sMAP – a Simple Measurement and Actuation Profile for Physical Information

Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler

Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
{stevedh,fxjiang,gtolle,jortiz,culler}@cs.berkeley.edu

Abstract

As more and more *physical information* becomes available, a critical problem is enabling the simple and efficient exchange of this data. We present our design for a simple RESTful web service called the Simple Measuring and Actuation Profile (sMAP) which allows instruments and other producers of physical information to directly publish their data. In our design study, we consider what information should be represented, and how it fits into the RESTful paradigm. To evaluate sMAP, we implement a large number of data sources using this profile, and consider how easy it is to use to build new applications. We also design and evaluate a set of adaptations made at each layer of the protocol stack which allow sMAP to run on constrained devices.

Categories and Subject Descriptors

J.7 [**Computer Applications**]: Computers in Other Systems

General Terms

Design, Management, Standardization

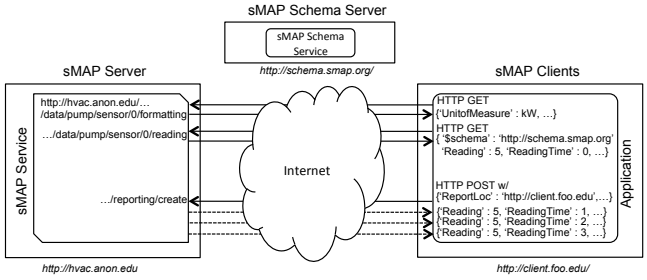
Keywords

Sensor data, Instrumentation

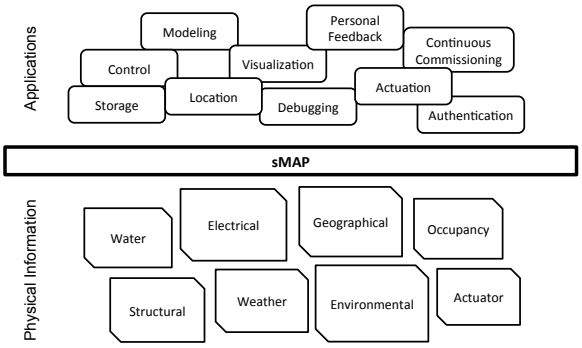
1 Introduction

As traditional instrumentation has become ever more networked, as wireless sensor networks have allowed instrumentation to become more diverse and more pervasive, and as physical information has become widely used in a broad range of applications, there is growing consensus that networked sensors should be viewed essentially as tiny embedded information servers. Web Services hold the potential to enable the integration of diverse sources of physical information, as do diverse conventional sources [1, 17, 29]. Uniform, machine independent access to self-describing physical information would obviate the innumerable drivers and adapters for specific sensor and actuator devices found in industrial building automation, process control, environmental monitoring solutions. Physical information is in many ways more challenging to handle than conventional data because its interpretation is so dependent on the behavior and context of the particular sensor or actuator; also, the diversity of sensors is huge. In many cases, the representation, transportation, and storage of this data must be extremely efficient. Thus, while it is easy to wrap readings in XML and transport them over HTTP, it is challenging to get widespread agreement on a simple, easily understood solution.

We propose a simple representation of measurement information and actuation events based on modern RESTful web service techniques that allows for arbitrary architectural composition of data sources, freeing application designers from tight frameworks and enabling widespread exploration



(a) An example of how sMAP is normally used.



(b) sMAP sits between producers and consumers of physical sensor data.

Figure 1. Schematic views of a single sMAP interaction, and where sMAP sits in relation to other components.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is premitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SenSys'10, November 3–5, 2010, Zurich, Switzerland.
Copyright 2010 ACM 978-1-4503-0344-6/10/11 ...$10.00

of the sensor-application arena. We take the practicality of implementing web services on all classes of devices to be a given and instead focus on defining a *particular* web service for exchanging physical data. Success means not just being appropriate for a single application but becoming a widely accepted interchange format for sensor data on the Internet.

The design space for a web service for physical information consists of three interlocking areas:

Metrology the study of measurement; *what* is necessary to represent a datum.

Syndication concerns *how* a data is propagated out from the sensor into a larger system.

Scalability relates to the *range of devices* and uses the service can support, from small embedded systems to huge Internet data centers.

Each of these concerns presents a set of design decisions, some of which have been previously addressed in the academic literature or by industrial efforts. In this work, we examine these previous solutions and build from them a single set of solutions which are designed to solve a specific problem: representing and transmitting physical information.

A prototypical interaction between a client and an instrument exposing a sMAP interface is shown in Figure 1(a). The client contacts the device over HTTP and discovers what resources, and thus what data is provided by the instrument. In the figure, the client first determines what the units of a particular channel and measurement point are, before taking a sample by fetching the `reading` resource. Finally, the client arranges for period reports to be delivered to him. Since all communication references a schema, the client can also fetch the schema to validate the data.

sMAP was not developed in a vacuum; it forms the basis for a build to electric grid testbed which provides fine-grained, multi-resolution sensor data about the building under inspection. The project currently provides around 2000 distinct measurement channels. These channels monitor electricity consumption, environmental quality data, HVAC parameters, weather data, and more. An early problem we confronted was how to integrate all of these sensors to allow "building applications" to be possible; the answer was sMAP. sMAP is designed to unify access to legacy instruments with a consistent representation and up-to-date technological underpinnings. While many recognize the need for such integration, much of the work is present in proprietary products and seek to interoperate only within the particular industrial sector. We believe that by presenting a simple but carefully-considered specification along with a substantial amount of data, we can bootstrap the process of making all physical information universally accessible. [1]

In the following sections, we present the design of our service, and evaluate it both on traditional metrics like code size and message complexity, but also by presenting how a diverse set of data sources are made available in this format. Furthermore, we preview some of the sMAP "applications" which are made possible by having a uniform data plane.

[1]Source code for a variety of sMAP servers, as well nearly all of the data feeds presented in this paper are available at `http://smap.cs.berkeley.edu`.

2 Background

The topics of compact web services, content syndication, and the representation of physical data have all been addressed in different communities.

2.1 Compact Protocol Design

Traditional Internet design discipline has dictated that application protocols should use simple ASCII or unicode messages for ease of debugging, implementation, interoperability, and extensibility. Much previous work has addressed the challenges of bringing Internet protocols including web services to embedded and other constrained devices [29, 39, 23, 27]. Since web service protocols like SOAP-WS seem best-suited to the types of services we are using, some work has focused on designing efficient compression formats for transmitting this data over links with small MTUs. For instance, Tiny Web Services explored methods of compressing a SOAP envelope using both generic algorithms like zip and LZW, and also XML-specific compression techniques. In their evaluation, techniques which can leverage knowledge of the underlying protocol resulted in the best compression ratios.

The paradigm of Representation State Transfer, or REST, provides an alternative to SOAP which is build around the idea of the transfer of "resources" [12]. RESTful services are built on top of HTTP, which avoids re-defining much of the functionality present in that protocol. Recent work has also explored bringing REST together with physical data, an important advance [32]. However, these services are still typically too verbose for embedded devices, an issue we address in our protocol design; at the lower layers we use techniques like `6lowpan` header compression and IEEE 802.15.4 links.

The result of this recent work on embedded web services should cause us to add a corollary: protocol design should also take into account the need for a stateless compression mechanism to and from a compact binary format.

2.2 Decentralized Architecture

Other work has federated large collections of sensors into a single system that may be treated uniformly. Creating systems which use data from a huge variety of sources is indeed important, and there are several architectural possibilities to choose from when doing so. Existing systems tend towards the "aggregator" model, where a centralized system draws in feeds from different sources and subscribers access the data indirectly through this central mediator. This architecture is simple and easy to understand, and has some advantages. The predominant examples of this system are Google PowerMeter and Microsoft SenseWeb.

Google PowerMeter provides a RESTful API for publishing data to the Google cloud, where it is then available through iGoogle and potentially other Google web properties [1]. Data publishers format their data into a flat name space, and use HTTP to periodically transmit the data to Google. It is designed only to put data *into* the system, and furthermore ties a particular feed to a single user; a concept that doesn't map onto how many physical sensors are employed as part of a larger process.

Microsoft SenseWeb is a system for the integrated sensing and querying of sensor data [17]. Users with an API key can use a .NET system to add data objects to the system. The overall architecture takes the form of a multi-stage stream processing system, where intermediate units can perform online data processing and forward the results to the next unit, forming a Directed Acyclic Graph.

pachube is a hosted web application for collecting and cataloging a large variety of sensor data, and has a HTTP/JSON interface similar to sMAP [2]. Once data is loaded into the system, a number of different visualization and analytics plugins are available to examine readings. This system is very relevant, and can be considered essentially an example of the type of universal application which can be built on top of a common sensor plane such as sMAP.

