Masters requirement fulfilled!

05/14/2003

Ugur Cetintemel

[illegible]

Energy-Efficient Dynamic Data Scheduling in Wireless Multihop Networks

Andrew Flinders
Comprehensive Research Project Summary
Department of Computer Science
Brown University, Providence, RI 02912

Sponsored by Professor Ugur Cetintemel
{awf,ugur}@cs.brown.edu

May 14, 2003

Abstract

This paper presents a power-saving data dissemination model for multihop, wireless networks through the use of an event-based broadcast schedule. The network is arranged as a publish / subscribe tree, with nodes subscribing to the data event-types that they are interested in. A broadcast schedule multiplexes upstream and downstream time-slots for each event-type across its dynamic allocation. Power consumption among wireless nodes is reduced by allowing each node to power down its radio during the portions of the schedule that do not match its particular event subscription. The event schedule is determined in both a centralized and distributed fashion, and is highly dynamic to suit the changing rates at which events are generated and distributed through the network.

1 Introduction

Designing energy-efficient wireless devices is a research goal of critical importance for a variety of networking domains, including sensor networks [23] and ad hoc mobile networks [24]. Today standalone, untethered network nodes are constrained by two severely limiting factors: (1) radio frequency communications are very costly (compared to other electrical hardware functions for computing nodes), and (2) low-capacity battery stores are oftentimes the only available sources of energy for such wireless devices.

At the physical layer (of the OSI network model), efforts to decrease the power-consumption levels of wireless devices have included utilizing lower-power radio propagation techniques, such as code division spread spectrum (e.g., DSSS and FHSS [22]). Also, digital data compression techniques [6] can cut down considerably on the sheer amount of raw data transmitted. At the datalink layer, medium access control (MAC) protocols govern when and how the nodes of the network coordinate to share a broadcast channel effectively. CSMA MAC protocols [2, 17, 9] are generally wasteful inasmuch as nodes must constantly monitor the broadcast channel in Rx mode – an expensive radio function. Other protocols, such as TDMA protocols [14, 16], have been designed to allow nodes to power down their antennas during particular time-slots of a predetermined schedule.

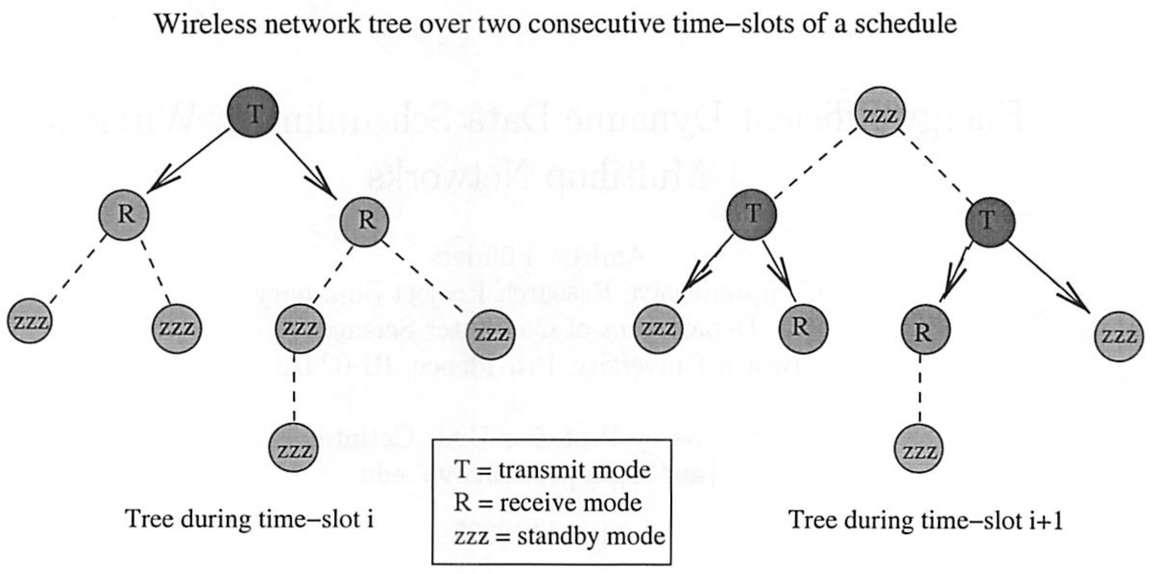
Wireless network tree over two consecutive time–slots of a schedule
T
R
zzz
Tree during time–slot i
T = transmit mode
R = receive mode
zzz = standby mode
Tree during time–slot i+1

Figure 1: Wireless network tree over two consecutive time-slots of a schedule. Each node is labeled with a red T or a green R (according to the schedule) indicating whether or not it transmits or receives data during the particular time-slot. Those nodes which are not interested in particular data items that their parents are transmitting are marked by a blue zzz (for sleep, or standby, mode).

