iDSRT: Integrated Dynamic Soft Real-Time Architecture for Critical Infrastructure Data Delivery over WLAN

Hoang Nguyen, Raoul Rivas, and Klara Nahrstedt

University of Illinois at Urbana-Champaign, Urbana IL 61801, USA
{hnguyen5, trivas, klara}@illinois.edu

Abstract. The real-time control data delivery system of the Critical Infrastructure (i.e. SCADA - Supervisory Control and Data Acquisition system) is important because appropriate decisions cannot be made without having data delivered in a timely manner. Because these applications use multiple heterogeneous resources such as CPU, network bandwidth and storage, they call for an integrated and coordinated real-time scheduling across multiple resources to meet end-to-end deadlines. We present a design and implementation of *iDSRT* - an integrated dynamic soft real-time system to provide fine-grained end-to-end delay guarantees over WLAN. *iDSRT* takes the deadline partitioning approach: end-to-end deadlines are partitioned into multiple sub-deadlines for CPU scheduling and network scheduling. It integrates three important schedulers: *task scheduler*, *packet scheduler* and *node scheduler* to achieve global coordination. We validate *iDSRT* in Linux and evaluate it in an experimental SCADA test-bed. The results are promising and show that *iDSRT* can successfully achieve soft real-time guarantees in SCADA system with very low packet loss rate compared to available commodity best-effort systems.[1]

Keywords: Multi-resource scheduling, Quality-of-service, WLAN.

1 Introduction

Distributed real-time embedded (DRE) systems are key components of applications for critical infrastructures such as electric grid monitoring and control. These applications may use multiple heterogenous resources such as CPU, network bandwidth and storage. For example, a Phasor Measurement Devices (PMU) in a power substation samples voltage and current at the rate of 60Hz. Sampled data is compressed and encrypted by an embedded processor and sent over a wired/wireless LAN. End-to-end delay of PMU data has to be guaranteed for real-time monitoring purpose. Another example is surveillance cameras

[1] This material is based upon work supported by the National Science Foundation under Grant CNS-0524695 and Vietnam Education Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of those agencies.

in a power substation (similar to [21]). A group of wireless cameras and sensors are placed around the substation for surveillance purpose. Each camera/sensor periodically captures a video frame compressed by an embedded processor and transmitted over a wired/wireless LAN. As shown in these examples, the heterogeneity and interactions of multiple resources in these applications call for an integrated and coordinated real-time scheduling across multiple resources to meet end-to-end deadlines. Unfortunately, even though scheduling for any single resource has been studied extensively, there has been little work done for integrated and coordinated real-time scheduling to meet end-to-end timing constraints (cf. see Section 6).

In this paper, we address the problem of integrated and coordinated scheduling of CPU and WLAN to meet end-to-end delay requirement. We use SCADA (Supervisory Control and Data Acquisition) systems for power substation monitoring as our case study. The general model of SCADA data WLAN is shown in Figure 1. This is a typical scenario specified in [1][2][3]. The scenario includes both real-time monitoring/control and non real-time management applications. Intelligent Electronic Devices (IEDs) periodically send sampling measurements (such as voltage, current, temperature) or video frames (for surveillance purpose) to a *gateway*. The gateway collects and processes sampling measurements (e.g. decompress, decrypt), issues necessary control actions to IEDs and reports necessary information to the control center. The delay requirement in this scenario is in the order of milliseconds [4]. In addition to the *real-time monitor and control* functionality, both the gateway and IEDs need to handle other *management* tasks. For example, the gateway may upload a configuration file to IEDs via a secure protocol (e.g. SSL).

We present a design and implementation of iDSRT - an integrated dynamic soft real-time system to provide fine-grained end-to-end delay guarantees over single-hop wireless networks. To guarantee end-to-end deadlines, iDSRT takes deadline partitioning approach. Specifically, end-to-end deadlines are partitioned into multiple sub-deadlines for CPU scheduling and network scheduling. The partitioning is done in such a way that the total system utilization is minimized for a given task set. To enforce sub-deadline guarantees at each scheduler, it employs

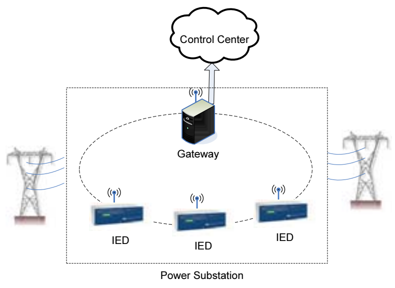

Fig. 1. SCADA data delivery deployment over Wireless LAN in a Power substation

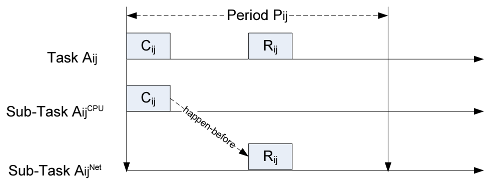

Fig. 2. Task Model

EDF (Earliest Deadline First) scheduling algorithm for both the *task scheduler*, called DSRT (Dynamic Soft real-time CPU scheduler), and the *packet scheduler*, called iEDF (Implicit EDF). The coordination between these two schedulers is executed by a *novel Coordinator entity*, called iCoord, sitting at the middleware layer. iCoord is the key component to deal with the inherent problem of scheduling for wireless network: the shared medium problem. Essentially, iCoord is a distributed node coordination scheduler that ensures every scheduler at each node coordinates with each other to meet end-to-end deadlines. Thus, iCoord plays the role of *node coordination scheduler*. Therefore, iDSRT has a unique approach: the integration of three important schedulers: *task scheduler*, *packet scheduler* and *node scheduler*.

In summary, our contributions in this paper are 1) the design of *an integrated architecture* with protocols and algorithms providing soft real-time end-to-end delay guarantees built on top of commodity Linux operating system and 802.11 MAC layer, 2) implementation of iDSRT including an augmented DSRT, iEDF and the Coordinator middleware and 3) performance study of iDSRT in an SCADA testbed of wireless nodes.

