized for high-latency or unreliable networks.

Fault-Tolerance :

A ‘fault’ is referred to as the origin of a failure which could be a node’s process failing or disconnecting for a period of time. Fault-Tolerance is the systems ability to continue operating in some proper capacity even if some of it’s components or nodes fail.

RxJS :

Reactive JavaScript (RxJs) is a library for composing asynchronous and event-based programs by using observable sequences

Observable :

Observables are the core type provided by RxJS which encapsulates an iterable collection or stream of objects and allows for operations on them through the use of map, filter, reduce, etc.

3. SCOPE

Our project's updated goals are to create the following:

- Refactor RxJS Error handling to be more Fault-Tolerant
- A mock environment for testing composed of an MQTT broker with various Node.js MQTT clients, used as an abstraction for various IOT devices, intermittently publishing data.
- An express server utilizing our refactored RxJS library to perform streaming data analytics on our mock testing environment.

4. TIMELINE

Our project workflow is more akin to an Agile software development process as opposed to a Waterfall methodology. What is meant by this is that the project does not have a strict set of dependencies but rather the project members as well as the project sponsor, Professor Tian Zhao, are continually discovering the project parameters and scope. Team members meet weekly with the product sponsor to discuss progress and plan the week's work to be complete until the next meeting. Members work on both research based solution discovery by reading academic papers provided by the sponsor as well as resources found on their own.

5. SOLUTION OVERVIEW

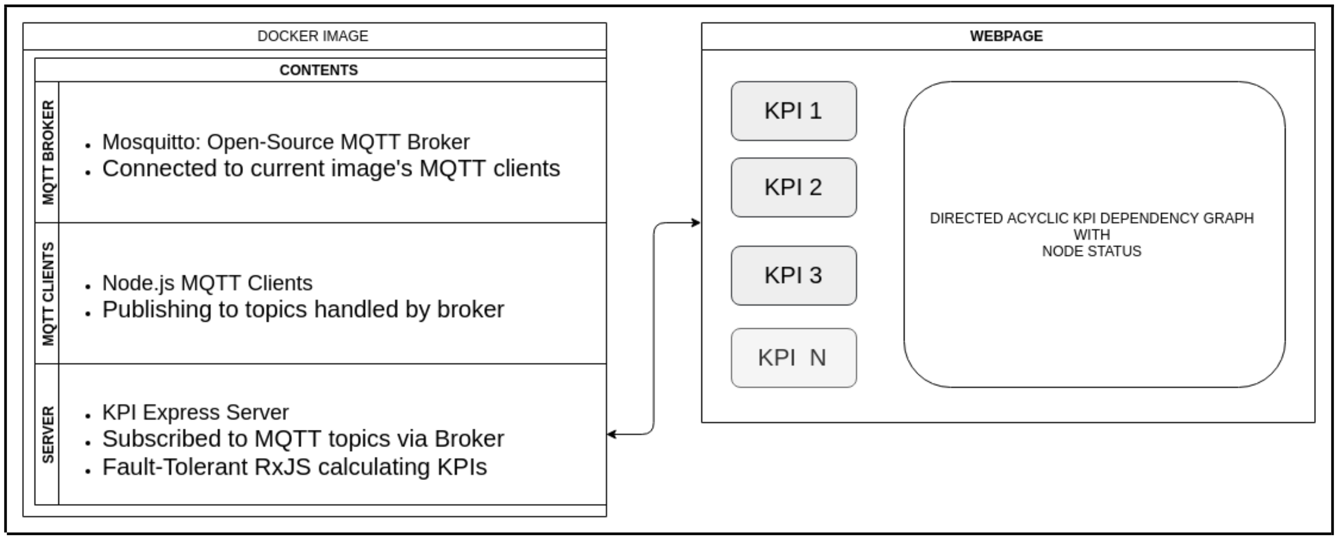
DOCKER IMAGE
CONTENTS
MQTT BROKER
Mosquitto: Open-Source MQTT Broker
Connected to current image's MQTT clients
MQTT CLIENTS
Node.js MQTT Clients
Publishing to topics handled by broker
SERVER
KPI Express Server
Subscribed to MQTT topics via Broker
Fault-Tolerant RxJS calculating KPIs
WEBPAGE
KPI 1
KPI 2
KPI 3
KPI N
DIRECTED ACYCLIC KPI DEPENDENCY GRAPH WITH NODE STATUS

