Periodically Updated Variables: Wide-Area Publish-Subscribe Middleware Supporting Electric Power Monitoring and Control*

Dave Bakken
School of EECS
Washington State University
Pullman, USA
bakken@eecs.wsu.edu

Carl Hauser
School of EECS
Washington State University
Pullman, USA
chauser@wsu.edu

Harald Gjermundrød
Dept. of Computer Science
University of Nicosia
Cyprus
harald@unic.ac.cy

Abstract

GridStat is a new kind of publish-subscribe data delivery service designed specifically, from the bottom up, for delivering real-time operational data of the electric power grid. We overview the data delivery needs of the electric power grid and analyze why neither the Internet and its existing technologies nor existing publish-subscribe products and research meet these needs. We introduce periodically updated variable and periodic update flow abstractions and describe how GridStat, as periodic update flow middleware implements these abstractions with very low latency and high availability.

1 Introduction

The communication systems for the electric power grids in North America were developed in response to the 1965 North-eastern US blackout. They are piecemeal combinations of largely antiquated technology and include virtually no exploitation of advances from the last few decades of research in distributed computing. As a result, the visibility that grid operators have into

real-time operations, their *situational awareness*, is poor [10]. This has been a contributing cause of the major 2003 blackouts in the US [23] and across the Italian-Swiss connection, as well as with virtually all major blackouts in recent decades. This is because these limited real-time communications capabilities in turn limit the kinds of protection and control that can be done [10, 22]. More on the problems with today's grid communication can be found in [5].

We have designed, implemented, and experimentally evaluated a new kind of middleware framework that provides the abstraction of a *periodically updated variable (PUV)* in order to support better situational awareness for the electric power grid. A PUV has two parts: a source variable and multiple remote caches on hosts where power grid applications run. Applications retrieve PUV values from their local cache which contains the latest update of that variable that the particular cache has received. PUV caches receive updates with a tunable, per-instance quality of service (QoS) specified by remote applications in terms of both freshness (maximum staleness) and availability. PUVs were designed to support the emerging needs of the electric power grid, which can exploit very low latencies for new protection and control schemes as well as for improved situational awareness [10]. Our research shows that PUV updating can be done with very low latency and high scalability because, unlike previous research into distributed shared variables, their application domain requires virtually no consistency across different variables or even across different caches of the same variable.

The PUV abstraction is implemented by the *periodic update flow (PUF)* abstraction, which is the QoS-managed flow of updates into a particular PUV cache instance. PUFs are implemented in managed publish-subscribe middleware which manages QoS in terms of maximum delivery latency, minimum delivery rate, and minimum number of redundant and disjoint paths for each PUF.

PUF middleware (PUFM), implementing PUVs and PUFS, is a further specialization of a kind of publish-subscribe middleware called *periodic push* [14] in which publishers emit events at periodic intervals. PUFs specialize the periodic push paradigm by exploiting the semantics of a series of PUV updates. Specifically, in other publish-subscribe systems, an arbitrary event in the system cannot, in general, be dropped en route because it cannot be ascertained how this may affect the subscribing applications. However, because a PUF represents the QoS needed for a given subscription, some updates can be dropped en route so long as QoS requirements of downstream subscribers are met. We call this *rate filtering*, and it directly enables PUFs to efficiently provide different freshness guarantees to different caches of a PUV while enhancing multicast efficiency (avoiding unnecessary traffic).

PUVs and PUFs are novel. The work most closely related to PUVs is PASS [26]. However, PASS only implemented Boolean variables, did not have per-subscriber QoS, and was not optimized for low latency and high bandwidth (indeed, it was designed for low-bandwidth military networks). We are aware of no periodic push middleware that captures the semantics of PUFs, specifically rate filtering, which is required to provide per-replica QoS to PUVs. The closest publish-subscribe middleware to PUFs is that tailored for wireless sensor networks (WSNs). However, the detailed paradigms, delivery mechanisms, overall design goals (*e.g.*, conserving energy) and environment for WSNs are quite different from the high-speed requirements of the electric power grid and the much richer computational resources that can support them. Mechanisms developed for WSNs thus are not generally appropriate for low latency, high throughput PUFs.

GridStat is a managed middleware framework we have been developing, in close collaboration

with electric power researchers, in order to meet the emerging and future needs for data delivery services in electric power grids [2, 4, 6, 3, 10, 22]. Indeed, PUVs and PUFs were conceived from bottom up analysis combining a detailed evaluation of the needs of the power grid identifying opportunities to leverage advances in a number of distributed computing areas.