From the commercial process-control world, the OSISoft PI System archives and presents time-series of readings from different sensors. Its major strength is the library of drivers for a large numbers of existing automation systems present in buildings and plants. Unfortunately, its technological underpinnings look backwards rather then forwards: applications are mostly designed using Microsoft's proprietary tools and protocols, and integrating a new sensor is a task for the company's software engineers: the software contains drivers for 430 interfaces and COM connectors to load data from existing sources.

A common thread of these systems is that they make it easy to put data *in*. In contrast, sMAP is designed to get data *out* to as many different consumers as possible.

2.3 Syndication

The related work closest in design to ours comes not from the field of sensor networks, but from content syndication on the Internet. IETF standards like Atom and RSS have shown how a very simple envelope container that can be automatically parsed can have dramatic uptake and spur a family of applications [26]. The key design principle is to be "really simple": do exactly one thing, and do it well.

Traditional work on information busses using the publish/subscribe model would meet our requirements for making data broadly available across the Internet; examples would include CORBA, and more recently the Advanced Message Queuing Protocol (AMQP) [37]. This type of functionality is also often integrated into "Message Oriented Middleware" such as Java Messaging System (JMS) or TIBCO. All of these products are enterprise oriented, and generally require significantly more infrastructure then a simple sensor can provide. The Extensible Messaging and Presence Protocol (XMPP) was initially developed as an instant messaging protocol, but has subsequently expanded to fill a role as messaging middleware [31]. The Sensor Andrew project [30] is exploring this design and we believe further work is merited on comparing these approaches.

To address problems with scalability inherent in the use of polling RSS and Atom feeds, two approaches have emerged, paralleling the choices we face with sensor data. One, the centralized version, is exemplified by **Google Reader**: a master server performs the polling on RSS feeds, and subscribers access the data not through the end servers but through Google. The second, a decentralized architecture, is used by **pubsubhubbub** and augments HTTP, a fundamentally client-server architecture with notifications and deliveries for RSS feed updates [13]. Although there are several techniques for doing this, pubsubhubbub delivers notifications to clients via HTTP POST requests. Since our design is decentralized, we draw on ideas from `pubsubhubbub` to provide scalable syndication of sensor data, even from devices with very few resources.

2.4 Data Representation

There are many industry efforts in use or under development which are comparable to sMAP. In general, these efforts are either targeted at very specific problem spaces and specify complete solutions, or are "lowest common-denominator" type specifications. Additionally, there is some related protocols and designs from academia which are very relevant.

One use case which has attracted a good deal of attention are Smart Home and Smart Grid applications. Driven by NIST efforts to identify and support interoperable smart grid protocols [3], bodies such as the Zigbee Alliance are working to identify and specify protocol stacks for this usage. The upcoming Smart Energy Profile 2.0, for instance, will provide for device-level interoperability for energy-consuming appliances in the home [33]. Although a complete specification is not available, published information indicates that will be built around an IPv6-based stack, with adaptations at each layer of the stack coming from the IEEE, IETF, and W3C (802.15.4e, 6lowpan, ROLL, CoRE, and EXI). Protocol work in these groups is ongoing, however, the data model is still unspecified; this is approximately where sMAP could fit.

Within the same space, a standard known as ANSI C12.19 defines a data model for revenue-grade electric meters [5]. C12.19 includes metadata for describing the electrical characteristics of the metering device, device identification, units of measure, and the measurement registers themselves. sMAP's data model is significantly simpler than C12.19's, embodies similar concepts for register readings and units of measure, but is not intended to capture the full complexity of common industry practice for revenue-grade electric metering.

SensorML, developed by the OpenGIS consortium is an XML specification designed to completely model sense points: it allows extensive specification of the sensor data and instrument type, in addition to context such as geolocation data and legal on the data [8]. The format is being driven by GIS-type applications and a significant amount of data is currently available in this format. sMAP's data model is in many ways similar to that developed by SensorML, but contains no "external" metadata – that is, metadata which depends upon the particular location of the instrument. By separating this type of information from the actual sensor readings, we increase flexibility; for instance, the coordinate-centric location models used by SensorML might not be appropriate for sensors located in a building, where the location is hierarchically defined by room and floor, or one in a process control application. sMAP also defines additional features to make it easy to syndicate data streams into online stream-processing systems.

Metadata	Use
Value	The quantity
Units	Interpretation
Measured Quantity	Type of measurement: water, electric, *etc*
Scaling coefficient	Conversion to engineering units without loss of precision
Global timestamp	Interpretation and alignment
Sequence number	Missing data detection; computing actual sampling period
Instrument range (min-max)	Establish dynamic range
Instrument identifier	Establish traceability

Table 1. Minimum metadata required to interpret a scalar data stream.

Another set of standards and schema emerging from the construction industry are the Industry Foundation Classes (IFC) and the related Extended Environments Markup Language (EEML) [4, 20]. Finally, Modbus is a very simple industry standard for transferring data to and from sensors, typically carried over serial links or in IP packets. The problem with Modbus's data model is that it does not exist; each instrument has its own set of registers and a map which a developer can use to translate those register locations into meanings.

The major shortfall we find with most of these efforts is that they specify either too little or too much. Both SensorML and IFC/EEML specify data models, but they leave means of publishing the unconstrained. While very useful for interoperability, it is impossible to meaningfully make an instrument which presents its data as "SensorML," and so does not eliminate the need to write "glue" code to interface with a new meter. Since sMAP defines both the data model and the access method, it is possible to have multiple interchangeable instruments presenting a consistent interface.

3 sMAP Design

sMAP's design comprises two underlying aspects: the *metrology* deals with what abstract quantities should be represented and how they should be organized; the *architecture* has to do with how the metrology is implemented in a real system. At its core, sMAP is a method of making available *discrete*, *scalar* data and control points. Although other data sources such as imagery and acoustic data are also common, we do not include them in the the design; they are addressed by existing work on multimodal sensor data repositories.

3.1 Metrology

Although scalar measurements consist of a single number, their interpretation depends on knowing how to convert that number into engineering units as well as a host of other information. The value is always a digitized signal from an instrument, measuring a property of the physical world. By providing an indication about what that property is and what units were used, a client can tell the basic type of instrument present. A timestamp and sequence number place the value within a stream of discrete values. A scaling coefficient extends the range of values representable in the format.

The problem of specifying units admits several possible solutions. In principle, all units can be derived from the seven independent SI units. However, specifying all units in terms of their derivation from these units is often cumbersome and leads to an overly-complex specification system for a design which is nominally "simple." Therefore, sMAP includes an enumerated list of commonly used units and supports other units using simple strings. This list includes at least one canonical unit for each type of measurement.

Traceability in measurement is the concept of being able to trace a given datum back to a calibration with a source of known uncertainties; normally, a chain of calibrations are made back to an authority such as NIST. In order to establish this, it is necessary to know not just what type of instrument was used, but the particular device in question so that the measurement may later be cross-referenced with calibration records. To meet this need, we allow including a device-specific unique identifier such as a serial number.

3.1.1 Measurement Location

Another frequently-encountered property of instrumentation is that a sensor will present multiple scalar quantities measured at the same point. For instance, an electric meter often measures voltage and current in addition to power, while a control point in an HVAC system will measure temperature and flow rate at the same location. Therefore, we define two terms dealing where where measurement occurs.

Measurement Point a specific point in a process which has been fitted with instrumentation. For instance, a circuit level meter in an electric system or a flow meter located on a pipe.

Channel A single stream of scalar values from a measurement point. Example channels include the a voltage or power reading from the circuit-level meter, or the instantaneous flow rate at a flow meter.

Using this decomposition, the source of a stream of sensor data from an instrument is uniquely identified by its measurement point and channel.

3.1.2 Modalities

sMAP is designed to facilitate three predominant modalities of interaction. Two, sensing and metering, are read-only. The third, actuation, allows a user to change the state of the device. *Sensing* refers to scalar quantities which are measured instantaneously. Typical examples are temperature, or voltage, or position. These quantities are typically measured at a constant rate, although data consumers downstream may re-sample the streams so as to align it with other data. *Metering* refers to values representing accumulated quantities. These are usually properties of flows: measuring the total volume of fluid flow, or amount of heat transferred. Finally, *actuators* are two-way; they allow a client to modify the physical state of some device. Because of the large diversity of actuators, sMAP cannot claim to represent all types. Instead, we provide a small library actuator templates and allow the protocol to be extended in the future.

