7-16-2021

An Open Source Cyberinfrastructure for Collecting, Processing, Storing and Accessing High Temporal Resolution Residential Water Use Data

Camilo J. Bastidas Pacheco
Utah State University

Joseph C. Brewer
Utah State University

Jeffery S. Horsburgh
Utah State University

Juan Caraballo
Utah State University

Recommended Citation

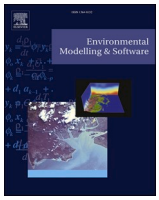
Bastidas Pacheco, Camilo, Brewer, Joseph, Horsburgh, Jeffery, and Caraballo, Juan. "An Open Source Cyberinfrastructure for Collecting, Processing, Storing and Accessing High Temporal Resolution Residential Water Use Data." Environmental Modelling & Software, vol. 144, no. 2021, 2021, pp. 1-16.

Environmental Modelling and Software 144 (2021) 105137

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft



An open source cyberinfrastructure for collecting, processing, storing and accessing high temporal resolution residential water use data

Camilo J. Bastidas Pacheco *, Joseph C. Brewer, Jeffery S. Horsburgh, Juan Caraballo

Department of Civil and Environmental Engineering and Utah Water Research Laboratory, Utah State University, Logan, UT, USA

ARTICLE INFO

Keywords:
Residential water use
Data management
Smart metering
Cyberinfrastructure
Information and communication technology

ABSTRACT

Collecting and managing high temporal resolution residential water use data is challenging due to cost and technical requirements associated with the volume and velocity of data collected. We developed an open-source, modular, generalized architecture called Cyberinfrastructure for Intelligent Water Supply (CIWS) to automate the process from data collection to analysis and presentation of high temporal residential water use data. A prototype implementation was built using existing open-source technologies, including smart meters, databases, and services. Two case studies were selected to test functionalities of CIWS, including push and pull data models within single family and multi-unit residential contexts, respectively. CIWS was tested for scalability and performance within our design constraints and proved to be effective within both case studies. All CIWS elements and the case study data described are freely available for re-use.

1. Introduction

Achieving higher efficiency in urban water management and planning requires understanding of how water is used at the household level. Daily patterns in consumption, potential for water savings and distribution of water use across end uses are essential inputs to water demand estimation, leak identification, design of programs to manage water demand, and water planning to ensure adequate supply (Giurco et al., 2008; Willis et al., 2011). Metering water use for billing purposes is a common practice in the United States, where meters are typically read monthly or quarterly. Our ability to characterize water demand is limited by the temporal resolution of the data collected. Higher resolution data can increase the accuracy of peak demand estimation and reduce leak volumes that can go undetected. Sub-minute resolution data is required to record and quantify end uses of water that have short duration (Cominola et al., 2018; Nguyen et al., 2015). However, obtaining this higher temporal resolution data at a scale larger than a few houses presents several challenges in terms of data collection, storage, management, and processing (Cominola et al., 2018), and doing it over an extended period of time can be unpractical (Cardell-Oliver, 2013).

Collecting a month of 10-s resolution data for a single meter, which is common in end uses of water studies (DeOreo et al, 2011, 2016; Mayer et al, 1999, 2004), produces more than 250,000 observations. Doing so at a water utility or municipality scale, which may have thousands of metered residential connections, presents obvious challenges associated with the volume of data that would be produced. Many utilities lack a dedicated information technology or data management staff, which means that new database management, software deployment, and data analysis tasks can be prohibitive. In these cases, and in the absence of sufficient cyberinfrastructure for automating data management tasks, high resolution data could be more of a roadblock for a water provider than a benefit. However, with adequate data collection and management tools, utilities may be able to realize more of the potential benefits associated with high temporal resolution data. This includes quantifying water use behavior to better enable planning that ensures adequate supply, the promotion of water conservation behavior among users (Liu et al., 2015), improving customer service quality for utilities (Beal and Flynn, 2015), tipping the cost-benefit balance in the smart metering adoption case, which remains undefined (Cominola et al., 2018), and enabling the proliferation of scientific work in this field.

The term "cyberinfrastructure" integrates hardware and software tools, as well as data networks (NSF, 2007). Cyberinfrastructure can help solve data management challenges and enable more widespread collection of higher temporal resolution water use data for utilities and researchers. In a broader context, cyberinfrastructure is improving the

* Corresponding author. 8200 Old Main Hill, Logan, UT, 84322-8200, USA.
E-mail addresses: camilo.bastidas@usu.edu (C.J. Bastidas Pacheco), jbrewer256@gmail.com (J.C. Brewer), jeff.horsburgh@usu.edu (J.S. Horsburgh), juan.caraballo17@gmail.com (J. Caraballo).

https://doi.org/10.1016/j.envsoft.2021.105137
Accepted 13 July 2021
Available online 16 July 2021

communication of results from hydrological models (Souffront Alcantara et al., 2017), helping monitor watershed health parameters (Szwilski et al., 2018), assisting in the automation of comparing climate model results (Sun et al., 2020), and it is now ubiquitous in multiple scientific domains (Hachmann et al., 2018; Shams et al., 2018; Wegrzyn et al., 2020).

Smart meters have potential to solve one of the challenges in the pathway to an advanced water cyberinfrastructure, high resolution measurement of water use. The term "smart meter" can be ambiguous (Boyle et al., 2013). Within this article, it is used to denote devices capable of recording water use with high resolution (i.e., sub-minute frequency) that can be integrated in automated systems for data management. Nearly a decade ago, it was anticipated that use of smart meters would grow over time (Boyle et al., 2013), and they are, in fact, becoming more widely available and adopted. With this emergence of smart meters, there has been an increase in the number of scientific publications using the high resolution data they produce for water demand analysis. Cominola et al. (2015) provide a comprehensive review. However, despite the increase in the number of publications using smart metering data to quantify end uses of water and water use behavior, the data management procedures, or tools, used in these studies are not well described, and most of the datasets used are not openly available (Di Mauro et al., 2020). In most of these studies, the focus has been on the tools and algorithms used for identifying water end uses and user behavior. Other components of the data management process are not described.

Available cyberinfrastructure for collecting, managing and analyzing this type of data remains scarce and of proprietary nature, with little available literature describing tools and procedures for data collection, management, and analysis. Meter manufacturers tend to have their own software systems designed for their metering technology, which complicates synthesis or integration of data from multiple systems and may help explain why research in this field has been conducted in a limited number of countries using a limited number of datasets. Many of these studies have used the same data logging device for data collection and the same software tool for end use analysis (Beal and Stewart, 2011; DeOreo et al, 2011, 2016; Mayer et al, 1999, 2004). Other studies have reused the same dataset to conduct different analyses. For example, Beal at al. (2013) present differences between perceived and actual water consumption, Willis et al. (2013) studied the impact of socio-demographic and efficient fixtures on water use, and Beal and Stewart (2011) presented end uses of water characteristics, all using the same dataset collected in Southeast Queensland, Australia.

The datalogging devices used in most high-resolution data collection studies lack communication capabilities, which limits the potential for automated integration with downstream cyberinfrastructure (e.g., telemetry, storage, management, and analysis applications). More recently, there has been increasing discussion around *smart cities*, *smart grids*, *smart water networks* and other related terms, despite there not being a wide agreement about their definition, what is meant by "*smart*," or the extension of their applications (Ardito et al., 2013; Hollands, 2008; Wissner, 2011). It is generally agreed that smart cities make use of information and communication technologies (ICT) in an attempt to assist cities in optimizing the use of their assets (Neirotti et al., 2014), water being one of the most important. Connectedness of data collection and its application is important in this context.

Advanced metering infrastructure (AMI) and ICT systems are vital for the successful deployment of a smart grid (Yan et al., 2013). In the energy sector, smart grids use smart technologies for metering, communication and automation and make use of digital information to improve reliability (U.S. Congress, 2007). The Internet of Things (IoT) has also been described as a potential enabler of smart grids in the water sector (Alghamdi and Shetty, 2016; Robles et al., 2014; Zanella et al., 2014), and, more recently, smart solutions that use IoT principles have been proposed (Amaxilatis et al., 2020; Stiri et al., 2019). Liu and Nielsen (2016) discussed existing technologies to develop an ICT system, or cyberinfrastructure, to enable smart meter analytics for the energy sector acknowledging the difficulties in processing and managing the large volumes of data generated. Similar systems have been proposed and discussed for water use analytics (Boyle et al., 2013; Li et al., 2020; Makropoulos, 2017; Moy De Vitry et al., 2019), but few implementations have been published due to the cost and complexity of these applications (Alvisi et al., 2019; Amaxilatis et al., 2020; Anda et al., 2013). In one notable example, Chen et al. (2011) conducted analysis using data collected on a smart water service architecture deployed for billing purposes on the city of Dubuque, IA. This system collects data every 15 min providing more advanced analysis to water consumers and providers (Erickson et al., 2012).

While multiple high-level designs of a smart water network have been described (e.g., Hauser et al., 2016; Li et al., 2020; Ye et al., 2016), implementations are scarce. Most of the smart water systems designs we reviewed lacked a full demonstration or prototype implementation. In some cases, important elements, such as performance metrics and implementation guidance were not fully described (Li et al., 2020). When demonstrations were presented, the focus was primarily on the results of the specific case study (i.e., the lessons learned about water use and/or behavior) and not on the design and implementation of the tools used to complete the tasks. The limited availability of data and tools for the water sector constitutes a significant barrier for the development of research and prevents the advancement and implementation of smarter water grids at a large scale (Mutchek and Williams, 2014). The closed-source nature of existing data collection hardware and data management software creates accessibility and interoperability issues that prevent the progress of smart water grids while curtailing the adoption of open architectures (Hauser and Roedler, 2015; Robles et al., 2014). The development of open source cyberinfrastructure for managing high resolution data can lay the foundations for the development of newer and better tools for water utilities, as well as standards for operations that result in increased interoperability. All of these actions could pave the road for more water demand research, and ultimately, advance technologies for the development of smart water grids.

Thus, in order to achieve the full potential of smart meters, cyberinfrastructure is needed to support utilization of the high resolution data they produce (Horsburgh et al., 2019; Mason et al., 2014). Developing effective cyberinfrastructure that can support both operational data collection and management (e.g., for billing, reporting and day-to-day management purposes) and exploration of data for research aimed at better understanding water use behavior is expensive and challenging (Stocks et al., 2019). Indeed, architectural designs and data structures for cyberinfrastructure supporting residential water use data must meet the needs of multiple users (i.e., water providers, water consumers, researchers) without disrupting a utility's necessary business functions. The research described here focused on the following research questions to advance the cyberinfrastructure and availability of software tools for collecting, managing and analyzing high resolution smart metering data: a) what is the general architecture for a cyberinfrastructure to support collection and management of high temporal resolution smart metering data, and b) how can that architecture be implemented to meet the needs of multiple potential users (e.g., water utilities, water consumers, researchers).

In this paper, we present a generalized architectural design for a Cyberinfrastructure for Intelligent Water Supply (CIWS) and a prototype implementation of each of the components within the architecture in support of multiple data collection, management and analysis case studies. The prototypes we developed demonstrate tools that are not currently available for researchers or utility managers and include: a) a data collection layer consisting of datalogging devices with data transmission capabilities, which are modifications from our previous work (Horsburgh et al., 2017; Bastidas Pacheco et al., 2020); b) a data management and archival layer that receives, processes, and stores data; and c) a data analytics layer that enables calculation of common water use metrics (e.g., average hourly water use, instantaneous peak, and end

uses of water disaggregation and classification). Components within these layers demonstrate the entire workflow consisting of data collection, communication, storage, management and archival, and visualization and analysis.

While CIWS was designed and implemented for research purposes, including appropriate mechanisms for protecting the identities of research participants where necessary, it facilitates implementation of high temporal residential water use analysis, which is of interest to not only researchers in the field, but also utility companies and water consumers and can provide information currently not available to them. The data collected and managed using CIWS is relevant for assessment and management of both water demand and for planning to ensure adequate water supply. We first describe the requirements for the system along with the overall architecture we designed to meet these requirements (Section 2). We then describe a set of case studies in which this overall architecture was prototyped and implemented using both existing and new open source hardware and software components (Section 3). Finally, we close with discussion and conclusions (Section 4).

2. Methods

2.1. CIWS design and overall software architecture

Our goal in developing CIWS was to create a generalized, modular architecture that can be used to automate the process from collection to analysis and visualization of high temporal resolution water use data. In our case study applications of CIWS, we combined existing and developed new, open source hardware devices and software tools to demonstrate an integrated solution for high-resolution residential water use data collection, management, and analysis. The CIWS architecture and our prototype implementation were designed to address the following requirements. While we present our prototype implementations in this paper, there may be multiple implementations of the generalized architecture that meet these requirements.

a) An open architecture that could be implemented using a variety of technologies;
b) Open source software development to facilitate its deployment and use by other users, reduce costs, and provide a platform for future improvement by others while advancing financial feasibility of larger scale implementations;
c) A modular design, so each component of CIWS can be used, or advanced, independently;
d) Accept input data from different meters and measurement devices (sensors) to address heterogeneity in urban water meter technology;
e) Capacity to manage “push” and “pull” data retrieval from the metering devices depending on available communication technologies and storing of data in a centralized server;
f) Scalable to accommodate a large data volume while remaining responsive to queries for subsets of time series data of varying sizes;
g) Support production of analysis and insights that meet the needs of different audiences.

