RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems

Woochul Kang, Krasimira Kapitanova, *Student Member, IEEE*, and Sang Hyuk Son, *Senior Member, IEEE*

***Abstract*—One of the primary requirements in many cyber-physical systems (CPS) is that the sensor data derived from the physical world should be disseminated in a timely and reliable manner to all interested collaborative entities. However, providing reliable and timely data dissemination services is especially challenging for CPS since they often operate in highly unpredictable environments. Existing network middleware has limitations in providing such services. In this paper, we present a novel publish/subscribe-based middleware architecture called Real-time Data Distribution Service (RDDS). In particular, we focus on two mechanisms of RDDS that enable timely and reliable sensor data dissemination under highly unpredictable CPS environments. First, we discuss the semantics-aware communication mechanism of RDDS that not only reduces the computation and communication overhead, but also enables the subscribers to access data in a timely and reliable manner when the network is slow or unstable. Further, we extend the semantics-aware communication mechanism to achieve robustness against unpredictable workloads by integrating a control-theoretic feedback controller at the publishers and a queueing-theoretic predictor at the subscribers. This integrated control loop provides Quality-of-Service (QoS) guarantees by dynamically adjusting the accuracy of the sensor models. We demonstrate the viability of the proposed approach by implementing a prototype of RDDS. The evaluation results show that, compared to baseline approaches, RDDS achieves highly efficient and reliable sensor data dissemination as well as robustness against unpredictable workloads.**

***Index Terms*—Cyber-physical systems (CPS), data distribution, feedback control, publish/subscribe, real-time systems.**

I. Introduction

MANY cyber-physical systems (CPS) [1] are sensor-rich distributed real-time embedded systems that closely interact with the physical world. In such systems, a large number of entities cooperate with each other to achieve their common goals. They collect data from the physical world using sensors and feed the sensor data into computing resources, which in turn make real-time decisions in cooperation by sharing data and information among participating entities. For instance, consider a team of firefighters involved in a search-and-rescue task during a building fire. PDAs carried by the firefighters collect data from nearby sensors to monitor the dynamic status of the building. Each individual firefighter's PDA has only limited information from nearby sensors. Therefore, in order to create a more global picture of the situation, all PDAs have to collaborate by sharing their locally collected real-time data [2], [3]. The building-wide situation assessment requires the fusion of data from all (or most) firefighters. Other examples of such CPS, requiring collaboration among a large number of participating entities, include future combat systems [4], vehicular networks [5], unmanned vehicle groups, and traffic control. For these applications, the timely, scalable, and reliable dissemination of sensor data to other collaborating entities is essential.

The primary difficulty of such systems, however, lies in the highly dynamic nature of the systems, both in computing resources and the physical processes. For instance, the availability of participating entities can change dramatically during runtime because of various reasons including temporary failures, noises in communication, mobility, etc. Further, the network layers exploited by such systems are usually unstable. For example, most of the aforementioned applications cannot afford to have fixed reliable networks. Current network middleware, however, cannot handle the highly dynamic nature of CPS.

In this paper, we present a novel publish/subscribe middleware architecture, called Real-time Data Distribution Service (RDDS). In particular, since we are more interested in providing timely and reliable data dissemination service under environments where workloads are bursty and communication is unstable, we focus on two aspects of RDDS to achieve such timely and reliable dissemination service. Our approach to handling bursty workload and unstable communication can be applied to network middleware in general. However, the effect is particularly pronounced in topic-based publish/subscribe (TPS) systems since TPS has been extensively used for decentralized applications that run over large-scale and mobile networks [6].

One of the core mechanisms of RDDS is semantics-aware communication using lightweight predictive sensor models. Since most physical processes in the real world have continuity, e.g., the change of ambient temperature, RDDS models data streams using computationally lightweight physical models. Both a publisher and its corresponding subscribers maintain the same model for each sensor data stream. A new sensor observation is transmitted from the publisher to the subscribers, and the respective sensor models at both sides are synchronized only when the prediction accuracy of the models becomes lower than the required bound. This model-based approach also provides the timeliness and reliability of sensor data since

Manuscript received July 18, 2011; revised October 27, 2011; accepted December 25, 2011. Paper no. TII-11-324. Date of publication January 11, 2012; date of current version April 11, 2012. Paper no. TII-11-324.

W. Kang is with Electronics and Telecommunications Research Institute, Daejon, 305-700, Korea (e-mail: wchkang@etri.re.kr).

K. Kapitanova and S. H. Son are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22903 USA (e-mail: krasi@cs.virginia.edu; son@cs.virginia.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2012.2183878

the subscribers can locally predict the current as well as future states of physical processes using the corresponding models without actual communication with the publisher. This ability can provide significant benefits for systems where real-time feedback is necessary but the network is slow or unstable.

Further, we extend the semantics-aware communication mechanism to the problem of guaranteeing Quality-of-Service (QoS) in TPS by integrating proactive and reactive adaptation of the quality of sensor data. The reactive feedback mechanism at the publishers and the proactive feed-forward mechanisms at the subscribers are integrated in order to enhance the quality of real-time data distribution. At the publishers, a desired utilization bound is achieved by adapting the model accuracy using feedback controllers. On the other hand, to properly set the data rates from the publishers, the incoming workload is predicted in a proactive manner at the subscribers. With this integrated control loops, we can provide the robustness against unpredictable workloads both at the publisher and its subscribers.

To show the viability of the proposed approach, we have implemented RDDS by extending OMG (Object Management Group)'s Data Distribution Service (DDS) [7] to include the mechanisms that handle unstable environments. Using this prototype, we evaluate RDSS on a testbed with realistic workloads. Our evaluation results demonstrate that RDDS takes advantage of the semantics of the sensor data to provide efficient and highly robust data dissemination. To the authors' knowledge, this paper presents the first attempt to provide QoS guarantees in TPS by exploiting the semantics of sensor data and the integrated double control loops. The remainder of this paper is organized as follows. Section II gives an overview of RDDS. Section III presents the details of the semantics-aware communication. Section IV describes our experimental results. The related work is discussed in Section V. Finally, we present conclusions and future work in Section VI.

II. Overview of RDDS

In this section, we briefly overview some of the mechanisms behind RDSS, such as the service model, the QoS negotiation patterns between publishers and subscribers, and how data topics are defined.

A. Service Model of RDDS

RDDS envisions a large-scale CPS, in which multiple publishers, $P = \{p_1, p_2, \ldots, p_n\}$, collect real-time data from physical as well as logical sensors, $S = \{s_1, s_2, \ldots, s_r\}$, and publish them to multiple subscribers, $C = \{c_1, c_2, \ldots, c_m\}$. A stream of sensor data from a sensor s_k is labeled as $stream_k$. Each publisher p collects data from a set of underlying sensors and publishes the corresponding streams of sensor data to a subset of C. A subscriber c consumes the streams from the publishers and may run real-time tasks to analyze the situation and provide timely feedback to control the physical processes. The distinction between publishers and subscribers is logical and one device, in practice, can play both roles. Fig. 1 shows an example that contains five entities, i.e., firefighters. Each firefighter is both a publisher and a subscriber to the sensor streams. Through topic T_{fire}, each entity publishes data streams from local sensors and also subscribes to data streams from remote sensors of other participating entities. Subscribing to T_{fire} gives a global view of the situation captured by the sensors in the system.

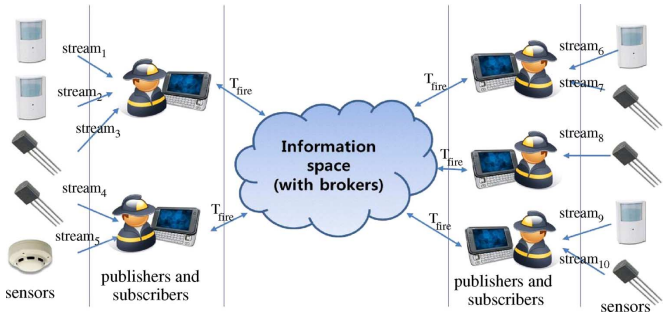


Fig. 1. Information space with RDDS.

While the delivery of sensor data from publishers to subscribers occurs directly between the two parties, the discovery of entities (publishers, subscribers, and data topics) and Quality-of-Service (QoS)/Quality-of-Data (QoD) negotiation occurs in a centralized manner using a broker. RDDS uses a centralized discovery mechanism because of its simplicity and performance. A distributed discovery mechanism could be more robust but causes much higher communication overheads due to the broadcasting or multicasting of discovery messages. To avoid being the single-point of failure, a broker of RDDS can be replicated.[1] The existence of sensor streams is advertised by publishers to subscribers via a broker or a group of brokers. Subscribers interested in consuming data from a group of sensors S_i, where S_i is a subset of S, join the subscription group by subscribing to the corresponding topic T_i. By subscribing to T_i, a subscriber can receive data streams from the sensors in S_i. Publishers need not know who consumes or processes the data streams, and subscribers need not know who produces them.

