Latency Minimizing Tasking for Information Processing Systems

James Horey, Brent Lagesse
Computational Sciences and Engineering
Oak Ridge National Laboratory
Oak Ridge, TN, USA
email: {horeyjl, lagessebj}@ornl.gov

***Abstract*—Real-time cyber-physical systems and information processing clusters require system designers to consider the total latency involved in collecting and aggregating data. For example, applications such as wild-fire monitoring require data to be presented to users in a timely manner. However, most models and algorithms for sensor networks have focused on alternative metrics such as energy efficiency. In this paper, we present a new model of sensor network aggregation that focuses on total latency. Our model is flexible and enables users to configure varying transmission and computation time on a node-by-node basis, and thus enables the simulation of complex computational phenomena. In addition, we present results from three tasking algorithms that trade-off local communication for overall latency performance. These algorithms are evaluated in simulated networks of up to 200 nodes.**

***Keywords*-sensor networks; scheduling; information processing; aggregation**

I. Introduction

Total time latency is an important metric for emerging information processing and knowledge discovery systems. For example, sensor networks play a key role in several emerging applications, including environmental monitoring [1], civil engineering [2], wild-fire monitoring [3], and emergency dispatch [4]. Many of these applications require complex analysis to occur in the network and for results to be transmitted back to users in a timely fashion. Due to the limited energy capacities of sensor nodes, much research has gone into energy-efficient communication protocols and system architectures [5]. Although still important, less attention has been focused on time latency. Other types of information processing systems, including MapReduce [6] clusters, exhibit similar timeliness constraints.

We present a tree-based model of information processing systems that enables us to evaluate total latency with respect to different network topologies and tasking strategies. In this paper, we design and evaluate three tasking algorithms that trade-off communication needs with total latency. The rest of this paper is organized as follows. The aggregation and communication model is reviewed in Section II. Section III discusses the tasking algorithms we use to generate task schedules for the sensor network. Results using these tasking algorithms are demonstrated in Section IV. Finally, we compare our work to related works (Section V) and offer a brief conclusion (Section VI).

II. Aggregation Model

We model information processing systems using a tree-based aggregation scheme. For sensor networks and other less capable systems, we also model the network basestation (i.e., a capable workstation connected to the sensor network) as the root of the tree. Tree-based routing structures are popular within the sensor network community due to the ease of implementation and ability to aggregate data at subtree roots [7]. Likewise, aggregation trees are commonly used in many data-parallel programming models [8].

In our model, each network node collects data (representing temperature, humidity, etc.) at some initial time. The data flows toward the root as each node transmits the data to its parent node. Along the way, the data will become aggregated if the data is passed through a node that has been assigned computation (e.g. *tasked*). Once the data has reached the root, the remaining data is aggregated. In our model, all the nodes are synchronized and time is measured in discrete timesteps.

In order to model a variety of network settings, each node in our simulation includes parameters to control relative computation time (number of timesteps to compute over n bytes of data) and transmission time (number of timesteps to transmit n bytes of data to the parent node). These parameters can be used to model a variety of information processing scenarios including sensor nodes, MapReduce clusters, and faulty nodes (where nodes have higher relative transmission time).

Once the parameter values are set, each node can dynamically be in one of three states: *idle* (waiting for data), *compute* (computing over some data), and *transmit* (transmitting data). Each node intially collects data (the amount of data is set by the user) and enters the transmit state. Upon receiving data, a node will enter the compute state assuming the node is tasked. If the node is not tasked, then the node enters the transmit state. It stays in these states for the time specified by relative computation and transmission parameters. Finally, if the node does not receive data, it enters the idle state.