The rest of the paper is organized as follows. Section 2 presents our system model, notations and assumptions. In Section 3.1, we show the architecture of iDSRT and an overview of its components. Section 3.2, Section 3.3 and Section 3.4 give the details of iCoord, DSRT and iEDF. Section 4 presents necessary details of iDSRT implementation. In Section 5, we show our evaluation of iDSRT. Section 6 gives the related work and finally, Section 7 concludes the paper.

2 Models and Definitions

2.1 Network Model

We consider a single-hop wireless network model where each node is within one hop to the gateway as shown in Figure 1. There are n clients (i.e. IEDs) $N_1, N_2, .., N_n$ and a server S (i.e. gateway). Client N_i has m_i ($m_i \geq 0$) real-time (RT) applications/streams and may have best-effort (BE) applications/streams running simultaneously. RT applications stream the data from the client to the server. Each RT application/stream will conform to its QoS specification in terms of end-to-end delay (EED) requirement.

EED is the sum of the delay at the sending side (i.e. at the client side), the propagation delay and the delay at the receiving side (i.e at the server side). Controlling any of these components will affect EED. Our system, however, *only controls the delay at the sending side*. We assume the propagation delay is negligible compared to other two delay components. Furthermore, the receiving delay incurred at the gateway, including computation delay and MAC transmission delay, is small too. The reason is that we assume the gateway is a device with powerful computation and communication capabilities compared to the clients. Hence, controlling of this small delay component does not have much effect on the EED and it is also not the focus of our study.

The sending delay consists of the computation delay incurred by the OS scheduling and the communication delay incurred by the network scheduling. This delay component can be controlled by assigning deadlines to the computation and communication sub-tasks at each client(see Section 2.2). As long as these sub-tasks are finished on time by the OS scheduler and network scheduler, the EED requirements can be met.

In our model, BE applications may stream data to the server. These applications, if not monitored and enforced properly, can affect the QoS performance of other RT tasks because they are not aware of real-time constrains. Typical BE applications in a power substation are FTP application for downloading/uploading devices' configuration or data encryption for secure communication. These network- and computation-intensive applications may exhaustively consume network and CPU resources in the system if not constrained.

2.2 Task Model

We model the RT streaming applications as RT networked tasks, at the client side, composed of the computation and communication sub-tasks. The end-to-end delay requirement of streaming applications is now transformed into the *end-to-end deadlines* of the RT networked tasks used for scheduling.

Formally, we denote A_{ij} for the jth RT networked task/application on the client N_i[2] where $i = 1..n$, $j = 1..m_i$. We also denote A_S as the networked task running on the server S. Each task A_{ij} has a period P_{ij}. It has two sub-tasks A_{ij}^{CPU} and A_{ij}^{Net} that needs to be processed in order (see Figure 2). That means, within period P_{ij}, the sub-task A_{ij}^{CPU} needs C_{ij} time unit for sampling and processing data. After the data gets processed, the sub-task A_{ij}^{Net} needs R_{ij} time units to send it to the server task A_S on server S over the wireless network G. The deadline D_{ij} of task A_{ij} is equal to the period P_{ij}. Both C_{ij} and R_{ij} are CPU and network resources consumed in time. C_{ij} is calculated by the number of consumed cycles over the CPU frequency. We assume the frequency of the CPU is fixed. Similarly, R_{ij} is the time of task A_{ij} and its underlying OS/network protocol stack to transmit a packet of size PS_{ij} bytes over the wireless MAC with measured bandwidth B_{ij} at node N_i, i.e. $R_{ij} = PS_{ij}/B_{ij}$ to the server S.

3 iDSRT Framework

3.1 Overview Design of iDSRT

Our first goal is to design/establish a scheduling and coordination framework of three important schedulers (i.e. the task scheduler, the packet scheduler and the node scheduler) that deliver end-to-end soft real-time guarantees in the system. The second goal is that the system should be able to run on a commodity platform (e.g. commodity Linux-based operating system and 802.11 MAC layer).

Each node N_i will consider time-sensitive scheduling of a) RT tasks $A_{ij}, i = 1..n, j = 1..m_i$ under competition of best-effort tasks, b) network packets of

[2] The terms "RT application A_{ij}" and "RT task A_{ij}" are used exchangeably.

connections belonging to the RT networked application A_{ij} and BE tasks at the node N_i and c) node N_i with respect to other nodes $N_k, k = 1..n, k \neq i$ due to the shared access to wireless medium.

The scheduling and coordination framework resides in the middleware, network and OS layers as shown in Figure 3 and it is called *iDSRT*. It allows RT and BE applications to run together and share resources in controlled manner. RT applications rely on iCoord (Integrated Coordination) - a distributed middleware component residing in the control plane of the protocol stack. It receives QoS specification from RT applications, performs RT application profiling, and does the QoS negotiation on behalf of the RT applications A_{ij}. Its central role is managing resource allocation within each node N_i and among nodes $N_i, i = 1..n$ and S in G to ensure end-to-end delay guarantees (see Section 3.2).

Any potential conflicts among RT tasks $A_{ij}, j = 1..m_i$ and BE tasks on node N_i are resolved by the Dynamic Soft-Real-time CPU Scheduler, called *DSRT* [15]. DSRT guarantees CPU resources for RT applications by using an adaptive EDF scheduling algorithm. It is "soft" because it does not manage other resources of the hardware and thus does not prevent the preemptions due to non-CPU hardware interrupts. However, the soft guarantees are within the timing bounds of SCADA tasks. Section 3.3 will give more details.

The last component in the iDSRT framework is the iEDF (Implicit Earlier Deadline First) packet scheduler. Essentially, iEDF is a network packet scheduler residing on-top of the MAC layer. It takes the implicit contention approach to schedule transmission slots according to the EDF policy. It manages the packet queue of each node and makes sure all nodes agree on the same packet to transmit over the shared medium within a specific time slot (see Section 3.4).