In our review of the literature, we found that existing designs of smart components or cyberinfrastructure for managing water systems are not fully standardized. However, most systems described or implemented to date are composed of multiple layers working in connection to achieve the overall goal (Li et al., 2020). We found that the number, name and function of these layers was different in each design; however, we observed some similarities. In practice, the number of layers included in an architectural design comes down to tradeoffs between the benefits of modularity and separation of concerns that can be achieved versus the complexity and potential fragility introduced with a larger number of layers. Separate layers can be autonomous such that changes to one layer do not have to affect the other layers. However, a greater number of layers typically involves more components that can fail.

Our overall architectural design for CIWS adopts this multi-layer paradigm (Fig. 1) and is composed of three main layers. The first layer is the Data Collection Layer and includes the physical instruments and sensors used to monitor water use. It has also been called the sensing layer (Ye et al., 2016), the physical layer (Hauser et al., 2016), or the instrument layer (Li et al., 2020). The second layer is the Data Management and Archival Layer, which handles data communication, parsing and archival. This layer has also been referred to as the network or function layer (Hauser et al., 2016; Li et al., 2020; Ye et al., 2016). The final layer is the Data Analytics Layer, which handles all the steps between queries to retrieve data from the archival component to final visualizations, analyses and presentations produced for utilities, water consumers, researchers, etc. (i.e., the consumers of the data). This layer has also been referred to as the application or the data fusion and analysis layer (Hauser et al., 2016; Li et al., 2020; Ye et al., 2016). Some of the other systems reviewed include elements for real time monitoring and control of observed variables and processes within the system, resulting in architectural designs with a larger number of layers. Since these elements were not needed in our case study use cases, a three layer model met all of the requirements listed above. A system with more layers may become more fragile; therefore, our design includes the minimum needed to meet the design considerations.

The architecture for CIWS and our prototype implementations were developed with a research focus – e.g., collecting, storing and managing high resolution water use data to enable advanced study of residential water use behavior. This type of research may be carried out by utilities, universities, or other agencies involved in research related to or management of urban water supply and demand. The typical deployment size in this type of work has been around 50 houses per city; however, some studies have analyzed up to 762 sites (DeOreo et al., 2016). In the latter case, the data was not collected simultaneously at all sites. Our aim was to develop a system that can handle, at minimum, the number of simultaneous data collection sites within the range of deployments observed in the past (40–60 houses). In the following sections, we describe in more detail the high-level design for each of the architectural layers, their key components, and their basic functionality.

2.1.1. Data collection layer

Data collection refers to the actual measurement of the variable or variables of interest, in this case, high temporal resolution water use. Here, we define high temporal resolution data as data collected at a sub-minute resolution. Typical investigations of water use behavior, such as separating and quantifying end uses of water within a home, require data to be recorded at 10-s or even finer resolution over data collection periods of weeks to months. With few exceptions, high temporal resolution data cannot be collected using existing, commercially available smart meters without adding additional hardware or software components (Cominola et al., 2018), which can be expensive (Horsburgh et al., 2017). Water metering technology typically consists of a physical meter that uses one of several measurement techniques paired with an analog or digital register on which a totalized volume of water use is recorded. Some registers, including those of commercially available smart meters, are capable of storing volume readings within internal memory; however, this is usually constrained to relatively short periods of time (e.g., weeks) at recording intervals longer than 1 min. Other registers report only the most recent volume reading and are designed for periodic (e.g., monthly or quarterly) readings either manually or automatically via radio. These practical limitations are driven by power, local data storage, and network bandwidth limitations of existing metering technology.

Some water use studies have added flow metering sensors directly on the water pipe leading to each appliance in a residential house (Kofinas et al., 2018; Di Mauro et al., 2019). Opting for this approach allows direct measurements of water use from each fixture, and by placing the measuring element inside the property, power and communications can be readily available. However, this approach is invasive and requires

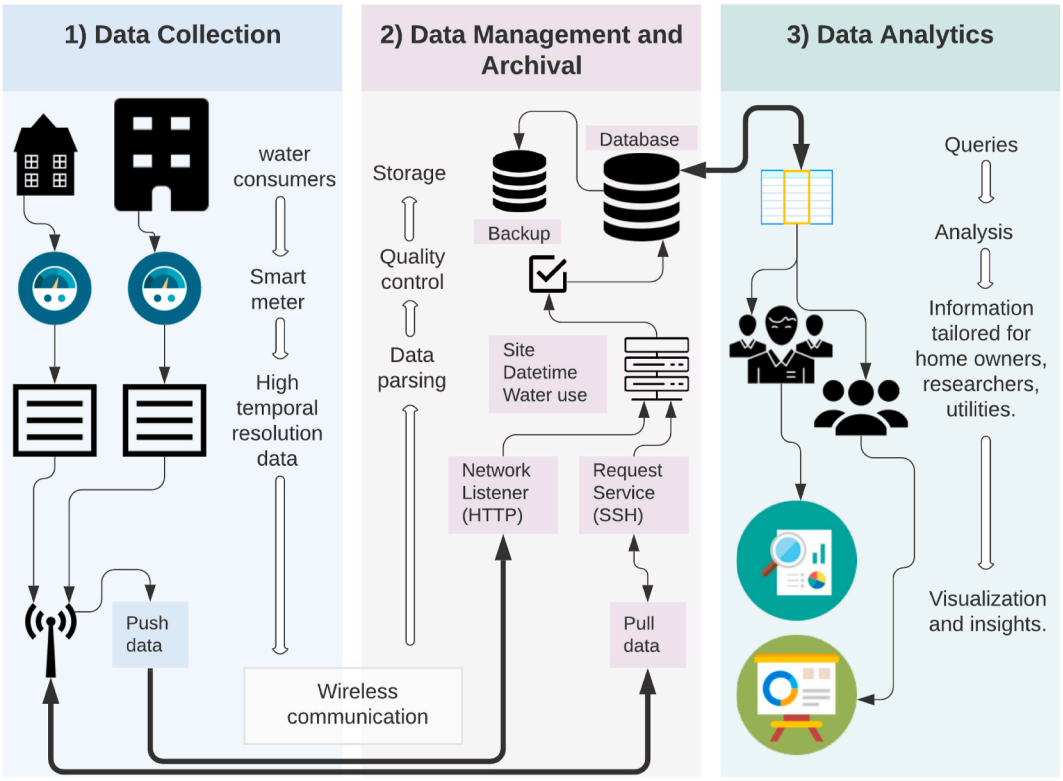


Fig. 1. Overall architecture design of CIWS consisting of three main layers: 1) Data Collection, 2) Data Management and Archival, and 3) Data Analytics. Arrows are used to indicate data and workflow movement between components. White arrows indicate the flow of data and information and black arrows show the connection between elements and layers.

modifications to the plumbing in each home where data is collected, which can increase costs and limit the applicability of this methodology at a medium or large scale. Therefore, we opted to focus our efforts on datalogging devices that can be coupled with the existing water meter available at the property. Datalogging devices designed to couple with existing meters are available (Bastidas Pacheco et al., 2020; F.S. Brainard & Company, 2020). These dataloggers essentially perform the same function as the meter's register, but have the capability of recording much more frequent observations over longer periods of time. To be fully integrated in a data management system like CIWS, the datalogging devices must also have communication capabilities. CIWS was designed to handle both push and pull data communication, making it adaptable for multiple scenarios. The term *push* is used to denote systems where the data is sent by each datalogger (client) to a centralized server, while *pull* refers to systems where a centralized server connects to each datalogger and requests data. Given the modular design of CIWS, it is possible to integrate dataloggers that lack communication capabilities, such as those used in most residential studies in the past. Under this scenario, a user can take advantage of the Data Management and Archival and Data Analytics Layers of CIWS, while using data files manually downloaded from the datalogging devices in the field.

2.1.2. Data management and archival layer

The Data Management and Archival Layer is responsible for the work required to process the data logged by the devices. The key component addressed in this layer relates to developing and using software elements to automate repetitive data management processes and enable an easier transition between large volumes of data collection and useful information generation. This layer is composed of multiple working elements (Fig. 1). For push based data transmission, a listener service is required to receive the data sent by the dataloggers. In pull based data transmission, a request service is used to achieve the same task. Once the data is received, it must be verified, parsed and transferred to a database component. The database component accepts and stores data for downstream analysis and decision making. Real-time monitoring of water use is typically not of interest in most research scenarios, where most data analysis happens after the data have been collected. Additionally, given the frequency with which observations are recorded (e.g., on the order of seconds), it is not practical to push or pull data every time a new observation becomes available. Based on this, CIWS was designed to collect and send files containing many observations rather than sending observations individually. This approach minimizes the communication load on the system because the data transfer process does not occur constantly, and it can be scheduled to meet specific needs.

The request service for pull based data transmission must execute the following tasks: a) connect to a datalogging device; b) check for new data files; c) request and transfer new files; d) read and parse the files, and e) upload the data into the database. Remotely accessing devices can be achieved using a variety of communication protocols like Secure Shell (SSH), which is a widely used method for similar tasks due to its simplicity, speed and security. In this model, the datalogging devices need to be powered on and connected to the network at the time the connection is established. Additionally, a key requirement is that each datalogging device must be located, addressed, and accessed directly, which also provides an opportunity for remote functionalities, such as software updates, troubleshooting, changing data collection settings, and others.

The listener service, which manages the data transferring process under the push model, must complete the following tasks: a) accept and validate the data sent from each datalogging device deployed, b) process incoming files, including parsing the information they contain, and c) saving the data received into the database. Under this approach, the communication elements of the datalogger only need to be powered up and functioning for the time it takes to send the desired information to the listener service, which can contribute to lower power requirements.

Additionally, there is no requirement for data logging devices to be uniquely addressed on a network as they can identify themselves within the content of the message they push to the listener service.

Multiple technologies that can potentially meet the data storage and accessibility design considerations (i.e., the database requirement) are available. The database must be able to manage large volumes of data and provide a platform for generating analytics of such data. The data managed by the system consist mainly of time series of flow observations, which are constantly being collected and written into the database. Thus, the databasing technology selected must provide: a) easy and fast querying between dates and times to enable manipulation of the data; b) high performance for read and write operations as the database is continuously being updated with new data and potentially accessed by multiple users; and c) scalability, as the volume of data to be stored in the database increases quickly as the monitoring network and time period over which data are collected grow. The database schema used to organize the data for CIWS was designed to maximize query efficiency while maintaining the ability to protect the privacy of water consumers by storing personally identifiable information outside of the database. Common queries to be conducted in projects where CIWS can be used include selecting all or part (time constrained) of the full resolution or time aggregated data for a single or multiple sites.

2.1.3. Data analytics layer

The Data Analytics Layer supports generalized interactions between data users and the database for the purposes of visualization and analysis of the data. The necessary functions executed in this layer include: a) user authentication to access existing data, b) querying data from the data base, c) data manipulation and analysis, and d) generation of reports and visualizations of interest for different target audiences. For the purposes of this research, three main target audiences were identified as users of information produced by the Data Analytics Layer: water consumers, utility managers, and researchers. While these categories of users are not necessarily exhaustive or mutually exclusive, the information that would be useful to these different users and the methods used to interact with the data are not the same. For instance, an individual residential user would need to be able to access and interact with the data from their home in a practical and non-technical way that does not require specialized software. Past studies have evaluated residential users preferences for water use feedback, finding that information about their prior water consumption, comparison of use with that of similar users, and details about their consumption can increase user understanding (Erickson et al., 2012; Liu et al., 2015).

Utility managers may want to access standardized plots or reports showing data from multiple users, and researchers may need much more freedom to formulate their own, custom queries to the database to subset, aggregate, or summarize data in useful ways. This implies that the Data Analytics Layer needs to support multiple mechanisms for accessing and interacting with the database. Authentication, authorization, and privacy for users with different privileges (read or write data in a database) to access online resources have been discussed for multiple applications (Christie et al., 2020; Heiland et al., 2015; Kim and Lee, 2017). High temporal resolution data products, such as distribution and timing of end uses, can raise privacy concerns among water consumers that must be considered when designing data presentation tools (Froehlich et al., 2012). Aggregation and summarization techniques can be used to present information for multiple water consumers while protecting privacy, and authentication and authorization can be used to limit what data is available for different users. CIWS considers the use of anonymized datasets throughout the system by identifying water consumers with a unique identifier. Linkage with the personally identifiable information about each water consumer is stored separately and is only available to those who have appropriate privileges and are allowed match water consumers with their data.