In RDDS, QoD is defined in terms of the precision bound of sensor data. The maximum tolerable precision bound of sensor data can be specified by the users. There exists a tradeoff between QoD and the freshness of data; to maintain higher freshness of the sensor data, a smaller precision bound is required at the cost of increased workloads. Hence, it is necessary to prevent the overload and subsequent message delays at each node while satisfying the given QoD goals.[2] To this end, the primary QoS metric in this paper is the CPU utilization bound at each node. Both parties, publishers and subscribers, can set the level of QoS and QoD, which they provide/require. Publishers and subscribers can negotiate a level of QoS and QoD that satisfies both parties, and that they agree to adhere to. A broker registers available entities and coordinates with both publishers and subscribers to reach an agreement on the QoS/QoD levels. The QoS/QoD negotiation follows a request/offer model in which the requested QoS/QoD has to be the same or weaker than the one being offered. A broker also maintains meta information, such as the liveness of participating entities, by periodically checking the heartbeat signals from the entities.

Fig. 2 shows an example of handshaking procedure among the entities of RDDS. It should be noted that the registration

[1]We plan to extend RDDS to support a distributed discovery mechanism in the future.

[2]The formal definitions of QoS and QoD are introduced in Section III-C.

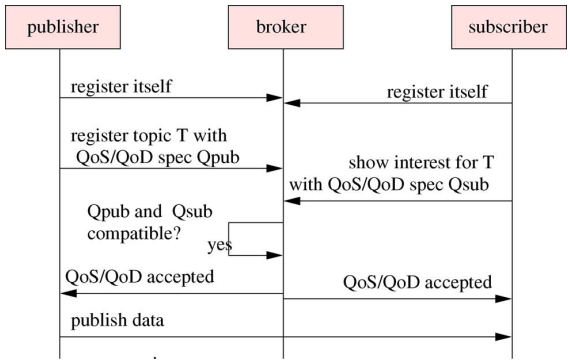


Fig. 2. An example of handshaking procedure.

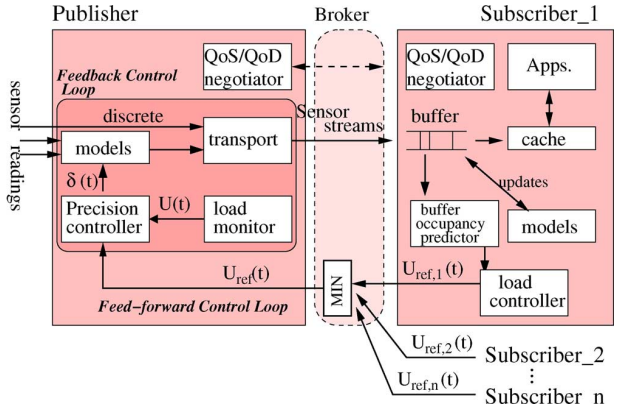


Fig. 3. RDDS architecture.

of entities of RDDS, e.g., publishers, subscribers, and topics, involves a broker, but actual delivery of sensor data occurs between the publisher and its subscribers.

B. Continuous Versus Discrete Data Topics

Data in RDDS is identified by *topic*, which allows publishers and subscribers to refer to data unambiguously. A topic associates a unique *name* and *data type* with the data itself. The specified data type is commonly understandable to both publishers and subscribers. Program 1 shows an example that defines two topic types, *TempSensorType* and *MotionSensorType*. Each topic has a *key* field, which is used to identify a specific sensor stream among instances of the topic. For example, in Program 1, the data stream from a specific temperature sensor s_i, where $\{s_i | s_i \in S\}$, can be identified by $sensorid$, which is a key for the *TempSensorType* topic.

Program 1: An example of topic type definition.

```
Topics BuildingSensors{
#pragma DATA_KEY "TempSensorType:: sensorid"
#pragma DATA_CONTINUOUS "TempSensorType"

struct TempSensorType{
        string sensorid;
        double temperature;
};
#pragma DATA_KEY "MotionSensorType:: sensorid"
#pragma DATA_DISCRETE "MotionSensorType"
struct MotionSensorType{
        string sensorid;
        Boolean is Present;
};
};
```

Each topic type belongs to either *DATA_CONTINUOUS* or *DATA_DISCRETE* categories. A topic that belongs to the *DATA_CONTINUOUS* category has streams of data from sensors that monitor continuously changing physical phenomena. For example, ambient temperatures have continuity in both long and short time scale. On the other hand, *DATA_DISCRETE*, which is the default for data types, represents data streams from sensors that have discrete values. For example, the presence of objects in a room, which is measured by motion sensors, has discrete values and is difficult to formulate in physical models. Hereafter, sensors having continuous and discrete properties are referred to as *continuous sensors* and *discrete sensors*, respectively. In this paper, we focus on the efficient dissemination of continuous sensor data.

III. Real-Time Data Distribution Service (RDDS)

This section discusses the architecture of RDDS, the semantics-aware sensor data dissemination, and the adaptive control of sensor data precision to guarantee the desired QoS.

A. System Architecture

Fig. 3 shows the architecture of RDDS. RDDS has an asymmetric structure for publishers and subscribers. For publishers, RDDS consists of a QoS/QoD negotiator, a precision controller, models for sensors, a load monitor, and pluggable transports. Each data stream from continuous sensors has corresponding models both at the publisher and the subscribers. Updates from continuous sensors go through their corresponding models while data from discrete sensors bypass the models. The models for continuous sensors are used to determine if the incoming sensor data should be delivered to the subscribers. New updates are disseminated to subscribers only if the observed sensor values deviate from the value predicted by the model by more than a specified precision bound δ. The precision bound δ is set dynamically to meet the desired QoS – the CPU load U_{ref}. The load monitor periodically reports the current CPU load to the precision controller, which in turn computes the CPU utilization error, i.e., the difference between the desired CPU load and the measured CPU load at every sampling instant. Based on the error, the precision controller determines if and how the precision bounds δ should be updated for the next sampling period.

For subscribers, RDDS consists of a QoS negotiator, a buffer, sensor models, a buffer occupancy predictor, and a load controller. First, the incoming sensor streams from the publishers are buffered. For continuous sensors, each sensor stream has a corresponding model, which is the same model as the one at the publisher. The models are updated only if new sensor observations arrive from a publisher. This is when the synchronization between the models at the publisher and the subscribers occurs. When there are no incoming updates from the publisher, the models at the subscribers periodically predict the current state

of their corresponding sensors and populate the buffer with the predicted data. This is how the subscribers maintain up-to-date sensor values even without communication with the publishers. To set the proper incoming data rates from the publishers, the buffer occupancy predictor estimates the expected buffer size. The load controller periodically sends a *load control signal* to the broker. The broker aggregates the load control signals from the subscribers, and takes the minimum, which has been generated by the most overloaded subscriber. The publisher uses this minimum value to adjust its target CPU load. Essentially, the feedback control loop at the publisher and the feed-forward loop at the subscribers form a double-loop to enhance the quality of data dissemination.

B. Semantics-Aware Communication in RDDS

Unlike caching-based approaches [8], in which communication and computation loads are reduced by exploiting recently cached values, RDDS exploits the fact that most physical processes have continuity in the real world. By taking advantage of the semantics of the sensor data, our approach not only reduces the computation/communication loads, but also provides reliable and timely data dissemination. As mentioned in Section II, the tag for a continuous sensor topic is DATA _CONTINUOUS. This tag is used as a hint that the continuous sensors need to be described using physical models in unstable and dynamic environments.

Both a publisher and its subscriber(s) have the same model m_i of a sensor s_i, which is the common interest of both parties. Algorithms 1 and 2 show the basic framework for model synchronization at a publisher and subscribers, respectively. Algorithm 1 shows what occurs at a publisher when a new observation arrives. Wen a publisher p_j receives a sensor observation v from sensor s_i, it looks up the model m_i and makes a prediction using this model. If the gap between the predicted value $\hat{v}$ from the model and the sensor observation v is less than the precision bound δ, the new sensor observation is discarded (or saved locally for logging.) This implies that the current models (both at the publisher and its subscribers) are sufficiently precise to predict the sensor observation within the given precision bound. However, if the gap is greater than the precision bound (line 2), the model is no longer able to capture the current behavior of the sensor output. In this case, m_i at the primary node is updated and v is multicast to all subscribers (line 3).

Algorithm 1: Updating a sensor model at a publisher.

Input: sensor observation v from sensor s_i
1. $\hat{v} =$ prediction from model m_i of s_i;
2. **if** $|\hat{v} - v| \geq \delta$ **then**
3. multicast v to subscriber(s);
4. update model m_i for s_i;
5. **else**
6. discard v;
7. **end**

Algorithm 2 is a reaction to the multicast from the publisher. Upon the reception of a new sensor observation v, a subscriber updates its own model m_i with v. A benefit of this approach is that an application can dynamically obtain the current value of a sensor s_i from its model m_i by using predictions instead of actually communicating with the publisher. This makes RDDS highly resilient to potential loss of data packets in unstable communication networks. Further, since the sensor readings can be estimated readily without communication delays, timeliness in data accesses can be achieved. As shown by Algorithms 1 and 2, communication between a publisher and its subscribers occurs only when the sensor models are not sufficiently precise.

