CENTRALIZED CONTROL OF WIRELESS SENSOR NETWORKS FOR REAL-TIME APPLICATIONS

Jianping Song [*] **Song Han** [*] **Aloysius K. Mok** [*]
Deji Chen [**] **Mark Nixon** [**]

[*] *Department of Computer Sciences*
University of Texas at Austin
Austin, TX 78712, USA
[**] *Emerson Process Management*
12301 Research Blvd Building III
Austin, TX 78759, USA

Abstract: Wireless sensor networks are usually composed of autonomous nodes. Each networked node constructs its own neighbor table, routing table, and schedules internal tasks on its own. However, the inherently distributed control in a sensor network conflicts with the deterministic requirements when we develop real-time applications on top of wireless sensor networks. In this paper we argue for the case of centralized control in distributed wireless sensor networks. We first describe the characteristics of wireless sensor networks and real-time applications. Then we explain the gap between the demand and supply. We back our arguments with some tentative simulation results. The idea of running real-time applications over wireless sensor networks is motivated by *WirelessHART*, a standard to apply process control over wireless mesh networks. Not surprisingly, *WirelessHART* adopts centralized network management.

Keywords: centralized control, wireless sensor network, real-time application

1. INTRODUCTION

Recently, wireless sensor networks have been a very popular research topic. However, most of the proposed applications, such as environment monitoring, are non-real-time (Song *et al.*, 2006). For other real-time related applications, such as ZigBee, the built-in control functions are limited. For example, ZigBee mesh network is designed mainly for office automation. It provides ways to efficiently manage building energy consumption as well as fire alarm systems, which entails real-time responses.

However, the characteristics of a wireless sensor network makes is ill suited for real-time applications. First of all, each sensor has its own task set and task scheduling. It is a big challenge to have all the sensors to cooperate to support a network wide real-time application. Secondly, the scarce resource on each node makes it hard to provide sophisticated cooperative mechanisms. Thirdly, the dynamic nature of a wireless network may make an existing schedule obsolete and invalid.

In this paper we shall investigate those issues and propose that a centralized scheduling scheme is the better way to go. We shall discuss the pros and cons of centralized and distributed approaches. Although centralized approach is hard in theory, it is simple and practical in real world. Our argument is further supported by the simulation results. In addition, our claim is backed by *WirelessHART*, a process control standard based on wireless sensor networks.

The rest of this paper is organized as follows. Section 2 introduces the concept of real-time tasks and how to model a real-time network application. Section 3 describes a wireless sensor network and its real-time tasks. Section 4 presents different approaches to schedule real-time tasks in a wireless sensor network and Section 5 compares both approaches with some simulations. Section 6 introduces *WirelessHART* standard. The paper is concluded in Section 7.

2. REAL-TIME TASKS

A real-time task T is a 3-tuple $\{C, D, P\}$, where C is the execution time, D is the relative deadline, and P is the period (Chen *et al.*, 1997). At the beginning of each period P, the task T requests an execution of length C that should be finished within D time units. Each request is called a *job*. Normally $D \leq P$. The task T fails if any of its jobs misses its deadline. A real-time task set S is a set of n real-time tasks $T_1, T_2, \ldots, T_n$. A task set S is schedulable by a scheduling policy on a single processor if no job of any task will miss deadline under the control of this policy. S is feasible if S is schedulable by at least one scheduling policy. We shall use the definitions above in this paper thereafter. There are also other real-time task models. For example, a task may have fixed initial start time. In some simple versions a task's deadline equals its period. For continuous real-time applications, a task is usually repetitive, hence the period P in the task definition. P is sometimes also called the minimum separation time to model tasks whose jobs are not exactly periodic. Many schedulability results apply even if P is the minimum separation time.

Research on real-time task scheduling on a single processor is considered mature now. Many well known scheduling policies, such as Earliest-Deadline-First (EDF) (Liu and Layland, 1973), (Stankovic *et al.*, 1998), Rate-Monotonic-Algorithm (RMA) (Sha *et al.*, 1994), and Priority-Ceiling-Protocol (Sha *et al.*, 1990), have already found their ways in real industrial applications. There has also been works done on multi-processor systems in which a task could be selected to run on any processor in the system. In this paper, we assume each node in the sensor network has a single processor and the scheduling algorithm on each of them could be solved with any of the well-known algorithms above once the task set is defined. Next we shall describe how to model real-time applications running on a distributed network.

