CPS-Net: In-Network Aggregation for Synchrophasor Applications

V. Arya[1], J. Hazra[1], P. Kodeswaran[2], D. Seetharam[1], N. Banerjee[1], and S. Kalyanaraman[1]

[1]IBM–Research, Bangalore, India, {vijarya1, jaghazra, dseetharam, nilanjba, shivkumar-k}@in.ibm.com
[2]University of Maryland, Baltimore County, MD, USA, palanik1@cs.umbc.edu

***Abstract*—Synchrophasors are sensors that sample power grids and publish these measurements over a network to a number of grid applications such as voltage monitoring, state estimation, visualization, etc. The sampled data is QoS sensitive and must be delivered reliably with minimal delays to the target applications. However, during network overloads or grid emergencies when the volume of data transmitted is high, it is important to gracefully degrade performance and data stream delivery in an application-specific manner.**

We propose CPS-Net, a flexible 3-layered network architecture that allows application-specified in-network aggregation of synchrophasor data streams during overload. The lowest layer provides basic path-specific QoS while the middle layer provides real-time wide-area publish-subscribe capabilities integrated with traffic engineering of data streams across multiple lower level paths and trees. The top layer provides a distributed stream processing infrastructure for application-specified aggregation functions. During network overload, the lower layer triggers the co-optimization of higher layers and application-specific aggregation of data is performed. The user is presented with a simple stream processing programming model and the details of the network, placement and composition of operators are abstracted away.

Initial simulation results, using a voltage stability monitoring smart grid application, show that CPS-Net architecture can gracefully degrade data streams for synchrophasor applications.

I. Introduction

Cyber-physical systems (CPS) combine sensing, networking and remote computation in the context of a physical system. In this paper, we focus on a class of wide-area, QoS sensitive cyber-physical systems and are motivated by the case of interconnecting synchrophasor data (publishers) with real-time Smart Grid applications (subscribers). Synchrophasors or Phasor Measurement Units (PMUs) are the new sensors that sample the electricity grids at 30-60Hz and publish these measurements as streams that need to be delivered reliably and in real-time to a number of Synchrophasor applications. A wide variety of synchrophasor applications have been proposed and the QoS requirements of these applications have been classified by the North American Synchrophasor Initiative (NASPI) into a set of classes [1].

While building a network that satisfies the basic QoS requirements is a well studied problem, the dimensions that make these networks interesting are the fact that:

- Application requirements need to be mapped onto a real-time wide-area publish-subscribe architecture requiring QoS support beyond simple point-to-point QoS.
- During overloads or critical events when sampling rates increase or more PMUs are active, it is important to gracefully degrade performance and data stream delivery in an application-specific manner.

978-1-4244-8953-4/11/$26.00

Graceful degradation of performance and QoS for many-to-many real-time, wide-area streams is hence the focus of this paper. We propose CPS-Net, a flexible 3-layered architecture that leverages the benefits of layering and point-to-point QoS, while allowing application-specified in-network aggregation of data streams during overload. The bottom layer provides basic path-specific QoS. The middle layer provides real-time wide-area publish-subscribe capabilities, integrated with traffic engineering of data streams across multiple lower level paths and trees. The top layer provides a distributed stream processing infrastructure for application-specified aggregation that helps in graceful degradation during network overload.

During underload, the top layer is quiescent, and all the PMU data from publishers is sent to subscribers. But, during network overload, there may not be sufficient capacity to deliver all the PMU data. One response to overload would be to randomly discard data: however this could degrade performance in unpredictable ways. From video streaming literature, we know that if information can be dropped in an application-sensitive manner, then the quality of experience for multimedia applications can be gracefully degraded as a function of the level of overload. Analogously, we aim to provide application-sensitive in-network aggregation functions that could be used during overload periods to achieve graceful degradation of synchrophasor applications. Specifically, the lower layer of our three-layer architecture during overload triggers the co-optimization of higher layers, and application-specific filtering and/or aggregation of data is performed.

The subscribers, while subscribing to specific content, can specify the data aggregation and filtering mechanisms that they are willing to accept, types of data to which such mechanisms can be applied to, and the timeframes during which those mechanisms are acceptable. The application writer would be best situated to express such aggregation and filtering functions, but would need a convenient API to express them. We propose to provide the application writer with a simple declarative API, based upon a stream computing programming model such as streamIt [2] or Spade and Infosphere Streams [3]. The declarative view is expressed in the stream

programming language while the details of the network, placement and composition of operators are abstracted away, as part of the distributed stream computing system.

Initial simulation results show that the CPS-Net architecture can gracefully degrade data streams for real-time synchrophasor applications during network overload and this is illustrated using a voltage monitoring smart grid application.

The rest of the paper is organized as follows. Section II motivates the architecture of CPS-Net through a grid application that would benefit from application specific in-network aggregation invoked during network overload. Section III describes the 3-layer architecture of CPS-Net. Section IV presents a declarative model that allows users to define application specific aggregation functions. Section V presents experimental results and section VI reviews prior work. Section VII concludes the paper with discussions and future work.

