Environmental Modelling & Software 84 (2016) 505–517

Contents lists available at ScienceDirect

Environmental Modelling & Software

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

Real-time environmental sensor data: An application to water quality using web services

Brandon P. Wong*, Branko Kerkez

Civil and Environmental Engineering, University of Michigan, United States

ARTICLE INFO

Article history:
Received 15 February 2016
Received in revised form
5 June 2016
Accepted 30 July 2016
Available online 25 August 2016

Keywords:
Real-time data
Web services
Sensor networks
Adaptive sampling
Internet of things (IoT)

ABSTRACT

While real-time sensor feeds have the potential to transform both environmental science and decision-making, such data *are rarely* part of real-time workflows, analyses and modeling tool chains. Despite benefits ranging from detecting malfunctioning sensors to adaptive sampling, the limited number and complexity of existing real-time platforms across environmental domains pose a barrier to the adoption of real-time data. We present an architecture built upon 1) the increasing availability of new technologies to expose environmental sensors as web services, and 2) the merging of these services under recent innovations on the *Internet of Things (IoT)*. By leveraging recent developments in the IoT arena, the environmental sciences stand to make significant gains in the use of real-time data. We describe a use case in the hydrologic sciences, where an adaptive sampling algorithm is successfully deployed to optimize the use of a constrained sensor network resource.

Software and data availability

The use case was implemented using the Xively Internet of Things platform and a Flask web-server (written in Python 2.7) running on an Elastic Beanstalk t2.micro instance provided by Amazon Web Services. All experimental data from this study are hosted on a secure Xively feed and available upon request. The source code and implementation parameters are available on a public repository: https://github.com/kLabUM/IoT. As of 2015, all of these tools are available at no cost for a project of the scale discussed in this paper. Web connectivity is required of all hardware and software. For additional information, contact bpwong@umich.edu.

1. Introduction

Recent advances in sensing, computation and communications have enabled a massive suite of low-cost, low-power connected devices. This is particularly true for modern wireless sensor networks (Christodoulou et al., 2010, Jin et al., 2010), which now support the reliable, low-cost, near-instant transmission of measurements from field-deployed sensors. For enterprise-scale web applications, *RESTful* web services have also witnessed a surge in popularity (Kübert et al., 2011) while advances in the hardware realm have been accompanied by new architectures and protocols that exploit the bidirectional communication and Internet-connectivity of embedded devices. As such, libraries and *application programming interfaces* (*APIs*) enable users to quickly deploy RESTful web services on almost any software or hardware platform. This is significant, as most new devices from popular hardware and datalogger manufacturers increasingly support web communication via Wi-Fi, Ethernet, cellular, and other physical channels. Through these efforts, the *Internet of Things* (*IoT*) has recently been proposed as the backbone that will route and manage the vast quantities of data collected by these sensor networks (Atzori et al., 2010, Christin et al., 2009). In many environmental applications, however, these technological advances merely serve as a convenience to reduce field visits, provide data visualization, and simplify data collection by streaming sensor data to central repositories for subsequent analysis. Real-time data are rarely used as part of automated workflows, analyses and modeling tool chains.

In the computer science communities, in particular the area of embedded systems, the definition of real-time carries with it explicit performance guarantees, such as deadlines and timing constraints (Lee and Seshia, 2011). Such a strict definition, however, may be too technical to appeal to the broader environmental communities. While an actual definition may be out of reach

* Corresponding author.
E-mail addresses: bpwong@umich.edu (B.P. Wong), bkerkez@umich.edu (B. Kerkez).

http://dx.doi.org/10.1016/j.envsoft.2016.07.020
1364-8152/

considering the diversity of applications in the environmental sciences, an underlying principle persists: *real-time data* are data available for use as soon as they are collected to make a decision within a constrained time window, independent of sampling frequencies. This principle does not seek to distinguish between notions of real-time or near real-time, as is often the case in many studies (Christodoulou et al., 2010, Zhang et al., 2012, Hefeeda and Bagheri, 2009).

While not ubiquitous across the broader environmental domains, the use of real-time data for decision-making is not novel in some fields. For example, in the atmospheric sciences, satellite data is assimilated daily into advanced models which are used by various scientists and decision makers (Lakshmanan et al., 2007), while across meteorology, real-time radar feeds and terrestrial sensors inform stake holders across agriculture, transportation and disaster response (Zhang et al., 2011). However, despite the availability of low-cost, low-power hardware and data platforms, the benefits of these real-time resources have yet to be leveraged broadly across the remaining environmental sciences.

Scientific data analyses are more commonly conducted after an experiment has been completed, which for many studies could last months or years. A reason for the lack of real-time data adoption relates to the fact that most scientists may simply be satisfied with *continuous,* rather than *real-time,* data. The use of sensors across the environmental sciences thus appears to be retroactive, rather than adaptive. This would suggest that the major benefit of real-time data relates to decision-making, where assimilation of sensor information into models will enable rapid response to extreme events such as floods, wildfires and earthquakes.

While the ability to respond to natural disasters is invaluable, significant benefits of real-time data arise to environmental researchers as well, especially in the detection of faulty sensors and data acquisition systems. This is particularly true for experiments in harsh or remote environments where site visits may be infrequent and equipment outages can result in significant lapses in continuous data streams. For such experiments, real-time alerts will go a long way toward improving the quality of continuous data sets.

Perhaps the most compelling benefit of real-time data relates to the ability to usher in a new generation of adaptive scientific experiments. By adding real-time functionality to non-real-time studies, scientists will be able to perform innovative studies that respond to dynamic experimental conditions. As illustrated in this paper, this includes the ability to guide an experiment in real-time to adaptively sample signals or locations of interest during the most relevant intervals, which will significantly improve the use of constrained experimental resources and thus the quality of scientific experiments.

Across many domains, the notion of real-time is often complicated by operational requirements, which drive a lack of consensus around the definition of the actual term. Regardless of application, however, the utility of real-time data is governed by constrained time windows during which decisions have to be made. These time windows can range from days (e.g. climate modeling and data assimilation (Zhang et al., 2012)) to minutes or seconds (e.g. flood or wildfire forecasting (Hefeeda and Bagheri, 2009)). Outside of these time windows the data can be classified as *historical*, thereby limiting their utility for immediate decision-making. A wealth of tools have been developed to store, process and visualize historical sensor data (Argent et al., 2009, Castronova et al., 2013, Horsburgh et al., 2009, Gronewold et al., 2013, Goodall et al., 2008), but these frameworks have yet to be extended to provide real-time functionality.

In this paper, we present a summary of existing efforts to enable the use of real-time data across a broad set of domains, showing that the complexity and limited number of these existing real-time data platforms limits their adoption by the environmental sensing community. The majority of these platforms requires persistent expert support and cannot always be easily ported to existing field equipment and sensor networks, even by experienced researchers who readily operate continuous sensing campaigns. With real-time data systems also come different operational requirements, including the ability to continuously update and operate on new data, communicate with remote sites, monitor the operational status of devices, and manage user privileges throughout the system. We discuss these barriers to adoption and present a solution built upon two cornerstones: 1) the shift of environmental sensors and actuators[1] to a more generic web service model, and 2) the merging of these services under the recent architectural innovations on the Internet of Things.

To that end, we introduce a web service-centric approach to enable a flexible, reliable and powerful means by which to store, transmit and analyze real-time data. By focusing on recent advances in the IoT arena, we will show that the environmental sciences stand to make rapid gains in the use of real-time data while simultaneously improving flexibility related to implementation and maintenance. Rather than building a new platform, we will show how existing IoT platforms already provide a backbone to integrate real-time data from web-enabled environmental sensors and devices to meet requirements of interoperability, support, reliability, and security. By leveraging the services provided by these platforms, these web-enabled sensors and devices can also seamlessly interact with a multitude of web resources, including powerful cloud computing services and web-based models. A use case from the hydrologic sciences illustrates how a script can be deployed as a web service within this framework to enable low-power sensor networks to adaptively sample dynamic water quality parameters during storm events. While not a one-size-fits-all solution, our approach is expected to conform well to the requirements of most environmental applications, particularly for those where large sensor networks are deployed.

2. Existing platforms and real-time data efforts

Data systems employed across the environmental domains may be broadly classified into two groups: 1) systems used for the storage, retrieval and visualization of data, and 2) data systems designed explicitly for real-time operations. While the former do not explicitly treat real-time data, they do provide powerful mechanisms by which to standardize data retrieval and storage (Christodoulou et al., 2010, Horsburgh et al., 2009, Argent et al., 2009, Gronewold et al., 2013, Goodall et al., 2008). Some of these platforms conform to a set of community standards (e.g. *WaterML*, *DelftFEWS,* etc., see (Taylor et al., 2010)) that reduce operational overhead and enable the seamless use of standard-compliant tools for scalable storage, management and visualization of data. However, interactions with data are often carried out through direct user queries to the system, with no or limited mechanisms in place to automatically notify users of new readings or events as they occur. Furthermore, such architectures are not typically designed to enable alerts or the discovery and access to field-deployed sensors or actuators, thus limiting their use in control-centric and decision-making applications.

