GridStat: A Flexible QoS-Managed Data Dissemination Framework for the Power Grid

Harald Gjermundrød, *Member, IEEE*, David E. Bakken, Carl H. Hauser, and Anjan Bose

***Abstract*—With the increase in the monitoring of operational data at very high rates in high voltage substations and the ability to time synchronize these data with global positioning systems, there is a growing need for transmitting this data for monitoring, operation, protection, and control needs. The sets of data that need to be transferred and the speed at which they need to be transferred depend on the application—for example, slow for postevent analysis, near real time for monitoring and as close to real time as possible for control or protection. In this paper, we describe GridStat, a novel middleware framework we have developed to provide flexible, robust, and secure data communications for the power grid's operations. Test results demonstrate that such a flexible framework can also guarantee latency that is suitable for fast wide-area protection and control.**

***Index Terms*—Computer control of power systems, distributed control, energy-management systems, power system measurements, power system monitoring, power system protection, quality of service (QoS), supervisory control and data-access (SCADA) systems, wide-area control.**

I. Introduction

THE communication systems for the electric power grids in North America were developed in response to the 1965 Northeastern U.S. blackout. Today's supervisory control and data access (SCADA) systems, which form the core of the communication system for monitoring and controlling the wide-area power grid, are based on the requirements and technology of that time period. In the years since then, the data collection capabilities of devices in substations, such as IEDs, and energy management systems' computing capacity in control centers (EMSs) have grown enormously.

The communications system is known to greatly limit the kinds of control and protection that can be performed in today's electric power grid [1], [2]. It is, however, relied on heavily by a wide range of power application programs spread throughout utilities. Programming distributed systems, where applications and services are spread across a computer network, is known to be very difficult, even for computer science experts in the field. In the last few decades, a new class of software, *middleware*, has arisen to help simplify the task of programming a distributed system [3]. Middleware is a layer of software above the operating system that provides higher-level building blocks for programmers to use. In doing so, it helps shield programmers from having to deal with diversity of CPU architecture, programming language, operating system, and network technology. It also allows application software to be ported to new environments much easier than if they were written without middleware [i.e., directly to a network application programming interface (API)], especially one that is specific to a particular operating system.

For these reasons, middleware frameworks such as CORBA have been widely used in such industries as commercial avionics, telecommunications, transportation since the 1990s and has been used by the U.S. military since the 1980s. Middleware also can be used to provide quality of service (QoS) such as delay guarantees and availibility (via redundancy management) while allowing the application program to not get locked into a particular QoS mechanism [4], [5].

We have designed a middleware framework called GridStat to support the emerging needs of the electric power system for a flexible communication system. It manages network resources to provide low-latency, reliable delivery of information produced anywhere on the network and sent to multiple other points (i.e., GridStat provides QoS-managed multicast). Using the middleware approach allows data providers and application developers to avoid the thornier issues involved in wide-area communication and to concentrate on application needs. GridStat was designed to meet the flexibility and QoS requirements outlined above, and ongoing work is addressing other miscellaneous (security and trust) requirements. More about the flexibility and QoS requirements for the power grid, with detailed citations of industry sources and standards as well as power researchers, can be found in [6].

In this paper, we describe the design, implementation, and performance of GridStat. The remainder of this paper is organized as follows. Section II describes the overall architecture of GridStat. The design of its major components is given in Section III. Section IV provides experimental results for GridStat, mainly delivery latency and throughput. described in the preceeding sections. Section V compares GridStat with related research and products in both computer science and the power grid. Finally, Section VI concludes.

II. GridStat Architecture

In the GridStat middleware architecture, network management and data delivery are handled by separate but interacting

Manuscript received September 27, 2007. First published June 03, 2008; current version published December 24, 2008. This work was supported in part by Grants CNS 05-24695 [CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)], in part by CCR-0326006 from the U.S. National Science Foundation, and in part by Grant 60NANB1D0016 (Critical Infrastructure Protection Program) from the National Institute of Standards and Technology, in a subcontract to Schweitzer Engineering Labs, Inc. The U.S. Department of Energy and Department of Homeland Security supported the National Science Foundation TCIP center. Paper no. TPWRD-00582-2007.

H. Gjermundrød is with the Department of Computer Science, University of Cyprus, Nicosia, Cyprus. He is also with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752 USA (e-mail: harald@cs.ucy.ac.cy).

D. Bakken, C. Hauser, and A. Bose are with the School of Electrical Engineering and Computer Science at Washington State University, Pullman, WA 99164-2752 USA (e-mail: bakken@eecs.wsu.edu; hauser@eecs.wsu.edu; bose@eecs.wsu.edu).

