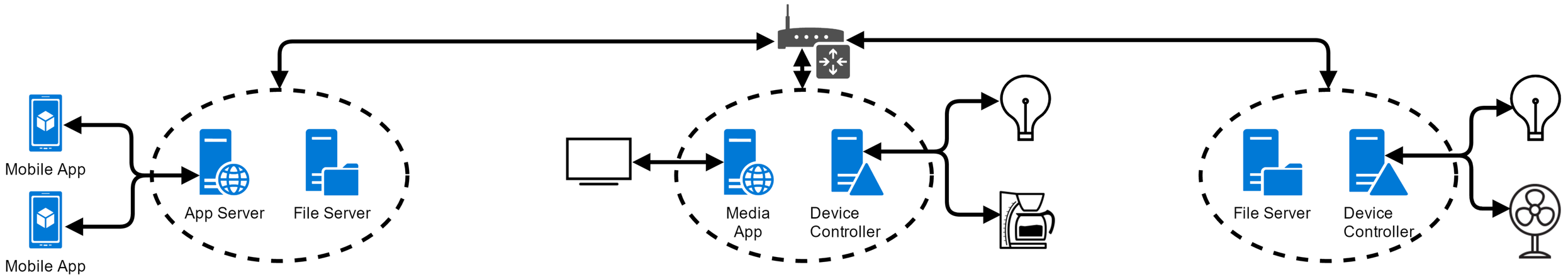
Project Molecule

Team May1739

Ryan Wade, Nathan Volkert,
Daniel Griffen, Alex Berns

Multi-Node Shared Application Environment for Smart Home Systems

Iowa State University
Advisor: Arun Somani



Overview

Problem Statement

- Smart homes are often cloud dependent which creates a single point of failure for their system
- Many existing smart home application are not interoperable with other systems

Solution

- Locally distributed to multiple nodes, reducing dependency upon cloud for operation of system
- Open source standard, with a unified application interface, allows other developers to create new functionality
- Fault tolerant by design to reduce the chance of system wide failure

Intended Users

- Enthusiasts seeking to integrate their disparate smart home systems into a single useable environment
- Business wanting to connect their product to the Molecule API

