Chapter 4

[OpenFlow/SDN highlighted in light blue]

[MPLS highlighted in yellow]

[Peer-to-Peer Pub-Sub (e.g., OMG DDS) in green]

GridStat:

High Availability, Low Latency and Adaptive Sensor Data Delivery for Smart Generation and Transmission

David E. Bakken[1], Harald Gjermundrød[2], and Ioanna Dionysiou[2]

[1]School of Electrical Engineering and Computer Science
Washington State University
Pullman, Washington, USA
bakken@eecs.wsu.edu

[2] Department of Computer Science
University of Nicosia
Nicosia, Cyprus
{gjermundrod.h, dionysiou.i}@unic.ac.cy

A slightly updated and hugely reformatted version of this draft will appear in the following book:
D. Bakken and K. Iniewski, ed. *Smart Grids: Clouds, Communications, Open Source, and Automation,* CRC Press, 2014, ISBN 9781482206111.

This text refers to "Chapter 3" on occasion. This chapter is
G. Zweigle, "Emerging Wide-Area Power Applications with Mission Critical Data Delivery Requirements", in D. Bakken and K. Iniewski, ed. *Smart Grids: Clouds, Communications, Open Source, and Automation,* CRC Press, 2014.

Chapter 3 is an almost complete rewrite of the power applications section in the following (which is also [59]):
D. Bakken, A. Bose, C. Hauser, D.Whitehead, and G. Zweigle. "Smart Generation and Transmission with Coherent, Real-Time Data. *Proceedings of the IEEE,* 99(6), June 2011.

It was updated in Chapter 3 with newer examples and also broadened to apply to entire categories of applications.

Table of Contents

1. INTRODUCTION 5

2. REQUIREMENTS AND GUIDELINES FOR COHERENT REAL-TIME DATA DELIVERY 5

2.1. System Model 6

2.2. Delivery Requirements for a WAMS-DD 7

2.3. Implementation Guidelines for a WAMS-DD 10

2.3.1. Context 10

2.3.2. Implementation Guidelines 10

2.3.3. Analysis 14

3. ANALYSIS OF EXISTING TECHNOLOGIES FOR A WAMS-DD 14

3.1. Technologies and Standards at the Traditional Network Layers 14

3.1.1. Very Inadequate 14

3.1.2. Better Yet Incomplete 15

3.2. Middleware technologies and standards 16

3.2.1. Definition of Middleware 16

3.2.2. Broker-based Publish-Subscribe Middleware 17

3.2.3. Peer-to-Peer Publish-Subscribe 18

3.2.4. Other Middleware 19

3.3. Existing Electric Sector Communications-Related Technologies and Standards 19

3.4. Summary 21

4. NASPINET 23

4.1. Architecture 23

4.2. Service Classes 23

5. WHY WAMS-DD MUST INCLUDE MIDDLEWARE NOT SOLELY NETWORK LAYERS 25

5.1. Middleware in General 25

5.2. Middleware and QoS+ 25

5.3. Middleware and DRs and IGs 26

5.4. Middleware and Legacy Systems 27

5.5. Middleware and Avoiding QoS Stovepipe Systems 28

6. SECURITY AND TRUST FOR WAMS-DD 29

7. GRIDSTAT 30
7.1. OVERVIEW 30
7.1.1. Capabilities 30
7.1.2. Performance and Scalability 32
7.2. RATE FILTERING FOR PER-SUBSCRIBER QOS+ 32
7.2.1. Forwarding Algorithm 32
7.2.2. Example of the Forwarding Algorithm 33
7.3. CONDENSATION FUNCTIONS 33
7.3.1. Condensation Function Architecture and Capabilities 33
7.3.2. Condensation Function Development 34
7.4. OPERATIONAL MODES 34
7.4.1. Mode Change Algorithms 35
7.4.2. Overview of performance 37
7.5. SYSTEMATIC ADAPTATION VIA DATA LOAD SHEDDING WITH MODES 37
7.6. REMOTE PROCEDURE CALL (RPC) 38
7.6.1. Mechanism Details 38
7.6.2. The 2WoPS Protocol 38
7.6.3. The Ratatoskr RPC Mechanism 39
7.7. SECURITY AND TRUST IN GRIDSTAT 39
8. A NEW WORLD FOR POWER APPLICATION PROGRAMMERS AND RESEARCHERS 40
9. CONCLUSIONS 41

Table of Figures

FIGURE 1: ARCHITECTURE AND SYSTEM MODEL OF A WAMS-DD 6
FIGURE 2: BROKER-BASED WAMS-DD 17
FIGURE 3: PEER-TO-PEER WAMS-DD 18
FIGURE 4: NASPINET CONCEPTUAL ARCHITECTURE (COURTESY NASPI) 23
FIGURE 5: MIDDLEWARE INTEGRATING LEGACY AND NEW COMMUNICATIONS SUBSYSTEMS 27
FIGURE 6: GRIDSTAT ARCHITECTURE 30
FIGURE 7: GRIDSTAT FORWARDING ENGINE OPERATION 31
FIGURE 8: ILLUSTRATION OF THE FILTERING STEPS 33
FIGURE 9: CONDENSATION FUNCTION MODULES 34
FIGURE 10: MODE EXAMPLE 35
FIGURE 11: RPC MODULE STACK 38

Table of Tables

TABLE 1: IMPLEMENTATION GUIDELINES AND THE DELIVERY REQUIREMENTS THAT MANDATE THEM 9
TABLE 2: COVERAGE OF DELIVERY REQUIREMENTS AND IMPLEMENTATION GUIDELINES BY EXISTING TECHNOLOGIES 22
TABLE 3: NASPINET TRAFFIC CLASSES AND THEIR ATTRIBUTES 24

1. INTRODUCTION

Electric power grid around the world are getting increasingly stressed by factors including inadequate transmission growth, system operators and power engineers retiring in large numbers, and integration of renewable sources of energy that whose physics is different from well-known sources such as hydro and coal. All these factors, and more, can be mitigated by greatly increasing sharing of sensor data between utilities (and ISOs/RTOs) in a grid, something that is extremely limited in power grids today. This is an important step in helping make grids more resilient, because inadequate situational awareness resulting from little inter-utility data sharing has been a major contributing factor in virtually all recent major blackouts. In such blackouts, there are always a few physical or operational root causes that are blamed. However, due to the poor situational awareness, caused in large part by inadequate communications, the grids' operators are unaware of emerging problems until it is too late to prevent a blackout.

Additionally, closed-loop applications such as distributed control and distributed protection are being increasingly deployed. However their data delivery requirements are the most extreme of any infrastructure in the world, much moreso than any other example of what is recently being called the **Industrial Internet** and a **cyber-physical system**. Thus, it is crucial to ensure that data delivery for power grids be appropriate for the task at hand. Conversely, adopting more generic data delivery infrastructures from other industries, such as air traffic control, that seem superficially similar, can cause blackouts. Likewise, using network mechanisms such as MPLS that provide very weak statistical guarantees and very crude control over the network (for example, MPLS only has 8 traffic categories) are similarly unwise.

In this chapter we describe GridStat, which is in essence an overlay network providing strong guarantees and specialized semantics for Wide Area Monitoring System Data Delivery (WAMS-DD). In configurations where GridStat's specialized routers don't have 100% penetration, they can be augmented by other mechanisms providing very strong network-level QoS guarantees in order to support closed-loop applications over the wide area. GridStat can also overlay (and help better manage and integrate) other networking subsystems with weaker QoS guarantees in a critical infrastructure. **This is in essence a software defined network (SDN)**. However, it is not a generic one: it is an SDN infused with semantics from both middleware and power grid applications and sensors that allow it to exert fine grained control over its traffic. This not only provides very low latency and extremely high availability, but also allows the SDN to change its delivery patterns in a small fraction of a second to adapt to power anomalies, benign IT failures, and cyber-attacks.

The remainder of this chapter is organized as follows. Section 2 describes baseline delivery requirements that any WAMS-DD system must meet. It then derives 20 implementation guidelines that those requirements mandate for any WAMS-DD to utilize. Section 3 then presents a detailed analysis of how well existing technologies at the network layers, at the middleware layers, and from the electricity sector meet these requirements and guidelines. Next, Section 4 overviews NASPInet, a WAMS-DD initiative in North America. After this, Section 5 explains why WAMs-DD must include the middleware layers. Section 6 then summarizes security and trust issues for WAMS-DD. Section 7 describes Gridstat. Section 8 summarizes how GridStat enables a new way of thinking and programming for programmers and researchers of power applications. Finally, Section 9 concludes.

2. REQUIREMENTS AND GUIDELINES FOR COHERENT REAL-TIME DATA DELIVERY

Data delivery in the power system today can be greatly improved by reducing the use of hard-coded protocols, developing more reusable systems, and providing real end-to-end performance guarantees. For example, in protection applications, over-provisioning of bandwidth provides low latencies and high availability in the steady state but not necessarily in the face of IT failures, bugs in software or hardware that cause spurious traffic, or cyberattacks. As more applications that can exploit coherent, real-time data delivery emerge, such as those outlined in Chapter 3, using isolated networks may soon become unsustainable, as will deploying a new communications system for each new application, application family, or project.

Fortunately, the state of the art in distributed computing, real-time systems, and fault-tolerant computing has, since the 1990s, support providing strong guarantees with data delivered to many applications. If designed, implemented,

and validated correctly, **a state-of-the-art data delivery system can greatly lower the barrier to enter (in both time and money) and enable deployment of new power applications** by simplifying the process of adding new sensors. If designed incorrectly, it will be difficult to maintain in the future because it will not be able to keep up with increasing demands. Further, these data delivery systems will have a long life, and no single network-level mechanism (for multicast, security, or QoS) can be assumed to be everywhere.

It is crucial, therefore, that data delivery systems between the mission-critical peer-to-peer automatic protection and control systems, and the power grid's operations IT backbone, have interoperability between different kinds of network mechanisms providing the same property, such as delay guarantees [1].

In this section, we examine how a WAMS-DD will be an enabling technology for the new and emerging power system. We first overview the performance and reliability requirements that a WAMS-DD must meet. We then present implementation guidelines, based on best practices in other industries and in the field of distributed computing systems, for achieving these requirements. Next we compare how existing technologies meet these delivery requirements and design guidelines when used in isolation without additional overlay networks. This includes technologies and standards at the network layers (and below), the middleware layer(s), and related ones from the power industry. We also discuss relevant research and development for wide-area middleware. After this, we discuss the emerging NASPInet effort and the GridStat data delivery middleware.

This section focuses on performance and availability guarantees. Cyber-security is discussed in Sections 6 and 7.7.

Note that the following analysis focuses on coherent but asynchronous data delivery for operations, but the emerging communications infrastructure will provide the additional benefit of distributing the time signal required for time-synchronized measurements and control. Typically time is received via GPS and distributed over a separate physical network using protocols such as IRIG. This results in a physical cable connection to the measuring devices such as PMUs. Combining time distribution with the communications network provides advantages such as simplicity and reliability [2]. Furthermore, for many applications, such as a control scheme or system protection scheme, which use separate mission-critical peer-to-peer communications, operation can proceed even if global time is lost, as long as they maintain a local coherent time signal. The communications infrastructure can provide this locally common time signal when the primary GPS signal is unavailable.

2.1. System Model

Fig. 1 depicts the architecture of a WAMS-DD. Application programs or firmware that emit a stream of updates are called publishers, which are denoted as Pub_1 through Pub_N in the diagram; Pub_1, for example, outputs updates to variables X and Y. Applications that receive these updates are called subscribers, which are denoted as Sub_1 through Sub_N. In the diagram, Sub_1 subscribes to Y from Pub_1 and to W from Pub_N.

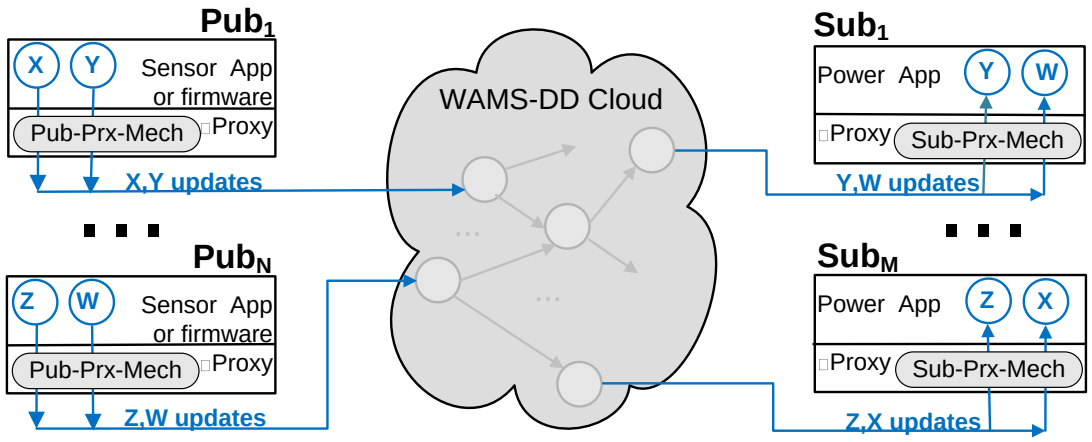

Figure 1: Architecture and System Model of a WAMS-DD

In the usual case in publish-subscribe (pub-sub) systems, neither publisher nor subscriber needs to know about the other; they are decoupled such that they only know about the variable they publish or subscribe to and how to contact the delivery system. In cases where the subscriber requires confirmation that the update came from its

legitimate publisher—which may be common with a WAMS-DD—data integrity techniques from the computer security field can be used by the data delivery system.

Creating a pub-sub delivery path requires two steps. Publishers register their variables with the delivery system (only once per variable, not once per subscriber), and later subscribers request a subscription to a given variable. For both publishers and subscribers, the delivery system returns a handle to a piece of code called a proxy, which is generated at compile time by the data delivery middleware. This proxy contains logic provided by the data delivery service, which, besides doing the usual middleware proxy activities such as packaging of the parameters into a message, is also a place where data delivery mechanisms may reside. In Figure 1 we denote a publisher-side proxy as Pub-Prx-Mech and the subscriber-side proxy as Sub-Prx-Mech.

After the variable is registered and subscribed to, updates to variables flow from publishers to subscribers, as shown in blue in Figure 1. To do this, they traverse what we call the WAMS-DD Cloud. This is opaque because, as shown later in this section, it can be implemented in different ways resulting in different tradeoffs. For the purposes of our system model, the WAMS-DD Cloud consists of a graph where the edges are network links and the nodes contain forwarding mechanisms that can forward a message on its way toward a subscriber.

We note that this **WAMS-DD is in practice a virtual overlay implemented in large part with existing links that a utility already has**. These existing links will have inherent tradeoffs, so considering the capabilities, most utilities/ISOs should conclude that their existing infrastructure will not support any closed-loop applications across a wide area at the very least. Thus, the existing links would in practice be augmented with adding key additional links from various sources, such a Tier 1 fiber provider, lighting up utility dark fiber, or buying some new links with technologies providing very strong guarantees (see Table 2 at the end of Section 3). Further, integration of the different kinds of network links being overlaid is discussed further in Section 5.

Updates from a publisher of a sensor variable thus traverse one or more paths to be delivered to a given subscriber. Along a given path, in the general case an update may be delayed, so that its required delivery latency cannot be met, or the update may be dropped due to failures in a network link or forwarding node or due to a cyberattack. However, the probabilities of an update not meeting its delivery requirements can be held extremely low by carefully designing the WAMS-DD and by allocating multiple paths for important updates. That is, a WAMS-DD can be constructed so that the on-time delivery probability is very high, so long as its design constraints are met. Informally, these include forwarding capacity per node, maximum link traffic, number and kind of benign failures, and cyberattacks, etc.

We now overview the delivery requirements in Section 2.2; then in Section 2.3, we describe implementation guidelines that can be used to meet these delivery requirements with extremely high probabilities. These probabilities offer the potential to practice using dual isolated networks for critical protection applications, while at the same time supporting many more application families with thousands of update flows. However, such delivery technologies clearly need to be proven in the field before any migration to them can begin to be contemplated.

2.2. Delivery Requirements for a WAMS-DD

The following delivery requirements (DRs) must be met by a WAMS-DD [3][4]; these do not include the details of cybersecurity-related requirements (which are overviewed in Sections 5.5 and 7.7). These DRs encompass the requirements of other network layers, which the WAMS-DD layer build upon. They are summarized in Table 1, along with their implementation guidelines, which will be overviewed in the next subsection.

Requirement 1: *Hard, end-to-end (E2E) guarantees must be provided over an entire grid because protection and control applications depend on the data delivery*. The guarantees must be deterministic: met unless the system's design criteria have been violated (e.g., traffic amount, number of failures, and severity of cyberattack).

Requirement 2: *WAMS-DDs must have a long lifetime and thus must be designed with future-proofing in mind*. This is crucial in order to amortize costs over many projects, utilities, grids, etc. The goal of NASPInet, for example, is to last at least 30 years. To do this, it is important to not have applications hardcoded directly using the APIs of a single QoS+ mechanism for a given kind of resource (e.g., encryption, network delivery). Rather, wrappers or, better, more comprehensive middleware should be used. Its libraries can be updated to add new lower-level QoS mechansims as they become available without having to recompile the application.

Requirement 3: *One-to-many is the normal mode of communications, not point-to-point*. Increasingly, a given sensor value is needed by multiple power applications. This is best done by publish-subscribe middleware, but in a worst case (unfortunately common in power grids) is done by network-level multicast (e.g, IP Multicast).

Requirement 4: *End-to-end guarantees must be provided for a wide range of QoS+*. Data delivery for the power system is not "one size fits all" [5], as shown in Chapter 3. For example, to provide very low latencies, very high rates, and very high criticality/availability to all applications would be prohibitively expensive. Fortunately, many applications do not require these stringent guarantees, but their less stringent requirements must nevertheless be met.

Examples of the wide ranges that must be provided follow:

- **A.** Latency and Rate: Ten milliseconds or less, up to seconds (or hours or days for bulk transfer traffic), .001 Hz to 720 Hz or more.
- **B.** Criticality/Availability: IntelliGrid [5] recommends five levels of availability of data, from medium up to ultra (99.9999%), i.e., 6 9s.
- **C.** Cybersecurity: Support a range of tradeoffs of strength of encryption and authentication compared to delay induced, resources consumed, and the exact flavor of authentication provided.

Requirement 5: *Some merging and future SIPS, transient stability, and control applications require ultra-low latencies* and one-way delivery on the order of a half or full power cycle (8–16 ms in the US) over hundreds of miles [6]. Thus, any forwarding protocols should not add more than a millisecond or two of latency (through all forwarding hops) on top of the speed of light in the underlying communications medium.

These latencies must be provided in a way that:

- **A.** Is *predictable and guaranteed for each update message*.

  Each sensor update needs to arrive with an extremely high probability within its required guaranteed deadline, not a much weaker aggregate guarantee over longer periods of time, applications, and locations such as is provided by multiprotocol label switching (MPLS) technology [7].
- **B.** Tolerates *non-malicious failures* in the WAMS-DD infrastructure.

  No system can tolerate unlimited kinds and numbers of failures. However, much like the power system must continue in the face of one or more known contingencies, the IT infrastructure on which it increasingly depends must still provide these hard, end-to-end guarantees in the face of failures (up to design limits).
- **C.** Tolerates *malicious cyber-attacks*.

  Power systems are known to be subjects of extensive study and probing by multiple organizations that have significant information warfare capabilities, including nation states, terrorist organizations, and organized crime. A WAMS-DD must adapt and continue to deliver data despite cyber-attacks of a designed severity (a bar that should be increasable over the life of the system). Note that a bug in hardware or software that generates spurious traffic can have an effect similar to that of a cyber-attack.