Digital Object Identifier 10.1109/TPWRD.2008.917693

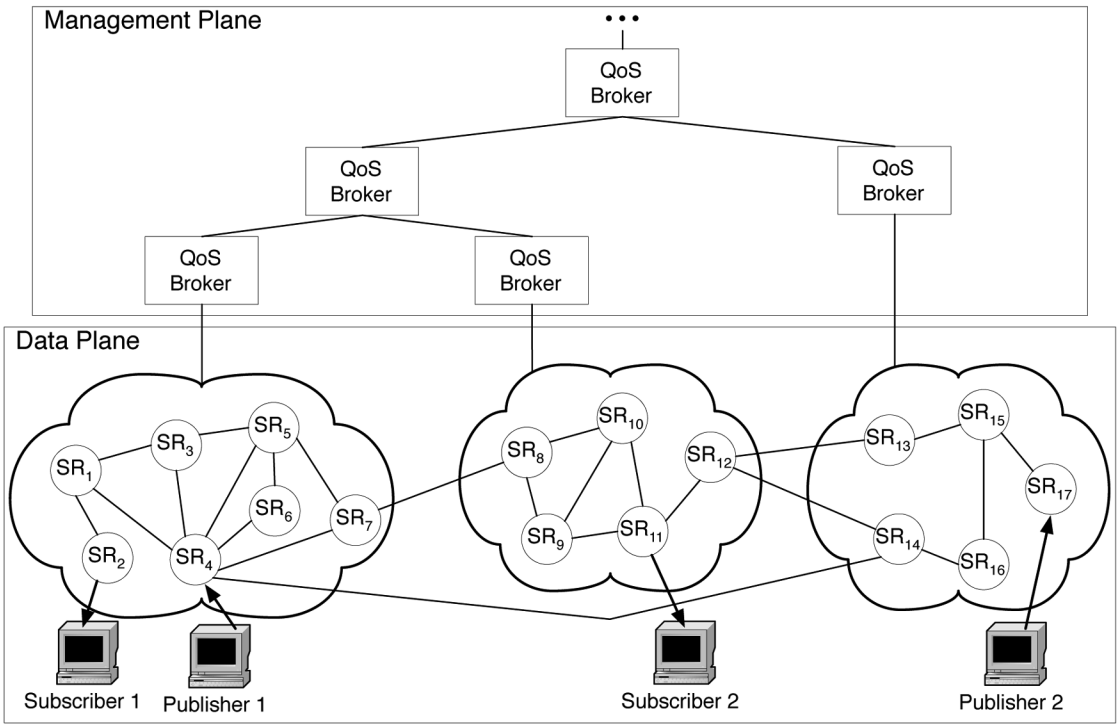


Fig. 1. GridStat architecture.

subsystems called *planes*. The *management plane* allocates resources and adapts the network in reaction to changing power system configurations or communication network failures. *Data plane* components are responsible for forwarding data from each source to potentially many destinations as directed by management-plane components.

GridStat's data plane supports a *publish-subscribe* communication model. Thus, items in each periodic data stream produced at a source (publisher) are distributed to destinations (subscribers) without the publisher having to explicitly track all the subscribing entities. This simplifies application programs and gives the system the flexibility to add subscribers and even change the characteristics of existing subscriptions at runtime. A publisher simply announces the availability of its data stream(s) to the management plane. Subscribers then request that the middleware set up delivery paths (with QoS, including rate, latency, and redundancy) to accomplish the delivery. GridStat middleware components at the publishers and subscribers provide programming interfaces (APIs) that applications use to make the announcements and requests. Similarly, the publisher and subscriber middleware APIs allow applications to send and receive data items.

The basic architecture of GridStat is illustrated in Fig. 1. The active components of the management plane are called GridStat *QoS brokers* and those of the data plane are called *status routers*. The QoS brokers are hierarchically arranged with policies set at higher levels in the hierarchy controlling more global aspects and allowing local concerns to be implemented using policies at lower levels. The hierarchical organization was designed to support decomposition of communication management and cyber-security policies along organizational and geographic boundaries. For example, the hierarchy enables utilities to have policies about bandwidth allocation for different applications within the company and to limit internal and external subscriptions to status variables. These policies can also be set at levels below the utility (regions of a utility) and above it (e.g., bandwidth allocated to different entities' subscriptions on an RTO's exchange network).

In contrast to the hierarchy of the management plane, the status routers have a flat organization. A collection of status routers in the same administrative domain and having the same resource management and cyber-security policies is called a *cloud*. A cloud of status routers is directly managed by a QoS broker that is at the bottom of the hierarchy, which we call a *leaf QoS broker*. The status routers forward each incoming data item on outgoing communication links that have a downstream subscriber for that item. Communication links between clouds are managed by the least-common-ancestor QoS broker of the brokers that manage the individual clouds, based on the policies in place at that QoS broker.

In announcing publications and requesting subscriptions, publishers and subscribers interact with a QoS broker using the middleware APIs. For a publication, the QoS broker simply notes that a publication is available and its publication rate. Subscriptions are more complicated: after verifying that a subscription request can be satisfied within the available resources, the QoS brokers communicate with the management interfaces of the status routers along the data delivery path to set up forwarding rules. Since subscribers can request *lower* update rates than publishers publish, and different subscribers of a given status variable can request delivery at different rates, some items may not need to be forwarded. Therefore, status routers implement *rate filtering* as part of their forwarding mechanisms, which means that different subscribers to the

same status item can get it delivered at a different rate. For example, a monitoring application in the same substation may need the data many times a second, in order to perform some protective action, while remote applications that are loosely tracking the status item (for example, to ensure that its producer is functioning) may need it only once per second or even less frequently. Conversely, since a subscription may require use of redundant paths, even a single subscription may cause an item to be forwarded on multiple links. Thus, multicast and heterogeneous delivery rates are efficiently supported: each update message only traverses any network link at most once. Utilizing this spatial redundancy, specifically where all redundant links are used actively (instead of as backups) provides essentially zero fail-over time when a link fails, compared with best practices failover time in a LAN "10 to 50 ms or greater" as reported in [7].