Figure 5.1: Demo Architecture

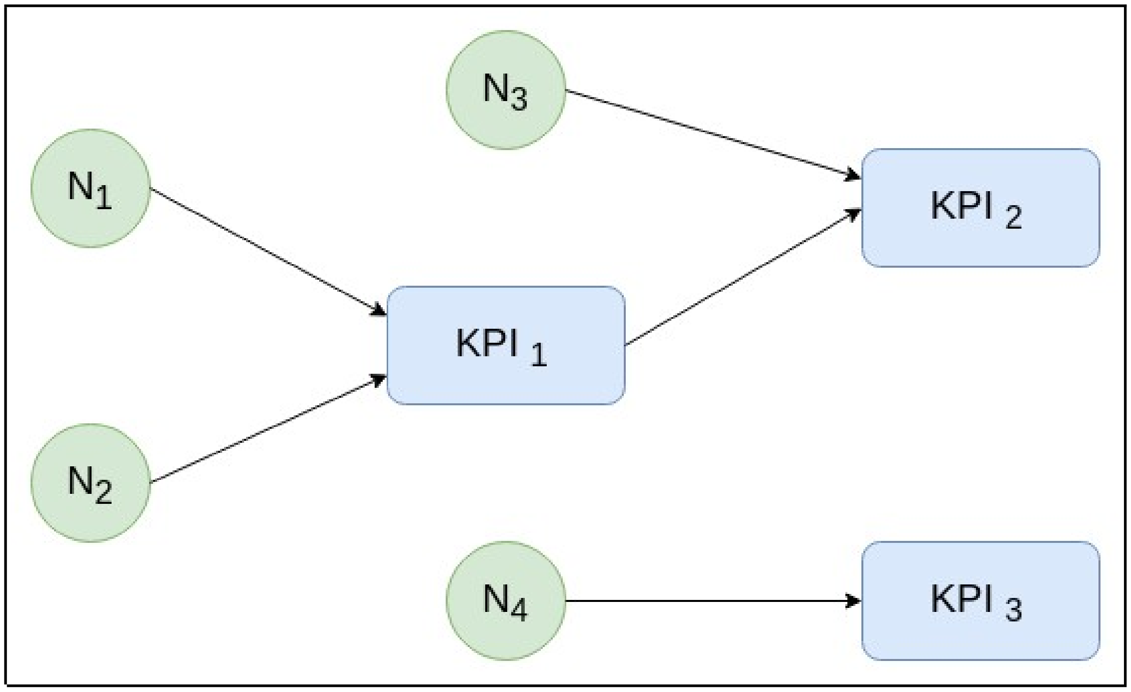

Figure 5.2: KPI Directed Acyclic Dependency Graph

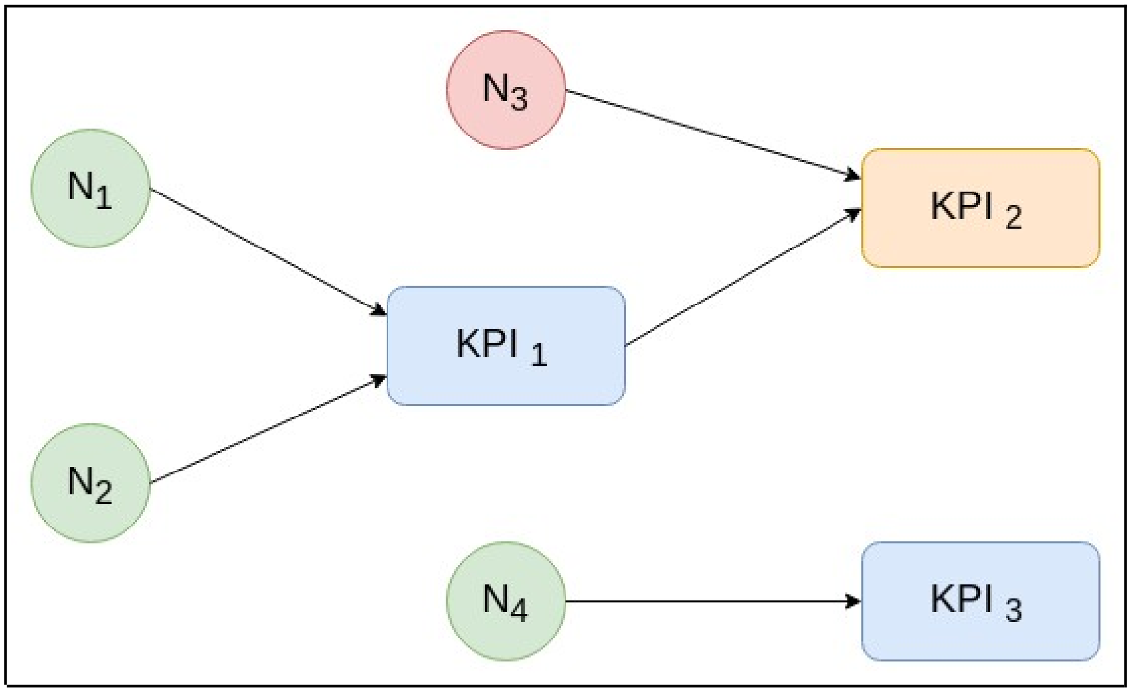

Figure 5.3: KPI Directed Acyclic Dependency Graph: N_3 Failure

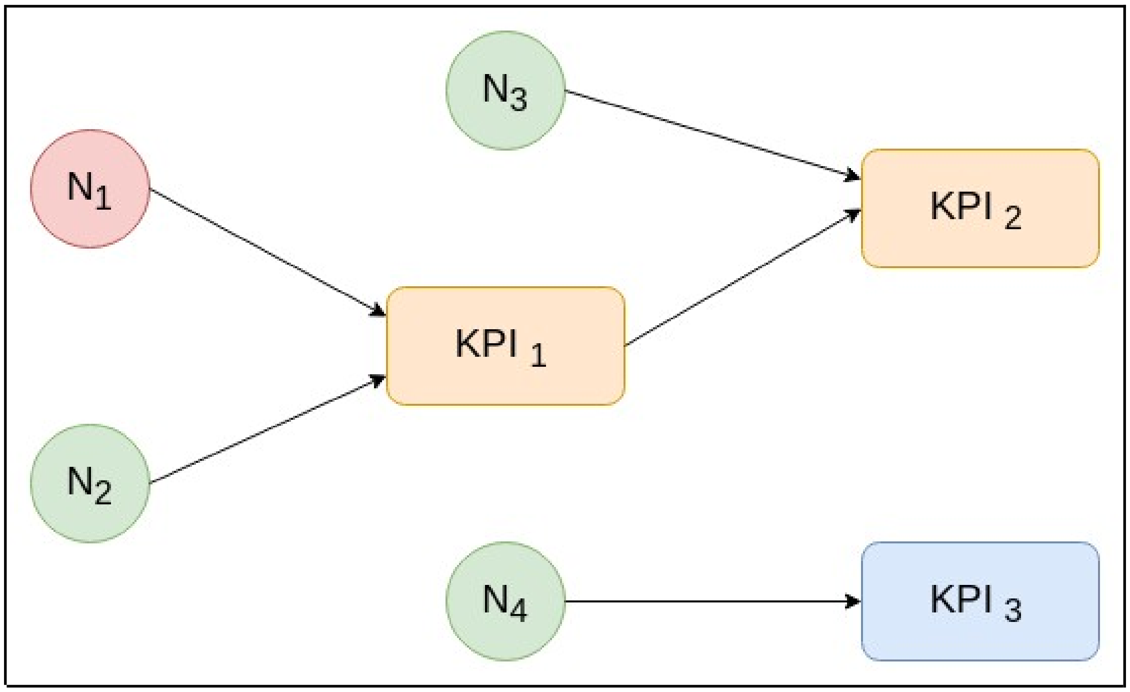

Figure 5.4: KPI Directed Acyclic Dependency Graph: N_1 Failure

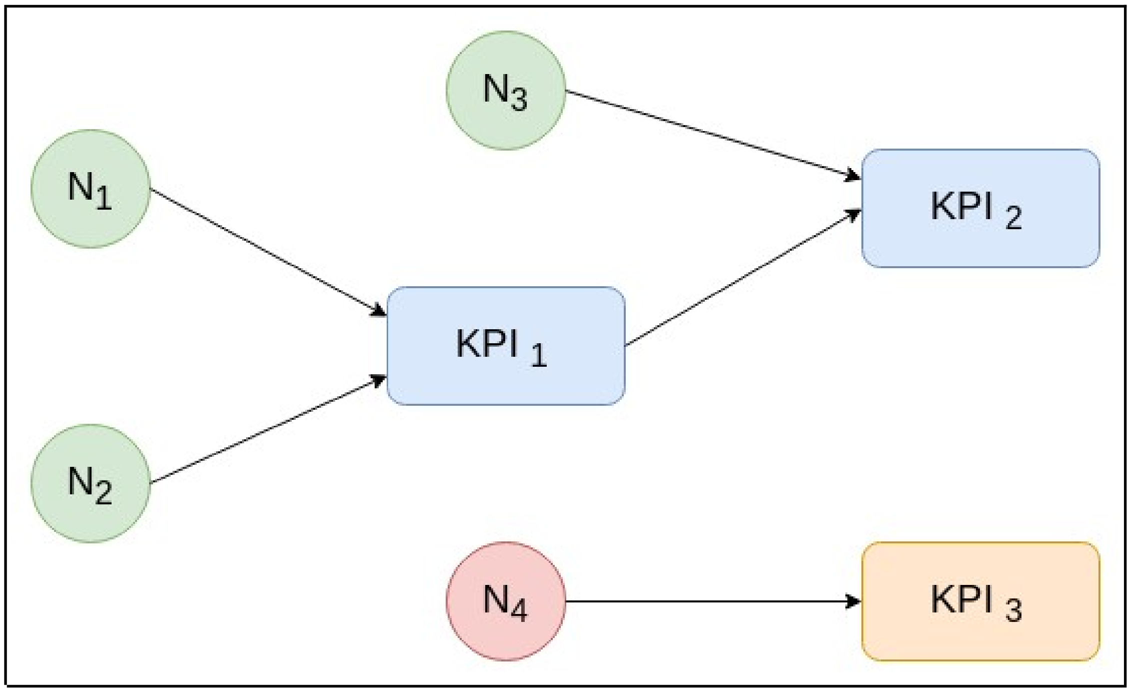

Figure 5.5: KPI Directed Acyclic Dependency Graph: N_4 Failure

6. FEATURES

Refactored Error Handling for RxJS:

- More robust error handling with errors being handled where they occur but also notifying dependent nodes down stream
- Better debugging statements logged to console with clear stack trace of error
- Fail-soft: attempt recovery on error rather than crashing entire system when node fails
- Example (See Figure 5.2 – 5.5):
    - In Figure 5.2 all nodes are in a non-error state and therefore all dependent KPIs are also valid and in no default error data is propagated downstream
    - In Figure 5.3 N_3 is in an error state and therefore this error is propagated down stream to KPI_2 but this does not stop KPI_2 entirely as N_3 sends with the error default data so that the output of the rest of the data can continue with the last valid value or the default from N_3. The same approach is found in the following figures, with yellow nodes representing a KPI aware of default data being sent from a node in an error state.

Proof of Concept (See Figure 5.1) :

