Elevator Pitch - MQTT Broker on Composite OS

Innovation in technology is a large driver in societal behavior changes. Within the past ten years, we've seen how the Internet of Things has found its way into our everyday lives. Our smart cars share data with manufacturers nearly instantaneously, our smart fridges know when the house residents have gone to sleep, and so on. With these innovations occurring rapidly, we must continue to optimize our implementations. The real-time systems need to continue their high computations with reduced transmission latency over their networks.

MQTT, as known as message queueing telemetry transport, is a cutting edge communications protocol that allows these real-time systems to communicate with each other more efficiently. Clients in a MQTT environment are categorized as publishers and/or subscribers. Publishers are clients who share data with any of their subscribers. Subscribers subscribe to a specific topic, listening for any new data to receive, and only receive information when there is information to be received. An integral component of MQTT is the broker; the broker provides all of the MQTT functionality by sending these communications to the appropriate clients. This creates a many to one connection between the broker and all the clients, as opposed to a many to many connection between all of the clients. In the real world, this can translate into a sensor in a smart city communicating with a passing vehicle about a sudden obstacle in the road in real time, where the car is a subscriber to the road sensors and the road sensors are publishers to all cars.

There are currently several MQTT implementations available. For a cloud-based implementation, Microsoft Azure and Amazon AWS have their own implementations available for developers to connect to their services' IoT hub devices. These cloud-based services provide C, Python, and JavaScript support. The most popular implementation is Mosquitto, which is implemented in C and is known to provide lightweight support for resource-constrained systems. However, Mosquitto's most discussed drawback is higher transmission latencies. In addition, both Mosquitto and cloud-based implementations have support for specific languages that provide heavier, costly run-times.

We propose MusQraTT, an implementation of the MQTT broker in Rust on CompositeOS. Rust is a programming language that has built-in memory safety, high performance, and safe concurrency. Rust will fill in the gaps that languages such as C, Python, and JavaScript leave in providing safe, lightweight transmission of data within real-time, resource-constrained systems. CompositeOS is a microkernel that prioritizes low latency and reliability. Building the Rust broker on CompositeOS allows us to remove system-level bottlenecks for performance and latency. Additionally, specific optimizations and modifications can be created to the broker and OS that prioritize a low-latency, safe implementation of MQTT that wouldn't be as achievable on an established OS, such as Linux.

Our goal is to achieve measurable improvements in transmission latency over popular competitors while creating a system suitable for resource-constrained applications on the Edge. This will include implementing an optimized MQTT broker in Rust, porting it to Composite, and creating a suite of tests that we can run on both our implementation and on competitors. The proposed technologies of Rust and Composite have built-in solutions to the current gaps in available MQTT broker implementations. Our project explores intersecting the two technologies to provide a low-latency and secure solution.