Operating Environment

- Project Molecule is designed to run on most Linux operating systems

Requirements

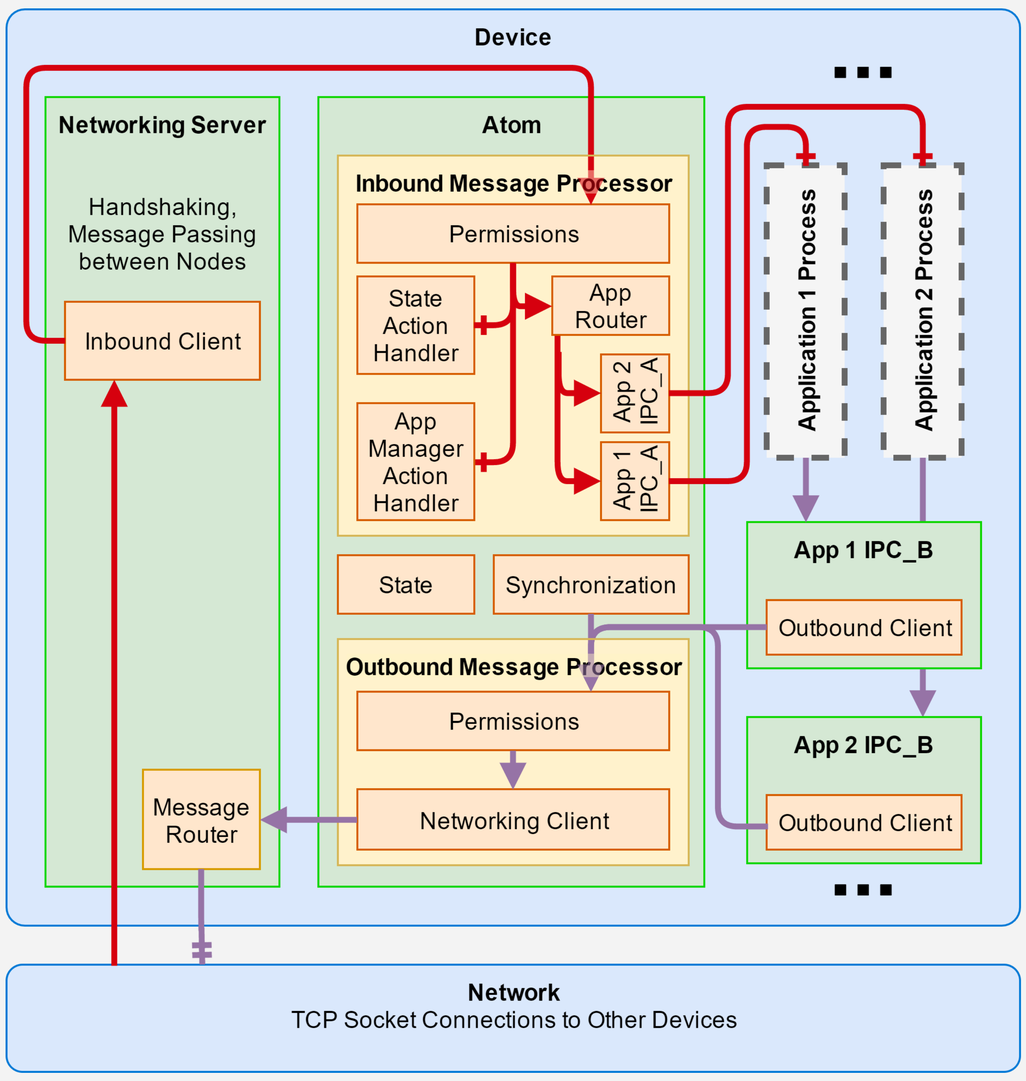
Functional Requirements

- A **Device** shall be able to host more than one **Application**
- **Applications** will be isolate from other processes on the **Device**
- **Applications** shall be able to communicate via **Messages**
- **Messages** shall contain origin, routing, **Action** (protocol), data, and stream information
- All **Messages** shall be routed by the **Device**
- An **Application** will be restricted to send/receive/broadcast **Actions** as defined in a configuration file available at system startup
- **Devices** shall synchronize configuration and user data with each other
- **Messages** shall be routed between **Devices** via a network
- A single **Device** failure must not bring down a multi-**Device** setup

Non-Functional Requirements

- Tools will be provided to create 3rd party **Applications**
- A stable API will be exposed and documented for 3rd party development
- Tools will be provided to configure and manage **Devices**

System Design



General Info

Green boxes represents a thread
White boxes represents a process
Red arrows denote messages originating outside the device
Purple arrows denote messages originating inside the device

-> right before a block designates that the message will be processed and routed by that block
┼ right before a block designates a terminal where a response is generated and sent back
┼┼ before the network layer denotes that the terminal is in another device
... denotes that an undefined number of application processes can be running at once

Client: Forwards message to appropriate server and waits for response
Permissions: Validates the source/destination/action pair of a message ensuring:
- SOURCE can send or broadcast the ACTION
- DESTINATION can process the ACTION

State Action Handler: Handles "STATE" requests from outside the Atom to create, read, modify, or delete key-value pairs in the state
App Manager Action Handler: Handles "APP_MANAGEMENT" application lifecycle actions, such as install, start, or stop, and updates the Action Routing Table in state appropriately
Router: Routes a message to its destination device/application
Application: 3rd Party executable which is isolated in a separate process
App # IPC_A: The request response Unix Socket for Atom Forwarded Messages
App # IPC_B: The request response Unix Socket for Application Originating Messages
State: Wraps a redis key-value database
Synchronization: Handles changes to state, and synchronizes them across devices

Technical Details

Language

- Rust
  - Development Language
  - Memory Safe

Tools

- Visual Studio Code
  - IDE Used by All Team Members
- Bash for Windows

Library

- Tokio
  - Provides an Async network Framework
- Serde
  - Serialize Data for Storage and Transmission
- Redis Library
  - Communication with Redis Database
- Rust-Crypto
  - Securing Network Communications

Project Management & Testing

Project Management

- Gitlab
  - Continuous Integration
  - Version Control
- Weekly Team Meetings
  - Scrum Meetings
  - Advisor Meetings
  - Collaboration Meetings
- More than 1200 Man-Hours

Project Testing

- Modularized Code
- Regression Testing
  - Automated with Gitlab
- Module Testing
  - Ensure Modules Work Before Integration
- Integration Testing
  - Applications Communicate Across Devices
  - Network Failures Handled Gracefully
  - Redundant Application Switching