A number of these systems are also designed for domain-specific applications, thus limiting their use across a broader set of domains. In most cases, end-users are required to implement and

[1] Sensors generate an electrical signal in response to stimuli from the environment. Actuators respond to an electrical signal and act upon their environment (e.g. a gate that opens or closes).

host these real-time systems, which introduces deployment and maintenance complexities in addition to those inherent to deploying and maintaining field-deployed sensors and actuators. This includes, but is not limited, to setting up dedicated servers, installing necessary tool chains, adopting specific programming languages, and guaranteeing system up-time.

A number of platforms have been designed to explicitly treat real-time data. *DataTurbine* (Tilak et al., 2007) is an open source, *Java*-based platform for managing and transporting data from sensor networks and video feeds. Designed for environmental applications, DataTurbine implements a *ring buffer*, much like a size-limited first in, first out queue, to temporarily store the most recent sensor data and reliably route data streams to visualization and storage modules. Given the emphasis on streaming, high data rates can be supported (over 16 Msamples/sec). DataTurbine was developed as a generic streaming data middleware for real-time data acquisition systems, independent from a specific application niche. Some uses of DataTurbine include oceanic studies, climate change research, earthquake engineering, and lake monitoring (Fountain et al., 2009). However, users are required to run individual DataTurbine services on all servers and field-deployed devices, which limits the number of supported data loggers and controllers to those compatible with Java. In addition to the complexities of setting up a monitoring system, the explicit emphasis on a particular programming language makes it difficult for users unfamiliar with Java to deploy the platform.

An extension to DataTurbine is *Wavellite*, an open source Java suite that supports real-time situational knowledge of heterogeneous datasets and observations (Stocker et al., 2014). The software interprets data as it streams in by using a suite of machine learning algorithms. One example study using Wavellite applied artificial neural networks to process aerosol and weather data to identify and characterize the formation of particles that could act as cloud condensation nuclei (Stocker et al., 2014). While powerful, the system is designed to support specialized operations and exhibits limited storage support. Moreover, since Wavellite is built upon DataTurbine, its deployment requires implementation expertise of both platforms.

IBM's *InfoSphere Streams* is another real-time data analysis tool chain that enables the rapid analysis of real-time data feeds before data is saved into databases (Rea, 2013). The *Streams* tool chain has been applied across a broad set of industries, including financial services and transportation, to continuously use machine learning to extract information for decision-making. However, given its emphasis on machine learning and its current price point (thousands to tens of thousands of dollars), this tool chain appears to be primarily geared toward larger groups and companies and may thus be out of reach of smaller scientific research groups.

One of the most established real-time data systems is UNIDATA's *Local Data Manager* (*LDM*) (Davis and Rew, 1994). LDM is a package of UNIX-based modules designed for event-driven applications, particularly those relating to atmospheric science data. Users must however host their own servers and setup any relevant UNIX-based tool chains before installing and maintaining LDM. Additionally, porting the system to domains outside of atmospheric sciences introduces further complexity, which may deter use by a broader community.

Another popular real-time data system is *Antelope*[2] which supports a number of language interfaces, including *C, Fortran. Perl, Python* and *MATLAB*. Originally designed for storing and streaming seismic data, the system is also built upon a ring buffer and accompanied by a suite of signal processing tools to analyze waveforms and detect events. The suite of tools resembles a real-time signal-processing platform targeted towards seismic applications. While some broader communities may be too unfamiliar with signal processing and its nuances to adopt this system, the suite of seismological tools may also be too specialized for those seeking more general real-time data functionalities.

2.1. Overcoming barriers to real-time data adoption

To address an increasing interest in real-time data applications across the environmental sciences, a working group was formed under the broader umbrella of the U.S. National Science Foundation's *EarthCube*[3] initiative. The EarthCube initiative was launched in 2011 to discover *transformative concepts and approaches to create integrated data management infrastructures across the geosciences*. A real-time data workshop was organized in 2013 to determine the needs experienced by a broad spectrum of scientific groups (Daniels et al., 2013). Discussions and surveys nearly unanimously reported that while there was significant interest in real-time data, users were unsatisfied with the limited set of existing tools, citing their complexity and ease of use (or lack thereof) as a major barrier to adoption.[4]

When deploying environmental sensors, the resources required to program firmware and maintain hardware already pose significant demands on research groups. Substantial additional overhead is incurred if real-time functionality is desired. Existing real-time data platforms impose significant requirements in the form of system architectures, operating systems, programming languages, and even sensing platforms, which makes their deployment labor intensive, even for users who already maintain sensor networks for continuous data. For example, *DataTurbine* requires users to manually start and maintain Java instances of the software both as servers and on field hardware, while packages such as LDM require users to compile binaries from source code to match their specific UNIX-based environments. To that end, physical protocol compliance has been proposed as a means of tying into these systems and to reduce implementation overhead. In one study, the authors suggest that to enable the use of their platform (Díaz et al., 2013) sensors should conform to a standard hardware interface, in particular an Ethernet port. Field experience and the sheer variety of sensor platforms significantly undermine the real-world applicability of such requirements and further illustrate the disconnect between those deploying sensors and those designing data platforms. These and other usability constraints raise the barrier to real-time data adoption by enforcing non-trivial design and implementation challenges on users.

The adoption of real-time data across the environmental sciences hinges upon the resolution of a number of broader challenges:

- *Interoperability*: Existing real-time data platforms impose non-trivial requirements on users. Real-time data systems should be designed to permit users to retain their existing tool chains and hardware platforms inasmuch as possible without imposing major additional requirements to maintain servers, compile libraries, or support specific hardware interfaces.
- *Support for real-world, low-power devices*: Sensor selection should be governed by the application and should not be limited by the capabilities of the underlying data infrastructure. Due to

[2] http://www.brtt.com.

[3] http://www.nsf.gov/geo/earthcube.

[4] Efforts are now underway to further study these findings and investigate real-time architectures under the EarthCube Cloud Hosted Real-time Data Services (CHORDS) project. http://earthcube.org/group/chords.

power constraints, real-world, battery-powered sensing platforms must duty cycle their web connectivity in exchange for battery life (Christodoulou et al., 2010, Jin et al., 2010, Hefeeda and Bagheri, 2009, Hartung et al., 2006). In such instances, it is unreasonable to expect persistent bi-directional communications between sensors and data services. To further limit the power draw from network communications, data must be transmitted as size-efficiently as possible. A real-time data framework must develop means by which to interact with such devices, balancing intermittent transmissions and wide-ranging bandwidths.

- *Reliability and usability*: Many presently existing real-time platforms and open source projects lack the infrastructure and reliability to support a large user base, including novice and technically savvy users alike. While these systems are being improved, we contend that reliable and feature-rich commercial platforms should be considered inasmuch as possible to allow those deploying sensors to leverage enterprise-scale reliability on their projects. This will permit users to focus on applying their domain knowledge towards developing applications and experiments, rather than data system design and administration.
- *Security*: Given the nature of real-time data, proper security measures must be taken to ensure that the streams of data from sensors and control of devices are protected via modern encryption and authentication techniques. Existing real-time platforms, such as DataTurbine and UNIDATA's LDM, recommend limiting web connections to specific trusted IP addresses and encrypting packets using a digital signature. However, keys to read, write, create, and delete web resources are not implemented in these platforms, and neither is *HTTP Secure* (*HTTPS*), a common protocol used for information-sensitive web applications such as email and online banking.

Rather than enforcing highly specific hardware and software requirements, web services are emerging as a powerful and versatile interfacing mechanism to connect disparate sensors, data sources and models (Castronova et al., 2013). The availability of reliable, low-cost wired and wireless technologies has increased significantly over the past decade, permitting most field devices to be connected to the Internet (Fleisch, 2010, Gubbi et al., 2013). New *Internet Protocol* (*IP*) addressing schemes are currently shifting from the traditional *IPv4* addressing to *IPv6* addressing, which will ensure any device or service can be uniquely identified and accessed (Mainetti et al., 2011, Teklemariam et al., 2013). This is particularly useful for wireless sensor networks, such as those based on Ethernet, mesh, Wi-Fi or cellular protocols, where such networking platforms are shifting toward IPv6-based connectivity to accommodate larger network deployments (Mainetti et al., 2011). As the scale of sensor networks continues to grow, requiring IP connectivity for real-time data applications is not just realistic but inevitable. While some of the existing or legacy commercial hardware platforms may not directly support Internet connectivity (TCP/IP), they can either be equipped with communications modules or will soon be replaced by modern platforms that support remote data access. Those platforms that ultimately do not adopt Internet connectivity may still perform well for applications requiring continuous, but not real-time data. As such, in the building of real-time data applications, a paradigm shift toward web services becomes a reasonable requirement for users wishing to take advantage of modern data platforms and cloud-hosted services.