The research contributions of this paper are:

- An analysis of the ways in which the electric power grid's emerging information infrastructure differs from the general Internet at large; and why this suggests the need for new data delivery services and protocols that exploit these differences.
- A new kind of distributed shared memory abstraction, PUVs, derived from this analysis of electric power grids' needs.
- PUF middleware, a new kind of periodic push publish-subscribe middleware that enables the implementation of PUVs in a fast and scalable way. PUVs and PUFs, when combined with the more controllable and predictable nature of power grid communication needs, enable a number of optimizations which are not otherwise possible.
- An overview of the design, architecture, and performance of GridStat, an implementation of PUF middleware.

The remainder of this paper is organized as follows. Section 2 provides more context in the form of power grid data communication requirements. It then introduces and defines PUVs and PUFs, as well as a publish-subscribe system model that supports them. Section 3 contrasts the emerging data delivery infrastructure for the electric power grid with the general-purpose Internet. It shows how the emerging data delivery network for the electric power grid must be more stable and managed than the Internet at large, and how these properties can be exploited by new mechanisms. Section 4 provides an overview of our GridStat middleware: its design decisions enabled by the more controlled and resource rich environment; its performance and scalability; and advanced mechanisms enabled by the semantics of PUVs and PUFs. Section 5 overviews related work and section 6 concludes.

2 Context of Power Grid Data Communication

In order to provide a data delivery service tailored to the needs of the electric power grid all requirements that the data is subject to must be specified. This section lists QoS and flexibility requirements for these data flows and then describes a system model that realizes them.

2.1 Data Delivery Requirements for the Electric Power Grid

We now overview the requirements for data delivery in the electric power grid, for today's applications and for future applications envisioned by electric power researchers, recent IEEE standards, *etc.* For the sake of brevity, we omit most references to such electric power sources—they can be found in [5]. The requirements outlined below do not include cyber-security and trust

management issues; for an overview of them, see [5] and the sources it cites.[1]

Quality of Service (QoS) requirements for data delivery include the following:

- **Delivery latency**: 4 msec end-to-end for some flows within substations, and 8–12 msec for flows external to substations, with greater latencies, up to several seconds, allowed for other flows. Emerging wide-area protection schemes can utilize low latencies (which of course include link latencies that are limited by the speed of light) to perform protective actions across hundreds of miles in one or two power cycles (i.e., approximately 16–32 msec in the United States and 20–40 msec in Europe).

- **Delivery rate**: Must support as high as 60 updates per second and as low as once per 30 days. In the future, 250 or more updates per second will be required.

- **Data availability**: must support as high as "Ultra" availability (99.9999%) to "Medium" availability (99.0%), depending on the power application program.

Flexibility requirements for data delivery include the following:

- **Efficient multicast**: a given publication must be deliverable to multiple subscribers in different locations in an efficient manner: a given update in the flow should not be sent over any given network link more than once.

- **Heterogeneity of delivery QoS**: for reasons of network efficiency and stability, different subscriptions (i.e., subscribers to the same update flow) must be able to specify different QoS requirements. Supporting such heterogeneity is possible only because a PUF is not forwarding arbitrary events but is rather forwarding updates whose downstream QoS requirements (latency, rate, availability/redundancy) are known.

- **Temporal synchronism**: one fairly new kind of sensor in the power grid, called a synchrophasor, requires temporally synchronized QoS management. This is explained further in context in the discussion of "synchronization between flows" in section 3.4.

2.2 Periodically Updated Variables

The PUV abstraction allows the value of a sensor to be distributed to different power applications that are widely dispersed across an electric power grid. A given sensor emits updates at a known minimum rate, *min_rate*, where the units are *updates/second*. Each subscriber to the variable specifies a *freshness*, in seconds (typically a small fraction of a second), which indicates how long the application can wait for an update. The maximum period for the sensor, *max_period*, is of course defined as the inverse of the *min_rate*. Assuming that an update can be delivered with a maximum latency, *max_latency* then:

$$Freshness = max\_period + max\_latency \tag{1}$$

[1] We note that [5] served as the basis for many of the requirements for a recent Dept. of Energy solicitation to create a detailed requirements specification for a grid-wide data delivery service called NASPInet [19]; work on this solicitation has recently begun [20].

This freshness, as well as availability, is specified on the granularity of a cache instance; i.e., different subscribing applications can be given different values of freshness and availability. The mechanisms for meeting both latency and availability requirements are discussed below.

A PUV's value can be either a typical basic type or a compound type. As mentioned above, a PUV has a single publisher, so consistency is simpler to provide than in distributed shared systems that can have multiple writers. Such a single publisher is, of course, a single point of failure. However, extending these ideas to replicated sensors providing fault tolerance adds very little complexity and overhead to the data delivery infrastructure, even though the different sensors would provide slightly different measurements.

2.3 System Model

The system model to support PUVs with real-time updates across a wide-area network is illustrated in Figure 1. The data delivery system is separated into two distinct parts:

- The *data delivery plane*, which delivers real-time operational data, and
- The *management plane*, which accepts or rejects requests for data (subscriptions) based on security and performance constraints. It also calculates the paths by which the data will be delivered from a publisher to its subscribers.