Note that in an actual wide-area deployment of GridStat, the status routers and network links would be engineered such that different kinds of communication media were used, with very different properties (error rate, bandwidth, latency, cyber security vulnerabilities, susceptibility to jamming, etc.). Further, they could be obtained from different providers: backbone Internet providers, cable modem ISPs, cellular phone providers (for some low-bandwidth, high importance traffic), broadband over power lines, etc. Thus, updates from a single publisher could be made to traverse different network technologies and different providers (and possibly different operating systems). This diversity has the potential to greatly enhance the reliability of data delivery, in part because the different paths from publisher to subscriber can potentially be much less likely to suffer from common mode failures, for example due to a single vulnerability in a single instance of technology (such as WiFi or Linux).

This section has overviewed the baseline capabilities of the GridStat framework focusing on the timely and robust delivery of updates. References [6] and [8] contain a description of more advanced features that GridStat supports.

III. Design of the Principal GridStat Data Plane Entities

As previously mentioned, the principal GridStat entities are *publishers, subscribers, status routers*, and *QoS brokers*. QoS brokers are the main component of the management plane and do not participate in data forwarding at all. Publishers, subscribers, and status routers are all data plane components. Understanding the performance, security and reliability of the GridStat framework requires an understanding of each of these entities and the interactions between them.

There are two types of interactions: *command interactions* and *forwarding interactions*. Data plane entities support both types of interaction and management plane entities support only command interactions.

Command interactions are implemented by *command modules* using CORBA client-server technology. Java RMI and Microsoft's .NET would be appropriate alternatives to CORBA—the important thing is to leverage a distributed system technology base to achieve interoperability between implementations using different languages and running on different operating systems.

Forwarding interactions are conducted in GridStat using an abstraction called an *event channel*, depicted by the lines connected to the status routers in Fig. 1. Each GridStat event channel represents a point-to-point communication capability between the entities that it connects. The event channel abstraction provides a great deal of flexibility concerning the underlying network that carries data between two entities: it could be an actual point-to-point link, or an IP network, or an ATM network, or a SONET ring, for example. GridStat was designed to be implemented on top of all of these kinds of networks. This is very important, because practical wide-area deployment GridStat would be likely to involve data paths crossing multiple network technologies and involving multiple utilities. Further, in practice it will be necessary to provide QoS guarantees such as latency that span these different network technologies.

The GridStat event channel is a building block for QoS-aware point-to-multipoint (multicast) communication which, in GridStat, has a distributed implementation provided collectively by the status routers and management plane. Of course, providing QoS in GridStat involves receiving certain QoS guarantees from the underlying network. For example, the public Internet currently does not support QoS guarantees yet a private internet, using the same protocols but carrying only GridStat traffic, could provide adequate QoS guarantees.

The data plane is structured as follows: each publisher is connected by an event channel to a status router. The event channel carries status items for all the publications made by the publisher. Similarly, a subscriber is connected to a status router by an event channel that carries status items for all the subscriber's subscriptions. Between the publisher and subscriber there is at least one delivery path through status routers. Each status router has potentially many incoming and many outgoing event channels that connect it to publishers, subscribers, and other status routers. A status router's job is to forward each incoming item on the outgoing event channels where it is needed. The forwarding rules are determined by the management plane when subscriptions are added, and then inserted in the status router using a command interaction.

As in any communication network, the delivery latency of a packet across a link is made up of several components.

Transmission time T=(L/b), the length of the packet (L) divided by the bandwidth of the link (b).

Propagation delay P=(D/c), the length of the link (D) divided by the speed of light in the particular communication medium (c). These are fixed by the packet length, network technology and distance between nodes.

Processing time C.

Queuing delay Q; both C and Q are determined by the performance of the routing hardware and software and the offered load. These are fixed by the packet length, network technology and distance between nodes.

Thus, per-link latency is T+P+C+Q. In GridStat, it is the end-to-end latency that is of interest so these quantities must be summed over all the status routers and links that a packet traverses.

The GridStat prototype implementation provides a baseline for evaluating the forwarding performance of status routers, primarily C and Q in the latency expression. C is directly under the control of the status router implementor—it depends on the speed of the routing hardware and the complexity of tasks that are performed on each incoming packet as well as the quality of the implementation of those tasks. The queuing delay incurred by a particular packet is also potentially under the control of the routing software but the guarantees that are possible are ultimately affected by the total offered load for each link. In order to limit the effect of queuing delays to those that are necessary, status routers implement rate filtering as follows: incoming packets that are not needed to meet the rate requirements of any downstream subscriber are dropped. There is a tradeoff involved in this, however: performing the computation to implement this feature increases C for packets that are forwarded so the relative contributions to latency of additional network traffic and additional computation must be evaluated.