2.2. Leveraging IoT platforms

Platforms from the *Internet of Things (IoT)* community have recently arisen in response to the rapidly increasing number of wireless devices, which are becoming ubiquitous in the measurement and automation of residential and industrial processes. IoT platforms have been designed to support sensor discovery, real-time data routing and remote device control. Rather than writing firmware code, users can use popular high-level languages, such as Python or MATLAB, to analyze data and actuate a remote device. Much of this logic can even be set up using configurable services with little to no additional programming required (e.g. trigger an alert if a sensor value exceeds a threshold). Users can directly subscribe to real-time data feeds or notifications, which are used to monitor the state of field devices and provide a means by which to transmit alerts. Despite their significant adoption across the sensor and automation communities (Atzori et al., 2010, Mainetti et al., 2011), IoT platforms have yet to penetrate a user base within the environmental domains. A variety of both open source and commercial IoT platforms exist (Table 1). When comparing the platforms of the IoT ecosystem, their core functionalities can be broken down into administrative features, transfer protocols and data management.

Each IoT platform supports data exchange via web services and a set of standard protocols, the majority of which relies on RESTful data transfer. Some platforms also support more modern, low-power protocols alternative to HTTP, such as MQTT and CoAP (Thubert et al., 2013, Watteyne et al., 2012), which provide additional sensor-centric functionalities and are designed to improve device battery lifetimes by optimizing the size of packet headers. While raw or comma-separated data formats can be exchanged between web services and devices, the majority of platforms support APIs and formats built around popular framing protocols such as *JavaScript Object Notation* (*JSON*) or *Extenstible Markup Language* (*XML*) (Lamela Seijas et al., 2013, Wang, 2011), which permit for rapid integration with various programming languages and tools. The content and syntax of the payload may vary based on the IoT platform, but many platforms, such as *ThingSpeak* and *Xively,* support a broad selection of operating systems and languages. Many platforms also provide APIs that encode data into the required payload syntax, thus reducing the amount of software development required of the user. As such, the payload is similarly encoded amongst these platforms and interoperability between IoT platforms can be achieved through relatively straightforward content mappings. These APIs are available even for low-level languages, such as C, which permits them to be ported to low-power microcontrollers and data-loggers. For those deploying sensor hardware, the steps to connect to an IoT platform involve a relatively small addition to already existing code. At its simplest implementation, this involves opening a TCP/IP port and transmitting a relatively intuitive JSON-encoded or *Comma Separated Values* (*CSV*)-encoded string. While the CSV encoding is relatively self-explanatory, the JSON string can often be generated on the IoT platform's website and pasted into the low-level code. For those wishing to support remote actuation of a field-deployed device, a callback function can also be implemented to parse and respond to messages received by the IoT platform.

All of the major IoT platforms support administrative control via public and private key access. Field-deployed devices and other web services (e.g. models, visualization services, etc.) that interact with these IoT platforms must be authenticated via these keys to access the services provided by an IoT platform. Key permissions can be generated and configured on the platform without needing to change device firmware. To our knowledge, such authentication systems are rarely implemented on existing real-time environmental data systems (LDM, Antelope, etc.), where access is instead controlled by less complex measures such as limiting connections to specific IP addresses or digitally signing packets with a shared

Table 1
Comparison of IoT platforms.

Platform	Account				Reading & writing				Data & tools						
	Free plan	Data logging	Public/Private Access	Multiple keys	Rest protocol	WebSockets	CoAP|MQTT protocol	API library	JSON	XML	CSV	Limited data retention	Metadata	Notifications, warnings	Mobile sensors	Visualizations	Widgets
Commercial																	
Amazon IoT		✓	✓	✓	✓		✓	✓	✓				✓	✓	✓	✓	
Beebotte	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓		✓	✓	✓
Exosite	✓	✓	✓		✓	✓	✓	✓	✓		✓		✓	✓		✓	✓
GroveStreams	✓	✓	✓	✓	✓				✓			✓	✓	✓	✓	✓	✓
Open.Sen.Se	✓	✓	✓		✓				✓							✓	✓
SensorCloud	✓	✓	✓		✓											✓	
Ubidots	✓	✓		✓	✓			✓	✓			✓		✓	✓	✓	✓
Xively	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	
Open Source																	
KSDuino	✓	✓	✓					✓				✓				✓	
Nimbits	✓	✓	✓	✓	✓				✓	✓				✓	✓	✓	
phant.io	✓	✓	✓					✓	✓	✓	✓					✓	
ThingSpeak	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓

password. Presently, however, few IoT platforms support multiple keys and permissions, which is particularly useful where one would like to limit control of specific resources for certain users. For example, depending on their role, a group of users may be allowed only to subscribe to and view real-time data from devices and may not be permitted to control sensors and actuators. With tiered authentication systems, one can dictate which web resources are available publicly or privately.

To varying degrees, IoT platforms support the long-term storage of data and metadata, thus serving both as real-time data and historical data platforms. Some platforms only focus on limited data handling and storage, primarily serving to buffer data for decision-making, actuation, and alerts. While storage may not be an explicit requirement for most real-time applications, long-term storage can be achieved by routing real-time feeds from an IoT platform into community-maintained and domain-specific databases and storage facilities (e.g. The CUAHSI Water Data Center[5]). Commercial IoT platforms operate on service-based models (Perera et al., 2014), where providers launch, scale and maintain the platform, allowing users to focus on the actual applications. Most platforms support a free plan, which permits small projects (1–50 sensors) to leverage the services without a fee. Typically, such plans limit the number of sensors and the frequency at which data can be transmitted. In the case where applications require additional sensors or data transmission at higher frequencies, commercial platforms offer very affordable pricing plans, where fees are charged on a per-use basis. For example, at the time this paper was written, Amazon's IoT Service[6] supported the transmission of one million messages (512 bytes per message) at a cost of $5 USD, which falls well below the cost that a small scientific group would have to expend on developing, hosting and maintaining a comparable data service.

While the open source platforms are powerful, the benefits of the commercial, enterprise-grade data services cannot be discounted. Most open source platforms must be installed and maintained by the user whereas commercial platforms are oftentimes cloud hosted and can be readily accessed within a few mouse clicks. As with any open source platform, significant expertise and resources are required to ensure robust functionalities that include, but are not limited to, routing feeds, issuing alerts, storing data, adhering to protocols and standards, and coordinating security and user privileges. Although open source versions of these platforms offer such features, it may be unrealistic to assume that all scientific users or decision makers have the expertise or resources to deploy and maintain these complex systems.

We contend that the burden of hosting and maintaining complex, real-time web service architectures should not be offloaded to the user, but, where feasible, should be deferred to reliable hosting providers. Most recently, this has been the case with the paradigm shift toward cloud computing, where commercial computing services are replacing local hosting and computations in various applications (Buyya et al., 2009), including many across the environmental domains. A similar paradigm shift in real-time environmental data services is needed. Commercial sensor data services are hosted and maintained by experts, permitting users to launch an instance at their convenience and focus on their applications rather than system administration. It should be noted that in this paper our goal is not to advocate any IoT platform in particular, but rather to promote their broader adoption.

[5] https://www.cuahsi.org/wdc.
[6] https://aws.amazon.com/iot.

3. Use case

Motivated by the need to improve our understanding of water quality in streams and rivers, our specific objective was to better understand the dynamics of nutrient loadings to urban and agricultural sources. Large nutrient loads are considered the primary cause of harmful algal blooms and *dead zones* witnessed most recently in the Great Lakes (Zhou et al., 2013) and Gulf of Mexico (Scavia et al., 2013). Despite the insight provided by dynamic models, our understanding of these nutrient loadings and their origins is still limited significantly by a lack of real-world data. As such, more measurements are needed to resolve the spatiotemporal dynamics of nutrient fluxes into aquatic ecosystems.

3.1. Challenges in the measurement of water quality

Compared to water flow, water quality parameters are still relatively difficult to measure (Gartia et al., 2012, Diamond et al., 2008). This is particularly true of some water quality constituents (e.g. nutrients, metals and bacteria), where in-situ sensors either do not exist or are too cost-prohibitive to be deployed at meaningful resolutions. In such instances, samples collected by automated samplers are an effective alternative to in-situ measurements (Gall et al., 2010). When triggered manually or through a timer mechanism, these samplers actuate a motor to pump a water sample directly from a stream into one of a number of available bottles that are later taken to a laboratory for analysis. Since each sample is time stamped, the laboratory results can then be used to correlate water quality parameters with known physical characteristics or measurements taken by accompanying in-situ sensors.