II. Motivation

CPS-Net is suitable for systems that have the following properties: (i) Large amounts of data flow from sensors to multiple subscribers. For most of the time, data transmission is periodic. But, under specific alarm conditions, there are synchronous bursts. (ii) Data from multiple sensors can be aggregated together or delayed. Aggregated or delayed data will still contain meaningful and relevant information. (iii) The applications can work with both raw and aggregated data streams. (iv) The applications can gracefully degrade their performance when the incoming data flow is quenched.

Many smart grid applications exhibit the above properties. We will use *Voltage Monitoring System*, a common smart grid application, to motivate CPS-Net. A voltage monitoring system measures voltage stability which is a serious concern for grid operators due to its importance in ensuring system security and power quality. Loss of voltage stability, also known as a voltage collapse, can either result from the inability of the power system to supply reactive power or by an excessive absorption of reactive power by the system.

Grid operators use a metric known as *Voltage Stability Index* (VSI) to determine the probability of a voltage collapse and to identify vulnerable buses (A bus is also known as a substation. One can think of the grid as a graph with buses as nodes and transmission lines as edges). VSI of bus i is defined as [4]:

$$VSI_i = \frac{\partial P_i / \partial \delta_i}{\sum_{j=1 j \neq i}^{n} B_{ij} V_j} \quad (1)$$

Where, n is number of buses in the system, P_i is the real power injected at bus i, V_j is the magnitude of voltage at bus j, δ_i is the phase angle of voltage at bus i and B_{ij}'s are elements of the network admittance matrix. $B_{ij} = 0$ if no transmission line connects bus i directly to bus j in the grid. The stability index of a grid is calculated as the minimum of the stability indices of all its buses and therefore depends on the most vulnerable bus. A *low* value of VSI indicates *instability* while a *high* value indicates *stability*. A grid is considered stable if VSI > 0.5 while the normal operating range is larger than 0.8. When VSI < 0.5, the grid experiences a voltage collapse.

A. Aggregation for Graceful Degradation

In a voltage monitoring system, PMUs measure and publish streams of voltage parameters (magnitude, phase angle) of buses within the grid. Applications subscribe to these streams in order to compute the VSI's of buses. During network overload, PMU streams traveling from the publishers toward a subscriber can be aggregated at intermediate nodes to minimize load. Rest of this section shows that the voltage monitoring system, like a number of other smart grid applications, can safely work with such aggregated data. We give a list of aggregation functions that can be applied in-network during network overload, thus gracefully degrading the quality of VSI's computed by the subscribers.

Data Prioritization Not all data from all PMUs are equally important for computing the VSI. In particular, following heuristics can be used to drop low priority data and reduce network load: (i) Data from PMUs at generating substations could be given a lower priority as they rarely have voltage problems. (ii) Data from PMUs deployed at transmission/distribution substations whose reported values are below their rated values must be given higher priority because such a dip indicates proximity to voltage collapse. Following steps can be used at intermediate nodes for data prioritization:

1: Collect PMU voltage magnitudes $X_t = [v_1, v_2, ..., v_n]$ (in per units) measured at a common time instant t.
2: Sort X_t in ascending order.
3: **for** $i = 1$ to n **do**
4: **if** $X_t^i <= v_{th}$ **then**
5: select X_t^i
6: **end if**
7: **end for**

where, v_{th} is the voltage threshold that is usually set to 1.0. The filter selects, for transmission, only data packets whose values are below the safety threshold.

Data Dropping Generally voltage phasor measurements do not change abruptly unless there are disturbances or faults in the system. During normal operation, the state of the system changes gradually, if at all. Therefore during overload, PMU streams whose data varies minimally (indicating normal conditions) can be dropped as follows:

1: Let $X_t = [x_t^1, x_2^2, ..., x_t^n]$ and $X_{t-1} = [x_{t-1}^1, x_{t-1}^2, ..., x_{t-1}^n]$ be the data at time instants t and $t-1$ respectively
2: Check data similarity of two consecutive two time stamps
3: **for** $i = 1$ to n **do**
4: **if** abs$(x_t^i - x_{t-1}^i) <= x_{th}^i$ **then**
5: drop data x_t^i
6: **end if**
7: **end for**

where, x_{th}^i is a threshold. A rule of thumb is to use 0.005 per unit for voltage and 0.5 degree for phase angles.

Data Clustering Several PMUs in the grid may report voltage values that are numerically close to each other. These could be clustered to reduce data volume as follows:

1: Let $X_t = [x_1, x_2, ..., x_n]$ be the data of time instant t. Let k be the number of clusters.
2: Choose k random data points as initial cluster center μ
3: Calculate the distance, $d(i,j)$ from the center of each cluster to each data point
4: **for** $i = 1$ to k **do**
5: **for** $j = 1$ to n **do**
6: $d(i,j) = (\| x_j - \mu_i \|)^2$
7: **end for**
8: **end for**
9: Assign each data point, x_j to the cluster i where $d(i,j)$ is minimal
10: Compute mean center of each cluster μ_i for all clusters
11: Repeat Steps 3-10 until all the data points are assigned to their nearest cluster centers.

