Final Scientific Technical Report for Award DE-EE0006353.000

An Extensible Sensing and Control Platform for Building Energy Management

Principle Investigators
Anthony Rowe, Mario Bergés, Carnegie Mellon University
Christopher Martin, Research and Technology Center, Robert Bosch LLC

This document contains a review of progress made over year 1. There are no distribution limitations related to the contents of this document.

Contents

Acknowledgment: "This material is based upon work supported by the Department of Energy under Award Number DE-EE0006353."

1 Introduction

Although there has been considerable work done on BAS, only 10% of commercial buildings utilize advanced monitoring and controls [1]. Most of this effort has focused on large and/or new commercial facilities that are 100,000ft^2 or greater in size. The vast majority of smaller buildings (food sales, food services, warehouses, etc) perceive BAS solutions as either too expensive or too labor intensive to justify installation efforts. Incentives are often further diminished if the owner of the complex is not the tenant and/or they are not aware of the benefits automation systems can provide. To truly unleash the energy saving opportunities and maintenance improvement potential that automation and monitoring systems can offer, there is a need for a robust and scalable solution that can integrate multiple building systems in an easy-to-configure manner. Typical building management systems could include control and monitoring of HVAC, lighting, plug-loads, security and fire alarm systems.

The goal of this project is to develop Mortar.io, an open-source BAS platform designed to simplify data collection, archiving, event scheduling and coordination of cross-system interactions. Mortar.io is optimized for (1) robustness to network outages, (2) ease of installation using plug-and-play and (3) scalable support for small to large buildings and campuses.

Most traditional BAS designs rely on dedicated wired (often proprietary) bus or network protocols. These protocols connect general-purpose controllers to a wide variety of transducers and run customized local control loops and schedules. All runtime data is logged at the central coordinator which is also responsible for any global device interactions. In more recent designs, the central coordinator might expose cloud services for remote management. In contrast, Mortar.io pushes tasks like data storage, scheduling and alarm generation closer to the edge devices. This allows for continuous data collection and distributed operation even during network outages.

Mortar.io uses a publish-subscribe architecture built on the XMPP protocol, originally designed for Internet-scale real-time chat. Each transducer and its associated meta-information is stored in an *event node* that contains various items of interest (data, schedule, maps, etc). Historical sensor data is stored in a distributed multi-resolution time-series data store called *Respawn* [2]. Respawn has the ability to migrate low-resolution (aggregate) data to different nodes in the system both to improve access performance through caching and provide fault-tolerance through data replication. In order to facilitate interoperability, we define a schema that describes properties of particular transducers (meta-data) as well as a method for capturing their inputs/outputs (time-series data).

Our plug-and-play infrastructure called QuickSet uses short-range communication and Zeroconf networking between a smartphone or tablet and a BAS device to securely register and pair new peripherals with the system. Since many BAS components are behind firewalls or on VLANs, a lightweight Public Key Infrastructure (PKI) allows the mobile device to configure components wirelessly without requiring direct network access itself. The PnP protocol, called *Quickset*, supports self-powered NFC communication for installing devices in new constructions before power or networking is available. We also describe and motivate an auto-discovery and auto-mapping service that can scan for devices on a network and utilize relationships between sensor streams to aid placing devices on a map. It is important to note that auto-discovery is not the same as plug-and-play. Plug-and-play is the process of securely pairing and collecting required operational information form a device. Auto-discovery can help this process, but is not a suitable replacement for it.

To verify scalability from complex medium sized buildings down to small installations, in Section 4 we show benchmarks that indicate that our platform can comfortably operate on low-cost embedded targets (for example a $20 Sitara ARM Cortex-A8 processor from Texas Instruments) and accommodate substantial amounts of traffic on set-top box and server hardware. The core components of the system are written in ANSI standard C with hooks to languages like Python, Erlang and Javascript for higher-level tasks. Each component has clearly defined interfaces and should be portable across platforms and compatible with alternative frameworks.

2 Project Overview

During year 1, the team has developed working prototypes of all the components of Mortar.io, based on functional requirements collected by our Bosch partners. We met all of our proposed milestones. During the first six months, the team focused on overall architecture design and the details of each component guided by the results of the functional requirements analysis and insight about common building operational pain-points during our monthly conversations with DOE. After the design was finalized, the second half of the year was spent on the implementation of the different components which are detailed in Section 3.

Many non-crictical tasks were also completed during this period, including: (a) the implementation of a web-based project management system and code repository, with fully-linked documentation of APIs (`http://dev.mortr.io`); (b) the addition of Lutron as a partner, along with the integration of their Quantum BACnet-enabled lighting controller into our project; (c) hosted a workshop at CMU (`http://openbasworkshop.org`) dedicated to creating a forum for discussion about the different problems and existing solutions for Building Automation Systems (BAS).

Though a detailed review of each component and design decision is presented later, we would like to highlight three of the completed tasks, which we believe summarize the progress to date and are all currently running on the benchtop setup (Figure 1) that will be used for the site visit in October:

Quickset: To facilitate device and network configuration in a secure manner – something that is currently lacking on most BAS solutions in the market, even those with auto-discovery functionalites – we designed a PnP protocol called Quickset. Quickset uses short-range communication and Zeroconf networking to securely register new devices to Mortar.io using a smartphone or tablet as the broker. To design this piece of technology we performed an extensive review of the existing solutions in the market, and consulted with security experts. Details of Quickset can be found in Section 3.2.3. We continue to track how various emerging industry standards are converging on similar approaches. As we move forward, we intend to maintain compatibility with best-in-class approaches. The Quickset architecture is modular and can be easily ported to vendor-specific platforms.

Respawn: To improve the reliability of historical data storage and the system's robustness to network outages, we integrated and adapted an existing distributed multi-resolution time-series data store called Respawn [2]. When integrated into our PubSub framework, this solution significantly reduces query latencies and provides replication to cope with potential network outages. We have fully integrated Respawn into Mortar.io, and have tested its reliability and performance with our benchtop setup. More details on the design choices and the technology can be found in Section 3.2.1.

Auto-Mapping: Although there are many ways to perform device discovery through existing protocols, a good portion of the meta-data associated with devices in the system is either unavailable, outdated or

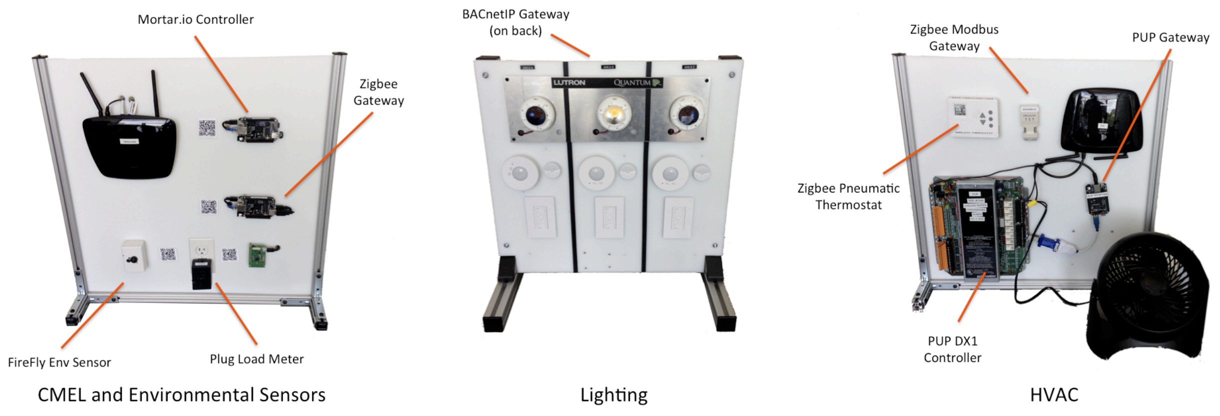


Figure 1: Three initial target systems on benchtop setups.

unverifiable. To solve this problem, we have designed a preliminary solution based on statistical analysis of historical measurements from different devices, seeking to estimate the dependence between these streams and use this information to infer pieces of meta-data that the correlated (or dependent) devices may have in common. This approach to auto-mapping is different from simple device-discovery, and we believe it has the potential to significantly improve the initial configuration of Mortar.io systems by making it more efficient, as well as to assist in the maintenance and upkeep of any installed system by keeping track of the inferred dependencies and flagging any anomalous trends that may indicate inconsistent meta-data. More details on our auto-mapping solution can be found in Section 3.3.2.

3 System Design

Mortar.io is built to scale from a single home or small business to large campuses. This is achieved through the implementation of a distributed system where each component has limited scope and functionality. Asynchronous communication via XMPP as well as an IPC Daemon provide a scalable, reliable, and easy-to-use communication architecture.

Figure 3 depicts an example Mortar.io deployment in a small buisness. Two hardware components central to the Mortar.io system are the controller and the gateway. The controller hosts the XMPP server which provides asynchronous inter-device communication. The controller may also host any Mortar.io services that may be used by multiple gateways. The use of XMPP enables the controller to be deployed on a small embedded server or be distributed over a cluster of high-end servers, according to the requirements of the installation.