A *publisher* is a piece of software attached to a sensor or other source of operational data. It registers one or more PUVs with the data management plane (described below) and indicates the minimum update rate for each. (This is often hard-coded in sensors in the power grid.) The publisher then begins to emit updates for each PUV variable. The updates for a given variable are delivered by the data delivery plane to one or more *subscribers*. A subscriber registers interest in a PUV and indicates its required freshness and availability (more on the availability below).

In Figure 1, publishers and subscribers are shown for clarity's sake on different sides of the data delivery plane; in practice, of course, they would be dispersed throughout. Additionally, we note that an *update* in a PUF is called an *event* in the more general publish-subscribe paradigm. We use this term specifically because updates to a given variable can be managed much more efficiently and quickly than a generic event in a publish-subscribe system.

The data delivery plane depicted in Figure 1 is also called the NASPInet Data Bus in an ongoing electric power effort [20]. As is true in virtually all publish-subscribe systems, the data delivery plane is subdivided into a graph of forwarding elements (FEs) that pass on an update to one or more outgoing links as dictated by the QoS requirements of the subscriptions. FEs are connected by links that provide a given amount of bandwidth as well as (stochastic or deterministic) bounds for minimum delay and drop rate. These links may be implemented directly on top of fiber or ATM, across a managed utility infrastructure using internet protocols and other off-the-shelf technology. Finally, publishers and subscribers are also connected to one or more FEs, though typically over a local area network connection with very good link properties.

Figure 1 depicts N publishers denoted Pub_1 through Pub_N. In this example, Pub_1 is the application that "owns" and updates the variable X in its programming language. The publisher turns the local variable X into PUV X, with the publisher at the root of its update tree and the cache instances of the PUV at its leaves. These cache instances reside in the subscribers, denoted

Sub_1 through Sub_M. Note that these multicast "trees" are really multi-trees, having multiple disjoint paths from each PUV publisher to each cache replica at its leaves. In the example, PUVs X and Y are updated by Pub_1 while Z is updated by Pub_N. Y is of course subscribed to by both Sub_1 and Sub_M, illustrating the multicast nature of PUVs. (The redundant paths are not depicted for the sake of clarity.)

The management plane exerts control over the data delivery plane to ensure that QoS requirements for all PUVs are met. As part of this task, it performs admission control: it takes in the QoS requirements for each subscription and creates multicast paths through the FEs to meet those requirements [11]. Subscriptions are created only if there are sufficient resources. All updates enter the PUV via a proxy object on the publisher side. The proxy performs traffic policing to ensure that the update rate the publisher registered (and the management plane accounted for) is not violated [18]. Additionally, an FE can discard packets not signed by a proxy so an intruder that gains physical access can only affect FEs to which it is directly connected. This offers some protection against denial of service attacks.

2.4 Periodic Update Flows

PUFs capture the semantics of a flow of updates to different cache instances of a PUV. PUF middleware (PUFM) is a managed middleware framework that implements multiple PUFs, *i.e.*, delivers updates to multiple PUVs, while meeting the QoS requirements for the PUVs it is supporting. PUFM thus comprises both the data delivery plane and the management plane from figure 1.

PUFs have much simpler semantics than general publish-subscribe systems in two key ways. First, there is no complex subscription matching: a subscriber simply asks for a (heirarchically-named[2]) variable. Second, the FEs along the update path for a given PUF do not store anything. Outgoing paths for each update are chosen using a simple computation. These factors enable PUFM to support much lower latency and much higher throughput (updates/second forwarded) than is possible with less specialized publish-subscribe middleware, as we describe in section 5. In

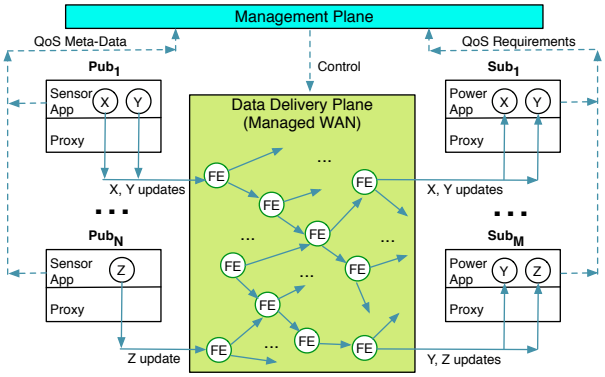

Figure 1. Power Grid Data Delivery System Architecture

section 4 below we describe GridStat, our implementation of PUFM. First, however, additional

[2]Naming is not discussed here due to space limitations.

details on the networking and application infrastructure of the electric power grid, provide context for the design decisions made in the GridStat PUF middleware.