Partial Computations VSI computations can be performed in a distributed manner since VSI of a bus depends only on the measurements at that bus and those buses directly connected to it. Therefore instead of forwarding raw PMU streams, nodes can participate in computations and only forward partially computed results. With usual notation, Eq. (2) shows that VSI of bus i can be estimated using partial summations that can be computed at different intermediate nodes of the network:

$$\frac{1}{VSI_i} = \frac{\sum_{j=1, j\neq i, j\in A}^{n} B_{ij}V_j}{\partial P_i/\partial \delta_i} + \frac{\sum_{j=1, j\neq i, j\notin A}^{n} B_{ij}V_j}{\partial P_i/\partial \delta_i} \quad (2)$$

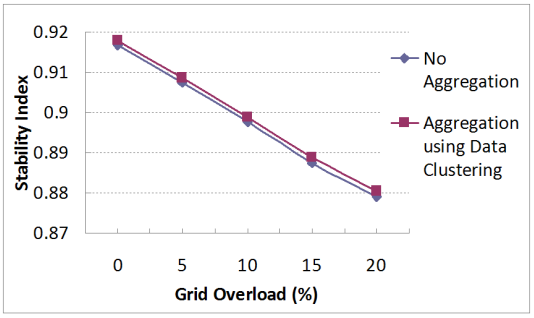

Fig. 1. Graceful degradation: Impact of in-network aggregations on the voltage stability index (VSI) computed at subscribers.

Figure 1 shows the results of performing application specific in-network aggregation when VSI of a critical bus in the grid was computed using both raw and clustered data (details of simulation setup are available in section V). VSI of the bus is shown under different operating conditions when the grid is overloaded by up to about 20%. Aggregation in these experiments reduced network traffic by as much as 40%. It is clear from the figure that in-network aggregation impacts VSI computation only by a negligible factor and therefore can safely be used to monitor the grid. Depending on the smart grid application, a number of different application-specific aggregation functions can be applied in-network to reduce network load and gracefully degrade application performance.

III. System Architecture

In this section we propose an adaptive architecture for smart grid applications. The proposed system architecture consists of the following three layers above a best-effort IP Network.

- *Application Layer / Stream Processing Layer* is the topmost layer that is responsible for specifying smart grid application requirements (e.g., QoS, Security, etc.), stream aggregation and distributed stream computations possible for a given application.
- *Publish-subscribe overlay* is the middle layer that disseminates data streams from one or more sources (e.g., sensors) to one or more subscribers with certain QoS level guarantees such as minimum end-to-end latency and data rate by leveraging underlying network level resources.
- *Data flow layer* is the bottom layer that handles IP level flow reservations and interacts with the publish-subscribe overlay to reserve suitable paths for the flows corresponding to the data streams with certain objective parameters such as bounded end-to-end latency, data rate and minimum energy expenditure, etc.

To provide necessary QoS to the smart grid applications, the three layers coordinate as follows. Smart grid applications subscribe for types of data they are interested in. While they register, they also specify their set of QoS requirements and the set of aggregation and distributed computing functions that they can tolerate. During normal network (underload) conditions, the middle layer ensures that the necessary QoS is provided to the application flows by leveraging flow based reservation mechanisms available in the Data Flow Layer. During network overload conditions, a trigger propagates from the Data Flow Layer to the upper layers. If the overload cannot be handled by the middle layer, it propagates the trigger further upward to the Streams Processing Layer, which triggers the aggregation and distributed computing functions to adapt to the external conditions (network congestion, higher energy prices, etc). The following sections explain the role of each individual layer in detail.

Data Flow Layer IP networks, based on a best-effort model, require auxiliary flow reservation mechanisms such as IntServ (a fine-grained approach that provides QoS to individual application or flow) or DiffServ (a coarse-grained approach that provides QoS to a large class of data or aggregated traffic), to ensure timeliness, reliability and availability in data delivery.

In our architecture, under network underload conditions, flows are reserved at IP layer with adequate resources to guarantee application requirements specified by the middle layer. In overload conditions, the flow reservation service raises an exception to the upper layer i.e. the middle layer.

Publish-Subscribe Layer The middle layer of our architecture consists of a publish-subscribe overlay that sets up data streams between the publisher of information (sensors) and the subscribers interested in that data (smart grid applications). The publish-subscribe communication paradigm provides a loosely coupled interaction model that is best-suited for large scale distributed systems such as smart grids [5]. Typically,

subscribers express their interest in an event or a topic and are subsequently notified of any event generated by an independent publisher, which matches the registered interest. The events are asynchronously propagated in the network and there is a full decoupling in time and space between the publisher and the subscriber.

Publish-subscribe systems do not provide QoS guarantees inherently, which is a critical requirement for smart grid applications. As a result, an auxiliary mechanism is required for a publish-subscribe middleware to provide QoS guarantees such as end-to-end latency, data availability etc. We use QoS-aware publish-subscribe systems such as Harmony [6] that employs a multi-hop overlay of brokers to select overlay paths that meet application latency requirements, on a per topic basis, while also maximizing the successful delivery of each topic. The publish-subscribe overlay, under network underload conditions, selects the best possible path from the Data Flow Layer to establish an overlay path that satisfies the application's QoS requirements. During overload conditions, when the overlay receives the trigger from the Data Flow Layer, it first tries to adapt to the overload by adjusting existing overlay paths or using multiple paths. If such an adaptation is not possible then it sends a trigger to the upper Streams Processing Layer.