Gateways connect Mortar.io to a building's infrastructure. A single gateway may interface with more than one system protocol. The software processes that bridge the different technologies are called adapters. Similar to the XMPP server on the controller, local instances of an IPC Daemon enable the passing of messages within a gateway's processes. This allows adapters to achieve better performance and maintain operation when during network failures.

Users interact with Mortar.io using either a mobile or a web interface. The mobile interface provides a simple, easy to use plug-and-play deployment tool for assisting in the installation and configuration of equipment, with a technique called Quickset (details in Section 3.2.3).

Figure 4 presents the software architecture of Mortar.io. The lowest Mortar.io layer is the Driver Layer, where we find the components that adapt information about devices on a wide range of protocols to the Mortar.io schema and then to the IPC message router. Adapters translate system specific information into the Mortar.io schema and distribute them using the IPC Daemon interface to the XMPP server. Adapters produce XMPP messages in response to events or readings from elements of the BAS. It is also capable of actuating BAS elements upon receiving an XMPP message. Mortar.io currently includes adapters for BACnetIP, ModBUS (RTU and TCP), ZigBee (for the FireFly WSN) as well as for NEST thermostats, Phillips Hue and the Inscope Enfuse energy metering system based on RESTful API. Drivers are run on Gateways, which may need special hardware interfaces for communication with specific building technologies. For example a serial connection to a bus, an ethernet connection to a separate network, or an 802.15.4 radio.

The Transport Layer provides asynchronous messaging services used by all other system components. XMPP enables inter-device communication. The IPC Daemon, run on each device, expedites communication among intra-device components while enabling operation to continue when the XMPP server is unreachable. It is used in gateways to allow other services to receive or intercept the messages of an adapter. Scalability is provided by XMPP federation mechanisms, which allow the server to scale with the deployment. Inherent to this message passing is the Mortar.io schema, detailed in Section 3.1.2, which is necessary to describe device and system information. The schema allows for standard descriptions of schedules, device meta-information, mapping information, actuation and data updates.

The Service Layer contains the components which build on top of the Data Layer and provide services used by applications or for management. The Quickset plug-and-play service supports the mobile application

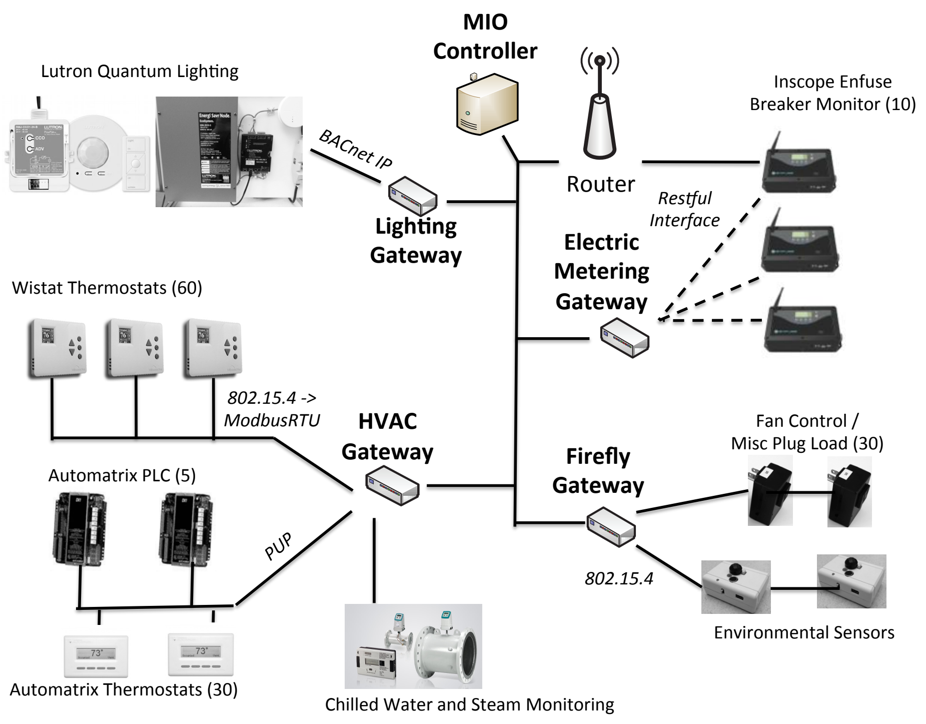


Figure 2: Scaife Hall Deployment Overview

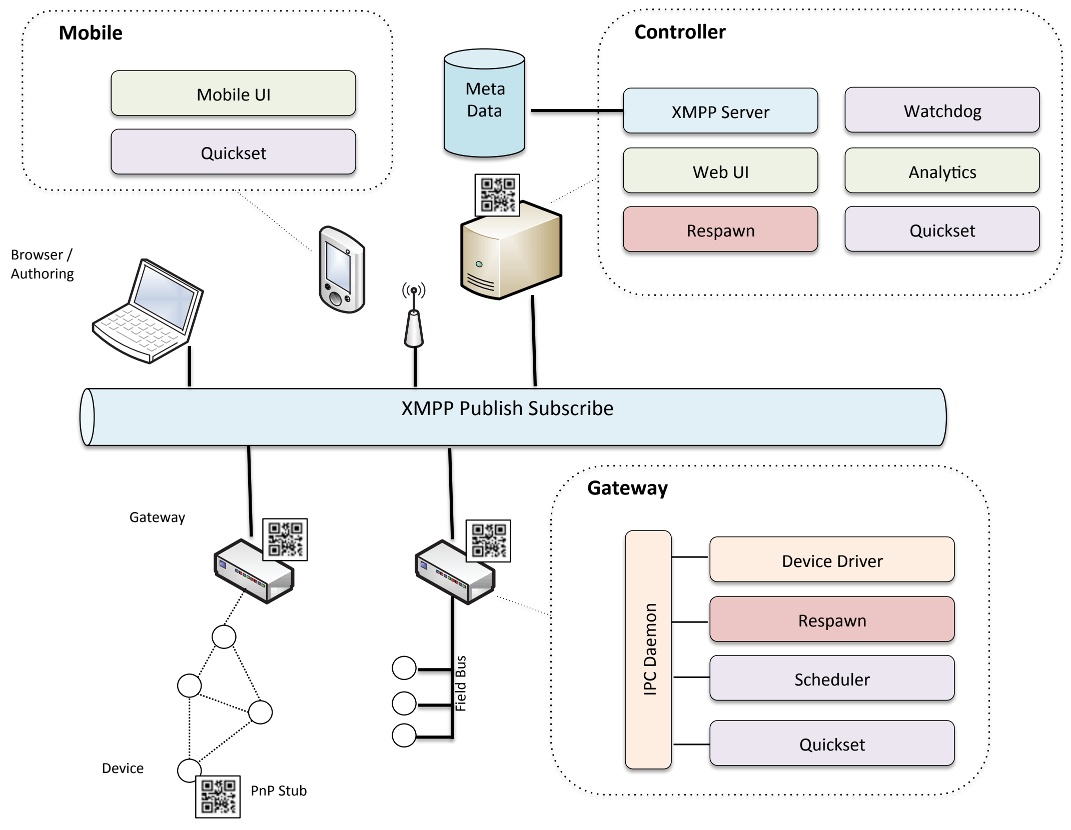


Figure 3: Mortar.io Overview

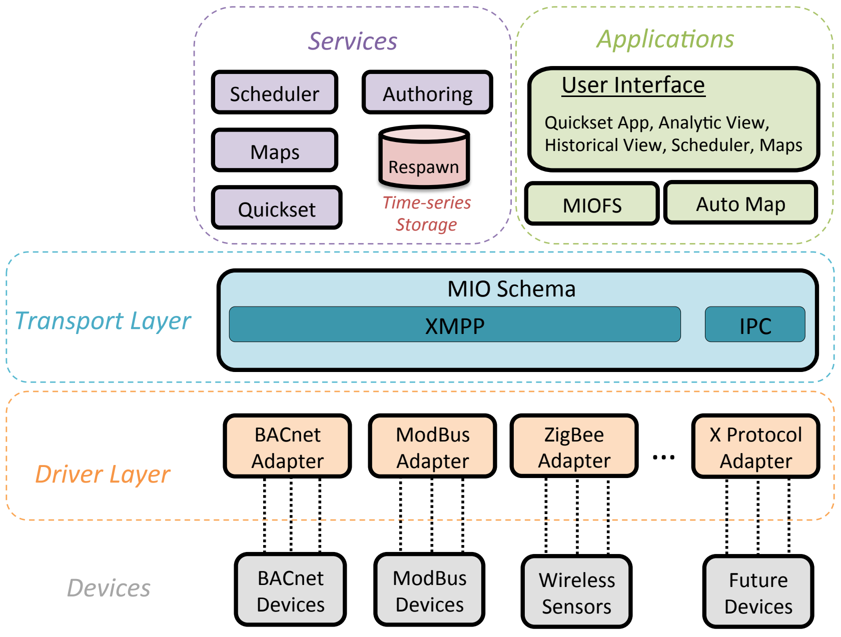


Figure 4: Mortar.io Software Architecture

during the installation of Mortar.io in a building, automating complex network and driver configuration tasks. Support for control automation is provided by the Scheduler service. The Maps service provides mapping information for each BAS element, giving users easy to understand logical or physical location information. Respawn, a time-series database is used to provide access to historical data, as XMPP will only retain the last few events for each sensor. The authoring service aids users in classifying and providing data for each sensor in large deployments. Further services exist or may be added in the future, though they are out of scope of this paper.