3.2 Integrated Middleware Coordination (iCoord)

iCoord is a distributed middleware component which coordinates all system scheduling components to ensure RT applications meet their deadlines. It operates in the control plane of the node's protocol stack to provide the node registration service, task profiling and coordination services. Its services are a set of middleware libraries whose computation overhead is charged to the calling tasks's computation. Figure 4 shows the middleware control architecture of iCoord. iCoord consists of two modules: *Local iCoord* residing on each client N_i and *Global iCoord* residing on the gateway S. Local iCoord is in charge of coordinating system components at each node N_i and communicates with Global iCoord to assist in inter-node scheduling with other nodes' Local iCoord(s). Global iCoord executes global services on server S, where Local iCoord executes local services on each client N_i. Together, they ensure distributed utility services, such as the coordination service, registration service and the profiling service. Figure 7 summarizes the protocol within iCoord.

Registration Service is a service that takes care of the registration of real-time applications. Essentially, every RT application has to register with iDSRT because un-registered applications are treated as BE applications. First, the registration is done via the *Local iCoord Registrator*. The registration request

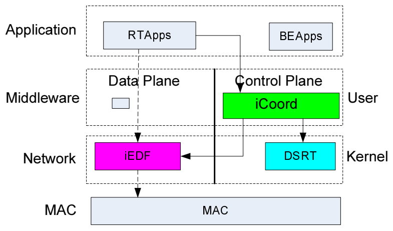

Fig. 3. End-to-End Integrated Dynamic Soft Real-time Framework (iDSRT)

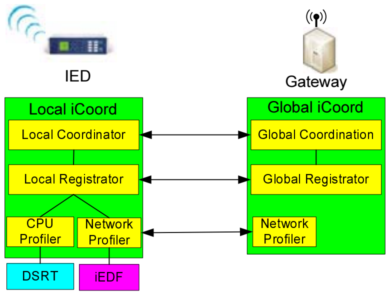

Fig. 4. Middleware control plane architecture iCoord

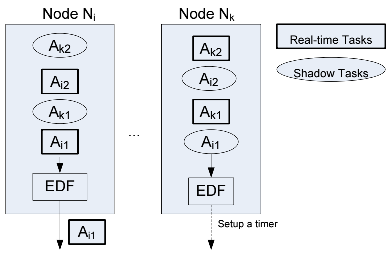

Fig. 5. Illustration of implicit contention scheduling

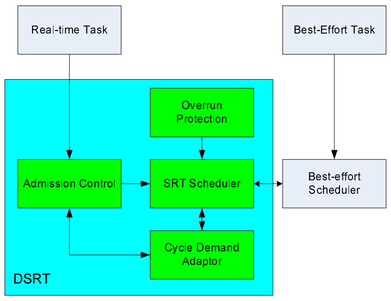

Fig. 6. DSRT Architecture

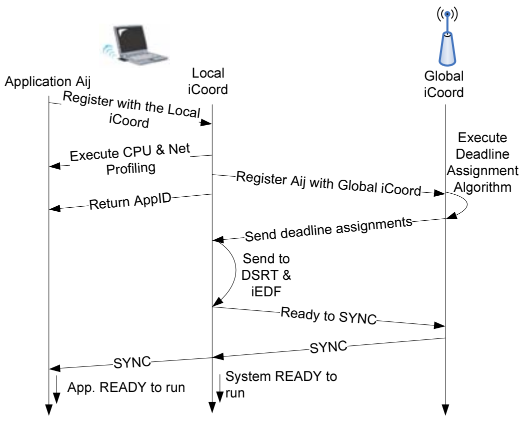
Application Aij
Local iCoord
Global iCoord
Register with the Local iCoord
Execute CPU & Net Profiling
Register Aij with Global iCoord
Execute Deadline Assignment Algorithm
Return AppID
Send deadline assignments
Send to DSRT & iEDF
Ready to SYNC
SYNC
SYNC
App. READY to run
System READY to run

Fig. 7. iCoord protocol

from an RT application A_{ij} includes 1) a tuple of $(pid, saddr, sport, daddr, dport)$ where parameter *pid*, *saddr*, *sport*, *daddr*, *dport* are the process identifier, the source address, the source port, the destination address, the destination port respectively. These parameters are used to uniquely identify each real-time communication application A_{ij}, 2) Period $P_{ij}(\mu s)$ and 3) a requirements $C_{ij}(\mu s)$ on CPU resource and network resource $R_{ij}(\mu s)$ measured by the profiling services.

The Local iCoord Registrator sends the registration information of this application to the *Global iCoord Registrator*. After the Global iCoord Registrator acknowledges the successful registration of the application A_{ij}, the Local iCoord Registrator returns a unique ID calculated from the tuple of registration information to the application. Finally, the Local iCoord Registrator invokes the CPU and network profiling services to approximate the CPU and network usage of the application (i.e. C_{ij} and R_{ij}). Finally, it sends the profiles of this task to the Global iCoord Registrator so that the node admission control, inter-node scheduling and coordination can be performed.

Profiling Service consists of the CPU and network profiler on each client N_i. These profilers are invoked after the registration phase. The CPU usage is measured by having DSRT run several instances of RT task A_{ij}. Similarly, the network profiling is done by measuring the packet round-trip-time between the networked application at the client N_i and the server S.

Coordination service is a distributed middleware component. Similar to the Registration service, the Coordination service has a Global Coordinator at the gateway S and a Local Coordinator at each node N_i. The Global Coordinator at the gateway gathers profiles of all RT applications from the Global Registrator and performs the deadline assignment algorithm (discussed in the next subsection). Then, it sends this information to all *Local iCoord Coordinators*. The information includes deadline assignments for the inter-node (i.e. D_{ij}^{Net}) and intra-node (i.e. D_{ij}^{CPU}) scheduling of all the tasks in the system G.