Streams Processing Layer This layer has three functions: (i) It exposes a programming interface for smart grid application development where it can take the following parameters as inputs: type of data streams the application needs, QoS requirements of streams, and the network functions that can be employed for graceful degradation of applications in terms of data transfer requirements. We have proposed a Declarative Subscription model (described next in section IV) for exposing such a programming interface for application development. (ii) It registers the application requirements with the underlying publish-subscribe overlay layer. (iii) It configures the underlying overlay network to invoke aggregate functions on intermediate nodes to achieve graceful degradation of application performance.

IV. Declarative Subscription Model

CPS-Net exposes a programming interface that allows grid applications to subscribe to streams published by monitoring devices and build stream computing applications using stream computing platforms such as Spade, StreamIt, etc. Users can build applications that run either locally at a node or harness the potential of network stream computation.

For instance, a common grid application is Voltage Monitoring (section II) which requires voltage streams from several different PMUs. A user can write a hierarchical voltage monitoring application using Spade language which eventually gets expressed as a data-flow graph. The application can then run either locally at a node or in a distributed manner across the network in which case nodes function as stream operators. Furthermore, the result of the application, a stream of voltage stability indices is also publishable and can be subscribed to by other applications. As a consequence, a library of common grid applications can be built that subscribers can tap into and help reduce traffic and computational redundancies.

CPS-Net provides an interface that helps programmers specify QoS requirements of bandwidth and latency for their applications as well as aggregation functions invoked during overload for graceful degradation.

A. Publish and Subscribe

Any substation, sensor, or any stream application can publish streams by specifying attributes such as sampling frequency, bandwidth, syntax of data items, and information about the stream.

```
PID = PublishStream( Frequency 30Hz,
dataItem <timestamp, voltage>,
bandwidth <100bytes, 200bytes>,
Info "Voltage stream of PMU 23" );
```

where we have suffixed parameters with their type for ease of understanding. The above function publishes a PMU stream containing data items that hold timestamp and voltage values of sizes 100 and 200 bytes respectively, at a frequency of 30Hz. A call to the above function registers the stream and gives it a unique ID that can be used by the subscribers. Not all streams can specify a hard constraint on the rate at which they output data in advance. For instance, stream applications can be triggered by alarm conditions in the grid and may increase the rate at which they output data. Similarly sensor devices may react to conditions in the grid and increase their sampling frequency. To accommodate these requirements, the publish-subscribe and data-flow layers monitor the load and the stream rates to effectively meet bandwidth and latency constraints of published streams.

A stream can be subscribed to by a stream computing application as

```
Stream S = SubscribeStream( PublisherID PID,
dataItem <timestamp, voltage>, latency 10ms);
```

The dataItem is specified by the subscribers as well so that they may subscribe to a part of the published stream. To support the above, the publish-subscribe layer interacts with the data-flow layer and sets up a data forwarding path between the publisher and the subscriber with a 10ms latency bound.

B. Data Aggregation Operators

CPS-Net allows users to attach aggregation functions to streams. These are functions that are applied in-network during overload and allow applications to gracefully degrade their performance. During overload, there will not be sufficient capacity to deliver all PMU data and therefore streams would need to either loose data elements or reach the subscriber with an unacceptable latency. This can lead to performance degradation in unpredictable ways. To allow for graceful degradation, the network supports two forms of aggregation: (i) Marginal aggregation of individual streams, (ii) Joint aggregation of data items across multiple streams subscribed to by an application. Marginal aggregation can be used by an application when it has subscribed to a raw stream and can still benefit by receiving averages or medians of data items over

consecutive time windows during overload. Joint aggregation can be used when an application subscribes to multiple streams and aggregation of time-aligned data across different streams may be acceptable to the application. Examples of joint aggregation functions are data dropping, data prioritization, and data clustering functions discussed in the context of voltage monitoring in section II. To support this functionality, users are allowed to group streams into sets and attach aggregation functions to these as follows:

```
AttachAggregationOperator(
StreamSet G1,...,StreamSet Gn, AlgorithmList L);
```

The AlgorithmList L contains an ordered list of calls to aggregation algorithms, each of which can be invoked to achieve a different level of aggregation. Algorithms further down the list provide stronger aggregation. Using the above function, applications attach aggregation functions to sets of streams and intermediate nodes would support in-network aggregations whenever streams belonging to those sets cross the node. For instance, consider a voltage monitoring application that subscribes to three PMU streams X, Y, and Z. Let us assume that it accepts joint aggregation across streams Y and Z using a K-Means clustering algorithm and marginal aggregation on stream X using a window-averaging algorithm WinAvg. K-Means and WinAvg are essentially regular stream algorithms that could be user-defined or part of a streams library. K-Means is defined to accept a set of streams as input along with a parameter that specifies the number of clusters. Fewer clusters imply stronger aggregation. On the other hand, WinAvg is defined to accept a single stream as input along with a window size parameter. Larger window size implies stronger aggregation. The following call will then attach aggregation functions to streams:

```
AttachAggregationOperator([X],
[WinAvg(4), WinAvg(8)] );
AttachAggregationOperator([Y, Z],
[K-Means(8), K-Means(4), K-Means(2)] );
```