3 Comparison of Electric Power Information Networks to the Internet

We now compare the networking infrastructure provided by the Internet with the present and emerging information network infrastructure of the electric power grid—electric power information networks or*EPInets*. The obvious question is *why not just use the Internet (or at least the Internet's common protocols such as TCP and UDP) to implement power system monitoring and control applications?* Answering this question requires understanding major differences in the operating context and required characteristics of the Internet and an EPInet. Distinctions can be drawn in four major areas: static properties, operational practices, performance characteristics and load characteristics.

3.1 Static properties

Static properties that distinguish EPInets from the Internet include overall network size and universal connectivity. The Internet is nearing a billion hosts and it forwards, on demand, data from any one of them to any other. In contrast, EPInets will support several orders of magnitude fewer hosts.[3]

An EPInet's purpose is not universal data delivery but rather high quality delivery to a known set of customers for a known (but slowly changing) set of applications.

3.2 Operational practices

Operational practices that separate the two kinds of networks include admission control and the frequency and control of topology changes.

Because the Internet is designed to serve all comers, at the equipment level new end hosts can be freely added. At the packet level IP backbones are expected to provide (at least) best-effort delivery for every packet that is presented to them. EPInets have *admission control perimenters* across which both addition of new equipment and acceptance of packet traffic are controlled. Traffic policing is of course an essential prerequisite to providing real-time service.

The Internet consists of networks under the independent control of many organizations that operate their networks without much inter-organizational coordination. One major consequence is that the Internet's topology changes frequently. Routers in the Internet can have their configurations changed by system administrators at any time, without warning to others on the net, leading to changes in paths taken by data. Another consequence is that no single location knows more than a tiny fraction of the network topology. Since knowledge of network topology is incomplete at every router, routing calculations have to be done in a distributed fashion. There is no notion of a path *per se*: each packet just gets forwarded towards its destination at each router

[3]The power grid in the eastern portion of the USA and Canada, for example, has 9270 busses in it [9, p. 257], and in the entire US there are 3500 companies that participate in the grid in one form or another (mostly utilities or higher-level coordinating entities; most of the utilities are fairly small in geographic scope) [23]. Thus, we believe that a very reasonable upper bound estimate on the number of future hosts in EIPnet is 10^5, requiring 10^3 to 10^4 FEs.

according to that router's current knowledge of the topology. Because of this, *Internet routing algorithms are subject to short-term instability when links or routers fail or are reconfigured. This is unacceptable for an EPInet.*

In an EPInet, topology changes will be coordinated ahead of time to ensure that QoS requirements of applications continue to be met. Furthermore, given the scale outlined above, it is feasible for a single location to have complete knowledge of all forwarding engines and their connections. For an EPInet, explicit paths are necessary in order to be able to guarantee predictable latency and availability, but paths do not have to be computed frequently because new sources of data (sensors in the grid) and the power applications that consume them are added on a long timescale (months and years). Also, since an EPInet changes much less frequently than the Internet, these paths do not have to be calculated dynamically at data delivery time (forwarding time), but can be computed offline when a subscription is requested. This suggests that relatively static routing algorithms are acceptable, providing that there are multiple disjoint paths set up from the start. The flip side is that the Internet is very robust, given the level of QoS that it provides, in the face of both intended and inadvertent changes. Care must be taken in an EPInet to ensure robustness at the promised level of QoS in the face of infrequent, but still expected, unplanned changes (due, for example to operational mistakes, accidents, and natural disasters). Accountability requirements for electric power operations also require that all configuration changes be logged.

3.3 Service and performance characteristics

EPInets, in order to meet their goals, are required to predictably deliver performance levels that are almost never observed in the Internet. The delivery latency achievable over the wide area in the Internet ranges from, at best a few milliseconds to hundreds of milliseconds, depending on a number of factors including congestion, and the length and bandwidth of the delivery path for each packet.

EPInet applications operate physical processes that must be monitored and controlled in real time. Lack of admission control in the Internet means that the level of predictability achievable cannot be good enough for these applications. EPInets limit their traffic load using admission control at the edges and internal instrumentation to monitor the networks' health. These are only the first necessary steps toward achieving guaranteed latency. EPInets must take advantage of their admission control boundaries by having mechanisms that can decide whether or not additional load can be served and also by having mechanisms to prioritize PUFs to ensure that each one's QoS requirements are met.

Packet loss is another obstacle to delivering the service required of EPInets. Admission control removes one source of packet loss, congestion, but sources such as bit errors and physical damage to links remain. Internet transports that provide reliable data delivery, principally TCP, rely on acknowledgements (to detect drops) and retransmissions (to repair them). However, this cannot provide predictable latency: when a packet is dropped, a latency deadline will almost certainly be missed. *Therefore, in an EPInet post-drop error recovery cannot be used.* Rather, multiple disjoint paths are used for each PUF to proactively tolerate message drops [11]. In this scheme, multiple copies of all updates are automatically sent by EPInet's transport service, one per disjoint path. Paths are established to ensure that the latency along each path meets the latency requirement of the PUF. Thus, if a packet is dropped on one path a different copy of it will arrive along one

of the other paths by the deadline.

