TIGRA: *Ti*mely Sensor Data Collection Using Distributed *Gra*ph Coloring

Lilia Paradis
Microsoft Corporation
Redmond, WA
lilia.paradis@gmail.com

Qi Han
Department of Mathematical and Computer Sciences
Colorado School of Mines, Golden, CO 80401
qhan@mines.edu

Abstract

In this paper we present a protocol for sensor applications that require periodic collection of raw data reports from the entire network in a timely manner. We formulate the problem as an NP-hard graph coloring problem. We then present TIGRA - a distributed heuristic for graph coloring that takes into account application semantics and special characteristics of sensor networks. TIGRA ensures that no interference occurs and spatial channel reuse is maximized by assigning a specific time slot for each node to transmit. Although the end-to-end delay incurred by sensor data collection largely depends on specific topology, platform, and application, TIGRA provides a transmission schedule that guarantees near-optimal delay on sensor data collection.

1. Introduction

Wireless sensor networks (WSNs) may be used in a variety of event-driven applications where the network is normally idle and only activated in response to a critical change in the observed phenomena. Once an event is detected, frequent and periodic updates from the sensors to the sink are needed for better understanding of event evolution to ensure prompt response. Sensor data here is very time-sensitive and a real-time communication protocol needs to be in place to ensure timely data delivery. In addition, these applications often require raw data from the sensor network. For instance, when using WSNs to monitor contaminant plumes, the raw data sensed by all the sensors will be used by numerical models run at the server to predict the behavior of the contaminant. Note that the need of raw data does not prevent the system from combining sensor readings from different nodes.

It is desirable to provide a guarantee on the latency in delivering data from multiple sources to a single sink. A known latency bound ensures real-time event detection and helps the sink to schedule application request. However, this presents several challenges. First, multihop communication coupled with the potential of combining several sensor reports into one data packet creates unique precedence constraints. Second, both primary and secondary conflicts in wireless networks must be avoided [5]. A primary conflict occurs when a node transmits and receives at the same time slot or receives more than one transmission destined to it at the same time slot. A secondary conflict occurs when a node, an intended receiver of a particular transmission, is also within the transmission range of another transmission intended for other nodes. Third, careful reuse of spatial wireless channel (i.e., more than one node can transmit at the same time slot) can help shorten the packet delivery latency, but interference constraints must be satisfied.

2. Problem Formulation

We consider wireless sensor networks with a single sink and multiple homogeneous data sources for applications that require raw data periodically. In this scenario, each sensor node periodically produces a new value and this value may need to traverse multiple hops to reach the sink. The reading from a node can be combined with the readings from other nodes on its way to the sink. Given a set of sensor values that are generated periodically, our objective is to schedule all the transmissions for each period to be completed in the shortest possible amount of time. Ideally, all the non-interfering transmissions can be scheduled at the same time slot to minimize overall delay.

Tree-based collection has been typically used in these applications. If the same routing tree topology is maintained, at each period every sensor node sends the same number of readings upstream to the sink, whether generated at the node or relayed for one of its child nodes.

Previous studies have found that sensor network topology is often semi-static and does not undergo very frequent changes.Therefore, a pre-determined transmission schedule is desirable, especially in short-term event-driven applications (typically on the order of hours or days) that we are targeting. A pre-determined transmission schedule not

0-7695-3113-X 2008
DOI 10.1109/PERCOM.2008.46

only implicitly avoids network congestion that are often caused by bursty data traffic, but also eliminates packet collisions. Both network congestion and packet collisions lead to packet drop and retransmission, thereby packet latency. Latency introduced by link or node failures will be considered in our future work. Clock is assumed to be synchronized among all nodes.

Batch Transmission: In many WSN applications, a sensor reading can often be represented with a small number of bytes, so more than one reading can fit into a standard transmission packet. We exploit this property to reduce the number of packets transmitted. Instead of individually sending each sensor reading, the readings are batched or combined at intermediate nodes and forwarded upstream along the tree. We refer to this as 'batch processing'. This processing differs from 'aggregated processing' where one single value is computed over several sensor readings based on application semantics. In batch processing, each raw sensor report is still maintained in the packets. The maximum number of readings that can be combined depends on the size of a sensor reading and the packet size limitations of the platform.