The Application Layer encapsulates userspace utilities. Applications may build only on top of the Transport Layer or also make use of components from the Service Layer. A locally hosted website provides the primary UI, described in Section 3.3.1. This provides an interface to the Schedule Editor, Analytics Engine and allows Remote Management. The Mobile App provides the same functionality as the website, while also providing an interface for the Quickset plug-and-play application, which enables effortless device installation and configuration. The Auto Mapper App was developed to facilitate the configuration of large deployments, assisting in determining the location of each device by detecting colocated devices through the correlation of events. MIOFS provides a file system abstraction for sensors and actuators based on FUSE.

3.1 Transport Layer

All remote messaging uses XMPP. As an optimization and to support local communication during network outages, we also have an IPC Daemon that runs on each local gateway.

3.1.1 XMPP PubSub

Mortar.io is built on a publish-subscribe messaging pattern, which runs as part of the XMPP messaging platform [3]. Data is published to an XMPP server and then distributed to subscribed entities. Fine grained access control is also provided by XMPP. More details about the different system entities, depicted in Figure 5, are given below.

Event Nodes: An event node is an address (or topic in similar systems) to which authorized users in

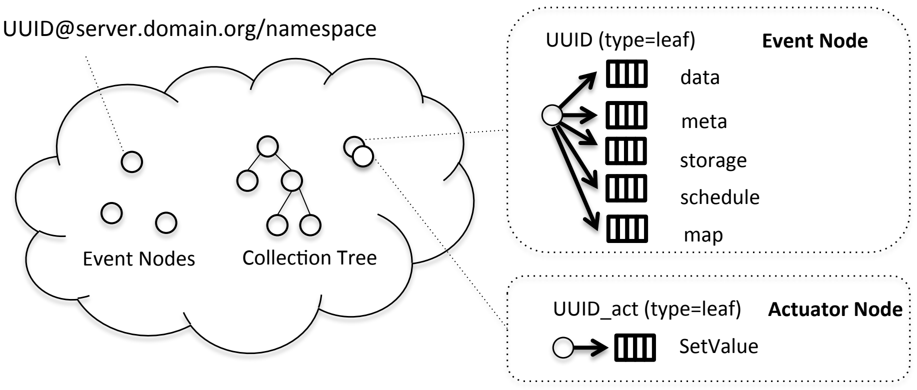


Figure 5: Publish-Subscribe Architecture

the system can publish and subscribe. For example an event node representing a sensor feed from a weather station would receive publications from the station's adapter, to which interested users can subscribe. The XMPP server stores and manages all existing event nodes of a given domain, receives publications and forwards them to the appropriate subscribers. Each event node is identified by a UUID, which is generated by Mortar.io upon creation.

Entities may organize their data in the form of items (subtopics), which can be stored and retrieved individually from event nodes. The weather station node may, for example, contain an item for each of its transducers one for the outside temperature, one for the outside barometric pressure, etc. Items may also contain meta-data such as the location of the device associated with the node, its manufacturer, model number, etc.

Mortar.io allows for the actuation of devices through actuation event nodes, which complement a device's standard event node. An actuation node contains an item for each actuating transducer of the associated device, addressed using the same UUID as the devices event node with "_act" appended. This decoupling of event and actuation nodes allows separate access control for device monitoring and actuation.

Collection Nodes: A collection node contains a list of event and/or collection nodes. It is a construct used to hierarchically group nodes for organization and access control purposes. For example a collection node may represent a geographical region such as a room in a building. Users may directly subscribe to collection nodes to receive publications from all child event nodes.

Access Control: Access control is handled on a per-node basis by XMPP. Each event node may have a list of users affiliated with it, including owners, publishers and subscribers. The node owner is the manager of the node and can configure which users are granted publish and subscribe access to it. Users need to be granted publish access individually by an owner, after which they can publish data (including meta-data) to the node.

Mortar.io employs the roster-based node access model for subscription management. It allows the creation of different user rosters (groups) to grant subscription privileges to nodes. For example a "guest" roster may allow users of that roster to only subscribe to a few sample nodes in the system, while an "administrator" roster allows for subscription to all nodes.

XMPP Servers: Many open-source and proprietary XMPP servers exist, varying significantly in their supported feature set. We have internally evaluated several of them and have settled on Ejabberd [4] since it implements all necessary XMPP features, provides good performance, is platform-agnostic and open-source.

Scalability is achieved by federating servers across domains. Server federation is an integral feature of most XMPP servers, which allows inter-server communication for sharing nodes, publications and subscriptions across domains.

3.1.2 Mortar.io API

LibMIO: Central to Mortar.io is LibMIO: A library providing the API to the Mortar.io system. LibMIO is an open-source, multi-threaded, C library, containing an XMPP messaging engine. It implements the Mortar.io schema, providing procedures ranging from server connection management, node creation/management, subscription management, to publication transmission and reception are included. Easy to use command line tools for most functions are also provided. The library can be compiled for several platforms including Linux, Windows, OSX and iOS. Porting LibMIO to other languages is planned for the near future. LibMIO also encapsulates the schema used to describe device and system information.

Mortar.io Schema: Mortar.io defines an XML based semantic structure for device meta-data and transducer values. The goal of this is to support the description and data associated with arbitrary device and transducer configurations.

The schema defines a device as a physical unit housing one or more transducers, such as a thermostat. Transducers are sensors or actuators that produce and/or consume streams of data and are associated with a device, such as the temperature or the set-point of the thermostat.

A high-level overview of the main components of the schema describing a thermostat can be found in Figure 6. The *meta-data* is stored in an event node's item titled 'meta' (all pre-defined items are shown in Figure 5). *Device* stanzas are the highest level of abstraction in the meta item and under them there can be any number of *transducers*. The *name* field provides human readable names and types of devices. It is also possible to include a geolocation stanza for the device, as well as the manufacturer and serial number of the device [5].

The *Transducer* item stores the human readable name of the transducer. A transducer is uniquely identified by "nodeid_name" and name must be unique within a device. The schema also provides the optional ability to specify the location, the manufacturer, and serial number of a transducer.

Unit provides a description of the values a transducer can publish to its 'data' item and gives the physical explanation of what that value represents. It requires a unit description from a list of allowed units. An enumerated unit type is possible and the Enumerated list may be specified in the enumeration sequence of properties. If the unit is a scalar it is also possible to specify attributes to expect from that scalar, including min and max values along with the rated resolution, precision, and accuracy of the sensor.

Data stanzas are published to the item entitled data. These stanzas hold the transducer values for the device. The name field identifies the transducer that generated the data, while the value attribute contains the value of the transducer's data stream at the time present in the timestamp. If a preprocessed value is available it may also be included.

3.1.3 IPC Daemon

The Mortar.io IPC Daemon, illustrated in Figure 7, provides a process management interface as well as an intra-gateway communication router. This provides the ability for services on a local gateway to continue operating during a network outage and reduces the processing overhead of maintaining a full XMPP session for each local component. The IPC acts as a single point of communication to the XMPP server which reduces the parsing overhead required to route each packet to a specific component running on a local gateway. It then provides an interface for adapters and services to listen for specific event nodes and items.

The IPC is also responsible for launching adapter, service, and application processes. A process watchdog is spawned to routinely check for the liveliness of all these processes, restarting them if necessary. It further manages their configuration, such as defining which processes should receive what data from what event nodes, actuation requests, and which items are relevant. This provides a route for local communication and therefore local communication is immune to network and internet outages.