Recently proposed transport protocol designs for the Internet, such as SCTP [17] and DCCP [13] attempt to address some of the shortcomings of the Internet's main transport protocols, TCP and UDP. SCTP allows independently ordered message streams to be carried on a reliable connection and it allows multi-homed hosts to gain improved availability by associating multiple host interfaces with each connection. SCTP does not support multicast and though more flexible than TCP in this regard, it nevertheless works too hard to recover from missing messages whose delivery no longer matters to real-time applications.

DCCP is designed to make datagram traffic a better network citizen by making datagram flows sensitive to network congestion. Although DCCP may ultimately have a role to play for non-real-time traffic in an EPInet, real-time traffic requires reserved bandwidth and congestion avoidance rather than congestion detection [21].

3.4 Load characteristics

The final way in which EPInets differ from the Internet is in the characteristics of the loads they are designed to carry. EPInets need to be flexible in order to meet the evolving needs of the power industry but the traffic they carry is more constrained than is found on the Internet. These constraints enable design choices that in turn allow EPInets to meet QoS requirements that are beyond reach in the Internet.

- **Basic forwarded data unit:** On the Internet, the unit of information that is forwarded by a router is an uninterpreted packet, which can be used for a wide variety of purposes (by design). In an EPInet, the unit of information that is forwarded is a single update of a PUV. This enables an EPInet data delivery service to exploit the semantics of periodic update flows, for example, in rate filtering mechanisms.

- **Traffic predictability:** The traffic on the Internet exhibits high short-term variability at the network and transport layers. The variability of Internet traffic means it is possible to achieve good performance, on average, at low cost. However, the variability also works against delivery of guaranteed good performance to specific flows.

  The traffic of an EPInet varies much less, both in the steady state and during many domain (i.e. power) anomalies. The set of application programs monitoring and controlling a critical infrastructure such as the electric power grid changes very slowly, on the time scale of months or even years. Each application has specific data requirements that change only in response to changes in the underlying power grid. Further, power grid operators have to explicitly plan for handling any single failure (*contingency*), for example, of a generating unit or transmission line. Contingency planning occurs far ahead of occurrences so any additional sensor data needed for responding to a contingency can also be planned. Because forwarding in an EIPnet's FEs is a simple calculation, their forwarding tables are fairly simple. Also, due in part to the temporal redundancy inherent in PUFs, their routing tables can be switched very quickly to adapt to changing information needs, such as those engendered by power contingencies or cyberattacks [1].

  Not all of the traffic carried by an EPInet will be PUFs. Control traffic and other less-predictable traffic, such as alarms and alerts, will also be carried. These will, however,

constitute a small fraction of the total traffic. The bandwidth isolation techniques used to provide guaranteed latency for PUF traffic [15] can also provide guarantees for control traffic while keeping non-real-time traffic from interfering. Techniques for handling aperiodic events such as those described in [25] can also be applied.

- **Elasticity of QoS requirements:** In the internet, few applications are designed to throttle back the traffic they impose on the data delivery system [27]. Further, we believe that most application domain experts have never considered how they might be able to throttle back their traffic under different circumstances (notable exceptions here being multimedia streaming applications). In an EPInet data delivery service, we can capture the required delivery rates for each subscriber and thereby exploit the semantics of periodic update flows.

- **Multicast:** Multicast in the Internet is the unusual case. Multicast in an EPInet is the usual case—monitoring traffic is inherently of interest at multiple locations—and different applications require different latency and reliability guarantees.

- **Synchronization between flows:** Recently, power grid sensors called synchrophasors (also called phasor measurement units, PMUs) have GPS-accurate clocks and produce 30–250 updates per second. Synchrophasors are a crucial emerging technology for the grid [16, 20] as they give operators more accurate understanding of the grid's operating status than today's state estimators do. Synchrophasors also introduce possibilities for automatic control that are not feasible today. When performing rate filtering on synchophasor PUVs, care must be taken to preserve the precise global snapshot provided by a set of synchronized updates from multiple PMUs. For more information, see [5, 8, 12].

The comparisons of this section are succinctly summarized in Table 1. The basic IP protocol is a useful foundation on which to build an EPInet but differences in scale, use, and requirements mean that different design choices can and must be made regarding operations, routing, transport protocols, and error recovery.

4 Overview of GridStat

GridStat is our implementation of PUF middleware and the architecture of Figure 1. The mapping of the components is as follows:

- A forwarding element (FE) from Figure 1 is called a *status router* (SR) in GridStat. The word "router" is used deliberately: a SR indeed routes messages along its way. However, due to an EPInet's more controlled and stable environment, described in section 3, static routing is employed which enables very low latencies to be achieved. GridStat's management plane can quickly change routes to adapt to failures (*i.e.*, based on internal instrumentation) or cyber-attacks (*e.g.*, based on intrusion detection systems). In practice, such changes will be needed many orders of magnitude less frequently than an IP router changes its routing table.