Table 2 shows the list of basic actuator types supported by sMAP. Although many more types are possible, these basic types cover a huge fraction of basic actuators actually in use in deployments.

For each actuator, sMAP supports `get` and `set` operations. For all types of default actuators, the `set` action in idempotent. sMAP supports *versioned set* so as to provide

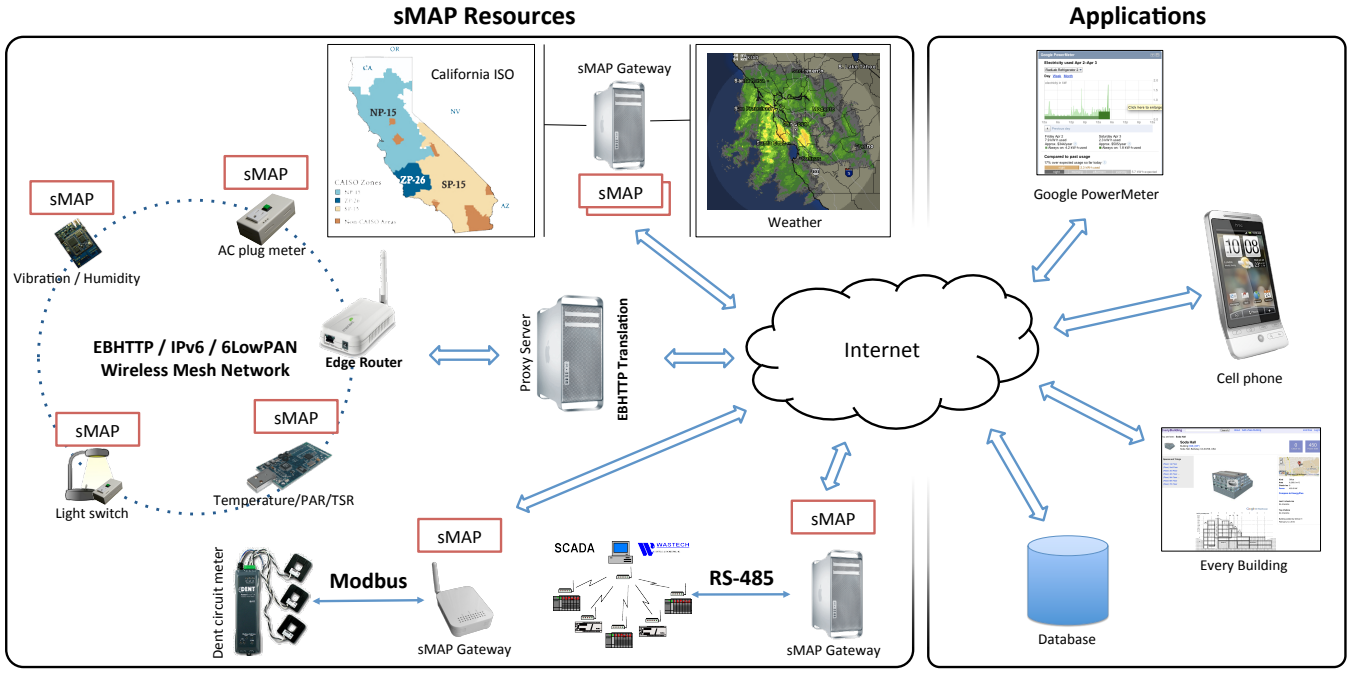


Figure 2. The full diversity of sMAP sources and gateways we have implemented under the aegis of a building monitoring project. Sources include Modbus/RS-485 meters, 6lowpan/IEEE802.15.4 wireless devices, and proxied data from weather and grid sources. sMAP serves as the data interchange layer sitting between those instruments, and applications including a Google PowerMeter gateway, mobile applications, databases, and EveryBuilding. EveryBuilding is a web application for modeling the relationships between spaces in a building, and managing information about the location of all the sense points.

Name	Description	Examples
Binary	two possible states	switches
N-State	a finite set of positions	stepper motors, discrete dimmers
Set point	choose a setting in a continuous range	thermostats, fan speed controllers
Control bands	influence a control loop by setting min and max range	thermostats

Table 2. Types of actuators supported by sMAP without extensions.

simple transactional semantics. To support this functionality, the `get` function returns a nonce in addition to the device state; in order to subsequently change the device state, the nonce in the `set` request must match the current device state (the nonce is changed with each successful `set`). This allows clients to implement functionality like "toggle" without confusing behavior when multiple parallel requests are present.

To meet the need for more complicated actuation, we also allow actuators to create new types which are not part of sMAP. Although it is recommended one of the actuator types in Table 2 if possible, the world of possibilities is large.

3.2 Service Design

The second design issue is how data information is mapped into actual protocol messages and documents. If we were targeting only direct, human consumption of the data, an HTML interface would be appropriate, and indeed this is what many instruments provide. Since our target is applications, a machine-to-machine service is desirable. We chose JSON as the object exchange format, since its data structures map naturally onto those of modern programming languages, obviating the use of a specific API like SAX or DOM to access the data. JSON can also be automatically translated to XML when required.

3.2.1 HTTP Mapping

Using HTTP, we expose all of sense points and channels as resources at standardized URLs with respect to a sMAP root and follow the paradigm of Representational State Transfer (REST) [12]. Haphazard use of this design pattern is much maligned, but a RESTful approach to web services design is characterized by a systematic use of its conventions. Because the abstractions and data model we developed map neatly onto resources, we hold that this is a good fit for physical information.

The four top-level resources in the sMAP profile are

`/data` contains resources for reading and controlling meters, sensors, and actuators.

`/reporting` allows control of periodic reports for syndication, discussed in Section 3.4.

`/status` contains a single universal universal field specifying if the device data is valid, as well as instrument-specific codes.

`/context` contains any information about the device's relationship to other devices. This includes the device's Global Unique Identifier.

All meter, sensor, and actuator data is placed

in the "data" top-level resource. Within this location, data is organized hierarchically in the form of `/data/<point>/<modality>/<channel>/<object>`. The measurement point and channel resources may be named by any valid URL-encoded string; Figure 3 shows the full URL for a "reading" object for a three-phase electric meter.

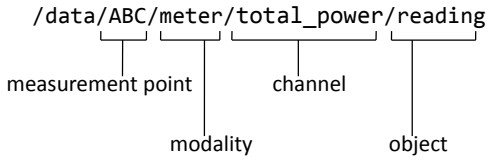


Figure 3. Canonical sMAP URL for the "total power" meter of a three-phase electric meter.

Each channel provides several leaf resources. The first, `reading`, contains the single latest sample from the device, along with a global timestamp and sequence number. The `formatting` resource contains all information necessary to interpret the reading object and convert to engineering units: scaling coefficients, units, and a key indicating what is being measured (*e.g.*, electricity, water, *etc*). The `parameter` resource provides information about the underlying measurement: generally, a sampling period/frequency and any instrument-specific parameters.

Using HTTP, these three resources must be available using the GET verb. Subject to access control, an instrument may also implement a POST form of the request. When this is used, a client can POST a new document to the appropriate resource to update sampling parameters or change the scaling coefficient.

```
/data                 # list sense points under resource data [GET]
  / [sense_point]     # select a sense points [GET]
    /meter            # meters provide this service [GET]
      / [channel]     # a particular channel [GET]
        /reading      # meter reading [GET]
        /formatting   # calibration and units [GET/POST]
        /parameter    # sampling parameter [GET/POST]
        /profile      # history of readings [GET]
/status               # device-specific status messages [GET]
/context              # instrument serial number or GUID [GET]
/reporting            # for syndication [GET]
  /create             # create a new periodic report [POST]
  /reports            # show current reports [GET]
    /txSkcQ6K         # a reporting instance [GET/DELETE]
```

```
POST requests supply JSON objects as arguments:
POST: http://meter1.cs.berkeley.edu/report
{ "ReportResource" : "/data/325/meter/*/reading",
  "ReportDeliveryLocation" :
              "http://webs.cs.berkeley.edu/receivereports.php",
   "Period" : 60, "Minimum" : 50, "Maximum" : 100 }
```

Figure 4. The sMAP resource hierarchy.

A forth object, `profile` is an optional resource which some instruments export. It consists of a circular buffer of recent measurements – typically the last N measurements, although the implementation might alternatively provide meter data over a fixed time interval. This resource is available for two reasons. The first is to enable a client to retry a request to ensure that 100% of data is delivered. The second is to allow applications like automatic meter reading where data is only fetched once per billing period.