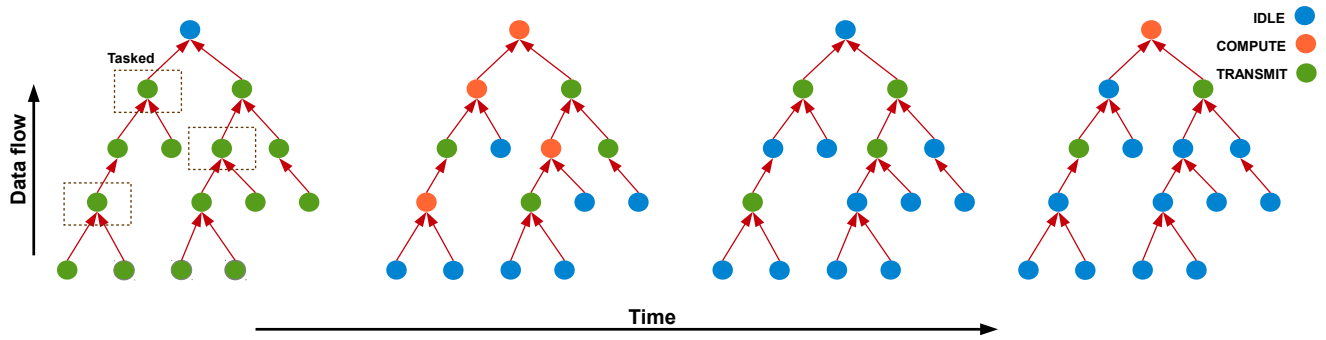

Figure 1. Illustration of the aggregation model. Data is generated by sensor nodes and is transmitted towards the root of the routing tree. Tasked nodes aggregate the data and transmit results.

An illustration of data being transmitted and processed is shown in Figure 1. In this illustration, every node has set its relative computation and transmission times to one timestep. Normally those parameters would have higher values, but we choose a single timestep to simplify the flow diagram. Initially all nodes are in an idle state (not shown in the figure). In the next timestep, the nodes enter the transmit state. As time progresses, the data is transmitted towards the root. Upon receiving data, tasked nodes enter the compute state. After aggregating the data, the nodes transmit their results.

III. Tasking Strategies

The goal of the tasking algorithm is to determine which of the eligible nodes in the network should be tasked. A node is deemed to be eligible if it is an internal node (versus a leaf node). If the node is tasked, it then performs aggregation over the data received from its subtree. Although the goal is relatively straightforward, we will demonstrate that different tasking schedules can yield very different total latencies (Section IV). We measure total latency as the time from the initial data collection to processing the final values at the tree root.

The challenge associated with tasking is that computation can often be more time intensive than communication. This will be the case when the communication is heavily constrained (e.g. a harsh environment, or sparse network), or if the computation is complex (e.g. complex reduce operations [9]). Poor tasking strategies may inadvertently cause bottlenecks in the network thus resulting in poor latency performance.

We assume that tasking strategies take into account a coverage value c. The coverage value represents the proportion of nodes that should be tasked, and can range from 0 (none of the nodes should be tasked) to 1 (all of the nodes should be tasked). The coverage value is important for systems in which nodes must remain responsive to the external environment (e.g., interactive systems, sensor networks). Consequently, it is often desirable to minimize the number of nodes executing the computation.

We have implemented and evaluated three different tasking algorithms: random, greedy, and genetic algorithm. These tasking strategies vary in the amount of information required from the network, and consequently yield different latency performances.

A. Random Algorithm

In the random algorithm each node decides independently whether it should be tasked with probability c (assuming the node is eligible). This results in a uniformly distributed random tasking. Because each node does not rely on information from any other node, this is both the simplest and least communication intensive tasking method. Although simple, the random algorithm is very useful as a null model by which to compare other, more complicated tasking algorithms [10].

B. Genetic Algorithm

We have also implemented and evaluated a genetic algorithm based tasking method. In this method, tasking schedules are represented as a genome. The genome consists of a vector v of length $c*n$ (n is number of eligible nodes). Elements of v are integers representing the individual tasked nodes. Initially each genome consists of a random tasking schedule. The genomes are then evaluated for fitness. In order to determine the fitness of a genome, the tasking schedule is instantiated and simulated on the network. The longer the execution time, the lower the fitness.