  It is crucial to note that the SR performs tasks such as per-subscriber rate filtering on an individual PUF basis, *something that cannot be done at the IP or even the transport layer.*

Characteristic	Internet	EPInet
Network size	10^9 interconnected hosts worldwide	10^5 hosts in a power grid(Eastern US, Western Europe etc.)
Network Design Goal	Provide best-effort delivery for any user and purpose	Provide guaranteed QoS in several dimensions for specific users and purposes
Admission Control Perimeter	None	Complete
Fraction of Managed Traffic	None/Very Little	Almost all. All traffic subject to policing
Central Topology knowledge	Not attempted, because of large scale and dynamicity	Feasible, because of small scale and slow changes
Topology changes	Often & without warning	Not often & virtually always with warning
Frequency of route changes	Frequent; route changes computed using distributed algorithms that may converge slowly in the face of changing topology	Infrequent; route changes computed centrally assuming stable topology
Latency Level Achievable	Slow to Medium	Very Fast
Latency Predictability	Poor	Very Good to Excellent
Recovery delay after dropped packet (with "reliable" delivery)	High (timeout waiting for data or acknowledgement)	Zero (redundant copy sent over disjoint path arrives virtually at the same time)
Forwarding Unit	Uninterpreted packet	PUV update
Traffic Predictability	Low	Very High
Elasticity of QoS requirements	None/Low	High
Multicast: multiple subscribers to a single update flow	A small fraction of the overall traffic; does not justify significant optimization	The common case. Multiple subscribers to a single update flow may have different latency and reliability requirements. Significant opportunity for optimization.
Synchronization between flows	No (a few multimedia apps notwithstanding, but these are not handled at network/transport layer, rather app or middleware)	Yes, temporal synchronism between flows originating at different sources (example: PMUs when rate filtering)

Table 1. Internet and EPInet characteristics

Rather, such actions have to be performed at the middleware layer because they need to utilize knowledge of data variables involved and the data-level QoS requirements.

- GridStat's management plane is a distributed collection of components called *QoS brokers*. The management plane handles subscription requests for clients, calculating a route that meets the subscriber's requirements and then updating forwarding tables in the SRs. The QoS brokers are organized hierarchically. This maps naturally onto the geographic and business organization of the electric power grid. QoS brokers are repositories for and enforcers of businesses' resource and security policies related to information dissemination. The hierarchy also enables the QoS Brokers to serve naturally as aggregators of information received from power sensors and data plane instrumentation.

Other key design decisions include:

- SRs perform both rate filtering and multicast. This directly enables per-cache latency and rate guarantees while at the same time supporting efficient multicast—no update is ever sent over the same link twice. This in turn enhances reliability and availability by reducing the amount of network traffic required for a given set of PUFs.

- Different levels of availability are provided to subscribers by delivering updates over multiple redundant paths, which are calculated by the management plane so that each path meets an individual subscriber's rate and latency requirements [11].

  The precise amount of availability provided by a given PUF cannot presently be calculated in GridStat, but we believe that providing good estimates of availability is reasonably straightforward engineering.

Multicast and rate filtering interact. They are implemented by a single forwarding mechanism that works as follows. The status variable ID and timestamp are extracted from each incoming packet. The ID is used as a key to look up the outgoing links with active subscriptions for that ID. The lookup yields for each link a list of subscription intervals. A calculation based on the interval and timestamp yields a *forward* or *do not forward decision* for that link. If any of the subscriptions on a link produce the decision *forward* then the packet is sent on the link; otherwise, it is dropped—there is currently no need for it downstream.

5 Related Work

5.1 Overview of Publish-Subscribe Research

Much of the research literature for publish-subscribe systems deals with either the expressive power of subscriptions or event correlation, or both. (For a few recent surveys, see [7, 14]. These research systems provide far more general services than what is required for the electric power grid (and supported in PUF-based PUVs). The flip side of this generality, however, is that their latency and scalability are far from what is required by the electric power grid, or what is achievable by more specialized and optimized systems such as GridStat.

For example, [14] presents a comprehensive taxonomy of publish-subscribe systems. It notes that event filters can be evaluated either at subscription time or at event propagation time. The

paper then notes that event services have traditionally evaluated event filters at propagation time. Indeed, it does not list any systems that do filtering at subscription time, and we are not aware of any. However, subscription-time evaluation not only avoids slowing down event propagation like propagation-time evaluation does, but as [14] notes "it may assess the availability of resources, authenticate a connection, or process admission control". This is precisely what PUF-based PUVs enable.