Algorithm 2: Updating a sensor model at a subscriber.

Input: update v from publisher p_i
1. update model m_i of s_i;
2. store to cache for later immediate data accesses;

It should be noted that the estimations from models at a publisher and its subscribers can be different when the network is unstable. For example, when sensor readings are missed during the communication from the publisher to its subscribers, the models at both parties could have different states; i.e., the models at the publisher could be more up-to-date. The error caused by such model discrepancy should not be significant when a lot of sensor readings are being delivered to the subscribers. However, for applications in which this error is critical, RDDS provides pluggable transport capability. With the pluggable transport capability, users can choose the network protocols to be used. For example, depending on whether the reliable delivery of data is critical, either TCP or UDP can be used. However, due to its communication overhead and latency, the TCP protocol might not be very suitable for real-time applications and therefore offsetting the benefit of using models. The impact of communication stability and the choice of communication protocol is discussed in Section IV.

1) State Estimation for Continuous Sensors: Since RDDS could potentially need to handle a large number of sensor streams, relying on complex models is computationally prohibitive. To this end, each sensor in RDDS is modeled with a simple state space model, where the state of each sensor x_k evolves according to the following equation:

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1}\mathbf{x}_k \tag{1}$$

where $\mathbf{F}_{k+1}$ is the state transition matrix relating $\mathbf{x}_{k-1}$ to $\mathbf{x}_k$. The state of each sensor having continuity can be described with differential equations. In RDDS, a physical process, such as temperature, measured by a sensor at time k is represented in a state vector with two state variables, $[x \quad dx/dt]^T$, where x is the current sensor value at time k and dx/dt is the derivative of x with respect to time t. The state transition matrix $\mathbf{F}_k$ can be defined as $\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$. We may need an additional state variable, such as d^2x/dt^2, to describe sensor dynamics more accurately. However, the cost of model maintenance increases proportionally to the number of state variables, which is why we chose to use only two state variables for the current implementation of RDDS. We believe that the dynamics of most simple physical processes, such as temperature, light intensity, and pressure, can be described using these two state variables.

However, sensor data is often imprecise due to measurement noises, unstable communication, and model inaccuracies.

Therefore, the state transitions do not exactly follow (1). To reduce the uncertainty in the system, RDDS exploits the Kalman filtering technique [9]. Kalman filters enable us to precisely estimate and predict the current state $\mathbf{x}_k$ and future state $\mathbf{x}_{k+1}$ from noisy sensor observations. Further, Kalman filters do not need large historical data for modeling. The parameters of a Kalman filter can be estimated at runtime and their accuracy is gradually improved when there are more sensor observations. For more details on Kalman filters, readers are referred to [9].

2) Maintaining Sensor Data Freshness: As explained in the previous section, there is no communication from a publisher to its subscribers unless a model is no longer accurate. However, at the subscribers, the freshness of the last received sensor data deteriorates over time. The period, in which the sensor value is valid, is called *absolute validity interval (avi)* [10]. To maintain fresh and precise sensor observations, even when there are no updates from the publishers, the sensor values need to be updated periodically before *avi* at the subscribers elapses. Since each subscriber maintains the sensor models, it can estimate the current and future sensor states. Periodically, the locally estimated state of each sensor is fed into the buffer. The *avi* of a sensor value can be derived from the precision bound, which is $\pm\delta$. Since the sensor value changes with speed dx/dt, the *avi* is $2 \times \delta/dx/dt$. Intuitively, when a sensor value changes rapidly, the data object should be updated more frequently to maintain the validity of the data. According to [10], to maintain data freshness, the update period should be as short as half of the *avi*.

Algorithm 3 shows how a sensor value is updated at a subscriber without communication to its publisher.

Algorithm 3: Periodic updates of a sensor value at a subscriber.

Input: δ_i, precision bound for sensor s_i

Input: dx/dt, the 2nd state variable of s_i

Input: t_{cur}, current time

1. $\hat{v} =$ prediction from model m_i of s_i;
2. update s_i's value to $\hat{v}$
3. set the next sensor value update time to $(t_{cur} + \delta_i/dx/dt)$

3) The Impact of Model Inaccuracy: In RDDS, the communication load is increased only when the models are not sufficiently accurate. In this section, we introduce intentional errors to show the impact of model inaccuracy.

We model a physical process that has two components in RDDS, $\mathbf{x} = [\,x \quad dx/dt\,]^T$, as well as an additional non-negligible third component, $d^2x/dt^2 = \alpha$. When the second component dx/dt is equal to c, the expected change of x after time t in RDDS is

$$\Delta\hat{x} = t \times \frac{dx}{dt} = ct. \tag{2}$$

However, the true change of x is

$$\Delta x = \int_0^t \frac{dx}{dt} dt = \int_0^t (\alpha t + c)\, dt. \tag{3}$$

In RDDS, updates and communication occur only when $|\Delta\hat{x} - \Delta x| \geq \delta$

$$|\Delta\hat{x} - \Delta x| = \left| ct - \left(\frac{\alpha t^2}{2} + ct \right) \right| \geq \delta. \tag{4}$$

Hence, the expected update rate r is

$$r = \frac{1}{t} \leq \sqrt{\frac{\alpha}{2\delta}} = \sqrt{\frac{1}{2\delta} \frac{d^2x}{dt^2}}. \tag{5}$$

This shows that the increase of the update rate is proportional to the square root of the third term. For instance, if δ is 1 meter in measuring the moving distance of a vehicle, the model inaccuracy incurs 1.56 additional updates per second since the acceleration of a typical starting vehicle is known to be less than $4.9\ \mathrm{m/s^2}$. Further, the effect of the third component is transient in many physical processes; d^2x/dt^2 approaches 0 for a comparatively long period after the starts. Instead of using a more complex model, which incurs constant overhead, we have developed a QoD adaptation mechanism, discussed in the following section, which handles transient and bursty workloads caused by model inaccuracies.

C. QoS/QoD Guarantees via Reactive/Proactive Adaptation

As discussed in the previous sections, the accuracy of the sensor data at the subscribers is determined by the model precision bound δ. However, it is a challenging task to set a proper precision bound δ at design or deployment time, since the maximally achievable data quality changes at runtime as the operating environment changes. For example, the system can be overloaded if δ is too small. Conversely, the accuracy of the sensor data may be too low if δ is too big. Hence, RDDS uses an adaptive control mechanism to dynamically adjust the precision bound δ at runtime.

1) QoS/QoD Specification: In RDDS, QoD and QoS are explicitly specified by the user. We define QoD in terms of the precision bound of the sensor data. Recall that $stream_i$ from sensor s_i has a precision bound δ_i. The actual precision bound of the stream is altered through precision bound scaling. Both at the publishers and the subscribers, the system specification $< Q, U_{\max} >$ consists of a QoD specification Q and a QoS specification $U_{\max}$. The QoD specification Q is given by $Q =< q_1, \ldots, q_n, \gamma_{\max} >$, where $q_1, \ldots, q_n$ denote the nominal precision bounds of the data streams $stream_1, \ldots stream_n$, respectively, and $\gamma_{\max}$ represents the maximum tolerable precision bound scaling factor, i.e., $\gamma \leq \gamma_{\max}$. Accordingly, the actual precision bound of $stream_i$ is given by $\delta_i = q_i\gamma$, which is less than or equal to the maximum precision bound $\delta_{i,\max} = q_i\gamma_{\max}$. The QoS specification $U_{\max} \in [0 \ldots 1]$ represents the CPU utilization bound of the node. $U_{\max}$ prevents the system from overloading while satisfying the target performance. At runtime, the target utilization bound U_{ref} is set such that $U_{ref} \leq U_{\max}$. It should be noted that there is no lower bound on the precision as, in general, users require the precision bound to be as small as possible (if the system is not overloaded.) Each publisher and subscriber set their own QoS goal $U_{\max}$ either at deployment time or runtime. Before subscribers subscribe to topic T from a publisher, they should reach an agreement on Q of T with the publisher.

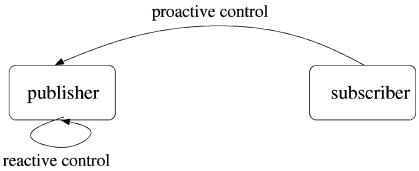


Fig. 4. The double-loop control architecture of RDDS.