Upon receiving the deadline assignment of all tasks, the Local Coordinator confirms with DSRT and iEDF about the acceptance of these local tasks. At the end of this phase, each Local iCoord Coordinator notifies the Global iCoord Coordinator that node N_i is ready, and all local components DSRT, iEDF and Local iCoord wait for the SYNC message from the Global iCoord Coordinator. In the last phase, the Global Coordinator waits for all acknowledgments from Local Coordinators and broadcasts the SYNC message. The SYNC message start the run-time of the whole system.

Deadline Assignment Problem: As mentioned in the previous section, we employ the EDF algorithm for CPU scheduler (DSRT) and the network scheduler (iEDF). These two schedulers (DSRT and iEDF) must coordinate with each other so that the end-to-end deadline of RT applications A_{ij} can be met. The approach we take is partitioning the end-to-end deadline into sub-deadlines for the CPU scheduler and network scheduler. Thus, as long as the CPU scheduler and the network scheduler can schedule the sub-tasks correctly, the end-to-end deadlines will be guaranteed. The deadline assignment algorithm is executed by the Global Coordinator whenever there is a newly arrival task. It is essentially a convex optimization algorithm where the deadline is split such that the total stress factor of the two sub-systems is minimized while still satisfying the admission control criteria of the CPU scheduler and the network scheduler. Please refer to our technical report for more information.

3.3 DSRT (Dynamic Soft Real-Time Scheduler)

DSRT is responsible for CPU task scheduling according to their deadlines. Specifically, on client N_i, it manages real-time CPU tasks $A_{ij}^{CPU}, j = 1..m_i$ as modeled in Section 2.2. To achieve this objective, DSRT is composed of three basic components, the Admission Control, the Earliest-Deadline-First (EDF) Scheduler and the Cycle Demand Adaptor.

On a node N_i, before using the realtime capabilities of the system, a new RT task A_{ij}^{CPU} must register itself with iCoord as a RT task in the DSRT. Specifically, it must specify its period, its worst case execution time and its relative deadline[3]. The admission control for DSRT on a node N_i is the EDF schedulability test. It means, $\forall L \in DLset, L \geq \sum_{j=1}^{m_i}(\lfloor\frac{L-D_{ij}^{CPU}}{P_{ij}}\rfloor + 1)C_{ij}$ where $DLset = \{d_{kl} | d_{kl} = lP_{ik} + D_{ik}^{CPU}, 1 \leq k \leq m_i, l \geq 0\}$ is the set including all tasks' deadlines less then the hyper-period of all periods (i.e. least common multiplier of $P_{i1}, .., P_{im_i}$).

If the condition is met, the task A_{ij}^{CPU} is added to the running queue of the EDF Scheduler and is scheduled to run in the next period. If the task cannot complete its job in the allotted time, due to demand cycle variations, the *Overrun Timer* will preempt the task to best-effort mode. In this case, the task A_{ij}^{CPU} will only be allowed to run after all other real-time tasks have used their allotted CPU time. The Overrun Timer removes the task from the running queue and adds it to the overrun queue. Tasks in best-effort mode compete against each other and use the standard OS non-realtime scheduler (Linux in the case of our implementation). Therefore, they cannot get a guaranteed CPU allocation.

If the deadline D_{ij}^{CPU} is not met, the Cycle Demand Adaptor will keep track of this event. If it detects that the change in the cycle demand is persistent and that assigned deadlines are not met, it will try to increase the allotted cycle demand for this particular task A_{ij}^{CPU}. In that case the Cycle Demand Adaptor will query the DSRT admission control to verify whether there are enough CPU resource to increase the allotted resource for the task A_{ij}^{CPU}.

3.4 iEDF (Implicit Earliest Deadline First Packet Scheduler)

iEDF is a distributed network scheduler that takes an "implicit contention" approach to perform the EDF packet scheduling algorithm [7][8]. Each client uses iEDF as its network scheduler. Conceptually, this network scheduler is actually an *outgoing-packet scheduler* working on top of the MAC layer. It manages how packets are prioritized to ensure they will meet the deadlines. Technical information will be given in Section 4.2.

[3] The information C_{ij} and D_{ij}^{CPU} is provided by iCoord as explained in Section 3.2.

iEDF is an implicit contention scheduling which uses EDF as the packet scheduling algorithm. At any time slot, all clients agree on a RT task A_{ij}^{Net} to access the shared wireless medium according to the EDF policy. Specifically, for a client N_i, RT tasks $A_{ij}^{Net}, j = 1..m_i$ running on N_i are called *local RT network applications* and other RT applications running on other clients are called *remote RT network applications.* iEDF at each client N_i maintains the deadline assignment and task information of remote RT network tasks in addition to its local RT network tasks disseminated via iCoord (see Section 3.2).

Once iEDF has all network task deadline information, it creates a "shadow network task" for each remote network task. The shadow network task has the same period, deadline and transmission time as the network task being shadowed. When the shadow network task $A_{kj}^{Net}, j = 1..m_k$ "executes" on $N_i, i \neq k$, it does nothing but sets up a timer to wake up after the transmission of A_{kj}^{Net}. On waking up, the shadow network task again notifies iEDF that the remote network task is supposed to finish. On this event, iEDF schedules another RT network task, either local or remote (shadow) for the next transmission. In this way, iEDF is doing the EDF scheduling algorithm in a distributed manner. Furthermore, packet collisions will rarely happen because iEDF at each client aims to ensure and comply to the global deadline assignment. Figure 5 shows an illustration of this implicit contention.

Even though the principle of iEDF is simple, there are couple of issues that we need to address. The first issue is the correct estimation of the transmission time of the shadow network task. For any particular transmission, the remote network task A_{kj}^{Net} may finish earlier than expected due to worst case profiling and estimation of R_{kj}. It may also finish later than expected due to the noisy and unreliable channel. In the former case, iEDF ignores the early transmission and accepts the waste of idle network resource. In the latter case, iEDF actually has to avoid starting another transmission to minimize the packet collisions. To resolve this issue, iEDF only needs to over-hear the wireless network to know when the remote network task finishes. This is a simple solution yet enough to resolve the scheduling issues. The second issue is that even though iEDF is a network scheduler and consumes non-negligible CPU resource for scheduling. To resolve this issue, we let the network task's CPU consumption to be charged to the computation time of the corresponding RT applications.