A real-time application running on a distributed network could be modeled as one real-time task set per node plus real-time data communications among the nodes. A real-time data communication between a source and destination could be considered as one real-time task in the source and one task in the destination, plus the synchronization requirement. Each communication task has an execution time, a relative deadline, and a period. The execution time is the transmission time of the communication. The destination must be in the listening mode when the source is transmitting. Fortunately, there are always ways to schedule the task set once we divide the problem into individual scheduling problems in each node. Since mainstream distribute networks such as Internet adopt best-effort data transmission mechanisms, there are few, if any, real-time applications on distributed networks. It is hard to provide guaranteed data delivery on best effort networks anyway.

Another challenge lies in the case when the source and the destination are not direct neighbors. In this case, all nodes on the path from the source to the destination should deliver the data to the next hop timely so that the total delay is no more than the relative deadline of the communication. Since each node on the path acts independently of one other, it is hard for the nodes down the path to dynamically adjust to the accumulated delay from earlier nodes. An easier way is to pre-configure each node's delay contribution. One possible solution is to pre-assign individual delays on each node. For example, in the RSVP protocol, the bandwidth and delay request is passed from the source to the destination. Each intermediate node returns with the range of commitment it could provide. Then the final allotment on each node is calculated and assigned. To enable real-time applications, some sort of centralized control is necessary.

There have been extensive research works on finding data paths that meet QoS requirements such as delay bounds. Those results could be applied to either centralized or distributed networks. For example, (Ishida *et al.*, 1998), (Jia and Varaiya, 2001), (Liu *et al.*, 2005) proposed different heuristic Delay-Constrained Least-Cost(DCLC) unicast routing algorithms in which the path from source to node is determined by breadth-first or depth-first search on the network graph. It could be done either in a distributed fashion or by a single node with global information. We should point out that DCLC algorithm would be very costly if it is carried out online. For periodic real-time applications, we could use DCLC algorithms offline and apply the result online during application execution.

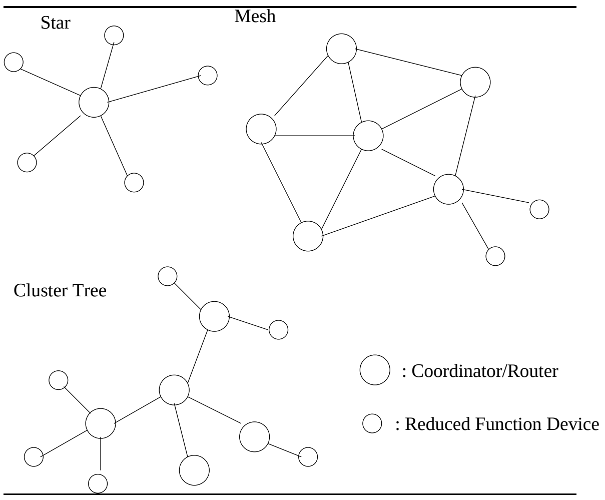

Fig. 1. ZigBee topology models

3. WIRELESS SENSOR NETWORKS

Wireless sensor network is a type of unreliable distributed network. Except for the challenges described in the section above, two unique features of wireless sensor networks makes the issue even worse: wireless communications and low-cost nodes.

Figure 1 shows the topology models of ZigBee networks (ZigBee, 2007). There can be two types of devices in a ZigBee network: Reduced Function Device (RFD) which can not relay data and Full Function Device (FFD) which forwards data from a collection of RFDs. Routers and Coordinators are FFDs. Coordinator manages routers and devices. Sometimes routers are mainline powered and connected to the base station by wireline. Figure 1 shows three basic topologies in ZigBee. In a star topology, sensors are connected to a central router/coordinator. In a cluster tree topology, routers form a tree. Sensors connect to tree nodes. In a mesh topology, stars and cluster trees are connected via their routers. A large scale ZigBee network is a mesh. We discuss sensor networks in the mesh topology in this paper.