2.2. Case study design and system testing

In order to evaluate the overall architecture design, we designed two case studies that demonstrate different aspects of the architecture presented in two distinct data collection environments. The first case study demonstrates data collection at individual single-family residential homes. It uses an autonomous datalogger with communication capabilities to collect high resolution water use data and demonstrates push-based transmission of the data to the Data Management and Archival Layer. The second case study demonstrates data collection within multi-unit residential structures on a University campus. It uses dataloggers with dedicated power supplies and network registrations to demonstrate pull-based transmission of the data to the Data Management and Archival Layer. In the second case study, we collected data for additional parameters needed to characterize the energy consumption related to hot water use. The collection of data for these parameters provides an example of CIWS flexibility. Both case studies share the same layers, but we describe the different elements used by each case study.

We created a full prototype implementation of the design layers presented in Fig. 1 for each case study and deployed them in an operational environment. These prototypes and deployments were created to demonstrate proof-of-concept for data collection and management components, the shareability of components within the architecture regardless of the data transmission method, and generalizability for our architectural design. We tested the system developed for scalability by simulating an increased number of sites and larger volumes of data.

Python 3.7 was chosen to develop all of the code and software associated with our case studies given that it is freely available and open source, it is a high-level programming language with a vast number of libraries available to complete an important number of functions required in our application, and it could be used across all three layers of our architectural design. Using Python also helped us meet the first three requirements described above as the code can be easily shared, read and modified by other programmers and scientists, and can be deployed in different operating systems, which increases reuse possibilities.

2.2.1. Case study 1 description

Water use in single family residential homes is quantified, to a large extent, using analog, positive displacement water meters. The volume of water that has passed through the meter is usually the only variable recorded by this type of meter. In most cases, water meters are enclosed in underground pits of varying depth, limiting power supply availability. These meters are typically read monthly, quarterly or at coarser resolutions by the utility for billing purposes either manually or via a roving radio that receives the most recent volume observation from each meter when the roving radio passes within range. Some more advanced networks include automated retrieval of the coarse resolution volume data, but very few have the capability to record and transmit high resolution data. Given that the vast majority of residential water meters in use today share these constraints, we chose this case study to demonstrate adding high resolution data collection and transmission capabilities to existing, analog water meters.

2.2.2. Case study 2 description

The Living Learning Community center (LLC) on Utah State University's (USU) campus was selected as a second case study for deploying CIWS within a set of multi-unit residential buildings. The LLC is one of USU's newer student housing options and houses approximately 500 students distributed among six dormitory buildings labeled building A – building F. The objective of this implementation was to characterize water and water-related energy use in five buildings (B–F). The importance of the water-energy nexus for optimizing conservation and sustainable management has been identified in the past (Hamiche et al., 2016; Kenway et al., 2016; Fang and Chen, 2017). However, collecting water and energy consumption data combined at a sufficient temporal resolution to analyze their relation is uncommon, and the

methods for linking water and energy use are not well established. This case study demonstrates a methodology for collecting water and water-related energy data in a multi-unit residential setting. Buildings B–F host approximately 90 students each. Building A hosts administrative offices, has a much lower student occupancy, and was excluded from the study. We chose a pull based model for this case study given the availability of dedicated power at each data collection site and the availability of USU's campus Wi-Fi network to enable communications and data transmission.

Three water meters are present in the water supply system for each of these buildings - hot-water supply, cold-water supply, and hot-water return. To monitor water and water related energy use within each building, two characteristics of each meter were measured, flow and water temperature, resulting in a total of six variables collected per building (Table 1). The hot-water return is a feature of the LLC's innovative hot water recirculation system. Hot water is continually circulated from three boilers to the LLC buildings at a constant, base flowrate of approximately 3 gallons per minute (gpm) or 11.4 liters per minute (Lpm). Increases from this base flowrate constitute hot water use. Unused hot water returns to the one of the three boilers for reheating and eventual recirculation. Cold water is supplied in a typical on-demand basis.

3. Results and discussion

3.1. Case study 1: push based data collection for single family residential homes

We selected a single family residential property to test the CIWS functionality under a push based data retrieval model. We collected two weeks of data at this property, between January 15, 2021 and January 28, 2021, for the implementation described. All water use results presented are for this time period. This home had five occupants, three of ages between 10 and 25 and two between 40 and 60 during the data collection period. It was built in 2006, has three bathrooms and a total parcel area of approximately 12,000 ft^2 (1114.8 m^2). We chose push based data retrieval for this case study because it is enabled by heterogeneous networking – i.e., any datalogger device capable of high resolution data collection and sending data over an available data network could be used without the need for each device to be uniquely addressable on a network. Additionally, power requirements can be reduced given that data logging devices do not have to listen for connections and requests from a centralized server but rather wake to transmit data on a user-configured schedule.

3.1.1. Data collection layer

At the property selected, a one inch (2.54 cm) Bottom Load (BL) Master Meter with an analog register was being used by the water utility to record monthly water use, transmit it to a roving receiver via a 3G radio and bill water usage. We added high temporal resolution data collection and transmission capabilities without affecting the normal operation of the utility's meter by installing a CIWS Water Meter Node (CIWS-WM-Node) datalogger to measure water use at a 4-s temporal resolution on top of the existing meter. The CIWS-WN-Node is an advanced modification of the CIWS datalogger (Bastidas Pacheco et al., 2020), which is an open source, Arduino-based datalogger that we designed to work with any magnetically-driven water meter. The CIWS datalogger uses a magnetometer sensor to measure the magnetic field around magnetically-driven residential water meters. It counts peaks in the magnetic field associated with movement of the magnetically-driven measurement element within the meter, and registers peaks as pulses that represent a fixed volume of water passing through the meter. These pulses are multiplied by a factor called the meter resolution (0.041619 gallons per pulse, or 0.1575 liters per pulse, for the case study meter), which is specific to each meter type, brand, and size, to obtain the volume of water that passed through the meter per unit of time. Meter pulse resolution values can be obtained from meter manufacturers or through a calibration procedure described by Bastidas Pacheco et al. (2020).

Table 1
Variables measured, measuring device, and units of observation at each LLC building.

Measured Variable	Measuring device	Units
1) Hot-water supply flow	Master Meter Octave Ultrasonic water meter with 4–20 mA current loop outputs	gpm
2) Cold-water supply flow		
3) Hot-water return flow	Master Meter Bottom Load Multi-Jet (BLMJ) water meter with pulsed output	pulses
4) Cold water supply temperature	DS18B20 digital thermometer with digital output	°C
5) Hot water supply temperature		
6) Hot water return temperature		

The CIWS-WM-Node we developed for this case study adds communication and computational capabilities to the CIWS datalogger by coupling it with a Raspberry Pi Model B or Model B+ single-board Linux computer. The components of the CIWS datalogger control all of the datalogging functions, whereas the Raspberry Pi computer can be powered on a user defined schedule to process and transmit data. The Raspberry Pi runs a version of the Linux operating system called Raspberry Pi OS (previously called Raspbian). Although the Raspberry Pi is capable of interfacing with a number of different wireless communication options, including Wi-Fi, radio frequency, cellular 3G, LTE, Bluetooth, and satellite, we chose to use the Raspberry Pi's built in Wi-Fi capabilities for this case study because the homeowner's Wi-Fi network was easily accessible. In broader application, however, any Internet data connection compatible with a Raspberry Pi could be used.

The CIWS-WM-Node datalogger outputs a comma separated values (CSV) file including a three line header with a unique identifier for the site at which the datalogger is installed, a unique identifier for the datalogger, and the meter resolution for the meter on which it is installed. The datalogger records three variables during the logging process: Datetime, Record, and Pulses (Bastidas Pacheco et al., 2020). The CIWS-WM-Node datalogging device was configured to chunk the data files by day (i.e., a new CSV file is created for each day) and send data files once per day to the Data Management and Archival Layer via an HTTP POST request. This functionality was developed as a single Python script (*data_transfer.py*). When the Raspberry Pi is powered on, it can conduct any computation required, and the *data_transfer.py* script is executed to send data files to the Data Management and Archival Layer for further processing. After a file is successfully sent via HTTP, it is moved to a different folder in the datalogger's local storage for backup.

3.1.2. Data management and archival layer

For our case studies, the Data Management and Archival Layer components were deployed within a VMWare ESXi server environment hosted at Utah State University on a single virtual machine (VM) running the Ubuntu Linux Server Version 18.04 (Bionic Beaver) operating system. Ubuntu is a free and open-source Linux distribution developed by Canonical Ltd. It is well supported, stable, and offers reliable file security. The VM was configured with a 64-bit architecture, four 2.3 GHz processor cores, eight GB of RAM, and 100 GB of hard disk space. We refer to this VM as the "Data Management and Archival server."

We developed three main components to complete the tasks described for this layer, the data posting service (DPS), the data loading service (DLS), and the operational database, each of which is described in the sections that follow. The DPS and the DLS were developed in a generalizable way to facilitate reuse and serve as the Network Listener shown in the center panel of Fig. 1. However, some specific details were

adapted to this implementation. For example, the data parsing works for the specific output format of the CIWS-Datalogger. The DPS and the DLS were deployed on the Data Management and Archival server and then configured via settings stored in a user-modifiable JavaScript Object Notation (JSON) file (named *configuration.json*) that details the information needed for their operation. For deployment, the configuration file must be placed in the same folder with the DPS and DLS.

3.1.2.1. Data posting service (DPS). The DPS is a listener web service that receives and processes data files pushed to the Data Management and Archival server from the CIWS-WM-Node dataloggers. The DPS works integrated with two common server technologies, the web server software that processes HTTP requests received by the server and a Web Server Gateway Interface (WSGI) that runs the DPS application in response to the requests. We chose NGINX (NGINX, 2021), which is a free, open source HTTP server, to serve as the web server software because of its high performance, stability, simple configuration, and low resource consumption. The WSGI was implemented using (Gunicorn, 2021), which is a Python WSGI HTTP server for Unix-like operating systems. Guidance for deploying the web server and WSGI software is available in the project's GitHub repository. The parameters included in the configuration files for the DPS and the DLS are described in Table 2.

The overall functioning of the DPS is as follows. Dataloggers send an HTTP POST request to the server that contains a data file (for our case study, one day of high resolution water use data for that home). These requests are received and handled by the NGINX web server, which passes them to the Gunicorn WSGI. Gunicorn then invokes and executes the DPS to authenticate the HTTP POST requests by using a token (*client_token* in Table 2), verifying the file type (CSV) and that the file does not already exist on the server, before moving it to a local folder on the server (*source_directory* in Table 2) for further processing by the DLS. The DPS is composed of three pieces of code: *app.py* which lists the functions needed to read the application configuration file, *auth.py* that lists all the functions for file authentication, and *web_service.py* which calls the previous two files and executes the tasks described. Fig. 2 illustrates the processes described and lists the elements involved.

The DPS was implemented using Bottle (Hellkamp, 2021), which is a WSGI micro web-framework for Python. Bottle is simple, fast, lightweight, and works without additional dependencies, making it ideal for running small applications like the DPS. Bottle built-in functionalities, such as its simple URL routing capabilities and the convenient access to file uploads, were used to facilitate the development of the DPS and avoid dealing with low-level details of HTTP requests handling and routing. We implemented a very simple, token-based authentication for the HTTP POST requests in our prototype to avoid SPAM content being submitted to the DPS. More sophisticated and secure authentication and authorization processes could be integrated in the future, if needed to provide greater security. A log file keeps track of the requests received by the DPS and actions executed (the log file is located in a directory described in Table 2). The log file records successful and unsuccessful (e.g., a file that already exists is sent to the server multiple times, a request that is rejected by not having appropriate authentication credentials) posting attempts. All events are logged in a single file, named *data_poster.log*, which is limited to 5 MB in size. When a log file exceeds this size, it is saved adding a sequential number at the end (*data_poster1.log* initially) and the current logging continues in the original log file.

Table 2
Parameters included in the configuration file for the data posting (DPS) and data loading (DLS) services. The configuration file follows the structure presented here.

Parameter		Description
log_directory		Directory where the log files are located.
source_directory		Directory where the files accepted by the DPS are placed. The DLS processes the files located in this directory.
target_directory		Directory where the CSV files will be moved to after the data is uploaded into the database for archival.
quarantine_directory		Directory where the CSV files will be moved to if an error occurs.
client_token		A public key used to generate upload tokens and authenticate upload requests.
secret_key		A private key used to generate the upload tokens.
database	*name*	Name of the InfluxDB database used.
	user	Username of the InfluxDB user used when connecting to the database.
	password	InfluxDB Password for the user selected.
	host	The host name of the server on which the InfluxDB database is installed.
	port	The Internet port number over which communications with the InfluxDB database server have been configured.