However, batch transmission processing creates additional precedence constraints when determining a transmission schedule. If m is used to indicate the maximum number of readings that one packet can have, in order to maximize energy savings from using batch transmission, the number of "saturated" packets that have m readings has to be maximized. To achieve that, unless a node is a leaf or the number of its descendant nodes is multiple of m, it should transmit only after receiving a packet from one of its children and combining its own reading with the existing payload.

Problem Statement: The problem can be stated formally as follows. Given a network represented by a graph $G = (V, E)$, where V is the set of nodes including the sink s, and E is the set of links that nodes use to transmit their readings. $n = |V|$ is the number of nodes in G. The routes of all nodes form a collection tree. All traffic is destined for the sink, so every data packet at a node is forwarded to the node's parent in the tree rooted at the sink s in multiple hops. A packet can consist of readings from m nodes. The problem, therefore, is to determine the smallest length conflict-free assignment of time slots during which the reading generated at each node may be combined with readings from other nodes and transmitted to the sink over the routing tree. In other words, we need to schedule the edges (i.e., transmission links) in E.

3. A Transmission Scheduling Algorithm

Before presenting the details of TIGRA - the transmission scheduling algorithm, we first explain various terms used in the transmission schedule (Figure 1). Node v_i calculates its round k_i based on the number of its descendants d_i: $k_i = d_i \mod m$.

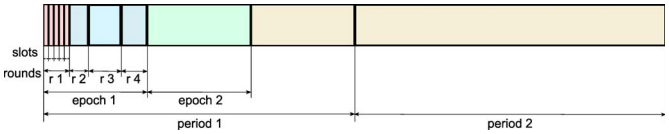

Figure 1. TIGRA timing hierarchy

The set of m rounds will be repeated multiple times until all the packets reach the sink. We refer to a set of m rounds as an *epoch*. In other words, each epoch consists of m rounds. The farther the nodes are from the sink, the fewer epochs they actively participate in.

A *Period* is specified by an application to indicate the desired frequency of data collection. Application requested period is expected to be longer than the length of the combined epochs necessary to transmit all the packets to the sink. If this is not the case, TIGRA informs the application of the best possible delay it can provide.

Each round is scheduled sequentially; therefore, interference and spatial channel reuse are only possible within the same round. To avoid interference and improve spatial channel reuse, each round is divided into a number of *slots*. The number of slots required can vary from one round to another; therefore, forcing each round to last equal time would introduce unnecessary delay. The length of each round is determined by the number of slots that are necessary to avoid conflicts in a given round.

TIGRA consists of three distinct phases (Figure 2): Round Determination, Slot Determination, and Data Collection.

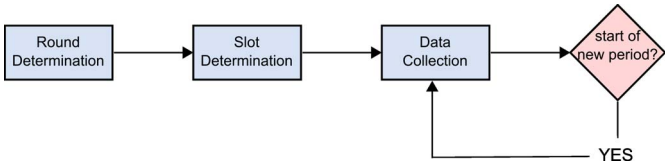

Figure 2. Three phases of TIGRA

Round Determination Phase: In this phase, a routing tree is built and each node v_i determines the number of its descendants d_i and its round k_i, the set of its children ch_i, its interference set I_i consisting of nodes not directly related to it but within one-hop communication range, the number of descendants d_j, and the round k_j for each child $v_j \in ch_i$. A node needs the information about the number of descendants for itself and its children to decide at which epoch it has to participate and at which it does not. The interference set discovered in this phase is not complete and will be further extended in the next phase.

Slot Determination Phase: This phase schedules the

transmissions within the same round, such that interference is eliminated and channel reuse is maximized. At the end of this phase, the nodes report the results of negotiation to the sink. Starting from the leaves, each node sends a packet with the maximum number of slots used for each round. If a node does not participate in a current round, the number of slots for such round is 0. Upon determining the maximum number of slots for each round, the sink floods the network with one more message that contains start times for each round relative to the start of each epoch. The start of each slot can be determined at the nodes locally since it is of standard size and each node knows the exact slots during which it will be transmitting or receiving. Details of this phase are presented in the next Section.