A. Design of the Publisher and the Subscriber

The GridStat publisher and subscriber are implemented in software shared libraries that can be linked with applications. The libraries implement the communication steps that are needed to establish and maintain a connection with a status router, including reconnecting when communication is lost. Library implementations are available in Java and C# (the favored language of Microsoft's .NET middleware framework). They have been designed so that publisher and subscriber applications can exist on a wide range of devices, ranging from embedded devices of limited computational power to high-powered computers running modeling and simulation applications running in control centers. It is important to note here that "subscriber" and "publisher" are merely roles that an application program or even hardware device may play and that any application or device may play either or both roles with respect to many publications. For an application, publishing is simply a matter of asserting the intent to publish, stating the intended publishing rate, then periodically calling the Publish method.

Subscribing involves identifying a particular status variable along with the desired update rate, latency, and level of redundancy. When it receives this request the status router forwards it to a QoS broker which arranges, using management plane facilities, the needed forwarding rules in the network of status routers. Neither the application, the Publisher, nor the Subscriber need be aware of the communication network topology (the number and location of status routers and the communication links between them). Routing is handled entirely by the management plane.

When the setup is complete the application can receive the most recently received status item for the subscription using either (or both) of two different interfaces. The *cache API* lets the application program treat the subscribed status variable as if it were a local variable. This frees the application from concerning itself with directly handling variable updates. The *callback API* lets the subscriber register a callback so that it is notified of updates as they arrive. This is useful, for example, for integration into a local database. Additionally, the application may request notification of QoS violations—for example, if an expected update is late or does not arrive.

B. Design of the Status Router

The purpose of the collection of status routers is to provide a message bus between publishers and subscribers that is specialized for forwarding streams of periodically-generated update messages. Compared with an IP router, a status router is specialized to provide multicast, rate filtering, and subscription modes (defined below) while supporting a range of latency, rate, and redundancy requirements, even for the same data item. These specializations are all based on the observation that resource allocation and routing decisions for a large status network are made infrequently, on the basis of subscriptions, rather than packet-by-packet as in an IP router.

Subscription modes are a feature of GridStat's status routers which allows the data plane of status routers to adapt quickly to changing operational situations. A mode contains the forwarding rules corresponding to a set of subscriptions. A mode change globally changes the set of subscriptions quickly without the need to recalculate routes, allocate resources, etc. Because such management calculations are complex, they take a significant amount of time for each created subscription. Adding a large number of subscriptions *during* a crisis or contingency, while possible, could result in unsatisfactory delays in providing relevant data to subscribers and thus could cripple the ability to add a few new subscriptions that could help "drill down" and discover the cause of the disturbance. Subscription modes get around this problem by allowing pre-contingency planning for communication needs in addition to the required practice of pre-contingency planning for responses in the power network.

Multicast has been mentioned previously as a key requirement for GridStat. It provides the property that *any entity with a legitimate need for data should be able to receive it in a timely fashion* while limiting the resources needed to support the property. Spatial redundancy—provisioning multiple disjoint paths from publishers to subscribers—is indirectly implemented by the status routers but no explicit mechanism is needed for this; it is simply a consequence of appropriate forwarding rules being established in the status routers based on routing computations performed by the QoS Brokers, [9]. GridStat implements the requirement for supporting heterogeneous rate delivery with rate filtering mechanisms in the status router.

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

Since one potential use of the status dissemination network is to support applications using synchronous phasor measurements there is a subtlety in rate filtering for synchronous phasor measurements: a large part of their benefit comes from the fact that, being taken simultaneously throughout the power grid, they

can be compared to compute, for example, voltage angles. Rate filtering is necessary for PMU data because these devices can produce 1–4 readings per power cycle, yet data can only be typically used (at least today) at one or two orders of magnitude lower rate. Without rate filtering, many status updates from PMUs (and other sources) would be wasting significant bandwidth. It would be most unfortunate, however, if the rate filtering were to filter update streams from different sources differently—delivering, say, 4 updates/s from one stream timestamped at 0, 250, 500, and 750 ms past the second while delivering update events from another stream timestamped at 125, 375, 725, and 875 ms past the second. GridStat's rate filtering is designed so that subscriptions with identical rate requirements for status variables with compatible publication intervals result in identical timestamps on the delivered update events for all subscriptions, [8], [10], [11].

We call this property *temporal synchronism*. It means that a state measurement program can collect PMU data from widespread locations and be assured that, despite rate filtering, the measurements that arrive from each PMU are taken at the same GPS time, for example all at 0, 250, 500, and 750 ms past the second.

C. Implementation

The GridStat prototype consists of Java implementations of all of the entities mentioned above. In addition there are C# publisher and subscriber implementations that interoperate with the Java status router implementations.

The next section reports performance measurements made on the prototype implementation. It should be noted that Java imposes a number of performance consequences on the system that affect the reported results.

- Interpretation overhead for virtual machine byte codes compared with native code.
- Just-in-time (JIT) compilation overhead for converting byte codes to native code.
- Garbage collection overhead.

As a result, we can be confident that the performance reported here (which is based on early 2004 hardware) is a lower bound on what the architecture is capable of achieving. Even with the Java implementation it is clear that performance requirements of many current and envisioned power system monitoring and control applications given in [6] are within the capabilities of the prototype.

IV. Experimental Results

In this section we overview the results of extensive performance experiments on the GridStat prototype. For more details, see [8], which includes 74 pages of detailed experimental results.