3.1.2.2. Data loading service (DLS). We developed the DLS to read the files received from the dataloggers from the source directory on the server, parse the unique site identifier information from the header of the CSV file and insert the data into the database for archival and use by the Data Analytics Layer. The DLS also verifies that the data received does not already exist in the database by checking the unique site identifier and datetime values of the data to avoid duplication of data in the database. The DLS uses the same configuration file as the DPS, described on Table 2. The DLS reads data files from a local/source directory and moves them to a local/target directory after successfully inserting the data into the operational database. If an error occurs, the files are moved to the quarantine directory. A log file records all the activity executed by the DLS, including any error observed in the process, such as invalid datetime stamps, invalid site identifiers, and attempts to load data that already exists in the database. This log file is named *data_loader.log*, and it is managed identically to the DPS log file. Both are located in the same folder (*log_directory* in Table 2).

We chose this implementation for several reasons. First, it enables preservation/archival of the original CSV data files recorded by the dataloggers. Second, the data are loaded into an operational database that is highly performant for querying and data retrieval in support of the Data Analytics Layer. Third, it enables all of the downstream components in the architecture to be used regardless of how the data files arrive on the server. For example, they can be automatically pushed to the server from the datalogger, pulled from the datalogger by the server (as in our second case study), or manually copied to the server in the case where data transmission is not automated. The DLS was implemented in a single Python script named *loader.py*.

3.1.2.3. Operational database. For the operational database component, we chose to use an existing technology given the availability of mature and robust database software. In our previous work related to investigating how to best manage large volumes of time series data, we tested the performance of four commonly used open source database technologies, including MongoDB, MySQL, PostgreSQL, and InfluxDB (Brewer, 2020). Based on our tests, we chose to use InfluxDB (InfluxData, 2021) due to its time series oriented data structure, rapid query performance, and favorable disk space requirements when compared to the other software technologies. InfluxDB is a popular time series database designed specifically for time series data in applications that require handling high data write and query loads. It provides a powerful structured query language (SQL)-like query language and has both open source distributions that can be installed and used for free (e.g., as we did on our Linux VM) and cloud deployments that can be implemented with usage-based pricing. InfluxDB has been used in multiple IoT and other applications, where it has been tested for large datasets (Balis et al., 2017; Di Martino et al., 2019; Rinaldi et al., 2019). InfluxDB also offers extensive support for multiple programming languages, including Python and R, which are commonly used for data science. This made it straightforward for us to use Python to insert data and to execute queries from the Data Analytics Layer.

InfluxDB databases are organized around the concept of a

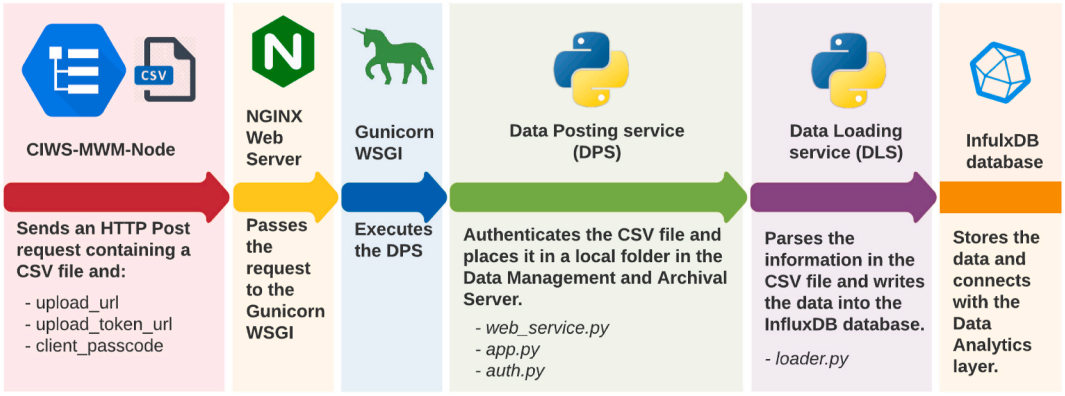


Fig. 2. Workflow and elements of the data management process for the push based implementation of the CIWS.

measurement, which can be thought of as a "table" that contains an indexed column named *time* containing the timestamp of each data *point*, where each data *point* is a row in the table. Additional variables are stored in columns that can be *tags* or *fields*. The main difference is that *tags* are indexed and are not required in a data structure, whereas at least one *field* is required, *fields* are not indexed. The column names for *tags* and *fields* are defined as *keys*. Generally, it is recommended that data values are stored as *fields*, and metadata as *tags* to improve query performance. In our design for storing data in InfluxDB, the number of pulses recorded by the datalogger during each time interval is included as a *field* (*key* = pulses), and the site identifier (*key* = siteID) and the datalogger identifier (*key* = dataloggerID) are included as *tags* (Table 3).

The data for all sites are stored in a single measurement within the Influx database. Raw data and quality controlled (QC) data are stored in separate measurements with the same structure. QC data is a copy of the raw data that is created after verifying that the volume registered by the datalogger is within ±5% of the volume registered by the meter (estimated using subsequent readings of the meter's register conducted during installation, during periodic site visits, and at removal of the datalogger). In some cases, known bad data were trimmed from the beginning and end of a valid deployment. Where the volume recorded by the datalogger did not match the volume recorded by the meter's register, the data were discarded and a new deployment was started. During our case study deployments, we did not observe any out of range, anomalous, or unreasonable pulse count values after this QC procedure. In consequence, additional QC modules were not implemented. However, additional QC procedures could be implemented in the future. All queries and analysis are conducted using the QC data.

The database is the point of connection between the Data Analytics Layer and the Data Management and Archival Layer, and its design must meet requirements from both layers to write and read data. Typically, database schemas are designed around the structure of the data to be stored and to facilitate the most common types of queries. This is usually a tradeoff between making it easy to insert data into the database while still providing highly performant queries. The simple database schema implemented in this case study (Table 3) mirrors the structure of the data files generated by the dataloggers, making it straightforward to insert data, but is also optimized to support the following queries: 1) selecting all of the data for a particular siteID; 2) selecting all of the data for a particular dataloggerID (e.g., to track the performance of a datalogger, which may be deployed at multiple sites at different times, and identify/correct any systematic errors); and 3) querying data for a specific time frame (e.g., between a beginning and ending date). Combining queries based on these three elements provides most of the functionality intended for CIWS and met all of the needs of our case study.

Table 3
InfluxDB database schema design in the push model implementation.

Influx Key	InfluxDB Type	Data Type	Example Value
time	Time Index	Timestamp	2020-01-01 00:00:01
siteID	Tag	String	"1"
pulses	Field	Integer	5
dataloggerID	Tag	String	"1"

Additional queries intended to allow comparison of data across multiple sites may also be of interest. Our design separates the time series data, which are stored anonymously in the InfluxDB database, from household information, which is stored in a separate CSV file, named *sites.csv*. The data stored in InfluxDB do not contain any identifiable information, which removes privacy concerns from the time series data. The separate *sites.csv* file may include sensitive, personally identifiable information (e.g., names, addresses, etc.) along with any other descriptive characteristics (the version of the *sites.csv* file for this study published in HydroShare has been anonymized). Data managers may wish to maintain multiple versions of the *sites.csv* file (e.g., one with all personally identifiable information about data collection sites and one that has been anonymized and could be released to a broader set of users). While this approach adds an additional step for certain types of queries (e.g., selecting data for all houses within a certain geographic area or of a certain built age) because the site information must be queried before the correct time series data can be retrieved, it provides a mechanism for protecting personally identifiable information and more flexibility for managing metadata about the sites. Removing or adding tags to existing measurements is significantly restricted in InfluxDB. In consequence, anonymizing the data stored in InfluxDB for publication is not needed, as the data stored is already anonymous. Queries against the time series data can always be executed using a siteID or set of siteIDs obtained via a prior query to the *sites.csv* file. It is also possible, but currently not implemented, to add all site metadata as tags in the InfluxDB measurement to eliminate this intermediate query step, if that is more convenient in a specific application.

Researchers and utility managers can access the data within the InfluxDB database with a non-administrator user account. InfluxDB allows for the creation of multiple non-administrator users and at least one administrator user. The administrator manages authorization for each non-administrator user. Non-administrator users can be restricted to write, read, or both. The free version of InfluxDB does not allow fine-grained authorization, which would be needed to restrict users to view only part of the data in a measurement. However, we did not see this as a significant drawback as high level users like researchers and/or utility managers would likely need to have unrestricted access to all of the data in an InfluxDB database. Furthermore, it is unlikely that the full resolution data would be provided to water consumers. Rather, a more likely scenario would be for a software application with a graphical user interface to be developed for presenting water consumers with feedback about their consumption. Authentication and authorization of users

could be handled separately by the software application in future deployments. Erickson et al. (2012) provide an example of an online water portal and discuss the privacy and user authorization concerns that impact the design of similar tools. Homeowners are typically presented with summary statistics and visualizations calculated for their property and may be provided with a summary-level comparison with other properties. However, they generally would not have access to view raw data for their own or other properties.

3.1.3. Data analytics layer

To illustrate the type of capabilities supported by the Data Analytics Layer, we developed Python tools that provide an example of the main aspects involved in this process: connection to the database, user authentication, and data retrieval via common queries. Once the data has been retrieved into a Python environment, it can be integrated with existing, and more advanced, data analysis and visualization tools. While it is beyond the scope of this paper to demonstrate all of the possible ways in which data can be retrieved from the database component and used within analytical applications, the tools we developed demonstrate the general patterns required for developing such tools and serve as a foundation on which others could be developed.

InfluxDB client programming libraries are available for several popular programming languages, including Python, Go, C#, Java, PHP, Ruby, Scala, JavaScript, and R, which simplifies software development using InfluxDB and facilitates desktop, mobile, and web application development. Using the Python client library for InfluxDB (InfluxDB, 2020), we first developed a set of functions for interacting with the InfluxDB database. These functions were implemented within a single Python script called *da_functions.py*. This script connects to the database using a set of configuration parameters that are included in a JSON file named *configuration.json*, which is similar to the one used by the DPS and DLS applications. Parameters in the JSON file include: *host, port, username, password, and database* (as defined in Table 2). The functions we developed in *da_functions.py* (Table 4) use the existing capabilities of the InfluxDB Python client library along with specific parameters provided by the user (e.g., siteID, time, dataloggerID as defined in Table 3) to provide a simple application programming interface (API) for querying data from the database. We anticipate that these functions will meet many of the most common data requirements for most researchers and utilities. The functions generate a Pandas dataframe (McKinney, 2010) with the resulting data if a single siteID is provided, and a Python list of Pandas dataframes when multiple siteIDs are provided. If a start date or end date are not included, the function will download the entire record available. If only a start date is provided the function will return everything from that date to the end of the record, in the opposite case, it will retrieve data from the beginning of the record to the specified ending data. If measurement is not provided, the functions will query from the quality controlled data (QCData). Raw data can be downloaded by specifying *measurement = 'RawData.'* For time aggregated data, the function parameter can include any Influx supported aggregation function (e.g., mean, median max, min, sum). The time resolution of the aggregated data supports any InfluxDB duration type (e.g., '1m' for 1 min data, '1h' for hourly data, '1d' for daily data, '1w' for weekly data). All the arguments in both functions are Python keyword arguments. They must be preceded by their identifier (or name) when executing the functions, i.e., *get_data(site = 1)* to return all the quality controlled data for siteID 1.

We then developed a Python Jupyter Notebook called *data_analytics.ipynb* that loads the functions listed and implements a basic workflow to produce metrics and analysis from the data collected. Jupyter Notebooks (Kluyver et al., 2016) allow creation and sharing of documents that contain live code, equations, visualizations and narrative text, which makes them ideal for prototyping visualizations and analyses for the Data Analytics Layer. The Notebook we developed imports data using the defined functions and then generates visualizations of common metrics of residential water use for presentation to water consumers. For example, Fig. 3 shows the average hourly water use (blue solid line), and the boxplots show the distribution of hourly water use for the period of data collection at the residential home we monitored. We can notice two periods of higher water usage, one during the morning and the other early in the afternoon, corresponding with patterns typically observed in hourly residential water use data. During this period, no outdoor water use occurred; therefore, the figure represents indoor water use only. The Notebook then demonstrates calculation of summary water use information for the data collection period. For example, average daily water use was 170.2 gallons (644.3 L), leading to a per capita average daily water use of 34 gallons (128.7 L). The maximum daily water usage observed during the period was 292.7 gallons (1077.9 L), the instantaneous peak was 10 gpm, or 37.95 L per minute (Lpm), and the maximum hourly usage registered was 74.1 gallons (280.5 L).

Another analysis of special interest using high-temporal resolution data is the identification of end uses of water. We used an open source algorithm developed by (Attallah et al., 2021), available via the HydroShare repository (Attallah and Bastidas Pacheco, 2021), within the Data Analytics layer to separate raw data into events and classify the resulting events into categories of end uses of water. The algorithm filters the data collected using a low-pass filter, making it easier identify single or concurrent events. Concurrent events are separated into single events, and the final table containing only single events is classified by using a combination of clustering to identify atypical or outlier events, and a fully-supervised machine learning methodology to assign labels to the remaining events. The machine learning model uses a Random Forest classifier (Liaw and Wiener, 2002) trained using a set of user-labeled and manually-labeled events to classify new events for individual residential homes (Attallah et al., 2021). We used the trained machine learning model to label the events generated during the data collection period at the residential home we monitored. While a potentially large number of analytics, visualization, and information can be generated from the labeled events, the Jupyter Notebook we developed presents a small subset of them (Fig. 4) as an example of products that can be generated from the raw data.