Data Collection Phase: In this phase, each node knows which epoch/round/slot it should transmit at based on the schedule determined by the previous two phases. Nodes send their reports to the sink periodically. as requested by the application. When a period starts, the first transmission round of the first epoch kicks off. Each node is responsible for setting the timers for the slots it participates in and being awake when it serves as a child or parent. Nodes go to "sleep" on their "off" slots or rounds. Upon receiving the report from a child, a node parses the packet, appends its own reading if the packet is not saturated and its own reading is not transmitted yet, and sends the packet upstream.

4. A Distributed Graph Coloring Algorithm for Slot Determination

Since precedence constraints are addressed by the Round Determination phase, the goal of the Slot Determination phase is to decide slot numbers of all transmissions within the same round, trying to maximize channel reuse while eliminating interference. If we add one link between all the node pairs that may potentially conflict to the original transmission graph G, then the slot determination problem becomes the scheduling of nodes in the new graph such that no two adjacent nodes have the same time slot. This is related to the NP-hard vertex coloring problem.

However, existing distributed graph coloring algorithms cannot be directly applied due to the following reasons. Although the data collection graph (or tree) is generated initially, the interference set needs to be dynamically determined in a decentralized manner given that we do not have location information of each node. This implies that the graph to be colored is not fully established before coloring begins and many links related to schedule conflicts need to be gradually discovered during coloring. In addition, wireless links are asymmetric, leading to directed graph. We hence design a new distributed graph coloring heuristic specifically appropriate for our application scenario.

Color Palette and Order of Coloring: One of the fundamental questions is the number of colors used in the coloring. The length of each round is determined by the number of colors used to color the vertices belonging to that round. The number of colors is determined by the amount of interference between the nodes in the round. Each node only transmits (as a child) during one of the rounds but can potentially be receiving transmissions (as a parent) from its children in any other round, therefore, each node has to actively participate in its own coloring as well as coloring for all of its children. There is no concern about interference between transmissions that are scheduled in different rounds since different rounds are scheduled sequentially; therefore, each node can maintain separate palette of available colors for each round that it participates in either as a sender or as a receiver.

The colors are represented by integers corresponding to a time slot assignment within that round. The number of colors in a palette is not predetermined, but new colors are only added when necessary. If all the transmissions were interfering with each other, each node would need a separate slot to transmit and the number of colors across all the palettes would be equal to the number of nodes n. In order to minimize the number of colors, the nodes always try to get the lowest available integer from their palette. As colors become unavailable when nodes overhear other nodes in the same round using them, those colors get deleted and the lowest available remaining color becomes the next candidate. As a result of this color palette mechanism, the coloring with a minimal number of colors will be produced.

A top-down coloring approach is more efficient in our application scenario with a tree-based collection structure. In TIGRA, a parent node assigns different colors to each of its children; as a result, only conflicts between non-related pairs of nodes (i.e., nodes with different parents) have to be resolved.

Details of the Algorithm: In the Slot Determination phase, each node v_i negotiates its color c_i with its parent p_i in its round k_i. Node v_i also assigns and negotiates color c_j at round k_j for each of its children v_j in ch_i. As a result, each node will get a time slot within its respective round k_i. The conflicts and additional interference not in the current interference set are detected by snooping and by specifically inquiring from nodes in interference set I_i about their colors if snooping did not succeed.

A node adds new colors to the palette only when needed as existing colors are deleted. This is ensured by constraining a node to always use the next available color in negotiation. For any palette at each node, the color can be either available or unavailable (deleted) at any given time. It is possible for a color to go from unavailable to available. If the color cannot be confirmed because of the conflict at one of the nodes, the other node makes that color available in its palette again - to avoid false blocking problem.

The following heuristic runs in a distributed manner at each node v_i. A node has three high-level states: *Listening*, *AsParent*, *AsChild*. Based on the Round Determination phase, each node knows at which round it should negotiate as either a child or a parent, the length of the negotiation round, and the start time of the Slot Determination phase. A node sets up timers for the beginning of each round it will participate in. When the timer expires, the node negotiates either as a child or as a parent.

When a node serves as a child, it transitions among 5 different states. It waits for its parent to assign a color to it. Upon receiving a suggested color, it negotiates based on whether the color is available in its palette for this round.

When a node serves as a parent, it may negotiate with more than one child node at the same time. A parent node starts with assigning the first available color to each child and broadcasting the assignments. Child nodes as well as any neighboring nodes receive the message. This allows for any interfering node that has a conflict to send a conflict message. The parent node then negotiates with each child individually based on the availability of the colors in both parent's and child's palettes.