4 Implementation

4.1 DSRT Implementation

DSRT was originally implemented by Chu et al. [15] in Linux Kernel 2.4. Due to incompatibilities with Linux Kernel 2.6, DSRT is implemented from scratch in Linux Kernel 2.6. However, our implementation of DSRT is considerably different and includes important contributions to the original work. The main contributions are discussed below.

DSRT originally used the Liu and Layland scheduling model [17], in which the deadlines are considered to be equal to the periods. The coordination algorithm in iDSRT requires a more generalized model in which *the real-time scheduler supports deadlines less than or equal to the periods*. Our implementation of DSRT uses this model instead. Other important difference is that our implementations used new mechanisms developed for precise task accounting, including the CPU timestamp counter available in most modern processors and the new High-Resolution Timer Interface available in the most recent versions of the Linux Kernel 2.6 [10]. The use of this new mechanism allowed us to reduce the number of modifications to the standard kernel. It also allowed us to provide better precision and scheduling granularity than the previous implementation.

DSRT implements nine new system calls allowing RT tasks to communicate with it. These system calls provide DSRT with information required to reserve CPU resource and prioritize a task according to its QoS requirements. In these system calls the task A_{ij}^{CPU} specifies average cycle demands used to calculate C_{ij} (dividing by the CPU frequency), deadline D_{ij}^{CPU} and period P_{ij}. DSRT provides information about the performance and the status of the RT task, including the number of times a task tried to overrun and the statistical CPU utilization.

Our DSRT implementation needs only one kernel patch on the file *sched.c* to provide CPU accounting for each task. Linux currently provides such mechanism in the kernel but only with maximum resolution of 1 *jiffy* (number of iterations of the kernel per second)[4] while we need high precision task accounting to the microsecond resolution. Simply increasing the *jiffy* resolution will cause enormous kernel overhead. In our implementation, we measure CPU usage of real-time tasks in cycles instead of jiffy. This is achieved by adding a hook in $schedule()$ function. This hook is called every time that a context switch is about to occur. It allows us to measure the elapsed cycles between the current and the previous context switch and therefore precisely account for the CPU time of each task. Once done, the number of cycles is converted to time unit by dividing the number of cycles by CPU frequency. The rest of the DSRT is implemented as a kernel module. We use high-resolution timers provided in the kernel to ensure that tasks wake up at precise time and to prevent overruns from greedy BE and RT tasks. The context-switching is implemented as a two-halves operations where interrupts from the timer signals a high-priority kernel thread to preempt the running application.

DSRT has a new data structure to store the QoS parameters $C_{ij}, D_{ij}^{CPU}, P_{ij}$ of the task containing information about the state of the RT task used by both the EDF scheduler and the Cycle Demand Adaptor. When a new RT task makes a request for QoS guarantess to DSRT, DSRT creates a new instance of this data structure (called *srt_task_struct*) containing information about the state of the particular RT task. This task is also cross-referenced with the *task_struct* structure defined by the Linux scheduler to ensure proper communication between the Linux scheduler and DSRT. More precisely, the data structure contains a pointer to the associated *task_struct* structure, necessary information about the

[4] Within Linux 2.6.10, a jiffy is by default 4ms.

state of the RT task in the DSRT scheduler[5], the period, the cycle demand requested, the number of deadlines missed, the number of periods in which the task tried to overrun and the statistical CPU usage in cycles.

Conceptually, DSRT implements 3 *runqueues* that allow the EDF scheduler and the overrun timer to schedule the tasks. The first runqueue is for the RT task process currently ready to run. The second one is for the RT task processes that are running in best effort mode because they overrun. The last one is for the RT task processes that are awaiting for the beginning of the next period. We implement these runqueues as a single list of processes sorted by the EDF policy. We use the information stored in the *srt_task_struct* about the state of the task to differentiate among different runqueues. We avoid implementing more runqueues due to unnecessary kernel overhead. The computational complexity of the DSRT scheduler is $O(log(n))$, where n is the number of RT tasks.

To further minimize the number of changes required to the Linux scheduler, the DSRT scheduler does not load or schedule the RT tasks directly, instead it relies on the Linux scheduler. The DSRT simply rises the priority of the running RT tasks to the highest RT priority available on the system, and requests a reschedule to the Linux kernel. This triggers a context switch and forces the Linux scheduler to pick the task that DSRT wants to be scheduled next. To preempt a running RT task to best effort mode when the overrun timer expires, it simply suffices to lower RT task's priority in the Linux scheduler to normal and rise the priority of another RT task. When all the RT tasks have completed the running job, they yield the CPU by invoking the *sched()* function. Upon the call, the Linux scheduler will take care of scheduling all the BE tasks including iDSRT aware and non-iDSRT aware tasks. The DSRT scheduler remains idle until one of the RT tasks begins a new period. This approach makes the implementation simple and ensure maximal compatibility with non-iDSRT aware tasks.

4.2 iEDF Implementation

iEDF is a queuing discipline in Linux. It communicates with the Local Coordinator via the */proc/* file system. This interface includes *create/modify/delete* a local (shadow) task A_{ij}^{Net} and a $SYNC$ signal with the Local Coordinator. iEDF maintains the information of the tasks A_{ij}^{Net} by a double linked list data structure. Each shadow task in $iEDF$ is implemented as a kernel timer that simulates the same behavior as the corresponding task. Note that even though iEDF may have many timers for shadow tasks, Linux Kernel 2.6 implements these timers as a single high resolution kernel timer to reduce the overhead.

iEDF maintains a FIFO queue for packets of each application. Each entry in a FIFO queue is a pointer to *sk_buff* kernel data structure of a real packet. Thus, iEDF works on pointers to packets to avoid any extra data copy overhead. In addition, iEDF maintains a bitmap representing applications that have packets in the FIFO queues (i.e. one bit per application). Whenever a packet of a RT

[5] Note that a RT task A_{ij}^{CPU} can also become BE task if it violates its assigned deadline D_{ij}^{CPU} (cf. Section 3.3).

application enters the queue, the corresponding bit is set to 1. This bit will be set to 0 when the last packet leaves the corresponding queue. To look up applications with non-empty FIFO queue, iEDF uses *__ffs()* operator on the bitmap to efficiently search for the 1-bit. Similar to DSRT, the complexity of iEDF scheduling is $O(\log(n))$, with n equal to the number of RT tasks.