Requirement 6: *Extremely high throughput is required*. Today's synchrophasor applications are generally limited to 30 or 60 Hz in the USA, in part because the communications systems they use are not designed to support higher rates. To not provide much higher sustainable throughput would greatly limit the number of new applications that can help the power system's stability. Indeed, not just synchrophasors but digital fault recorders (DFRs) and IEDs in substations provide a wealth of data. It is quite conceivable and likely that "If you build it, they will come" and there will be many thousands of synchrophasors, relays, DFRs, and other sources of sensor updates across a grid. These devices can output at 720 Hz and sample at 8 kHz, but their full output is not always used remotely due to communications limitations. If key relay or DFR data could be delivered from a set of devices across a grid at 720 Hz, many new opportunities would open up for transient protection without using expensive dedicated networks or "drilling down" into the root causes of an ongoing power contingency using additional contingency-specific data.

We are not aware of any commercial or military market for a wide-area data delivery infrastructure that has the stringent requirements of a WAMS-DD including ability to enforce complete perimeter control, ability to know the vast majority of the traffic ahead of time, and other factors incorporated into the implementation guidelines described next in this paper. The reason is quite simple; electric power is the only market with such stringent

requirements. However, these requirements are achievable using state-of-the-art distributed real-time embedded computing [8], [9], as long as a careful end-to-end analysis is done [10] and the core data delivery mechanisms are not saddled with unnecessary features. Much broader reliability has been explored in the fault-tolerant distributed computing community, from where appropriate lessons, both good and bad, should be heeded [11], [12].

Table 1: Implementation Guidelines and the Delivery Requirements That Mandate Them

DR1: Hard E2E WAN Guarantees	DR2: Future-Proofing	DR3: One-to-many comms.	DR4: Wide Range of QoS+…	4A: Latency and Rate	4B: Criticality/Availability	4C: Cybersecurity	DR5: Ultra-Low Latencies…	5A: Per-Update and Predictable	5B: Tolerating Failures	5C: Tolerating Cyberattacks	DR6: High Throughput	IG*x* Prerequisites	**Summary of Implementation Guideline IG*x***
X								X	X				IG1: Avoid post-error recovery mechanisms
X				X				X	X	X	X		IG2: Optimize for rate-based sensors
		X									X		IG3: Provide per-subscriber QoS+
		X									X		IG4: Provide efficient multicast
												2, 3	IG5: Provide synchronized rate down-sampling
X					X			X			X		IG6: Don't depend on priority-based "guarantees"
X	X			X	X	X							IG7: Provide end-to-end interoperability across different/new IT technologies (multicast, QoS+)
X								X			X		IG8: Exploit *a priori* knowledge of traffic
X								X	X	X		8	IG9: Have systematic, quick internal instrumentation
X								X					IG10: Exploit smaller scale of the WAMS-DD
X								X				8–10	IG11: Use static, not dynamic, routing
X								X	X	X			IG12: Enforce complete perimeter control
X								X	X	X	X	12	IG13: Reject unauth. messages quickly and locally
											X	2, 8	IG14: Provide only simple subscription criteria
								X			X	2	IG15: Support transient, not persistent, delivery
								X			X		IG16: Don't over-design consistency and (re)ordering
											X	2, 8, 14–16	IG17: Minimize forwarding-time logic
X	X			X	X	X							IG18: Support multiple QoS+ mechanisms for different operating conditions
								X			X	17	IG19: Inspect only message header, not payload
X								X			X		IG20: Manage aperiodic traffic

2.3. Implementation Guidelines for a WAMS-DD

We now discuss practical issues that arise when trying to construct a WAMS-DD that meets the above requirements.

2.3.1. Context

The requirements outlined in the previous section were kept to a bare minimum. In order to achieve them, however, we believe it will be necessary to utilize a number of *implementation guidelines* (IG), many which are quite different from what is provided in today's best-effort Internet and what has been the conventional wisdom in networking research. In this section we enumerate and explain these IGs. However, they are necessary (or at least highly advisable) in order to meet the stringent DRs.

Some of the IGs below (e.g., IG4 and IG5) are actually deemed requirements for NASPInet [13], but we describe them here as IGs because it is possible to build WAMS-DD without them (though we believe that would be inadvisable for an inter-utility backbone such as the proposed NASPInet (see Section 4)). These IGs are drawn from a number or sources, including our knowledge of what the state of the art in distributed computing has demonstrated is feasible, best practices in other industries, and decades of experience gained in DARPA wide-area application and middleware projects of ours and others.

We note that these guidelines refer **to best practices of how to *build* a WAMS-DD. Other will apply on how to *use* one and will need to be developed as best practices**. These are beyond the scope of this chapter, but are crucial. For example, in our experience, many power engineers assume that with synchrophasors, they should have the phasor data concentrators (PDCs) inside their utility at many levels. However, this is a very bad idea for updates that need to be delivered with ultra-low latencies (DR5). A PDC aggregates many PMU signals, does error correction and angle computation, then outputs the collection of this information for a given PMU time slot (this is called *time alignment*). Such a PDC may have dozens of PMU signals coming into it, so doing time alignment means that *the output has to wait until the slowest PMU update arrives*. In this case, the updated sensor values will have suffered significant delays even before they leave the utility to be transported by a wide-area WAMS-DD such as NASPInet. Thus, for those updates that require ultra-low delivery latency, any PDC or other time alignment should be placed as close to the subscribers as possible (ideally in their local area network), even at the cost of a small amount of either wasted bandwidth and duplication of PDCs. Similarly, data that is required with extremely low latency should not have a database in its path: it can be entered into a database after it is sent out, but the database must not slow down the fast delivery path.

We also note that the scope of these IGs involves only the data delivery system for WAMS-DD. It does not include the supporting services that will be required for configuration, security, path allocation, resource management, etc. It will be important for the WAMS-DD that the use of these tools avoids hard-coding choices, but rather allows them to be specified in a high-level policy language (or at least a database) [14], [15], [16]. For an example of a hierarchical version of such services (a "management plane"), see [17], [18].

2.3.2. Implementation Guidelines

Table 2 provides an overview of the IGs and the DRs that require the given IG. We now explain each of the IGs in turn.

Guideline 1. *Avoid post-error recovery mechanisms*. Traditional protocols for the internet in general and reliable multicast protocols from the fault-tolerant computing research community use post-error recovery. In these protocols the receiver either sends a positive acknowledgement (ACK) when it receives a message, or it sends a negative acknowledgment (NACK) when it concludes that the message will not arrive. However, both add considerable latency when a message[1] gets dropped: three one-way latencies are required plus a timeout that is much greater than the average one-way message latency.

The better alternative is to send sensor updates (messages) proactively over multiple disjoint paths, each of which meets the latency and rate requirements [19], [20]. Indeed, if multiple independent messages, each going

[1] We use the term "message" rather than "packet," because in many cases we are describing middleware-layer mechanisms above the network and transport layers.

over a QoS-managed path, cannot meet the delivery deadline, then sending ACKs or NACKs is very unlikely to help, and indeed will almost certainly only make things much worse.

Guideline 2. *Optimize for rate-based sensors.* WAMS-DD can be made with higher throughput and robustness if they are not over-engineered. General-purpose publish-subscribe systems offer a wide range of traffic types, because they are designed to support a wide range of applications. However, in a WAMS-DD, the vast majority of the traffic will be rate-based. Design accordingly.

Guideline 3. *Provide per-subscriber QoS+*. It is crucial that different subscribers to the same sensor variable be able to have different guarantees in terms of latency, rate, and criticality/availability. If not, then a lot of bandwidth will be wasted: all subscribers will have to be delivered that sensor's updates at the most stringent QoS+ that any of its subscribers requires.

Guideline 4. *Provide rate- efficient multicast.* In order to achieve the highest throughput possible, it is imperative to avoid unnecessary network traffic. Thus, never send an update over a link more than once. Also, as a sensor update is being forwarded through the network, if it is not needed downstream in the multicast tree (e.g., those subscribers require it at a lower rate than other subscribers), the update message should be dropped which can best be implemented using an in-network *rate down-sampling* mechanism as is done in GridStat [19], [20].

These first four guidelines add up to a need for multi-cast routing heuristics that provide multiple, disjoint paths to each subscriber with each path meeting the subscriber's latency requirement. A family of heuristics developed for this multi-cast routing problem [21], [22] confirms the feasibility of the approach at the anticipated scale (see IG10) if routing decisions are made statically (IG11).

Guideline 5. *Provide synchronized rate down-sampling*. In providing rate down-sampling, it is important to not down-sample in a way that destroys the usefulness of some data. For example, synchrophasors are used to take a direct state measurement at a given microsecond. If some subscribers require only a small fraction of the updates for a set of synchrophasor sensors, it is important that the updates that reach the subscriber at each interval carry the same timestamp. For example, if a subscriber only requires a tenth of the updates from two different variables, then it would not be useable to get updates {#1, #11, #21, …} from one synchrophasor and updates {#2, #12, #22} from another synchrophasor, because the given measurements (e.g., #1 vs #2) do correspond to the same time (they are not the same snapshot), which is the main point of synchrophasors.

Guideline 6. *Don't depend on priority-based "guarantees".* Publish-subscribe delivery systems typically offer a way to specify a priority, so if the traffic gets too heavy less important traffic can be dropped. However, this does not provide a hard end-to-end guarantee to subscribing applications, and even applications that are not of the highest criticality still need their DRs to be met. Instead of priorities, mechanisms must be used that exploit the characteristics of WAMS-DD (as outlined in these guidelines) to provide each subscriber firm assurances that its guarantees will be met so long as there are not more than the agreed upon number of failures or severity of cyber-attack.

Guideline 7. *Provide end-to-end interoperability across different/new IT technologies (providing multicast, latency, rate, etc.).* A grid-wide WAMS-DD will *ipso facto* have to span many utility and network organizations. It is unlikely that the same mechanisms will be present across all these organizations. And, even if they are today, if the WAMS-DD gets locked into the lower-level APIs and semantics of a given multicast or QoS mechanism, it will be difficult to "ride the technology curve" and utilize newetter mechanisms that will inevitably become available over the long lifetime of the WAMS-DD. This is a stated goal of the GridWise community, for example [23]. Fortunately, it is possible to use middleware to span these different underlying technologies in order to provide guarantees that span this underlying diversity.

Guideline 8. *Exploit a priori knowledge of predictable traffic.* Internet routers cannot make assumptions or optimizations based on the characteristics of the traffic that they would be subjected to, because they are intended to be general-purpose and support a wide range of traffic types. WAMS-DD, however, have traffic that is not just rate-based, but is almost all known months ahead of time (e.g., when an engineering survey is made of a new power application). This common case can be optimized, as described in later IGs below.

Guideline 9. *Have systematic, quick internal instrumentation.* In order to provide end-to-end guarantees across a wide area despite failures and cyber-attacks, IG8 must be exploited to provide systematic and fast instrumentation of the WAMS-DD. This allows much quicker adaptations to anomalous traffic, whether accidental or malicious in origin.

Guideline 10. *Exploit smaller scale of the WAMS-DD.* This is a crucial if the challenging delivery requirements are to be met over a wide area with reasonable cost. However, this requires rethinking the conventional wisdom in networking research and commercial middleware products.

A WAMS-DD for even the largest electric grid will be orders of magnitude smaller in scale than the Internet at large[2], so it is feasible for the entire configuration to be stored in one location for the purposes of (mostly offline) route selection. Additionally, academic computer science researchers historically consider something that is $O(N^2)$ for path calculation with N routers or forwarding engines to be infeasible; see for example [18]. However, this assumption ignores two key factors for NnDB. First, N is not in the neighborhood of 10^8 as in the Internet, but rather is more likely ~10^3 at least for the next 5-10 years; this is even $O(N^2)$ algorithms are feasible at this scale. Second, as a rule, power engineers do not decide that they need a given sensor's values seconds before they really need it, due in part to the fact that today's data delivery infrastructure requires them to recode hard-coded socket programs and then recompile. Rather, power engineers plan their power contingencies (and what data they will need in them) months ahead of time with detailed engineering studies, and similarly for their monitoring, protection, control, and visualization needs. Thus, the routing/forwarding decisions involved in path selection can be done offline well ahead of time, while still allowing for handling a modest number of subscription requests at runtime.

It is also feasible for router-like forwarding engines to store state for each flow. Having a router keep per-flow state has long been considered a bane to networking researchers, because it is considered to be prohibitively unscalable. However, with the much smaller scale, and the much more limited type of applications for a WAMS-DD, storing per-flow state is not only feasible but it is a requirement for providing IG3 (per-subscriber QoS+) with IG4 (efficient multicast); this is something that our GridStat project has been advocating for many years [17]. However, recently networking researchers are realizing the necessity of storing per-flow state to provide any reasonable kind of QoS [25].

Guideline 11. *Use static, not dynamic routing and naming.* Much stronger latency guarantees can be provided when using complete knowledge of topology coupled with static routing. Complete topology knowledge is a reasonable assumption in a managed NnDB, given that it will be a carefully managed critical infrastructure with complete admission control. Also, almost all of the sensors and power applications will be known well ahead of time, so optimizations for static (or slowly-changing) naming can potentially be useful and can be done while still providing more flexible and dynamic discovery services at a much lower volume. We note that networking and security researchers generally assume that the membership of multicast groups (or a set of subscribers) may change rapidly; see for example [18]. However, as noted above, that is not the case with WAMS-DD.

Guideline 12. *Enforce complete perimeter control*. All traffic put onto an WAMS-DD must pass admission control criteria (permissions based on both security and resource management) via a management system: the publisher registering a sensor variable (at a given rate) and the subscribers asking for a subscription with a given rate and end-to-end latency. This is essential to provide guarantees at a per-message granularity. It also enables quicker adaptations. This should ideally be done in the publishing application's addresss space, via a middleware proxy, so that spurious traffic does not consume any network or router capacity.

Guideline 13. *Reject unauthorized messages quickly and locally*. Messages that have gone around the admission control perimeter should be rejected as soon as possible, ideally at the next WAMS-DD forwarding engine, rather than going most or all the way across the WAMS-DD consuming resources along the way. Detection of such unauthorized packets is an indicator of anomalous traffic and evidence of a failure or cyber-attack that needs to be reported to the management infrastructure. When sufficient evidence is collected over sufficient time, an appropriate adaptation can occur, but such rejection would be in practice reported to a management infrastructure given that it is an anomaly.

Guideline 14. *Provide only simple subscription criteria.* This is exactly the opposite of what is usually done with general purpose publish-subscribe in either academic research or commercial products: both tend to favor complex subscription criteria which are expensive to evaluate as each update is forwarded through the system (think of complex "topics") [26]. For example, in GridStat, the subscription criteria are latency, rate, and number of paths, and, as noted below, the forwarding decision is done completely based on rate, with static routing. Note

[2] For example, in the entire USA there are approx 3500 companies that participate in the grid [24]. We thus believe that the number of router-like forwarding engines that would be required for a NnDB backbone (at least in the case of broker-based publish-subscribe; defined later) is at most 10^4 and likely only around 10^3.

also that the lower-level ID of a sensor variable could still be looked up through a complicated discovery service; this guideline is concerned with avoiding complex forwarding logic.

Guideline 15. *Support only transient delivery, not persistent delivery*. Most publish-subscribe systems offer persistent delivery, whereby if an event cannot be immediately forwarded it is stored for some time and then the delivery retried. This harms throughput, however, as well as potentially the per-packet predictability (because it requires storing the data). In our experience it is completely unnecessary for real-time visualization, control and protection, due to the temporal redundancy inherent in rate-based update streams: the next update will be arriving very soon anyway, so the usefulness of a given update fades very quickly. Thus, it is inadvisable to complicate the critical paths of delivery mechanisms to support persistent delivery (though it can be provided "on the side" by other mechanisms). Furthermore, in the power grid, historian databases are already required for archiving data for regulatory reasons, so there is no reason to complicate the design or otherwise bog down the fastest and highest availability mechanisms of WAMS-DD to delivery historical data[3].

Guideline 16. *Don't over-design for consistency and (re)ordering*. Research in fault-tolerant multicast tends to provide different levels of ordering between updates from the same publisher, or between different clients of the same server, as well as consistency levels between different replicas or caches of a server. There is no need for anything like this in an WAMS-DD, Present data delivery software provides no kind of consistency at all, so power applications assume nothing in terms of consistency and ordering. The only requirement for such consistency that we have found is reflected in IG5 for synchrophasors, and the only ordering of any kind is where a PDC combines updates from different PMUs into one message to pass onwards. With devices such as synchrophasors that have accurate GPS clocks the order of events can be directly known and no delivery ordering mechanism is required other than that which is done by a PDC.

Guideline 17. *Minimize forwarding-time logic*. In order to provide the highest throughput, the forwarding logic that decides how a packet or update is to be forwarded on should be kept as simple as possible. On the GridStat project, forwarding decisions are made based solely on the subscription rate of subscribers downstream in the multicast tree [17], [20]. Given that the traffic is rate-based (IG2) and known ahead of time (IG8), and that subscription criteria are kept simple (IG14), and only transient delivery is supported (IG15), and that there are no consistency semantics (IG16), much logic can be pushed off to subscription setup time or even offline. This reduces the logic necessary when an update arrives at a forwarding engine (or P2P middleware mechanisms at an edge) and hence greatly increases throughput and decreases latency.

Guideline 18. *Support multiple QoS+ mechanisms for different runtime conditions*, beyond just network resources, in a unified, coherent way. A given mechanism that provides guarantees of latency and security, for example, will not be appropriate for all the runtime operating conditions in which a long-lived WAMS-DD may have to operate. This is because different implementations of a given QoS+ mechanism can require very different amounts of lower-level resources such as CPU, memory, and bandwidth [27]. Also, in a very big WAMS-DD, many QoS+ mechanisms will not be present everywhere, or close, so spanning them in a coherent manner and such that applications do not have to be aware of the above limitations is important. This is discussed more in Section 5.

Guideline 19. *Inspect only packet header, not payload*. In order to provide the highest throughput and lowest latency, ensure that subscription criteria and consistency semantics allow a forwarding decision to be based solely on a packet header. This is not possible for publish-subscribe middleware that has complicated subscription topics as is typical with commercial and research systems. For them, data fields in the payload also have to be inspected.

Guideline 20. *Manage aperiodic traffic*. Any traffic that is aperiodic (i.e., not based on rate but on a condition) must be isolated from rate-based periodic traffic and managed accordingly. This can be done deterministically, for example with (OSI Layer 1) optical wave division multiplexing (OWDM) hardware. Further, aperiodic traffic should be aggregated intelligently—for example, based on updateable policies rather than hardcoded settings—instead of sending all alarms/alerts to the next level up for processing.

[3] We note that such post-event historical data can be delivered by the same physical network links as the fast traffic with traffic isolation mechanisms; indeed, this is one of the main traffic categories for the emerging NASPInet.

2.3.3. Analysis

It is important to recognize that you can't have the highest level of all the properties described in the Design Requirements for every sensor variable. As noted in [1]:

1. Different properties inherently must be traded off against others.
2. Different mechanisms for a given property are appropriate for only some of the runtime operating conditions that an application may encounter (especially a long-lived one).
3. Different mechanisms for the same non-functional property can have different tradeoffs of lower-level resources (CPU, bandwidth, storage)
4. Mechanisms most often can't be combined in arbitrary ways

Even if you somehow could have them all at once, it would be prohibitively expensive. Given these realities, and the fact application programmers rarely can be experts in dealing with the above issues, middleware with QoS+ properties supported in a comprehensive and coherent way is a way to package up the handling of these issues and allow reuse across application families, organizations, and even industries.