The number of sample bottles is limited in an automated sampler and its power consumption is very large due to its motorized mechanical components. Given these resource constraints, it becomes necessary to optimize sampling times and frequencies to capture events of interest. Sampling too fast can cause the number of available samples to be depleted before an event is captured. Furthermore, “wasted” samples occur when a storm does not happen as predicted, while “missed” samples occur if the duration of a storm is longer than anticipated. Most often, events of interest include storms, which can cause significant quantities of surface water to flow into nearby streams and rivers, thereby discharging nutrients that have accumulated on land. Nutrient loadings during the beginning of a storm, or the *first flush*, are often considered an indicator of the effect of nearby land use practices on water quality. When measuring the flow of water in a stream or river, the first flush is often evident as a spike in the hydrograph signal (Field, 2002, Sansalone and Cristina, 2004). To capture these events, automated samplers have mainly been used on an as-needed basis, where units are placed on a site in anticipation of storm events and programmed to take readings at regular intervals. The feasibility of using automated samplers thus becomes burdensome in terms of cost, battery consumption and manual labor.

3.2. Adaptive sampling of hydrologic signals

The drawbacks of automated samplers can be minimized through *adaptive sampling* (Årzén, 1999, Åström and Bernhardsson, 1999) where, rather than evenly sampling a signal, a controller or algorithm persistently updates a model of a phenomenon using real-time data and then samples only during events of interest. In our approach, the algorithm was a configurable web application that queried a public weather forecast to determine the probability of impending precipitation. The algorithm monitored the hydrograph signal in real-time to determine sudden state changes, such as a rapid rise in the hydrograph. A rule-based optimization procedure was then used to determine when to take the next sample. A theoretical description and evaluation of the algorithm are given in (Wong and Kerkez, 2014). The algorithm encoded the objective of minimizing the samples required to characterize the first flush of the hydrologic catchment by triggering a sample of water quality to be taken right before a storm (based on weather predictions), a number of samples during the rising limb and inflection points of the hydrograph, and a smaller number of samples following the hydrograph recession. As such, the algorithm guided the automated sampler to respond to both weather forecasts and changes in measured flow values. The implementation assumed a sensor node equipped with an Internet connection, which received sampling commands from a web application.

3.3. Hardware

A water quality sensor node was developed using the NeoMote wireless sensing platform (Zhao and Kerkez, 2014). This FPGA-based platform (Cypress PSoC5LP) is programmed in C and features an ultra-low power ARM-Cortex M3 microprocessor, 20-bit low-noise analog to digital converter, configurable on-board storage via an SD card, and variable, low-noise power supplies for sensors. Given the urban study site (Ann Arbor, MI Lat. 42.264855, Lon. -83.688347), cellular coverage was readily available, which thus enabled the use of a low-cost IP-enabled cellular module (Telit CC864-DUAL) for Internet connectivity. The NeoMote platform consumed an average current of 30 micro-amps. The cellular module consumed significantly more, requiring nearly 200 milli-amps during transmission events. To conserve power, the cellular module was duty cycled, where power was cut entirely to the module when it was not being actively used to transmit sensor readings. With the addition of a low quiescent current (1 micro-amp) lithium-ion solar charge controller and a solar panel, the sensor node was designed to operate for years without the need for battery replacements or line power.

The sensor node (Fig. 1) was interfaced with an automated sampler (ISCO 3700) using a transistor-transistor logic interface (TTL). The automated sampler had a 24-bottle capacity, a standby current of 10 milli-amps and an energy consumption of 2 Amps at 12 VDC during sampling. Even with duty cycling, the use of the automated sampler provided the largest constraint on battery resources, further emphasizing the need to limit sampling to only events of interest. A suite of hydrologic sensors was also attached to the data logger, including an ultrasonic depth sensor (MaxBotix MB7384) and a pressure transducer (Solinst 3001 Levelogger® Edge) for stage measurements. The data logger interfaced with the ultrasonic sensor in TTL serial mode, the pressure transducer via SDI-12, and the conductivity sensor via analog output. The data from the ultrasonic sensor was used to derive an estimate of the hydrograph stage, which was then used by the adaptive sampling algorithm to determine when to trigger the next water quality sample. Data from the pressure transducer was initially used to verify the hydrograph estimates derived by the cheaper ultrasonic sensor as a means of vetting its use for future studies. While not used in this study, an analog conductivity sensor (Campbell Scientific CS547A) was also connected to the platform to assess benefits of triggering water quality samples based on conductivity thresholds.

3.4. Software architecture

To enable rapid deployment and reliable and secure operations,

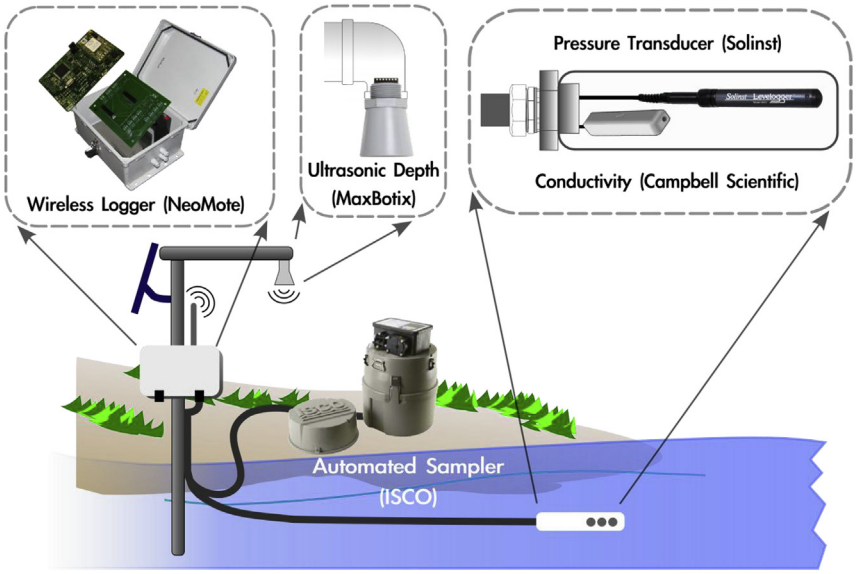

Fig. 1. IP-enabled hydrologic sensor node which consists of a wireless logger, automated sampler, and a suite of sensors to measure depth and conductivity.

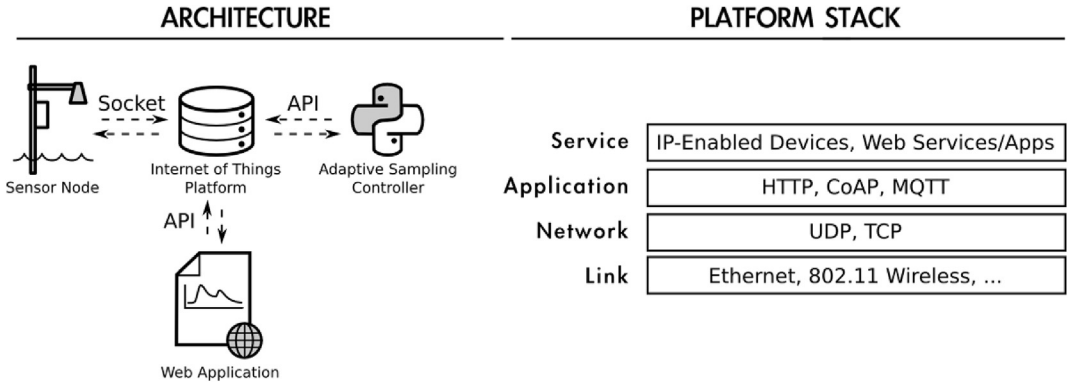

Fig. 2. Web service based real-time data architecture and platform stack.

we designed and implemented a real-time data architecture around an Internet of Things platform. At the time that this paper was written there were a number of IoT platforms known to the authors (Table 1). During the platform selection process, the features that were considered included authentication and security mechanisms, data storage, throughput limitations, device management interfaces, and available libraries or APIs. The choice to build our architecture around the platform offered by *Xively*[7] was driven primarily by the availability of easy-to-use libraries for a diversity of programming languages, as well as the ability to support multiple authentication keys and user privileges. Given the emphasis on web services, the same architecture and features could have also been implemented using other IoT platforms with some minor payload syntax modifications.

At the lowest level, the platform assumes that all devices and applications are IP-enabled, whether through wired (Ethernet) or wireless (Wi-Fi, cellular, etc.) interfaces. Data transfer can take place either via TCP or UDP protocols, which was chosen based on application-specific performance requirements. While data can be exchanged in a raw format through these low-level socket connections, a number of application-layer protocols, including HTTP, HTTPS and other low-power protocols, such as CoAP and MQTT, are supported to permit the system to interface with popular tools and programming languages. This significantly reduced implementation overhead while simultaneously providing a large support infrastructure in the form of a broad user community, and thus enabled us to focus on the implementation of adaptive sampling rather than the details of low-level data transfer.