There are two primary metrics of interest in this overview. The first, *forwarding latency*, is the amount of time it takes an SR to forward an update event. The second, *load scalability*, is the number of update forwards per second which can be sustained without degrading forwarding latency. Other metrics such as the *drop rate*, the fraction of forwarding events dropped by an SR, were measured extensively but are not presented here in detail for reasons of brevity.

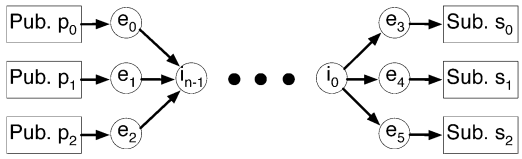


Fig. 2. Baseline experimental topology.

The experiments were conducted in a rack-mounted system whose main computers are 19 Dell Power Edge 1750s with a Dual Xeon 2.8 GHz CPU, 533 MHz system bus, 1 GB of DRAM, and a triple 1 Gb/s Ethernet interface connected by switched Ethernet. The system software included Fedora 2 Linux (kernel 2.6.10-1.9_FC smp) and Java 2 Platform Standard Edition 5.0 (version 1.5.02) from Sun Microsystems, using UDP sockets.

A. Baseline Forwarding Experiments

The experimental topology for the baseline forwarding experiments is given in Fig. 2. In our experiments, the performance metrics are measured on a *reference system*, a single publisher p_0 and a single subscriber s_0. These subscription endpoints are on the same computer so the end-to-end latencies can be measured very accurately using the local clock. Additionally, there are two *load systems*, which consist of two pairs of publishers and subscribers, (p_1, s_1) and (p_2, s_2). Each publisher and subscriber is connected to an *edge status router (ESR)*; these are denoted e_i. In these experiments, the reference system and the load systems share a chain of *n* SRs, *backbone status routers (BSRs)*, denoted i_0 through i_{n-1}.

Fig. 3 depicts the forwarding latency and load scalability observed in our experiments. Here, the number of BSRs, *n*, is varied from 1 to 7, and both the forwarding latency and its standard deviation increase close to linearly with *n*. With a load of approximately 18 K forwards/s, the forwarding latency for each BSR (derived from the slope from 1 to 7 BSRs) is 0.360 ms. When the load is approximately 50 K forwards/s, the forwarding latency for each BSR is 0.532 ms. We do not present results beyond approximately 50 K forwards/s, because at this point the BSR's CPU gets overloaded, causing the forwarding latency increases more than linearly and the drop rate (well below 1% at lower rates) increases drastically.

B. Multicast Forwarding Experiments

The previous experiments measured costs associated with forwarding update events through a linear chain of BSRs (i.e., a point-to-point communication). In this experiment, we generalize this to measure multicast performance, where BSRs forward each update to multiple outgoing destinations. The experimental topology for these experiments is given in Fig. 4. The BSR i_0 through i_{n-1} each has two outgoing communication links. Subscribers s_1 through s_n subscribe to all variables published by both p_1 and p_2. As in the baseline experiment publisher p_0 and subscriber s_0 constitute the reference system.

Fig. 5 depicts the forwarding latency and load scalability observed in our experiments. Here, the number of BSRs, *n*, varies from 1 to 4, and both the forwarding latency and its standard deviation increase close to linearly with *n* for a load of 9 K and 18

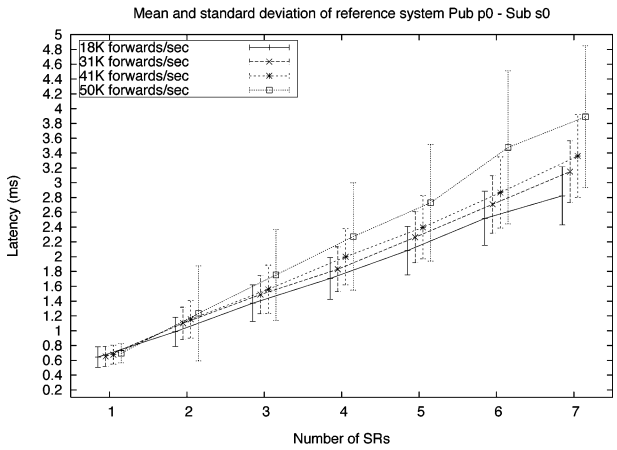


Fig. 3. Baseline forwarding latency and load scalability.

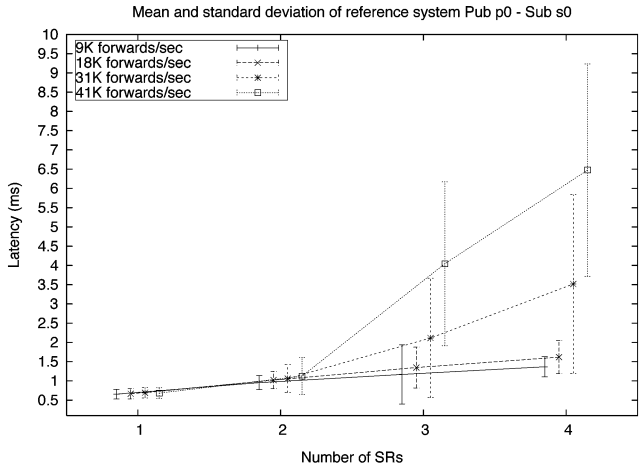


Fig. 5. Multicast forwarding latency and load scalability.