TABLE I
SUMMARY OF ADAPTATION MECHANISMS IN RDDS

	Publisher	Subscriber
Control mechanism	Reactive control	Proactive control
Monitoring variable	CPU utilization	Data incoming/processing rates
Control input	δ: precision bound	U_{ref}: target utilization bound at publishers
Sampling interval	5 second	120 seconds

2) Double-Loop Control Strategy in RDDS: To guarantee the desired QoS and QoD, RDDS exploits two control loops, as shown in Fig. 4. A key intuition that affects the architecture of the control loops is that the dynamics of RDDS manifest in two different time scales. Since TPS is sender-initiated, publishers can counteract current violations of the QoS immediately. Adaptation at the publishers can follow in a short time scale. However, global load propagation from publishers to subscribers occurs relatively slowly since workloads are filtered at the publishers at first. Further, the communication latency between publishers and subscribers makes it difficult, if not impossible, to react instantly to workload changes. Due to the communication delays, corrective behavior at the publishers based on current status may be incorrect. To address this problem, the second control loop is proactive and occurs in a longer time scale. The potential buffer overflows at the subscribers are monitored proactively with longer periodic intervals and are reported to the publishers.

The control strategy of RDDS is summarized in Table I, and is discussed in detail in the following sections.

3) Reactive Adaptation at Publishers: A well-established feedback control theory is applied to support the desired utilization bound at the publishers. To evaluate the relationship between the model accuracy δ and the CPU load U, we estimate the CPU load at the k^{th} sampling instant via the CPU loads and the model accuracy at the previous p sampling instants. We express this relationship in a difference equation in the discrete time domain

$$U(t) = \sum_{i=1}^{p} (a_i U(t-i) + b_i \delta(k-i)) \tag{6}$$

where p is the system order. This difference equation models the dynamics of the RDDS publisher. The model coefficients a_i's and b_i's can be derived via *system identification*[11]. After the modeling, we design a controller for the model. The goal of the controller is to ensure that the measured CPU utilization, $U(t)$, is equal to the target utilization, U_{ref}. To support both the average and the transient performance, we apply PI (proportional integral) control, which can support the long-term stability via I control in addition to P control for short-term reactions. At the k^{th} sampling instant, the PI controller computes the control signal $\delta(t)$, which is the model accuracy adjustment required to support U_{ref}

$$\delta(t) = \delta(t-1) + (K_P + K_I)e(t) - K_P e(t-1) \tag{7}$$

where K_P and K_I are controller parameters. The desired properties, such as settling time and overshoot, are determined by choosing the right values for the controller parameters. In general, there is a tradeoff between the stability and settling time of a system. We used the Root Locus technique [12], which is the most common controller design technique, to choose the appropriate controller parameters.

There is also a tradeoff in the choice of the sampling interval. If the sampling interval is too short, the measured output of the system can be highly variable, which can make the controller too sensitive to transient changes of the system. Conversely, if the sampling interval is too long, the speed of control is slow and the dynamics of the system cannot be captured appropriately. We performed an experiment on a testbed to understand the impact of the sampling interval. In the testbed, 32 nodes both publish and subscribe sensor streams and the CPU load at one of the nodes is measured (the details of the testbed and its configuration parameters are discussed in Section IV). Fig. 5 shows the CPU loads when two different sampling intervals, 1 and 10 s, are applied. All parameters are fixed during the observation; hence, there are no external factors to affect the workload of the system. However, Fig. 5(a) demonstrates that the CPU load has high variability when the sampling interval is 1 s. When 1-s sampling period is used, if the control rule of (7) is applied, the controller could react to the stochastic changes in the system output, i.e., the measured CPU load, potentially making the system unstable. In contrast, when the sampling interval is 10 s as in Fig. 5(b), the response of the controller is delayed until the next sampling point. When a 10-s sampling period is used, the average control delay is 5 s.

The desired controller response time is application specific. Our testbed, which is discussed in Section IV-B, can tolerate a delay of a few seconds since the sensor data is gathered on a per-second basis. To this end, we choose to use a 5-s sampling interval, which, we believe, is a good tradeoff since it satisfies the application-specific requirements. For more details on the controller design and tuning, readers are referred to Hellerstein *et al.* [12].

4) Proactive Adaptation at Subscribers: The primary purpose of the feed-forward loop at the subscribers is to set the proper arrival rate of the incoming sensor data streams from the publishers. Since the potentially long network delays between publishers and subscribers could render the reactive adaptation approaches inappropriate, the corrective behavior based on the current status may be incorrect when the control signal is applied at the publishers. An alternative solution for a subscriber would be to use queueing theory to proactively predict potential buffer overflows and adjust the rates of the incoming sensor streams. Queueing theory provides a predictive framework such that the expected buffer occupancy and delays can be inferred from the input loads. When a subscriber sets its target buffer occupancy to b_{occ}, the desired buffer size is the maximum buffer

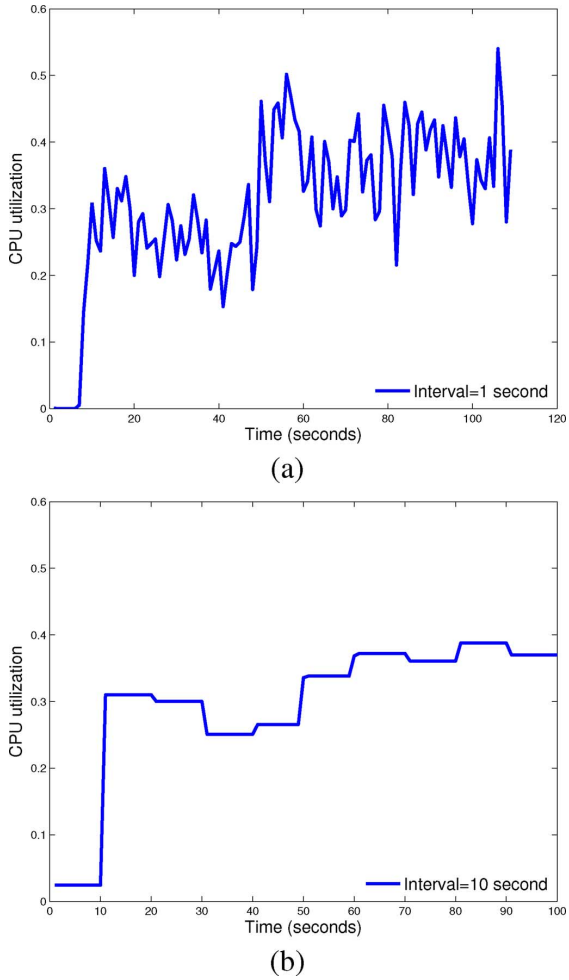


Fig. 5. The measured CPU utilization with different sampling intervals. (a) Sampling interval = 1 s. (b) Sampling interval = 10 s.

size $b_{\max}$ times b_{occ}. Little's law [13] tells us that the average size of buffer L in the system at the sampling instant n is

$$L(n) = \lambda(n)W(n) \tag{8}$$

where λ is the arrival rate of the sensor streams and W is the average data processing delay. This law is independent of the probability distributions involved, and hence requires no assumptions about the distribution of sensor data arrivals and processing. The desired sensor data arrival rate for the next period $\lambda d(n+1)$ can be predicted when the target buffer length $b_{\max} \times b_{occ}$ and the average data processing delay $W(n)$ are known. When the current target utilization bound at a publisher is U_{ref}, the target utilization bound for the next period is adjusted to

$$U_{ref}(n+1) = \frac{\lambda \hat{d}(n+1) U_{ref}(n)}{\lambda(n)}. \tag{9}$$

It should be noted that Little's law assumes a long observation window. Hence, the monitoring period at a subscriber should be long enough to have a probabilistically meaningful amount of sensor data.

The target utilization bound for the next period is generated at each subscriber; for a subscriber k, the target utilization bound $U_{ref,k}(n+1)$ is generated. However, instead of sending this control signal directly to its publisher, which requires the publisher to keep a separate state for each subscriber, each subscriber forwards the control signal to the broker. At the broker, the feed-forward control signals from the subscribers are aggregated and the minimum of them is taken. This minimum is the final feed-forward control signal from the subscribers to the publisher. This conservative approach ensures that the congestion at the most overloaded subscriber k, which has the minimum value of $U_{ref,k}(n+1)$, is reduced.

One problem with the above algorithm is that taking a minimum of the control signals can make the whole system vulnerable to denial-of-service (DoS) attacks. For instance, if one of the nodes is compromised, the node can break the whole system by sending a control signal that is unacceptably low. To address this problem, the broker might set a lower bound on U_{ref} and only accept control signals that are greater than the lower bound. Alternatively, more sophisticated DoS detection [14] and admission-control [15] mechanisms can be considered at the broker to filter out signals from compromised nodes.

IV. Evaluation

A. Performance Evaluation Goals

The objectives of the performance evaluation are to: 1) assess the efficiency of semantics-aware communication in TPS and 2) determine if the integrated proactive/reactive adaptation mechanism can provide guarantees on target CPU loads according to a QoS specification. For the first objective, in Experiment #1, we have studied and evaluated the behavior of the algorithms under various conditions, where a set of parameters have been varied. The second objective is investigated in Experiment #2 by comparing the adaptation performance of RDDS while its controller is turned on and off.