To illustrate the flexibility afforded by web services, our architecture (Fig. 2) implemented three separate web services on three separate devices, each of which was programmed in a different language. Each service was implemented using Xively's RESTful API and was written in a programming language most suitable to its purpose and ease of implementation. The first web service was written in C and executed on the sensor node to transmit data and receive sampling commands via a cellular data connection. The second web service was the adaptive-sampling algorithm (controller), whose logic was controlled by a Python script, which could be executed on a local machine or web server to send sampling commands via a RESTful interface in response to the real-time sensor measurements. A third, client-side service was implemented in JavaScript and used a RESTful interface to interface with the IoT platform. It provided access to historical data and allowed a user to issue commands to the sensor node via a website. Each service was individually authenticated and interacted through the

[7] https://xively.com.

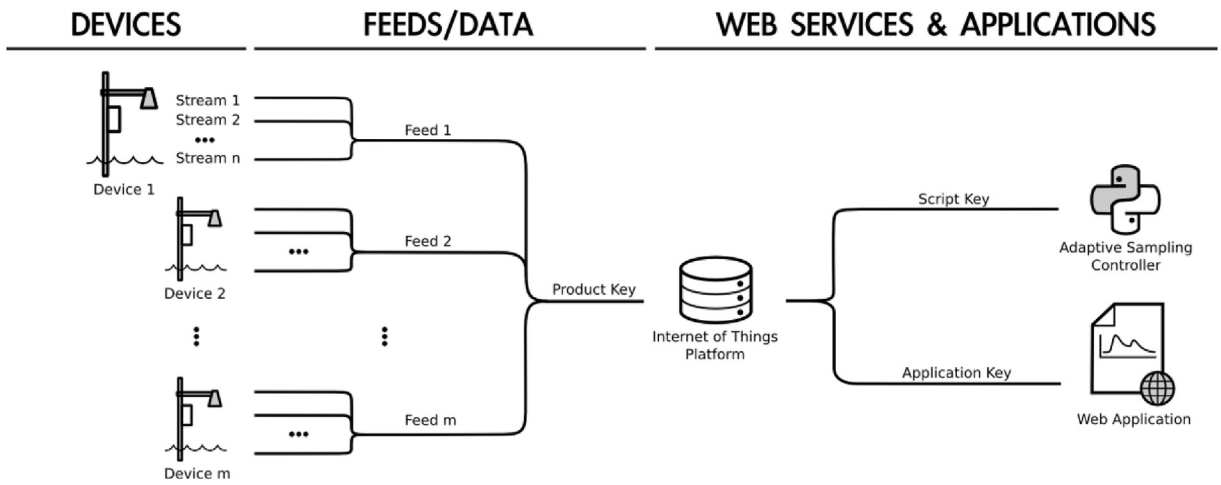

Fig. 3. Data transport, discovery and storage. All data and services interact through the IoT Platform. Each service is assigned a separate authentication key based on its role and privileges.

IoT platform, which was hosted by a commercial service. The code for this entire reference implementation is available on our public repository.[8]

The IoT platform not only interfaced each web service, but also enabled a suite of data discovery and management mechanisms. This allowed any number of authenticated web services to query the system for historical data and metadata, register for alerts, and to obtain direct links to real-time feeds. Organization, exchange and storage of data interacted through a hierarchical structure that is applicable to most types of data streams from any number of devices (Fig. 3). Each field device (sensor node) was assigned a unique data *feed*, which was further subdivided into individual *streams* (individual sensors and actuators). Users accessed individual feeds or groups of feeds based on their privileges and authentication keys. For example, some users could only view sensor data and metadata, while others can also control sensors or actuators.

Data and commands were embedded in this structure and exchanged between the individual web service applications using CSV (sensor node) and JSON encodings (sampling controller and web browser application). While XML encodings were also an option, JSON was chosen due to its smaller packet size to reduce cellular transmissions and increase battery life. Given the popularity of these encoding formats across Internet services (Lamela Seijas et al., 2013, Wang, 2011), powerful libraries now exist for almost any programming language to simplify the conversion of sensor readings to formatted data packets, further reducing programming-related overhead. While our application did not explicitly demand the use of domain-compliant syntaxes (e.g. *WaterML, SensorML, DelftFEWS*, etc.), this feature can be added as a relatively lightweight web service or library that maps between the desired syntax formats to drive the adoption of the proposed architecture by domain-specific communities.

4. System implementation

Our use case architecture was implemented in three web service modules: the embedded sensor node (programmed in C), the adaptive sampling controller (programmed in Python), and a front-end visualization and control interface (programmed in JavaScript). All services were tied together via the Xively IoT platform, which served as the interface and data storage mechanism.

4.1. Sensor node

The majority of the time, the sensor node operated autonomously at a constant measurement frequency, sampling the suite of sensors and transmitting data via the cellular connection. The sensor node's sampling frequency or sampling schedule could be changed remotely via an IoT web service request by authorized users, in particular by the adaptive sampling controller. To control any additional sensor nodes, a user only had to know the unique data feed and authentication key assigned to each device.

Given the rising popularity of ultra low-power micro-controllers (Diamond et al., 2008), including ARM- (Gartia et al., 2012), AVR- (Hartung et al., 2006) and 8051-based architectures (Jin et al., 2010), C continues to be the de-facto programming language for the majority of embedded devices. The exchange between the sensor node and the IoT platform was also programmed in C and the Xively platform offers a comprehensive C library, which includes the methods that provide the additional functionality of RESTful communications and data formatting to exchange information with the platform. A number of older or popular data loggers (such as those made by Campbell Scientific[9]) are written in proprietary or legacy languages, for which there may not be an explicit IoT library. Nonetheless, these loggers still support TCP/IP functionality via a number of communicating links. To that end, we decided to forgo the existing Xively library to illustrate the steps that could be followed to interface most IP-enabled data loggers to the IoT platform. To transmit data, our code opened a TCP/IP port and wrote a CSV or JSON-delimited set of sensor values using a RESTful command to the IoT platform. To receive commands from the IoT platform, the node listened on a given port and parsed an incoming string for relevant commands. On our C-based platform, this was achieved in as little as four lines of code by leveraging an existing TCP/IP library.

4.2. Adaptive sampling controller

Implemented as a Python script, this controller sought to maximize the probability of capturing first flush events while

[8] https://github.com/kLabUM/IoT.

[9] http://www.campbellsci.com/.

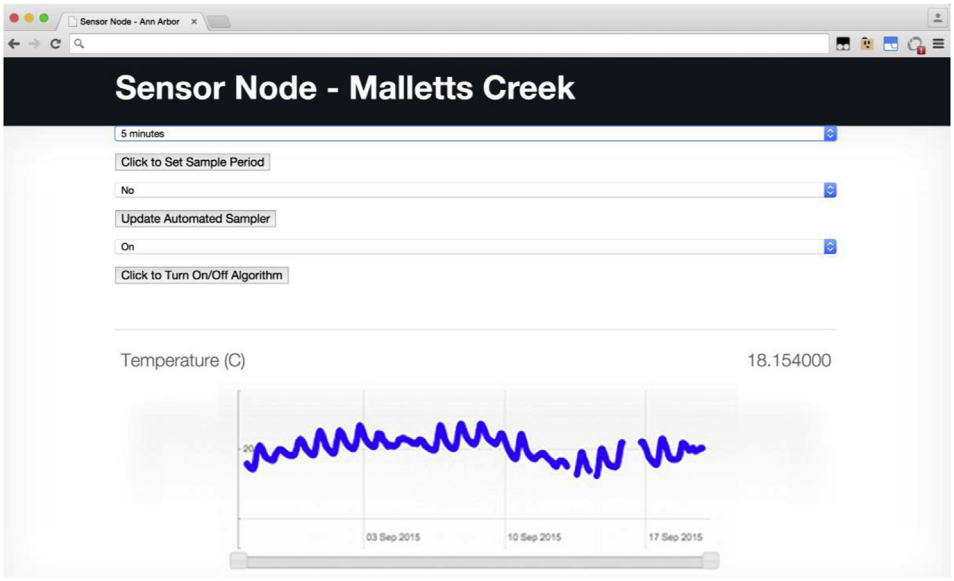

Fig. 4. Example web front-end used to monitor readings from the sensor node and control its sampling frequency.

minimizing the number of water quality samples. The controller persistently updated its knowledge of local weather forecasts by leveraging *Weather Underground*'s *Weather API web service*[10]. Once notified of the most recent sensor node measurements through Xively, the controller then updated the node's sampling frequency (a variable stored on Xively) based on anticipated storm events.