The IPC Daemon setups and maintains NTP, firewall settings, and Zeroconf service registration for

Meta Item

```
<Device
        name="WISTAT_SH214"
        type="Thermostat"
        timestamp="2014-07-15T13:13:21.318744-0400"
        serialNumber="071.005">
        <Transducer name = "Temperature">
                <Unit units="Celcius"minValue="0"maxValue="50"/>
        </Transducer>
        <Transducer name="Set Point">
                <Unit units="Celcius" minValue="10" maxValue="25"/>
        </Transducer>
        <Transducer name="Line Pressure">
                <Unit units="Pascal" minValue="0" maxValue="30"/>
        </Transducer>
        <Transducer name="Mode">
                <Unit units="Enum"
                        <Map name="Cool" val="0" />
                        <Map name="Hoeat" val="1" />
                        <Map name="Standby" val="2" />
                />
        </Transducer>
</Device>
```

Data Item

```
<Data name="Temperature" value="72" timestamp="2014-07-15T13:13:21.318744-0400" />
<Data name="Set Point" value="70" timestamp="2014-07-15T13:13:21.318744-0400" />
<Data name="Line Pressure" value="8" timestamp="2014-07-15T13:13:21.318744-0400" />
<Data name="Mode" value="0" name="Cool" timestamp="2014-07-15T13:13:21.318744-0400" />
```

Figure 6: Example Thermostat Messages

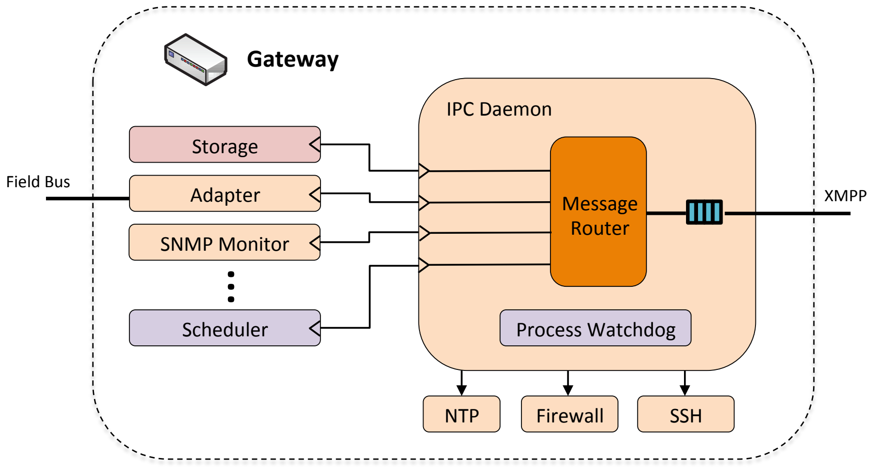


Figure 7: Gateway IPC Architecture

serverless messaging [3]. It also enables SNMP and starts an SNMP adapter, which provides machine usage statistics.

3.2 Services

Services provide functionality to the devices on the Mortar.io system. They typically subscribe to and/or actuate devices. Many of the services run locally on the gateway to provide as continuous a service as possible, thus relying heavily on the local IPC communication. We now describe in more detail some of the core services in Mortar.io:

3.2.1 Respawn Datastore

Respawn is a distributed datastore capable of serving large volumes of time-series data from a continuously updating datastore with access latencies low enough to support interactive real-time visualization. Respawn targets sensing systems where resource-constrained edge devices may only have limited or intermittent network connections linking them to a backend. Data is downsampled as it is ingested, creating a multi-resolution representation capable of low-latency range-base queries. Low-resolution aggregate data is automatically migrated for improved reliability and load management. Edge nodes automatically identify and migrate blocks of data that contain statistically interesting features. Respawn is able to run on ARM-based edge devices connected to a cloud-backend with the ability to serve thousands of clients with sub-second latencies.

Respawn was originally introduced in [2] along with data migration as a technique for reducing query latencies. In contrast, Mortar.io employs Respawn migration to improve the reliability of historical data storage and robustness to network outages. Figure 8 outlines two possible configurations for Respawn in a Mortar.io deployment. In the first configuration (a), three gateways store raw data locally and simultaneously act as XMPP clients, publishing down-sampled copies of their data. These aggregate copies are stored by the building's local XMPP server as a secondary location for servicing queries. The second configuration (b) is identical to the first, with the exception that the XMPP data streams are stored in three locations, yielding triple-redundancy for reliable storage.

3.2.2 Event Scheduler

The role of a scheduler is to subscribe to the schedule item of an event node and then publish actuation commands for scheduled events. The format for these schedules is XCal, the text in the event field contains

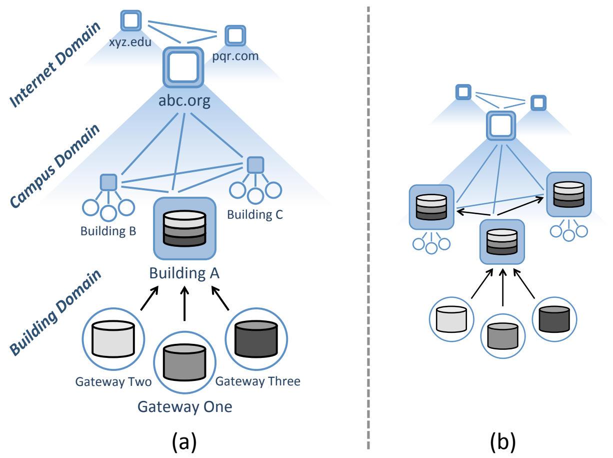


Figure 8: Respawn Distributed Storage with a Single Replica (a) and Triple-Redundancy (b).

the actuation command to publish at the prescribed time. The schedules are generated at the controller (through a web interface) or on a mobile app, but are then dispatched directly to the gateway device using the meta pubsub node. The schedule is then run locally on the gateway so that it can continue operation during network outages. XCal is expressive enough to capture a rich set of calendar events such as one-time events, weekly , monthly , yearly and overide events. Each of these schedules is stored in the scheduling item of each device's event node. The last update to each node is what the scheduler executes at the next time interval. For this reason, conflicts must be resolved at a higher-level. For example, in the scheduling interface the system should alert users if they are overiding an already defined schedule. The final schedule must be stored in each event node, so all information is available to any agent or user that has correct access permissions. Commands available for each devices are defined in its meta node and can be easily accessed by user interfaces.

3.2.3 Quickset Plug-and-Play

Installation accounts for a large portion of costs associated with adopting a BAS. Quickset empowers non-technical users by facilitating device and network configuration in a secure manner. The user interface for Quickset is provided through an App running on a mobile phone or tablet equipped with either bluetooth or NFC. The proximity communication (being able to talk from the tablet directly to the device) relaxes the restriction of being on the same subnet as the device being configured. This is advantageous when the device is being installed in a private network or is not reachable via Wi-Fi.

Each Mortar.io enabled device constains what is called a *stub* that will be provided by the device vendor. The stub is a visual (QR) or digital (BLE/NFC) code that contains public key information, a link to the device's meta data description, the UUID of the device and an extension field that provides an option for device specific data that the manufacturer may want to include to aid in installation.

Controllers and gateways connected to the Mortar.io system are tagged with a private/public key included in the stub. The public key is made available via a QR code and bluetooth (or NFC) are used as the local channel. By using the device's public key, a secure channel can be established over Bluetooth or NFC, over which device configuration can be performed. A benefit of using NFC is that much of this configuration could be done at once, with the NFC enabled devices powered off. We now describe the Quickset transactions for controllers, gateways and devices.

Controller: The XMPP server resides on the controller. The configuration of the controller occurs over a secure local channel. The controller comes with a stub containing a public key, present on the QR code, and the public key of the installer, set upon installation of the Operating System. The controller and the installer verify each other by encrypting their messages with the public key of the controller when the installer sends communications, and with the shared private key when messages are sent from the controller.

It is then necessary to configure the network settings. If enabled on the network, Zeroconf provides a way for the Controller to broadcast the XMPP service domain and port number over DNS-SD. Once the two have verified each other it is necessary for the Controller to generate a shared private key that the installer will use when setting up the gateway. Once a network connection is established the installer sets the XMPP server credentials and domain. An easy to use configuration interface for the server is made available on the local channel. The transactions for this process are detailed in Figure 9. Figure 12 shows example screenshots of the Quickset app running on a phone.

Gateways communicate via the XMPP server on behalf of adapters and services. A local IPC Daemon manages interactions between adapters and services running locally on the gateway. Quickset begins the IPC Daemon once the network configuration of the Gateway is set and confirmed. A registration process will also begin which gives Quickset an interface to register new devices and allows the installer to view what adapters and services run on the device.