After evaluating the fitness of each candidate, the top genomes are pair-wise selected for crossover. During crossover, approximately half of each parents' tasking schedules are selected to create a new schedule. Afterward the new schedule undergoes mutation, in which a small number of elements in the new schedule are randomly changed to another eligible node. New schedules are created until the desired population number is reached. This is continued iteratively for a pre-set number of generations. For our

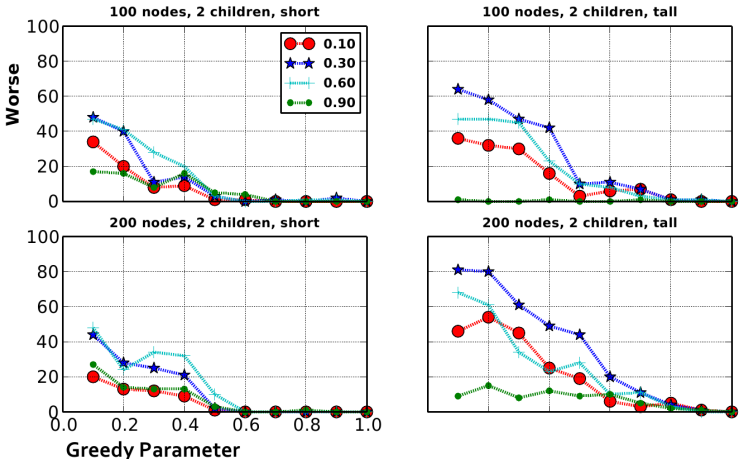

Figure 2. Number of schedules that perform worse than random as the greedy parameter (g) is increased for different coverage values ($0.10 - 0.90$)

evaluations, we used a moderate population size (200), and 50 generations. Increasing the population size and number of generations did not noticeably improve the schedule latencies.

Unlike the random algorithm, the GA produces near optimal tasking schedules. However, the GA requires knowledge of the entire state of the sensor network, and must execute on a centralized basestation. In addition, GA execution can often take a very long time depending on the number of generations evaluated and the desired population size.

C. Greedy Algorithm

We have also implemented a distributed, greedy algorithm. Like the random algorithm, this algorithm executes on individual nodes (as opposed to executing on the basestation). Each node first recursively tasks all nodes in its local subtree. Once the subtree has been recursively tasked, the node then tasks itself and estimates the total latency within its subtree (but not for the entire network). This is done by simulating the subtree with the given tasking schedule. Afterwards, the node de-tasks itself and re-estimates the total latency with the new schedule. The node finally chooses the strategy that yields the lower latency.

Unlike the random algorithm, the greedy algorithm requires a node to communicate with its immediate children. After tasking, each child must propagate its tasking schedule to the parent so that the parent can effectively estimate the subtree latency. In our current algorithm, a node cannot influence whether any of its immediate children are tasked or not, it can only control its own tasking. Although we do not expect this algorithm to perform as well as the GA, this algorithm is expected to yield better results than the random algorithm.

IV. Evaluation

The network model and each of the three tasking algorithms were implemented and evaluated in simulation with network sizes of up to 200 nodes. Each simulated network was organized into a random routing tree in which each node had a single parent and multiple children. We evaluated trees with both a maximum of 2 or 3 children, and varied the overall height of the routing trees. The relative computation time was set to twice that of the transmission time. In addition, the basestation was set to be five times faster than the other sensor nodes. Although other relative computation speeds can be used, we chose these values to reflect a relatively conservative information processing system.

A. Greedy Algorithm Parameters

For the greedy algorithm we include an additional parameter, g, that determines the minimum subtree size necessary to execute the greedy scheduler. When g is set to 0, all the nodes are scheduled in a greedy fashion. When g is set to 1, all the eligible nodes are tasked using the random scheduler. This parameter can be used to fine-tune the amount of local communication in the network. The larger the value, the less the communication.

Although one may initially suspect that setting g to 0 always performs better (relative to other values of g), we have not found this to be the case. In order to compare the greedy scheduler to the random scheduler, we first assign the same pseudo-random seed to both schedulers. That way when the greedy scheduler executes using a large g value, it will make the same âĂIJrandomâĂİ decisions as the random scheduler (at least on the subtrees). In this way, we can evaluate how well the greedy decisions affect the total latency.