While most failures in wireline transmissions can be attributed to collisions, wireless transmissions can fail in more diverse ways. We could no longer assume the reliability of a single path during the lifetime of a real-time application. We have to accept the fact that data may be diverted on different paths at different times which, interestingly enough, is one advantage of wireless sensor networks. Real-time scheduling must consider retrying data package on alternative paths.

Another advantage of wireless sensor network is the low cost of sensors that enables massive deployment. Low cost means minimal memory size and low power usage. Real-time scheduling, on the other hand, requires somewhat complicated algorithm and process power. The dynamic nature of a wireless network entails even more scheduling complexity. It would be prohibitive for a sensor node to provide full support for a global real-time application.

4. CENTRALIZED VERSUS DISTRIBUTED CONTROL

In this section we compare centralized and distributed control of wireless sensor networks from the perspective of supporting real-time applications. By centralized control we mean a node does not generate its own schedule; rather, it executes a schedule generated by and downloaded from a central scheduler such as the base station. The node simply collects communication statistics and forwards them to the central scheduler. On the other hand, by distributed control we mean a node is autonomous. It schedules its own tasks and data processing. It also processes requests from its neighbors and the host. In a centralized network, the central controller generates routing paths and distributes them to each node; in a distributed network, each node builds its own routing information by talking with each other.

We define a real-time application as a function on the base station with an input set and an output set. The input set is a collection of nodes $n_1, n_2, \ldots, n_k$ with their respective sensor values $v_1, v_2, \ldots, v_k$. The output set is also a collection of nodes and the values that will be assigned to them. Figure 2 shows an example on the enlarged mesh network in Figure 1. In Figure 2, the application T runs on node A every 10 seconds. At each execution it reads sample data from sensor nodes C and G, and downloads the result to actuator node H. The dotted arrows show the data directions. Note that there can be multiple concurrent real-time applications on the mesh. There are two alternative paths from C to A: $C \to B \to A$ and $C \to D \to A$. Similarly, there are two alternative paths from G to A: $G \to F \to D \to A$ and $G \to F \to E \to A$. Finally, there are two alternative paths from A to H: $A \to D \to F \to H$ and $A \to E \to F \to H$.

Node D has three types of tasks, routing data for other nodes, handling its own data, and its internal management tasks.

In the following sub-sections we compare centralized and distributed control from five aspects. We shall use node D as an example in some cases.

4.1 Run time scheduling

In centralized case, node D will receive exact data routing schedule from the central manager. Fixed time slots will be allocated every 10 seconds

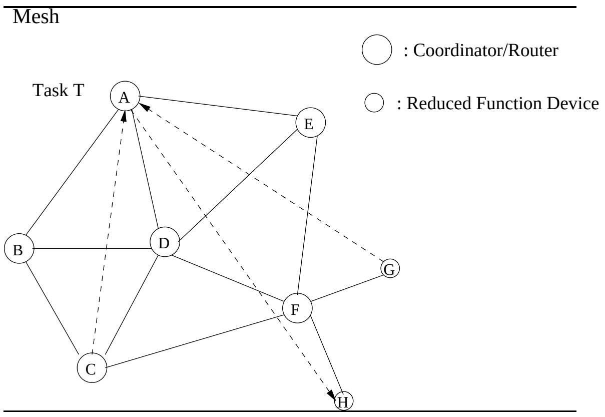

Fig. 2. Real-time application

to route data from application T. The central manager will also schedule node D to route data for other applications and send its own data. When not servicing the above schedules, D is free to handle its internal tasks. In centralized case, D will not cause any deadline misses for T.

If D's schedule is determined by itself, it will treat T's data using its own judgement. The central manager might dictate when D could listen or transmit, or D has to compete for channel access. Either way, T's data no longer has pre-determined delay on D. D may not necessarily assign higher priority to forward data of other applications. Further more, even if D has dedicated time slot for data routing, it could not differentiate data from different applications. This means that applications sharing paths will interfere with each other at the path level, which poses more obstacles to meet real-time requirements.

In centralized control, the schedule is generated offline by the central manager; the networks nodes runs the schedule as simple as using a lookup table. In distributed control, the schedule in each node is usually generated online, which normally takes more memory space and execution time.