3.2.2 JSON

JSON, the Javascript Object Notation is a language-independent data interchange format [10]. Since JSON is a representation with little structure, we further constrain its use within sMAP through the following rules: (1) All objects follow a schema, and contain a reference to it. (2) All schemas are versioned, and each object contains which version it implements. (3) Schemas are named by URI. A sample instance of a sensor `reading` object is shown in Figure 5.

```
{"$schema" : {"$ref" :
      "http://schema.anon.edu/sensor/reading"},
  "Version"         : 1,
  "Reading"         : 0,
  "ReadingTime"     : 1270239192,
  "ReadingSequence" : 48879 }
```

Figure 5. Instance of a sensor `reading` object.

3.3 Externalizing Metadata

No physical measurement is ever made in isolation. Beyond the extra information needed to interpret the raw reading, the reading's *context* includes the physical location of the meter and the logical relationships between the meter, the physical space, the users of that space, and other instrumented quantities. Without this information, the reading cannot be truly useful.

In designing sMAP, we had to decide whether to include this additional information in the basic profile. Experience with several implementations showed that representing this information requires a model of entity relationships that extends far beyond the metered quantity. This model is not the same in all places, even for the same meter, and will differ significantly between applications and users of sMAP.

Fundamentally applications need to associate arbitrary metadata with sMAP data sources. To support this, we include durable identifiers along with sMAP data, which are sufficient to permanently attribute the source of a measurement. However, it is the job of an external system to manage the relationship of these identifiers to the objects in the overall system.

3.4 Syndication

Another central challenge is that HTTP is fundamentally client-server, and so does not support "callbacks" or asynchronous client notifications [11] whereas sensor data is often "pushed" from the source device to a repository, analysis service, or other aspect of the operational infrastructure. To support such periodic reports, as well as alarms within the HTTP framework, we allow a sMAP server to also function as a client in order to deliver reports. This is superior to forcing data clients to poll because data generation is often event driven. Support for this mode of operation is located in the `/reporting` resource, which has two sub-resources. `create` is used to create a new "reporting instance", while `reports` allows inspection of currently configured reports.

In order to install a new report and begin delivering data, a client uses a POST request to send an object to the `/reporting/create` resource. The object contains enough information to allow the server to begin sending data to the URL specified in the DeliveryLocation field.

The effect of using periodic reporting is identical to performing a GET on the local resource specified in the Deliv-

Metadata use	Field Name	Resource location
Raw value	Reading	`reading`
Global timestamp	ReadingTime	`reading`
Reading sequence number	ReadingSequence	`reading`
Units	UnitofMeasure	`formatting`
Units	UnitofTime	`formatting`
Scaling coefficient	Multiplier, Divisor	`formatting`
Quantity type	SensorType	`formatting`
Underlying sampling rate	SamplingPeriod	`parameter`
Freshness	IntervalSinceLastReading	`parameter`
Instrument identifier	GlobalIdentifier	`/context`

Table 3. Location of important pieces of data within the sMAP HTTP resource hierarchy. The `reading`, `formatting`, and `parameter` resource are unique per-channel, while `/context` is shared by all measurement points.

eryLocation field, except that the data is pushed to the specified DeliveryLocation URL.

3.5 Adaptations

Some of the sensor and meter devices that implement sMAP are connected to the Internet through resource-constrained networks, and these networks may not have the throughput required to send full HTTP requests containing JSON objects every time a sensor reading needs to be published to a client. Furthermore, the devices themselves may have very limited resources (such as code-size and memory) to implement complicated protocols. sMAP is defined to use HTTP for data access and JSON for object representation, but it can function over compressed versions of these protocols as well.

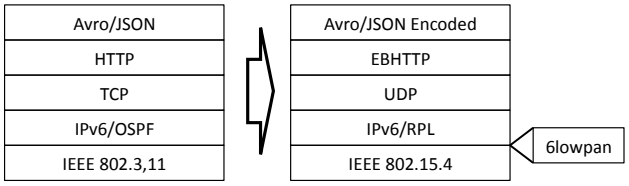


Figure 6. Layers in the protocol stack, and the protocols in use in the full version of sMAP, next to the version adapted for embedded devices.

A prominent emerging approach concerning protocol design for constrained devices hold that the best way of supporting these devices is to use traditional Internet protocols like TCP and IP as the canonical form, but to make adaptations where necessary to improve efficiency or robustness. We implemented a compact version of sMAP which runs on 802.15.4 links, using the stack shown in Figure 6. We have drawn from emerging IEEE and IETF standards for several layers of the stack. We draw on adaptations from standards bodies for the transport layer and below, and define our own Application and Object Exchange adaptations layers: EBHTTP and Binary JSON. Clients connecting over the Internet are not typically aware of the fact that adaptations are taking place, since they are performed by Edge Routers and HTTP proxies.

Field	Contents
DeliveryLocation†	URL where data is to be sent
DeliveryResource†	Local resource to be reported
Period†	Requested Interval between reports
MinPeriod	Minimum report interval
MaxPeriod	Maximum report interval
ExpireTime	When to stop sending reports
Capability	Permission to create

Table 4. Information necessary to cause a sMAP server to begin periodic or event-driven reporting. †required information.

3.5.1 Link, Network, and Transport

It has become readily apparent that it is possible to run IPv6 over links with very small MTUs [14]. To make this possible, the IPv6 headers are compressed to elide information which is common to the subnet, and reduces the size of an IPv6 header from 40 to 7 octets in the common case. Furthermore, work is currently underway to develop a routing protocol for constrained networks, called RPL.

For our prototype implementation of sMAP on constrained devices, we used draft versions of the `6lowpan` standards, and a predecessor to RPL known as HYDRO [25, 35]. These are used in the `blip` package which provides UDP and TCP interfaces on devices running TinyOS [21]; an evaluation of the full http/tcp/blip implementation is presented in Section 4.2.1.

3.5.2 EBHTTP

When reliable in-order delivery is not required, TCP is a poor choice for embedded devices, both because its 20-byte headers per segment are relatively large, but also because it is somewhat "chatty" – delivering a single message takes at least two round-trips. The simplest adaptation is to simply use UDP for transport. Within constrained networks, we can also run sMAP over Embedded Binary HTTP (EBHTTP). EBHTTP is a binary-formatted, space-efficient, stateless encoding of the standard HTTP protocol, intended for transport of small named data items, such as sensor readings, between resource-constrained nodes [36]. By using EBHTTP, we reduce the number of message bytes needed to transport sMAP data, while maintaining the URL structure and HTTP method semantics of sMAP.

EBHTTP was designed with several goals in mind:

1. run over UDP, to minimize transport overhead
2. be implementable on constrained devices with minimal code and RAM overhead
3. support unacknowledged delivery, for periodic reporting purposes

4. maintain HTTP semantics, enabling direct stateless transcoding
5. reduce HTTP protocol overhead to a minimal byte count

The basic EBHTTP header consists of 2 bytes specifying the method and a control field; all other data is carried as Type-Length-Value (TLV) encoded sections of the method. These may include a request URI, HTTP headers (either compressed or uncompressed), and body data. This encoding also allows multiple EBHTTP messages to be packed into a single UDP datagram or TCP segment. Packing multiple EBHTTP messages into a UDP datagram further reduces transport overhead, while running EBHTTP over TCP adds reliable delivery beyond the simple stop-and-wait semantics supported by EBHTTP over UDP.

Within the overall sMAP architecture, a less-constrained host can run a EBHTTP transcoding proxy. This proxy translates between EBHTTP requests and full HTTP requests, forwards those requests to the originally intended destination, and then translates the HTTP responses back into EBHTTP responses. This transcoding proxy need not be at the edge of the constrained network, because EBHTTP can run over unconstrained networks as well as constrained ones. When communicating with the nodes within the compressed network, clients use the reverse proxy, making HTTP requests for the sMAP resources, which are then translated into EBHTTP requests and sent to the node itself.

3.5.3 Packed JSON

JSON documents are more compact then XML documents; however, they are still relatively large for environments with very small link MTUs. Therefore, we defined a stateless compression format for JSON which takes advantage of the constraints placed on the use of JSON within sMAP, based on the Apache Avro project's binary encoding. When given a JSON instance document and the associated schema, the compressor produces a packed binary output much smaller then the input document.

```
{ "type" : "record",
  "name" : "formatting",
  "fields" : [
    {"name" : "Multiplier", "type" : "long"},
    {"name" : "Divisor", "type" : "long"},
    {"name" : "UnitofTime", "type" : "TimeUnit"},
    {"name" : "UnitofMeasure", "type" : ["MeasurementUnit", "string"]},
    {"name" : "SensorType", "type" : ["MeasurementType", "string"]}
  ]
}
```

Figure 7. The schema for the `formatting` resource.