Finally, because of length constraints it is not possible in this paper to fully discuss the cyber-security issues that arise in a WAMS-DD. Clearly, a WAMS-DD providing universal connectivity creates cyber-security challenges beyond those arising in a conventional, single-utility SCADA system. Cyber-security also interacts with DRs and IGs: for example, techniques used for message confidentiality and authentication must not impose too much additional latency, yet the multicast requirement appears to limit use of symmetric-key cryptography for authentication. See Sections 5.5 and 7.7 for more information, and of course the references cited therein.

3. ANALYSIS OF EXISTING TECHNOLOGIES FOR A WAMS-DD

We now analyze how existing technologies and standards meet the above DRs and IGs. We review in turn traditional network layer technologies, middleware technologies, and electric sector technologies. This section ends with Table 2, which very clearly depicts how almost all technologies are very inadequate for WAMS-DD, especially when closed-loop applications must be supported (or at least not designed out, either deliberately or inadvertently).

3.1. Technologies and Standards at the Traditional Network Layers

The "network" layers, i.e., ISO Layers 1-3, are where almost all WAMS-DD products and research to date have been drawn from. We analyze representative technologies and categorize them under subsections "Very Inadequate" or "Better yet Incomplete". Their limitations are very clearly summarized in Table 2 at the end of this section.

It is important to note that a number of these IGs have the appropriate network-level mechanism to implement the given IG, but they can't do it without help from higher layers: the middleware (or the application if middleware is not used) and also sometimes application management. This is to help parameterize the mechanisms with higher-level information outside the scope of the network level. This is discussed further in Section 5.

3.1.1. Very Inadequate

Traditional network protocols, including the OSI-2 "Data Link" layer (e.g., Ethernet), OSI-3 "network" layer (e.g., IP), and the OSI-4 "transport" layer (e.g., TCP, UDP, SCTP) do not provide end-to-end QoS+ guarantees or multicast [28], [29]. This is because they are at lower networking layers and end-to-end functionality is not their intended use. All of these lower-layer protocols can be part of the end-to-end solution that WAMS-DD sits above. Nevertheless, some systems do apply them in ways that are nearly end-to-end in scope, and therefore we now examine these protocols and extensions to them to see how they meet the requirements and guidelines if they were implemented as the complete end-to-end solution. We note that a major limitation of network-level solutions is that, based on our long investigations in the field, there is no other application domain that has anywhere near as severe of delivery requirements as the bulk power grid does.

IPv6 flow labels [30] associate each "reservation" with an application-to-application network socket connection, which contains many different sensor update streams with a wide range of required QoS+. Packets are processed in a flow-specific manner by the nodes that have been set up with a flow-specific state. The nature of the specific treatment and the methods for the flow state establishment are out of scope of the specification.

IP multicast (IPMC) provides efficient multicast for a single, non-replicated flow. However, if multiple IPMC addresses are used as a replication mechanism to attempt to provide redundant path delivery, there is no guarantee that the corresponding multicast trees will be disjoint, which is important not only for efficient multicast (IG4) but also for providing low latencies in the face of failures (DR5B). Further, IPMC addresses must be used carefully in even a mid-sized infrastructure; for example, due to instabilities it can cause it has been banned from many cloud computing centers [31], [32], [33]. Finally, IPMC also does not, by itself, have other end-to-end capabilities that are necessary for a WAMS-DD, as clearly shown in Table 2.

MPLS is designed to give Internet Service Providers (ISPs) a set of management tools for bandwidth provisioning, not to provide fine-grained (per-update) QoS [34]. Its guarantees are weak and statical compared to the needs of a critical infrastructure. For example, it gives aggregate economic guarantees over user, location, and protocol, not hard guarantees (DR1) for each update (DR5A). As noted in [35b], "**DiffServ does not guarantee that flows classified with a higher priority will really observe a better quality of service than lower priority ones"**. Given that **MPLS is a variant of DiffServ**, this is directly applicable. It is worth noting that in the relatively tranquil steady state MPLS may give extremely good latency and availability. However, in power contingencies, IT failures, cyber-attacks, or other situations that place stress on the WAMS-DD the guarantees are nonexistent.

Further, **different ISPs can implement MPLS in different ways**. There are no facilities for combining flows across different ISPs, as would be required in a WAMS-DD, or for predicting the end-to-end delays.

MPLS also has severe limitations with respect to the granularity of its "guarantees" and adaptation. The MPLS header only has 3 bits for the class field, so **each flow is assigned one class of only 8. In terms of managing flows, this is extremely coarse**. This is a far cry from providing guarantees, management, adaptation, etc. on a per-subscriber-per-publisher-each-update basis with hard guarantees. This is why MPLS meets few of the requirements and implementation guidelines, as shown in Table 2.

MPLS has some fault tolerance mechanisms, such as a fast reroute feature, detour merging, and end-to-end path protection. However, these mechanisms presently provide a minimum latency of about 50 ms, which is too long for the emerging SIPS and other applications described in Chapter 3. Finally, MPLS is a closed system whose performance can be matched by OpenFlow while having an open and extensible system (though, as shown below, one at present far from adequate for closed-loop applications) [35].

Virtual local-area networks (VLANs) and virtual private networks (VPNs) do not meet the DRs listed above because their purposes are orthogonal to the DRs. A VPN or VLAN could be part of a WAMS-DD, but VPN and VLAN technologies alone do not meet the requirements and can add to latency and decrease throughput.

Pragmatic General Multicast (PGM) is a transport-layer multicast protocol [36]. Implementation by Microsoft is known as Reliably Delivered Messages (RDM). PGM runs over a datagram multicast protocol such as IP multicast, to provide basic reliable delivery by use of negative acknowledgements (NACKs). PGM uses a rate-based transmission strategy to constrain the bandwidth consumed. However, it does not provide real-time guarantees. For brevity, it is not summarized in Table 2, but its coverage of the DRs and IGs should be clear from the technologies that it depends upon, which are covered there.

Spread can be considered a high-level multicast protocol that provides a range of ordering strengths across a wide-area network (WAN) [37], [38]. It supports ordered delivery and the resulting consistency, even in the face of network partitions, and it is used largely for replicating databases. It has no real-time mechanisms.

3.1.2. Better Yet Incomplete

Asynchronous Transfer Mode (ATM) and Synchronous Optical Networking (SONET) are networking technologies sometimes employed in WANs. They offer strong latency guarantees on a per-message basis. ATM does not support multicast (DR3) or multiple disjoint paths (DR4B). Given ATM's strong latency guarantees, however, at the right granularity, the ATM protocol can be part of a WAMS-DD that overlays ATM and other protocols. Of course, such an overlay network would by definition be middleware, which is needed for WAMS-DD for other reasons discussed in Section 5.

OpenFlow is a network communications protocol that gives remote access to the forwarding plane of a network switch or router [39], [40]. It does this by abstracting away common features of routers and switches to provide constructs such as flow tables and an instruction set that allows remote access to those constructs. Many vendors

have implemented OpenFLow on their products to varying degrees of completeness and consistency [41]. Finally, OpenFlow can implement all the features of MPLS (a closed system) and with similar performance in power settings [35].

One fundamental limitation **of OpenFlow is that it is *ipso facto* a lowest-common-denominator (LCD) approach**, having to be mapped to a wide range of devices. Very few of these have any mission critical requirements, let alone anything close to their LCD. Worse, **OpenFlow has no notion of global time**. While it is possible to provide some degree of predictable performance by very carefully managing flows, providing strong guarantees with tight deadlines and low jitter under difficult conditions is problematic at best. Finally, OpenFLow has no primitives that can do anything like rate downsampling even on a single flow (i.e., IG3 and IG4), let alone synchronized with another flow (IG5).

The standard internet integrated services (int-serv) has a guaranteed service (GS) that strives to provide strong delivery guarantees. It does provide hard E2E guarantees (DR1). However, it guarantees a maximum latency, not an average, so it has to be very conservative in what it promises. Further, it is not designed to provide low jitter, and it can be quite significant compared to closed-loop delay requirements (it can easily be well more than these requirements). Still, while it does not meet most of the DRs and IGs, using int-serv GS is significantly better than using MPLS.

The signaling mechanism of int-serv, RSVP, does to admission control. However, it is done in the router, not in the application's address space (as is simple to do with middleware proxies, which are in the application's address space), so spurious traffic can still slow down routers. Also, using this admission control with a fine enough granularity, i.e., to ensure that each publication from a given publisher applications does not go over rate, would be cumbersome with RSVP and take a large number of ports, potentially exhausting the 16-bit space of UDP ports.

Anagran technology [25], which is based on earlier Caspian technology, has flow-based routers that keep per-flow state to provide better QoS via flow management. It has not historically released much information on its router. The survey [35b] notes that Anagran technology provides only weak statistical guarantees, not strong ones as required for closed-loop applications. It also has no per-sensor, per-subscriber granularity, and nothing close: its flows use a 5-tuple in IPv4 (source and destination address and port, protocol) or an IPv6 flow label (see above) if available. This is the opposite of what is needed in critical infrastructures: to be unfair to provide strong guarantees! Further, Anagram technology automatically classifies flows into a number of classes, rather than being provided with the QoS+ requirements for each publisher and subscriber.

Anagran's corporate lifecycle may well be a lesson for the realities of finding QoS+ mechanism from other industries that are appropriate for closed-loop applications: the broader market was not such that QoS+ mechanisms stronger than MPLS but weaker than what was needed for closed-loop applications did not find a market big enough to sustain Anagran. It is worth noting that the company a had a lot of deep technical expertise and credibility, given that it was founded by Larry Roberts, the person who envisioned and commissioned the ARPANET in the late 1960s, and it was very well funded by investors.

Net Insight's Nimbra platform [42] comes much closer than any other network-level technology to meeting the DRs and IGs that closed-loop applications mandate, and is actually the one counterexample we are aware of regarding the negative lessons from Anagran's demise. Net Insight's primary market has been broadcast television backhaul and distribution networks, but its technology is very applicable to closed-loop applications: Net Insight uses tight time synchronization (without requiring GPS) to provide both very strong latency guarantees and stronger security, even with multicast delivery. Nimbra also has wrappers for some network technologies such as MPLS, SONET, etc. and presumably it also will for future networking technologies.

3.2. Middleware technologies and standards

There is a wide range of commercial, off-the-shelf (COTS) middleware frameworks providing different kinds of services with some relevance for a WAMS-DD. We first consider middleware supporting the pub-sub paradigm. There are two distinct architectures for pub-sub middleware, each with advantages and disadvantages.

3.2.1. Definition of Middleware

Middleware is a layer of software above the network but below the application that provides a common programming abstraction across a distributed system (computers connected by networks and communicating only

by message passing, or constructs built on top of it) [43], [44]. Middleware exists to help manage the complexity and heterogeneous mix of resource that are inherent in distributed systems. Middleware provides higher-level building blocks (these abstractions) which, compared to socket programming, can make code more portable, make them more productive, and have fewer errors due to handling the lower-level communications details. Indeed, comparing programming with middleware to socket programming is much like comparing programming with a high-level language such as C# or Java with assembler programming: relatively few people do it any more, and for very similar reasons.

Middleware masks the kinds of heterogeneity that are inherent in any wide area distributed computing system in order to enhance interoperability and make it much simpler to program. All middleware masks heterogeneity in network technology and in CPU architecture. Almost all middleware masks heterogeneity across operating systems and programming languages. A notable exception of the latter is Java's Remote Method Invocation (RMI), which is the remote invocation system for Java (note that the Java language is not middleware, RMI is). Java RMI does this so it can exploit features in the Java programming language such as serialization and reflection. A programmer can still use other middleware with Java, for example the Common Object Request Broker Architecture (CORBA) object oriented middleware or the publish-subscribe Data Distribution Service (DDS), both from the Object Management Group (OMG). Finally, some middleware does a very good job at masking heterogeneity across implementations of the same standard from different vendors; the OMG in particular does this well.

Middleware can also shield programmers of many (but not all) of the difficulties inherent in programming a distributed computing system. For example, it can largely hide from the programmer the exact location of an object, the fact that an object may be replicated or may relocate, etc.

It is for these reasons, and more, that **middleware has been considered "best practices" for 15-20 years in virtually any other industry** that has applications spread across a network. It is our hope that it becomes much more prevalent in power grids. Section 5 explains why middleware is required for any reasonable WAMS-DD.

3.2.2. Broker-based Publish-Subscribe Middleware

Broker-based (BB) pub-sub middleware systems rely on an infrastructure of broker nodes to forward messages toward subscribers. **The Data Delivery Plane (DDP) for a WAMS-DD, though not necessarily in many commercial systems, is a tightly managed WAN because it implements IG12 (complete perimeter control) and has extensive internal instrumentation (IG9).**

A BB pub-sub WAMS-DD is depicted Figure 2. A node in the DDP is called a Forwarding Engine (FE) and is a device specialized for the particular BB pub-sub framework. We depict the mechanisms that a BB WAMS-DD system can exploit in green; these consist of the FEs and the proxies in the same process as the applications (publishers and subscribers).

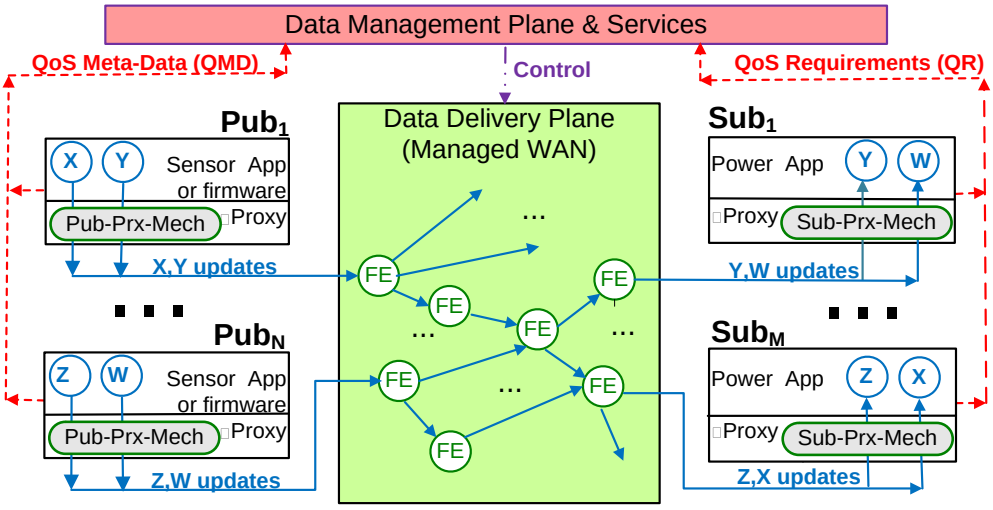

Figure 2: Broker-Based WAMS-DD

In BB WAMS-DD systems intended for mission-critical applications, there is often a separate "plane" for managing the system[4] and providing services. This Data Management Plane (DMP) is depicted in red in Figure 2; it is shown here as a single entity but is often distributed. Publishers provide the DMP with basic QoS meta-data (QMD) about their publications, e.g., the rate at which they will output updates. Subscribers provide QoS requirements (QR) including rate and latency. The DMP then exerts control over the DDP (depicted in purple) in order to provide the delivery guarantees, e.g., by updating a forwarding table for an FE.

BB pub-sub systems require a broker/server infrastructure to be installed; you can't just buy an IP router from Cisco or others. This can be a disadvantage, which often, for small and medium scales, cannot be amortized over enough applications to be justified. BB pub-sub systems have an advantage, however, in that they place intelligence inside the network, not just at the edges. This enables, for example, efficient multicast (IG4) and rate down-sampling throughout the data delivery system, not just at the edges. It also creates the potential to reject unauthorized packets at their next "hop" through the system (IG13).

Additionally, BB systems can exploit mechanisms in the graph of FEs in order to meet more of the IGs. For example, such an FE can be used to provide per-subscriber QoS+ (IG3), provide synchronized rate down-sampling (IG5), exploit *a priori* knowledge of traffic (IG8), and exploit the smaller scale of the WAMS-DD (IG10), i.e., it can contain per-subscriber state, a forwarding table entry for every subscription for which it forwards updates. An example of a BB WAMS-DD is GridStat.

3.2.3. Peer-to-Peer Publish-Subscribe

Peer-to-peer pub-sub systems place mechanisms for reliability and filtering only at the edges of an infrastructure. A canonical architecture for a peer-to-peer pub-sub configuration of a WAMS-DD is given in Figure 3. For the DDP, peer-to-peer systems typically rely on a combination of IP multicast and Ethernet broadcast to be as efficient as possible. Note that in this figure we omit the DMP, which is often not present as a separate core entity in peer-to-peer systems; the edge mechanisms collectively implement it.

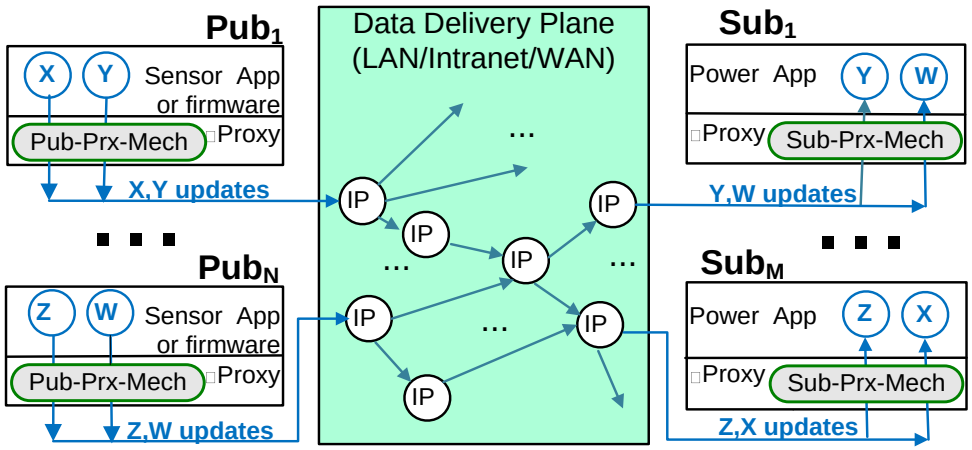

Figure 3: Peer-to-Peer WAMS-DD

One other thing to note in Figure 3 is that the controllable mechanisms for affecting traffic lie at the edges, in the proxies. Certainly a peer-to-peer WAMS-DD will exploit IP multicast as much as possible, but this has its limits, as described previously. Because its control mechanisms are at the edges, both QMD and QR are communicated to other proxies that collectively provide the delivery guarantees. Similarly, the only WAMS-DD-specific mechanisms are in the proxies, so control messages also must go there.

Peer-to-peer pub-sub systems have an advantage in smaller and medium sized deployments. However, for larger scales, the **lack of mechanisms in the backbone core for rate down-sampling and quicker fault tolerance, adaptation, and instrumentation prevent them achieving extremely low latencies in the presence of failures**.

[4] In telecommunications parlance, this is often called the "Control Plane," hence our use of the term "plane." Telecommunications nomenclature also refers to WASM-DD as the "data plane."

A federated combination of peer-to-peer and BB pub-sub systems has the potential to offer much of the best of both worlds. Here, peer-to-peer pub-sub systems are employed near the edges, i.e., within a single utility or sometimes within an ISO. Between utilities or ISOs, BB pub-sub systems are used in order to support higher throughputs and the lowest possible latencies over distance. A federated amalgamation of peer-to-peer systems would feature a globally unique namespace for variables, and utilities and could seamlessly pass messages with standardized wire and message formats [1].

3.2.4. Other Middleware

SeDAX is low-level middleware that provides both publish-subscribe and querying capabilities [45]. It is implemented with an overlay network on top of TCP/IP that is organized into a Delaunay Triangle network. SeDAX mainly targets the distribution side of the power grid and as such aims to handle many more hosts than is present in a mission critical WAMS-DD. Similarly, its availability requirements for flows are only 99.5%, which is far from the 6 9s required by DR4b (which comes from the Electric Power Research Institute in [5]). Finally, it does not provide sub-second latency let along strong latency guarantees. It is thus not appropriate for WAMS-DD.