Real-time support over the network could not be achieved if network nodes schedule tasks on their own.

4.2 Data path generation

Centralized management has advantage in generating routing paths. By taking into account all possible links and their signal strengths, the central controller could derive the best routing table for each node based on the load, number of hops, signal strength, and more importantly, deadline requirements. All the nodes need to do is to finds its neighbors and measure the signal strength with the neighbors and pass the information to the central manager. For example, in Figure 2 we list two paths for each datum transmission of application T. The paths are balanced.

It would be difficult if each node forms its own knowledge of the network by itself. A good path may be favored by all data transmissions and the nodes on the popular path will exhaust their battery before other nodes. People could always find good distributed routing protocols, but no matter what those protocols are, they could always be implemented on a central controller.

4.3 Device join and leave

We now look at how a node handles the come-and-go of its neighbors. Suppose before a device joins the network, the networks is running applications without any real-time violations. After the device joins the network, it would request data transmission with certain nodes in the network. This added data traffic, if not managed correctly, could interfere with other applications and cause them to fail. In the distributed control, a neighbor of the new device may assign the new device's data the same priority as that of existing applications while it should have assigned the lowest priority to it. On the other hand, it is very difficult for the individual node to make this decision alone.

With centralized control, the newly joined device may not even be able to transmit data without admissions from the central manager. After the central manager decide that the addition of the new node will not affect existing applications, the new device could then safely be admitted into the network and transmit its data.

Distributed control is better when a device dies or voluntarily leaves the network. If the device is a leaf node, there is not much difference between the two control mechanisms. If the device is also a router, the data from its neighbors must be rerouted. Centralized control will be less responsive to re-routing demand as the new schedule would have to be updated by the central manager, while in distributed case, sensors could handle rerouting by themselves locally.

4.4 Collision avoidance

Another advantage of centralized scheduling is collision avoidance. In random channel access scheme, a node having data to transmit first listens on the channel, if the channel is clear, it starts transmitting. If the channel is occupied, the node has to back off and retry later. This mechanism works very well when network traffic is low. However, once many nodes try to transmit at the same time, there would be lots of collisions which may lead to miss deadlines. As for centralized scheduling, a time slot is exclusively used by one transmission. Retry is only to deal with outside interference described in the next subsection.

4.5 Temporary interference

Temporary interference is common in wireless networks. There are many ways to mitigate this problem, such as Direct Sequence Spread Spectrum (*DSSS*) and Frequency Hopping Spread Spectrum (*FHSS*). However, those mechanisms could not eliminate the problem. We have to consider it within the scheduling policies in order to meet real-time requirements. When calculating the worst data transmission delay, we should take into account the retries and re-routing. For temporary interference, retry and re-routing requires similar work for centralized and distributed controls. In both situations, the sender will resend the packet to the same neighbor and, if unsuccessful, try the neighbor on the alternative path. Distributed control may have advantage in that it could retry on re-route according to the failure information. For example, it could choose alternative neighbor that it had the most successful transmissions in the past. A simple node with centralized control might retry the same failed route until told by the central manger otherwise.

Note in general centralized control reduces the scheduling computation in individual nodes, which in turn reduces the cost of the sensors and increases the battery life. Both real-time and non-real-time applications can benefit in this aspect.

5. PERFORMANCE COMPARISON

To compare centralized and distributed network management, we simulate the application T presented in Figure 2. The objective is to check to what extent T will miss deadlines in various network settings. In the simulation, we assume that all other applications are non real-time and they contribute to the background load of the network. The load is defined in terms of the amount of data traversing each node, which includes the data generated by itself and its children.