Since all JSON documents used in sMAP have both a reference to their schema and schema version, the first five octets of the binary coding contain a four-octet hash of the schema name, and a single octet of version identifier. Following this is the actual data. Most of the compression is achieved by eliding all the strings appearing in the schema and replacing them with indices. Integers are packed using a variable length "zig-zag" coding. Figure 8 gives an example of this compression process.

To complete the web-services definition, we also specify an new HTTP header, `X-Avro-Schema` to allow a server

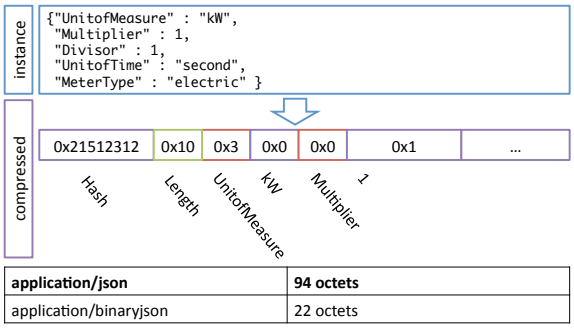


Figure 8. An example of statelessly compressing a JSON `formatting` object.

to specify the name of a schema the response body is encoded with; this must be present when the content type is `application/x-avro`. Just like with EBHTTP, we build a stateless HTTP proxy which can silently convert between packed and unpacked representations of JSON documents. However, nothing in the architecture constrains this proxy to run on the same network as the instrument, and it is general, capable of being an intermediary for any type of sensor.

4 Implementation and Evaluation

The three major design spaces which this work touches are *metrology*, *syndication*, and *scalability*. In order to evaluate the success of our design, we use these to provide a set of questions we can use to evaluate sMAP.

Metrology → Completeness and Generality. To show that our design for metrology is simple yet effective, we show that a large variety of common data sources from electric, environmental, process control, and meteorological systems can be presented within the system. We also examine a set of deployed systems from the literature.

Syndication → Application Use. sMAP is being used as the data plane of a host of higher-level applications. By having a common interchange format, these applications are "portable" from one collection of sensors to another.

Scalability → Practical Implementations. We show that sMAP can be practically implemented on devices ranging from powerful web farms to minuscule embedded devices without losing the essential benefits of the approach.

4.1 Complete and General

The first question we examine is how general sMAP is: what is the diversity of data it can represent. sMAP was developed during a process to create a *building-to-grid* testbed, for experimenting with novel interfaces and feedback mechanisms between a commercial building and the electric grid. The building in question is Cory Hall, the Electrical Engineering building at Berkeley. First commissioned in 1955, Cory hall consumes approximately 1MW of electricity in addition to steam and chilled water used for heating and cooling. Our testbed construction commenced with the instrumentation of several hundred sense points with thousands of channels, capturing much of the energy spend in addition to

environmental characteristics. Many of the feeds of sMAP data in Table 5 were developed as part of this project.

4.1.1 Building AC Power

Commercial buildings typically distribute power in a tree from a few feeds into the building substation, which are split into multiple three-phase branches and finally broken down to single-phase circuits at panels. We have instrumented each of these levels in our Electrical Engineering building, and made the data available via sMAP. Figure 9 outlines the hierarchical nature of electricity distribution and locates the various meters used to instrument the building.

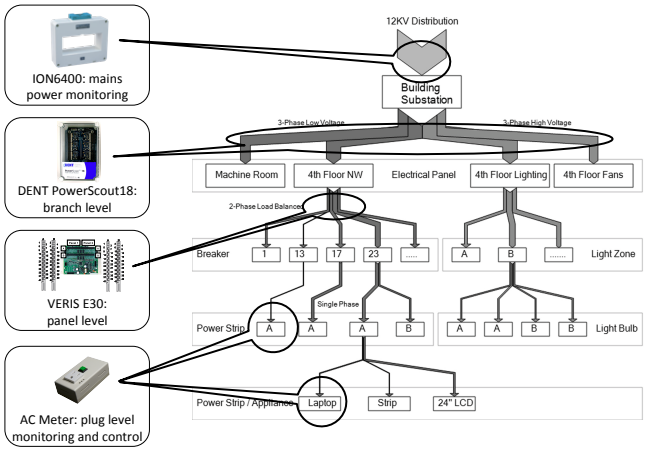


Figure 9. A Sankey diagram of the electrical distribution within a typical building, with monitoring solutions for each level broken out. All of these sources are presented as sMAP feeds.

At the top two levels, the substation and branch level, electricity is distributed in three phases through several transformers where it is stepped down from the the 12.5kV feed into the building. The instrumentation at these points consists of a large number of Current Transformers (CTs) and Rogowski Coils, which are present on the feed into the building and on individual phases of each branch. The building in question has two primary feeds from the grid, which are then split into 12 branches; this translates into 42 separate phase measurements (three per branch).

To distribute this data using sMAP, each branch or feed presents a sMAP interface to the world. Since each of these branches contains three phases, each sMAP instance presents several measurement points corresponding to each phase. In addition to these single-phase measurements, there are several "virtual" measurement points which corresponds to measurements from different combinations of CTs like phase-to-phase measurements and total system data. Using the flexibility of sMAP to name measurement points by any valid URL resource, each branch or feed exports a total of seven measurement points: A, B, C, ABC, AB, BC, and AC.

Each of these points also contains multiple sensors and meters: for branch-level meters, there are seven sensors and three meters. These correspond to measurements like real, reactive, and apparent power, current, phase-to-phase voltages, power factor, and several other quantities; some example data is shown in Figure 6.

Modality	Channel
meter	true energy
meter	reactive energy
meter	apparent energy
sensor	true power
sensor	reactive power
sensor	apparent power
sensor	current
sensor	displacement power factor
sensor	apparent power factor
sensor	line frequency

Table 6. Channels for each phase and total system measurement on a three-phase electric meter.

Once electricity has been distributed through a branch, it is further stepped down to 120V for circuit-level distribution. These circuits split off from wiring panels located on each floor. To instrument this level of the distribution hierarchy, we used a meter with 40 single-phase meters which is designed to be installed inside a breaker box. To map this arrangement onto sMAP, each circuit is treated as a single measurement point with several channels. Since this meter is much simpler, it only provides per-circuit energy consumption information (kWh).

The meters in use are typical of modern electrical monitoring: they provide a Modbus interface running over RS485. In order to make them available over the Internet using sMAP, we use a Modbus – Ethernet adaptor to convert bridge to an IP subnet, and then run a gateway service on a server which periodically polls the devices and caches their last reading for use in sMAP. Since each manufacturer typically has their own map of Modbus registers, the gateway must be customized for each new brand of meter; of course, this effort is all transparent to clients who receive normally-formatted sMAP data.

4.1.2 Plug-load Power

The final level of electrical distribution is plug-level, where individual appliances are connected to the circuit. To monitor at this resolution, we used a custom device called the ACme. ACme's [15, 16] are single-phase plug-load meters that measure power and energy of typical AC appliances in households and offices. In addition, they are capable of controlling connected devices using an internal relay. ACme is a typical mote-class device based on a msp430 microcontroller and a cc2420 802.15.4 radio, shown in Figure 11. ACme is representative of a large class of sensors and meters found in commercial environments that measure physical phenomenons at a single *point* in some process. Examples of devices in this class include flow meters, temperature sensors, light sensors, and motion sensors.

ACmes use *blip*, our open-source IPv6 stack to form an ad-hoc network. Since blip supports both TCP and UDP, there are multiple ways a protocol like sMAP can be scaled down to this device. We compare the options for this in Section 4.2.