B. Emulation Testbed

We have implemented a prototype of RDDS on a testbed. A collaborative search-and-rescue scenario in a building fire from [16] is adapted and emulated on our emulation testbed. In this scenario, a team of firefighters is sent to participate in the search-and-rescue operation. Each firefighter carries a PDA, which collects data from nearby sensor nodes in the building via wireless communication and publishes them to peer nodes. This sharing of real-time sensor data via RDDS gives a global view on the situation for each firefighter, rendering timely reaction to the situation.

Fig. 6 shows the testbed. The testbed employs one Nokia N810 Internet tablet [17] and a PC cluster to emulate the PDAs of 32 firefighters. The N810 device is equipped with 400 MHz TI OMAP processor, 128 MB RAM, 256 MB flash memory, 802.11b Wi-Fi radio, and runs *Maemo*, which is a modified version of GNU/Linux slimmed down for mobile devices.[3] Since the number of available PDAs is limited, a PC cluster with up to 32 computing nodes is used to enable a large-scale evaluation.

[3]Maemo is based on GNU/Linux 2.6.21 kernel and compliant with POSIX standards.

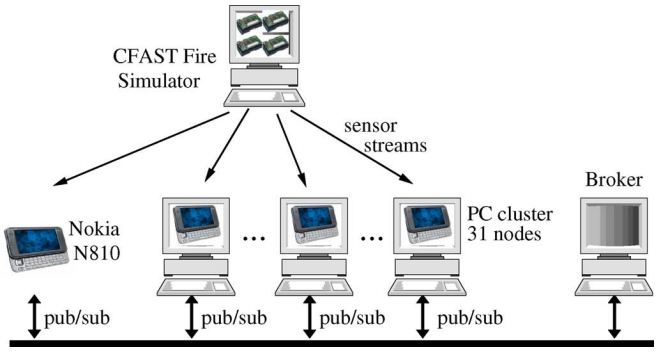


Fig. 6. RDDS testbed.

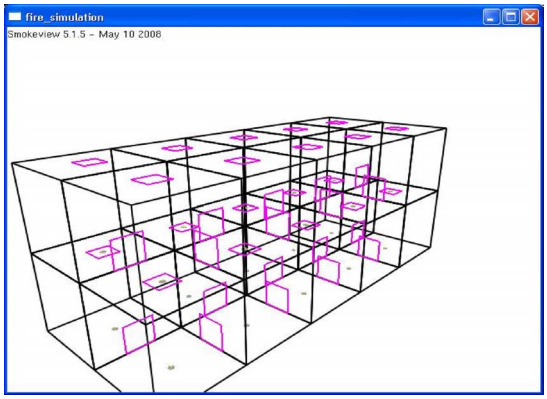


Fig. 7. The modeled building with CFAST and SmokeView.

Each PC node has a dual-core 1.5 GHz processor, 1 Gbyte of memory, and runs on Linux 2.6.31. One of the cluster nodes is dedicated for the brokering service. The remaining 31 cluster nodes are used to emulate firefighters' PDAs. Another PC with a 2.67 GHz quad core is used to generate sensor streams. The N810 device and the PC cluster are connected via 802.11b Wi-Fi. For the transport layer of communication, UDP transport protocol is used in default. All emulated PDAs, either on N810 or on the PC cluster, perform the same functionality. However, the real measurements of performance, e.g., CPU utilization, are performed in the N810 device. The N810 device, we believe, represents emerging mobile computing platforms, which are expected to interact with ubiquitous sensors in CPS.

Sensor streams are simulated using CFAST (The Consolidated Model of Fire and Smoke Transport) fire simulator [18], [19] from the National Institute of Standards and Technology (NIST). Using the CFAST simulator, a wide-range of fire scenarios can be simulated in detail by configuring the input parameters, which include the geometry of the compartments, the initial fire source and burning objects in the compartments, flow vents, and wall materials. Traces are generated from the simulator offline for repeatability and scalability of the experiments. Each trace corresponds to the history of temperature change at a specific location in the modeled building. Fig. 7 shows the model of the building generated with CFAST and the accompanying tool SmokeView [20]. In the runtime of each simulation, these traces are replayed and sent to participating nodes with a 1-s interval.[4]

A summary of the testbed is shown in Table II.

TABLE II
TESTBED SETTINGS

Parameters	Values
Number of firefighters, $\|C\|$	1 - 32
Networking	802.11b Wi-Fi
Transport protocol	UDP and TCP (UDP in default)
Type of sensors	temperature in a building fire
# of sensors per participant	500-1000
Sensor observation interval	1 second

TABLE III
BASELINE APPROACHES

OpenDDS	State-of-the-art TPS implementation
Approx-Caching	Data distribution using recently cached sensor values
RDDS	RDDS with model-based predictive data distribution

C. Baselines

We compare our RDDS scheme with the baseline schemes shown in Table III.

OpenDDS [22] is a state-of-the-art implementation of the OMG DDS specification. It disseminates data from a publisher to subscribers without exploiting the semantics of the sensor data. Approx-Caching is a value-driven approach, in which a publisher multicasts sensor observations only if the difference between the current value and the last multicast is greater than a threshold δ.

D. Experiment #1: Performance of Semantics-Aware Communication in TPS

1) Scalability: First, we show the performance of semantics-aware sensor data dissemination in RDDS. To this end, the workload changes are monitored while the number of participating firefighters is changed from 1 to 32. During the experiment, the controllers at publishers/subscribers are turned off; in other words, the precision bound δ is fixed and does not change over time. For both RDDS and Approx-Caching, the precision bounds δ of all sensor streams are set to 0.5 °C. According to the discussion in Section III-B2, when avi is 1 s, which is a rough requirement from [21], and $dx/dt = \pm 1$ °C/s, δ needs to be 0.5 °C to maintain the freshness of sensor data.

All evaluation results are based on at least 10 runs, and an average of 95% confidence intervals are taken.

Figs. 8 and 9 show the scalability of RDDS and the baselines when we change the number of firefighters from 1 to 32. Each firefighter receives data streams from 500 nearby temperature sensors. As the number of firefighters increases, the total number of sensor streams covered increases accordingly. Fig. 8 shows the total number of messages sent and received at each participant as the number of firefighters increases. However, it should be noted that the slope of the line is much flatter in RDDS than in the baseline approaches since RDDS filters out most of the incoming data from sensors as long as its sensor models can predict the values within the precision bound δ. For instance,

[4]Real-time queries for search-and-rescue tasks can be invoked on a per-second basis [21].

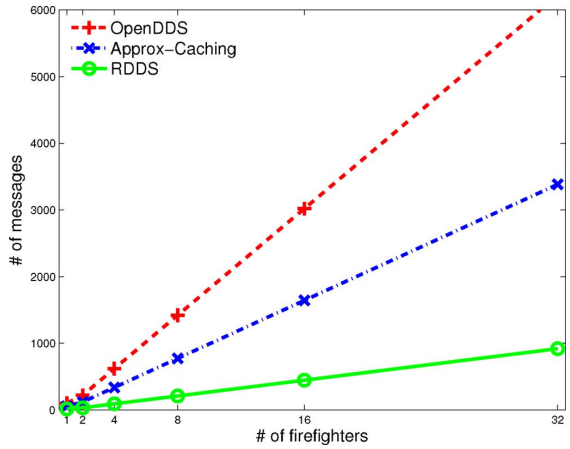


Fig. 8. Number of messages ($\delta = 0.5$).

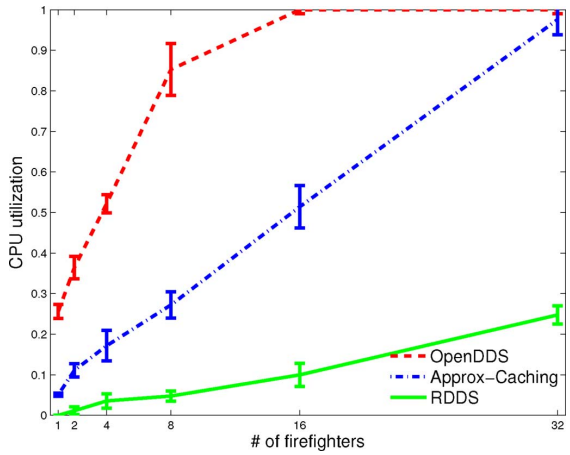


Fig. 9. CPU load ($\delta = 0.5$).

RDDS filters out 84% of the original data when 16 firefighters are deployed while Approx-Caching filters out only 44%. This high filtering performance of RDDS implies that it can be extremely scalable in low-bandwidth networking environments.

The amount of communication is highly related to the CPU load since each message incurs processing overhead. Fig. 9 shows the CPU load in the same experiment. The CPU load increases proportionally to the amount of communication in all approaches. Maintaining a proper level of the CPU load is particularly important for CPS applications that need to guarantee the timely dissemination of critical sensor data. We can see in Fig. 9 that OpenDDS and Approx-Caching are becoming overloaded when the numbers of participants are 16 and 32, respectively. In contrast, the CPU load of RDDS remains under 0.2 even when 32 participants are deployed.