During the negotiation either as a parent or a child, a color only gets finalized after two confirmation messages are broadcast - one from the child and the other from the parent. This ensures that all the nodes in both neighborhoods have a chance to learn about the choice, and potentially interfering nodes in either neighborhood can send a conflict message to the pair of negotiating nodes.

Theorem 1. *TIGRA will always provide an interference-free transmission schedule.*

Theorem 2. *TIGRA will eventually terminate.*

5. Performance Evaluation

The objective of the performance study is to validate our proposed algorithm TIGRA, evaluate and compare its performance against existing algorithms under different network settings using simulation.

Simulation Settings: Our empirical studies thoroughly compares the performance of TIGRA against two existing protocols by varying different parameters: network topology (grid or random deployment), network size, and node density. As a baseline comparison, we compare TIGRA to a basic data collection protocol, referred to as BASIC, which first builds a routing tree and then collects data from each node. No mechanisms above the MAC layer are developed to support timeless. However, up to five retransmissions are allowed for messages that do not successfully reach the next hop. In addition, we compare TIGRA to SPEED [1]. We chose SPEED for comparison because SPEED is a real-time protocol designed to minimize deadline miss ratio in sensor networks and it is compatible with most existing best-effort MAC protocols, which is consistent with TIGRA.

All three protocols are implemented in TOSSIM 2.0The default TOSSIM MAC protocol is CSMA. Each simulation with the same parameter settings is run 20 times. All results also show 95% confidence intervals. Performance metrics of interest are data collection latency, overhead of collection, and packet delivery ratio. The collection latency is the delay of the last packet received by the sink in each period, i.e., the overall latency of collecting all sensor readings.

Simulation Results: The performance of BASIC, SPEED, TIGRA is tested under both grid and random network deployment. We observe that the performance trend and comparison under random network topology is very similar to that under grid topology, so the following discussion is focused on grid topology.

Impact of network size: As expected, latency for both scheduling and collection phases in TIGRA increases as the network size increases (Figure 3 (left)). Latency for both SPEED and BASIC algorithms increases to a certain point and then flattens out or decreases. Figure 3 (middle) shows that TIGRA uses fewer number of messages than SPEED or BASIC while still providing significantly higher packet delivery ratio (Figure 3 (right)). This is due to batch transmission and the fact that retransmissions are minimized if not eliminated by TIGRA.

While the latency comparison above appears pessimistic for TIGRA, the results shown in Figure 3 (right) reveals the real reason behind this. Although we do not consider node or link failures here, Both SPEED and BASIC rely on CSMA-based MAC layer to avoid interference which does not scale well, they lose many packets due to interference and the packet delivery ratio decreases significantly as the network size increases. TIGRA , on the other hand, negotiates a precise schedule for each node's transmission and explicitly avoids interference. This is better demonstrated in Figure 4, which shows cumulative sensor reading delivery ratio as collection time progresses for a constant network size and density. Given enough time, TIGRA achieves 100% packet delivery ratio, whereas BASIC and SPEED deliver a certain number of sensor readings in the early stage of the collection period, after which they do not deliver any more readings to the sink.

Impact of network density: Figure 5 (left) shows how the network density affects the latency. As network density increases, so does the amount of interference. For a network with lower density, TIGRA results in a taller tree with higher number of rounds compared with a denser network which leads to a flatter tree with more slots per round. Hence, TIGRA's scheduling and collection delay stay relatively constant as the network density changes. We also observe that latency of BASIC and SPEED is lower than that of TIGRA. Figure 5 (middle) shows that TIGRA incurs less