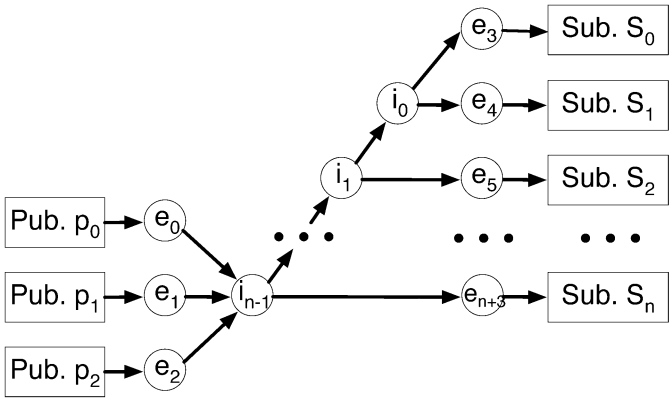


Fig. 4. Multicast experimental topology.

K forwards/s. With a load of approximately 18 K forwards/s, the forwarding latency for each BSR (derived from the slope from 1 to 4 BSRs) is 0.317 ms. When the load is ~41 K forwards/s, the forwarding latency for each BSR is 1.932 ms. For this experiment the CPUs got overloaded with a lighter forwarding load (due to the multicast) so the 50 K forwards/s is not presented. Note that for $n = 1$ no multicast occurs; it only occurs when $n > 1$. The sharp increase in the end-to-end latency for the 31 K and 41 K forwards/s can be explained by the CPU load on i_1 through i_{n-1} for these experiment runs. Due to a limitation of a Java library class the garbage collector is extensively executed for these runs (for more detail see [8]). We expect this sharp increase to be greatly reduced in the C language version of the status routers that we are presently building.

C. Discussion

The GridStat prototype demonstrates that the flexibility and QoS management needed for a next-generation SCADA replacement is feasible on off-the-shelf commodity computers and portable software (Java code). However, much better performance is achievable (these numbers are conservative), for many reasons. First, the computers used in the experiments were state-of-the-art as of early 2004, but today even a mid-level PC is considerably faster, and as of 2007 have quad cores (4 processors). The forwarding latency reported above also includes Ethernet latency through the switch to the next SR. Second, using Java incurs overheads associated with byte-code interpretation and just-in-time compilation as well as delays due to garbage collection. We are starting a second version of the SR prototype in C, that will eliminate these overheads and hence be faster.

For today's applications, handling a sustained load of 50 K forwards/s in a small or medium-sized substation's communication processor is likely to be sufficient. Additionally, this forwarding latency of approximately 0.5 ms is also adequate for many situations, including those given in the IEEE 1646 specification and the QoS requirements in [6]. For example, a path spanning a control area 800 km across might go through 5–10 SRs, for a cumulative latency of approximately 2.5–5.0 ms in addition to the propagation time (3.9 ms for this distance). Thus, not counting latency in the destination application program, overall latency of approximately 6.4–8.9 ms is achievable. Thus, GridStat allows wide area communication for new kinds of remote protection or control at a latency that is a small fraction of the 3 to 4 ac cycles typically required for locally triggered breaker opening today.

This performance also opens up new possibilities for remote protection and a new class of controls to contain cascading, even with the above numbers on Java and 2004 vintage hardware. [12] predicts that disturbances travel at 500 km/s in an example system with 64 generators, and different kinds of fast controls such as transient stability controls typically need to react in as little as 100 ms and as much as 1–2 s. GridStat can in realistic situations deliver data much faster than electrical disturbances, but much research would be required to develop best practices on how and when such remote schemes would be beneficial.

The above experimental results were conducted using general purpose computers. Network processors are another hardware platform that could be used for status routers, [13]. A typical network processor features a single, general purpose CPU and a number of specialized packet processors called Micro-Engines—specialized CPUs that are optimized for network protocols. They can perform many kinds of activities related to packet forwarding, such as computing UDP checksums, in a

single instruction cycle. A network processor interfaced to several Ethernet and fiber inputs can do IP forwarding between them at line speeds (1–10 Gb/s or more).

We are presently wrapping up an initial investigation into how fast GridStat routing can be performed on a network processor [14]. The main goal of this project is to see how fast GridStat-style routing can be on network processors rather than to implement the full functionality of today's Java-based status router. The initial experimental results were conducted on Intel's simulator configured to mimic the Radisys ENP2611 PCI card. The ENP2611 integrates an Intel IXP2400 network processor chip having 8 packet processors, 4 network connections (1 at 100 Mb/s, 3 at 1 Gb/s), and buffer and program memory on a PCI card that plugs into a host computer. The simulator is a full implementation of the IXP2400 chip that also faithfully simulates the externals such as the network links and the various on-board memories. The simulator runs the same binary code as the hardware itself and makes it possible to obtain exact cycle counts and instruction traces for status router activities.

The ENP2611 is capable of performing a simplified GridStat routing algorithm, supporting only subscription periods that are a power of 2 ms, at 1 Gb/s line speeds.[1] This corresponds to approximately 2 million GridStat update forwards per second per output link, using the same packet size (62 bytes) as the Java experiments reported above. Measured forwarding delays ranging from 1–3 μs have been obtained on different experiments.