Figure 2 illustrates the number of schedules that perform worse than random as g is increased. Each set of points

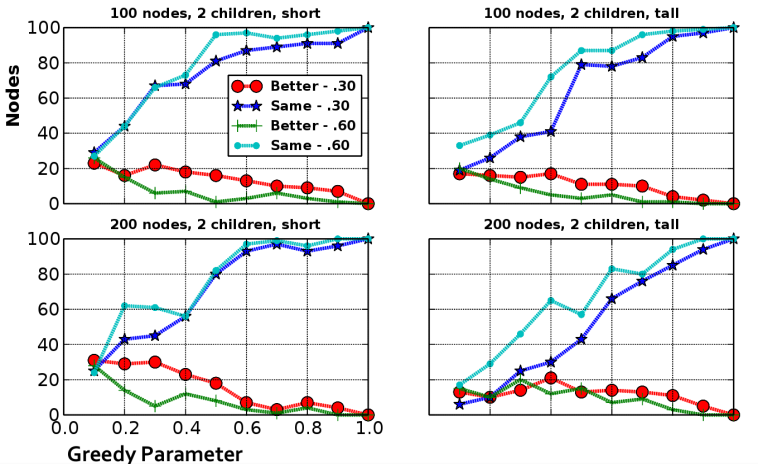

Figure 3. Number of schedules that perform better than random as the greedy parameter (g) is increased for different coverage values ($0.10 - 0.90$)

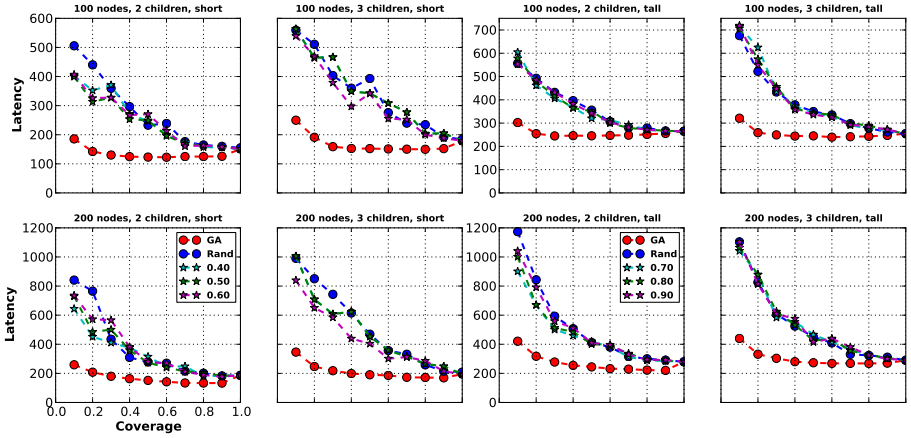

Figure 4. Total latency of tasking algorithms at different coverage levels.

are for a different coverage value (0.10 âĂŞ 0.90). Because setting g to 1 is equivalent to the random scheduler, the number of worse schedules is zero at $g = 1$ (all the schedules are equivalent). We observe that as g increases the number of worse schedules quickly decreases in most network configurations (g 0.40). Although the least greedy scheduler ($g = 0.90$) generates the fewest worse schedules, this is largely because this scheduler is mostly random with very few greedy decisions. We can observe that the most greedy scheduler ($g = 0.10$), usually performs better than the other schedulers (2040 worse schedules). Somewhat surprisingly, when g is set to 0.30, the scheduler performs worse relative to $g = 0.60$.

