OPQ Version 2: An Architecture for Distributed, Real-Time, High Performance Power Data Acquisition, Analysis, and Visualization

Anthony J. Christe, Sergey I. Negrashov, Philip M. Johnson, Dylan Nakahodo, David Badke, David Aghalarpour
Collaborative Software Development Lab
University of Hawaii at Manoa
Oahu, Hawaii
(achriste, sin8, johnson, dylankhn, davidrb, aghalarp) @ hawaii.edu

***Abstract*—OpenPowerQuality (OPQ) is a framework that supports end-to-end capture, analysis, and visualizations of distributed real-time power quality (PQ) data. Version 2 of OPQ builds on version 1 by providing higher sampling rates, optional battery backup, end-to-end security, GPS synchronization, pluggable analysis, and a real-time visualization framework. The OPQ project makes real-time distributed power measurements which allows users to see both local PQ events and grid-wide PQ events. The OPQ project is split into three major parts: back-end hardware for making power measurements, middleware for data acquisition and analysis, and a front-end providing visualizations. OPQBox2 is a hardware platform that takes PQ measurements, provides onboard analysis, and securely transfers data to our middleware. The OPQ middleware performs triggering on the OPQBox2 sensor network and performs high-level PQ analysis. The results of our PQ analysis and real-time state of the sensor network are displayed using OPQView. We've collected distributed PQ data from locations across Oahu, Hawaii and have demonstrated our ability to detect both local and grid-wide power quality events.**

I. Introduction

As power grids transition from a centralized distribution model to a distributed model, maintaining a stable grid requires fine grained knowledge of how distributed renewables are affecting the state of the grid. Monitoring PQ on a distributed generation smartgrid requires a consumer level distributed sensor network. How do we collect, analyze, and visualize power quality (PQ) on the power grid scale using data gathered from residential utility customers?

To answer this question, we developed and deployed an open source hardware and software system focused on consumer level monitoring across Oahu.

Our system is able to aggregate distributed PQ measurements, request raw PQ data from specific devices, perform high level real-time PQ analysis and classification, and display PQ at both the consumer level and the grid level. At its core is a multi-layered framework consisting of a hardware backend for measuring distributed PQ data, a middleware for triggering and higher level PQ analysis, and a frontend for consumer friendly local and grid level PQ visualizations.

Recently, we've collected [TODO: HOW MUCH] data from [TODO: HOW MANY] devices from devices distributed across Oahu. We found [TODO: WHAT].

Finally, we could produce these boxes for under $100 each. This is roughly an order of magnitude cheaper than other power quality monitoring equipment and is better suited for our applications.

II. Related Work

[TODO:]

III. OPQ Version 2 Architecture

A. OPQBox2

OPQBox2 is a modern, high performance, sensor network centered power quality monitor. OPQBox2 provides high resolution sampled data of up to 50kS/s at 16bits. Optional GPS synchronization and battery backup can be added to the OPQBox2 to tailor it to a specific power quality measurement.

While the sampling and GPS synchronization is controlled by the realtime DSP, all of the signal processing and communication is controlled by a Raspberry PI single board computer. The Raspberry Pi collects 10 AC cycles of ADC measurements at a time and computes the fundamental frequency and V_{rms}. These values are sent to the cloud triggering service, OPQ-Makai, via WiFi while the raw waveforms are buffered on the OPQBox2 for 1 hour. The waveform data is split into chunks of ten cycles each. Each chunk is stored is a Redis sorted-set, indexed by timestamp. Range queries on raw waveform data occur in $\mathcal{O}(n \log n + m)$ time. Older entries in the buffer are expired via a garbage collection process after a configurable amount of time.

The acquisition cloud service may request raw waveforms from the collection of OPQBox2s if the triggering data stream indicated a power quality event. When the triggering server requests data from an OpqBox2, it requests a range of data. This data is queried in Redis, reserialized into a single large data packet, and transfered back to the requesting server over the same ZeroMQ channel that it arrived on. All of the communication between the OPQBox2 and the cloud services are accomplished via an encrypted ZeroMQ connection.

In this paper we verify the synchronization provided by Network Time Protocol (NTP) by comparing power data collected from two different boxes. We can verify the synchronization by cross-correlating both signals with each other. The resulting signal from the cross correlation is at a maximum equal to the delay of the signals.

We also confirm the frequency resolution of the OPQBox by supplying it a 60 Hz signal from a function generator and observing the data recorded by the box. We plot the frequency data and found that our box can accurately record the frequency supplied by the generator.