The network is defined as follows: Node A is mains powered host and has unlimited computation power. Nodes B, C, D, E, and F are sensors that also route data. They have the same hardware and require the same amount of execution time to collect sensing data. G is a sensor leaf with less process power and H is an actuator leaf. Neither G nor H routes data. It takes 1 time unit, which is 10 milliseconds, to transmit or receive one data packet. All data is forwarded to the host. In the centralized mode, a node transmits data according to pre-configured schedule from the central manager. In the distributed mode, a node uses a First-In-First-Out (*FIFO*) queue to store packets. We assume the network runs periodically every 10 seconds. All simulations are run

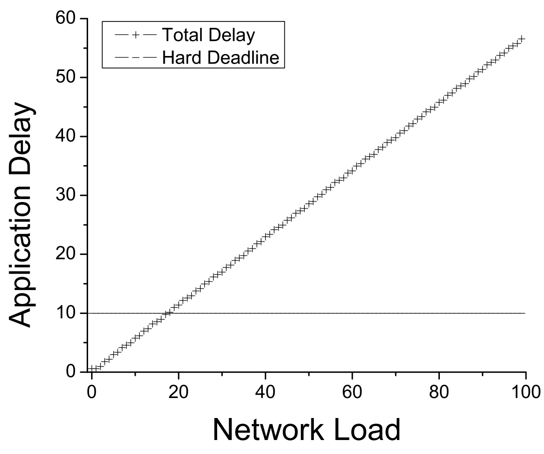

Fig. 3. Application delay vs. network load

for 10 seconds. After which no node will generate any new data and we continue forwarding data that has not arrived at the destination yet. To manage the complexity, we stipulate that nodes A, C, and F transmit at odd time slots and receive at even time slots. Conversely, nodes B, D, E, G, and H enters transmitting mode in even time slots and listening mode in odd time slots. In the first two simulations we introduce no communication interference. We then test with random interferences during each data transmission in the third simulation.

In the first experiment, we explore the effect of network load on the total delay of application T, which is the period from the time the data is generated at the source nodes to the time when the data arrives at the destinations. For the purpose of simplicity, we use the same data size for each node and we do not introduce transmission collision. The simulation result for the distributed management is presented in Figure 3. From this figure, we can see that application T will miss its hard deadline (10 seconds) if the network load increases to 18% and the application delay increases linearly with the rise of the network load. The increase of network load does not affect centralized case because the transmission for T's data has dedicated time slots. This test also reflects the case where new devices join the network. For centralized case, they do not add to network traffic because they are not allowed to send out data until the central control admits them into the network; for distributed case there is no such mechanism and they add to the network traffic immediately.

In the second experiment, we introduce transmission collisions and find the total number of retries for a given system load. A node randomly backs off a few time slots on transmission failure and then retries. The number of back-off slots ranges from 0 to 20. We do not test centralized case as real-time data does not need retry when there is no