In order to capture a more complete picture, we must also observe the number of better solutions with varying g and c values. Figure 3 illustrates the number of schedules that performed better or equivalent to random as g increases for different coverage values (0.30, 0.60). We observe that as g increases, the number of better schedules slowly decreases, while the number of equivalent schedules increases. This indicates that having a low g value yields the most number of better solutions. However, this also yields very few equivalent schedules and many worse schedules. These results indicate that there is a fundamental tradeoff in the greedy scheduler. Lower g values (i.e., the greedier the tasking strategy) will have a larger number of *better* and *worse* schedules. Finally, we also observe that the greedy scheduler simply does not generate that many better schedules (< 20 / 100).

B. Latency vs Coverage

To study the effects of varying the coverage parameter (c), we varied c and measured the total latency for several network configurations. We measured the latency for the random, GA, and greedy schedulers. For the greedy scheduler, the greedy g parameter was varied from 0.40 âĂŞ 0.90. Each

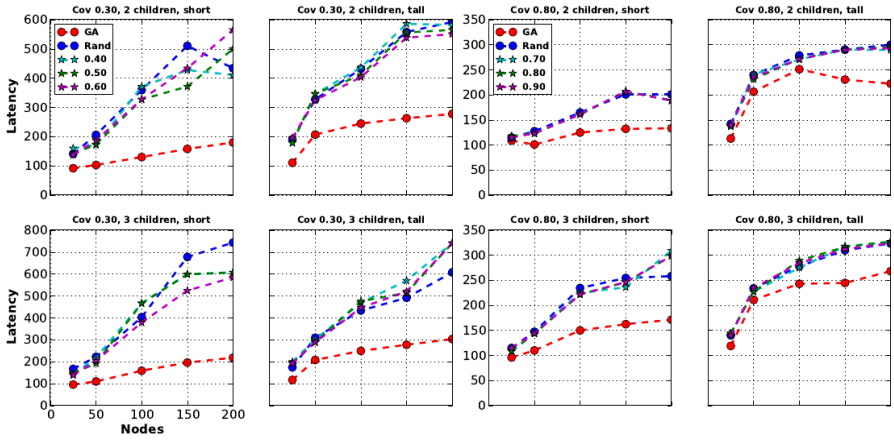

Figure 5. Total latency of tasking algorithms at different coverage levels.

data point was averaged over ten independent runs.

As Figure 4 illustrates, the GA consistently produced the lowest latency schedules. This is expected since genetic algorithms are designed to search for optimal solutions in complex spaces. As the coverage increases, the latency decreases for all schedules and eventually converges. Even though computation takes twice as long as communication, the communications savings are often sufficient to make it worthwhile to aggregate. Somewhat surprisingly, with the GA, latency usually flattens out after a small amount of coverage (> 0.40). Indeed, in a few cases, the latency actually increases when the coverage is set to a high value (> 0.90). These observations indicate that when using the GA scheduler, the user can choose an appropriately low coverage value without sacrificing total latency.

Surprisingly, the greedy schedulers only perform marginally better than the random scheduler. Most of the greedy schedulers perform better with low coverage values (< 0.40). In addition, the greedy schedulers tend to perform better on well-balanced trees. This is partially expected since well-balanced trees enable the greedy scheduler to make more âĂIJdecisionsâĂİ at each depth (i.e. there are more eligible nodes at any particular depth). As coverage increases or as the trees become less balanced, the greedy schedulers perform equally as the random scheduler. These results indicate that a future distributed scheduler should focus on vertical scheduling (across the tree depths) for further advantages.

C. Latency vs Nodes

We also evaluated the effects of increasing the number of nodes in the sensor network. This information can be used to bound the approximate total latency as nodes are introduced or removed from a system. Like the previous experiments, the sensor networks were organized into trees with a maximum of two or three children, and also organized into a well-balanced "short" tree, and unbalanced "tall" tree.

As illustrated in Figure 5, the total latency increases quite differently depending on the sensor network configuration. For networks with low coverage ($c = 0.30$), the GA latency increases slowly from 100 timesteps to < 300 timesteps. However, for the random and greedy schedulers, latency increases approximately linearly. For trees with high coverage ($c = 0.80$), latency increases more slowly for all network configurations. Indeed, for the GA scheduler, adding more nodes (> 100) does not significantly increase the total latency. These results imply that there is much room for improvement in future distributed tasking algorithms.