As a consequence, depending on network load, in-network aggregation of streams Y, Z will occur by using K-Means algorithm with parameters 8, 4, or 2. Similarly aggregation of X would occur using WinAvg algorithm with parameters 4 or 8. Although in the above example, same algorithms were used in the AlgorithmList, different algorithms may be used to achieve different levels of aggregation. In order to support the above in-network aggregation service, during overload, the application and publish-subscribe layers interact to perform two tasks: (i) A forwarding node N1 is located on the path from publisher of X to the subscribing application and the appropriate aggregation function invoked on node N1 (ii) A forwarding branch point N2 on paths from publishers of Y and Z to the subscriber is located, which forwards both the streams and the aggregation function is invoked on node N2.

V. Experiments

In this section we study the impact of different in-network aggregation functions on the voltage monitoring application.

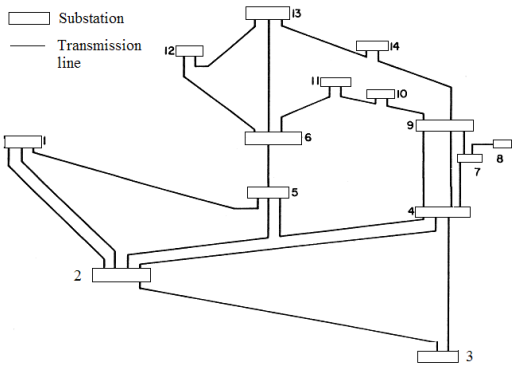

Fig. 2. A small *IEEE 14 Bus Test System* containing 14 buses/substations connected by 15 transmission lines. Experiments use a larger 300 bus system.

We simulated the benchmark IEEE 300-bus test system [7] (A small 14 bus system is shown in Fig. 2). The system has 300 buses/substations, 304 transmission lines, 107 transformers and 69 generating stations. It has three geographical zones. Zone 1 consists of 157 substations, zones 2 and 3 consist of 80 and 63 substations respectively.

Each bus/substation has a PMU that generates a stream of samples (in IEEE C37.118 format) at a rate of 120Hz. Each sample contains voltage and current phasors, line flows, and system frequency information. PMUs report their streams to respective local Phasor Data Concentrator (PDC) nodes. That is, PMUs in Zone 1 report measurements to PDC1, PMUs in Zone 2 to PDC2 and PMUs in Zone 3 to PDC3. These PDCs in turn forward the measurements to a central super PDC (SPDC) that hosts the voltage stability monitoring application. We allow in-network aggregation functions at each of the 3 PDC nodes in the network.

In order to verify the simulation setup and correctness of the monitoring scheme we gradually overloaded the grid from its normal load (23525 Megawatts) to its maximum load (24250 Megawatts). These figures and normal load conditions at each substation are available in [7]. Using load conditions in [7] as a starting point, we gradually overloaded the grid until its maximum, using a step size of 10 Megawatts. After each step, the voltage phasors for the new load are recomputed using the standard *Fast Decoupled Power Flow* method [8]. This is an iterative method that requires a termination criteria. We used a power convergence limit of 10^{-3} per unit.

As illustrated in Figure 3, the voltage stability indices of critical buses/substations decrease correctly in response to overloading. The figure shows that the bottom two curves corresponding to buses 178 and 179 have low stability indices at the normal loading. However, their stability indices do not change with increase in load as both substations have generators that always maintains constant voltage irrespective of increase or decrease of load. On the other hand, buses 280-289 are most affected by overload and can be marked as critical buses for triggering preventive control. Since bus 282 is the most critical bus, we plot its VSI in the next experiment.

We evaluate the impact of in-network aggregation of PMU streams on the accuracy of VSI calculations using aggregation algorithms described in section II. Figure 4 shows

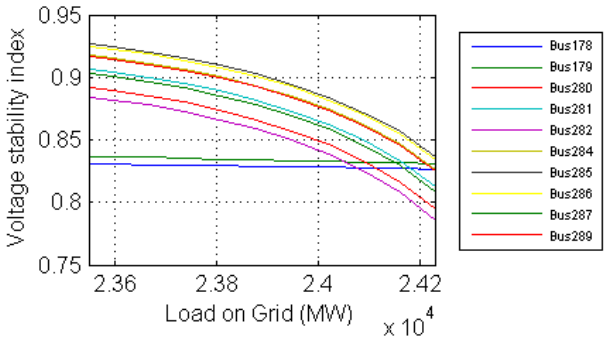

Fig. 3. Falling Voltage Stability Indices of 10 buses of IEEE 300 bus system in response to grid overload. Bus 282 has the lowest stability index.