B. OPQHub

OPQHub is OPQ's middleware system and is responsible for requesting fine-grained data from OPQBox2's and for performing high level PQ analysis. OPQHub is split into two components, OPQMakai, a pluggable triggering and data request component and OPQMauka, a pluggable real-time PQ analysis component.

All communication between the OPQBox2 and the analysis stack passes through two connection brokers. These brokers allow us to terminate encryption at the cloud boundary, thus simplifying the analysis framework.

ZeroMq is used throughout our cloud infrastructure to connect the analysis services to brokers.

Triggering stream consists of the feature reduced data(frequency and V_{rms} measurements) for each device in the OPQ2 network. This stream is brokered via the Triggering broker, and analyzed by the set of analysis processes called Makai. If multiple devices show temporally coherent deviation from the norm, Makai will request the raw data from the OPQBox2 devices via the Acquisition broker.

Further analysis of the raw waveform is performed by OPQMauka analysis and classification system. Each of the plugins in OPQMauka are responsible for different PQ classification and analysis tasks. We currently implement plugins that detect and report PQ measurements, voltage sags and swells, frequency sags and swells, and perform ITIC event classification of voltage events. We've also developed plugins that, along with the triggering framework OPQMakai, allow us to detect these events on a distributed level.

The publish subscribe nature of the plugin system allows plugins to only subscribe to channels they're interested in and produce results that other plugins may be interested in. This allows us to form a pipeline of processing on the cloud side. For example, the voltage analysis plugin will monitor voltage thresholds and report voltage dips and swells. When the voltage plugin encounters an event, it will produce a voltage event message. The ITIC plugin will consume voltage event messages and classify them by their ITIC classification.

Data products from Mauka are stored in MongoDB and presented to clinets use OPQView.

C. OPQView

OPQView serves as the front-end user interface of the OPQ system. It is implemented via MeteorJS, an open source JavaScript web framework that offers real-time and reactive bi-directional communication between client and server. Unlike other existing full-stack web frameworks, Meteor was developed specifically to offer a reactive front-end experience, making it a fitting choice for the dynamic and data-driven nature of our application. Meteor, although an emerging technology, is built upon Node.js - a widely used JavaScript runtime environment. Meteors integration with Node.js grants access to NPM (node package manager) and its large assortment of proven and tested JavaScript modules which we utilize to further enhance the OPQView user experience.

Broadly speaking, the OPQ system consists of three interdependent components - OPQBoxes, OPQHub, and OPQView. OPQView essentially serves as the end-point of this system, where power data eventually propagates to for user consumption. As such, OPQView is responsible for displaying collected power quality data in a useful and meaningful manner. OpqView does not receive its data directly from OPQBoxes, but rather through the intermediary OPQHub, which is responsible for collecting and analyzing data from OPQBoxes - classifying power quality events as they occur. OPQView maintains a communications interface with OPQHub through the use of a MongoDB database, from which it is able to reactively retrieve real-time power quality data. This data includes voltages and frequencies of individual devices, as well as classified individual power quality events and their corresponding waveform data. OPQView not only serves as a front-end to display real-time data, but also a platform to safely and securely store a complete history of power quality data for each individual device - allowing for users to further analyze historical PQ event data well after they have occurred on the grid. In addition to local event data individual to each device, users can also view distributed events - collections of detected local events that are likely caused by the same grid-wide power event. Over time, OPQHub can determine patterns in these distributed events, allowing OPQView to display communities of devices which have a propensity to experience related power events - leading to further insights on how power events tend to propagate through the local grid. The real-time nature of our data allows for a wide potential of visualization techniques - including real-time heat maps of device voltages and frequencies across the grid. OPQView also supplies the administrative interface for OPQBoxes, allowing device owners to adjust their devices privacy and sharing settings as needed. Device owners can choose to publicly share their devices data, or instead only allow a specified set of users access.

Ultimately, we feel that OPQView will serve as a unique platform for power quality analysis and visualization. Meteors strong support for reactive data models remains its main draw - and is the primary reason we have chosen it for OPQViews implementation. The dynamic and data driven nature of our smart-grid application, and the power grid system as a whole, makes it an ideal and exciting choice.

IV. Results

[TODO:]

V. Conclusion

We designed a low cost, real-time, open-source framework capable of aggregating distributed PQ measurements, requesting raw PQ data from specific devices, performing high level real-time PQ analysis and classification, and displaying PQ at both the consumer level and the grid level.

Our system detected multiple local and grid-level PQ events and can also display the current real-time state of the power grid. [TODO: Finish conclusion]