V. Related Works

In-network aggregation in sensor networks is not a new concept [11], [12]. Systems [13], [14], including TinyDB [15], employ an SQL-like language to specify aggregation operations (e.g. average, minimum, etc.). Unlike our work, however, these systems primarily focus on energy efficiency by actively manipulating the tree routing structure. In addition, the models employed by these systems do not explicitly consider relative transmission and computation time, making them unsuitable for latency analysis.

Real-time cyber-physical systems have gained much attention recently due to their importance in many real-world applications. Our scenario addresses a slightly different problem than the one posed in typical real-time sensor networks. We assume that the aggregation operations can be controlled explicitly by the user using a coverage parameter. Abdelzaher et al. [16] uses a control feedback mechanism to minimize latency. However, their work does not explicitly take into account relative computation time. The question we address assumes that a routing mechanism is already in place [17], and that the challenge lies in scheduling the

reduce operations given a routing tree, coverage value, and relative computation time.

Previous works have also explored the role of aggregation in the total latency [18], [19]. However, these works focus on homogeneous environments (e.g. the relative speed of the basestation is not considered) and do not explicitly model computation and transmission time. Also, these works actively select the routing path, while our algorithms assume that the routing path is determined by the operating system and is not directly controlled by the aggregation scheduler.

VI. Conclusion

We've presented an aggregation-focused model of sensor networks that can be used to study the trade-offs between computational coverage and total latency. Our model explicitly takes into account transmission and computation times, and enables users to define different values for the basestation. In addition, we've presented three different tasking algorithms that operate over model to produce aggregation schedules of varying quality. In the future, we expect to continue exploring distributed tasking algorithms for information processing systems. We've shown that the gap between highly optimized schedules that use global information is quite large relative to our distributed algorithms. This gives us encouragement that future distributed tasking algorithms can still make large gains.

VII. Acknowledgements

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat monitoring: Application driver for wireless communications technology," in *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, 2001.

[2] K. Chebrolu, B. Raman, N. Mishra, P. K. Valiveti, and R. Kumar, "Brimon: A sensor network system for railway bridge monitoring," in *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2008.

[3] C. Hartung, R. Han, C. Seielstad, and S. Holbrook, "Firewxnet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments," in *ACM Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2006.

[4] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, "Codeblue: An ad hoc sensor network infrastructure for emergency medical care," in *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2004.

[5] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004.

[7] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[8] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. Kumar, and G. J. Currey, "Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language," 2008.

[9] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: interfaces and implementations," in *ACM Symposium on Operating Systems Principles (SOSP)*. New York, NY, USA: ACM, 2009, pp. 247–260.

[10] J. Horey, "Challenges in scheduling aggregation in cyber-physical information processing systems," in *Workshop on Knowledge Discovery Using Cloud and Distributed Platforms (with ICDM)*, 2010.

[11] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions Networking*, vol. 11, no. 1, pp. 2–16, 2003.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[13] J. Horey, E. Nelson, and A. B. Maccabe, "Tables: A spreadsheet-inspired programming model for sensor networks," in *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2010.

[14] Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," in *ACM SIGMOD Conference*, 2002.

[15] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Transaction Database Systems*, pp. 122–173, 2005.

[16] T. Abdelzaher, T. He, and J. Stankovic, "Feedback control of data aggregation in sensor networks," in *IEEE Conference on Decision and Control*, Atlantis, Paradise Island, Bahamas, 2004.

[17] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "Speed: A stateless protocol for real-time communication in sensor networks." Los Alamitos, CA, USA: IEEE Computer Society, 2003.

[18] X. Chen, X. Hu, and J. Zhu, "Minimum data aggregation time problem in wireless sensor networks," in *International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, Wuhan, China, 2005.

[19] B. Yu, J. Li, and Y. Li, "Distributed data aggregation scheduling in wireless sensor networks," in *IEEE Conference on Computer Communications*, Rio de Janeiro, Brazil, 2009.