- Testing environment composed of Docker Image running Mosquitto, an open source MQTT broker, and multiple Node.js clients publishing dummy data to the broker.

- An express server subscribed to our topics from the Mosquitto broker will be pulling the streaming data into KPI calculations facilitated by our Fault-Tolerant RxJS.
- A simple web interface to view the status of nodes and the output of the calculated KPI's.
- If time permits our web interface would be able to trigger the simulation of a disconnect for any specific node while displaying the log and fault tolerant response of the system.

7. CONSTRAINTS

The main constraint, as also expressed by our sponsor, is the lack of our knowledge in this specific domain. The learning curve for functional programming is steep and the difficulty is compounded when attempting to learn functional reactive programming. Our project scope also has been subtly changing in response to the sponsors insights from week to week.

Another constraint is the viability for changing the error handling of RxJS in the module it’s self or having to revert to wrapping the error handling functionality in our own module. We will work towards the former and adopt the latter approach if RxJS functionality is not conducive to the changes we would like to make.

Lastly, we are testing this in a very specialized environment with simulated conditions. We assume that IOT devices using the MQTT protocol will publish to parent topics under device identifying sub-topics. For example, a topic such as ‘Temperature’ may be considered a parent topic and ‘Temperature/source1’ along with ‘Temperature/source2’ may be considered subtopics related to source1 and source2 temperature data respectively. We also assume that all KPI calculations are not interdependent and that no circular dependencies are created in the dependency graph.

8. CONCLUSION

RxJS is a broadly used and well maintained functional reactive programming library. However, error handling is often a manual process that can leave KPI calculation dependencies in a stopped condition leading to hard fails of systems. With our addition of fault-tolerance to the error handling of RxJS, users will be better able to handle non-system critical faults gracefully in the calculation of various dependencies. There are a number of constraints including time and our limited domain knowledge but the insights and progress members make this semester can always be carried over into further progress in future semesters. Since our project is not aimed at reconstructing something already existing our goals are not concrete but rather the objectives of our project are flexible from weekly insights into the problem space.

9. REFERENCES

Mogk, R., Baumgartner, L., Salvaneschi, G., Freisleben, B., & Mezini, M. (2018). Fault-Tolerant Distributed Reactive Programming.