5 Evaluation

5.1 Experiment Setup

We evaluate iDSRT in a SCADA test-bed of 7 nodes. Each node is a IBM T60 Dual Core 1.66Ghz laptop with 802.11a/b/g Atheros-based wireless card. We disable one core to emphasize the impact of DSRT on CPU scheduling. All laptops run Linux kernel version 2.6.16 with high-resolution timer patch.

We setup the laptops as shown in Figure 8. There is one laptop acting as a gateway. The rest of the laptops are clients. These laptops are emulated IEDs in a wireless SCADA testbed. Each client laptop connects to a real IED such as a Digital Relay and a Phasor Measurement Unit via serial ports. The gateway laptop connects to a SCADA server via Ethernet. To ensure that the gateway is more powerful than the clients, we set the CPU frequency of the server to the highest one (1.66 Ghz) and the CPU frequency of the clients to 1Ghz. All of laptops operate on 802.11a mode and are placed such that they are within the transmission range of each other. The network operates on the channel that has least interference to minimize external effects.

5.2 Scenarios

The RT application in the experiment is a regular periodic task creating/reading and sending a packet every 30ms. This is a typical RT task and time requirement for IED devices measurements (e.g. Phasor Measurement Unit devices) as specified in [4]. Each packet encapsulating a PMU measurement with the RTP-like header has the size of 128 bytes. The RTP-like header contains the sequence number for calculating packet losses and time-stamp for clock synchronization and delay calculation. Other parameters of RT applications such as computation time, network transmission time, sub-deadlines are measured and assigned by iDSRT. To scale up the experiments, each client may have more than one RT application. Specifically, we keep adding the RT applications in the system until the admission control fails. Note that for any number of RT applications in the system, these RT applications are distributed equally to clients. For example, if there are 10 RT applications in the system, each client has $\lfloor 10/6 \rfloor = 1$ application and the remaining four are assigned to any four clients.

Our goal is to to make sure that the system can provide real-time guarantees even there are competing BE applications. On each client we run a very CPU-intensive BE application to compete for the CPU resource. Furthermore, we setup three BE TCP flows from three clients to the server. These three TCP flows will always try to send as much as they can when getting a chance.

5.3 Evaluation Metrics

We compare our iDSRT system against three other systems. The first one, named as "BestEffort" is the combination of commodity Linux and 802.11 MAC. The second one, named as "DSRT only", is the system with DSRT enabled and iEDF disabled (i.e. DSRT and 802.11 MAC) and the last one, named as "iEDF only" has iEDF enabled only and DSRT disabled (i.e. Linux CPU scheduler and iEDF) . The metrics we use are 1) RT end-to-end delay from a client N_i to the gateway S, and 2) the percentage of packet losses of RT applications A_{ij} and 3) the percentage of missing deadlines of RT applications A_{ij}.

All measurements above are done at the gateway S. In every experiment, each application sends 1000 samples, which takes $30ms \times 1000 = 30s$ to finish. The end-to-end delay is measured as the time difference from the packet sent at the client to the time that packet received at the server[6]. The percentage of packet losses are measured by counting the missing sequence numbers. Similarly, the percentage of missing deadlines is measured as the number of packets that are received later than the deadline at the gateway. Also, in each scenario, experiments are repeated 5 times to get the average measurements.

5.4 Experiment Results

End-to-End Delay: Figure 9 shows the end-to-end delay of different systems. The x-axis represents the total number of applications. The y-axis shows the average end-to-end delay in *ms*. In general, only iDSRT can guarantee the deadlines while the other systems cannot. BE system cannot handle the RT applications well because it does not prevent CPU-intensive application and TCP flows from exhausting CPU and network resources. This makes sense because BE system is designed for general purpose, not for real-time purpose.

DSRT-only system prioritizes RT processes and schedules them according to their deadlines. The CPU resource for RT processes is "reserved", i.e. BE processes cannot compete for that reserved resource. That is the reason why DSRT-only system performs better than the BE system. However, because DSRT-only system can only provide RT guarantee on the CPU resource and it lacks of the RT network scheduler, the end-to-end delay cannot be guaranteed.

iEDF-only system performs worst due to two reasons. The first one is the lack of support from the RT CPU scheduler (i.e. DSRT). The BE CPU scheduler (i.e. Linux scheduler) is done in a round-robin fashion and is not aware of application deadlines. Consequently, packets arrive to iEDF in an aperiodic fashion, which may be earlier or later than the slots iEDF reserves for network transmission. If a packet of a RT task arrives to iEDF later than the slot reserved for its network part, it can only be transmitted until the next reserved time slot releases. This causes cascading missing deadlines of a RT task and can only be fixed with a coordination from the CPU scheduler. This explains why it performs worse than the BE system. The other reason is the shared nature of wireless medium among

[6] The clocks of these clients are synchronized on every wireless transmission.

clients which makes consumed network resource grow quickly as the number of RT applications increase. That is why iEDF does not scale as good as DSRT.

iDSRT with the integration of DSRT, iEDF and iCoord performs the best. This result validates our design idea in which node scheduler, packet scheduler and task scheduler have to coordinate very well. Missing any components will not be sufficient to provide soft real-time guarantees.

It is also important to emphasize that iDSRT needs a good profiling and an admission control. In our scenario, iDSRT cannot accept more than 13 RT applications due to the admission control. We did run more than 13 applications and the results basically show that, not admitted RT tasks perform much worse than in BE system because most resources in iDSRT are reserved for admitted RT tasks. This, again, underscores the importance of QoS guarantees: once RT tasks are accepted, they will receive what promised by the system.