At the observed home, toilet events account for 36.1% of the total indoor volume used, showers 26.3%, clothes washer 13%, faucets 12.4%, and bathtub events 11.1%. Unclassified events, defined as events

Table 4
Functions implemented for querying data in the Data Analytics Layer.

Query	Python implementation
Get raw data for one or multiple sites, between specific dates, or the entire record.	*get_data*(site, startdate = None, enddate = None, measurement = 'QCData')
Get time aggregated data for one or multiple sites, between specific dates, or the entire record.	get_agg(site, function, t_res, startdate = None, enddate = None, Measurement = 'QCData')

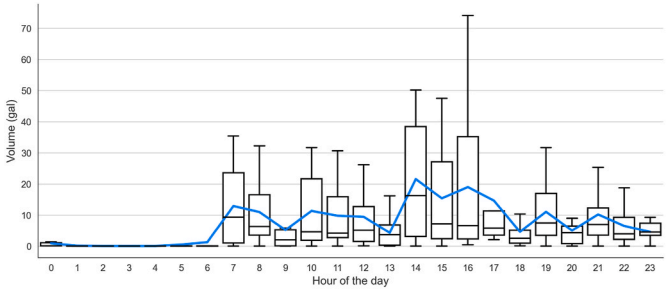


Fig. 3. Hourly distribution of water use for the single family residential home between January 15, 2021 and January 28, 2021. The blue solid line shows the hourly average water use, and the boxplot presents hourly water use variability.

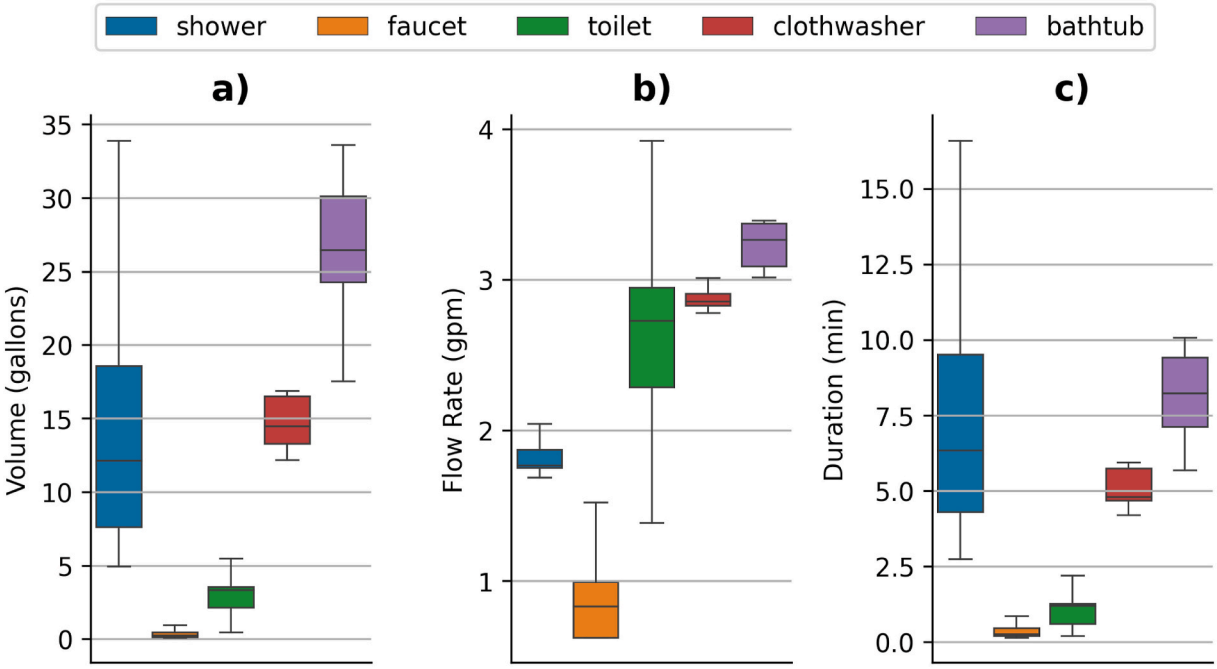


Fig. 4. Illustrative examples of high-temporal residential water use data analytics for the case study home between January 15, 2021 and January 28, 2021. The figure presents boxplots of a) the volume of events, b) the flow rate of events, c) the duration of events. In all cases, the data is grouped by end use type. Outliers were removed to improve the quality of visualization for short duration and low volume events (faucet and toilet events).

lasting 4 s or less and consisting of a single "pulse" recorded by the meter (approximately 5 ounces, or 0.15 L of water), account for approximately 1% of total use. Unclassified events include very short water use events (e.g., ice making refrigerators, short faucet events) and leaks. Fig. 4 shows the distribution of the volume a), flow rate b), and duration c) for each category of indoor water use. Unclassified events were excluded from Fig. 4. Faucet events had a median flow rate of approximately 0.8 gpm (3 Lpm). Water-efficient bathroom faucets, as defined by the United States (U.S.) Environmental and Protection Agency (EPA) in their Water Sense program (EPA, 2020), operate between 0.8 gpm at a pressure of 20 pounds per square inches (psi), or 137.9 Kilopascals (kpa), and 1.5 gpm (5.7 Lpm) at 60 psi (413.7 kpa). Compared to this EPA standard, the flowrates we observed from the faucets at the study property are efficient. A similar conclusion can be reached by comparing the median flow rate of shower heads at the study property (approximately 1.8 gpm, or 6.8 Lpm) with EPA Water Sense standards (limiting the maximum flow rate to 2.0 gpm, or 7.6 Lpm).

In previous studies from multiple U.S. cities, shower durations averaged 7.8 min (DeOreo et al., 2016). The average shower duration observed at the study property was approximately 8 min, with a median value of 6.3. Approximately 25% of the shower durations were longer than 9.5 min (Fig. 4). The average gallons per flush (gpf) for toilets at the study property was 2.78 (10.5 L), significantly higher than the 1.28 (4.8 L) recommended by the EPA (EPA, 2020), indicating there is potential for reducing water usage by retrofitting the property with water-efficient toilets. There is relatively little variability in the durations of toilet and clothes washer events, as observed in Fig. 4 c. For these events, the characteristics are dependent on the type, brand and setting used. Shower events reflect the largest variability, as expected, due to personal preferences of the different occupants of the property. Code to reproduce the results in this section and the raw data collected are publicly available in HydroShare (Bastidas Pacheco et al., 2021). The workflow that can be used to reproduce the results presented in this section consists of the following: a) InfluxDB is installed locally with instructions provided, b) the database described in Table 3 is created, c) the database is loaded with the raw data provided using *InfluxDB_Loading.ipynb*, and then d) *data_analytics.ipynb* is executed on the database, producing all the results described.

3.2. Case study 2: pull based data collection within multi-unit residential buildings

For results of this case study, we present only the data collection and management infrastructure required. The specifics details about estimating and water-related energy use estimates using the data collected are reported elsewhere by Brewer (2020). The functionality of the Data Analytics Layer is independent of the selected data communication method (push or pull) because the Data Analytics Layer interacts only with the operational database. Given that the data collected by both case studies and the resulting database are similar, the considerations for implementing the Data Analytics Layer are equivalent to those of the first case study presented (e.g., ability to support queries, data privacy, etc.) and the technology of the implementation would follow the same process. To avoid duplication of results, we have chosen not to present an implementation of the Data Analytics Layer with this case study. However, similar functionalities related to this case study are discussed in our previous work (Brewer, 2020) and available in an online data resource (Brewer and Horsburgh, 2020).

3.2.1. Data collection layer

An enhanced version of the water meter datalogger presented by Horsburgh et al. (2017) was used to collect data for the variables listed in Table 1. This device was named the CIWS-EWM-Logger, where EWM denotes "electronic water meter" for the electronic output signal of the meter types it works with. The CIWS-EWM-Logger was designed to be installed on commercial water meters of the types typically used in multi-unit residential buildings and where a dedicated power source is readily available at the meter's location. The CIWS-EWM-Logger also uses a Raspberry Pi 3 Model B or Model B + Linux computer running Raspberry Pi OS. The Raspberry Pi in this device controls the functioning of the datalogger and has integrated ethernet and Wi-Fi capabilities for connecting to a network while operating. Given the location of the water meters in utility closets with no wired ethernet ports, we chose to use Wi-Fi to enable communications with the dataloggers. Connecting a

device to USU's Wi-Fi network requires registration of the device's hardware address, after which, each device is assigned a unique host name that is routable on USU's network. Thus, each datalogger could be located and connected to within the network, which allowed for remote work interactions with the datalogger. For example, the firmware of the loggers could be updated, their functioning could be evaluated in real time, and data could be pulled from them via SSH at any time. While this specific configuration relies on characteristics of USU's Wi-Fi network, we anticipate that Wi-Fi networks like USU's would be available in many application contexts. The functionality described here would function identically for wired ethernet connections.

The CIWS-EWM-Logger was specifically modified to read the output of each of the meters available on the LLC buildings along with water temperature values from three separate sensors. The CIWS-EWM-Loggers we deployed can be used with any water meter or sensor that has a 4–20 mA current loop output, analog voltage output, digital output readable by the Raspberry Pi via its General Purpose Input/Output (GPIO) ports, or pulsed output. The Master Meter Octave meters provide output through a 4–20 mA current loop module where the output current is directly proportional to the flow rate through the meter. The necessary transformations from current to voltage and then to flow rate were performed by the CIWS-EWM-Logger (Brewer, 2020), and a time series of water flow in gallons per minute at a user-configurable temporal resolution was generated. The BLMJ meter outputs a pulsed signal (voltage) where every pulse represents a gallon of water that has passed through the meter. In this case, the count of pulses, which equals the number of gallons, was registered by the CIWS-EWM-Logger at the same user-configured temporal resolution. The DS18B20 digital thermometers provided digital 9-bit to 12-bit Celsius temperature measurements to an accuracy of $\pm$ 0.5 °C and were wired directly to the Raspberry Pi with a single wire for each sensor and do not require an external power supply.

The CIWS-EWM-Logger in each building logged data to a CSV file that was saved in a local directory within the Raspberry Pi's file system. For this deployment, data was collected at a 1-s time interval and includes the following columns: time (datetime of the measurement using the YYYY-MM-DD HH:MM:SS format), *buildingID* (B, C, D, E, or F), *coldInFlowRate, coldInTemp, hotInFlowRate, hotInTemp, hotOutFlowRate* and *hotOutTemp* with units indicated in Table 1. In the quality controlled data, the hot water return flow was transformed to gallons per minute for uniformity.

3.2.2. Data management and archival layer

To support pull based data retrieval, we developed an application called the Data Transfer Manager (DTM) to serve as the Request Service shown in Fig. 1. It was implemented as a single Python script named *transfer_manager.py* and follows the same convention used by the DPS and the DLS, reading configuration data from a JSON file. As in the first Case Study, the DTM and the operational database were deployed on a VM with similar characteristics to the one described in Section 3.1.2. We used InfluxDB as the operational database for this case study as well given the similarity in the type of data and requirements among both case studies and to show generalizability.

The DTM manages all data communications under the pull based model. Operation of the DTM was scheduled using Linux's native CRON functionality, which allows the user to specify how often the DTM program is executed. Upon being triggered by the scheduled CRON job, the DTM first reads the configuration file described in Table 5 and then proceeds through a list of defined tasks to manage transfer of data from each remote data collection site to the Data Management and Archival Layer:

1. Connect to each datalogger listed in the configuration file using Paramiko, a Python library that enables SSH connections for safely accessing network services over unsecured networks (Forcier, 2021).
2. Parse the datalogger's Linux file system for new datalog files and download them to the server with Secure File Transfer Protocol (SFTP), an extension of SSH that offers secure file transfer capabilities over any reliable data stream. Tasks 1 and 2 in this list are executed by a function named *connect()* in the *transfer_manager.py* Python script.
3. Upload new data into the InfluxDB database. This task is completed by the *write_to_db()* function in the *transfer_manager.py* Python script.

Table 5
Parameters included in the configuration file for the DTM. The configuration file follows the structure presented here.

Parameter		Description
connections		The number of threads used for concurrent connection with hosts.
log_directory		Path where the log files are stored in the Data Management and Archival server (must have write permissions for that directory).
hosts		A list of datalogger host names or IP addresses to connect to.
database	name	Name of the InfluxDB database to connect to.
	user	Username for a user with permission to write data to the InfluxDB database.
	password	Password for a user with permission to write data to the InfluxDB database.
	host	Database server hostname or IP address.
	port	The port number over which communications with the InfluxDB database server have been configured.
	measurement	Name of InfluxDB Measurement where the data will be saved.
sshinfo	username	Username used to connect to remote dataloggers via SSH.
	password	Password used to connect to remote dataloggers via SSH.
slack_webhook		Slack webhook to send error messages through the Slack messaging service.