The algorithm was initially developed and tested on an Internet-connected desktop, after which it was deployed as a dedicated web application using Amazon's cloud-based *Elastic Beanstalk*[11] service. This service permits non-expert users to develop code on their own workstation and launch it as a web service by simply uploading the script to the platform. The process involves no further programming beyond what is already written on the desktop computer, which makes it appealing to users who do not wish to support their own dedicated server. The service self-balances computational loads, is pre-configured to support a variety of programming languages, and removes any hosting requirements on the part of the user. Not unique to AWS, a number of other cloud-based platforms offer similar services, including, but not limited to AppFog, CloudBees, Google App Engine, Engine Yard, Heroku, OpenShift, and Windows Azure.[12]

While the adaptive sampling controller could have also been implemented on the actual sensor node in C, we envision future applications where sampling frequencies are guided by measurements made by a distributed network of sensor nodes, rather than just local measurements. In such cases an off-site sampling controller is not only more easily maintained and deployed, but also capable of coordinating a global response to signals from multiple sources. Furthermore, implementing the sampling logic in Python permitted the sampling logic to be updated rapidly (and remotely) through a web interface without having to update the lower level firmware of field devices.

[10] http://www.wunderground.com/weather/api.
[11] http://aws.amazon.com/elasticbeanstalk.
[12] https://www.appfog.com/, http://www.cloudbees.com/, https://appengine.google.com/, https://www.engineyard.com/, https://www.heroku.com/, https://www.openshift.com/, http://azure.microsoft.com/en-us/.

4.3. Visualization interface

A front-end web application (Fig. 4) was implemented as a webpage using the Xively JavaScript API to visualize data and system states. The API provided methods to authenticate with the Xively platform and exchange data in JSON format, which is widely compatible with popular visualization platforms (*d3. js*; for examples, see (Bostock et al., 2011)). Data and commands were transmitted directly to Xively, while a subscription feature in the API enabled callbacks whenever new readings were received from the sensor node. As such, data on the interface were visually updated as soon as they were received by Xively, without requiring the user to refresh the page. A set of controls also allowed users to trigger the sensor node remotely, permitting them to override the logic of the automated sampler if needed.

4.4. Web service interactions

The sensor node was programmed to spend the majority of time in a *sleep state*, where cellular and sensing capabilities were turned off to conserve battery resources, which was required to enable long-term, battery-powered deployments in remote areas. Upon transmitting a new sample, the sensor node remained connected to the Internet for a short duration, giving external services enough time to respond to the new measurements if needed. When in the sleep state, the node did not immediately respond to commands sent by the adaptive sampling controller or the web interface. Rather, it checked for the need to update its sampling schedule once it obtained an Internet connection during its next wakeup cycle. This flow of actions also removed the burden on the adaptive sampling controller to monitor the connectivity of the node, which allowed both processes to remain uncoupled.

A typical set of actions (Fig. 5) involved a sensor node taking readings and transmitting them to the IoT platform, which then pushed a notification to the adaptive sampling algorithm and web visualization interface, both of which were subscribed to the data feed via their respective library callback mechanisms. The adaptive sampling algorithm then computed the optimal sampling frequency and updated it if necessary, in which case the sensor node was notified via a push notification through the IoT platform. If the

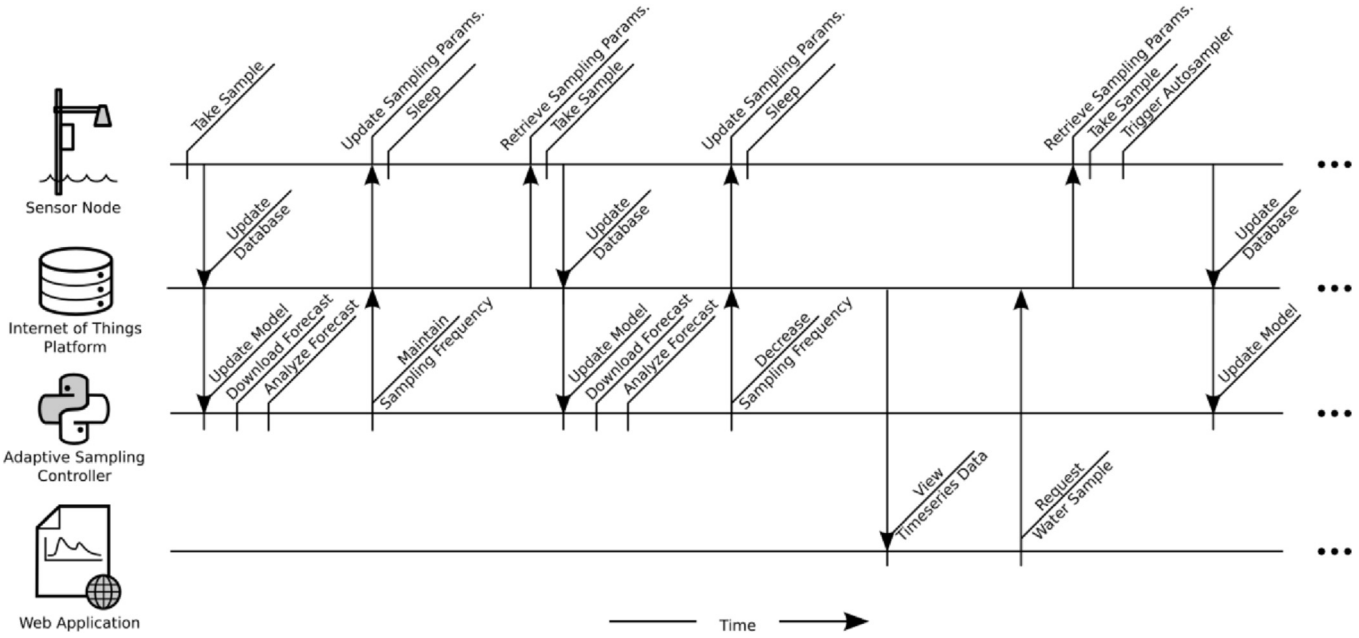

Fig. 5. Hydrologic use case action flow.

sensor node was in a sleep state when the readings had to be updated (for example, due to an unforeseen storm forecast), the adaptive controller updated the sampling frequency on the IoT platform. Upon regaining Internet connectivity, the sensor node could then compare this variable with its current settings and update itself if needed.

5. System performance and discussion

Results from the use case indicate that the sensor node, when guided by the off-site, real-time adaptive sampling controller, resolved local hydrographs while simultaneously collecting water quality samples during events of interest (Fig. 6). In particular, the node was very effective at managing the number of water quality samples required to characterize the "first flush" behavior of the study basin. Specifically, the node captured valuable baseflow samples right before the onset of a storm, while spacing out the remaining samples to measure water quality during the inflection points, peak, and recession of the hydrograph. At least six samples were used to characterize the dynamics of each distinct rain episode. In many instances, only a single set of samples (no more than 24) was necessary to capture storm events across multiple days without the need to service the node or replace sampling bottles. This would not have been feasible without a real-time sampling approach and has vastly improved the quality of our existing experiments.

During the entire three-month study period, only two baseflow samples were triggered falsely as a result of inaccurate weather forecasts, suggesting that publicly available weather feeds may have high potential to improve urban water experiments. The samples were analyzed for total suspended solids (TSS) to assess impacts of upstream stormwater runoff, showing that peak solids

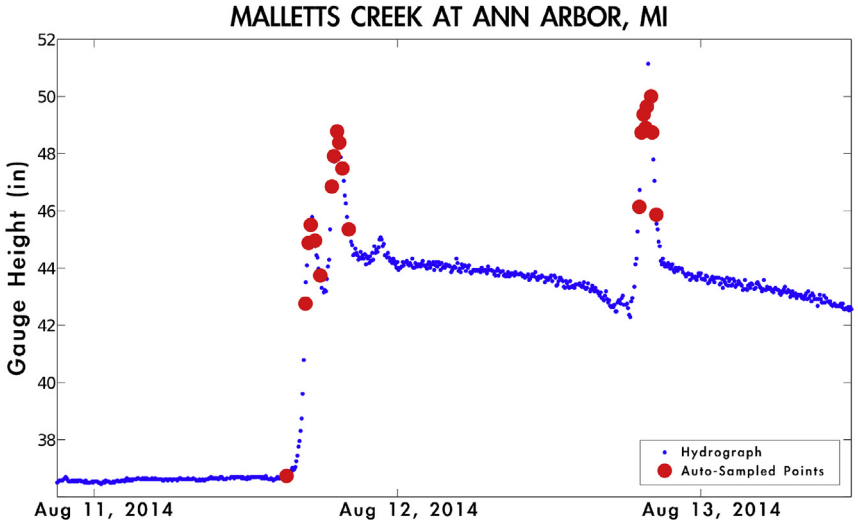

Fig. 6. Measured hydrologic signal: stage height (blue) and instances of adaptively sampled water quality data (red). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

correlated with peak flows. An in-depth water quality analysis, described in (Wong and Kerkez, 2014), concluded that no first flush behavior was observed in the catchment with regard to TSS, which may have significant implications to existing stormwater management practices in the basin. It should be noted that the analysis of a number of other water quality parameters has to conform to maximum holding times (e.g. 24 h maximum for phosphate), as the constituents may react while the sample is held in the bottle. In these instances, the benefit of real-time data is twofold: permitting for alerts to be sent when samples have to be picked up, and secondly minimizing the number of samples that need to be analyzed in the laboratory.