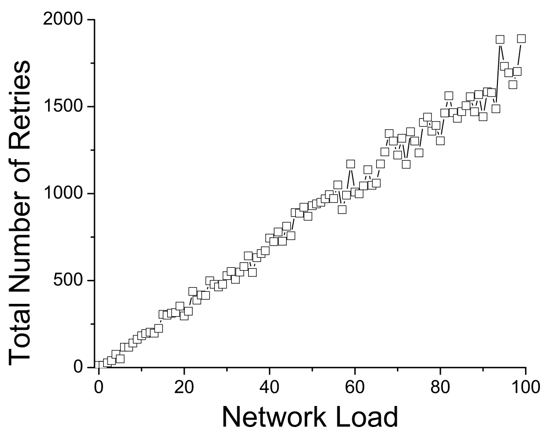

Fig. 4. Retries as a function of system load

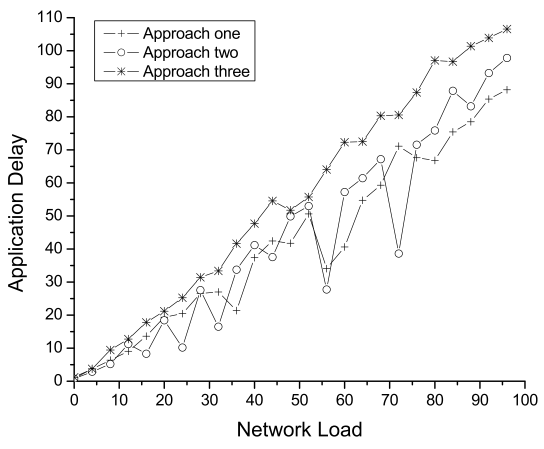

Fig. 5. Application Delay with outside interferences

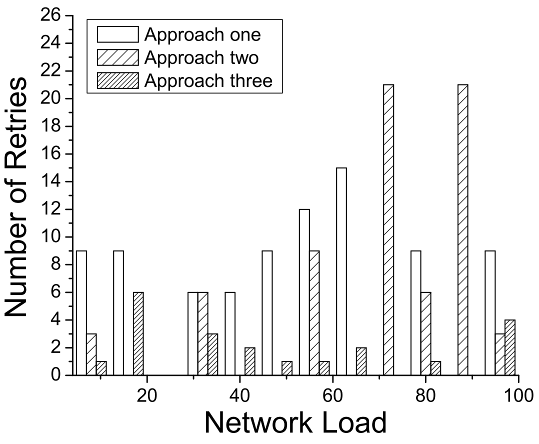

Fig. 6. Retries comparison

outside noise. Figure 4 shows the result for the distributed case. Note that due to random back-offs, the number of retries does not strictly increase monotonically with the system load. As system load increases, a lot of time slots are wasted in the distributed management case.

At last, we introduce some outside interferences into our simulation. In this experiment, we randomly spread noise on the communication links between two neighbors for a random period of time up to 20 time slots and we assume that there is no transmission collision. In the simulation, the node will keep retrying until the transmission succeeds. Thus we count the total transmissions that lead to a successful communication. We list the result for the three pieces of data in application T. As described in Section 4, there are two alternative paths for each data. We compare two retry methods. In the first method, a node tries the other neighbor upon first failure; in the second method, a node retries the same neighbor for one more time before going to the other neighbor. If there is no alternative, the node will keep using the same neighbor until the noise disappears. Figure 5 shows the comparison of the application delay for the two approaches. From this figure, it is easy to find that the first method (try alternative first) excels under our current noise settings. If the noise lasts long enough, retries on the same neighbor always fails. Figure 6 demonstrates the comparison of retries between these approaches. It is hard to draw clear conclusions from this figure. First of all, our simulation setting does not give preference to any of the two retry methods. If the noise always last long, retrying the same neighbor would make no sense and simply waste retry effort. If a connection is always clear for a relatively long time, first trying the successful neighbor from last transmission has a better chance of succeeding again. In a centralized case the central manager may dictate what retry method to use, while in a distributed case individual node could adopt the best method on the fly.

6. WIRELESSHART

Wireless sensor networks have been extensively studied. And there have been some real applications reported, such as environment and habitat monitoring. Recently, people start to explore the possibility of running time-critical industrial applications on wireless networks. In this section we shall introduce one such effort, an industrial standard for wireless process control. This standard adopts centralized network management. We include this section here to provide empirical support for our argument for centralized path routing and schedule management.

HART (HART, 2007) is a very popular industrial fieldbus standard. There are millions of HART devices installed in the field. The massive deployment can be attributed to its simplicity, low cost, ease of use, and high value. Currently, HART is

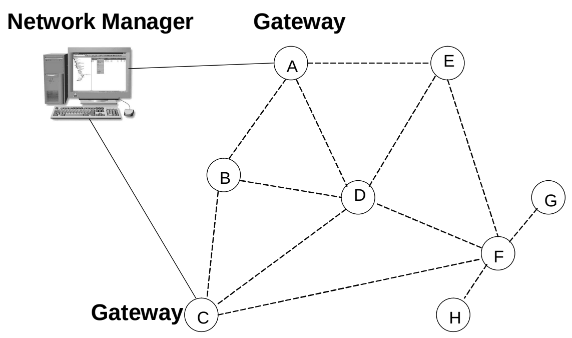

Fig. 7. WirelessHART mesh network

based on wireline transmissions. The next HART version, which was ratified on September 7, 2007, includes wireless transmissions. *WirelessHART* enables host application to access existing HART-enabled field devices through wireless channels. It also supports the deployment of battery-operated wireless-only HART field devices.