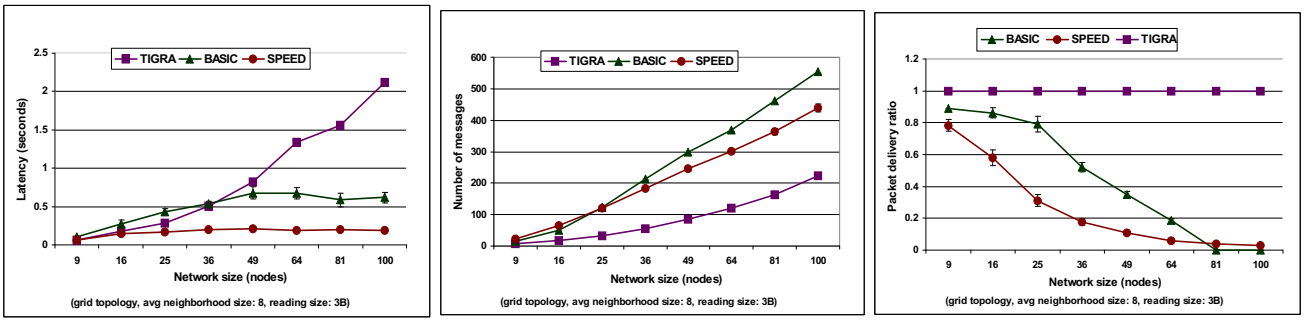

Figure 3. Impact of network size

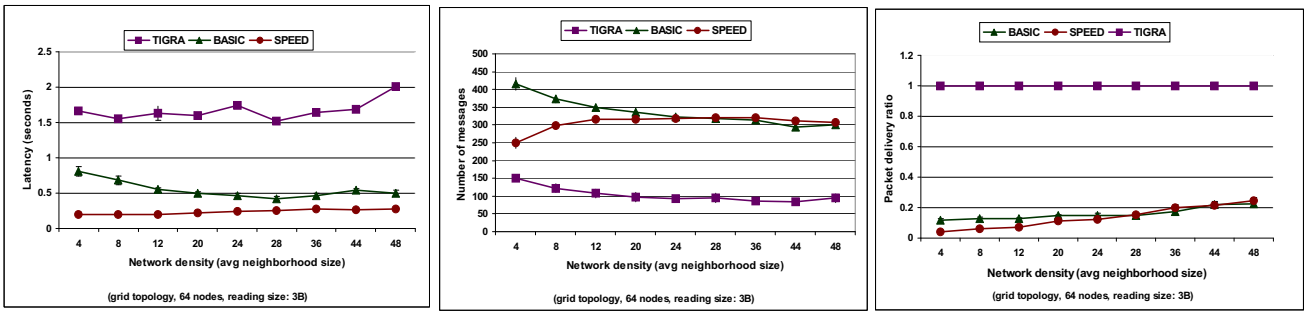

Figure 5. Impact of node density

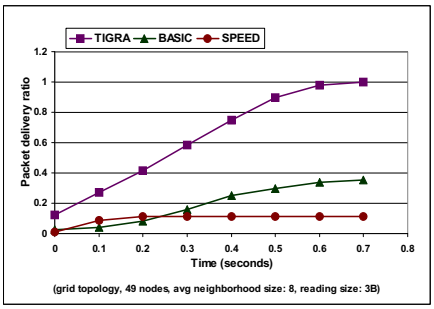

Figure 4. Cumulative packet delivery ratio

collection overhead than SPEED or BASIC, at the price of occasional scheduling. Message overhead of the collection stage decreases with higher density due to lower height of the collection tree. The higher collection latency of TIGRA can be explained by results in Figure 5 (right).

Conclusions Existing techniques for supporting real-time communication in WSNs cannot be directly applied to our problem at hand due to either different assumptions [2, 3] or different objectives [1, 4]. TIGRA can eliminate packet collisions and avoid network congestion, two major factors for latency. Other causes for latency include node failures due to battery depletion or environmental influence and link failures due to external objects and conditions. Since recovery from these faults typically involve retransmission, thereby packet delivery latency. An interesting direction for future extension of TIGRA is adding fault tolerance to data collection while trying to decrease end-to-end latency.

References

[1] T. He, J. A. Stankovic, C. Lu, and T. F. Abdelzaher. Speed: A stateless protocol for real-time communication in sensor networks. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS) 2003*, 2002.

[2] H. Li, P. Shenoy, and K. Ramamritham. Scheduling communication in real-time sensor applications. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2004.

[3] H. Li, P. Shenoy, and K. Ramamritham. Scheduling messages with deadlines in multi-hop real-time sensor networks. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2005.

[4] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002.

[5] R. Ramaswami and K. Parhi. Disributed scheduling of broadcasts in a radio network. In *Proceedings of the IEEE Infocom*, 1999.