These numbers suggest that the overhead of GridStat could be negligible: in any wide area deployment the forwarding latency added by GridStat would be much smaller than the propagation time (i.e., the speed of light over the distances involved) and the application program latencies at the endpoints. The throughput is also very high. It is important to remember, however, that this prototype does not yet implement the full functionality of GridStat's Java status routers. More research is needed to evaluate ways of providing complete GridStat functionality on newer network processors, including finding ways to provide the temporal synchronism needed by PMU data streams.

Finally, in this context we feel it is important to comment on a practice in the power grid today. Often sensor data is placed directly into a database, and then different applications retrieve if from there. We note that putting a database on the critical path for data delivery adds unnecessarily to latency, so the data is less fresh when it finally gets to the application that can act upon it. Just storing and then retrieving it from a database adds to the end-to-end latency (from the sensor to the application). Worse, the application often must explicitly pull the data from the database, which means a round trip network message. With a publish-subscribe system such as GridStat, the data is pushed out when it arrives, and only a one-way network latency is necessary, potentially saving up to half of the long-distance networking costs.

V. Related Work

GridStat is unique in a number of respects: its explicit support and management for streams of status updates, its integrated rate filtering and multicast mechanisms, its integrated QoS and disjoint path routing algorithms, and the fact that it was designed explicitly for the electric power grid. There are some related efforts in the power industry. The IEC 61850 specification includes an elaborate and elegant model-driven approach to substation automation, including self-describing substation devices, [15]. It "provides a comprehensive model for how power system devices should organize data in a manner that is consistent across all types and brands of devices." Within its current scope, it seems to have great potential. We believe that GridStat and IEC 61850 are highly complementary: GridStat provides wide-area delivery (and could easily be extended to deliver messages in the standard's GOOSE message format), while IEC 61850 does not, itself, define any wide-area delivery mechanisms. IEC 61850-capable devices live at the edges of the communication network. There is a consensus in industry that, at the present, more device configuration applications are needed to help exploit the full potential of IEC 61850. These applications can serve as effective tools in deploying devices using this standard. But this exact same kind of device type information could be very useful with tools to help configure and manage a GridStat-like, sophisticated communication infrastructure for the power grid.

An information architecture for the power grid is proposed in [16]. It contains proposals for different ways to structure interactions between control centers and substations, and reliability analyses of different schemes. However, it does not propose any communications mechanisms, and relies on off-the-shelf network technology whose reliability and latencies are not controllable and which does not meet most of the requirements outlined in [6].

In computer science research there has been some work related to GridStat. The closest is PASS, [19]. PASS provides a limited form of status dissemination, specifically, binary "up/down" status dissemination for remote devices. It does not provide QoS management, support heterogeneous delivery rates, or provide redundant delivery paths (though is very useful for its intended domain: wide-area military networks in the field with low bandwidth). Other publish-subscribe systems do not provide any kind of rate filtering, because they do not capture the semantics of a status variable flow. A discussion of additional research in computer science disciplines that is related to GridStat can be found in [6], [8].

VI. Conclusion

The communication infrastructure in today's power grid is based on the technology of the 1960s. It greatly limits the kinds of protection and control that can be employed and hampers the situation awareness of the grid's operators. The communication infrastructure is being augmented in a piecemeal fashion by newer network technologies such as optical fiber and Ethernet. However, to date there has been no utilization of advances in distributed computing technologies, that offer much more flexibility, portability, and adaptability than can be achieved using network-layer technology.

GridStat is a publish-subscribe middleware framework that has been designed to meet the data delivery requirements for the electric power grid. Its performance is adequate for today's monitoring and control requirements.

[1]The restriction to powers of 2 eliminates the need to divide by arbitrary numbers, an operation that is very expensive on this hardware.

For more details on ongoing and future work, see [6].

ACKNOWLEDGMENT

The authors would like to thank S. Abelsen, G. Hauser, L. Hoffman, N. Schubkegel, K. Swenson, and E. Viddal for their comments on drafts of this report, and N. Schubkegel for his artwork for some of the figures. The authors thank J. Dagle, H. Gill, D. Kopczynski, M. Larsson, P. Overholt, B. Sanders, E. Schweitzer, N. Suri, K. Tomsovic, and M. Venkatasubramanian for their useful comments on this research. They thank Schweitzer Engineering Labs for their donation of equipment and PNNL for use of equipment.

REFERENCES

[1] C. Hauser, D. Bakken, and A. Bose, "A failure to communicate: Next generation communication requirements, technologies, and architecture for the electric power grid," *IEEE Power Energy Mag*, vol. 3, no. 2, pp. 47–55, Mar./Apr. 2005.

[2] K. Tomsovic, D. Bakken, M. Venkatasubramanian, and A. Bose, "Designing the next generation of real-time control, communication and computations for large power systems," *Proc. IEEE*, vol. 93, no. 5, pp. 965–979, May 2005.

[3] K. Geihs, "Middleware challenges ahead," *Computer*, vol. 34, no. 6, pp. 24–31, Jun 2001.

[4] J. Zinky, D. Bakken, and R. Schantz, "Architectural support for quality of service for CORBA objects," *Theor. Practice Object Syst.*, vol. 3, no. 1, pp. 55–73, Apr. 1997.