Another category is called streaming queries (also known as complex event processing). It consists of a network of computer nodes that manipulate data streams through continuous queries in order to selectively propagate data, merge streams with existing data, or store data in a distributed database. Such systems are not designed to provide hard end-to-end WAN guarantees (DR1) with per-message granularity (DR5A) while tolerating failures (DR5B). Given their intended application domain, they also do not follow most of the IGs.

Finally, over the last decade or so crude middleware based on **web technologies such as HTTP, XML, and "web services"** has become popular in other industries, in large part because it is easy to get around firewalls with it. We note that scalability and throughput of such systems is highly questionable due to the many integration layers they typically add to make it possible to glue together just about any application to another [46]. Ken Birman, a leading expert in reliable distributed computing whose software has been widely fielded notes in [47] (emphasis is ours):

> *It doesn't take an oracle to see that the mania for web services, combined with such rampant online threats, contains the seeds of a future debacle. We're poised to put air-traffic control, banking, military command and control, electronic medical records, and other vital systems into the hands of a profoundly insecure, untrustworthy platform cobbled together from complex legacy software components.*

Unfortunately, one can add the smart grid to this list: a number of utilities and organizations see web services as a key enabling technology for the smart grid (for example [48], [49], [13]). This is unfortunate because it inadvertently rules out building a extremely low latency, extremely high availability, extremely high throughput WAMS-DD suitable for the more challenging closed-loop applications.

This is generally done because power engineers, or IT staff at utilities (which tend to be understaffed) buy into the hype because they do not understand the technology deeply.. But this begs a crucial question: is it really wise to have the operations of increasingly stressed power grids depend on a technology whose *raison d'être* is that to circumvent firewalls? We think not. *Caveat emptor!*

3.3. Existing Electric Sector Communications-Related Technologies and Standards

Middleware is rarely used in today's electric power systems, despite being considered best practices in many other industries for 15-20 years [1]. Further, the extreme conditions for closed-loop applications are far more stringent than in any other industry. Additionally, it is an unfortunate fact that almost all of the communications standards in the electric sector have been developed by power engineers or others who are not expert at building communications protocols, aware of the state of the art and practice, etc.

It is not surprising, then, that there seems to be no WAMS-DD technologies developed for the power grid that meet most or all of the DRs above. Part of this limitation is because commonly used power technologies are intended for a substation scope, with the only QoS+ "mechanism" being over-provisioning of bandwidth. When moving from a LAN to a WAN environment, there are implicit design decisions that cannot be solved by layering a new "WAN-appropriate" API over existing LAN-based protocols [27]. We now overview some of the more common power protocols and standards related to communications.

OPC-UA [53] was designed for substations. It uses TCP, which was not designed for predictable latency and does not support multicast. Subscribers and publishers "ping" each other to verify if the other is up, which not only does

not scale but also ignores best practices for pub-sub systems which have been well known by distributed computing researchers and practitioners since the 1980s.

IEEE C37.118 is a standard for synchrophasors that includes standard message formats. C37.118 is being revised to allow different data delivery mechanisms to be used. If successful, then C37.118 synchrophasor updates should easily be deliverable by any WAMS-DD transport.

The IEC 61850 communications standard was also designed primarily for applications associated with a substation automation system (SAS). It was conceived and created by protection providers in order to move data and information to, from, and among intelligent protection, control, and monitoring devices instead of legacy SCADA and RTU methods. .

IEC 61850's companion Common Information Model (CIM), IEC 61970 Part 3, can potentially be of use in a WAMS-DD, especially when the harmonization with C37.118 is completed, in particular in helping automate default QoS+ settings and perhaps adaptation strategies for a wide variety of sensors and applications that use them. This would be significant value added. Further, if IEC 61850 MMS and GOOSE APIs are successfully extended across the WAN, then IEC 61850 may well be able to successfully use a WAMS-DD transport. This would be of huge benefit to the many utilities that have a lot invested in IEC 61850 applications.

However, **this will only be true if the WAMS-DD transport for IEC 61850 is carefully designed to meet the DRs in Section 2.2. Unfortunately this seems doubtful** for a number of reasons, including:

- Presently, IEC 61850-90-5 targets delays of 50–500 ms, completely **rules out its use with closed loop apps** outside a single substation and also SIPS/RAS/SPS, including the ones from Chapter 3.
- Overall the standard seems to many observers with a computer science or software engineering background to be for more complex than it has to be, given the problem it is tackling. Indeed, to the authors it seems more like something that a mechanical engineer in 1975 would write, not a software engineer from 1995 (let alone today).
- Vendors do not, perhaps without any exceptions, implement the full standard. That makes **interoperability between vendors very problematic**, the marketing hype notwithstanding.
- There is no reference implementation and inter-vendor test suite, which makes true cross-vendor interoperability almost impossible because vendors have to test implementations pairwise, which is in turn problematic because, as noted above, few if any implement the full standard.
- There are **no tools for configuring** and otherwise using the standard that work well (or even fairly completely) **across different vendors' implementations**. This of course is very contrary to the goals (and hype) of the standard.
- It takes twice the bandwidth to deliver synchrophasor data as C37.118 does with no extra useful information.
- Subscriber applications have to be able to detect missing and duplicate data. Much better would be if middleware did this so that every application programmer (many of which are power engineers, or at least are not very familiar with distributed computing) has to do it themselves.
- GOOSE authentication originally mandated the use of RSA signatures. However, they are to slow for closed-loop applications even with a high-end PC, let alone a more modest embedded device in a substation (and one that cannot devote all its time to RSA calculations!) [50], [51]. There is no silver bullet here: approaches that use a shared key for each publisher and its subscribers are **vulnerable to a subscriber spoofing a publisher** if they do not have other help from other mechanisms.
- With 61850 in a substation developers and deplorers have to be very careful that the **multicast (which there is done via Ethernet Multicast) does not overload small embedded substation devices** (ones that don't even need to receive most or almost all of the traffic). Over a WAN, that kind of spamming is much more likely to cause bandwidth problems than CPU problems, which can cause other problems.
- It was **designed by a committee with a "kitchen sink" approach** before anyone had done close to a full implementation, which is very problematic. Indeed, internet pioneer David Clark famously stated about the internet's development and the Internet Engineering Task Force (IETF) that defines its protocols [52]:

  We reject: kings, presidents, and voting.
  We believe in: rough consensus and running code.

MMS does not have data delivery mechanisms. It can map onto the OSI protocol stack (which was not adopted in practice) and TCP/IP; see [28], [29].

An information architecture for the power grid is proposed in [54], which contains an analysis of 162 disturbances between 1979 and 1995. This paper demonstrates that information systems have an impact on power grid reliability and points out major deficiencies in the current communications scheme. The paper contains proposals for different ways to structure interactions between control centers and substations, and it also contains reliability analyses of

different schemes. However, it does not propose communication mechanisms and relies on off-the-shelf network technologies, which do not meet many of the DRs and IGs. Sadly, this is common, almost universal, practice in communications mechanisms and analyses in grids today.

3.4. Summary

The coverage of the DRs and IGs by various commercial and research technology is given in Table 2. The columns of this table are existing networking and middleware technologies, while the rows are the DRs and IGs outlined previously in this section. The table cells denote how well the given technology meets the given IG or DR. These have the following values: '**Y**': yes; '-': no; 'S': some; 'L': likely (but not confirmed); '?': unknown (documentation is not sufficient); 'D': doubtful (but not confirmed, because the product has insufficient documentation or it's a broad product category but not mission critical); 'F': future plans (architected for this); 'φ ': Not applicable (and thus, by definition, does not provide).

In some cases it is not possible to discern the exact coverage of a DR or an IG due to lack of detail in a research paper, product documentation, etc. However, rather leaving them unspecified, if we believe we understand the technology's goals, environment, etc, enough to speculate, we grade it as a 'L" or 'D' rather than simply a '?'. This is possible because it is highly unlikely that the developers of the mechanisms inadvertently over-designed their mechanisms to meet WAMS-DD requirements that they never were designed for (or even the designers aware of!).

Table 2: Coverage of Delivery Requirements and Implementation Guidelines by Existing Technologies. Note that '-' means the requirement is not met. Also note how few 'Y' entries there are in almost all columns.

Network-Level										Middleware						Elect-ric Sector			
Very Inadequuate					Better yet Incomplete														
TCP, UDP, SCTP	IPv6 Flow Labels	IP Multicast	MPLS	VLANs & VPNs	ATM	OpenFlow	Int-serv guar. svcs	Anagram	Net Insight Nimbra	SeDAX	BB COTS Pub-sub	P2P COTS Pub-Sub	Streaming SQL/CEP	Military RT Apps	SOA/web services	IEC 61850, OPC UA	DNP3, MMS	GridStat	Delivery Requirement or Implementation Guideline
-	?	-	-	-	**Y**	?	**Y**	-	**Y**	-	D	D	-	D	-	-	-	**Y**	DR1: Hard E2E WAN guarantees
-	S	-	-	-	-	S	-	-	S	S	**Y**	**Y**	D	**Y**	**Y**	-	-	**Y**	DR2: Future-Proofing
-	?	**Y**	**Y**	?	-	**Y**	-	**Y**	**Y**	**Y**	**Y**	**Y**	D	**Y**	S	-	-	**Y**	DR3: One-to-many (pub-sub or multicast)
																			DR4: Wide Range of QoS+:
-	-	-	**Y**	?	**Y**	S	-	D	**Y**	-	**Y**	**Y**	-	**Y**	-	-	-	**Y**	4A: Latency & Rate
-	-	-	-	-	-	D	φ	-	**Y**	-	**Y**	**Y**	-	**Y**	-	-	-	**Y**	4B: Criticality/Availability
S	-	S	S	?	-	φ	φ	S	S	**Y**	**Y**	**Y**	-	**Y**	S	-	-	**Y**	4C: Cyber-Security
																			DR5: Ultra-Low Latencies:
-	-	-	-	-	**Y**	-	-	-	**Y**	-	D	D	-	D	-	-	-	**Y**	5A: Per-message & predictable
φ	φ	φ	φ	φ	?	D	-	?	S	**Y**	D	-	-	-	S	-	-	**Y**	5B: Tolerating failures
φ	φ	φ	φ	φ	-	D	-	?	**Y**	?	?	?	D	S	-	-	-	**S**	5C: Tolerating Cyber-Attacks
Y	**Y**	**Y**	**Y**	?	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	-	-	-	**Y**	DR6: High Throughput
-	φ	-	-	-	**Y**	**Y**	**Y**	**Y**	**Y**	-	?	-	D	?	-	-	-	**Y**	IG1: Avoid post-error recovery mechanisms
-	-	-	-	-	-	-	-	-	S	-	?	?	D	?	-	-	-	**Y**	IG2: Optimize for Rate-Based Sensors
φ	-	-	D	-	-	S	-	-	S	-	?	?	?	?	D	-	-	**Y**	IG3: Provide Per-Subscriber QoS+
φ	φ	S	?	-	-	S	S	S	S	-	?	D	D	?	-	-	-	**Y**	IG4: Provide rate-efficient multicast
-	-	-	-	-	-	-	-	-	-	-	-	-	D	-	-	-	-	**Y**	IG5: Provide Synch'd Rate Down-Sampling
φ	-	φ	-	-	-	-	-	-	**Y**	φ	?	?	D	?	-	-	-	**Y**	IG6: Don't count on priority "guarantees"
-	-	-	-	-	-	S	φ	φ	S	S	**Y**	**Y**	S	**Y**	**Y**	-	-	**Y**	IG7: Provide E2E interoperability across different or new IT technologies (multicast, QoS+)
-	?	-	?	-	-	-	-	-	S	-	?	?	L	?	-	-	-	**Y**	IG8: Exploit *a priori* knowledge of traffic
-	φ	-	?	-	-	-	-	?	-	?	?	?	D	D	-	-	-	**S**	IG9: Have systematic, quick internal instrument.
-	-	-	-	-	-	-	-	-	-	-	?	?	D	?	-	-	-	**Y**	IG10: Exploit smaller scale of the WAMS-DD
-	-	-	-	-	-	S	S	S	**Y**	-	?	?	D	?	-	-	-	**Y**	IG11: Use static, not dynamic, routing
-	-	-		-	-	φ	S	φ	φ	-	-	-	D	-	-	-	-	**Y**	IG12: Enforce complete perimeter control
-	?	-		-	-	S	S	S	S	-	?	?	D	?	S	-	-	**Y**	IG13: Reject unauth. packets quickly & locally
-	**Y**	-	-	-	-	φ	φ	φ	φ	-	D	D	-	D	-	-	-	**Y**	IG14: Provide only simple subscription criteria
-	**Y**	**Y**	-	-	-	**Y**	**Y**	**Y**	**Y**	-	S	S	D	D	-	-	-	**Y**	IG15: Support transient, not persist., delivery
Y	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	?	?	D	?	S	-	-	**Y**	IG16: Don't provide unnecessary consistency
Y	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	-	?	?	-	?	-	-	-	**Y**	IG17: Minimize forwarding-time logic
-	-	-	-	-	-	S	-	-	-	-	?	?	-	?	-	-	-	**Y**	IG18: Support multiple QoS+ mechanisms for different operating conditions
Y	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	?	-	-	-	-	-	-	-	**Y**	IG19: Inspect only packet header, not payload
-	-	-	-	-	-	-	-	-	-	-	D	D	-	D	-	-	-	F	IG20: Manage aperiodic traffic

4. NASPINET

The North American Synchrophasor Initiative (NASPI) is a government-industry consortium dedicated to effective deployment of synchrophasors in the United States. In this section we describe its WAMS-DD initiative. Unfortunately, for a number of reasons including budgetery, regulatory oversight, and technical culture, as of late 2013 there has not been any deployments of anything resembling a full-features NASPInet, i.e., one implementing all DRs (and ideally most or all IGs) in Table 2. Nevertheless, it is an example worth studying.

4.1. Architecture

The NASPI network (NASPInet) concept is the only effort worldwide we are aware of that is dealing with end-to-end WAMS-DD issues at a more-than-superficial level. To support the use of synchrophasors, NASPI has been developing the notion of NASPInet (Nn), which has two main components, the data bus (NnDB) and the phasor gateway (NnPG).

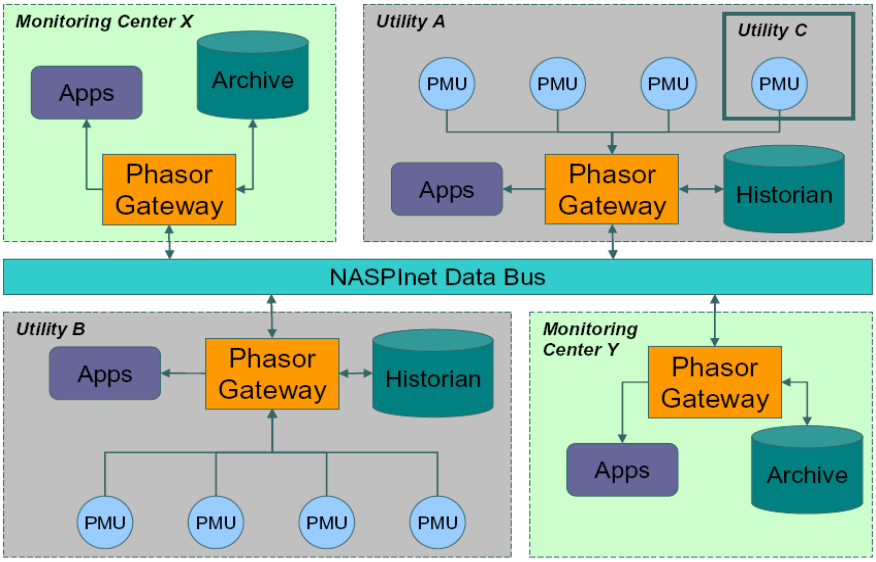

Figure 4: NASPInet Conceptual Architecture (courtesy NASPI)

The NnDB is the electricity version of what is sometimes called an enterprise service bus (ESB), which provides communications services for business-to-business exchanges. NnDB satisfies the DRs described earlier in this paper if it is implemented with appropriate technologies. The NnPG is the edge component of Nn, interfacing the utility or ISO to the NnDB. This is depicted in Figure 4.

Not that, while NASPInet has been originally conceived as being only between utilities (and ISOs/RTOs of course), it is not limited to this. The same technologies would work in a large utility that wanted to have much more control over its network and applications to provide WAMS-DD properties. This would be particularly important for closed-loop applications.

4.2. Service Classes

Five initial service classes have been identified for the NnDB in recognition of the fact that different kinds of traffic with different delivery requirements must be carried.

A. Feedback Control (e.g., small signal stability)
B. Feed-Forward Control (e.g., enhancing state estimators with synchrophasors)
C. Post-Event (post-mortem event analysis)
D. Visualization (for operator visibility)
E. Research (testing or R&D)

Table 3: NASPInet Traffic Classes and their Attributes [13]

NASPInet Traffic Attribute	Real-Time Streaming Data			Historical Data	
	Class A Feedback Control	Class B Feed-Forward Control	Class C Visualization	Class D Post Event	Class E Research
Low Latency	4	3	2	1	1
Availability	4	2	1	3	1
Accuracy	4	2	1	4	1
Time Alignment	4	4	2	1	1
High message rate	4	2	2	4	1
Path redundancy	4	4	2	1	1

Table key:

4: Critically important, 3: Important, 2: Somewhat important, 1: Not very important

Each class has associated qualitative requirements for such properties as low latency, availability, accuracy, time alignment, high message rate, and path redundancy. This is shown in Table 3.

Distinguishing the classes in this way is an important first step for a WAMS-DD system. The performance and availability requirements in Chapter 3 and the DRs and IGs in Section 2, can be considered a significant refinement of these NASPInet traffic classes. However, we note that the NnDB classes also consider the lowest required latency to be 100 ms, which is demonstrably far too high for some applications. This kind of application will only get more important and widespread as grids get more stressed each year.

One issue with the class definitions is that a customer, such as a utility, ISO, RTO, or NERC, one cannot specify only what it wants from a telecom provider, e.g., a "Class A" network and be done with it. This will not necessarily result in a WAMS-DD that meets the requirements across multiple traffic classes. For example, if too much traffic of "easier" classes is on the network, then you will not get Class A guarantees. Rather, to provide the DRs identified above, one needs to do resource management within the data delivery service. Application management and network management components must account for all traffic associated with each subscription using a given level of QoS+. This is part of enforcing a number of IGs, including IG8 (exploit traffic knowledge), IG9 (systematic, quick, internal instrumentation), IG12 (complete perimeter control), IG13 (reject unauthorized packets quickly and locally), and IG20 (manage aperiodic traffic). Further, implementing this requires middleware, as we discuss next.

We note that**, despite the US Government spending more than $4B of federal fiscal stimulus money on the power grid over the last 5 years, there has been no significant progress in deploying anything even vaguely resembling a full-featured NASPInet**. There are a number of reasons for this. First of all, advanced communications is something that organizations are not bureaucratically set up to deal with. Second, power engineers and researchers almost always assume that communications is good enough. Third, through the imperfect and power-centric bureaucratic process, through no fault of its own the company hired to write a NASPInet specification, while excellent at power issues, had zero experience with anything but the old hard-coded grid communications; nobody working for them has even a BS in computer science, or even an undergraduate minor, despite the fact that WAMS-DD is very advanced computer science.

Further, even the most ambitious communications upgrade as part of this investment is not only not using publish-subscribe middleware, it is not even using IP multicast. This was because they (the WISP project in the WECC grid in western North America) had so many problems getting vendor software running with 200 PMUs rather than the 10 PMUs it was used to, that they had no time for significantly modifying the communications. They know that

with just point-to-point communications they will not scale to many more PMUs. Further, they are only using MPLS to provide QoS. As noted above, it is very deficient for WAMS-DD, and they are *de facto* ruling out any closed-loop applications.