Missing deadlines: Figure 10 shows the percentage of missing deadlines of the four systems under various total number of applications. The x-axis shows the total number of applications. The y-axis is the percentage of missing deadlines. Generally, BE system and DSRT-only system have similar percentages of missing deadlines (30% to 40%). In these systems, the main cause for missing deadline is the lack of network scheduling. iEDF-only performs worst as expected due to the lack of support from CPU scheduling and higher resource-consuming rate of each application. iDSRT performs best and has only around 15% missing deadlines.

This figure also shows the nature of “soft” RT guarantees. The reasons for missing deadlines, even in the case of iDSRT, are preemptions due to hardware interrupts, non-preemptive nature of network scheduling (i.e. iEDF cannot preempt a packet being sent) and the unreliable nature of the wireless medium. These are also the reasons for the small fluctuations in the graphs. However, iDSRT with a low average end-to-end delay (around 15ms) and a reasonable missing deadlines (around 15%) is still the best to achieve soft RT guarantees.

To further support the need for the coordination, we show the maximum delay of each systems in Table 1. This table essentially shows the worst end-to-end delay of each system in our experiments. It is clearly shown that iDSRT has the smallest worst end-to-end delay with the deviation of 5ms. In other words, iDSRT misses 15% of deadlines but the deviation is *bounded* in 5ms.

Packet Losses: Figure 11 shows the packet losses of four systems. All systems have very low packet loss rate (less than 0.1%). iEDF in this case shows the

Table 1. Maximum EED (ms)

#Apps	Best Effort	DSRT-only	iEDF-only	iDSRT
6	90.06	90.54	51.58	34.17
7	93.75	88.40	69.90	35.38
8	96.88	92.34	82.82	35.59
9	482.05	101.30	102.43	33.78
10	508.31	109.15	138.21	33.59
11	505.76	97.77	154.13	34.17
12	516.69	128.25	164.42	34.63
13	558.37	136.11	169.35	35.25

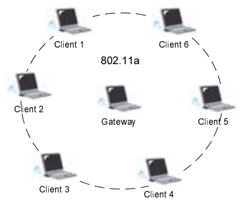

Fig. 8. Wireless SCADA testbed setup

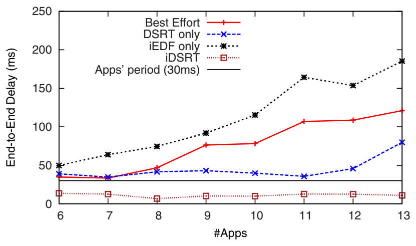

Fig. 9. End-to-End Delay

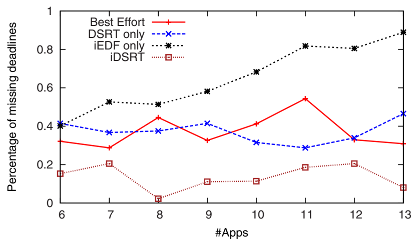

Fig. 10. Missing deadlines

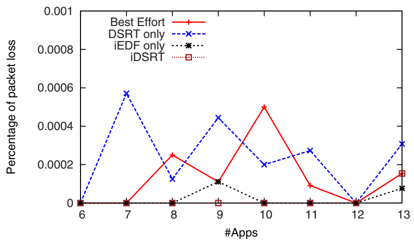

Fig. 11. Packet Loss

advantage of very low (almost no) packet losses due to the distributed scheduling mechanism. iDSRT, again, inherits the advantage of both DSRT and iEDF to achieve almost no packet loss.

6 Related Work

There has been large amount of research work that address individual, part of the end-to-end RT problem and can be classified into three categories: real-time operating system, real-time wireless network and end-to-end delay guarantee. The first category addressing real-time operating system has been extensively explored. Typical work in this category includes hard real-time solutions such as RTLinux [6]; firm real-time solutions such as Rialto [16], SMART[19], KURT [23] and soft real-time solutions such as DSRT [15], GraceOS [27]. RTLinux [6] is a period scheduler but it does not have admission control, the scheduler is a priority-based and non-preemptive. Thus, it does not support applications having deadlines and does not fit in our framework. The Rialto OS [16] focuses on the reservation of the resources. Its application model does not consider deadlines or periods but rather utilization constraints that the system must meet. The Resource Kernel is similar to our proposed work, they provide feedback to the application and use the concept of deadlines and periods. Their approach is not to coordinate the different reservations but to use a priority inheritance algorithm with a bounded waiting time. SMART [19] uses an EDF scheduler and a weighted fair queue scheduler. It uses the Liu and Layland model [17], however they do not consider other resources or any coordination. KURT [23] uses a high-resolution timing system and a tickless kernel. However, it does not provide any guarantees and does not use admission control or reservation of resources. GraceOS [27] is a power-aware soft real-time OS. Its goal is to minimize the power usage of the system based on the QoS constrains specified by each application.

The second category addressing real-time wireless network has been also well explored. A typical work is the 802.11e standard [5]. Although 802.11e becomes the standard and commercially available, its prioritization mechanism does not work well when there are multiple flows with the same priority. In fact, it even increases more collisions due to its aggressive medium access parameters such as smaller CW_{min} and CW_{max}. Besides 802.11e, there are plethora of work involving with MAC design such as dynamic contention adaptation [25][26], RI-EDF [8] or Wireless Token Ring [9]. Even though these schemes can work well, they require MAC modifications. iDSRT requires no modifications of the MAC layer and thus has the advantage of deployability and compatibility. We believe that with the wide-spread availability of 802.11-based hardware, it is much cheaper and more applicable to have a solution working on top of and independent of 802.11 MAC. Several works sharing this view include Overlay MAC [20] and middleware-based control [14] [13]. These systems, however, do not integrate the CPU scheduling into the real-time system and cannot provide a complete end-to-end delay guarantee. iDSRT does this by integrating all three important schedulers: task scheduler, packet scheduler and node scheduler.