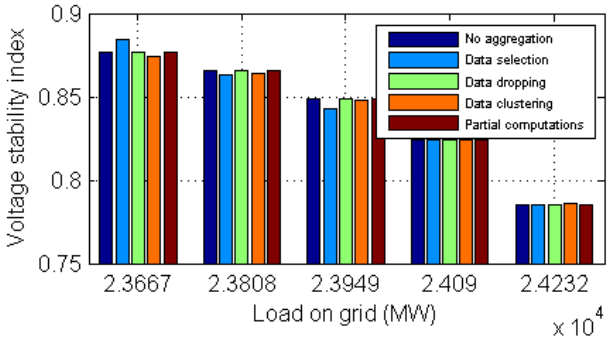

Fig. 4. Voltage Stability Index of bus 282 computed using a number of different in-network aggregation algorithms.

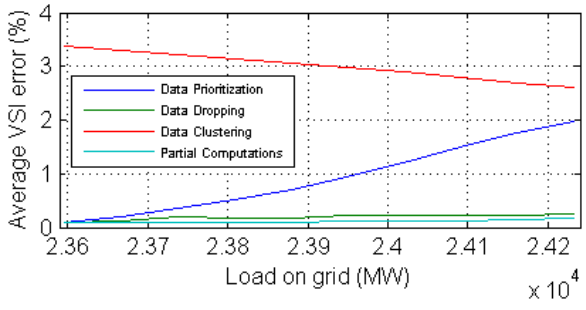

Fig. 5. Errors in Voltage Stability Index averaged over 300 buses with different in-network aggregation algorithms.

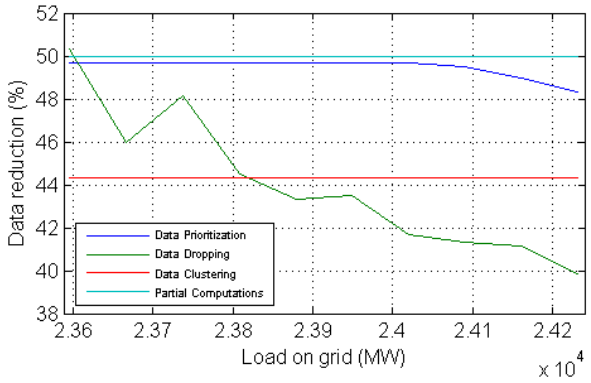

Fig. 6. Reduction in traffic volume with different in-network aggregation algorithms.

the computed VSI for bus 282 under different operating conditions when the grid is overloaded. We can see that the aggregation functions alter the VSI only by a small factor. The VSI estimate computed using various aggregation algorithms remains close to the VSI computed using raw PMU streams and in general all the aggregation algorithms perform well.

Figure 5 compares the different aggregation algorithms and plots the error in estimated VSI's averaged across *all* 300 buses in the grid. Overall, all aggregation algorithms perform well and the errors vary between 0-4%. One can observe that partial computation method perform best compared to other aggregation functions. This is not surprising given that distributed computations do not introduce any error. The error in partial computations results from loss of time synchronization between measurements. Error for the data dropping algorithm is also low and remains below 1%. On the other hand, error for data prioritization increases linearly with load. Lastly, the error for data clustering is high, but it decreases with overload.

Figure 6 shows the percentage reduction in traffic volume with different in-network aggregation algorithms. The figure shows that in general 40-50% data reduction is possible by using aggregation algorithms designed for voltage monitoring application. In case of data prioritization, the possibility of reduction in traffic gradually decreases at higher loading as most of the substations suffer from voltage problems with increased overload. For data dropping algorithm, reduction in traffic gradually decreases from 50-40% as voltage phasors change significantly with increase in load. Similarly, clustering (with 334 clusters) results in about 44% reduction of traffic while partial computations (with 3 PDC nodes) results in the largest reduction of about 50%.

The above experiments clearly demonstrate that certain cyber-physical applications such as voltage monitoring can safely work with in-network aggregated data. Thus this property can be exploited to reduce traffic volume during network overload and gracefully degrade application performance.

VI. Related Work

The work most closely related to ours is GridStat [9], a middleware framework that provides QoS constrained data communications for wide area power grid operations. GridStat supports condensation functions deployed on status routers that look similar to the aggregation functions in our work. However, there are three subtle differences. Firstly, in GridStat the aggregation functions are specified by the application designer to migrate application logic to the middleware layer by taking advantage of the common event patterns. On the hand, in CPS-Net, in-network aggregations are triggered dynamically based on conditions such as the network load. Secondly, since CPS-Net uses a complete stream computing platform, a rich set of application specific aggregation functions become available. GridStat supports relatively simpler functions im-

plemented by the application developer through the calculator module. Thirdly, while the focus of GridStat condensation functions is on optimal use of network resources, CPS-Net allows graceful degradation of application performance though application-sensitive aggregation functions.

There has also been a considerable amount of work in the field of sensor network aggregation [10], [11], [12], [13]. However the focus of these works is on the efficient in-network computation and reducing communication costs, thereby maximizing network lifetime. On the other hand, we focus on aggregating data to minimize load and gracefully degrade performance of time-critical grid applications.