The configuration of the gateway occurs over the local channel. The network is configured as in the controller. Once on the network, the gateway will find the XMPP server if Zeroconf networking is available. If Zeroconf is not available its domain will also need to be configured over the local channel. The gateway

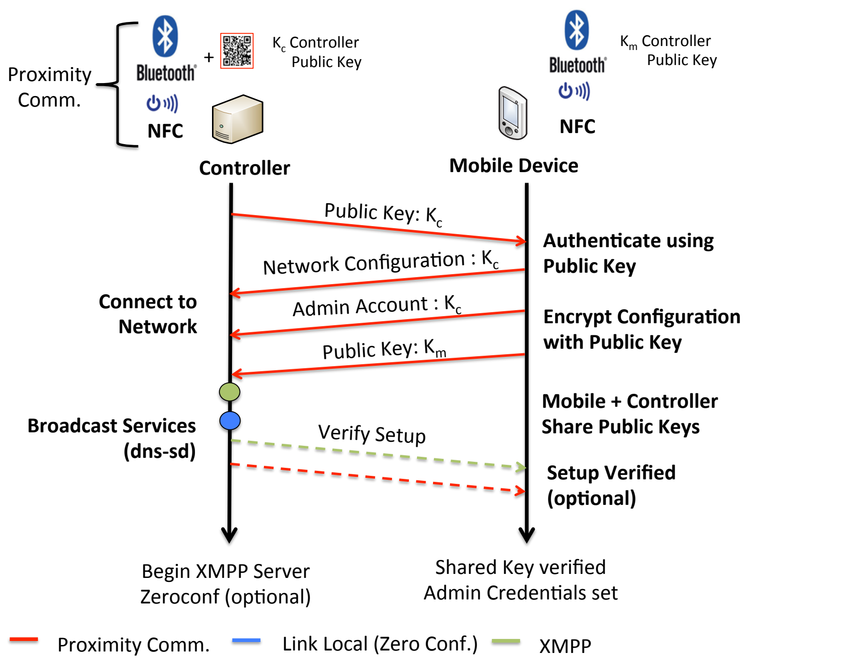

Proximity Comm.
Bluetooth
NFC
+
K_c Controller Public Key
K_m Controller Public Key
Controller
Mobile Device
Public Key: K_c
Authenticate using Public Key
Network Configuration : K_c
Connect to Network
Admin Account : K_c
Encrypt Configuration with Public Key
Public Key: K_m
Mobile + Controller Share Public Keys
Broadcast Services (dns-sd)
Verify Setup
Setup Verified (optional)
Begin XMPP Server Zeroconf (optional)
Shared Key verified Admin Credentials set
Proximity Comm.
Link Local (Zero Conf.)
XMPP

Figure 9: Quickset Plug-and-Play Transactions for Adding a Controller

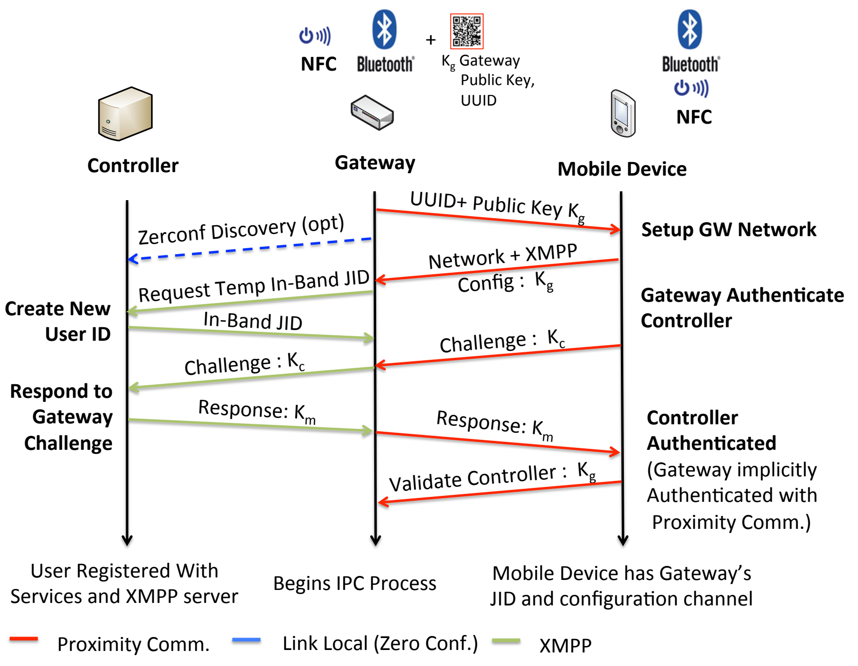

NFC
Bluetooth
+
K_g Gateway Public Key, UUID
Bluetooth
NFC
Controller
Gateway
Mobile Device
UUID+ Public Key K_g
Setup GW Network
Zerconf Discovery (opt)
Network + XMPP Config : K_g
Request Temp In-Band JID
Gateway Authenticate Controller
Create New User ID
In-Band JID
Challenge : K_c
Challenge : K_c
Respond to Gateway Challenge
Response: K_m
Response: K_m
Controller Authenticated (Gateway implicitly Authenticated with Proximity Comm.)
Validate Controller : K_g
User Registered With Services and XMPP server
Begins IPC Process
Mobile Device has Gateway's JID and configuration channel
Proximity Comm.
Link Local (Zero Conf.)
XMPP

Figure 10: QuickSet Plug-and-Play Transactions for Adding a Gateway

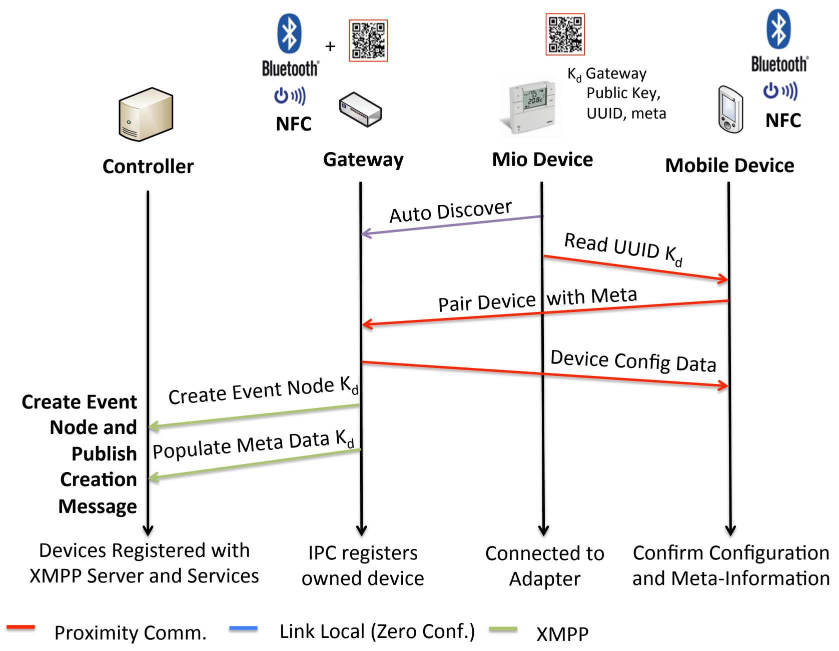
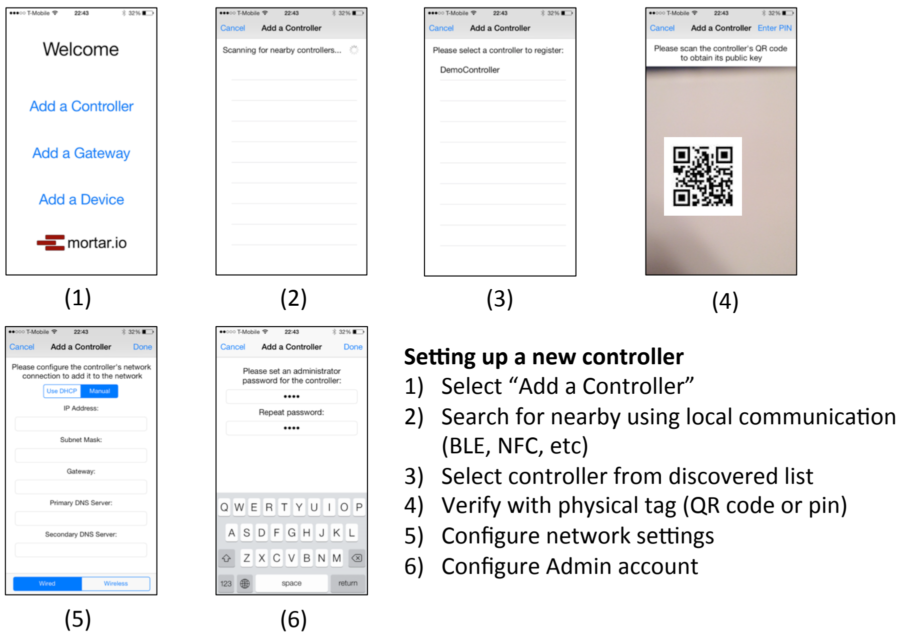


Figure 11: QuickSet Plug-and-Play Transactions for Adding a Device


Welcome
Add a Controller
Add a Gateway
Add a Device
mortar.io
(1)
Cancel Add a Controller
Scanning for nearby controllers...
(2)
Cancel Add a Controller
Please select a controller to register:
DemoController
(3)
Cancel Add a Controller Enter PIN
Please scan the controller's QR code to obtain its public key
(4)
Cancel Add a Controller Done
Please configure the controller's network connection to add it to the network
Use DHCP Manual
IP Address:
Subnet Mask:
Gateway:
Primary DNS Server:
Secondary DNS Server:
Wired Wireless
(5)
Cancel Add a Controller Done
Please set an administrator password for the controller:
Repeat password:
(6)

Setting up a new controller
1) Select "Add a Controller"
2) Search for nearby using local communication (BLE, NFC, etc)
3) Select controller from discovered list
4) Verify with physical tag (QR code or pin)
5) Configure network settings
6) Configure Admin account

Figure 12: QuickSet Plug-and-Play Mobile Application

will then connect to the XMPP server using its Public Key as a username. To verify the server and Quickset registration agent, the administrator transmits a shared secret to the gateway, which the gateway transmits to the controller. The gateway then transmits the controller's response to the configuration device. With this the configuration device can confirm or deny the controller. If confirmed, the gateway begins the IPC Daemon and connects to the XMPP server. This process is illustrated in the center diagram of Figure 10.