Fig. 10 shows the breakdown of the CPU loads in the same experiment. In the graph, three major tasks contribute to the overall measured CPU load; *cx3110x task* is the cx3110x WI-FI driver for interrupt handling, *OMAP McSPI/O task* is the DMA transfer driver, and *RDDS task* is the RDDS task itself. The combined CPU load of *cx3110x task* and *OMAP McSPI/O task* represents the CPU overhead to process the incoming/outgoing data. As Fig. 10 shows, data communication

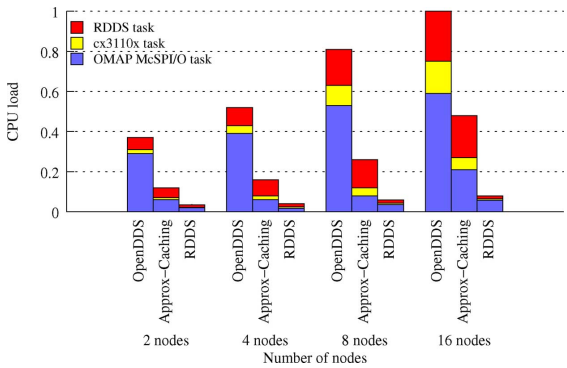
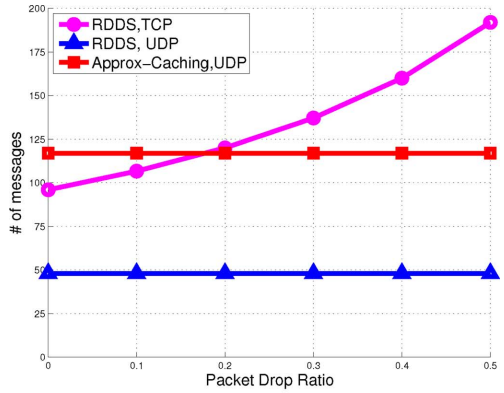


Fig. 10. The breakdown of CPU load.

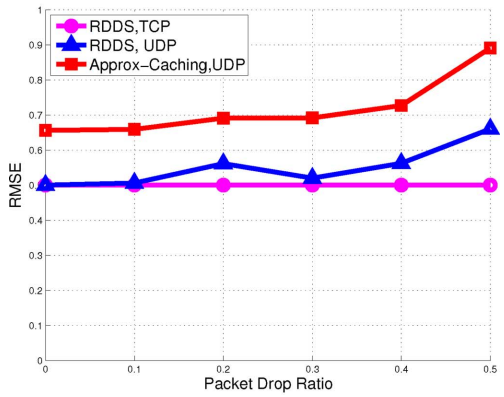
is the primary source of CPU time for all approaches, and its portions increase as the size of collaboration groups increases. Given the high computation CPU overhead for communication, RDDS gets most benefit by reducing the communication needs. For example, when the number of firefighters doubles from 8 to 16, RDDS's CPU load incurred by data communication increases by 0.03 while Approximate-Caching incurs 0.17 CPU load increase. This result shows that RDDS's semantics-aware communication using models is especially effective in reducing the data communication overhead.

2) Impact of Lossy Communication: This section discusses the impact of lossy communication and the choice of transport protocol. In our evaluation, packets are dropped with random probability and we measure the total number of exchanged messages and the quality of the data at one of the nodes. Two transport protocols, UDP and TCP, are used to test the impact of the communication protocol. The quality of the data is quantified by the *root mean square error* (RMSE) : $\mathrm{RMSE} = \sqrt{(\sum_{k=1}^{n}(x_k - \hat{x}_k)^2)/n}$, where x is the ground trough, $\hat{x}$ is the estimated value from the models, and n is the duration of the simulation.

Fig. 11 shows the results when the packet drop ratio is changed from 0 to 0.5. A packet drop ratio of 0.5 indicates that half of the messages are lost randomly during communication. We can see from Fig. 11(a) that, when UDP is used, the total number of exchanged messages is not affected by the lossy communication since no retransmission is done for the lost messages. In contrast, the communication load for the TCP-based approach increases proportionally to the packet drop ratio. For instance, when the packet drop ratio increases from 0.2 to 0.3, around 14.5% more messages are exchanged to guarantee reliable message delivery. However, as shown in Fig. 11(b), the increased communication load does not significantly improve the quality of the data or the accuracy of the models. For example, when the packet drop ratio is changed from 0.4 to 0.5, the RMSE for the TCP-based approach increases by less than 0.1. This increase is considered insignificant since the precision bound δ in this evaluation is 1. Fig. 11(b) also shows that the quality of the model does not increase monotonically as more sensor data is fed to the model. This is due to the fact that some data can degrade the performance of the models. For instance, the model's estimation quality improves and the

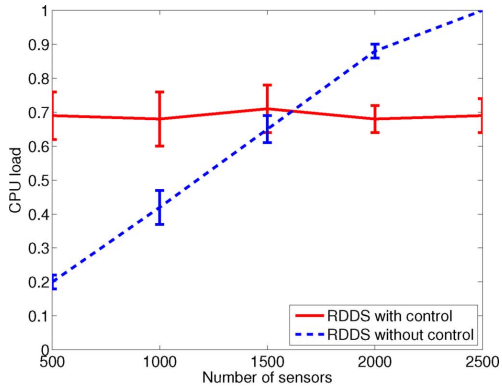


(a)

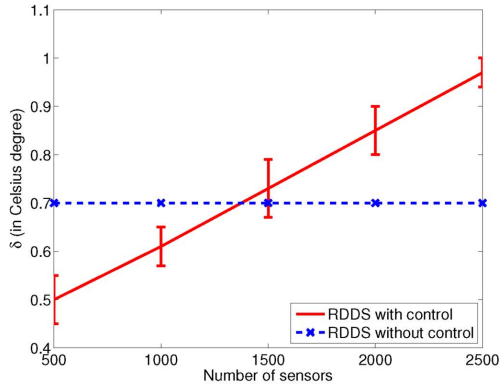


(b)

Fig. 11. The impact of unreliable communication and transport protocols ($\delta = 1$). (a) Communication load. (b) RMSE.



(a)



(b)

Fig. 12. Average performance. (a) CPU load. (b) Precision bound (δ).

RMSE decreases from 0.57 to 0.52 when the packet drop ratio increases from 0.2 to 0.3.

The results in Fig. 11 show that RDDS's semantics-aware approach is highly resilient to lossy communication. Further, reliable transport protocols should be used only for messages that require 100% reliable delivery since they can result in high communication overhead without a meaningful gain in the quality of the models or the data.

E. Experiment #2: Adaptability to Unpredictable Workloads

We evaluate the adaptability of RDDS against unpredictable workloads. For the evaluation, 32 participants are deployed and the QoS controller at each publisher/subscriber is turned on. The QoD is given by the maximum precision bound $\delta_{\max}$, which is 1°C, and the QoS is given by the maximum CPU utilization bound $U_{\max}$, which is 0.7. We compare RDDS when the QoS controller is turned on and off.

1) Average Performance: The adaptability of RDDS is evaluated by changing the workload. The number of sensor streams per firefighter is increased from 500 to 2500. For RDDS without controllers, the precision bound is set to 0.7 °C. Fig. 12 show the average performance. Fig. 12(a) demonstrates that RDDS with a controller achieves CPU load which is very close to the target one under all workloads. In contrast, the CPU load fluctuates significantly between under-utilization and overload when no control is applied and the load changes between 0.2 and 1.0. Violating the CPU load goal implies that the dissemination of sensor data can be delayed significantly. Fig. 12(b) shows the changes of the precision bound δ during the experiment. In RDDS, δ increases linearly as the workload increases.

2) Integrated Control Loop and Transient Performance: The average performance is not enough to evaluate dynamic systems like RDDS. In addition, transient performance, such as settling time, should be considered as well.

In this experiment, we introduce sudden changes in the workload in order to observe the transient behavior of RDDS. First, the reactive feedback control loop is tested by introducing a sudden surge of the workload at N810 node. Secondly, the proactive feed-forward control loop is tested by intentionally delaying the processing time of each message. For the feedback control test, at the 56th sampling instant, the number of sensor streams per firefighter surges from 500 to 1500 as a step function. For the feed-forward test, at the 80th sampling instant, the data processing rates at the N810 drops by 50% of the original. The processing rate is dropped by intentionally doubling the processing time of the messages.