Rest of this section reviews prior work related to QoS, declarative networks, smart grid networking, and stream computing.

QoS QoS in networking is a well-studied topic. The early evolution of network QoS is surveyed in Aurrecoechea et al. [14] covering the progress from Parekh-Gallagher model of shaping and fair queuing as a model for end-to-end bandwidth and latency guarantees through techniques like leaky bucket, weighted fair queuing, effective capacity for admission control, and IETF IntServ architectures to realize the Parekh-Gallagher model. In the second half of the 90's, the unification of ideas from frame-relay, ATM and IP networks in the form of multi-protocol label switching [15], and differentiated services [16] allowed QoS to take qualitative forms (eg: gold, silver, bronze services) to support business considerations, and the mechanisms to support aggregate flows and per-hop behaviors rather than individual flow-level reservations. These advances paved the way for virtual private networking (VPN) that combined QoS, security features, and routing to establish IP as a unified layer for private networks. The Hose model [17] for VPNs provides a convenient "hose" abstraction for customers with associated capacity guarantees. This capacity can be used for traffic between termination points of customer's VPN (possibly hundreds of locations). The technical challenge that was solved was how to provision capacity at the underlying MPLS layer which tends to be point-to-point in nature.

In summary, QoS in networking has advanced considerably to be able to offer sophisticated private networking services on top of which middleware infrastructures can operate. These architectures have been developed in the context of operators who support multiple customers. A key difference in the private networks for cyber-physical systems is that these tend to be built as separate infrastructures from the Internet, and to service the need of one customer, or a small set of closely related customers. Therefore it is important to consider economies and synergies across the network layer QoS and higher layer requirements to extract maximal efficiencies.

In the context of publish-subscribe systems, IndiQos system [18] was one of the first to study classical QoS parameters such as latency, bandwidth, availability, jitter, and loss ratio. Yang et al. in [6] apply a number of techniques such as proactive and reactive overlay path routing to select paths that satisfy application specified QoS constraints in terms of latency. However, in these systems, the QoS-related parameters are decoupled from the information being exchanged. On the other hand, in CPS-Net, we take advantage of the properties of exchanged information to provide the requested QoS under current network conditions.

Declarative Networks The main focus of work on declarative networks has been on declarative specification of routing and network protocols. However in CPS-Net, our focus is on using a declarative paradigm that helps grid applications specify in-network aggregation functions that allow applications to gracefully degrade their performance during network overload. Huebsch et al [19] describe PIER, a massively distributed Internet scale query processor where the idea is to relax traditional database consistency constraints and use a distributed hash table based routing substrate for scalable distributed query processing. Loo et al. [20], [21] describe their work in the declarative specification and implementation of overlays as well as routing protocols using developed declarative languages such as overlog and NDLog respectively. In [22], Calvert et al. describe the design and implementation of Concast, a programmable network service geared towards multipoint to point channels such as multicast feedback applications. The receiving application could specify computations to be performed on the set of sent datagrams before the resultant packet is actually delivered to the receiver.

Smart Grid Networking Developing networking architectures for smart grids is an active area of research with most architectures designed assuming an IP like substrate. These architectures also typically assume a private network based on Internet standards that are owned by the utilities, with the flexibility to add functionalities that are required to meet the demanding requirements of smartgrid applications in terms of timeliness and reliability. Tomsovic et al. [23] provide a high level overview of the current power grid and the changing landscape that necessitates wide area communication infrastructures for the power grid. The North American Synchrophasor Initiative (NASPI) proposed a distributed Publish-Subscribe Architecture, NASPInet, for supporting PMU applications that is composed of PMUs, Phasor Data Concentrators (PDCs) for aggregating data from multiple PMUs and Phasor Gateways (PGWs) for interfacing enterprise PMU applications with the smart grid network. The authors in [24] study the latency and bandwidth requirements of NASPInet dataflows under a variety of network architectures and security solutions.

Stream Computing Another related body of work is stream computing where the goal is to compute continous queries over data streams [25], [26], [27]. In [28], the authors introduce an adaptive query processing mechanism called eddy which dynamically changes the order of operators in a query plan based on the availability of resources. In [29], the authors present the design of the Borealis Stream computing platform that supports dynamic revision of query results and updates to the running query itself, along with a combined server-sensor optimization engine. Tatbul et al. in [30] present a scheme for probabilistically dropping windows involved in aggregate

queries, when the input rate is larger than the available system resources. Most of these systems focus on optimizing the quality and scalability of the stream computing system itself, whereas we focus on applying the stream computing paradigm to network flows to reduce network overload and satisfy application performance requirements.

VII. Discussion and Future work

This paper has presented CPS-Net, a three-layer architecture for a specific class of wide-area real-time cyber-physical systems. The architecture supports application-sensitive graceful degradation of performance and QoS for many-to-many real-time, wide-area data streams and is applied in the context of synchrophasor networks for smart grid applications.