Device Registration: It is then necessary to add end devices. If manual configuration is required for the adapter, Quickset gives access to the adapter's configuration. If autodiscovery is an option, the user can direct the adapter to autodiscover devices in that network. The end devices are then tagged with a public key, their meta-data, and their UUID. In both cases event nodes are registered with the XMPP server through the IPC Daemon. The installer can then configure the meta-data of the device. Figure 11 includes more information about this transaction.

3.3 Applications

In this section, we discuss a few of the built-in ”user-level” applications provided by Mortar.io. In many cases, these are services like Auto-Mapping and Auto-Discovery that are core functions in Mortar.io, but are defined as user-level since they are built ontop of LibMIO functions.

3.3.1 User Interface

Mortar.io features a web-based user interface for registration, management and actuation of transducers. The interface is completely state-less, reading node configuration data directly from XMPP nodes using the Mortar.io API. This enables the system to be highly scalable across multiple domains, allowing the user interface to run on a local or remote web server, decoupled from the XMPP backend.

Most API calls from LibMIO are supported via the user interface, ranging from the creation of event nodes, configuration, actuation, to placement on a map to indicate the location of the associated physical device. Live plotting of incoming sensor data is supported using XMPP over BOSH and historical data views are provided by Respawn.

3.3.2 Auto-Discovery and Auto-Mapping

For field-bus devices that may not support QuickSet, Mortar.io provides two types of services designed to simplify discovery and configuration. The first is a port-scanning utility that can be used with protocols that provide query-able device description data. One common example of this is BACnet's *Whois* message or many of the set top boxes that scan for the Philips HUE gateway. Essentially, this service allows Mortar.io to discover any existing devices on known field-bus protocols present in the building. However, this approach falls short of inferring all the necessary meta-data for effectively utilizing each device. For instance, information about the location of each device within the building would be missing (or outdated) in most field-bus protocols.

The second approach to simplifying configuration and setup is an *Auto-Mapping* service. The Auto-Mapping service tries to logically and spatially cluster unlabeled data streams based on the historical measurments alone. This service would run in the background and perform a continuous correlation analysis between all the different streams in order to discover the statistical dependencies that exist between them and use these dependencies to make inference about the relative (or absolute) location of the devices within the building. For instance, sensors that are co-located would inevitably have correlated responses due to common physical phenomena that are affecting them (e.g., temperature). Besides inferring location meta-data, the auto-mapping service also uses the learned correlations to populate other meta-data items between devices that have similar responses. Lastly, this service can also detect faulty situations by tracking the correlation values over time and signaling whenever there is a significant deviation.

From an academic perspective, there is little past work on the problem of identifying the location of sensors in a building directly from their measurements. Some researchers have focused on auto-generating room connectivity graphs from sensor data [6], discovering co-located sensors using non-parametric feature transformations and correlation analysis [7], as well as generating blueprints [8] through various heuristics. Similarly, discovering relationships between two or more information models by analyizing the similarities in topological relationships between components contained in them, has been a succesful approach to map control points and building systems [9]. However, these approaches have been developed primarily for residential buildings, which generally have simpler floorplans and less complex building systems, and it is not yet known how well they will generalize to more complex environments, especially given some of the assumptions that support them (e.g., single-storey buildings).

In order to test whether statistical dependence measures between sensor values can be used to identify functional and spatial relationships, measurements obtained from the test facilities will be analyzed across different time scales. A statistical correlation analysis framework, based on copula theory, will be developed and tested using these datasets, and the results will be validated by comparing the inferred relationships with those extracted from the building blueprints and interviews with facility managers.

Prior work by our team on this topic [10] suggests that a simple linear correlation measure (i.e., Pearson's) is not sufficient to uncover the spatial location of sensors in a building. One can extend this by using an information-theoretic measure such as Mutual Information to estimate the dependence between the random variables being collected. Mutual information between two random variables X and Y is defined as $I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log \left(\frac{p(x,y)}{p(x)p(y)} \right)$ where $p(x,y)$ is the joint probability distribution of X and Y, and $p(x)$ and $p(y)$ are the marginal distributions of these two variables, respectively.

A more formal and expressive framework to solve this problem is Copula Theory [11], which allows one to separate marginal distributions of random variables from the dependence strucutre that defines their joint-probability density. This is of particular importance in this domain, since estimating densities directly from i.i.d. high-dimensional samples from sensors will suffer from the curse of dimensionality [12]. Identifying the marginal distributions of the components of the copula can be performed using traditional density estimation techniques. However, for identifying the copula function, there are many different options and this task will involve an analysis of the performance of different parametric and non-parametric copula inference techniques. In years 2 and 3 we will investigate how this novel copula-based framework can be applied to study the dependence between sensed stimuli in the buliding.

The final technique we plan to test for Auto-Mapping is using active control to trigger responses and learn the dependence structure in an active way. This approach is often used in the lighting industry to toggle overhead lights in order to verify the position of in-room light-level sensors. The same principle can be extended to HVAC and plug-load sensors.

3.3.3 Watchdog

The watchdog is an application responsible for monitoring the liveliness of a node based on its publication rate. When subscribed to a particular node, it will trigger an external event such as an email notification, or XMPP publication if too few publications are being made. It can be configured to determine the average publication rate of a particular node to set its activation threshold or a user supplied value may be used. The watchdog provides an additional layer of monitoring on top of the process watchdog mentioned in 3.1.3.

4 System Benchmarks

The two most crucial components in Mortar.io that define its compute and memory requirements are the underlying publish-subscribe server and the time series datastore. In this section, we benchmark both components on a variety of representative hardware platforms.

4.1 PubSub Benchmarks

While XMPP Server Class clusters have been effective at handling large deployments with hundreds of thousands of users, we are interested in what minimal requirements there are for smaller scale scenarios. Buildings that require a larger scale deployment than discussed bellow generally have the resources to invest in desktop class platforms and may already have some server infrastructure in place.

The benchmark setup shown in Figure 13 consists of S subscribers P publishers and a N event nodes. The size of the message payload is also parameterized and in these experiments is approximately 1.5KB. The publishers publish as many messages as they can and the subscribers listen for those packets. The figures show the throughput in number of messages per second. The size of the message was also scaled and tested but did not affect the throughput as the CPU is the bottleneck. In a small building there is generally thermostatic and CMEL control along with environmental sensors. For these systems with less than 500 deployed devices we propose the use of a BeagleBone Black embedded linux board from Texas Instruments. These are low-cost, $45, embedded Linux computers with 512MB of RAM, a 1GHz ARM Cortex-A8 processor.

The throughput is shown in Figure 14. As can be seen, the number of subscribers has much less of an impact than the number of event nodes and number of publishers. In these small buildings it is reasonable to assume the number of systems supported will be small and that at most around 10 publishers will be necessary. Each of these publishers could adapt (manage) hundreds of devices. We assume a global update rate of 1 message per minute is required on average by each event node. A throughput of 17 messages a second are necessary to maintain this system, plus a few messages for asynchronous operations. Easily met with 50 messages/second.

In medium sized buildings there may also be rooftop units and more complicated HVAC control. Here building management may become more complex. Here it is reasonable to have a couple hundred users. Here small computers can be used to run the system. We took the next set of benchmarks on an Intel NUC, this device has a Intel i5 1.8GHz processor, 8GB of RAM, and 120GB of solid-state memory. This setup cost around $400.

The throughput is displayed in Figure 15. As can be seen, the number of subscribers has much less of an impact than the number of event nodes and number of publishers. In these medium buildings, it is reasonable to assume the number of systems supported will increase from the scenario of a smaller building. Each publisher could manage hundreds of devices. Assuming the order of devices is 10,000 and a global update rate of 1 message/minute. This means a minimum requirement of 170 messages/second for a system of 10,000 devices is easily met by the 300 messages/second throughput.

It is important to note the trends that exist in Figure 14, with fewer then 1,000 event nodes there is a decrease in performance as the number of event nodes increases. In the Figure 15 the performance increases as event nodes are added. This is due to the server being able to benefit from improved cache locality.