As demonstrated in this chapter, **a full-featured WAMS-DD is very possible, but it has to be done by computer scientists leveraging the state of the art in real-time distributed computing, not by power researchers and engineers**, or for that matter by software vendors repackaging an existing publish-subscribe infrastructure from other domains. The pressure is building, however: grids are getting more stressed each year for a host of reasons that can be mitigated by a proper WAMS-DD.

5. WHY WAMS-DD MUST INCLUDE MIDDLEWARE NOT SOLELY NETWORK LAYERS

In this section we explain how implementing a WAMS-DD requires the use of middleware. In general, by the stated goals of the smart grid community middleware is required [1] (this paper, whose first author is this chapter's first author, won a best paper award for demonstrating this proposition). Further, is simply not possible to implement a WAMS-DD without information provided by middleware, sometimes augmented by application monitoring. In other cases, it is possible to build some features and properties without middleware, with middleware's help the lower-level mechanisms can do a much better job. In general, the network level only knows about packets and IP addresses, not middleware-layer application variables (e.g., published sensor variable updates). It also knows nothing about the power applications that subscribe to these updates, and their importance in different situations. For example, the GridStat's systematic adaptation described in Section 7.5 cannot possibly be implemented without help above the network layers.

In this section we consider why middleware in general is required for WAMS-DD. We next describe how it is needed when QoS+ is required, as of course it is in any WAMS-DD. We next discuss how middleware is needed to properly implement some of the DRs and IGs. Finally, we discuss how middleware is very helpful in integrating legacy systems.

5.1. Middleware in General

Middleware is required to accrue the benefits overviewed in Section 3.2.1. Additionally, in practice, in any system of moderate size and complexity there will be products from different vendors. While they may implement the same protocols, their configuration and management may well be different. For example, even router vendors will admit that the way you configure IPMC is different across vendors. Thus, if for no other reason, most such systems will have at least a thin layer above the network for applications to use.

A utility therefore has three choices regarding this:

1. Have each application programmer recreate its own thin layer (these are sometimes called "application-level protocols" by network researchers)
2. "Roll your own" middleware in-house, and
3. Use commercial middleware

Application programmers already have enough to deal with, so making them also handle #1 is highly inadvisable (but common practice in the electricity sector, regrettably). Rolling your own middleware is certainly better than #1 because it allows the implementation of the thin layer to be shared across application programs. However, utilities may not have sophisticated network programmers to do this, and if they do most likely their time could be spent better on other tasks. This leaves us with #3: commercial middleware. Here experts in networking, middleware, QoS+ mechanisms, interoperability, etc create middleware that is reused over many companies and projects. And, as noted in Section 3.2.1, programmers then are more productive and the code they write is more portable and manageable.

5.2. Middleware and QoS+

When QoS+ properties are required, as of course they are in a WAMS-DD, then it is even more important to use middleware. Without it, application programmers have to combine lower-level QoS+ mechanisms in arbitrary ways. For example, **how is an application programmer going to know how to combine network QoS from MPLS or int-serv guaranteed services with IPSec and non-repudiation?** These mechanisms were designed

independently of each other and interactions between them can be very subtle. It is thus a really bad idea to assume that application programmers can combine them in reasonable ways knowing any inherent tradeoffs or limitations of their combination.

By far the better way to provide QoS+ properties in a WAMS-DD is for a middleware developer to combine multiple mechanisms, evaluate and measure their combination, document any tradeoffs and limitations the application programmer should be aware of, etc.

In addition to providing QoS+ properties in the steady state, there are additional reasons why WAMS-DD requires middleware: coordinated adaptation driven from the top down. In the face of failures, cyber-attacks, or other runtime IT problems, it is extremely undesirable to have different mechanisms adapting independently of each other and without any higher-level guidance. For example, having ACK and NACK messages clog the WAMS-DD is a really bad idea. It is much better to have coordinated, end-to-end adaptation driven by knowledge of the applications via knowledge captured in applications' middleware.

5.3. Middleware and DRs and IGs

As mentioned previously, a number of the DRs and IGs can't be implemented properly, or even at all, without the help of middleware. These include the following:

- **DR2** (future proofing): applications that use middleware can exploit new and better QoS+ mechanisms as they become available, because the applications are not locked into lower-level specific QoS+ mechanisms. The applications do not need to be reprogrammed or recompiled, just a new version of their middleware library linked in.
- **DR5B** (ultra-low latency tolerating benign failures): providing this high level of fault tolerance without jeopardizing low latency guarantees requires knowledge of the applications using the WAMS-DD, not just socket-level information on packets.
- **IG2** (optimize for Rate-based sensors): In order to optimize for rate-based sensor data delivery it is necessary to know the rates that the different applications require. QoS-enabled middleware APIs capture this, while network-level mechanisms are generally unaware of application rates.
- **IG3** (provide per-subscriber QoS): similar reasons as **IG2** above.
- **IG4** (provide efficient multicast): similar reasons as **IG2** above.
- **IG7** (provide E2E interoperability across different/new IT technologies (multicast, encryption, authentication, network-level QoS, etc): network-level mechanisms don't cover the entire space here, and with few exceptions can't generally be composed with other kinds of network mechanisms providing the same property (e.g., latency) without a higher-level layer above them both, which is of course by definition middleware.
- **IG12** (enforce complete perimeter control): Network QoS mechanisms generally do not know much about the traffic delivered on them, including their rates. Further, if over-rate traffic comes to a router that knows the rate, it is still consuming resources on that router rejecting it. Middleware with QoS APIs can of course capture rate and other QoS_ parameters. Further, middleware generally use a proxy object that is returned when the application connects to the middleware. The proxy object is then what the application (publisher or subscriber) uses to send and receive sensor updates, respectively. This proxy object can do rate-based policing to ensure that no more than the guaranteed rate escapes the application and taxes the first hop across the network.
- **IG18** (support multiple QoS+ mechanisms for different operating conditions and configurations): In a big WAMS-DD no single mechanism for providing a QoS+ property such as lantency will be present everywhere in the WAMS-DD. Sometimes the path from publisher to subscriber has to traverse multiple QoS+ mechanisms, for example MPLS and Nimbra. Similar to IG7, a layer is needed above the network layer to sensibly connect these different mechanisms providing the same property, and to know the limits and tradeoffs inherent in their combination.
- **IG20** (manage aperiodic traffic): *Aperiodic traffic*, also called event-triggered or condition-based, must obviously be isolated from rate-based traffic (also called *time-triggered* or *periodic*). This can be done at the network layer. However, it is also necessary to manage that traffic with knowledge of the data and applications using the data. This is because in many circumstances there may be more data than can fit in the bandwidth allocated to it. Fortunately, there is often a lot of redundancy in this

traffic (e.g., a large number of alerts/alarms sounding from a single root cause, or fairly unimportant messages that can be logged and ignored, at least for some time and/or in some circumstances). Doing this management requires middleware to capture semantics of the data and application. It also requires higher-level management to intelligently aggregate the data using policies or other high-level mechanisms, but this requires the information that middleware provides.

5.4. Middleware and Legacy Systems

Middleware is often used not to write complete systems from scratch, but rather to integrate legacy systems that were not designed to interoperate well with different kinds of systems. It does this by adding a middleware wrapper, which surrounds the legacy subsystem and maps down from a high-level middleware interface down to the interface specific to a given subsystem. When middleware wraps a legacy system in such a way, then the APIs and also the management infrastructure can be written at a high-level consistently across the different legacy subsystems, because the middleware shields them from the lower-level details of the individual legacy systems. This allows the subsystems to interact in a very high lowest common denominator (the middleware layer and its high-level abstractions), with strong type checking, and allowing the use of exception handling mechanisms across different programming languages. This is of great benefit.

As an example, client-server distributed object middleware such as CORBA is often used to wrap legacy servers. This way a client written in Java or Python can invoke a server object on a server written in C#, C, or FORTRAN, even though the last two languages are not object-oriented.

The previous example involved client-server interactions, but the technique of wrapping legacy subsystems with middleware also applies to publish-subscribe infrastructures such as WAMS-DD. It can be highly beneficial to do so, because communications infrastructures for utilities can be a jumbled mess of communications subsystems that were not written to work well together (not only due to the relative backwardness of the industry on IT matter, but also because many of the bigger vendors believe that inter-vendor interoperability can provide a path for them to lose their lucrative vendor lock-in). Such wrapping is a necessity, not just to be able to create a coherent WAMS-DD out of these legacy subsystems, but also because in the future no one mechanism for a given QoS+ property (such as latency) is ever going to "take over the world" and completely displace all others. Given the large investment in the legacy systems, and the fact that (at least if utilized correctly) they can still provide significant value, it will always be the case that there are multiple subsystems to try to seamlessly interoperate across

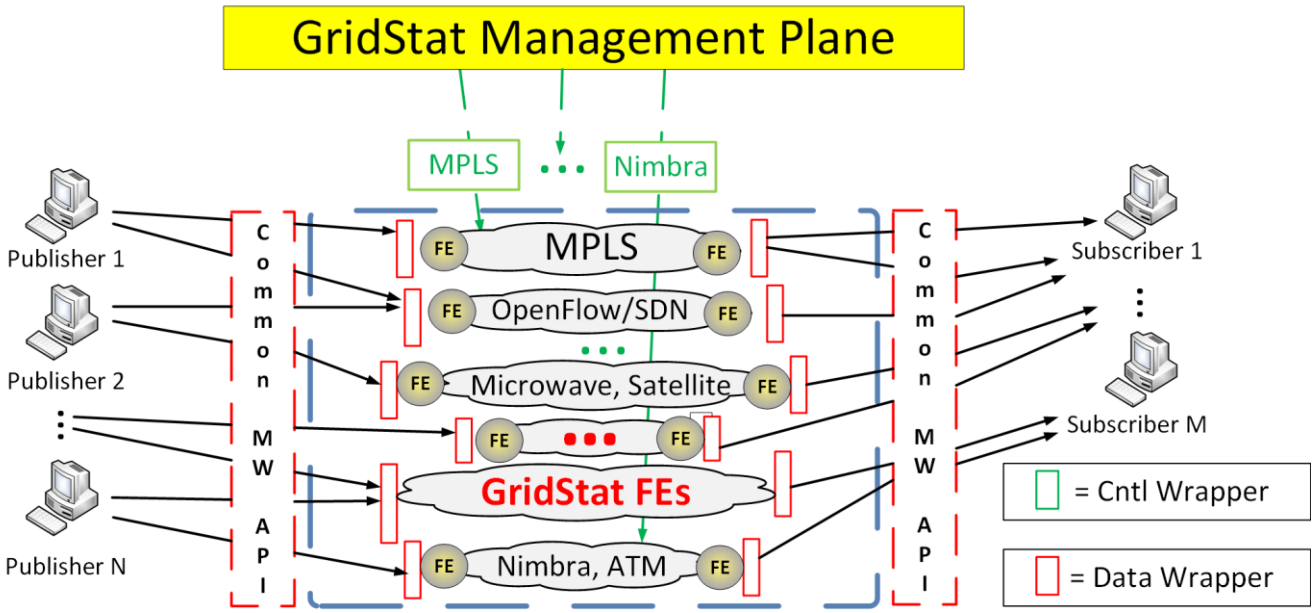

Figure 5: Middleware Integrating Legacy and New Communications Subsystems

[Notice OpenFlow & MPLS & ATM ☺]

We now give an example of this in Figure 5. Here an existing utility communications infrastructure includes a large number of legacy protocols such as **MPLS, microwave links, and others (the "…" could include BPL/PLC, satellite links, and even 4G cellular**, for example to remote substations where its impractical to run fiber or copper). This existing infrastructure has been strengthened by programming some of the existing routers using OpenFlow and by adding stronger guarantees with subsystems of GridStat Forwarding Engines and Nimbra devices. This could in turn be managed as a coherent whole by GridStat's management plane.

In this configuration, non-GridStat subsystems look to GridStat like a single link between two Forwarding Engines. However, the management plane can exhibit control over them to strengthen QoS via a control wrapper that has a high-level middleware control API and the wrapper then maps down to the subsystem's specific control APIs (the control wrappers and communications are shown in green). Finally, the data wrappers at the boundaries of the subsystem not only provide a common, high-level middleware API but could also be used to do rate-based policing similar to as described above using middleware proxies.

Given such a high-level coherent WAMS-DD a management plane such as GridStat's**, the different subsystems can be treated as a unified whole**, and route across different subsystems, taking into account not only the performance characteristics of the subsystem but also its reliability and of course considering aggregate bandwidth. Further, the subsystems in Figure 5 may have fairly different failure characteristics and security vulnerabilities, which can greatly help the overall end-to-end data availability from publisher to subscriber. The reader may be curious why old microwave links, which have much lower bandwidth and often higher drop rates, would even be included. However, they have very different failure characteristics than most of the other subsystems and this can add a lot of value, so long as their maintenance is not becoming cost-prohibitive. These links do have much less capacity than the other technologies, so only the most important traffic must be sent over it, perhaps at a lower rate. Of course, this requires middleware to capture these semantics for the management system.

Indeed, it is crucial to note that **even if a WAMS-DD has only weak QoS mechanisms such as MPLS or even simple IPv4, it will still need the management and policing, wrappers, etc. that is shown on the edges below**. If it does not have this, then things may work well in the calm steady state, but when things get bad the predictability of the latency and other properties is extremely suspect. In short, when you need it the most you could count on it the least. This is a potentially disastrous way to implement WAMS-DD for closed-loop applications.

Sadly, this is the situation at almost all utilities today. **The only real QoS "mechanism" is massive over-provisioning of bandwidth. This works quite well most of the time in the calm steady state, but when there are benign IT failures, let alone cyber-attacks, then the WAMS-DD is likely to perform quite unacceptably**. This in itself could be a major vulnerability: terrorists can not only try to change actuator and relay settings to cause problems in the grid, they may also attack the WAMS-DD so that it is much more difficult for utilities and ISOs/RTOs to understand what is going wrong and fix it before a large blackout is caused. Indeed, such attackers would of course attack when the grid is already severely stressed. The conditions under which a grid will be stressed, and what power assets can cause the most problems if changed, is relatively easy for a determined adversary with a long-term view (e.g., Al Queda) to ascertain long ahead of time.

5.5. Middleware and Avoiding QoS Stovepipe Systems

Finally, we close this section with a system-level admonition that is crucial for the long-term benefits that WAMS-DD can provide, if done right. A *stovepipe system* is a legacy system that is comprised of different parts that are very difficult to combine or refactor. They have been identified as major problems in big systems 20-30+ years ago, and much software engineering and system architecting has been devoted to avoid them on many newer systems (at least ones in other industries that are not far behind best practices as the electricity sector is).

But, given that WAMS-DD has to provide QoS+, and mission critical nature of its applications, stovepipe systems have another dimension. [1] provides the following definition:

QoS Stovepipe System (QSS): a system of systems whose subsystems are locked into low-level mechanisms for QoS and security such that

a) It cannot be deployed in many reasonable configurations, or
b) Some programs cannot be combined because they use different lower-level QoS mechanisms for the same property (e.g., latency) that cannot be directly composed, or
c) It cannot be upgraded to "ride the technology" curve as better low-level QoS and security mechanisms become available.

Of course, it is crucial that WAMS-DD installations avoid QoS Stovepipes, and middleware, while no "silver bullet", is a key enabling technology for this. It is for this reason (and others from this section) that many military programs in the US and elsewhere require the use of QoS-enabled middleware; examples can be found in [1].

6. SECURITY AND TRUST FOR WAMS-DD

One of the US Department of Homeland Security's Strategic plan objectives for the fiscal years 2008-2013 has been to protect and strengthen the resilience of the nation's critical infrastructure and key resources. WAMS-DD, being a middleware that could be deployed in a critical infrastructure such as the power grid, must provide support for key security services (authentication, confidentiality, integrity, availability) not only at the system level, but at the data level as well. There are many points of interactions among a variety of participants and a local change can have immediate impact everywhere. Thus, to cope with the uncertainties associated with security, there must be provision for a trust service that assesses the trustworthiness of the received data that comes through nontrivial chains of processing. Trust, when used properly, has the potential to enable the entities that operate within critical infrastructures to share confidential, proprietary, and business sensitive information with reliable partners.

Even though WAMS is becoming one of the popular technologies for upgrading the electric power grid, **still there is no consensus on a security standard or even a set of comprehensive security guidelines that will guarantee the secure generation, distribution, and consumption of data.** It is a nontrivial and rather challenging task to provide trustworthy and secure interactions in such operating environments, where private, public, and national entities must collaborate in a way that information is not compromised due to either malicious or accidental attacks. However, there is research done on the security aspects of an envisioned power grid communication infrastructure that could be used as a basis to form the security requirements for WAMS-DD. It is out of the scope of this chapter to map the security spectrum for WAMS-DD.

One of the first research efforts to define the security, trust, and QoS requirements in a data delivery middleware tailored for the power grid is described in [55]. Starting with the application requirements for a variety of interactions (e.g. communication between substations and control centers, communication between control centers, etc.), the authors proceed with specifying the implications of the requirements for security, trust, and QoS.

The work in [56] complements the aforementioned specifications by presenting a conceptual flexible trust framework that models an entity's trust as a relation whose state gets updated as relevant conditions that affect trust change. Trust is expressed as a set of security, behavioral, and QoS properties. The novelty of the model is its ability to integrate trust reasoning and monitoring into a single model by extending the traditional concept of trust conditions into more expressive expectations, which include not only expected values for particular properties but also covering, aggregating, and triggering mechanisms that manipulate the observed value. In addition, and unlike other approaches, the model derives end-to-end trust assessments without using transitive indirect trust explicitly in the derivations. A recent work by [57], presents a theoretical model based on Bayesian decision theory that automatically assesses the trustworthiness of entities in the electric power grid.

One of the few published works on WAMS security is the one in [58]. The authors interviewed the owners of two WAMS networks in order to identify the primary implementations of WAMS, and thus address security challenges in the light of each implementation. The three WAMS implementations are an isolated WAMS network, an isolated WAMS that interfaces with the control center network, and finally WAMS is fully integrated into the control center network infrastructure. An analysis of WAMS security concerns is presented accompanied with a number of threat scenarios. The vulnerabilities identified by the authors are addressed in terms of these three implementation approaches because their impact differs in each case.

7. GRIDSTAT

GridStat is a data delivery service designed to support the DRs discussed in this chapter. Its research results have influenced the shape of NASPInet [4]. The GridStat research started in 1999 by looking at the QoS+ requirements of innovative power applications being developed by power researchers and analyzing closely what the state of the art in applied distributed computing systems could support. After significant gaps were identified, the detailed design and then development of the GridStat framework began in 2001.

GridStat is a BB pub-sub system that meets all of the DRs from this chapter except for 5C: Tolerating Cyberattacks, which it does to a modest degree (by redundant paths); covering this more completely is near-term future research and will exploit the systematic adaptations described in Section 7.5. GridStat also implements all but three of the IGs, which have similarly been planned for and are also near-term future research.

In this section we first provide an overview of GridStat. Next we describe GridStat's novel mechanisms that allow different subscribers to be delivered different QoS+, which is crucial for a WAMS-DD. We then describe condensation functions, a mechanism that allows arbitrary computations to be placed in a WAMS-DD configuration. After this we describe GridStat's mechanisms for rapidly adaption its subscription base. Finally, we describe GridStat's remote procedure call mechanism and then overview its security and trust infrastructure.

7.1. Overview

7.1.1. Capabilities

The architecture of GridStat is given in Figure 6. The architecture is modular and it consists of two modules, namely the *data plane* and the *management plane.* Each of the modules is responsible for part of the overall functionality of the framework.