From a metrological point of view, a single ACme is a device with a single measurement *point* – the plug – and multiple *modalities* – it senses power, meters energy, and actuates a relay. The actuation present on an ACme is particularly simple: a relay can switch the attached device. Since this

Name	Sensor Type	Physical Layer	Sense Points	Channels
Cory Hall Submetering	Dent 3-Phase	Modbus/Ethernet	40	1600
Cory Hall Building Power	ION and PQube	HTTP/Ethernet	3	150
Cory Lab Temperature	TelosB [28]	802.15.4 + Ethernet	4	8
Cory Lab Machines	ACme [15]	802.15.4 + Ethernet	8	16
Cory Chilled Water	HeatX Meter	Modbus/Ethernet	1	11
Cory Roof Environmental	Hydrowatch Node [34]	802.15.4 + Ethernet	4	36
Soda Sun Blackbox	Fan Speed; Environmental	HTTP/Ethernet	10	84
Soda Lab Machines	ACme	802.15.4 + Ethernet	40	80
Soda Lab Panel	Veris E30 Meter	Modbus/Ethernet	1	42
LBNL Building 90	ACme	802.15.4 + Ethernet	70	140
Berkeley Weather	`wunderground` and Viasala WXT520	HTTP + Serial	2	20

Table 5. Deployments with data available with sMAP

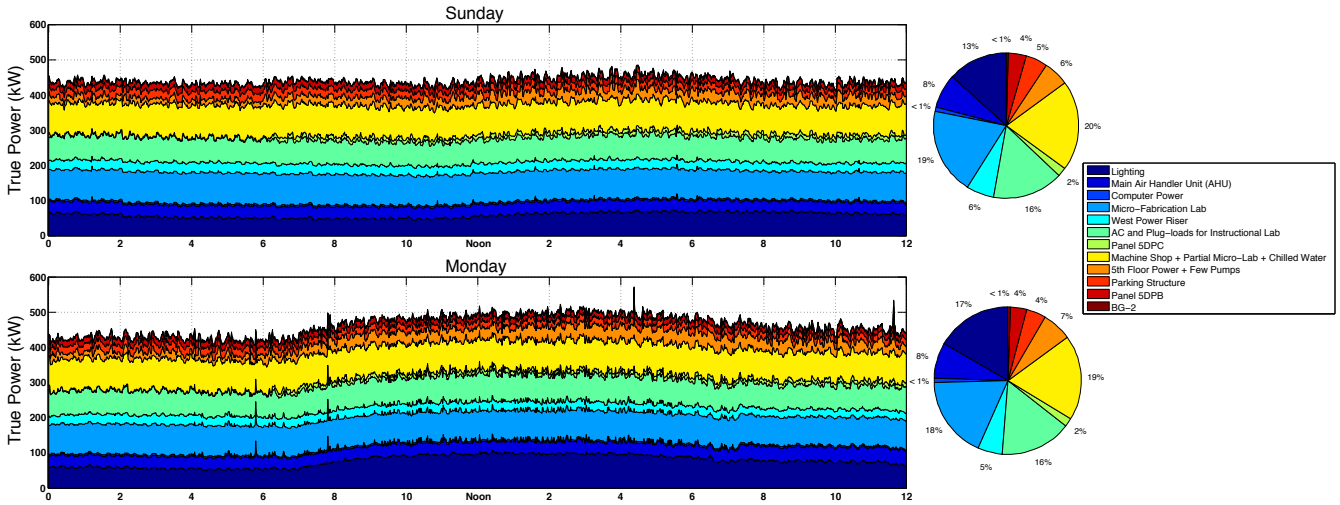


Figure 10. A detailed breakdown of electrical usage inside the Electrical Engineering building over two days. Data is pushed to a client database by a sMAP gateway connected to three Dent circuit meters, each with six channels. Power is used primarily by lighting, HVAC, and a micro-fabrication lab, as expected. Interestingly, the total power consumed on Sunday is $440kW$ while on Monday is $462kW$, an increase of less than 5% between a weekend and a weekday, indicative of an inefficient building. The difference between day and night is small as well. The only load with an obvious spike in power is lighting at around 7am on Monday, whereas most loads stay the same throughout the day and night.

fits into the library of sMAP actuators as a binary actuator, nothing new needed to be developed to enable this form of control.

4.1.3 *External Data: CA ISO and WUnderground*

sMAP can integrate existing, external data sources to help define the context in which our study building operates. Two forms of data relevant to our work were data from the California Independent System Operator (Cal ISO) concerning the total state-wide electricity demand, and weather data from various locations.

Weather data typically has multiple sensors and meters, instrumenting precipitation, wind, temperature, and pressure. These are easily translated to the appropriate abstractions as sMAP modalities; furthermore sMAP preserves some important metadata about the weather meter such as its station identifier, and make and model of the physical hardware. Numerous mote-class sensors would typically be incorporated to monitor interior environments.

Because sMAP is relatively prescriptive – the hierarchy and object definition does not leave much freedom – the amount of non-shared code is very small; around 100 lines of Python each case. In fact, the amount of non-boilerplate code is even smaller; integrating the weather feed required only 43 new lines of code.

4.1.4 *Other Deployments*

It is in some ways unfair to evaluate sMAP only based on the particular data *we* thought were important. To cast a wider net, we examined several years of SenSys and IPSN proceedings for papers describing the deployment of whole systems, or the development of new sensors; this listing is present in Table 7.

We found that although the *system design* typically addressed novel issues, the actual data retrieved tended to be very simple: typically, slow time-series of a few parameters. Since mote-class devices were the most common, the measurement point/channel abstraction appears to work well (as for the ACme). In fact, none of the deployments appeared to have a more complicated data model then a three-phase electric meter.

An important exception to this paradigm was best demonstrated by Lance (volcano monitoring) and the Heritage Building project: sMAP does not address high-frequency data like seismometer or accelerometer data. Although these are in principle time-series of sensor data, it appears to us

Deployment	Modality	Sensed Quantities
RACNet [22]	Datacenter	temperature
ACme [15]	Electricity	true power, apparent power
Lance [38]	Geophysical/Volcano	seismometer
MEDiSN [19]	Healthcare	ECG, blood oxygenation level, pulse rate, *etc*
GreenOrbs [24]	Environmental/Trees	illuminance
Flood Warning [6]	Water	river level
NAWMS [18]	Water	flow rate
PermaDAQ [7]	Alpine environmental	temperature, conductivity, crack motion, ice stress, water pressure
Heritage Buildings [9]	Structural health	temperature, deformation, acceleration
HydroWatch [34]	Water cycle	temperature, light, humidity

Table 7. Deployments from SenSys and IPSN in the past several years.

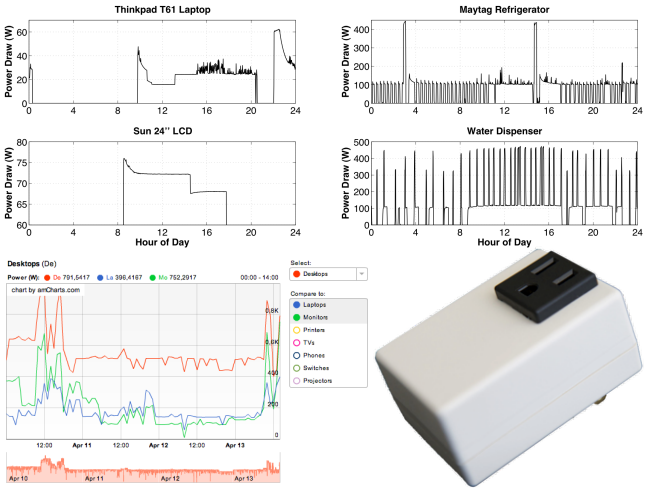


Figure 11. ACme wireless plug-load meters (bottom right) are used to monitor power consumption of various appliances, including a laptop and a LCD display in the top left, a refrigerator and a water dispenser in the top right, and aggregate consumptions in bottom left.

that sMAP is a sub-optimal design point for representing data from these applications; this is not a surprise since it was not an original design goal. We may need to address this with future protocol modifications. Another observation is that none of these deployments involved a significant actuation component. While there have been deployments involving actuation reported in the literature, this has not been a dominant research focus. It is, of course, common in building environmental conditioning system.

4.2 Scalable

sMAP must be scalable in both directions: "up", to millions of connected clients or subscribers, and "down" to tiny embedded devices.

Scaling sMAP up is, if not simple, a well-understood problem given that it runs over HTTP. Using the HTTP caching model, sensors can and do define how long their readings will be valid for using the "Expires" header – presumably, until the next sample is taken. Intermediate caching proxies then offload queries onto multiple servers. The syndication design we have built is also scalable, since a single sMAP source may be republished other places on the Internet. For instance, a sMAP instance running on embedded device may communicate only with a caching proxy which also offloads most syndication duties. This is very much the model which **pubsubhubbub** uses [13] for scaling RSS and Atom feeds.

A more challenging test is whether sMAP can scale down to fit on embedded devices. While it is possible to expose embedded devices' functionality via web services by using an application-layer gateway, this is a limiting architecture because it requires the gateway to be aware of the functionality of each connected device. In the sMAP architecture, a gateway or proxy may still be present for caching, but it is a transparent component between the client and the server.