5.2 Flow-Based Variable Related Research

Table 2 summarizes research and deployed software that is related to PUF middleware. None of these are, in our opinion, closely related to GridStat in most or all key areas: for example, none meet the NASPInet requirements outlined in section 2.1; none are designed to provide ultra-low delivery latency; and none directly allow different subscribers to a PUV to be provided a different level of QoS. Table 2 has three sets of rows: first for commercially available or open source middleware; second for PUFs and the most closely related work to them; and third for other categories of research systems. In this table the columns are properties that are germane to meeting the requirements in section 2.1.

QoS Per-Subscriber: can the system provide a different level of QoS for different subscribers of the same topic or variable? PUF-based middleware is the only kind we know of that supports this directly, though PASS and WSN-based PUF systems allow this indirectly by hand-coding forwarding engines. These hand-coded variants of per-subscriber QoS cannot provide hard real-time guarantees or ultra-low latencies, and are rather intended to help minimize battery usage (in WSNs) or use of scarce wide-area network bandwidth in military operations (PASS).

Permanence: what does a forwarding element do if an update cannot be delivered to the next destination in its delivery path (with the final hop of course being a subscriber)? With transient communication, the event is dropped if an event/update cannot immediately be forwarded to the next destination. With persistent communication, a message is stored at a forwarding element for as long is it takes to deliver it to the next hop. We note that systems that provide persistent communication employ mechanisms that are unsuited for providing ultra-low latency and providing real-time performance.

Abstraction: what abstraction does the framework provide the programmer, both for the publisher and the subscriber? Most publish-subscribe systems provide an abstraction of an event (similar to a message). Only GridStat and WSN-PUFs provide the abstraction of a data variable.

Real-Time: does the system provide hard (H) or soft (S) real-time guarantees, and across a LAN or a WAN? (Of course, if it is provided across a WAN it also is across a LAN, and all systems provide best-effort, even across a WAN, so this is not listed.) We note also that of course real-time means predictable timeliness, not always ultra-fast, so even systems that provide hard real-time behavior are not necessarily designed to provide the very low latencies required for the power grid.

Redundancy: do the PUFs in the system provide spatial redundancy (multiple disjoint paths) or do they have natural temporal redundancy in their flows (the next update for a given variable is coming shortly)?

System	QoS Per-Sub	Permanence	Abstraction		Real-Time		Redundancy	
			Publisher	Subscriber	Hard	Soft	Spatial	Temporal
MOM/queues	no	Persistent	Event	Event	no	LAN	Some	no
File Transfer	no	Persistent	File	File	no	no	no	no
Web Svcs/SOA	no	(Undefined)	Varied	Varied	no	no	no	no
GridStat & FbV	Yes	Transient	Variable Update	Updated Variable	WAN	WAN	Yes	Yes
PASS	Codeable	Persistent	N/V pair	Lg. Record	no	no	Codeable	Codeable
WSN-PUF	Codeable	Transient	Variable Update	Updated Variable	no	LAN	no	Yes
RT AV Streams	no	Transient	AV stream	AV frames	no	LAN	no	Yes
COSMIC	no	Transient	Event	Event	LAN	LAN	no	no
RT TAO	Codeable	Transient	Event	Event	LAN	LAN	Codeable	no

Table 2. Related work

6 Conclusions and Future Work

In this paper we presented a new kind of publish-subscribe middleware, based on periodic update flows, and the periodically updated variable paradigm that it supports. These abstractions, and the GridStat middleware framework that implements them, are tailored for the very low latencies and high availability required in the emerging electric power information network. PUF middleware exploits the much more controlled nature of EPInets, as compared to the Internet, to achieve fast, reliable delivery of updates.

Our implementation of PUF middleware, GridStat, has a number of advanced features that are beyond the scope of this paper. Work has been done to provide confidentiality and integrity for the data plane, with security modules that can be replaced securely and without the risk of man-in-the-middle attacks [18]. A two-way RPC mechanism has also been developed, with requests and replies delivered over QoS-managed GridStat PUFs [24]. The RPC mechanism supports pre-conditions and post-conditions over PUVs to provide additional safety for controlling a remote actuator.

GridStat was first demonstrated in 2002, and has had live utility data since 2003. In 2007 it was involved in a pilot project spanning two United States Department of Energy National Laboratories. The QoS Brokers are written in Java, and there are both Java and C++ versions of the status router. We anticipate that GridStat will be involved in two or more large pilot projects starting in the next 12–18 months.

Future work planned for GridSTat involves developing policy languages for many configuration and adaptation capabilities in GridStat; hardening and documenting the code; finishing a network processor-based status router implementation; replicating QoS Brokers to tolerate benign and then Byzantine failures; ongoing work on authentication mechanisms for the management plane; and many other pragmatic tasks such as supporting data formats currently used in the electric power grid.