Figure 7 shows a *WirelessHART* mesh network with the same topology as that in Figure 2. The network connects to the host network through one or more gateways. In Figure 7, there are two gateways A and C. There is one network manager in *WirelessHART* that controls the whole network. It runs on a mains powered gateway or outside the network. In Figure 7 it runs on a host workstation. A node in the network reports its communication statistics to the network manager. Based on the information, the network manager creates routing graph, configures communication schedules, and downloads this information to each node. A node can only communicate at some pre-configured time slots. The configured time slot defines the transmitter and receiver, the message type, and the wireless channel to use. The schedule produced by the network manager includes time slots for retry, re-route, unscheduled data, and others. Without first getting some time slots a node cannot transmit or receive data. Thus a new device joining the network will not interfere with existing traffic until a new schedule is created and downloaded. Once the schedule is downloaded to each node in the network, the network manager steps out of the way, listens to communication statistics, and re-generates and downloads new schedule when necessary. And the nodes simply execute the schedule that collectively runs the applications over the network.

Now we explain how the network manager could allocate time slots for application T described in Section 4. Suppose that the network is configured to retry the same path first and then the alternative path. There will be time slots of $(C \rightarrow B)$, $(C \rightarrow B)$, and $(C \rightarrow D)$ scheduled for C to send out its data, for the next hop time slots $(B \rightarrow A)$, $(B \rightarrow A)$, $(D \rightarrow A)$, and $(D \rightarrow A)$ would be scheduled for C's data. The same type of time slots will be scheduled for the data from node G to A and from A to H. The order of time slots is also kept so that earlier hops are scheduled before later hops. Also, the control module running in A is scheduled after the sensor data have arrived and before the time slot allocated to send data from A to H. All these time slots must be scheduled within T's period of 10 seconds and be repeated every 10 seconds. Unless the network fails on all data transmission retries, it shall meet the real-time requirements of application T.

Industrial standards, especially those in the process control industry, are very conservative in that safety is utmost important. ZigBee has difficulties getting into the process control market, as it uses a fixed physical channel: any interference on that channel will render the network useless. New technologies such as distributed wireless control presents more risks than centralized control. Even with centralized control, the initial target of *WirelessHART* is still industry monitoring and non-critical simple loop control.

7. CONCLUSION

In this paper we explore the case of running real-time applications on wireless sensor networks. As a sensor network is essentially distributed, each sensor is autonomous, which means it schedules internal tasks and data communications by itself. To reach this goal, we propose centralized control mechanisms, instead of distributed control. We back our claim by first analyzing the features of real-time applications and wireless networks. We then compare the differences between centralized and distributed management. The tentative simulation results also support our conclusions.

Applying wireless technologies in real-time applications is relatively new and challenging. Some industry organizations are also pursuing this idea. WirelessHART is a result of such pursuit. WirelessHART defines an industrial process control protocol suite on top of wireless communications. It proves from another viewpoint that centralized network management is preferable to distributed control for real-time applications.

REFERENCES

Chen, D., A.K. Mok and S.K. Buruah (1997). On modeling real-time task systems. *Lecture Notes in Computer Science - Lectures on Embedded Systems.*

HART (2007). http://www.hartcomm.org.

Ishida, K., K. Amano and N. Kannari (1998). A delay-constrained least-cost path routing protocol and the synthesis method. *The 5th Internation Conference on Real-Time Computing Systems and Applications.*

Jia, Z. and P. Varaiya (2001). Heuristic methods for delay constrained least cost routing using k-shortest-paths. *INFOCOM.*

Liu, C.L. and J.W. Layland (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM.*

Liu, W., W. Lou and Y. Fang (2005). An efficient quality of service routing algorithm for delay-sensitive applications. *Computer Networks.*

Sha, L., R. Ragunathan and S. Sathaye (1994). Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *In Proceeding of the IEEE.*

Sha, L., R. Rajkumar and J. Lehoczky (1990). Priority inheritance protocols: An approach to real-time system synchronization. *IEEE Trans. on Computers* **39**, 1175–1185.

Song, J., A.K. Mok, D. Chen and M. Nixon (2006). Using real-time logic synthesis tool to achieve process control over wireless sensor networks. *The 12th IEEE Conference on Embedded and Real-Time Computing Systems and Applications.*

Stankovic, J., M. Spuri, K. Ramamritham and G. Buttazzo (1998). *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms.* Kluwer Academic Publishers. Boston, USA.

ZigBee (2007). http://www.zigbee.org.