[5] Quality Objects (QuO) homepage 2006. [Online]. Available: http://quo.bbn.com.

[6] 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," School Elect. Eng. Comput. Sci., Washington State University, Tech. Rep. EECS-GS-009, May 2007. [Online]. Available: www.gridstat.net/TR-GS-009.pdf.

[7] M. Schumacher, C. Hoga, and J. Schmid, "Get on the digital bus to substation automation: Next-Generation measuring systems with a real-time ethernet process bus," *IEEE Power Energy Mag.*, vol. 5, no. 3, pp. 51–56, May/Jun. 2007.

[8] K. H. Gjermundrød, "Flexible qos-managed status dissemination framework for the electric power grid," Ph.D. dissertation, Washington State Univ., Pullman, 2006.

[9] V. S. Irava, "Low-cost delay-constrained multicast routing heuristics and their evaluation," Ph.D. dissertation, Washington State Univ., Pullman, WA, Aug. 2006.

[10] R. A. Johnston, "Obtaining high performance phasor measurements in a geographically distributed status dissemination network," M.S. dissertation, Washington State Univ., Pullman, Aug. 2005.

[11] R. A. Johnston, C. H. Hauser, K. H. Gjermundrød, and D. E. Bakken, "Distributing time-synchronous phasor measurement data using the gridstat communication infrastructure," presented at the 39th Annu. Hawaii Int. Conf. Syst. Sci., Kauai, HI, Jan. 4–7, 2006.

[12] J. Thorp, C. Seyler, and A. Phadke, "Electromechanical wave propagation in large electric power systems," *IEEE Trans. Circuits Syst.*, vol. 45, no. 6, pp. 614–622, Jun. 1998.

[13] D. Comer, *Network Systems Design Using Network Processors*. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[14] K. Swenson, "High performance gridstat status routing using network processors," M.S. dissertation, Washington State Univ., Pullman, Aug. 2007.

[15] R. Mackiewicz, "Technical overview and benefits of the IEC 61850 standard for substation automation," in *Proc. Power Syst. Conf. Expo.*, Oct. 29–Nov. 1 2006, pp. 623–630, IEEE.

[16] Z. Xie, G. Manimaran, V. Vittal, A. Phadke, and V. Centeno, "An information architecture for future power systems and its reliability analysis," *IEEE Trans. Power Syst.*, vol. 17, no. 3, pp. 857–863, Aug. 2002.

[17] The Integrated Energy and Communication Systems Architecture vol. IV, 2004. [Online]. Available: http://www.epri.com/IntelliGrid/.

[18] W. Mittelstadt, P. Krause, P. Overholt, D. Sobajic, J. Hauer, R. Wilson, and D. Rizy, "The DOE wide area measurement system (WAMS) project—demonstration of dynamic information technology for the future power system," presented at the EPRI Conf. Future Power Delivery, Washington, DC, Apr. 1996.

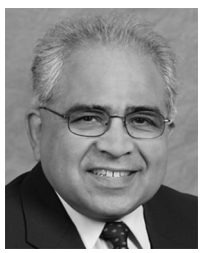
[19] J. Zinky, L. O-Brien, D. Bakken, V. Krishnaswamy, and M. Ahamad, "PASS-a service for efficient large scale dissemination of time varying data using CORBA," in *Proc. 19th IEEE Int. Conf. Distributed Comput. Syst.*, Austin, TX, Jun. 5, 1999, pp. 496–506, IEEE.

Harald Gjermundrød (M'99) received the B.S., M.S., and Ph.D. degrees in computer science from Washington State University, Pullman, in 1999, 2001, and 2006, respectively, and the Dipl.-Ing. degree from Oslo University College, Oslo, Norway, in 1998 (including a year as an Socrates/Erasmus student at the Robert Gordon University).

Currently, he is a Postdoctoral Associate of Computer Science at the High-Performance Computing Systems Laboratory at the University of Cyprus. His research focuses on distributed computing systems, middleware, and grid computing. He has worked on projects funded by the National Institute of Technology and the National Science Foundation in the U.S. and was a Reviewer for several scientific conferences.

David E. Bakken is an Associate Professor of Computer Science with the School of Electrical Engineering and Computer Science, Washington State University (WSU), Pullman. His research interests include middleware, distributed computing systems, fault tolerance, and quality-of-service frameworks. Prior to joining WSU, he was a Scientist at BBN Technologies where he was an original co-inventor of the Quality Objects (QuO) framework. He has been a Consultant for Amazon.com, Network Associates Labs and others, and has also worked for Boeing.

Carl H. Hauser is an Associate Professor of computer science with the School of Electrical Engineering and Computer Science, Washington State University (WSU), Pullman. His research interests include concurrent programming models and mechanisms, networking, programming language implementation, and distributed computing systems. Prior to joining WSU, he worked at the Xerox Palo Alto Research Center and IBM Research for a total of more than 20 years and was a coauthor of a seminal paper on epidemic multicast algorithms.



Anjan Bose is a Regents Professor in Power with the School of Electrical Engineering and Computer Science at Washington State University (WSU), Pullman. His research interests include energy control centers, power systems analysis, and power system operations. He has worked for Consolidated Edison and for Control Data.

Mr. Bose is a member of the U.S. National Academy of Engineering.