4.2.1 Embedded Implementation

We have implemented all the components of the stack presented in Figure 6; it is used to make sMAP data from AC plug load meters available. As a comparison, we also have an implementation of sMAP which uses Packed JSON on top of HTTP and TCP (rather then EBHTTP).

Component	Size
Application Code	936
HTTP	542
TCP	3534
Routing (HYDRO)	2890
IP + ICMP	4382
6lowpan	2262
Link	4926
Total	19472

Table 8. Code size of key elements of an sMAP implementation using Binary JSON, HTTP, and TCP.

When sMAP meters and sensors are connected to a bandwidth-constrained network, such as 6lowpan/IEEE802.15.4, we need to scale sMAP "down" by minimizing the number of bytes transmitted over the network. Both EBHTTP and packed JSON help to reduce the cost of a sMAP request. To evaluate the effect of these adaptation layers, we compare the size of a sMAP request using HTTP and JSON with the same sMAP request using EBHTTP and packed JSON. The request is a GET of the `/data/ABC/sensor/true_power/reading` resource, using `curl` for HTTP/JSON and our EBHTTP client for EBHTTP/Packed JSON. Table 9 shows the results of the comparison.

With EBHTTP and Packed JSON, the overall request is reduced to 10% of its uncompressed size. This reduction comes from:

1. removing the TCP handshaking and header overhead by using UDP

	HTTP (octets)	EBHTTP (octets)
Request		
TCP/UDP	190	8
HTTP/EBHTTP	152	4
URL	35	35
Response		
TCP/UDP	192	8
HTTP/EBHTTP	123	4
JSON/Packed JSON	167	25
Total	859	84

Table 9. Comparison of HTTP and EBHTTP request sizes

2. converting the ASCII HTTP request and response headers into the EBHTTP binary format
3. removing the unneeded HTTP headers sent by the client
4. converting the ASCII tags from the JSON response into short binary tags

In addition to the message size reduction, we reduce the size of the embedded code needed to implement this service by replacing the TCP stack with a UDP stack, replacing the text-based HTTP parser with a small EBHTTP parser, and replacing the text-based JSON encoder with a simpler packed JSON encoder.

This compression is not without cost. By switching to UDP, we lose the segmentation and flow control provided by TCP, and switch to a simpler stop-and-wait reliability model. But, this is often an acceptable tradeoff when using sMAP to communicate with constrained devices.

4.3 sMAP Applications and Syndication

sMAP decouples device details from applications that consume their data. Applications only consume sMAP data and interact with sMAP sources through HTTP/JSON. We have constructed various applications that consume and process sMAP data. There is a visualization client, a storage and data processing service, a virtual building application, and a mobile phone application for personal energy footprint tracking. We have also integrated sMAP with Google PowerMeter.

Because sMAP externalizes metadata management, applications must handle the metadata associated with stream sources. Metadata management is application-specific and is handled as such. For instance, our stream storage system treats the metadata as another timeseries data stream and manages the binding between the metadata stream and its associated data stream. The virtual building application stores metadata on building entities and objects, and constructs graphical relationships between them.

4.3.1 Visualization

We have built two visualization clients for sMAP. The first provides a front-end for interacting with sMAP data sources, called the sMAP console shown in Figure 12. The application crawls the sMAP resources and uses the information returned to generate an interface for any sMAP source. Using the console, users can type in any source URL, and easily view what sensors, meters, and actuators are available; this gives users an easy way to explore the functionality of a new instrument. Furthermore, if the sMAP channel supports

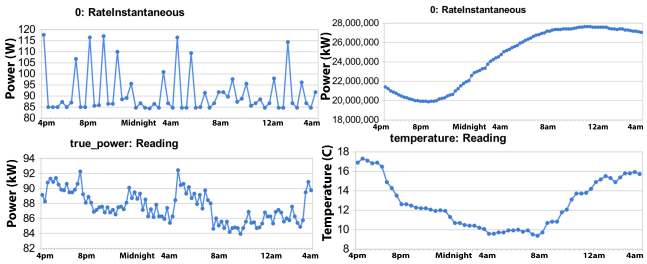


Figure 12. The sMAP console application can be used to plot any sMAP-compliant data sources, such as: power consumption of a desktop computer (top-left); portion of the California energy usage for a day from CalISO (top-right); lighting load monitored by a Dent circuit meter (bottom-left); and the local temperature (bottom-right).

the `profile` resource, the application plots whatever data is available.

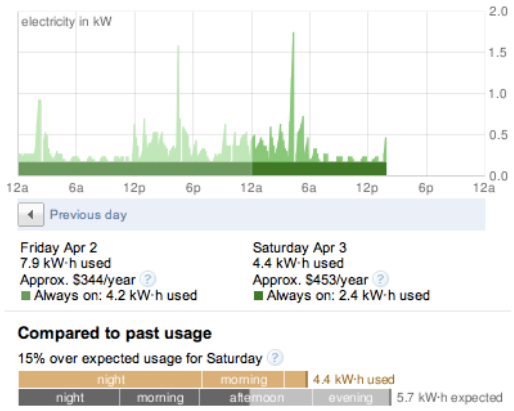


Figure 13. A bridge sends sMAP data to Google PowerMeter. This iGoogle widget shows the real time power consumption of a refrigerator.

The second visualization client feeds sMAP data to Google PowerMeter[1] to display the trends over time, as shown in Figure 13. Google PowerMeter accepts only energy data, and in our case a client daemon periodically republishes sMAP data. The data model used in PowerMeter allows only a flat namespace of sensors which are reported very infrequently; it is designed for publishing data like whole-building consumption to users' profiles. In the future we hope there will be a standard which allows parties to avoid additional protocol translation engines.

4.3.2 Storage

Our metadata management service provides archival storage and query facilities on sMAP data. The Integrated Sensor-Stream Storage System (IS4) is a publish-subscribe storage system with a historical and real-time query engine. Although IS4 is a generic data store it has specific mechanisms for importing sMAP data sources.

Publishers push data into IS4 and IS4 pushes data to subscribers. However, unlike the very simple sMAP profile, IS4 is built to compute various queries and aggregates on the data coming in from sMAP and republish the results of these queries as new streams; in this sense it is first and foremost

an ambitious attempt to build a distributed data-store for all of our data. IS4 organizes input streams using different resource types and defines rules for their hierarchical organization. For instance, an IS4 "*device*" resource is a collection of publishers, which map directly to a sMAP "measurement point."

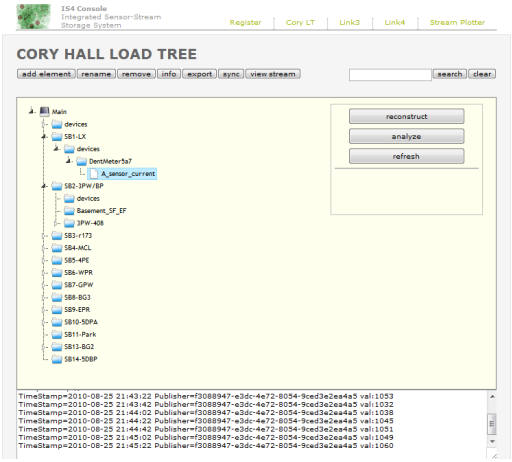


Figure 14. The Integrated Sensor-Stream System (IS4) manages the metadata for our Cory Hall instrumentation project: it tracks the associations between sensors and what they are sensing. It also allows archiving data coming from sMAP. This example uses IS4 to build up a hierarchical representation of the Cory Hall electric tree.

Currently, our focus in IS4 is on providing a way to manage data and metadata streams in buildings. IS4 captures relationships by constructing a resource hierarchy that names measurement instruments relative to their context; that is, their physical location and the location of the measured quantity. This allows us to represent the relationships between systems, spaces, and instruments. IS4 keeps track of changes in these relationships over time; logical changes that reflect physical changes, such as the installation of a new sensor. Figure 14 shows a portion of the electric load tree captured in IS4, branching from whole-building power down to individual panels and receptacles.

4.3.3 EveryBuilding

Another application which takes a different perspective on sensor metadata management is EveryBuilding. EveryBuilding provides a space for users to create virtual representations of the buildings they live and work in. Users incrementally create building models made up of interconnecting spaces (typically rooms), graph relationships between these spaces, and associate sensors with data streams associated with the spaces. EveryBuilding makes use of sMAP as a source of live physical data streams.