In the third category, essentially, previous work has shown the need for integrating the task scheduler and the network scheduler referred to as multi-resource coordination/reservation and scheduling problem [11][12][24][22][21]. In [22][18], the approach is to allocate the resources such that the end-to-end delay can be guaranteed while optimizing the general resource utilization. Xu et al. [24] tries to provide best end-to-end QoS level for an application under the constraints of resource availability in wired networks. In [11][12] end-to-end delay is achieved by assigning deadlines for each resource such that the number of future applications admitted is maximized. However, these works do not address the need for the *coordination among the nodes* because it considers the wired networks. In the wireless scenario, nodes share the wireless medium and thus need to coordinate with each other. The node scheduler is required and this motivates the need for the Coordinator. iDSRT addresses both issues. Thus, it works in the wireless scenario as shown in the paper and should work in the wireline scenario.

7 Conclusions

We have shown an integrated soft real-time scheduling framework, i.e. multi-resource allocation and scheduling for periodic soft-real-time tasks in wireless LAN environment. This is the first integrated system that considers both scheduling and coordination of three important entities in WLAN: the RT tasks, the RT packets and the nodes that share the wireless medium. The result of iDSRT clearly show that augmented Linux and 802.11 WLAN technologies are feasible for critical infrastructures such as PowerGrid SCADA systems and can yield delay and loss guarantees currently only achievable over the wired network with modified general purpose kernels. We believe that iDSRT allows an easy deployment of general purpose hardware and software in PowerGrid substation, while preserving a major requirement of the real-time guarantees.

References

1. General electric wireless SCADA/Telemetry networking, `http://www.microwavedata.com/applications/scada/`
2. SEL-3022 wireless encrypting transceiver, `http://www.selinc.com/sel-3022.htm`
3. IEEE P1777/D1: Draft recommended practice for using wireless data communications in power system operations (February 2007)
4. IEEE Standard 1646: Communication delivery time performance requirements for electric power substation automation (September 2004)
5. IEEE standard 802.11e (September 2004), `http://standards.ieee.org/getieee802/802.11.html`
6. Ayers, Yodaiken, B.V.: Introducing real-time linux. Linux Journal 1997(34es), 5 (1997)
7. Caccamo, M., Zhang, L.Y., Sha, L., Buttazzo, G.: An implicit prioritized access protocol for wireless sensor networks. In: Proceedings of the IEEE Real-Time Systems Symposium, RTSS (2002)
8. Crenshaw, T.L., Hoke, S., Tirumala, A., Caccamo, M.: Robust implicit EDF: A wireless MAC protocol for collaborative real-time systems. Transaction on Embedded Computing System (2007)
9. Ergen, M., Duke Lee, R.S., Varaiya, P.: WTRP: Wireless token ring protocol. IEEE Transaction on Vehicular Technology (2004)
10. Gleixner, T., Molnar, I.: ktimers subsystem, `http://lwn.net/articles/152363/`
11. Gopalan, K., cker Chiueh, T.: Multi-resource allocation and scheduling for periodic soft real-time applications. In: Proceedings of ACM/SPIE Multimedia Computing and Networking (2002)
12. Gopalan, K., Kang, K.-D.: Coordinated allocation and scheduling of multiple resources in real-time operating systems. In: Proceedings of Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT (2007)
13. He, W., Nahrstedt, K.: Impact of upper layer adaptation on end-to-end delay management in wireless ad hoc networks. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS (2006)
14. He, W., Nguyen, H., Nahrstedt, K.: Experimental validation of middleware-based QoS control in 802.11 wireless networks. In: 3rd International Conference on Broadband Communications, Netwoks, and Systems, BROADNETs (2006)
15. hua Chu, H.: CPU Service Classes: A Soft Real Time Framework for Multimedia Applications. PhD thesis, UIUC (1999)
16. Jones, M., Alessandro, J., Paul, F., Leach, J., RoOu, D., RoOu, M.: An overview of the rialto realtime architecture (1996)
17. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1), 46–61 (1973)
18. Nahrstedt, K., hua Chu, H., Narayan, S.: QoS-aware resource management for distributed multimedia applications. Journal on High-Speed Networking, Special Issue on Multimedia Networking (1998)

19. Nieh, J., Lam, M.S.: The design of SMART: A scheduler for multimedia applications. Technical Report CSL-TR-96-697 (1996)
20. Rao, A., Stoica, I.: An overlay MAC layer for 802.11 networks. In: 3rd International Conference on Mobile Systems, Applications, and Services (2005)
21. Shankaran, N., Koutsoukos, X.D., Schmidt, D.C., Xue, Y., Lu, C.: Hierarchical control of multiple resources in distributed real-time and embedded systems. In: Euromicro Conference on Real-time systems (2006)
22. Sourav Ghosh, J.H., Rajkumar, R., Lehoczky, J.: Integrated resource management and scheduling with multi-resource constraints. In: Proceedings of the IEEE Real-Time Systems Symposium, RTSS (2004)
23. Srinivasan, B., Pather, S., Hill, R., Ansari, F., Niehaus, D.: A firm real-time system implementation using commercial off-the-shelf hardware and free software. In: Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium, RTAS (1998)
24. Xu, D., Nahrstedt, K., Viswanathan, A., Wichadakul, D.: Qos and contention-aware multi-resource reservation. In: IEEE International Symposium on High Performance Distributed Computing, HDPC (2000)
25. Yang, Y., Kravets, R.: Achieving delay guarantees in ad hoc networks through dynamic contention window adaptation. In: IEEE Conference on Computer Communication, INFOCOM (2006)
26. Yang, Y., Wang, J., Kravets, R.: Distributed optimal contention window control for elastic traffic in wireless LANs. In: IEEE Conference on Computer Communication, INFOCOM (2005)
27. Yuan, W.: GRACE-OS: An Energy-Efficient Mobile Multimedia Operating System. PhD thesis, UIUC (2004)