Fig. 13 shows the results. The CPU load increases immediately at the 56th instant. However, it is stabilized to the target CPU load within five sampling periods. The precision bound δ also increases to approximately 0.68 °C to achieve the target CPU load. Further, at the 100th sampling instant, the load adaptation signals from the subscribers are applied to change the target CPU load U_{ref} from 0.7 to 0.4. The precision bound again adjusts to approximately 0.93 °C to achieve the new target CPU load. This result also shows that the difference of control periods

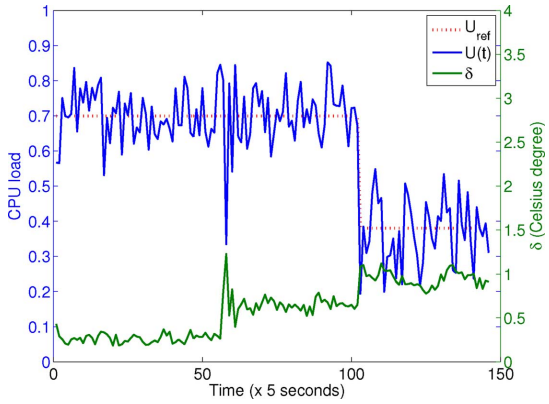


Fig. 13. Transient behavior.

between publishers and subscribers, which are 5 and 120 s, respectively, is long enough to eliminate interference between the two controllers.

V. Related Work

A. Publish/Subscribe Paradigm

The publish/subscribe model has seen a lot of attention and has been applied to a large variety of protocols and applications [6]. In the industrial automotive and building automation domain, *fieldbus* networks, such as WorldFIP [23], CAN [24], LonWorks [25], and BACnet [26] have used publish/subscribe designs for decades [27]. Kaiser *et al.* developed a real-time publish/subscribe model for distributed real-time systems that use a controller area network (CAN) bus [28]. Academically, since the pioneering publication by Oki *et al.* in 1993 [29], publish/subscribe systems have received a lot of attention. A broad spectrum of research topics has been studied, including group communication [30], reliable application-layer multicast [31], mobility [32] and implementation issues [29], to name a few.

However, these systems ignore the problems that arise when their operating environments are unstable and highly unpredictable. In contrast, our work is the first in addressing such issues.

B. Internet-Scale Sensing Services

A number of Internet-scale sensing services were designed to organize and to disseminate wide-area sensor data to a large number of users. The Sensor Andrew network developed at Carnegie Mellon University is developed around the publish/subscribe architecture. However, for scalability issues, Sensor Andrew to only be push-based, and thus more centralized [33]. One of the project employing the query-response paradigm is IrisNet, an architecture developed at Intel Research [34]. However, IrisNet was primarily intended for Internet connected desktop PCs and inexpensive commodity off-the-shelf sensors such as Webcams, rather than for resource-constrained sensor networks. Multiple research groups have worked on collaborative query-response sensing services like SenseWeb [35] from Microsoft Research and SensorWeb [36] from the Kno.e.sis Center. These systems are targeted towards visualizing and sharing data with end-users. Similarly, web applications such as Pachube [37] and Noisetube [38] provide access to numerous sensors and actuators for use in user-generated applications is facilitating a world in which a massive data collection is put to use of individual users as well as society.

Most of these systems assume stable and non-real-time environments where the timeliness and reliability of sensor data dissemination is not a critical issue. Further, these systems do not provide a mechanism to handle unpredictable workload changes. In contrast, RDDS targets a collaborative real-time applications under highly unpredictable environments which are common for CPS.

C. Data Semantics

Understanding the semantics of the data going through the system has been very beneficial to a wide variety of applications. Multimedia multicast uses data semantics to provide best-effort, large-scale, multipoint communication, for applications such as shared whiteboards, multiplayer games, and software distribution [39], [40]. Real-time databases rely on data semantics to improve the performance of user transactions and the concurrency control in these systems [41].

Exploiting models of observed physical phenomena in order to reduce the communication loads has been an active research issue for sensor networking. In BBQ [42], time-varying multivariate Gaussian and Kalman filters are used at the base station to minimize data acquisition costs. PRESTO [43] uses a seasonal ARIMA model to predict the temperature changes with less communication among sensors. However, previous model-based approaches in sensor networking are application-specific, and are not general enough to be used in different contexts. In contrast, the semantics-aware dissemination mechanism in RDDS can be seamlessly integrated into typical data diffusion frameworks including TPS.

D. Quality-of-Service (QoS)

There is a large body of literature on QoS in the networking and Internet environments [44], especially regarding network layer services, such as IntServ [45] and DiffServ [46], to provide end-to-end delay guarantees. Another area of interest is RPC-based middleware, including CORBA [47] and JMS [48] that support QoS levels for their communication services [49].Behnel *et al.* provide an overview of relevant QoS metrics and describe their meaning in the context of publish/subscribe systems [50]. Mahambre *et al.* focus on providing reliability as a QoS metric for publish/subscribe systems, which is a proportion of published events received by a subscriber [51]. However, to the best of our knowledge, QoS guarantees on metrics that have been widely studied in the direct communication paradigm, such as latency, bandwidth, availability, jitter or loss ratio, are not adequately addressed in publish/subscribe systems [52]. In contrast, in our work, the CPU utilization at publishers and subscribers is the primary QoS metric since CPU overloads and congestion at intermediate as well as end nodes are one of the primary sources of end-to-end delays.

E. Control Theory

Control theory, one of the most widely used mathematical frameworks to control the behavior of dynamic systems [12],

is at the core of our model precision control. Due to its robustness against unpredictable operating environments, control theory has been applied to manage the performance of computer systems, such as Web servers [53], and to provide QoS management and real-time scheduling [54]. Sha *et al.* first investigated the benefits of integrating a reactive feedback controller with a queueing-theoretic predictor to guarantee the delays in Web servers [55]. However, queueing theory is not very effective to model the bursty workloads of CPS that RDDS targets. It is demonstrated in [56] that the performance of queueing-model-based feedback control degrades in the presence of bursty workload. In contrast, RDDS combines a control-theoretic feedback controller and a queueing-theoretic predictor in a double control loop, in which the latter sets the reference point that the former tracks.

VI. Conclusion and Future Work

In this paper, we introduced RDDS and its two core mechanisms to handle highly unpredictable nature of CPS environments. First, the semantics of sensor data is exploited to provide a timely, reliable, and scalable dissemination of sensor data. Further, RDDS integrates a reactive feedback controller and a proactive queueing-theoretic predictor in a double control loop to enhance the QoS of TPS. We present performance evaluation using actual experimental prototypes. The results are very encouraging in that the proposed semantics-aware dissemination scheme in RDDS significantly reduces computation and communication overhead. Further, the integrated double control loop provides robustness against unpredictable changes in workloads, which are typical in dynamic CPS.

In the future, we plan to enhance RDDS in several different directions. First, RDDS will be extended to include different modeling schemes. Currently, RDDS supports only a simple modeling scheme using state vectors, without exploiting the correlation among distributed sensors. By utilizing the correlation among sensors, we may further increase the accuracy of RDDS models. However, the cost of using sophisticated models should be evaluated. Second, we are interested in building a testbed that is more realistic. In the original design of RDDS, participating entities are supposed to construct an ad-hoc mesh network dynamically. However, our current testbed uses a fixed network of both wired and wireless connections, ignoring the effect of ad-hoc routing of a mesh network.

References

[1] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," in *Machine Learning in Cyber Trust*. New York: Springer, 2009, pp. 3–13.

[2] K. Sha, W. Shi, and O. Watkins, "Using wireless sensor networks for fire rescue applications: Requirements and challenges," in *Proc. IEEE Int. Conf. Electro/Inform. Technol.*, 2006, pp. 239–244.

[3] J. Wilson, V. Bhargava, A. Redfern, and P. Wright, "A wireless sensor network and incident command interface for urban firefighting," in *Proc. 4th Ann. Int. Conf. Mobile and Ubiquitous Systems: Networking Services, MobiQuitous'07*, Aug. 2007, pp. 1–7.

[4] R. Sanchez, J. Evans, and G. Minden, "Networking on the battlefield: Challenges in highly dynamic multi-hop wireless networks," in *Proc. Conf. Military Commun., MILCOM'99*, 1999, vol. 2, pp. 751–755.

[5] D. Reichardt, M. Miglietta, L. Moretti, P. Morsink, and W. Schulz, "Cartalk 2000: Safe and comfortable driving based upon inter-vehicle-communication," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2002, vol. 2, pp. 545–550.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[7] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *Proc. 23rd Int. Conf. Distrib. Comput. Syst. Workshops*, May 2003, pp. 200–206.

[8] C. Olston, B. T. Loo, and J. Widom, "Adaptive precision setting for cached approximate values," *SIGMOD Rec.*, vol. 30, no. 2, pp. 355–366, 2001.

[9] *Applied Optimal Estimation*, A. Gelb, Ed. Cambridge, MA: MIT Press, 1974.

[10] K. Ramamritham, S. H. Son, and L. C. Dipippo, "Real-time databases and data services," *Real-Time Syst.*, vol. 28, no. 2-3, pp. 179–215, 2004.

[11] L. Ljung, *Systems Identification: Theory for the User*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[12] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. New York: Wiley, 2004.

[13] J. D. C. Little, "A proof for the queueing formula $\mathrm{L} = \lambda \mathrm{w}$," *Oper. Res.*, vol. 9, no. 3, pp. 383–387, 1961.