GridStat's data plane is responsible for delivering the sensor data. A single sensor datum is called a *status variable*, and can be subscribed to by multiple subscribers. The data plane consists of *publishers*, which generate the sensor data, *subscribers*, which consume the data, and a graph of *Forwarding Engines (FE)*, that are specialized middleware-layer routers ("brokers" in Publish-subscribe parlance) that deliver the status variables.

GridStat's management plane is responsible for managing the data plane, including making admission control decisions. It is organized into a hierarchy of *QoS brokers (QB)* that is designed to map directly onto the geographically-based hierarchies that the power grid and other critical infrastructures are normally organized into. A *leaf QB* is the lowest level in this hierarchy and is responsible for, and directly connected to, a collection of forwarding engines that are organized into a single administrative domain called a *cloud*. Brokers can host policies for resource allowance (how much bandwidth an organization or app can receive), security permissions (potentially default ones overridden in some cases), adaptation strategies, data aggregation, and specifying other configuration

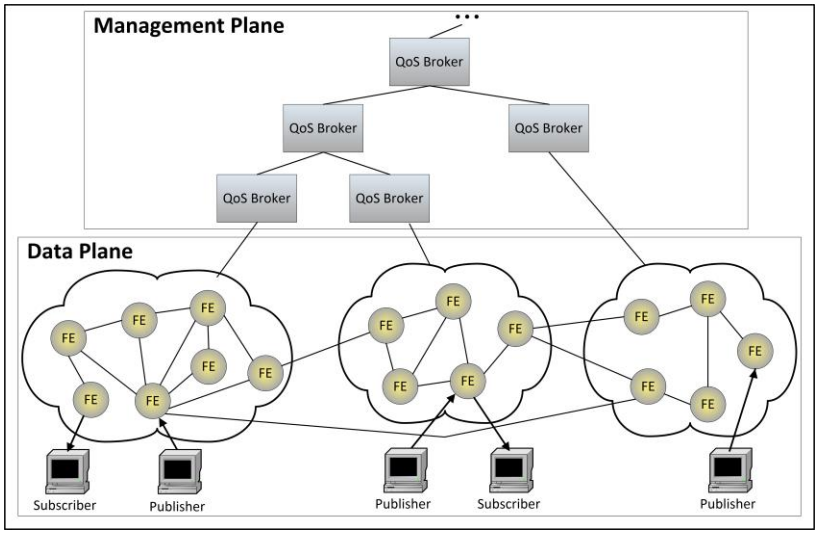

Figure 6: GridStat Architecture

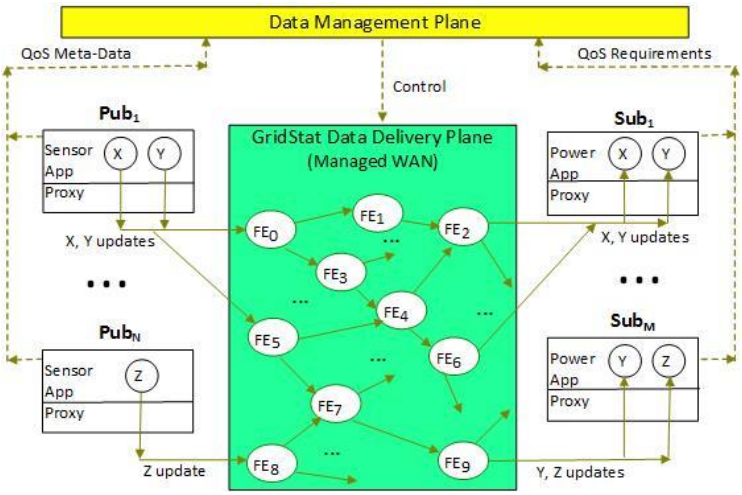

Figure 7: GridStat Forwarding Engine Operation

GridStat's data delivery semantics are novel in three broad ways in order to better support WAMS-DD. First, they are a specialization of the publish-subscribe paradigm tailored to leverage the semantics of periodic streams of data updates. For example, in a normal publish-subscribe system, an event that is being forwarded from a publisher to its subscribers cannot be arbitrarily dropped, because it is in general impossible to know how it will affect the application programs that subscribe to it. However, in GridStat the subscribers' required rate, latency, and number of redundant (disjoint) paths are part of the API to subscribe to a given status variable. With this knowledge, the management plane configures a forwarding engine so it can discard updates, an operation we call *rate filtering*, when downstream subscribers have gotten an update recently enough to satisfy their latency and rate requirements. This can potentially save a large amount of bandwidth, because many sensors in the power grid produce updates with a hard-coded rate that is deliberately conservative, i.e. at a rate above what its engineers believed any application would require. The absence of rate filtering means that each subscriber has to subscribe to a status variable at the highest rate that any subscriber has to subscribe at, which is for example what IP multicast requires. With many subscribers, and only a few needing high-rate updates, this can be extremely wasteful in a WAMS-DD.

The second novelty is that GridStat implements the synchronized rate down-sampling (IG5), a requirement that GridStat researchers first described and what is necessary when doing rate filtering on synchrophasor data. The third way in which GridStat's data delivery is novel is that different subscribers to a given status variable can be provided with different rate, latency, and number of redundant paths. *We call this QoS+ heterogeneity*. This allows a power device such as a relay that is physically very close to the publisher to be given a high rate and low latency, while remote subscribers that only need to loosely track the variable can be given a much lower rate and higher latency. If this flexibility were not supported, then the remote subscribers would have to receive the high rate of delivery that the device in the same substation is given, thus consuming unnecessary resources. This is what IP multicast requires, because there is no rate filtering of any kind.

These operations are depicted in Figure 7, where the management plane is not shown as a hierarchy for simplicity. When updates of any publication such as Y arrive at a FE, it knows Y's publication rate and the rate at which each of its outgoing links needs Y at. Consider the case where Y is being published by Pub_1 at 120 Hz and where Sub_1 requires it at 120 Hz but Sub_2 requires it at 30 Hz. At the FE (in this example this could be FE_4) where updates to Y split off on different links towards Sub_1 and, Sub_2, the former link will pass on all updates while the later will drop ¾ of them. Also, if Sub_1 receives Y over two different paths, an update will traverse a different set of FEs to reach it. In the example about the first path would be FE_0 -> FE_1 -> FE_2 and the second path would be FE_0 -> FE_3 -> FE_4 -> FE_2. In a proxy (piece of middleware code beneath the application) duplicates updates for Y will be dropped.

We are aware of no system that has such broad coverage, or close of the DRs and IGs, as should be clear from Table 2. Systems that provide either application-driven rate filtering or QoS+ heterogeneity are uncommon, and we know of no system that provides both. Yet, as is shown above, in Chapter 3, and in [59], leading-edge and emerging power applications can greatly benefit from having such capabilities.

7.1.2. Performance and Scalability

On a 2010-era PC in Java, GridStat adds less than 0.1 ms per overlay hop. Because, in any real deployment of GridStat, this would be dominated by the speed of light, it is essentially free, adding no more than 1 msec over the speed of the underlying networks, because there would be no more than about 10 hops. This PC version can handles ~500K forwards/sec at each forwarding engine. This could be further improved if we made the assumption that rates that can be subscribed to are power-of-two multiples of a base rate; e.g., 15, 30, 60, … Hz. Further, given how simple the rate-based forwarding logic is and how inefficient Java is at high-performance I/O, a C version (which is planned for the very near term) should do 10-20X the Java version, or roughly 5-10M forwards/sec. This is likely more than adequate for the backbone needs of major grids for at least 5 years, given the slow growth in inter-utility sharing.

On 2003-era network processor hardware, GridStat adds ~0.01 ms/hop and scales to a few million forwards/sec [60]. Using 2013 era network processing hardware (which is at least ten times more scalable than 2003's versions) and a small cluster, these results should easily be extendable to achieve 50–100M forwards/sec, while rejecting unauthenticated messages, monitoring traffic patterns, and checking for (and reporting) evidence of intrusions and cyberattacks. Custom hardware implementations in FPGA or ASIC would likely be highly parallelizable as well as "unrolling" the logic in the software FE's two-deep loops. It is thus likely support significantly higher than 100M forwards/sec., and likely up to line speeds on a 2.5Gbps or faster fiber link. This is possible because the logic in a GridStat FE is simpler than a general purpose IP router: it's doing less deliberately! And it is very likely more than adequate for the wide-area needs of power grids for many decades to come.

7.2. Rate Filtering for Per-Subscriber QoS+

In order to get an operational view of a wide-area system, its monitoring and control system must be able to deliver a snapshot of the state of the entire system (or at least a snapshot of a set of interesting measurements.) There are two requirements in order to achieve this. First, the measurements must be taken at the same time. Second, as described above, if any rate filtering of the measurements is performed by the communication system, the temporal relationship between the events must be preserved.

GridStat's forwarding logic provides deterministic multicast rate-filtering of event streams for temporally related status variables. This feature is provided by the underlying data structure and the routing algorithm that the FEs are using, hence it is transparent to the end-points and always activated for all events. In addition, there is no need for any coordination between the FEs or intervention from the management plane. The multicast feature is a side-effect of the data-structure used in the FE, and as a result no extra computational resources are needed. The computational resource usage by the filtering algorithm is further analyzed in [61].

7.2.1. Forwarding Algorithm

The forwarding algorithm is performed for each event that is received by a FE. The message format includes a header that stores the variable id and the timestamp when the specific measurement was taken. For each subscription that is established over a specific FE, the following information is included during the subscription setup command: variable id, published interval, subscribed interval, and outgoing link. With this information the data structure for the forwarding algorithm is built (for more info see [61]).

The three steps that are taken during the event forwarding are the following:

1. The first step is to add to the event's timestamp half of the publishing interval, i.e. shift the event's timestamp by half the publishing interval to the right.
2. From the result of Step 1, take the mod of a subscription interval. Note that this is done for all the unique subscription intervals that there are subscription path for this specific variable.
3. If the result of Step 2 is less than the publisher interval, then the event is forwarded.

The three steps can be illustrated with the following pseudo code:

```
if ((event.TS + ceil(pubInt/2) % subInt[0..i]) < pubInt)
     forward event
else
     drop event
```

7.2.2. Example of the Forwarding Algorithm

The example in Figure 8 illustrates how events from four variables (p1, p2, p3, p4) that publish events every 50ms will be forwarded through an FE for a subscriber downstream that subscribed to receive the set every 100ms. Due to various reasons, the events from the different publishers are published with slightly different timestamps.

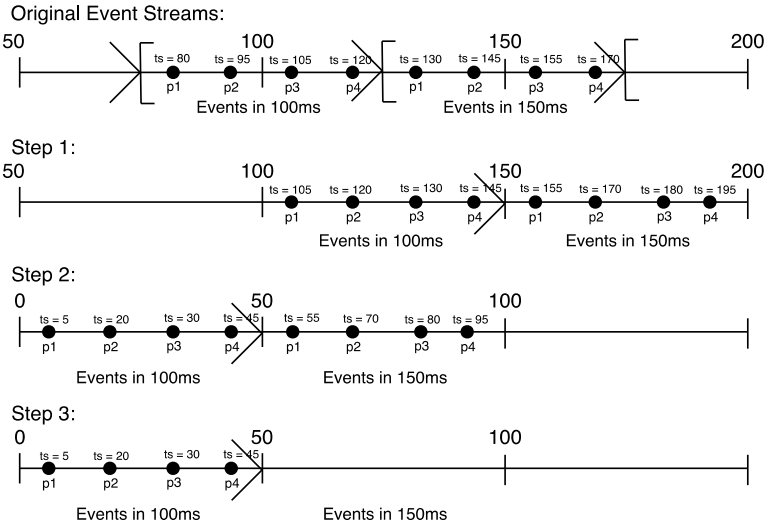

Figure 8: Illustration of the Filtering Steps

According to the algorithm, the following will occur in the FE:

1. Adding 25ms (publication interval divided by 2) to the timestamp of each of the events. After the addition, the timestamps that belong to the 100ms group will have a timestamp of 100ms to 149 ms.
2. The subscription interval is 100 ms, and that means the events in the group with timestamps 100, 200, 300, etc. should be forwarded. Taking the mod of the result of Step 1 results in events with timestamps 0 – 99.
3. After step 2, there are two phases with events from the publishers because subscription divided by publication interval equals 2. The framework will always pick the first phase, so the result of Step 2 is checked to test if it is less than the publication interval (the first phase). If this is the case, then the event is forwarded; if not, it is dropped. Events with original timestamp value between 75 and 124 will be forwarded, while the others (timestamp value between 125 and 174) will be dropped.

7.3. Condensation Functions

In an environment where multiple entities must cooperate in order to maintain the stability of the system, there will be some common patterns that the various entities will be interested in. Without support from the control and monitoring system, these patterns will have to be determined at the end-points. If the determination for these patterns could be provided as a service by the monitoring and control application, then a number of benefits can be observed. The most obvious benefit is the reuse of application logic, but also reduction of computational and network resource as the computation will be performed only once close the source and shared to all the interested parties.

GridStat provides the *condensation function* mechanism that allows an end-user to specify a transformation of multiple raw event streams into a new stream during transmissionThis transformation allows the end-user to migrate application logic into the middleware layer, with great flexibility. Thus, GridStat condensation functions can be considered a WAMS-DD version of a Java plugin.

7.3.1. Condensation Function Architecture and Capabilities

A condensation function consists of four modules, as shown in Figure 9: input filter, input trigger, calculator, and output filter, with input filter and output filter modules being optional.

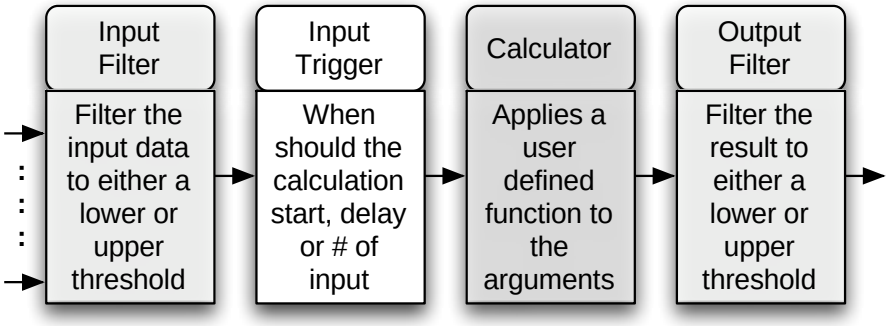

Figure 9: Condensation Function Modules

The input filter module filters status events from the input event streams. The filtering process compares the status event value with the upper and/or lower pre-defined thresholds and decides whether to drop the event or not.

The trigger module is responsible for receiving events from the input status variables and initiating the calculation function. For each of the input variables, there is an entry in the data structure where the trigger module stores the latest value from each of the variables. The calculator module is initiated by one of the three built-in triggering mechanisms (others can be added). The built-in triggering mechanisms are: time triggered, number of received events, and number of alert events received.

The calculator module must be provided by the end-user. The user will have to overwrite two pure virtual methods namely, the Initializer method (initializes the object), and the Calculator (transforms the input to an output).

Last, the output filter is similar to the input filter with the difference that it filters the result produced by the calculator. The trigger module implements the same interface as an outgoing link. What this entails is that during the routing process an event routed to a condensation function has the same interface as if it were routed to any of the outgoing links. As a result, the condensation function does not interfere with the flow of the routing process.

7.3.2. Condensation Function Development

The GridStat framework provides a GUI tool that assists the end-user in defining a condensation function. If the input stream or the produced output stream uses other than the build-in data-types, an IDL compiler will be used to unmarshal and marshal the events for filtering and the calculation.

The placement of the condensation function within the data plane is the responsibility of the management plane. The GUI tool sends the specification for the condensation function to the leaf QB, through an FE. The leaf QB will then find an appropriate FE and place the condensation there. In addition, it will set up all the subscription paths that are the input for the condensation function as well as register the resulting event stream as a first class variable, just like all the other publications.

More detailed information on the condensation function can be found in [62].

7.4. Operational Modes

Routing algorithms that disjoint path routing with additive constraints (latency) are computationally very complex. It would in practice be a very bad if there were thousands of new subscription requests in a crisis: that would likely be an inadvertent but effective denial of service attack on the subscription infrastructure. **Fortunately, power grid operators do not choose random sensor variables to start observing during a power contingency. Rather, they have (paper) checklists for each power contingency and/or mode which were created months or years ago by engineering studies.**

GridStat's mode mechanism exploits this fact to enable its subscription base to react very rapidly. It allows multiple forwarding tables to be preloaded in its FEs then rapidly switched when appropriate [63]. This action is called a *mode change*. Depending on the GridStat deployment, FEs can utilize several forwarding tables corresponding to the operating modes, while inactive forwarding tables lie dormant until their corresponding mode is activated.

GridStat also supports a hierarchy of modes; we overview them now but for more details see [63]. QBs also inherits mode sets from its ancestors in the hierarchy. A QB will have one set of forwarding tables for each mode set, and will have exactly one mode in each mode set active.

7.4.1. Mode Change Algorithms

A mode definition consists of an ID, a name and a set of data plane subscriptions (a forwarding table), and is owned by a single QB in the management hierarchy. Modes defined and owned by a QB constitute a mode set, and exactly one of the modes in a mode set is active at any time; that's the operating mode. This means that every QB always has exactly one of its modes active, which could be a default and unnamed mode if no modes are defined. A QB that operates in a mode assures that all +subscriptions (and parameterizations thereof) contained in the subscription set of that mode are active in the data plane by using the forwarding table corresponding to the active mode.

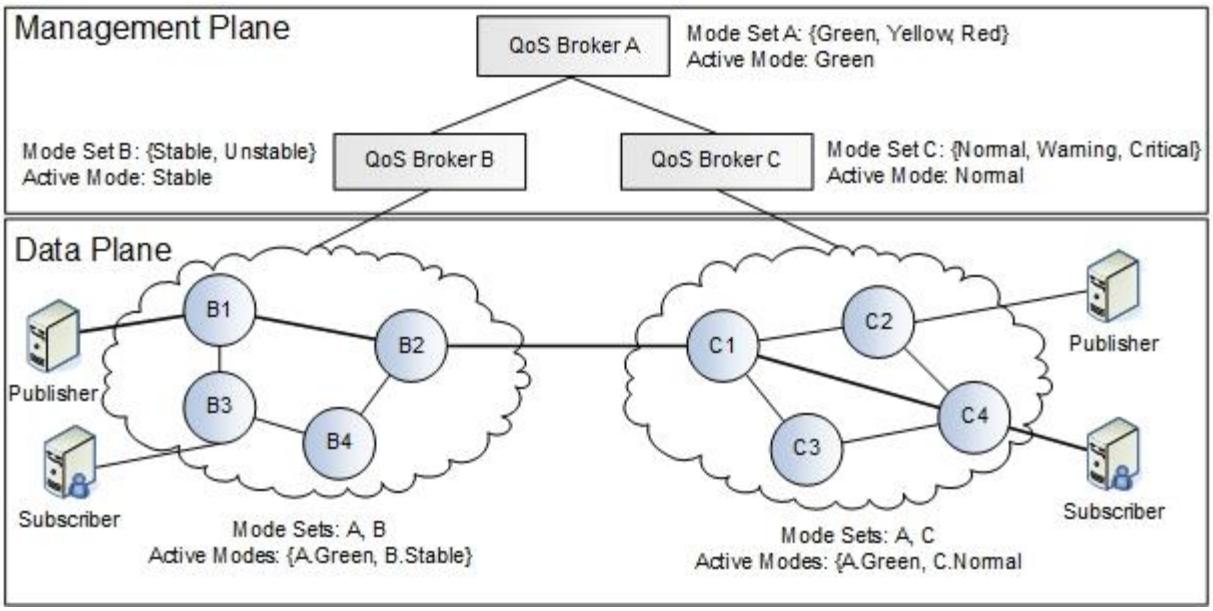

Figure 10: Mode Example

A QB can quickly switch forwarding tables in the data plane through modes. Figure 10 shows how operating modes are used in both the management plane and in the data plane. In this example, QB-A is the parent of both QB-B and QB-C.