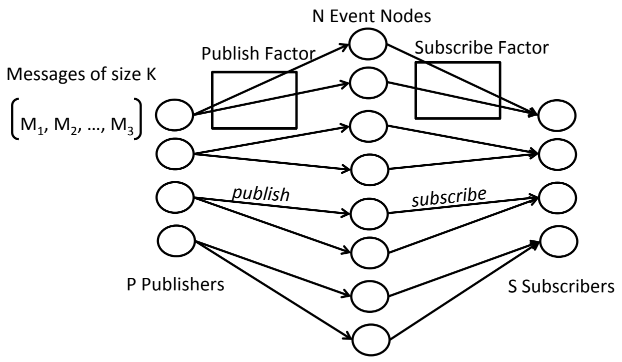


Figure 13: Benchmark Parameters

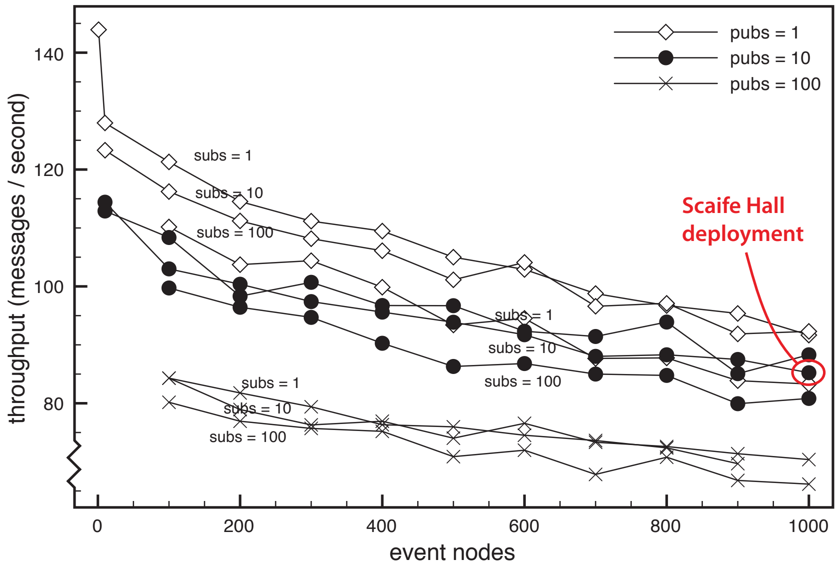


Figure 14: PubSub Benchmarks on TI BeagleBone Black

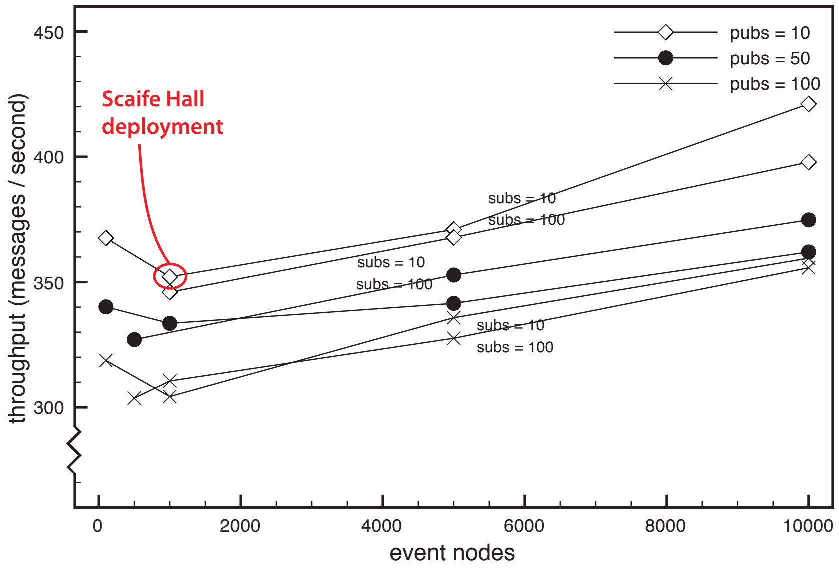


Figure 15: PubSub Benchmarks on Intel NUC

As publishers publish close together in time, their temporal locality improves the cache performance of each communication session. At around 1,000 nodes this effect outweighs the extra overhead associated with storing larger datastructures on the server. Eventually at an extremely large number of event nodes (100K+) the performance begins to decrease as expected. We omit this to more clearly highlight the difference between the NUC and BeagleBone. The scaling trend matches well with the expected usage scenario of Mortar.io. In general, there will be few publishers in charge of publishing transducer values for larger field-bus networks, of which easily tens to hundreds of devices could be attached. We see that performance is not greatly affected by the number of subscribers.

4.2 Respawn Benchmarks

Respawn has two main design features: (1) highly-efficient multi-resolution storage for timeseries data on an embedded target and (2) the ability to migrate data to multiple locations within the system. In order to evaluate Respawn's performance on an embedded target, we benchmarked its ingest throughput as it varies with data rate per stream. Ingest throughput is affected by the number of streams and their data rates because Respawn employs a fixed size memory buffer which must be partitioned to accommodate a set of streams. More partitioning increases the frequency at which buffered data must be flushed to non-volatile storage, resulting in more computational overhead.

Figure 16 captures throughput performance on the BeagleBone platform by showing the relationship between the data rate of each stream and the maximum number of streams supported. As stream rates increase so does total throughput, however, the maximum number of streams at a desired rate drops because more data per stream is being captured. The performance curve also represents the range of scenarios in which Respawn could be used for historical data storage. For small scale deployments, Respawn can support 58 streams at 100Hz or 11 streams at 1000Hz, as well as a single streams at 10000Hz. Large deployments are supported at slower speeds; 560 streams can be captured at 1Hz or 1950 streams at 0.1Hz. By leveraging local storage, the combination of Respawn and XMPP can support on average one order of magnitude more time-series data than XMPP without local storage.

4.3 Estimated Building Performance

We now frame these performance benchmarks in terms of a real building system. Table 1 shows a list of each gateway, the number of devices attached and the number of transducers on each device that we will be deploying in Scaife Hall during year 1. For example, there are 57 WiStat thermostats connected to a single 802.15.4 gateway. Each WiStat reports six different transducer values. The message rate shows how often the device sends data. In the case of event triggered devices the rate is listed as *push* since they are not periodic feeds. In total, to meet our require system load we need to be able to handle 3691 transducer feeds (443 event nodes) with a total overall message rate of 7.64 messages per second. Each message is approximately 2KB in size. These data rates are achievable with both the BeagleBone as well as the Intel NUC platform. In practice, the system should operate well below the message per second limit to accommodate bursts of actuate or event traffic. In Scaife Hall we estimate needing approximately 10 publishing gateway devices with 443 event nodes. Assuming an equal number of subscribers, the system is capable of handling more than 90 messages/second on a single BeagleBone.

5 Related Work

Multiple academic, government and industry research projects have looked at the problem of designing building automation frameworks. Commercially deployed BAS are largely purpose-built, isolated and proprietary vertical systems from vendors like Johnson Controls, American Automatrix, Honeywell, Bosch, Lutron and Siemens, to name a few. These systems are optimized for local building tasks such as heating and cooling, lighting control or security. Even though many of the newer systems are Internet-connected and provide

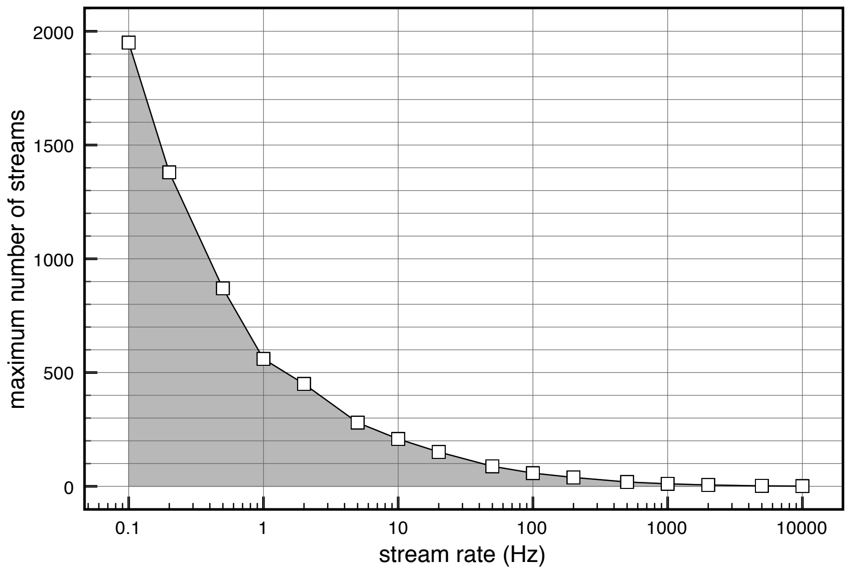


Figure 16: Relationship Between Rate of Respawn Streams and Maximum Number of Streams on BeagleBone

Device	**Qty**	**# of Trans.**	**Rate (msg/min)**	**Total Feeds**	**Msgs / Sec**
FireFly Env. Sensor	115	10	1	1150	1.92
WiStat Thermostat	57	6	0.2	342	0.19
FireFly Plug Load	114	10	push	1140	n/a
Automatrix Thermostat	30	5	1	150	0.5
Chilled Water + Steam	2	2	1	4	0.03
Enfuse Panel Meter	10	3	1	600	5
Lutron Quantum Pts	115	5	push	575	n/a
Total	443			**3961**	**7.64**

Table 1: Devices and Message Rates for Scaife Hall

storage and monitoring capabilities, few enable integration with other systems.