An additional function in the DTM, named *send_error(),* was developed to inform data managers about errors in the data transfer process. Errors are sent via Slack, a cloud-based instant messaging service (Slack Technologies, 2021). Messages are formulated as a JSON payload that is sent to a unique URL provided by Slack as a webhook. Information detailing which datalogger file caused the error is included in the message. Fig. 5 describes the overall functionality of the DTM, indicating the key tasks mentioned. For this case study, data transferring and parsing are executed by a single element (*transfer_manager.py*), which requires fewer moving parts and minimizes the amount of time between the data being retrieved from the remote dataloggers and having them show up in the operational InfluxDB database. This is a slightly different approach than the one presented for Case Study 1, which allows more flexibility in the system. The DTM can work concurrently on a user defined number of datalogger devices at the same time (*connections* in Table 5). The optimal number of threads is dependent on the number of CPU cores of the server. For our testing, we set the number of threads to 6, matching the number of dataloggers in the LLC buildings.

As in the first case study, the raw data and quality controlled data were stored in the same InfluxDB database in different measurements. Brewer (2020) describes the quality control procedures for the data collected in this case study. The database schema used for this case study is similar in structure to that of the first case study. The data included in the database copies all columns from the CSV files recorded by the dataloggers. BuildingID serves as the SiteID and is the only column stored as a tag. All additional variables (the recorded data values for each variable) are stored as fields.

3.3. Scalability and Performance Metrics

While we experienced no performance issues in the case study deployments, we performed scalability testing to investigate the

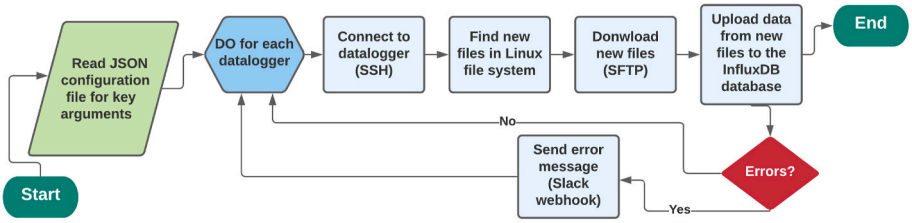


Fig. 5. General functionality of the DTM.

performance of the system beyond the scale of our case studies. We conducted individual tests of the DPS, the DLS, and the DTM, simulating larger numbers of dataloggers and HTTP POST requests, in the case of the DPS and DLS, and a larger number of remote datalogger hosts, in the case of the DTM, to be processed by the system.

Scalability of the DPS is dependent upon its ability to handle many HTTP POST requests from many dataloggers posting data at the same time. The DPS was tested by sending multiple HTTP POST requests, each with a CSV file containing one day of randomly generated data with values recorded every 4 s (for consistency with the implementation of Case Study 1). The files were sent using a Python script implemented using the Asyncio library (Python Software Foundation, 2021a) from a MacBook Pro laptop computer with a 2.3 GHz 8-Core Intel Core i9 processor and 16 GB of memory. Asyncio is a library that can be used to write code that executes concurrently, allowing the code to send multiple simultaneous, or nearly simultaneous, requests to the DPS. There are limitations in the number of concurrent requests that can be sent from the same computer, as well as in the number of dataloggers that can send data at the exact same time in a filed deployment, considering computing power, speed of connection, and synchronization.

We simulated an increasing number of concurrent HTTP POST requests to the DPS (10, 50, 100, 200 and finally 500), and each operation was repeated ten times to characterize server/network variability. The total duration of each repetition, calculated as the end time of the last HTTP POST request minus the start time of the first request, on average, was 0.6 s, 2.05 s, 3.58, 6.91 s, and 16.7 s for 10, 50, 100, 200, and 500 requests, respectively. We observed no transmission errors or requests rejected by the server during our testing process. Fig. 6 shows the durations of HTTP POST requests, separated by the batch size (10, 50, 100, 200, and 500) for each one of the 10 repetitions conducted. We observed that the median duration of POST requests was larger for the 10-request batches compared to all other batches, but longer durations were observed for some requests in larger batches, which is expected as the DPS is busy with an increasing number of requests. Median times are consistent for batches with more than 50 POST requests. These times are affected by the processing power of the machine sending the request, the resources available on the remote server, and the speed and quality of the Internet connection but are provided here as an indicator of the performance of our prototype implementation. These tests indicate that the DPS can handle 500 nearly simultaneous POST requests in under 20 s with most individual requests being handled in under 0.2 s.

To test the DLS, we simulated different data loading scenarios ranging from loading one CSV file for a single site to loading one file for 500 sites. The testing procedure consisted of placing CSV files containing one day of data with values recorded every 4 s in the source directory and then executing the DLS. Each operation was repeated ten times. Table 6 presents the mean and standard deviation of each scenario along with the average time for loading a single file to facilitate comparisons. The DLS can load 1 day of data from 100 different sites in less than 50 s. There are differences between loading n files from the same site and loading 1 file from n sites, which can be explained by the way data are organized within the InfluxDB database. Although all of the data values are stored in the same InfluxDB measurement, InfluxDB logically groups data values by shared measurement, tag set, and field key. Writing data with multiple siteID tag values takes longer. Both scenarios are realistic applications. The first scenario (n files from 1 site) simulates loading data collected from dataloggers lacking communication technologies. The second scenario (1 file from n sites) represents a deployment like the one described in Case Study 1 with a larger number of sites.

We used the six dataloggers described in Case Study 2 to test the DTM. Each data logger sent 1 day of data during all tests. The functionality that allows the system to identify existing data or files was removed, allowing the system to upload existing CSV files and re-write existing data to the InfluxDB without restrictions. This configuration

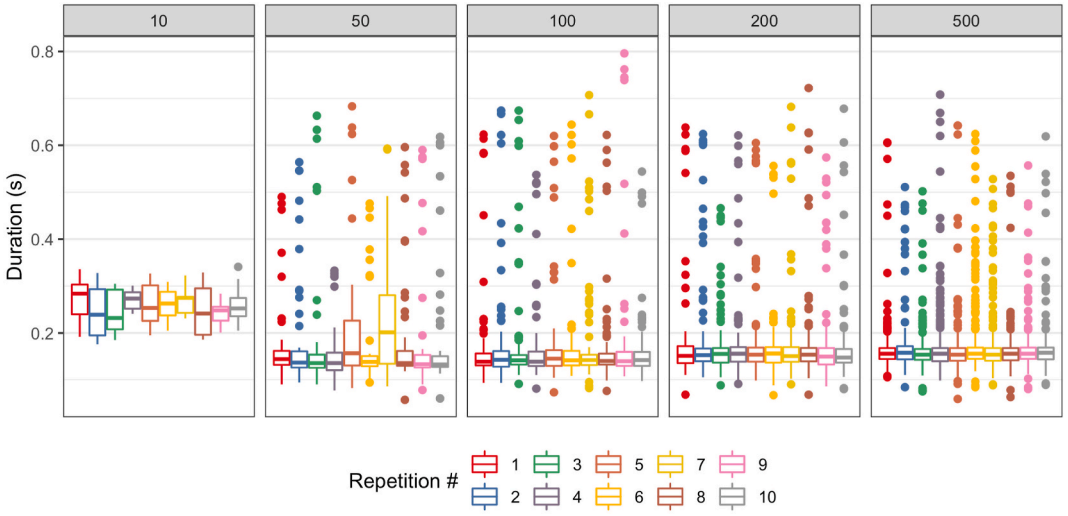


Fig. 6. Boxplot of processing times, separated by the number of HTTP POST requests in the batch (10, 50, 100, 200, and 500) for each repetition, from 1 to 10. Duration is calculated as the final processing time minus the starting time of each individual POST request.

Table 6
Results from the DLS testing. Every operation was repeated 10 times.

Load Operation	Average duration (seconds)	Standard deviation (seconds)	Average time for processing 1 file (seconds)
1 file from 1 site	0.37	0.06	0.37
10 files from 1 site	3.96	0.14	0.40
1 file from 10 sites	4.67	0.23	0.47
50 files from 1 site	19.92	0.67	0.40
1 file from 50 sites	23.87	0.33	0.48
100 files from 1 site	39.87	1.05	0.40
1 file from 100 sites	47.48	0.89	0.47
500 files from 1 site	195.19	2.98	0.39
1 file from 500 sites	240.70	3.00	0.48

enabled us to simulate a larger number of connections by repeating dataloggers in the hosts list included in the DTM configuration file (described in Table 5). The number of dataloggers was gradually increased (6, 48, 96, and finally 480), and the DTM was executed ten times for each number of dataloggers, processing one CSV file containing one day of 1-s resolution data for each datalogger. The DTM was set to execute six threads at a time, meaning that it can be simultaneously connected to and downloading data from six dataloggers at a time, for consistency with the application of Case Study 2. During our testing, only 6 dataloggers were available, which meant that it was possible for the DTM to attempt connecting to and processing data from the same logger multiple times simultaneously. This can negatively affect the time reported if a host is not immediately available for processing when the system is trying to connect to it. Table 7 lists the duration and standard deviation after ten runs with an increasing number of datalogger hosts. Using our test configuration, it took less than 50 min for the DTM to process data from 480 hosts.

We tested the system up to, and with much larger numbers than the 40–60 sites in our design considerations and observed no real limitations for using CIWS in deployments roughly an order of magnitude larger, even with our relatively limited testing server. The DLS and the DTM include writing to the database as part of their tasks, and the times observed satisfy the stated requirements for our application. As a final test, we tested the database by conducting standard queries from a Python environment, using the same laptop computer. We observed the amount of time required to downloaded one day, one week, and one month of data for 1, 5, and 10 sites along with the time required to load the data into a Pandas dataframe object (Table 8). All queries were conducted using the function *get_data()* described on Table 4. The *timeit* Python module (Python Software Foundation, 2021b) was used to repeat each query 10 times and measure execution times. Downloading one month of data (a common record length in studies collecting high resolution residential water use data) for ten sites into a Pandas dataframe takes less than 1 min. The log files and code to reproduce all the results of this section are publicly available in HydroShare (Bastidas Pacheco et al., 2021).

The cost of deploying CIWS to support data collection at residential houses using the equipment described for Case Study 1 can be broken down as follows: a) the cost of CIWS-NODE Datalogger devices, which is approximately $180 multiplied by the number of houses to be enrolled simultaneously, and; b) the cost of hosting a server with characteristics similar to our testing server (4 processor cores, 8 GB of memory, 100 GB of storage). At the time of this writing, hosting this machine using the Amazon Elastic Compute Cloud would cost approximately $57 per month (Amazon, 2021), although there are multiple hosting alternatives for the server that could be used and that would impact the cost estimate provided. The approximated cost of building the datalogger device used in Case Study 2 is $85.

Table 7
Results from the DTM testing.

Number of datalogger hosts	Average duration (seconds)	Standard deviation (seconds)
6	41.7	1.57
48	279.4	9.35
96	551.5	9.21
480	2831	252

Table 8
InfluxDB downloading times for different queries. In all cases the data was downloaded and loaded into a Pandas dataframe.

Days of data	Number of sites	Average duration (seconds)	Standard deviation (seconds)
1	1	0.17	0.02
1	5	0.81	0.03
1	10	1.62	0.04
7	1	1.16	0.04
7	5	5.74	0.07
7	10	11.39	0.07
30	1	4.51	0.27
30	5	22.46	0.52
30	10	45.47	1.24

4. Conclusions and future work

A complete cyberinfrastructure system that uses a layered approach to collect and manage high-temporal resolution water use data was developed and implemented. The system was designed focusing on the scale of data collection that would be required for research projects conducted by utilities or other researchers. Having a standardized cyberinfrastructure like CIWS can increase the value of the data collected by allowing more straightforward data collection and management, as well as facilitating the analysis and understanding of data collected in different projects, cities and utilities. CIWS can be used to manage data collected or used for multiple purposes - e.g., collecting data to support estimates of design parameters for future home developments, guiding the planning of water conservation campaigns, assessing the effectiveness of rebate programs, assisting in the definition of utility rates, and defining future demand and infrastructure needs.

Our case studies showed that CIWS can work with any datalogging devices that generate CSV files containing time series of water use data, but it can also be used in the collection of other variables, as demonstrated in experimental Use Case 2. By integrating low cost data collection devices and open-source cyberinfrastructure we sought to increase the accessibility of tools for conducting high-temporal resolution data collection in support of residential water use studies. CIWS can reduce not only the cost of such studies, but also technical barriers by providing a framework to collect and manage the data.

CIWS can manage push and pull based data communication. Since each functionality is implemented separately, future users of CIWS can select push or pull, or a combination of both, depending on the needs and settings of their application. The work performed within the Data Management and Archival Layer depends on whether the push or pull model is used. In the pull case, the data is pulled from the device by a request service, whereas in the push case the data is managed by a network listener web service that accepts incoming files and processes them. Both use the same database component, which means that the Data Analytics Layer can operate independent of how the data are transferred. The demonstrations we presented of the Data Analytics Layer serve as a proof of concept and show the foundation upon which more sophisticated tools could be built that can be used to communicate results with multiple interested parties.