QB-A defines a mode set that contains `{Green,Yellow,Red}`. However, it has no cloud directly under its control, so this mode set is used to influence forwarding in its children, B and C. The active mode at this time for QB-A is `Green`. If it later changes its mode then FEs belonging to its descendants will change forwarding tables.

QB-B defines the mode set with labels `{Stable,Unstable}`, and its active mode presently is `Stable`. Thus, QB-B will utilize forwarding tables for both `A.Green` and `B.Stable`.

QB-C defines the mode set with labels `{Normal,Warning,Critical}`. Its active mode is `Normal`. Thus, QB-C will utilize forwarding tables for both `A.Green` and `C.Normal`.

A mode change operation that switches the forwarding tables in all of the involved forwarding engines at the expected time is called a consistent mode change operation. Otherwise, the operation is called an inconsistent mode change operation, and additional recovery mechanisms must be utilized in order to restore the operating modes on the forwarding engines that are considered inconsistent. An inconsistent mode change operation will most likely result in some subscribers not receiving all the events that they have subscribed to, or receiving a few updates to a new mode while still receiving updates that should only come in the old mode. However, these glitches will only happen for the duration of the mode change, which is very small.

The recovery mechanism is provided in order to tolerate some degree of network failures and to eventually ensure consistent mode change operations. The recovery mechanism is triggered by the QB that detects missing mode change acknowledgments, and attempts to resolve these situations when possible.

Two mode change algorithms have been implemented in GridStat, namely the hierarchical mode change and the flooding mode change. The hierarchical mode change algorithm uses the management hierarchy to disseminate mode change operations and gather acknowledgements from forwarding engines and QBs. The hierarchical mode change algorithm enables all subscriptions registered to operate in the coordinator's current and new mode to flow during the entire mode change. Thus, subscribers with subscriptions registered in both the current and new mode continue to receive status events during hierarchical mode change operations that switches between those modes.

The flooding mode change algorithm disseminates mode change operations directly out on the data plane by using the limited flooding mechanism in GridStat. When a forwarding engine receives the operation it forwards the operation on all outgoing event channels, except the event channel from which it received the operation. As forwarding engines may receive multiple copies of the same operation, redundant copies are discarded. FEs are informed to change from the current mode to the new mode at some future timestamp.

The two mode change algorithms provide different tradeoffs. The hierarchical mode change algorithm is a resource intensive algorithm which is split into five message phases in order to enable transferred subscriptions present in both modes in a mode change, e.g., from Green to Yellow, to flow. A message phase indicates that the coordinator has to initiate and propagate a mode change phase (message) down to all forwarding engines in its hierarchical scope, and the next phase cannot be initiated until the previous phase has completed. The coordinator must receive aggregated acknowledgements from all forwarding engines and QBs. The flooding mode change algorithm, on the other hand, is a best-effort algorithm and disseminates mode change operations directly out on the data plane where forwarding engines are informed to switch at a predetermined future time. The flooding mode change algorithm is efficient, in terms of resource usage and performance, but does not guarantee any subscriptions to flow during the mode change operation. The flooding mode change algorithm relies on the forwarding engines' ability to switch modes at the exact same time and therefore requires all forwarding engines to be time synchronized.

We now provide more detail on both of the mode change algorithms.

Hierarchical Mode Change Algorithm

As mentioned earlier, the hierarchical mode change algorithm is divided into five distinct phases that enable transferred subscriptions to be present in both modes of a mode change operation. Furthermore, the five phases eliminate any FE overload scenarios during a hierarchical mode change operation.

The following steps show how the hierarchical algorithm affects the FEs in a mode change from Green to Yellow:

1. The inform phase: The FEs that have subscribers attached to them, informs the subscribers about the upcoming mode change.
2. The prepare phase: The FEs that have publishers connected to them switch to the temporary forwarding table Green ∩ Yellow. The highest subscription interval (lowest rate) of transferred subscriptions issued in this phase in order to reduce the load on downstream FEs. This phase ensures that subscription traffic that belongs in both modes (Green and Yellow) is forwarded through the FE network. This step only eliminates subscriptions, and hence cloud resources cannot get overloaded.
3. The internal change phase: FEs that has neither publishers nor subscribers connected to them switch to Yellow's forwarding table. Since all other FEs operate in a temporary forwarding table and only forward a smaller set of subscriptions (in mode Green and Yellow), the FEs can safely switch to mode Yellow without overloading any FEs downstream.
4. The edge change phase: The FEs that are currently in the temporary forwarding table Green ∩ Yellow will switch to Yellow's forwarding table.
5. The commit phase: The FEs inform their subscribers about the completed mode change.

Flooding Mode Change Algorithm

The flooding mode change algorithm disseminates mode change operations directly out on the data plane by using the limited flooding mechanism in GridStat [64]. The leaf QB that is responsible for the specific cloud initiates the flooding by using its embedded publisher. When a FE receives the operation it forwards the operation on all outgoing event channels, except the event channel from which it received the operation. As forwarding engines may receive multiple copies of the same operation, redundant copies are discarded. Forwarding engines are informed to change from the current mode to the new mode at some future timestamp. Upon receiving the mode change command, a forwarding engine immediately responds with an acknowledgment to its leaf QB. It will then activate the new mode at the destined future timestamp.

The flooding mode change algorithm offers better statistical delivery guarantees to the data plane, than the hierarchical mode change algorithm. This is because of the amount of redundant paths in the data plane and is thus more resilient to network failures than the hierarchical mode change algorithm. Whereas the hierarchical mode change algorithm attempts to preserve the subscriptions registered in the two involved modes, the flooding mode change algorithm switches directly to the new mode, even though some of the FEs have not acknowledged that they will do the switch.

7.4.2. Overview of performance

The two algorithms presented above are quite different with respect to complexity. This is also reflected in their performance. In an experiment setting with three layers of QBs and five clouds with five forwarding engines in each cloud, with the packet loss varied from 0% to 8%, with the minimum consecutive packet loss varied from 0 – 3 and the maximum consecutive packet loss varied from 0 – 5 the following were observed: The flooded mode change algorithm reached all forwarding engines in approximately 45ms, while the hierarchically algorithm required 1200–3200ms depending on the link loss and burstiness setting. This was observed when the link latency was set to an overly-conservative 8ms for each of the links in order to emulate a wide-area power grid (the speed of light in fiber or copper is roughly 125 miles/msec); this includes the links between the QBs as well as the links between the FEs. More detailed information about the performance of the mode change algorithms can be found in [63].

7.5. Systematic Adaptation via Data Load Shedding with Modes

The mode change algorithms described above, coupled with other information that can easily be captured by middleware and an application management system, provides powerful capabilities in helping a WAMS-DD to not only to adapt to IT problems, but to also optimize its benefit for the power problems being dealt with at any given time, not just the (usually calm) steady state.

Electric utilities do *load shedding*, whereby they cut off power to some customers as a last resort to try to avoid a blackout. We call this *power load shedding.*

There is a communications analog for WAMS-DD: *data load shedding*. For example, GridStat's QoS+ APIs are for rate, latency, and #paths. However, they require not only the desired amount (which would be provided in the steady state) but also the worst case that the application can tolerate. This implicitly gives permission to throttle back some flows from their desired QoS+ towards the worst case if necessary, by reducing the rate or the #paths, or both. This can help the WAMS-DD adapt to fewer functioning resources being available due to benign IT failures or malicious cyber-attacks. But it can also help free up capacity in order to add new subscriptions specific to a given contingency. For example, dozens of new synchrophasor sensors could be subscribed to, the rates of some existing subscriptions could be increased, and a few feeds from high-rate digital fault recorders added (at 720 Hz or more). This could help operators have a much better understanding of what is happening in the grid at that moment. This could be done very rapidly: in a fraction of a second not only are the data feeds changed but also visualization windows popped up to alert operators and give them quick insight into the ongoing power problems.

Given GridStat's semantics, such **data load shedding can easily be done in a systematic way that dynamically optimizes (or at least greatly improves) the value of the WAMS-DD to the utility based on present operational circumstances**. A very simple scheme would be where different subscriber applications are tagged with their importance (think of a number from 0 to 10, for example), not only for the steady state but that application's importance in different contingencies. This data can then be used to decide which subscriptions get throttled back towards their worst case with any of a number of ways. This adaptation could further be optimized using more complex *benefit functions*[5]. With benefit functions, an application (or system integrator) would specify how much benefit a given level of QoS+ provided the application; again, for simplicity, think of a number from 0-10 here. Examples of a benefit function specification, where individual list items are (rate:benefit), might be {(<30:0),(30:3), (60:7), (≥120:10)}. This means that there is no benefit to the application of a rate less than 30 Hz, no added benefit by going above 120 Hz, and the values between have the benefits listed.

[5] Benefit functions are more commonly called *utility functions*, but we use the former term in the electricity sector to avoid confusion.

7.6. Remote Procedure Call (RPC)

GridStat's one-way delivery mechanisms are sufficient for delivering updated values from remote sensors. However, they are inadequate for a round-trip remote procedure call (RPC), for example to an actuator in the power grid or between control centers.To overcome this limitation of the one-way delivery mechanism, the Ratatoskr round trip RPC mechanism is built over GridStat [65]. It is a highly tunable RPC mechanism tailored to the needs of the power grid. The RPC mechanism implemented is a simple invocation of remote subroutines without any object-oriented semantics, which provides a familiar programming environment and lend itself well to serial programming. More importantly, the mechanism proves the feasibility of a two-way control system on top of the GridStat framework.

7.6.1. Mechanism Details

Ratatoskr is a two level protocol stack that is built on top of the GridStat network (see Figure 11). The bottom level is the 2WoPS protocol that provides for two-way communication, while the top level provides an RPC mechanism (Ratatoskr RPC). The 2WoPS protocol utilizes the QoS semantics provided by the GridStat network (spatial redundancy), and it introduces two additional redundancy techniques: temporally redundant sends and ACK/retries. Ratatoskr RPC provides extensive customization of the redundancy levels for each call, which provider for various tradeoffs.

Further, pre- and post-conditions built into the call semantics provide additional safety mechanisms for application designers. These are described after we first explain the details of the RPC protocol

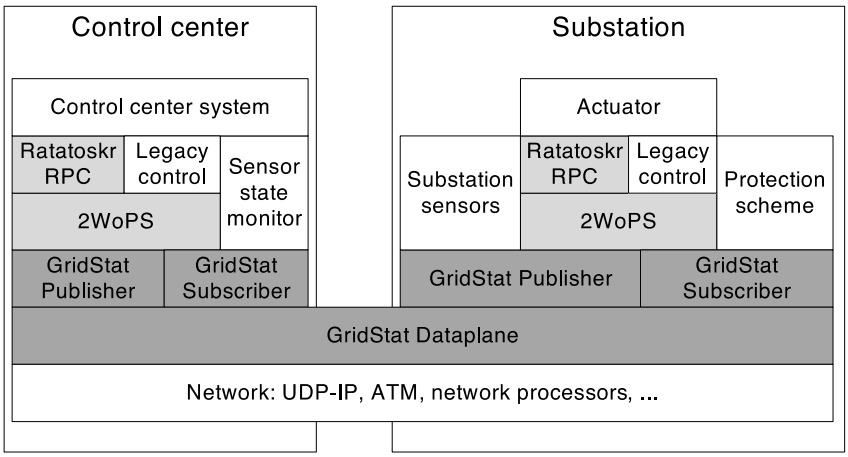

Figure 11: RPC Module Stack

7.6.2. The 2WoPS Protocol

To achieve a two-way communication style needed for RPC signaling, Ratatoskr uses two separate GridStat publish-subscribe connections. Each RPC host utilizes both a publisher and a subscriber - the publisher for sending data as published status variables and the subscriber for receiving data. Currently, the 2WoPS protocol supports only communication between two hosts, but the GridStat framework has support for multicast of status events so future versions could add one-to-many and group communication styles.

The protocol aims to provide a high degree of fault tolerance while retaining predictable delivery times. To this end, three redundancy techniques are employed to strengthen delivery guarantees. These are:

1. *Spatial redundancy*: GridStat provides QoS that a subscription stream should be forwarded through multiple disjoint network paths. The 2WoPS protocol uses this feature to provide special redundancy for its communication.
2. *Temporal redundancy*: While spatial redundancy provides protection from a range of network failures, the network costs and delay properties of a large number of paths could prove prohibitive in cases where a very high degree of fault tolerance is required or where the network topology is unsuited for spatial redundancy. Ratatoskr offers temporal redundancy by allowing each data unit sent by the 2WoPS protocol to be repeatedly published without waiting for a round-trip. A data unit published n times has a *temporal redundancy* level of n and tolerates $n-1$ losses.
3. *ACK/resend*: While spatial and temporal redundancy provide a high degree of fault tolerance, they do not provide any feedback to the application about successful delivery, and do not provide protection from failures that affect the whole

network, such as periods of heavy congestion. Ratatoskr provides a third technique for redundancy by having message receivers ACK successfully received 2WoPS messages, and allowing the user to specify that messages that do not produce an ACK should be resent up to *n* times. A message that is set to be resent up to *n* times when ACKs are missing has an *ACK/retry level* of *n*.

7.6.3. The Ratatoskr RPC Mechanism

Ratatoskr RPC (RRPC) is a remote procedure call protocol for power-grid control communication built on top of the 2WoPS protocol. It draws extensively on the features provided by the 2WoPS protocol. Delivery guarantees for calls is achieved using GridStat's QoS enabled network communications.

In addition to increased safety through fault-tolerance, pre- and post-conditions on calls are built into the RPC semantics. Pre-conditions are conditional expressions over GridStat status variables that are evaluated at the server (a power grid actuator) before execution of an RRPC call, and if the expression is false, the call to the physical actuator is not executed. Post-conditions are similar expressions evaluated after the call has executed, and negative results are reported back to the application via exception. This can provide a physical confirmation that the actuator succeeded even if all of the redundant reply messages are lost. This helps circumvent a well-known limitation in distributed computing—that in an asynchronous system, even with only one failure, a client can not know if its request was processed by the server or not [66]—by using a physical feedback loop. It also handles the case of a failed actuator, where the firmware returns a successful return code but the physical device is stuck open or closed.

An example of pre-condition in power-grid operation could be the control of an isolator. Isolators are actuators that connect and disconnect de-energized power circuits. A pre-condition could be to verify that a line is de-energized before attempting isolation. An example of post-condition could be the control of a load tap changer. Load tap changers are components of certain transformers that allow adjustment of output voltage. A post condition could be to verify the new voltage level after load tap changer operation, or even to generate a status report from all connected devices and send it back to the client.

7.7. Security and Trust in GridStat

As discussed in Section 6 above, security and trust are important for the operation of any information infrastructure supporting the power grid, and there are multiple dimensions upon which they must be addressed. **GridStat research has led to implementation within the framework of mechanisms for securing and authenticating published messages [67] as well as mutual authentication between components of the GridStat system [68], in novel ways appropriate for a WAMS-DD**. Both mechanisms are built using an architecture for *composable and replaceable security modules* under the control of security managers operating in cooperation with QBs. This architecture was motivated by two principal facts. First, the long lifetime of power grid IT infrastructure strongly suggests that security implementations developed today will not remain secure over the decades that the system will be deployed. Second, no single existing protocol, particularly for message authentication, simultaneously provides both adequate security and adequate latency to meet the needs of all potential applications for the GridStat system: it is necessary to make tradeoffs between the level of security provided and the performance, and these tradeoffs will be different for different applications and data streams [69]. The security module architecture addresses both needs. A number of different encryption, authentication, and obfuscation modules have been built for GridStat. It is relatively easy to build modules to explore new cryptographic protocols having different performance tradeoffs. For example, recent work on protocols based on one-time public-key signature protocols has been incorporated as a new module choice offering a favorable latency/security tradeoff for some situations [70, 71]. The module replacement mechanism is designed to allow the modules in use for a particular stream or node authentication purpose to be securely replaced during operation, even if the algorithm used by the replaced module has been compromised. The scheme for this relies on sharing a collection of symmetric key material between a security manager node and the nodes that it manages at the time that the nodes are commissioned. This pre-shared key material is slowly consumed over the life of the system when module replacements occur.

Another aspect concerns securing the nodes of the security management system themselves. Compromising the security management node associated with a particular QB potentially compromises communications of the management subtree that the QB anchors. To address this, a distributed security service has been designed using threshold cryptographic techniques to ensure that compromising even several nodes does not compromise operation of the system [72].

At present, the GridStat framework does not contain any specific support for trust decision making, though the work in [57] designed to support GridStat or any other critical infrastructure publish-subscribe system. A utility operator's decisions rely on data received and inferred. In an adversarial world, it must be possible for operators to assess how likely the data that they are provided truly reflect the state of the system or whether they have been manipulated or corrupted. Security mechanisms like those described above are extremely important for reducing the likelihood of tampering, but decisions should not be taken with a blind assumption that the security mechanisms are completely effective in all cases (for example, data could be manipulated by tampering with sensors, ahead of the point where a digital signature is applied).

Research undertaken so far addresses what such mechanisms should look like [73] and how they can be used to support utilities' operational decision making [57], [74].

8. A NEW WORLD FOR POWER APPLICATION PROGRAMMERS AND RESEARCHERS

The capabilities provided by GridStat, or any WAMS-DD that meets the DRs and IGs and implements security appropriate for WAMs-DD as outlined above, are a quantum leap in data delivery for electric power grids and other critical infrastructures. However, they also open up new ways of thinking for power researchers and engineers in terms of their power applications. We summarize this in this section.

As a baseline, power applications can be written assuming that the WAMS-DD will be

- **Extremely fast**: less than 1 msec or so added over the speed of light in the underlying network layers
- **Extremely available**: and providing the performance guarantees even in the face of failures
- **Appropriately cyber-secure**: security mechanisms appropriate for long-lived embedded devices and are as lightweight as possible so as not to preclude closed-loop applications
- **Tightly managed**: on a per-subscriber-per-publication basis to provide the above strong guarantees
- **Adaptive**: can change its subscription base very fast.

In order to exploit this much improved baseline, power researchers and engineers need to ask themselves a number of questions, including:

- What Qos+ — {rate,latency,redundancy} — does my application *really* need for its remote data?
- Why fundamentally does it need this level of QoS+?
- How sensitive is my application to occasional longer delays, period drops (perhaps a few in a row), or data blackouts for longer periods of time?
- How can I formulate and test hypotheses for the above?

Further, note that the above is only for the steady state. However, the questions should also be asked for different power contingencies. For example:

- How important is my application in a given contingency (e.g., simply assign a number 0-10)?
- How much lower {rate,latency,redundancy} can I live with in a contingency (here, even at a lower rate, note that these are still strong guarantees)?
- Can I program my application to run at different rates (assuming, if necessary, that anti-aliasing is done appropriate for that rate), or is there is a fundamental reason that my application has to run at a given rate?
- What extra data feeds (or existing ones at higher rates) could I use in a given contingency, assuming that they could be available within a second of the contingency being identified? How can I best use these for either operator situational awareness or closed-loop responses?

From the above, it may not be clear that power researchers need to work closely with computer scientists in order to help the power grid be more resilient. The status quo today is that power researchers tend to ignore communications issues and computer scientists do not have nearly enough useful information about the power grid's applications as well as its pragmatic IT constraints in order to offer significant help. However, if they work together, much can be achieved. For example, power researchers can assume much "better" communications (strong guarantees, systematic adaptation, etc) while computer scientists can come up with even better WAMS-DD mechanisms if they know not only emerging "killer apps" for power but also the tradeoffs that power applications can live with.