[14] G. Carl, G. Kesidis, R. R. Brooks, and S. Rai, "Denial-of-service attack-detection techniques," *IEEE Internet Comput.*, vol. 10, pp. 82–89, Jan. 2006.

[15] M. Srivatsa, A. Iyengar, J. Yin, and L. Liu, "Mitigating application-level denial of service attacks on web servers: A client-transparent approach," *ACM Trans. Web*, vol. 2, pp. 15:1–15:49, Jul. 2008.

[16] "Fire information and rescue equipment (FIRE) project," 2008. [Online]. Available: http://fire.me.berkeley.edu/

[17] "Nokia N-Series," 2008. [Online]. Available: http://www.nseries.com/

[18] R. D. F. G. Peacock and W. W. Jones, *CFAST—Consolidated Model of Fire Growth and Smoke Transport (Version 5): User's Guide,* National Institute of Standards and Technology, Gaithersburg, MD, 2005, NIST Special Publication 1034.

[19] "Fire growth and smoke transport modeling with CFAST," 2008. [Online]. Available: http://cfast.nist.gov

[20] "Fire Dynamics Simulator and Smokeview (FDS-SMV)," 2010. [Online]. Available: http://fire.nist.gov/fds/

[21] X. Jiang, N. Chen, J. Hong, K. Wang, L. Takayama, and J. Landay, "Siren: Context-aware computing for firefighting," in *Pervasive Computing*, A. Ferscha and F. Mattern, Eds. Berlin, Germany: Springer-Verlag, 2004, vol. 3001, Lecture Notes in Computer Science, pp. 87–105.

[22] "OpenDDS," 2010. [Online]. Available: http://www.opendds.org

[23] L. Almeida, E. Tovar, J. Fonseca, and F. Vasques, "Schedulability analysis of real-time traffic in WorldFIP networks: An integrated approach," *IEEE Trans. Ind. Electron.*, vol. 49, pp. 1165–1174, Oct. 2002.

[24] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, pp. 239–272, 2007.

[25] D. Loy, D. Dietrich, and H.-J. Schweinzer, Eds., Open Control Networks: LonWorks/EIA 709 Technology Norwell, MA, Kluwer, 2001.

[26] S. Bushby and H. Newman, "The BACnet communication protocol for building automation systems," *ASHRAE J.*, vol. 33, pp. 14–21, Apr. 1991.

[27] J.-P. Thomesse, "Fieldbus technology in industrial automation," *Proc. IEEE*, vol. 93, no. 6, pp. 1073–1101, Jun. 2005.

[28] J. Kaiser and M. Mock, "Implementing the real-time publisher/subscriber model on the controller area network (CAN)," in *Pro. 2nd IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput.*, 1999, pp. 172–181.

[29] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen, "The information bus: An architecture for extensible distributed systems," in *Proc. 14th ACM Symp. Operating Syst. Principles, SOSP'93*, New York, 1993, pp. 58–68.

[30] D. Powell, "Group communication," *Commun. ACM*, vol. 39, no. 4, pp. 50–53, 1996.

[31] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for lightweight sessions and application level framing," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 784–803, 1997.

[32] Y. Huang and H. Garcia-Molina, "Publish/subscribe in a mobile environment," *Wirel. Netw.*, vol. 10, no. 6, pp. 643–652, 2004.

[33] A. Rowe, M. E. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. H. Garrett, J. M. F. Moura, and L. Soibelman, "Sensor Andrew: Large-scale campus-wide sensing and actuation," *IBM J. Res. Develop.*, vol. 55, pp. 1–14, 2011.

[34] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "Irisnet: An architecture for a worldwide sensor web," *IEEE Pervasive Comput.*, vol. 2, pp. 22–33, Oct. 2003.

[35] A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *IEEE MultiMedia*, vol. 14, pp. 8–13, 2007.

[36] A. Sheth, C. Henson, and S. S. Sahoo, "Semantic sensor web," *IEEE Internet Comput.*, vol. 12, pp. 78–83, Jul. 2008.
[37] "Pachube – Data infrastructure for the Internet of things," P. team, 2011. [Online]. Available: http://www.pachube.com
[38] N. Maisonneuve, M. Stevens, and B. Ochab, "Participatory noise pollution monitoring using mobile phones," *Info. Pol.*, vol. 15, pp. 51–71, Apr. 2010.
[39] Y. Chawathe, S. McCanne, and E. Brewer, "RMX: Reliable multicast for heterogeneous networks," in *Proc. IEEE INFOCOM*, 2000, vol. 2, pp. 795–804.
[40] S. Dao, E. Shek, A. Vellaikal, R. R. Muntz, L. Zhang, M. Potkonjak, and O. Wolfson, "Semantic multicast: Intelligently sharing collaborative sessions," *ACM Comput. Surv.*, vol. 31, Jun. 1999.
[41] M. Xiong, K. Ramamritham, J. A. Stankovic, D. Towsley, and R. Sivasankaran, "Scheduling transactions with temporal constraints: Exploiting data semantics," *IEEE Trans. Knowl. Data Eng.*, vol. 14, pp. 1155–1166, Sep. 2002.
[42] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Model-based approximate querying in sensor networks," *VLDB Journal*, vol. 14, pp. 417–443, 2005.
[43] M. Li, D. Ganesan, and P. Shenoy, "Presto: Feedback-driven data management in sensor networks," in *Proc. 3rd Conf. Networked Syst. Design Implementation, NSDI'06*, 2006.
[44] X. Xiao and L. Ni, "Internet QoS: A big picture," *IEEE Network*, vol. 13, no. 2, pp. 8–18, Mar. 1999.
[45] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Commun. Mag.*, vol. 40, no. 5, pp. 116–127, May 2002.
[46] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An Architecture for Differentiated Service,* RFC Editor, 1998.
[47] V. Fay-Wolfe, L. C. DiPippo, G. Cooper, R. Johnston, P. Kortmann, and B. Thuraisingham, "Real-time corba," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 10, pp. 1073–1089, 2000.
[48] R. B. M. Happner and R. Sharma, "Sun microsystems. Java message service specification," 2000. [Online]. Available: http://www.sun.com/products/jms
[49] D. Schmidt and F. Kuhns, "An overview of the real-time corba specification," *Computer*, vol. 33, no. 6, pp. 56–63, Jun. 2000.
[50] S. Behnel, L. Fiege, and G. Muhl, "On quality-of-service and publish-subscribe," in *Proc. 26th IEEE Int. Conf. Workshops Distrib. Comput. Syst., ICDCSW'06*, Washington, DC, 2006, p. 20.
[51] S. P. Mahambre and U. Bellur, "An adaptive approach for ensuring reliability in event based middleware," in *Proc. 2nd Int. Conf. Distrib. Event-Based Syst., DEBS'08*, New York, 2008, pp. 157–168.
[52] F. Araújo and L. Rodrigues, "On QoS-aware publish-subscribe," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst., ICDCSW'02*, Washington, DC, 2002, pp. 511–515.
[53] Y. Diao, N. Gandhi, and J. Hellerstein, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server," *Network Oper. Manage.*, pp. 291–234, Apr. 2002.
[54] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Syst.*, vol. 23, no. 1-2, pp. 85–126, 2002.
[55] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *Proc. 23rd IEEE Real-Time Syst. Symp., RTSS'02*, Washington, DC, 2002, p. 81.
[56] X. Liu, R. Zheng, J. Heo, Q. Wang, and L. Sha, "Timing performance control in web server systems utilizing server internal state information," in *Proc. Joint Int. Conf. Autonomic and Autonomous Syst. Int. Conf. Networking and Services, ICAS-ICNS'05*, 2005, p. 75.

Woochul Kang received the Ph.D. degree in computer science from the University of Virginia, Charlottesville, in 2009.

He is a Research Scientist at the Electronics and Telecommunications Research Institute (ETRI), Korea. Currently, he is investigating a distributed middleware architecture that enables efficient and timely access to real-time sensor data in large-scale distributed cyber-physical systems (CPS). His research interests include cyber-physical systems, real-time embedded systems, large-scale distributed systems, sensor networks, and feedback control of computing systems.

Krasimira Kapitanova (S'11) received the B.S. degree in computer science and technologies from Technical University Sofia, Sofia, Bulgaria, and an M.C.S. degree from the University of Virginia, Charlottesville. Currently, she is working towards the Ph.D. degree in computer science at the University of Virginia.

Her research interests include event description and detection in wireless sensor networks, QoS management, testing, and machine learning.

Sang Hyuk Son (S'84–M'85–SM'98) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, the M.S. degree from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, and the Ph.D. degree in computer science from the University of Maryland, College Park, in 1986.

He is a Professor with the Department of Computer Science, University of Virginia, Charlottesville. His research interests include real-time and embedded systems, database and data services, QoS management, wireless sensor networks, and information security.

Prof. Son is on the Executive Board of the IEEE Technical Committee on Real-Time Systems, for which he served as the Chair during 2007–2008. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS and the *Real-Time Systems Journal*.