We focused our design and implementation on a system that is capable of transferring high temporal resolution water use data from water meters to a centralized infrastructure for storage and subsequent analysis. In a research context, this is preferable, as researchers may not know at the outset of a study all of the specific analyses they may want to

perform with the data and, thus, keeping all of the data is necessary. However, transferring large volumes of data to a centralized data management system poses challenges when scaling a system like this to larger deployments. While technically possible over Wi-Fi or cellular data networks, the availability of Wi-Fi is limited, and cost of data transfer over a cellular data network may be prohibitive. As an alternative, we are now investigating edge computing techniques using our CIWS-WM-Node datalogger to process the high resolution water use data on the logger to produce summary data products that are much smaller and can be transferred over a network with far less bandwidth and at lower cost. The tradeoff is that the full resolution data are never transferred or saved in the long term.

CIWS combines multiple open-source technologies. The modular design makes it easier to replace or update technology elements in the system, if needed. Similarly, additional tools can be added to system - e.g., more advanced analytics tools and enhanced authentication protocols. The analytics presented show potential for conservation programs and can assist in the design of future urban water infrastructure. All of the components we developed are publicly available for reuse, and we envision future improvements to the system once the tools are used in other studies. The system testing, performance metrics, and deployment demonstrate that CIWS can meet and significantly exceed the design considerations in terms of scale and performance. We saw no impediment for using CIWS, or a similar system in larger deployments than the ones tested, by increasing the processing power of the virtual machine, or deploying multiple instances. The server we used for testing had only moderate system specifications and could either be run on private server hardware or could easily be hosted within a commercial cloud service provider at a reasonable monthly cost.

Software and data availability

Name of Software: Cyberinfrastructure for Intelligent Water Supply (CIWS);

Developers: Camilo J. Bastidas Pacheco, Joseph C. Brewer, Jeffery S. Horsburgh, Juan Caraballo, Elijah West.

Contact: jeff.horsburgh@usu.edu.

Year First Available: 2021.

Required hardware and software: We used open source dataloggers for the data collection efforts in this study. Datalogger hardware details are provided by Bastidas Pacheco et al. (2020) and Horsburgh et al. (2017). Data management and archival components of CIWS were designed to run on a Linux server and were tested using Ubuntu. The data analytics components we demonstrate require a computer running the Windows, Linux, or Macintosh operating system. Instructions for how to deploy the system are available in the project's GitHub repository.

Availability: Source code for the Data Management and Archival Layer software components described in this manuscript is freely available and can be downloaded from the CIWS Server GitHub repository (https://github.com/UCHIC/CIWS-Server). The *src* folder in that repository contains a folder named *ciws_ci* and a folder named *data_transfer_manager* where the elements related to Case Study 1 and Case Study 2 are located, respectively. The *doc* folder contains a deployment guide for CIWS. The data described in Case Study 1 and the source code of the Data Analytics Layer software are publicly available in HydroShare (Bastidas Pacheco et al., 2021) with instructions for reproducing the results presented in that section. The data described in Case Study 2 and tools used to analyze it are also publicly available in HydroShare (Brewer and Horsburgh, 2020). The log files from Section 3.3 (Scalability and Performance Metrics) and code used to generate the results presented are available in HydroShare (Bastidas Pacheco et al., 2021). Design files, instructions for assembly, and firmware for the open source dataloggers are available on the GitHub sites for the CIWS Water Meter Node datalogger (https://github.com/UCHIC/CIWS-WM-Node) and the CIWS Electronic Output Water Meter datalogger (https://github.com/UCHIC/CIWS-EWM-Logger).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research was funded by the United States National Science Foundation under grant number 1552444. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would like to acknowledge Providence City and Utah State University Housing and Facilities for their cooperation and support in the data collection efforts. The authors would also like to acknowledge support from Nour Atallah, Arle J. Beckwith, and Rob J. Tracy in the data collection efforts and Elijah West for his contribution in software development. We also acknowledge and thank the owner of the residential home and the students in the LLC buildings that participated in the data collection campaign.

References

Alghamdi, A., Shetty, S., 2016. Survey toward a smart campus using the internet of things. In: Proceedings - 2016 IEEE 4th International Conference on Future Internet of Things and Cloud, FiCloud 2016. Institute of Electrical and Electronics Engineers Inc., pp. 235–239. https://doi.org/10.1109/FiCloud.2016.41

Alvisi, S., Casellato, F., Franchini, M., Govoni, M., Luciani, C., Poltronieri, F., Riberto, G., Stefanelli, C., Tortonesi, M., 2019. Wireless middleware solutions for smart water metering. Sensors 19, 1853. https://doi.org/10.3390/s19081853.

Amaxilatis, D., Chatzigiannakis, I., Tselios, C., Tsironis, N., Niakas, N., Papadogeorgos, S., 2020. A smart water metering deployment based on the fog computing paradigm. Appl. Sci. 10, 1965. https://doi.org/10.3390/app10061965.

Anda, M., Le Gay Brereton, F., Brennan, J., Paskett, E., 2013. Smart metering infrastructure for residential water efficiency: results of a trial in a behavioural change program in Perth, Western Australia. In: Information and Communication Technologies for Sustainability. ETH Zurich, Zurich. https://researchrepository.murdoch.edu.au/id/eprint/22422/. (Accessed 3 April 2021).

Ardito, L., Procaccianti, G., Menga, G., Morisio, M., 2013. Smart grid technologies in europe: an overview. Energies 6, 251–281. https://doi.org/10.3390/en6010251.

Balis, B., Bubak, M., Harezlak, D., Nowakowski, P., Pawlik, M., Wilk, B., 2017. Towards an operational database for real-time environmental monitoring and early warning systems. In: Procedia Computer Science. Elsevier B.V., pp. 2250–2259. https://doi.org/10.1016/j.procs.2017.05.193

Bastidas Pacheco, C.J., Horsburgh, J.S., Tracy, R.J., 2020. A low-cost, open source monitoring system for collecting high temporal resolution water use data on magnetically driven residential water meters. Sensors 20, 3655. https://doi.org/10.3390/s20133655.

Bastidas Pacheco, C.J., Horsburgh, J.S., Caraballo, J., Attallah, N., 2021. Supporting Data and Tools for "An Open Source Cyberinfrastructure for Collecting, Processing, Storing and Accessing High Temporal Resolution Residential Water Use Data. HydroShare. https://doi.org/10.4211/hs.aaa7246437144f2390411ef9f2f4ebd0.

Beal, C.D., Flynn, J., 2015. Toward the digital water age: survey and case studies of Australian water utility smart-metering programs. Util. Pol. 32, 29–37. https://doi.org/10.1016/j.jup.2014.12.006.

Beal, C., Stewart, R.A., 2011. South East Queensland Residential End Use Study: Final Report, Urban Water Security Research Alliance. https://research-repository.griffith.edu.au/bitstream/handle/10072/46802/80687_2.pdf?sequence=1. (Accessed 3 February 2021).

Beal, C.D., Stewart, R.A., Fielding, K., 2013. A novel mixed method smart metering approach to reconciling differences between perceived and actual residential end use water consumption. J. Clean. Prod. 60, 116–128. https://doi.org/10.1016/j.jclepro.2011.09.007.

Boyle, T., Giurco, D., Mukheibir, P., Liu, A., Moy, C., White, S., Stewart, R., 2013. Intelligent metering for urban water: a review. Water 5, 1052–1081. https://doi.org/10.3390/w5031052.

Brewer, Joseph C., 2020. Characterizing Water and Water-Related Energy Use in Multi-Unit Residential Structures with High Resolution Smart Metering Data. All Graduate Theses and Dissertations, p. 7976. https://doi.org/10.26076/669a-93b0.

Brewer, J., Horsburgh, J.S., 2020. Characterizing Water and Water-Related Energy in Multi-Unit Residential Structures with High Resolution Smart Meter Data. HydroShare. http://www.hydroshare.org/resource/b6bbdcd9b120430b9a54974a798961f1.

Cardell-Oliver, R., 2013. Water use signature patterns for analyzing household consumption using medium resolution meter data. Water Resour. Res. 49, 8589–8599. https://doi.org/10.1002/2013WR014458.

Chen, F., Dai, J., Wang, B., Sahu, S., Naphade, M., Lu, C.T., 2011. Activity analysis based on low sample rate smart meters. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press, New York, New York, USA, pp. 240–248. https://doi.org/10.1145/2020408.2020450.

Christie, M.A., Bhandar, A., Nakandala, S., Marru, S., Abeysinghe, E., Pamidighantam, S., Pierce, M.E., 2020. Managing authentication and authorization in distributed science gateway middleware. Future Generat. Comput. Syst. 111, 780–785. https://doi.org/10.1016/j.future.2019.07.018.

Cominola, A., Giuliani, M., Piga, D., Castelletti, A., Rizzoli, A.E., 2015. Benefits and challenges of using smart meters for advancing residential water demand modeling and management: a review. Environ. Model. Software 72, 198–214. https://doi.org/10.1016/j.envsoft.2015.07.012.

Cominola, A., Giuliani, M., Castelletti, A., Rosenberg, D.E., Abdallah, A.M., 2018. Implications of data sampling resolution on water use simulation, end-use disaggregation, and demand management. Environ. Model. Software 102, 199–212. https://doi.org/10.1016/j.envsoft.2017.11.022.

DeOreo, W.B., Mayer, P.W., Martien, L., Hayden, M., Funk, A., Kramer-Duffield, M., Davis, R., Henderson, J., Raucher, B., Gleick, P., 2011. California Single-Family Water Use Efficiency Study, Report Prepared for the California Dept. Of Water Resources. Aquacraft Inc., Boulder, CO. https://cawaterlibrary.net/document/california-single-family-water-use-efficiency-study/. (Accessed 12 April 2021).

DeOreo, W.B., Mayer, P.W., Dziegielewski, B., Kiefer, J., Foundation, W.R., 2016. Residential End Uses of Water, Version 2. Water Research Foundation. https://www.waterrf.org/resource/residential-end-uses-water-version-2. (Accessed 5 May 2021).

Di Martino, S., Fiadone, L., Peron, A., Vitale, V.N., Riccabone, A., 2019. Industrial internet of things: persistence for time series with NoSQL databases. In: Proceedings - 2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2019. Institute of Electrical and Electronics Engineers Inc., pp. 340–345. https://doi.org/10.1109/WETICE.2019.00076

Di Mauro, A., Di Nardo, A., Santonastaso, G.F., Venticinque, S., 2019. An IoT system for monitoring and data collection of residential water end-use consumption. In: 28th International Conference on Computer Communication and Networks (ICCCN), pp. 1–6. https://doi.org/10.1109/ICCCN.2019.8847120.

Di Mauro, A., Cominola, A., Castelletti, A., Di Nardo, A., 2020. Urban water consumption at multiple spatial and temporal scales. A review of existing datasets. Water 13, 36. https://doi.org/10.3390/w13010036.

Erickson, T., Podlaseck, M.E., Sahu, S., Dai, J.D., Chao, T., Naphade, M., 2012. The Dubuque Water Portal: evaluation of the uptake, use and impact of residential water consumption feedback. In: Conference on Human Factors in Computing Systems - Proceedings. ACM Press, New York, New York, USA, pp. 675–684. https://doi.org/10.1145/2207676.2207772.

Fang, D., Chen, B., 2017. Linkage analysis for the water–energy nexus of city. Appl. Energy 189, 770–779. https://doi.org/10.1016/j.apenergy.2016.04.020.

Froehlich, J., Findlater, L., Ostergren, M., Ramanathan, S., Peterson, J., Wragg, I., Larson, E., Fu, F., Bai, M., Patel, S.N., Landay, J.A., 2012. The design and evaluation of prototype eco-feedback displays for fixture-level water usage data. In: Conference on Human Factors in Computing Systems - Proceedings. ACM Press, New York, New York, USA, pp. 2367–2376. https://doi.org/10.1145/2207676.2208397.

F.S. Brainard & Company, 2020. Meter-Master Model 100EL and 100AF Flow Recorders, 2020. https://meter-master.com/product/model-100el-100af/. (Accessed 4 April 2021).

Giurco, D., Carrard, N., McFallan, S., Nalbantoglu, M., Inman, M., Thornton, N., White, S., 2008. Residential End Use Measurement Guidebook: A Guide to Study Design, Sampling and Technology. Prepared by the Institute for Sustainable Futures UTS and CSIRO for the Smart Water Fund, Victoria. https://opus.lib.uts.edu.au/bitstream/10453/35089/1/giurcoetal2008resenduse.pdf. (Accessed 5 April 2021).

Hachmann, J., Afzal, M.A.F., Haghighatlari, M., Pal, Y., 2018. Building and deploying a cyberinfrastructure for the data-driven design of chemical systems and the exploration of chemical space. Mol. Simulat. 44, 921–929. https://doi.org/10.1080/08927022.2018.1471692.