Overall, adaptive sampling significantly reduced fieldwork and improved the power consumption of the sensor node when compared to traditional sampling approaches. Compared to these approaches, which may take water samples once an hour (or more rapidly) during a storm to resolve relevant features of the hydrograph, our implementation was able to more effectively and more densely collect samples storm events. This reduced power draw by nearly 75% since the automated sampler was the largest battery drain due to its mechanical components. The system also provided a number of alerts to users, such as when sample bottles were about to expire, which limited site visits significantly. This reduced requirements on manpower and freed up experimental resources, permitting a multi-node network to be effectively maintained by a small team of investigators. Adaptively sampling these and other signals, however, was highly contingent upon responding to in-situ measurements and weather forecasts in real-time, which required a real-time architecture, such as the one proposed here.

By leveraging a commercial cloud-hosted IoT cloud platform, our real-time use case yielded significant benefits. Using a free account with a cloud-hosted IoT platform, we were able to make our real-time data available on the Internet through a password-protected, web-accessible endpoint, as well as interface our sensor node with a cloud-hosted adaptive sampling algorithm and web application. Development was focused largely on the sensor node and adaptive sampling algorithm. Particularly, no IoT platform outages were experienced throughout the three-month summer sampling campaign (July 1 to October 1, 2014) due to guaranteed uptimes by the platform provider. Building the architecture around a commercial platform also ensured that the overall system would benefit from security and system updates at no expense to the user. The IoT platform also served as an effective data storage, retrieval and visualization engine for continuous sensor streams. As such, the experiment was afforded the benefits of a conventional, non real-time platform as well.

The majority of our use case efforts focused on implementing modular web services, each of which was written in a different programming language and deployed on a system deemed most suitable for its use. The interoperability of web services provided significant flexibility during development and deployment, as it permitted us to focus on the application and leverage our core competencies rather than having to conform to specific languages, operating systems and hardware architectures. Moreover, adjustments could be made to individual web services without affecting or compromising the functionality of the rest of the system. For example, implementing our Python-based adaptive sampling controller as a web service allowed us to rapidly change sampling strategies without having to modify any low-level firmware on the sensor node.

In the example use case, all of the web services interacted through the IoT platform. In such an architecture, feeds and alerts must be routed rapidly enough to meet the needs of the application. Latency thus becomes a concern when framing any architecture around an IoT platform. To address this concern, we carried out an experiment in which two web services were created and connected via the Xively IoT platform. The controlled experiment was designed to emulate an adaptive sampling procedure where a sensor node first transmits a reading that is interpreted by an off-site controller, which then instructs the sensor node of its new sampling schedule. Three hundred data packets were transmitted from one web service, forwarded by Xively to the second service, and then transmitted back through Xively to the first service. The total travel time was measured, yielding an average of 0.2 s round-trip (min 0.002 s, max 2 s). This overall latency was not only guided by the response time of IoT platform, but by other factors such as network connectivity and bandwidth. In our hydrologic use case, where the average sampling interval was rarely required to drop below five minutes, this response time was more than adequate to meet the needs of the application. Such a performance should also adequately meet the needs of the vast majority of real-time environmental applications, most of which rarely require sub-second temporal resolutions for purposes of control and decision-making. Applications requiring very fine-grained response times (milli-to micro-seconds) can still leverage the majority of features offered by IoT platforms but should consider more localized, on-board signal processing and control where possible (e.g. on the sensor node, as opposed to off-site services). In the case where an IoT platform still does not meet the needs of such applications, it can still serve as a directory and discovery mechanism that interfaces web services, which can then communicate with each other directly. Such customized architectures should, however, rarely be required for the vast majority of environmental sensor network applications.

Aside from latency concerns, a web service architecture could suffer from connectivity outages as well. As pointed out by (Castronova et al., 2013), the overall functionality of a web-coupled architecture may suffer if it is entirely reliant upon being interconnected by the Internet, especially when services are hosted at different locations. Connectivity outages (for example, those experienced in wireless sensor network applications) may thus intermittently affect portions of a real-time architecture. To that end, a level of autonomy should always be built into individual services to ensure that they maintain their core set of functionalities even if connectivity is compromised. In our use case, the sensor node would continue to sample and transmit data at a default sampling interval, even if the adaptive sampling controller were to experience an outage. On-board storage on the sensor node also maintained a local copy of the data that could then be re-uploaded in the future, ensuring a continuous stream of data regardless of IoT functionality.

In most real-time environmental applications, particularly those relating to smaller-scale scientific studies, commercial platforms will often provide low-cost or free operations, with minimal overhead to setup and begin using real-time services provided by the platform. However, the management and control of commercial IoT platforms is subject to provider policies and subject to future changes. As is the case with most commercial software and systems (e.g. cloud computing services), potential drawbacks include throttled usage and loss of support for certain features. The benefits offered by open source platforms make them a viable real-time alternative for users willing to commit resources to both developing maintaining the platform. While they require more expertise during setup and maintenance, open source platforms may be a viable option for more advanced users already experienced in developing and deploying web applications as they allow more control over their system and their data. These platforms provide more low-level configurability to the user and collected data does not have to reside on third party databases. Furthermore, users are not explicitly restricted by usage limitations or by data formats.

Nonetheless, when considering the use of IoT platforms and web services for real-time environmental applications, users will also ultimately need to weigh the benefits of and drawbacks of commercial platforms against their open source counterparts.

6. Conclusions

Recent advances in sensing, computation and communications have enabled the rapid deployment of real-time data systems for environmental applications. In particular, most modern sensor systems can now seamlessly connect to the Internet via standard web protocols, permitting the use of web services as an ideal interoperability mechanism between sensors, actuators, models and decision support systems.

The ability to respond to data as it is measured brings two major benefits to environmental applications: 1) it enables a means by which to significantly improve the quality and reach of experiments (as illustrated by our hydrologic use case), and 2) it serves as a powerful tool for decision-making and control (e.g. contaminant warning systems, flood control, etc). Even with the current ecosystem of open source platforms, the deployment of current real-time environmental data systems is largely non-trivial, which significantly limits their adoption. To that end, we have shown that commercially available IoT platforms, which have been designed for a broad suite of applications, provide a secure and scalable mechanism for processing, storing, and visualizing ever increasing amounts of data.

The flexibility afforded by the web service-driven nature of these platforms loosens the architecture-, hardware- and software-specific requirements that often underpin several existing real-time data platforms. As illustrated by our hydrologic use case, this flexibility reduces the barrier of entry for most environmental applications as it permits users, novice and experienced, to build upon existing projects and work with the programming languages and platforms that they find most appealing. Regardless of the ease of use, demands on the user are not entirely eliminated. An initial investment to develop the appropriate skillset to work with IoT platforms will inevitably have to be made by end users. We contend, however, that learning how to integrate web services into already existing code provides a compelling value proposition given the large community of adopters and supporters that is growing in the IoT space.

While the availability and features of open source IoT platforms continue to expand, environmental applications presently stand to gain the most from leveraging commercial IoT systems, which offer a vast suite of features at the click of a mouse. Presently, these enterprise-quality platforms have the potential to enable the ubiquitous use of real-time environmental sensor data and to usher in a new generation of adaptive scientific experiments.

Acknowledgements

The University of Michigan provided financial support for the project. We would like to thank Ric Lawson and Lauren Burns at Huron River Watershed Council for providing ideas and field support during the evaluation of the use case. We would also like to thank the EarthCube CHORDS team, Sam Hatchett, Jon Pollack, and Bradley Eck for engaging with us on great conversations around real-time data. Ho-Zhen Chen and Yanglin Zhao were also instrumental in the deployment of the sensor nodes.