References

[1] S. Abelsen. Adaptive GridStat information flow mechanisms and management for power grid contingencies. Master's thesis, Washington State University, 2007.

[2] D. Bakken, A. Bose, and S. Bhowmik. Survivability and status dissemination in combined electric power and computer communications networks. In *Proceedings of the Third Information Survivability Workshop (ISW-2000)*, October 2000.

[3] D. Bakken, A. Bose, C. Hauser, I. Dionysiou, H. Gjermundrød, L. Xu, and S. Bhowmik. Towards more extensible and resilient real-time information dissemination for the electric power grid. In *Proceedings of Power Systems and Communications Systems for the Future, International Institute for Critical Infrastructures*, Beijing, China, September 2002.

[4] D. Bakken, T. Evje, and A. Bose. Survivable status dissemination in the electric power grid. In *Proceedings of the Information/System Survivability Workshop, in Supplement Proceedings of the International Conference on Dependable Systems and Networks (DSN-2001)*, July 2001.

[5] D. E. Bakken, C. H. Hauser, H. Gjermundrød, and A. Bose. Towards more flexible and robust data delivery for monitoring and control of the electric power grid. Technical Report EECS-GS-009, Washington State University, May 2007.

[6] I. Dionysiou, K. H. Gjermundrød, and D. Bakken. Fault tolerance issues in publish-subscribe status dissemination middleware for the electric power grid. In *Supplement of the 2002 International Conference on Dependable Systems and Networks (DSN-2002)*, Washington, DC, June 2002.

[7] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Mermarec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, July 2003.

[8] H. Gjermundrød. *A flexible QoS-managed status dissemination middleware framework for the electric power grid*. PhD thesis, Washington State University, USA, August 2006.

[9] J. D. Glover and M. S. Sarma. *Power System Analysis*. Brooks/Cole Thompson Learning, 2002.

[10] C. Hauser, D. Bakken, and A. Bose. A failure to communicate: Next-generation communication requirements, technologies, and architecture for the electric power grid. *IEEE Power and Energy*, 3(2):47–55, March/April 2005.

[11] V. S. Irava. *Low-cost delay-constrained multicast routing heuristics and their evaluation*. PhD thesis, Washington State University, USA, August 2006.

[12] R. A. Johnston, C. Hauser, K. H. Gjermundrød, and D. Bakken. Distributing time-synchronous phasor measurement data using the GridStat communication infrastructure. In *Proceedings of 39th Annual Hawaii International Conference on System Sciences (CD/ROM)*, Kauai, Hawaii, January 2006.

[13] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), Mar. 2006.

[14] R. Meier and V. Cahill. Taxonomy of distributed event-based programming systems. *The Computer Journal*, 48(5), 2005.

[15] S. Muthuswamy. System implementation of a real-time, content based application router for a managed publish-subscribe system. Master's thesis, Washington State University, 2008.

[16] D. Novosel, V. Madani, B. Bhargava, K. Vu, and J. Cole. Dawn of the grid synchronization. *Power and Energy Magazine, IEEE*, 6:49–60, January-February 2008.

[17] L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002.

[18] E. Solum. Achieving over-the-wire configurable confidentiality, integrity, authentication and availability in GridStat's status dissemination. Master's thesis, Washington State University, 2007.

[19] Specification for the North American Synchrophasor Initiative (NASPI), April 2008.

[20] Statement of work: Specification for North American Synchrophasor Inititiative (NASPI), May 2008. http://www.naspi.org/resources/dnmtt/quanta_sow
.pdf.

[21] A. S. Tanenbaum. *Computer networks (4th ed.)*. Prentice-Hall, Inc., 2003.

[22] K. Tomsovic, D. Bakken, M. Venkatasubramanian, and A. Bose. Designing the next generation of real-time control, communication and computations for large power systems. In *Proceedings of the IEEE (Special Issue on Energy Infrastructure Systems)*, May 2005.

[23] U.S.-Canada Power System Outage Task Force. *Final report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations*, March 2004. https://reports.energy.gov/BlackoutFinal-Web.pdf.

[24] E. Viddal. Ratatoskr: wide-area actuator RPC over GridStat with timeliness, redundancy, and safety. Master's thesis, Washington State University, 2007.

[25] Y. Zhang, C. Gill, and C. Lu. Reconfigurable real-time middleware for distributed cyber-physical systems with aperiodic events. In *The 28th International Conference on Distributed Computing Systems (ICDCS 2008)*, June 17-20 2008.

[26] J. Zinky, L. O'Brien, D. Bakken, V. Krishnaswamy, and M. Ahamad. PASS: A service for efficient large scale dissemination of time varying data using CORBA. In *International Conference on Distributed Computing Systems*, pages 496–506, Austin, TX, June 1999.

[27] J. A. Zinky, D. E. Bakken, and R. E. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1), 1997.