Hamiche, A.M., Stambouli, A.B., Flazi, S., 2016. A review of the water-energy nexus. Renew. Sustain. Energy Rev. 65, 319–331. https://doi.org/10.1016/j.rser.2016.07.020.

Hauser, A., Roedler, F., 2015. Interoperability: the key for smart water management. Water Supply 15, 207–214. https://doi.org/10.2166/ws.2014.096.

Hauser, A., Sud, T., Nicolas Foret, C., Electric Stuart Combellack, S., Jonathan Coome, T., Quintilia Lopez, S., Elkin Hernandez, I., Water Salil Kharkar, D.M., Water Amin Rasekh, D., Michal Koenig, S., Remy Marcotorchino, Q., Nicolas Damour, S., 2016. Communication in Smart Water Networks SWAN Forum Interoperability Workgroup. https://www.swan-forum.com/wp-content/uploads/sites/218/2020/12/SWAN-White-Paper_Communication-Protocols.pdf. (Accessed 3 May 2021).

Heiland, R., Koranda, S., Marru, S., Pierce, M., Welch, V., 2015. Authentication and authorization considerations for a multi-tenant service. In: SCREAM 2015 - Proceedings of the 2015 Workshop on the Science of Cyberinfrastructure: Research, Experience, Applications and Models, Part of HPDC 2015. Association for Computing Machinery, Inc, New York, New York, USA, pp. 29–35. https://doi.org/10.1145/2753524.2753534.

Hollands, R.G., 2008. Will the real smart city please stand up? Intelligent, progressive or entrepreneurial? City 12, 303–320. https://doi.org/10.1080/13604810802479126.

Horsburgh, J.S., Leonardo, M.E., Abdallah, A.M., Rosenberg, D.E., 2017. Measuring water use, conservation, and differences by gender using an inexpensive, high frequency metering system. Environ. Model. Software 96, 83–94. https://doi.org/10.1016/j.envsoft.2017.06.035.

Horsburgh, J.S., Caraballo, J., Ramírez, M., Aufdenkampe, A.K., Arscott, D.B., Damiano, S.G., 2019. Low-cost, open-source, and low-power: but what to do with the data? Front. Earth Sci. 7 https://doi.org/10.3389/feart.2019.00067.

Kenway, S.J., Binks, A., Scheidegger, R., Bader, H.-P., Pamminger, F., Lant, P., Taimre, T., 2016. Household analysis identifies water-related energy efficiency opportunities. Energy Build. 131, 21–34. https://doi.org/10.1016/j.enbuild.2016.09.008.

Kim, H., Lee, E.A., 2017. Authentication and authorization for the internet of things. IT Prof 19, 27–33. https://doi.org/10.1109/MITP.2017.3680960.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., 2016. Jupyter Notebooks—a publishing format for reproducible computational workflows. In: Loizides, F., Scmidt, B. (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas - Proceedings of the 20th International Conference on Electronic Publishing, ELPUB 2016. IOS Press BV, pp. 87–90. https://doi.org/10.3233/978-1-61499-649-1-87.

Kofinas, D.T., Spyropoulou, A., Laspidou, C.S., 2018. A methodology for synthetic household water consumption data generation. Environ. Model. Software 100, 48–66. https://doi.org/10.1016/j.envsoft.2017.11.021.

Li, J., Yang, X., Sitzenfrei, R., 2020. Rethinking the framework of smart water system: a review. Water 12, 412. https://doi.org/10.3390/w12020412.

Liaw, A., Wiener, M., 2002. Classification and regression by randomForest. R. News 2, 18–22.

Liu, X., Nielsen, P.S., 2016. A hybrid ICT-solution for smart meter data analytics. Energy 115, 1710–1722. https://doi.org/10.1016/j.energy.2016.05.068.

Liu, A., Giurco, D., Mukheibir, P., 2015. Motivating metrics for household water-use feedback. Resour. Conserv. Recycl. 103, 29–46. https://doi.org/10.1016/j.resconrec.2015.05.008.

Makropoulos, C., 2017. Thinking platforms for smarter urban water systems: fusing technical and socio-economic models and tools. Geol. Soc. Spec. Publ. 408, 201–219. https://doi.org/10.1144/SP408.4.

Mason, S.J.K., Cleveland, S.B., Llovet, P., Izurieta, C., Poole, G.C., 2014. A centralized tool for managing, archiving, and serving point-in-time data in ecological research laboratories. Environ. Model. Software 51, 59–69. https://doi.org/10.1016/j.envsoft.2013.09.008.

Mayer, P.W., DeOreo, W.B., Optiz, E.M., Kiefer, J.C., Davis, W.Y., Dziegielewski, B., Nelson, J.O., 1999. Residential End Uses of Water. American Water Works Association. https://www.sdu.dk/~/media/Files/Om_SDU/Institutter/ITI/Forskning/NATO%20ARW/Literature/Residential%20end%20uses_of%20water.pdf.

Mayer, P.W., DeOreo, W.B., Towler, E., Martien, L., Lewis, D.M., 2004. Tampa Water Department Residential Water Conservation Study: the Impacts of High Efficiency Plumbing Fixture Retrofits in Single-Family Homes. A Report Prepared for Tampa Water Department and the United States Environmental Protection Agency.

McKinney, W., 2010. Data structures for statistical computing in Python (SCIPY). In: van der Walt, S., Millman, J. (Eds.), Proceedings of the 9th Python in Science Conference, pp. 56–61. https://doi.org/10.25080/Majora-92bf1922-00a.

Moy De Vitry, M., Schneider, M.Y., Wani, O., Manny, L., Leitao, J.P., Eggimann, S., 2019. Smart urban water systems: what could possibly go wrong? Environ. Res. Lett. https://doi.org/10.1088/1748-9326/ab3761.

Mutchek, M., Williams, E., 2014. Moving towards sustainable and resilient smart water grids. Challenges 5, 123–137. https://doi.org/10.3390/challe5010123.

Neirotti, P., De Marco, A., Cagliano, A.C., Mangano, G., Scorrano, F., 2014. Current trends in smart city initiatives: some stylised facts. Cities 38, 25–36. https://doi.org/10.1016/j.cities.2013.12.010.

Nguyen, K.A., Stewart, R.A., Zhang, H., Jones, C., 2015. Intelligent autonomous system for residential water end use classification: Autoflow. Appl. Soft Comput. 31, 118–131. https://doi.org/10.1016/j.asoc.2015.03.007.

Rinaldi, S., Bonafini, F., Ferrari, P., Flammini, A., Sisinni, E., Bianchini, D., 2019. Impact of data model on performance of time series database for internet of things applications. In: I2MTC 2019 - 2019 IEEE International Instrumentation and Measurement Technology Conference, Proceedings. https://doi.org/10.1109/I2MTC.2019.8827164.

Robles, T., Alcarria, R., Martin, D., Morales, A., Navarro, M., Calero, R., Iglesias, S., Lopez, M., 2014. An internet of things-based model for smart water management. In: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014. IEEE Computer Society, pp. 821–826. https://doi.org/10.1109/WAINA.2014.129.

Shams, S., Goswami, S., Lee, K., Yang, S., Park, S.J., 2018. Towards distributed cyberinfrastructure for smart cities using big data and deep learning technologies. In: Proceedings - International Conference on Distributed Computing Systems. Institute of Electrical and Electronics Engineers Inc., pp. 1276–1283. https://doi.org/10.1109/ICDCS.2018.00127

Souffront Alcantara, M.A., Kesler, C., Stealey, M.J., Nelson, E.J., Ames, D.P., Jones, N.L., 2017. Cyberinfrastructure and web apps for managing and disseminating the national water model. J. Am. Water Resour. Assoc. 54 (4), 859–871. https://doi.org/10.1111/1752-1688.12608.

Stiri, S., Chaoub, A., Bennani, R., Lakssir, B., Tamtaoui, A., 2019. Internet of things connectivity-based smart grids in Morocco: proof of concept and guide to massive deployments. In: 5th IEEE International Smart Cities Conference, ISC2 2019. Institute of Electrical and Electronics Engineers Inc., pp. 129–135. https://doi.org/10.1109/ISC246665.2019.9071734

Stocks, K.I., Schramski, S., Virapongse, A., Kempler, L., 2019. Geoscientists' perspectives on cyberinfrastructure needs: a collection of user scenarios. Data Sci. J. 18, 1–15. https://doi.org/10.5334/dsj-2019-021.

Sun, Z., Di, L., Cash, B., Gaigalas, J., 2020. Advanced cyberinfrastructure for intercomparison and validation of climate models. Environ. Model. Software 123. https://doi.org/10.1016/j.envsoft.2019.104559.

Szwilski, T.B., Smith, J., Chapman, J., Lewis, M., 2018. Cyberinfrastructure supporting watershed health monitoring and management. WIT Trans. Ecol. Environ. 228, 245–256. https://doi.org/10.2495/WP180231.

U.S. Congress, 2007. Energy Independence and Security Act - SMART GRID. United States of America. https://www.govinfo.gov/content/pkg/STATUTE-121/pdf/STATUTE-121-Pg1492.pdf. (Accessed 3 March 2021).

Wegrzyn, J.L., Falk, T., Grau, E., Buehler, S., Ramnath, R., Herndon, N., 2020. Cyberinfrastructure and resources to enable an integrative approach to studying forest trees. Evol. Appl. https://doi.org/10.1111/eva.12860.

Willis, R.M., Stewart, R.A., Williams, P.R., Hacker, C.H., Emmonds, S.C., Capati, G., 2011. Residential potable and recycled water end uses in a dual reticulated supply system. Desalination 272, 201–211. https://doi.org/10.1016/j.desal.2011.01.022.

Willis, R.M., Stewart, R.A., Giurco, D.P., Talebpour, M.R., Mousavinejad, A., 2013. End use water consumption in households: impact of socio-demographic factors and efficient devices. J. Clean. Prod. 60, 107–115. https://doi.org/10.1016/j.jclepro.2011.08.006.

Wissner, M., 2011. The Smart Grid - a saucerful of secrets? Appl. Energy 88, 2509–2518. https://doi.org/10.1016/j.apenergy.2011.01.042.

Yan, Y., Qian, Y., Sharif, H., Tipper, D., 2013. A survey on smart grid communication infrastructures: motivations, requirements and challenges. IEEE Commun. Surv. Tutorials. https://doi.org/10.1109/SURV.2012.021312.00034.

Ye, Y., Liang, L., Zhao, H., Jiang, Y., 2016. The system Architecture of smart water grid for water security. In: Procedia Engineering. Elsevier Ltd, pp. 361–368. https://doi.org/10.1016/j.proeng.2016.07.492.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M., 2014. Internet of things for smart cities. IEEE Internet Things J 1, 22–32. https://doi.org/10.1109/JIOT.2014.2306328.

InfluxData, 2021. InfluxDB. https://www.influxdata.com/products/influxdb/ (accessed 5 May 2021).

EPA, 2020. United States Environmental Protection Agency. The water Sense label. https://www.epa.gov/watersense (accessed 1 May 2021).

Forcier, Jeff, 2021. Welcome to Paramiko. http://www.paramiko.org/ (accesed 5 May 2021).

Gunicorn, 2021. Gunicorn. https://gunicorn.org/ (accessed 3 May 2021).

Hellkamp, Marcel, 2021. Bottle: Python web framework. https://bottlepy.org/, 2021 (accessed 1 May 2021).

InfluxDB, 2020. API documentation. https://influxdb-python.readthedocs.io/en/latest/index.html (accessed 1 February 2021).

NGINX, 2021. About NGINX. https://nginx.org/en/ (accessed 5 May 2021).

NSF, 2007. National Science Foundation. Cyberinfrastructure Vision for 21st Century Discovery. https://www.nsf.gov/pubs/2007/nsf0728/index.jsp (accessed 4 May 2021).

Python Software Foundation, 2021a. Asyncio — asynchronous I/O. https://docs.python.org/3/library/asyncio.html (accessed 5 April 2021).

Python Software Foundation, 2021b. Timeit — measure execution time of small code snippets. https://docs.python.org/3/library/timeit.html (accessed 5 April 2021).

Slack Technologies, 2021. Slack. https://slack.com (accessed 5 May 2021).

Attallah, N., Bastidas Pacheco, C.J., 2021. Supporting data and tools for "Tools for Evaluating, Developing, and Testing Water End Use Disaggregation Algorithms", HydroShare, http://www.hydroshare.org/resource/3143b3b1bdff48e0aaebcb4aedf02feb.

Attallah, N.A., Horsburgh, J.S., Bastidas Pacheco, C.J., 2021. Tools for Evaluating, Developing, and Testing Water End Use Disaggregation Algorithms. Manuscript submitted for publication.

Amazon, 2021. Amazon EC2 Secure and resizable compute capacity to support virtually any workload. https://aws.amazon.com/ec2/?nc2=h_ql_prod_fs_ec2&ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc (accessed 2 May 2021).