In recent years, the research community has begun developing various systems to help improve programmability and interoperability across systems. In [13], the authors present BOSS: a set of operating system services and API to facilitate writing applications for buildings, with a particular focus on large facilities. BOSS is built on top of the sMAP [14] data store with readingDB at its core and is structured around interactions using RESTful services. While quite an improvement over standard relational databases (MySQL, etc), readingDB does not natively support data migration / replication for caching and fault tolerance. Along with BOSS, there are a handful of other systems like Building Depot [15] that use RESTful architectures to manage transactions. In contrast, Mortar.io uses a push-based publish-subscribe model that is optimized for data streams as opposed to small infrequent transactions. While BOSS and Building Depot's system architecture can be applied to buildings of different size, they are primarily focused on large buildings with complex control requirements. Mortar.io provides many similar services and abstractions, but with a focus on scaling down to small buildings powered by a handful of embedded controllers. Mortar.io also provides plug-and-play functionality and user interfaces designed to support non-expert users as opposed to researchers.

A variety of transactional publish-subscribe architectures have been applied to buildings as well as grid-level management problems. Volttron [16] is an agent execution platform that was initially targeted towards smart grid agent management. A lighter weight implementation called Volttron Lite [17] has also been used for inter-building communication of Roof Top Units (RTUs) management. Volttron also employs sMAP for data collection and then uses ZeroMQ[18] as its underlying messaging protocol. SensorAct [19], The OGEMA [20] and Sensor Andrew [21] have similar goals to Mortar.io. The Mortar.io platform grew out of the Sensor Andrew project and is now being used to replace its underlying backend at CMU. Mortar.io is more specifically curtailed towards building energy systems with a more rigid schema in order to simplify machine-to-machine communication. The interfaces is less focused on how to describe new sensors and instead biased towards selecting an expert-defined sensor and then entering information about its use and location. We believe in the future, experts (not average users) will be extending interfaces and defining device semantics.

SensorAct is very similar to Sensor Andrew with the addition of a high performance time series database and improved scheduling capabilities. Mortar.io expands upon these solutions by adding distributed multi-resolution time series storage as well as distributed event scheduling. SensorAct supports the notion of tasklets that are essentially LUA scripts. Though advanced users can program Mortar.io systems using our LibMIO, the focus of Mortar.io is to allow common users to achieve customizations through scheduling and event triggers rather than fully expressive scripting. All of these functions are accessible through the user interface. OGEMA uses Java and OSGi to provide application developers the ability to program BAS specific devices. It is optimized to run on embedded targets and provides similar access control capabilities as Mortar.io. However, it does not support distributed data storage and its PnP capabilities are much more device- and programmer-centric rather then defining how a user should register or integrate a device.

Efforts in the Internet-of-Things space include highly-scalable device messaging protocols and interaction systems like MQTT[22], XMPP[3], CoAP[23], Alljoyn[24], HomeKit[25], HomeOS[26], The Thing System[27], Xively[28], Sensorpedia[29], etc. Most of these systems are not optimized specifically for building tasks (e.g. set point scheduling, trending, diurnal mode changes, etc.) and typically depend on the cloud.

6 Project Products

The main deliverable from this project is the open-source reference implementation of our architecture which can be found at `http://dev.mortr.io` along with associated documentation at `http://dev.mortr.io/projects/mortar-io/wiki/Documentation`. Source code can be found at `http://git.mortr.io/mio/mio`.

We also showed a live demonstration of the Mortar.io system at the BuildSys conference [30] and released a technical report about the internal operations [31].

References

[1] Underhill R.M. Goddard J.K. Taasevigen D. Piette M.A. Granderson J. Brown R. Lanzisera S. Kuruganti T. Katipamula, S. Small- and medium-sized commercial building monitoring and controls needs: A scoping study. *Pacific Northwest National Laborator Technical Report PNNL-22169*, 2012.

[2] M. Buevich, A Wright, R. Sargent, and A Rowe. Respawn: A distributed multi-resolution time-series datastore. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 288–297, Dec 2013.

[3] http://www.xmpp.org/ (viewed 4/12/2008).

[4] ejabberd:. http://www.ejabberd.im/. viewed 7/15/2014.

[5] Joe Hildebrand and Peter Saint-Andre. XEP-0080: User Location. http://xmpp.org/extensions/xep-0080.html, 7 May 2014.

[6] Carl Ellis, James Scott, Ionut Constandache, and Mike Hazas. Creating a room connectivity graph of a building from per-room sensor units. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '12, page 177183, New York, NY, USA, 2012. ACM.

[7] Dezhi Hong, Jorge Ortiz, Kamin Whitehouse, and David Culler. Towards automatic spatial verification of sensor placement in buildings. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 13:1–13:8, New York, NY, USA, 2013. ACM.

[8] Jiakang Lu and Kamin Whitehouse. Smart blueprints: Automatically generated maps of homes and the devices within them. In Judy Kay, Paul Lukowicz, Hideyuki Tokuda, Patrick Olivier, and Antonio Krger, editors, *Pervasive Computing*, number 7319 in Lecture Notes in Computer Science, pages 125–142. Springer Berlin Heidelberg, January 2012.

[9] Xuesong Liu, Burcu Akinci, James H. Garrett Jr, and Mario Berges. Requirements and development of a computerized approach for analyzing functional relationships among HVAC components using building information models. In *CIB W078 - W102*, France, 2011.

[10] Anthony Rowe, Mario Berges, Gaurav Bhatia, Ethan Goldman, Raj Rajkumar, James H. Garrett, Jos M. F. Moura, and Lucio Soibelman. Sensor andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development*, 55(1.2):6:1 – 6:14, January 2011.

[11] Gal Elidan. Copulas in machine learning. In Piotr Jaworski, Fabrizio Durante, and Wolfgang Karl Hrdle, editors, *Copulae in Mathematical and Quantitative Finance*, Lecture Notes in Statistics, pages 39–60. Springer Berlin Heidelberg, 2013.

[12] Barnabas Poczos, Sergey Krishner, Pal David, Csaba Szepesvari, and Jeff Schneider. Robust nonparametric copula based dependence estimators. NIPS 2011 Copula workshop, 2011.

[13] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. Boss: Building operating system services. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 443–457, Lombard, IL, 2013. USENIX.

[14] Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler. smap: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210, New York, NY, USA, 2010. ACM.

[15] Yuvraj Agarwal, Rajesh Gupta, Daisuke Komaki, and Thomas Weng. Buildingdepot: An extensible and distributed architecture for building data storage, access and sharing. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '12, pages 64–71, New York, NY, USA, 2012.

[16] Jereme Haack, Bora Akyol, Brandon Carpenter, Cody Tews, and Lance Foglesong. Volttron: An agent platform for the smart grid. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1367–1368, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[17] Jereme N. Haack, Srinivas Katipamula, Bora A. Akyol, and Robert G. Lutes. *VOLTTRON Lite: Integration Platform for the Transactional Network.* Oct 2013.

[18] http://zeromq.org (viewed 7/15/2014).

[19] Choi H. Singh A. Singh P. Srivastava M. Arjunan P., Batra N. Sensoract: A privacy and security aware federated middleware for building management. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '12, pages 80–87, New York, NY, USA, 2012. ACM.

[20] Nestle D. *OGEMA Technology Breif.* Fraunhofer IWES Technical Report, 2012.

[21] Rowe A., Berges M., Bhatia G., Goldman E., Rajkumar R., Soibelman L., Garrett J., Moura J. Sensor Andrew: Large-Scale Campus-Wide Sensing and Actuation. *IBM Journal of Research and Development: Special Issue on Smarter Cities and Sensed Infrastructures*, 2010.

[22] Locke D. *MQ Telemetry Transport (MQTT) v3.1 Specification.* IBM Technical Report, August 2010.

[23] Bormann C. Shelby Z., Hartke K. *The Constrained Application Protocol (CoAP).* Internet Engineering Task Force (IETF), 2014.

[24] http://www.alljoyne.org/about (viewed 7/15/2014).

[25] http://developer.apple.com/homekit/ (viewed 7/15/2014).

[26] Colin Dixon, Ratul Mahajan, Sharad Agarwal, A.J. Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 337–352, San Jose, CA, 2012. USENIX.

[27] http://thethingsystem.com (viewed 7/15/2014).

[28] https://xively.com (viewed 7/15/2014).

[29] B. L. Gorman, D.R. Resseguie, and C. Tomkins-Tinch. Sensorpedia: Information sharing across incompatible sensor systems. In *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, pages 448–454, May 2009.

[30] Christopher Palmer, Patrick Lazik, Maxim Buevich, Jingkun Gao, Mario Berges, Anthony Rowe, Ricardo Lopes Pereira, and Christopher Martin. Mortar.io: A concrete building automation system: Demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, BuildSys '14, pages 204–205, New York, NY, USA, 2014. ACM.

[31] Christopher Palmer, Patrick Lazik, Maxim Buevich, Jingkun Gao, Mario Berges, Anthony Rowe, Ricardo Lopes Pereira, and Christopher Martin. Mortar.io: A concrete building automation system. In *Carnegie Mellon University Technical Report*, 2015.