References

Argent, R.M., Perraud, J.-., Rahman, J.M., Grayson, R.B., Podger, G.M., 2009. A new approach to water quality modelling and environmental decision support systems. Environ. Model. Softw. 24, 809–818.
Årzén, K., 1999. A simple event-based PID controller. In: Proc. 14th IFAC World Congress, Anonymous, pp. 423–428.
Åström, K.J., Bernhardsson, B., 1999. Comparison of periodic and event based sampling for first-order stochastic systems. In: Proceedings of the 14th IFAC World Congress, Anonymous Citeseer, pp. 301–306.
Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: a survey. Comput. Netw. 54, 2787–2805.
Bostock, M., Ogievetsky, V., Heer, J., 2011. D^3 data-driven documents. Visualization and Computer Graphics. IEEE Trans. 17, 2301–2309.
Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25, 599–616.
Castronova, A.M., Goodall, J.L., Elag, M.M., 2013. Models as web services using the open geospatial consortium (OGC) web processing service (WPS) standard. Environ. Model. Softw. 41, 72–83.
Christin, D., Reinhardt, A., Mogre, P.S., Steinmetz, R., 2009. Wireless sensor networks and the internet of things: selected challenges. In: Proceedings of the 8th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, pp. 31–34.
Christodoulou, S., Agathokleous, A., Kounoudes, A., Mills, M., 2010. Wireless sensor networks for water loss detection. Eur. Water 30, 41–48.
Daniels, M., Chandrasekar, V., Graves, S., Harper, S., Kerkez, B., Vernon, F., 2013. Executive summary: earthcube real-time workshop results. In: Integrating Real-time Data into the EarthCube Framework, Boulder, Colorado, Anonymous.
Davis, G.P., Rew, R.K., 1994. The Unidata LDM: programs and protocols for flexible processing of data products. In: Proceedings, Tenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Nashville, Tennessee, American Meteorology Society, Anonymous.
Diamond, D., Coyle, S., Scarmagnani, S., Hayes, J., 2008. Wireless sensor networks and chemo-/biosensing. Chem. Rev. 108, 652–679.
Díaz, L., Bröring, A., Mcinerney, D., Libertá, G., Foerster, T., 2013. Publishing sensor observations into Geospatial Information Infrastructures: a use case in fire danger assessment. Environ. Model. Softw. 48, 65–80.
Field, R., 2002. Wet-weather Flow in the Urban Watershed: Technology and Management. CRC Press, Hoboken.
Fleisch, E., 2010. What is the internet of things? An economic perspective. Econ. Manag. Financ. Mark. 125–157.
Gall, H.E., Jafvert, C.T., Jenkinson, B., 2010. Integrating hydrograph modeling with real-time flow monitoring to generate hydrograph-specific sampling schemes. J. Hydrol. 393, 331.
Gartia, M.R., Braunschweig, B., Chang, T.W., Moinzadeh, P., Minsker, B.S., Agha, G., Wieckowski, A., Keefer, L.L., Liu, G.L., 2012. The microelectronic wireless nitrate sensor network for environmental water monitoring. J. Environ. Monit. JEM 14, 3068–3075.
Goodall, J.L., Horsburgh, J.S., Whiteaker, T.L., Maidment, D.R., Zaslavsky, I., 2008. A first approach to web services for the National Water Information System. Environ. Model. Softw. 23, 404–411.
Gronewold, A.D., Clites, A.H., Smith, J.P., Hunter, T.S., 2013. A dynamic graphical interface for visualizing projected, measured, and reconstructed surface water elevations on the earth's largest lakes. Environ. Model. Softw. 49, 34–39.
Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M., 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29, 1645–1660.
Hartung, C., Han, R., Seielstad, C., Holbrook, S., 2006. FireWxNet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In: Proceedings of the 4th International Conference on Mobile Systems, Applications and Services, Anonymous ACM, pp. 28–41.
Hefeeda, M., Bagheri, M., 2009. Forest fire modeling and early detection using wireless sensor networks. Ad Hoc Sens. Wirel. Netw. 7, 169–224.
Horsburgh, J.S., Tarboton, D.G., Piasecki, M., Maidment, D.R., Zaslavsky, I., Valentine, D., Whitenack, T., 2009. An integrated system for publishing environmental observations data. Environ. Model. Softw. 24, 879–888.
Jin, N., Ma, R., Lv, Y., Lou, X., Wei, Q., 2010. A novel design of water environment monitoring system based on WSN. In: Computer Design and Applications (ICCDA), 2010 International Conference on, Anonymous IEEE, V2-593-v2-597.
Kübert, R., Katsaros, G., Wang, T., 2011. A RESTful implementation of the WS-Agreement specification. In: Proceedings of the Second International Workshop on RESTful Design, Anonymous ACM, pp. 67–72.
Lakshmanan, V., Smith, T., Stumpf, G., Hondl, K., 2007. The warning decision support system-integrated information. Weather Forecast. 22, 596–612.
Lamela Seijas, P., Li, H., Thompson, S., 2013. Towards property-based testing of RESTful web services. In: Proceedings of the Twelfth ACM SIGPLAN Workshop on Erlang, Anonymous ACM, pp. 77–78.
Lee, E.A., Seshia, S.A., 2011. Introduction to Embedded Systems: a Cyber-physical Systems Approach. Lee & Seshia.
Mainetti, L., Patrono, L., Vilei, A., 2011. Evolution of wireless sensor networks towards the internet of things: a survey. In: Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on, Anonymous IEEE, pp. 1–6.
Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D., 2014. Context aware computing for the internet of things: A survey. Commun. Surv. Tutor. IEEE 16, 414–454.
Rea, R., 2013. IBM InfoSphere Streams."Redefining real time analytic processing". In: IBM Software, Thought Leadership White Paper, pp. 1–8.

Sansalone, J.J., Cristina, C.M., 2004. First flush concepts for suspended and dissolved solids in small impervious watersheds. J. Environ. Eng. 130, 1301–1314.
Scavia, D., Evans, M.A., Obenour, D.R., 2013. A scenario and forecast model for Gulf of Mexico hypoxic area and volume. Environ. Sci. Technol. 47 (18), 10423–10428, 130904143022002.
Stocker, M., Baranizadeh, E., Portin, H., Komppula, M., Rönkkö, M., Hamed, A., Virtanen, A., Lehtinen, K., Laaksonen, A., Kolehmainen, M., 2014. Representing situational knowledge acquired from sensor data for atmospheric phenomena. Environ. Model. Softw. 58, 27–47.
Taylor, P., Valentine, D., Walker, G., Sheahan, P., Pratt, A., Dornblut, I., Grellet, S., Heidmann, C., Wilhelmi, J., Christian, M., 2010. Harmonising Standards for Water Observation Data-discussion Paper (Open Geospatial Consortium).
Teklemariam, G.K., Hoebeke, J., Moerman, I., Demeester, P., 2013. Facilitating the creation of IoT applications through conditional observations in CoAP. EURASIP J. Wirel. Commun. Netw. 2013, 1–19.
Thubert, P., Watteyne, T., Palattella, M.R., Vilajosana, X., Wang, Q., 2013. IETF 6TSCH: combining IPv6 connectivity with industrial performance. In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on, Anonymous IEEE, pp. 541–546.
Tilak, S., Hubbard, P., Miller, M., Fountain, T., 2007. The ring buffer network bus (RBNB) dataturbine streaming data middleware for environmental observing systems. In: e-Science and Grid Computing, IEEE International Conference on, Anonymous IEEE, pp. 125–133.
Wang, G., 2011. Improving data transmission in web applications via the translation between XML and JSON. In: Communications and Mobile Computing (CMC), 2011 Third International Conference on, Anonymous IEEE, pp. 182–185.
Watteyne, T., Vilajosana, X., Kerkez, B., Chraim, F., Weekly, K., Wang, Q., Glaser, S., Pister, K., 2012. OpenWSN: a standards-based low-power wireless development environment. Trans. Emerg. Telecommun. Technol. 23, 480–493.
Wong, B.P., Kerkez, B., 2014. Adaptive, decentralized, and real-time sampling strategies for resource constrained hydraulic and hydrologic sensor networks. In: HIC 2014 – 11th International Conference on Hydroinformatics, New York, USA, Anonymous.
Zhang, J., Howard, K., Langston, C., Vasiloff, S., Kaney, B., Arthur, A., Van Cooten, S., Kelleher, K., Kitzmiller, D., Ding, F., 2011. National Mosaic and Multi-Sensor QPE (NMQ) system: description, results, and future plans. Bull. Am. Meteorol. Soc. 92, 1321–1338.
Zhang, Y., Bocquet, M., Mallet, V., Seigneur, C., Baklanov, A., 2012. Real-time air quality forecasting, part I: history, techniques, and current status. Atmos. Environ. 60, 632–655.
Zhao, Y., Kerkez, B., 2014. Cellular-enabled water quality measurements. In: AGU Fall Meeting 2014.
Zhou, Y., Obenour, D.R., Scavia, D., Johengen, T.H., Michalak, A.M., 2013. Spatial and temporal trends in lake erie hypoxia, 1987–2007. Environ. Sci. Technol. 47, 899–905.