Finally, note that the above only involved communications. The computational resources available to make power grids more stable should not be assumed to be static, in particular thanks to cloud computing [75]. Indeed, the GridCloud project is combining GridStat researchers with Cornell researchers who develop advanced cloud

computing mechanisms to provide a comprehensive, end-to-end solution for mission critical cloud computing. Such a system will keep the same high throughput as generic, commercial cloud computing mechanisms improve consistency and predictability. The consistency is improved by removing the deliberate (and, as it turns out, unnecessary) tradeoff to allow occasional inconsistencies in cloud data to preserve high throughput. In terms of performance, GridCloud is also implementing ways to make the aggregate performance and ramp-up time of cloud computations much better.

Given such an improved computational infrastructure, then, questions for power researchers include how they could use:

- Hundreds of processors in steady state
- Thousands or tens of thousands of processors when the grid is approaching a blackout
- Data from all participants in a grid enabled quickly when a blackout is being approached

Such improved computational support will likely have a huge impact on power grid operations.

The questions in this section represent a completely new way of thinking for power researchers. However, it has the potential to make the power grid much more resilient than simply using today's applications (or ones planned for tomorrow) than simply assuming the communications and computations are limited and non-adaptive.

9. CONCLUSIONS

In this chapter described requirements for building WAMS-DD, and implementation guidelines the authors believe are necessary for a proper WAMS-DD or are at least highly advisable. It then analyzed how completely these requirements and guidelines are met by network-level technologies, middleware-level technologies, and technologies developed in the electricity sector. Following this the chapter overviewed the NASPInet initiative, and explained how NASPInet and other WAMS-DDs must employ middleware (as well as, of course, network-level mechanism that the middleware layers on top of). Finally, we described GridStat in detail, and overviewed the completely new way of thinking of new power application programs that it enables.

ACKNOWLEDGEMENTS

The GridStat project has been going on since 1999 so there are many people to acknowledge for their support and guidance besides the authors, and we apologize in advanced for any omissions. GridStat co-PI Prof. Carl Hauser has been an invaluable partner in the development of GridStat. Prof. Anjan Bose has been an invaluable collaborator from the power grid side. Dr. Thoshitha Gamage has contributed heavily to GridStat in the last few years, especially in cyber-physical systems and cyber-security.. Graduate students Erlend Viddal, Stian Abelsen, Venkata Irava, Erik Solum, Ryan Johnston, Joel Helkey, Kim Swenson, Sunil Muthuswamy, Supreeth Sheshadri, Ping Jiang, Rasika Chakravarthy, Michael Carter, and Punit Agrawal have all contributed heavily, in addition to the second and third authors during their doctoral studies. Power researchers who have helped influence GridStat's applications and requirements include graduate student Tao Yang and power researchers Dr. Chuanlin Zhao and Prof. Mani Venkatasubramanian.

Lead GridStat staff programmer David Anderson has contributed immensely to GridStat's implementation as a full-time programmer for the last 4 years as and undergraduate programmer before that. Undergraduate staff programmers include Andrew Lytle, Raeann Marks, Loren Hoffman, Nathan Schubkegel, Alex Schuldberg, Ryan Hoffman, Kyle Wilcox, Travis Gomez, Chad Selph, Constance Bell, Simon Ngyuen, Nick Newsome, and DeAndre Blackwell. Suggestions, questions, and advice from a large number of people have helped influence GridStat. These are too numerous to list, but include Jeff Dagle, Greg Zweigle, Phil Overholt, Patricia Hoffman, Dave Bertagnolli, Rich Stephenson, Jeff Gooding, Mahendra Patel, Ritchie Carroll, Ken Martin, Floyd Galvan, Lisa Beard, Neeraj Suri, Deb Frincke, Jim McNierney, Dan Brancaccio, Dick Wilson, Dan Lutter, and Don Kopczynski.

Many funding agencies have provided financial support for GridStat. These include US Department of Energy grant DE-OE0000097 (TCIPG), US Department of Energy grant DE-OE0000032 (GridSim), ARPA-E grant DE-AR0000230 (GridCloud), grants CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)) and CCR-0326006 from the US National Science Foundation; and by grant #60NANB1D0016 (Critical Infrastructure Protection Program) from the National Institute of Standards and Technology, in a subcontract to Schweitzer Engineering Labs Inc. We thank the US Dept. of Energy and Dept. of Homeland Security

for their part in funding the NSF TCIP center. We thank Schweitzer Engineering Labs for their donation of equipment and PNNL for their loan of equipment.

REFERENCES

[1] D. E. Bakken, R. E. Schantz, and R. D. Tucker, "Smart Grid Communications: QoS Stovepipes or QoS Interoperability," in *Proc. 2009 Grid-Interop*, GridWise Architecture Council, Denver, Colorado. [Online]. Available: http://gridstat.net/publications/TR-GS-013.pdf.

[2] K. Fodero, C. Huntley, D. E. Whitehead, and B. Kasztenny, "A Novel Scheme for Wide-Area Time Synchronization," in *Proc. 2010 Developments in Power System Protection, Managing the Change, 10th IET International Conference*, pp. 1 – 5, April 2010.

[3] 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," Washington State University, Pullman, WA, Tech. Rep. EECS-GS-009, May 2007.

[4] North American Synchrophasor Initiative, "Quanta Statement of Work.", May 2008. [Online] http://www.naspi.org/resources/dnmtt/quanta sow.pdf

[5] Electric Power Research Institute (EPRI), "The Integrated Energy and Communication Systems Architecture," vol. IV: Technical Analysis, 2004.

[6] S. Horowitz. D. Novosel, V. Madani, M. Adamiak, "System-Wide Protection," *IEEE Power and Energy Mag.*, vol. 6 no. 5, pp. 34–42, Sept. 2008

[7] E. Rosen, A. Vishanathan, and R. Callon, "RFC-3031: Multiprotocol Label Switching Architecture," *The Internet Society*, 2001. [Online]. Available: http://datatracker.ietf.org/doc/rfc3031.

[8] Y. Krishnamurthy, V. Kachroo, D. A. Karr, C. Rodrigues, J. P. Loyall, R. E. Schantz, and D. C. Schmidt, "Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications," in *Proc. 2001 ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems*, vol. 36, no. 8, pp. 230-237, 2001.

[9] R. E. Schantz and D. C. Schmidt, "Research Advances in Middleware for Distributed Systems: State of the Art," in *Proc. 2002 IFIP World Computer Congress*.

[10] J. Saltzer, D. Reed, and D. Clark, "End-to-End Arguments in System Design," *Trans. on Computer Systems, Association of Computing Machinery*, vol. 2, no. 4, pp. 277–288, Nov. 1984.

[11] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group Communication Specifications: A Comprehensive Study," *ACM Computing Surveys*, 33(4), pp. 1–43, Dec. 2001.

[12] X. Défago, A. Schiper, and P. Urbán, "Totally Ordered Broadcast and Multicast Algorithms: Taxonomy and Survey," 36(4), pp. 372–421, Dec. 2004.

[13] Y. Hu et al. *Data Bus Technical Specifications for North American Synchro-Phasor Initiative Network (NASPInet)*, May 2009. https://www.naspi.org/File.aspx?fileID=587.

[14] R. E. Schantz, J. P. Loyall, M. Atighetch, and P. P. Pal, "Packaging Quality of Service Control Behaviors for Reuse," in *Proc. 2002 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*.

[15] P. P. Pal, J. P. Loyall, R. E. Schantz, J. A. Zinky, R. Shapiro, and J. Megquier, "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration," in *Proc. 2000 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*.

[16] D. E. Bakken, "Quality of Service Design Considerations for NASPInet," *presentation to the North American Synchrophasor Initiative (NASPI) Work Group Meeting*, Feb. 4, 2009. Available via www.naspi.org.

[17] K. H. Gjermundrød, I. Dionysiou, C. H. Hauser, D. E. Bakken, and A. Bose, "Flexible and Robust Status Dissemination Middleware for the Electronic Power Grid," School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, Tech. Rep. EECS-GS-003, Sept. 2003.

[18] R. Bobba, E. Heine, H. Khurana, and T. Yardley, "Exploring a Tiered Architecture for NASPInet," in *Proc. 2010 IEEE Conference on Innovative Smart Grid Technologies*.

[19] K. Tomsovic, D. E. Bakken, M. Venkatasubramanian, and A. Bose, "Designing the Next Generation of Real-Time Control, Communication and Computations for Large Power Systems," in *Proc. IEEE Special Issue on Energy Infrastructure Systems*, 93(5), May 2005.

[20] K. H. Gjermundrød, D. E. Bakken, C. H. Hauser, and A. Bose, "GridStat: A Flexible QoS-Managed Data Dissemination Framework for the Power Grid," *IEEE Trans. Power Delivery*, 4(1), pp. 136–143, 2009.

[21] V. S. Irava and C. H. Hauser, "Survivable Low-Cost Low-Delay Multicast Trees," in *Proc. 2005 IEEE Global Telecommunications Conference*.

[22] V. S. Irava, "Low-Cost Delay-Constrained Multicast Routing Heuristics and Their Evaluation," Ph.D. dissertation, Washington State University, Pullman, WA, Aug. 2006.

[23] GridWise Architecture Council, *Interoperability Constitution Whitepaper (v1.1)*, Dec. 2006.

[24] U.S.-Canada Power System Outage Task Force, "Final Report on the August 14th, 2003 Blackout in the United States and Canada," 2004. [Online]. Available: https://reports.energy.gov/

[25] L. Roberts, "A Radical New Router," in *Proc. 2009 IEEE Spectrum*, pp. 35–39.

[26] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, "The Many Faces of Publish-Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, June 2003.

[27] J. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, April 1997.

[28] K. P. Birman, J. Chen, K. M. Hopkinson, R. J. Thomas, J. S. Thorp, R. van Renesse, and W. Vogels, "Overcoming Communications Challenges in Software for Monitoring and Controlling Power Systems," in *Proc. IEEE*, 9(5), May 2005.

[29] K. Hopkinson, G. Roberts, X. Wang, and J. Thorp, "Quality of Service Considerations in Utility Communication Networks," *IEEE Trans. Power Delivery*, 24(3), July 2009.

[30] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "RFC3697: IPv6 Flow Label Specification," *The Internet Society*, 2004. [Online]. Available: http://www.faqs.org/rfcs/rfc3697.html.

[31] Ymir Vigfusson, Hussam Abu-Libdeh, Mahesh Balakrishnan, Ken Birman, Robert Burgess, Haoyuan Li, Gregory Chockler, Yoav Tock, "Dr. Multicast: Rx for Data Center Communication Scalability", Eurosys, April 2010 (Paris, France). ACM SIGOPS 2010, pp. 349-362.

[32] Dmitry Basin, Ken Birman, Idit Keidar, Ymir Vigfusson, "Sources of Instability in Data Center Multicast", ACM Principles of Distributed Computing (PODC), Zurich, Aug. 2010. LADIS 2010 Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware.

[33] K. Birman. *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*. Springer, 2012.

[34] A. Farrel (ed), "Network Quality of Service Know It All," *Elsevier*, 2009.

[35] A. Sydney, J. Nutaro, C. Scoglio, D. Gruenbacher, and N. Schulz, "Simulative Comparison of Multiprotocol Label Switching and openFlow Network Technologies for Transmission Operations", *IEEE Transactions on Smart Grid*, 4(2), June 2013, 763–770.

[35b] Robert Wojcik and AndreJ Jajszczyk, "Flow Oriented Approaches to QoS Assurance", *ACM Computing Surveys*, 44(1), January 2012.

[36] PGM Reliable Transport Protocol Specification, RFC 3208, IETF, 2001.

[37] Y. Amir, C. Danilov, and J. Stanton, "A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area Group Communication," in *Proc. 2000 IEEE First International Conference on Dependable Systems and Networking*.

[38] [Online]. Available: www.spreadconcepts.com

[39] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, 38(2), April 2007, 69–74.

[40] Open Network Foundation, *OpenFlow Switch Specification 1.3.2*, April 25, 2013.

[41] S. McGillicuddy, "Not all OpenFlow Hardware is Created Equal: Understanding the Options", Open Network Foundation, 25 September 2013, available via www.opennetworking.org.

[42] Net Insight, "The Nimbra Platform", http://www.netinsight.net/Global/Documents/Products/Brochures/Nimbra_Brochure.pdf.

[43] K. Geihs, "Middleware Challenges Ahead", *IEEE Computer*, June 2001, p. 24–31.

[44] D. Bakken, "Middleware", unpublished article, 2001, http://eecs.wsu.edu/~bakken/middleware.pdf.

[45] Young-Jim Kim, Jaehwan Lee, Gary Atkinson, Hongseok Kim, and Marina Thottan. "SeDAX: A Scalable, Resilient, and Secure Platform for Smart Grid Communications". IEEE Journal on Selected Areas in Communications, 30(6), July 2012, 1119–1136.

[46] K. Birman, "Like It or Not, Web Services Are Distributed Objects!" *Communications of the ACM*, 47:12, pp. 60–62.

[47] K. Birman, "The Untrustworthy Web Services Revolution," *IEEE Computer*, 39:2, pp. 98–100, Feb. 2006.

[48] PJM, *PJM 2007 Strategic Report*, April 2, 2007, http://www2.pjm.com/documents/downloads/strategic-responses/report/20070402-pjm-strategic-report.pdf.

[49] Open Secure Energy Control Systems (OSECS), www.osecs.com.

[50] J. Zhang and C.A. Gunter, "Application-Aware Secure Multicast for Power Grid Communications", in IEEE International Conference on Smart Grid Communications (SmartGridComm), Gaithersburg, MD.

[51] C. Hauser, T. Manivannan, D. Bakken. "Evaluating Multicast Message Authentication Protocols for Use in Wide Area Power Grid Data Delivery Services", in *Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS)*, IEEE, Maui, Hawaii, January 4-7, 2012, 2151-2158.

[52] D. Clark, "A Cloudy Crystal Ball – Visions of the Future". In Proceedings of the Twenty-fourth Meeting of the Internet Engineering Task Force (IEF), July, 1992, p. 539-543.

[53] W. Mahnke, S. Leitner, and M. Damm. *OPC Unified Architecture*, Springer, May 2009.

[54] Z. Xie, M. Govindarasu, V. Vittal, A. Phadke, and V. Centeno, "An Information Architecture for Future Power Systems and Its Reliability Analysis," *IEEE Trans. Power Systems*, 17:3, pp. 857–863, Aug. 2002

[55] Hauser, Carl H., Bakken, David E., Dionysiou, Ioanna, Gjermundrød, K. Harald, Irava, Venkata, Helkey, Joel and Bose, Anjan. "Security, Trust and QoS in next-generation control and communication for large power systems*," International Journal of Critical Infrastructures*, Inderscience, 4(1-2), 2008, 3–16.

[56] Ioanna Dionysiou, Deborah Frincke, Carl Hauser, and Dave Bakken, "An Approach to Trust Management Challenges for Critical Infrastructures*", Proceedings of the 2nd International Workshop on Critical Information Infrastructures Security (CRITIS07)*, Lecture Notes in Computer Science Series volume 5141, pp. 173–184, Malaga, Spain, October 2–5, 2007, Springer Berlin.

[57] Yujue Wang, Carl Hauser: An Evidence-Based Trust Assessment Framework for Critical Infrastructure Decision Making. IFIP Working Group 11.10 Conference on Critical Infrastructure Protection 2011: 125-135.

[58] M.D. Hadley, J.B. McBride, T.W. Edgar, L.R. O'Neil, and J.D. Johnson, "Securing Wide Area Measurement Systems ", Technical Report PNNL-17116, Pacific Northwest National Laboratory, 2007

[59] Bakken, D.E.; Bose, A.; Hauser, C.H.; Whitehead, D.E.; Zweigle, G.C., "Smart Generation and Transmission With Coherent, Real-Time Data," Proceedings of the IEEE , vol.99, no.6, pp.928,951, June 2011

[60] K. Swenson, "Exploiting Network Processors for Low Latency, High Throughput, Rate-Based Sensor Updated Delivery," M.S. Thesis, Washington State University, Pullman, WA, 2009.

[61] Gjermundrod, H.; Hauser, C.; Bakken, D., "Scalable Wide-Area Multicast with Temporal Rate Filtering Distribution Framework," Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on , vol., no., pp.1,8, Aug. 31 2011-Sept. 2 2011

[62] Harald Gjermundrød and David E. Bakken and Carl H. Hauser, Integrating an Event Pattern Mechanism in a Status Dissemination Middleware, in *Proceedings of 1st International Conferences on Pervasive Patterns and Applications (PATTERN'09),* IEEE, Athens, Greece, November 2009, 259–264.

[63] Stian F. Abelsen and Harald Gjermundrød and David E. Bakken and Carl H. Hauser, "Adaptive Data Stream Mechanism for Control and Monitoring Applications", in *Proceedings of 1st International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE'09),* IEEE, Athens, Greece, November 2009, 86–91.

[64] K. H. Gjermundrød, "Flexible QoS-Managed Status Dissemination Middleware Framework for the Electric Power Grid," Ph.D. dissertation, Washington State University, Pullman, WA, Aug. 2006.

[65] Erlend S. Viddal and David E. Bakken and Harald Gjermundrød and Carl H. Hauser, "Wide-Area Actuator RPC over GridStat with Timeliness, Redundancy, and Safety". In *Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'2010),* Krakow, Poland, Feb 15-18, 2010, 17–24.

[66] Michael J. Fischer, Nancy A. Lynch, and Michael S. Patterson, "Impossibility of Distributed Concensus with One Faulty Process", *Journal of the ACM,* 32, April 1985, 374–382.

[67] E. Solum, C. H. Hauser, R. Chakravarthy, "Modular Over-the-Wire Configurable Security for Long-Lived Critical Infrastructure Monitoring Systems, in *Proc. 2009 3rd ACM Int'l Conf. on Distributed Event-Based Systems.*

[68] R. Chakravarthy, C. H. Hauser, and D. E. Bakken, "Long-Lived Authentication Protocols for Critical Infrastructure Process Control Systems," *Fourth IFIP WG 11.10 Int'l Conf. on Critical Infrastructure Protection*," Mar. 2010.

[69] Carl H. Hauser, Thanigainathan Manivannan, David E. Bakken: Evaluating Multicast Message Authentication Protocols for Use in Wide Area Power Grid Data Delivery Services. HICSS 2012: 2151-2158.

[70] Kelsey Cairns, Carl Hauser, and Thoshitha Gamage. "Flexible Data Authentication Evaluated for the Smart Grid". In IEEE Smart Grid Communications Conference, Oct. 21-24, 2013, Vancouver, Canada. To appear.

[71] Kelsey Cairns, Thoshitha Gamage, and Carl Hauser. "Efficient Targeted Key Subset Retrieval in Fractal Hash Sequences." In Proceedings of the 20th ACM Conference on Computer and Communications Security, Nov. 4-8, 2013, Berlin, Germany. To appear.

[72] Punit Agrawal. A fault-tolerant key storage service with proactive recovery. Master's Thesis, School of Electrical Engineering and Computer Science, Washington State University, August 2012. Available: http://www.dissertations.wsu.edu/Thesis/Summer2012/p_agrawal_091412.pdf

[73] I. Dionysiou, "Dynamic and Composable Trust for Indirect Interactions," Ph.D. dissertation, Washington State University, Pullman, WA, Aug. 2006.

[74] Carl H. Hauser: Trust research to address uncertainty in security for the smart grid. In *Proceedings of the IEEE Innovative Smart Grid Technologies (ISGT)* 2012: 1-2.

[75] Kenneth P. Birman, Lakshmi Ganesh, and Robbert van Renesse, "Running Smart Grid Control Software on Cloud Computing Architectures", Workshop on Computational Needs for the Next Generation Electric Grid, Cornell University, April 19-20, 2011. Ithaca, NY.