Users of EveryBuilding can connect their sensors to live data streams from sMAP by entering the URL into a web formAfter the user enters a new sMAP URL, EveryBuilding takes advantage of the self-describing nature of the design by taking the following steps:

1. GET the sMAP `/profile` resource for the channel and initialize the sensor's data stream with the most recent readings available in the channel
2. GET the sMAP `/formatting` resource for the channel, and store the unit of measure and scaling coefficient along with the sensor's sMAP subscription record
3. GET the sMAP `/parameter` resource for the channel, and store the SamplingPeriod element as an indicator of how often EveryBuilding should expect a new reading
4. POST a subscription request to the sMAP root's `/reporting/create` URL to subscribe itself to future data items on that channel. The subscription's DeliveryLocation is the sensor's own reporting URL within EveryBuilding, which accepts HTTP POST requests containing sMAP `/reading` resources, applies the scaling coefficients, and stores the new readings into the sensor's data stream

EveryBuilding supports other data access methods for sensor readings, including CSV upload and manual data entry, but sMAP subscription has proven to be a useful tool for obtaining live physical data streams within a web application context. Using archived data, EveryBuilding is able to feed the models into simulation tools like Energy+ and compare the predicted energy spend to real data.

4.3.4 Personal Energy Footprint

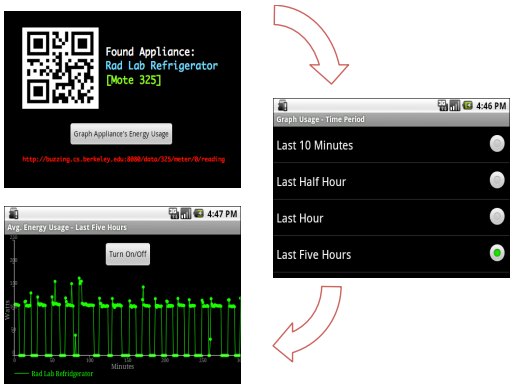


Figure 15. Personalized energy feedback for Android devices.

In an attempt to increase energy-consumption awareness, we built a mobile-phone application placing energy data from sMAP-enabled sensors on mobile phones 15. As a user moves from room to room, they scan a 2D bar code identifying the room and the sMAP appliances in that room. The application obtains data feeds from the appliances in that room that are public or that belong the user, and presents aggregates or plots. A key concept is linking the physical space and social concept of personal responsibility with live data. Since sMAP also provides a way to express a control interface, controllable appliances also appear available for direct interaction.

5 Conclusions

sMAP is a simple and compact version of what is typically found in web-service protocols, and has the potential to subsume much of what is currently being advocated in standards bodies. In the course of finalizing the design, we worked through many design facets to develop a solution

which fills a niche in the space. By providing only a durable instrument identifier, we make it possible to build other systems which manage the context of the instrument, creating a clear separation of concerns between sMAP, which represents data, and other systems, which track metadata. We also investigated how sMAP can be run efficiently on constrained devices; the solution, of using separate adaptations for each layer is both principled, since it preserves the original architecture, and efficient, reducing message size and complexity. In short, this concrete solution is strong enough to stand on its own as an interface for simple sensors and meters.

sMAP pushes a broader architectural transition taking place in sensor network construction, away from monolithic application stacks and towards distributed, service-based architectures. Using this simple primitive, we are able to integrate data from a huge diversity of sources without restriction to any underlying technology, and build applications quickly and independently.

Acknowledgements

We would like to thanks our shepherd, Suman Nath, and the anonymous reviewers for helping to improve the paper. Scott McNally was instrumental in managing the Cory Hall instrumentation project, ably assisted by Albert Goto and Mike Howard. This work is supported in part by the National Science Foundation under grants CPS-0932209 and CPS-0931843.

6 References

[1] Google powermeter. http://www.google.org/powermeter/.

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

[3] Smart grid interoperability standards project. http://www.nist.gov/smartgrid/.

[4] End user guide to industry foundation classes, enabling interoperability. Technical report, 1996.

[5] Ansi c12.19: Meter tables. Standard, 2004.

[6] E. A. Basha, S. Ravela, and D. Rus. Model-based monitoring for early warning flood detection. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 295–308, New York, NY, USA, 2008. ACM.

[7] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. Permadaq: A scientific instrument for precision sensing and data recovery in environmental extremes. In *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 265–276, Washington, DC, USA, 2009. IEEE Computer Society.

[8] M. Botts and A. Robin. Opengis sensor model language (sensorml) implementation specification. OpenGIS Implementation Specification OGC 07-000, Open Geospatial Consortium Inc., 07 2007. Version: 1.0.0.

[9] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The torre aquila deployment. In *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 277–288, Washington, DC, USA, 2009. IEEE Computer Society.

[10] D. Crockford. Rfc 4627 - the application/json media type for javascript object notation (json). Technical report.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616 – hypertext transfer protocol – http/1.1. Technical report.

[12] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[13] B. Fitzpatrick, B. Slatkin, and M. Atkins. Pubsubhubbub core 0.3 – working draft. Technical report.

[14] J. W. Hui and D. E. Culler. Ip is dead, long live ip for wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, New York, NY, USA, 2008. ACM.

[15] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. Design and implementation of a high-fidelity ac metering network. In *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 253–264, Washington, DC, USA, 2009. IEEE Computer Society.

[16] X. Jiang, M. V. Ly, J. Taneja, P. Dutta, and D. Culler. Experiences with a high-fidelity wireless building energy auditing network. In *SenSys '09: Proceedings of the 7th ACM conference on Embedded network sensor systems*, 2009.

[17] A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.

[18] Y. Kim, T. Schmid, Z. M. Charbiwala, J. Friedman, and M. B. Srivastava. Nawms: nonintrusive autonomous water monitoring system. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 309–322, New York, NY, USA, 2008. ACM.

[19] J. Ko, R. Musăloiu-Elefteri, J. H. Lim, Y. Chen, A. Terzis, T. Gao, W. Destler, and L. Selavo. Medisn: medical emergency detection in sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 361–362, New York, NY, USA, 2008. ACM.

[20] C. Leung. Extended environments markup language. http://www.eeml.org/.

[21] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[22] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. Racnet: a high-fidelity data center sensing network. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 15–28, New York, NY, USA, 2009. ACM.

[23] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim. Tinyrest - a protocol for integrating sensor networks into the internet. In *REALWSN '05: Workshop on Real-World Wireless Sensor Networks*, 2005.

[24] L. Mo, Y. He, Y. Liu, J. Zhao, S. J. Tang, X. Y. Li, and G. Dai. Canopy closure estimates with greenorbs: sustainable sensing in the forest. In *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 99–112, New York, NY, USA, 2009. ACM.

[25] G. Montenegro, N. Kushalnagar, J. Hui, , and D. Culler. Rfc 4944 – transmission of ipv6 packets over ieee 802.15.4 networks. Technical report.

[26] M. Nottingham and R. Sayre. Rfc 4287 – the atom syndication format. Technical report.

[27] Å. Östmark, J. Eliasson, P. Lindgren, A. van Halteren, and L. Meppelink. An infrastructure for service oriented sensor networks. *Journal of Communications*, 1(5):20–29, 2006.

[28] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 48, Piscataway, NJ, USA, 2005. IEEE Press.

[29] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 253–266, New York, NY, USA, 2008. ACM.

[30] A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, and L. Soibelman. Demo abstract: The sensor andrew infrastructure for large-scale campus-wide sensing and actuation. In *IPSN*, pages 415–416. ACM, 2009.

[31] P. Saint-Andre. Rfc 3920 – extensible messaging and presence protocol. Technical report, October 2004.

[32] L. Schor, P. Sommer, and R. Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *BuildSys '09: Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 31–36, New York, NY, USA, 2009. ACM.

[33] *Smart Energy Profile 2.0*. Zigbee Alliance, 2010.

[34] J. Taneja, J. Jeong, and D. Culler. Design, modeling, and capacity planning for micro-solar power sensor networks. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 407–418, Washington, DC, USA, 2008. IEEE Computer Society.

[35] A. Tavakoli. *Exploring a Centralized/Distributed Hybrid Routing Protocol for Low Power Wireless Networks and Large Scale Datacenters*. PhD thesis, EECS Department, University of California, Berkeley, Nov 2009.

[36] G. Tolle. Embedded Binary HTTP (EBHTTP). IETF Internet-Draft draft-tolle-core-ebhttp-00, Mar. 2010.

[37] S. Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10:87–89, 2006.

[38] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 169–182, New York, NY, USA, 2008. ACM.

[39] D. Yazar and A. Dunkels. Efficient Application Integration in IP-based Sensor Networks. In *Proceedings of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, Berkeley, CA, USA, Nov. 2009.