As the examples given in the paper (e.g. VSI) indicate, there are a variety of application-specific aggregation methods as well as policies for specifying aggregation. Such policies could be a function of transient network overload, or based upon other factors such as price of power, or spatio-temporal and administrative considerations. CPS-Net needs to be extended to robustly incorporate some of these considerations in specific application domains. For instance, one open problem to consider is if different applications (consumers of sensing data) specify aggregation of data in different ways, how would a network-level operator combine these requirements, and handle conflicts, while meeting network-capacity constraints?

Each of the topics: distributed streaming, wide-area real-time publish-subscribe, and QoS in networking have been studied in their respective fields. However, since cyber-physical systems will lead to the deployment of new privately and economically operated information infrastructures, it is crucial to extract synergies between these layers, while carefully managing the increase in complexity that may impact reliability and robustness. We intend to explore these tradeoffs in future work as we build CPS-Net using industry-strength hardware / software platforms.

References

[1] D. E. Bakken, A. Bose, C. H. Hauser, E. O. Schweitzer, D. E. Whitehead, and G. C. Zweigle, "Smart generation and transmission with coherent, real-time data," Washington State University, Tech. Rep., August 2010.

[2] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "Streamit: A language for streaming applications," in *Compiler Construction*, 2002, pp. 179–196.

[3] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "SPADE: the system S declarative stream processing engine," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008, pp. 1123–1134.

[4] A. K. Sinha and D. Hazarika, "Comparative study of voltage stability indices in a power system," *Electrical Power and Energy Systems*, vol. 22, no. 8, pp. 589–596, 2000.

[5] P. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, pp. 114–131.

[6] H. Yang, M. Kim, K. Karenos, F. Ye, and H. Lei, "Message-oriented middleware with QoS awareness," in *Proceedings of International Conference on Service Oriented Computing (ICSOC)*, 2009, pp. 331–345.

[7] [Online]. Available: http://www.ee.washington.edu/research/pstca/pf300/ieee300pti.txt

[8] B. Stott and O. Alsac, "Fast decoupled load flow," *IEEE Trans. Power App. Syst.*, vol. PAS-93, pp. 859–869, 1974.

[9] H. Gjermundrod, D. E. Bakken, and C. H. Hauser, "Integrating an event pattern mechanism in a status dissemination middleware," *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World*, vol. 0, pp. 259–264, 2009.

[10] K. Kalpakis, K. Dasgupta, and P. Namjoshi, "Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks," *Comput. Netw.*, vol. 42, no. 6, pp. 697–716, 2003.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a Tiny AGgregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, p. 131, 2002.

[12] A. Manjhi, S. Nath, and P. B. Gibbons, "Tributaries and deltas: efficient and robust aggregation in sensor network streams," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2005, pp. 287–298.

[13] Z. He, B. S. Lee, and X. S. Wang, "Aggregation in sensor networks with a user-provided quality of service goal," *Inf. Sci.*, vol. 178, no. 9, pp. 2128–2149, 2008.

[14] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A survey of qos architectures," *Multimedia Syst.*, vol. 6, no. 3, pp. 138–151, 1998.

[15] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach, Fourth Edition (The Morgan Kaufmann Series in Networking)*, 4th ed. Morgan Kaufmann, March 2007.

[16] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service," Internet Engineering Task Force. [Online]. Available: http://tools.ietf.org/html/rfc2475

[17] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," in *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1999, pp. 95–108.

[18] F. Araujo and L. Rodrigues, "On QoS-aware publish-subscribe," *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, pp. 511–515, 2002.

[19] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, S. Shenker, and I. Stoica, "Querying the Internet with PIER," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003, p. 332.

[20] B. T. Loo, P. Maniatis, T. Condie, T. Roscoe, and J. M. Hellerstein, "Implementing Declarative Overlays," in *In Proceedings of ACM SOSP*, 2005.

[21] B. Loo, J. Hellerstein, I. Stoica, and R, "Declarative routing: Extensible routing with declarative queries," in *Proceedings of the ACM SIGCOMM*, 2005.

[22] K. Calvert, J. Griffioen, B. Mullins, and A. Sehgal, "Concast: design and implementation of an active network service," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, pp. 426–437, Mar. 2001.

[23] K. Tomsovic, D. Bakken, V. Venkatasubramanian, and a. Bose, "Designing the Next Generation of Real-Time Control, Communication, and Computations for Large Power Systems," *Proceedings of the IEEE*, vol. 93, no. 5, pp. 965–979, May 2005.

[24] R. Hasan, R. Bobba, and H. Khurana, "Analyzing NASPInet data flows," *2009 IEEE/PES Power Systems Conference and Exposition*, pp. 1–6, Mar. 2009.

[25] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *In PODS*, 2002, pp. 1–16.

[26] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "Telegraphcq: Continuous dataflow processing for an uncertan world," 2003.

[27] S. Chandrasekaran and M. J. Franklin, "Streaming queries over streaming data," 2002.

[28] R. Avnur and J. M. Hellerstein, "Eddies: Continuously adaptive query processing," in *In SIGMOD*, 2000, pp. 261–272.

[29] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *In CIDR*, 2005, pp. 277–289.

[30] N. Tatbul and S. Zdonik, "Window-aware load shedding for aggregation queries over data streams," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 799–810.