This paper presents a hybrid approach, one that weaves together the division of the shared wireless medium access with the application-determined *semantics* of the data to be broadcast. The scheme differs fundamentally with other access protocols through its temporal allocation of medium access to specific *data-types*, rather than to specific network nodes. The latter approach is used by many infrastructure protocols, such as EC-MAC [14], Bluetooth [16], and many cellular protocols. The approach presented leverages the ability of a wireless network to broadcast data to all interested parties, as opposed to sending data redundantly as multiple unicasts. Furthermore, nodes only need to listen to the wireless channel during the transmission times of data items which *interest* them – at other times, they are allowed to save energy by powering down their antennas. For wide applicability, the system is designed for both infrastructure as well as multi-hop, ad hoc wireless networks to support a variety of wireless topologies and applications.

The protocol presented organizes a wireless network into a multi-hop network tree. The root of the tree creates a data dissemination schedule and propagates this schedule throughout the tree, so that all nodes may adhere to it. The schedule is divided into time-slots, and each time-slot indicates which type of data can be sent (or received), and whether it is for downstream (away from the root) or upstream (toward the root) communication. Figure 1 displays a small network tree during two consecutive time-slots. Each node is labeled with a red T or a green R (according to the schedule) indicating whether or not it transmits or receives data during the particular time-slot. Those nodes which are not interested in particular data items that their parents are transmitting are marked by a zzz (for sleep, or standby, mode).

The idea behind the system is that nodes can save energy by powering down their radios to standby mode when they have no data to send, and when they do not wish to receive the data being transmitted (whose *type* semantics are indicated by the schedule). Each node has a specific subscription profile, indicating which data types it does and does not wish to receive. In a traditional wireless network, nodes must listen promiscuously to the wireless channel for all data being transmitted, lest they might miss something important. TD-DES allows nodes to selectively listen for data which interests them and save energy otherwise.

Because data must be scheduled before it is sent, the main tradeoff of the system is increased power efficiency in exchange for sub-optimal message delivery latency. This paper outlines this novel wireless networking paradigm and protocol in detail, presents experimental simulation results of the system to demonstrate its effectiveness, describes previous and current related research that has inspired and contributed to this project, and outlines goals for future testing, implementation, extension and development of the system.

2 TD-DES Protocol

This paper presents a data event scheduling protocol, which is called TD-DES (short for Time Division Dynamic Event Scheduling). It is intended as a datalink/network layer overlay to a CSMA/CA wireless MAC layer (such as 802.11 MAC DCF [2], MACA [17] and FAMA [9]), rather than a MAC layer in itself. TD-DES is considered part of the datalink layer as it governs shared medium access among contending network nodes. But it is also considered part of the network layer inasmuch as such access is decided by the type of data being broadcast, which in turn dictates *which* sets of nodes are exchanging data at a particular moment in time (based on the network nodes' *event subscriptions*).

The protocol's main function is to govern when each node of a network receives data, transmits data, and powers its radio down to a low-power standby mode. These *radio modes* – Tx, Rx, and standby – are cycled among as functions of time determined by the network's *event schedule*, which is generated primarily by the root node and propagated down the tree as part of a downstream *control event*. By minimizing the amount of time spent in Rx and Tx modes and maximizing the time spent in standby mode (while not sacrificing the reliable and timely dissemination of data events), the power consumption of the system can be dramatically reduced.

TD-DES also provides an integrated network construction layer, which organizes a subject wireless network into a shortest-path tree topology. The layer is adaptive to topological changes that result from node failures, additions and mobility. The layer does not provide specific source-node-to-destination-node messaging capabilities (for example, using globally unique id's such as IP addresses) as do traditional routing protocols [13, 7, 20], but rather provides a means of disseminating data events throughout the network to all interested parties.

Tree-based wireless topologies have become popular in the recent past [10, 21] mainly because they require much less control information to be transmitted between nodes than do networks which use traditional routing protocols such as DSR [13], DSDV [7], and AODV [20]. In addition, the amount of state that needs to be maintained at each node is soft by comparison (as nodes must keep track only of neighboring parent and child nodes).

For many wireless applications, particularly sensor network applications, comprehensive node-to-node routing information is not always needed. For example, a sensor network may simply require that all the wireless sensor motes forward sensor samples up the network tree toward the *root* node, which may be a centralized network *sink* with greater computational, storage, and transmission capabilities than the rest of the nodes in the network. Such is the sensor network paradigm presented by TAG [21], directed difusion[12] and other recent research efforts.

2.1 Data events and application-defined QoS parameters

By an event, we mean a particular type of data message with its own distinguishing, application-specific semantics. For example, suppose we have a sensor network deployed over a forested region whose purpose is to detect forest fires. A sensor node might issue a **fire_detected** event to the network if its thermal sensor registered a very high temperature reading. The event would be disseminated through the network to all those nodes subscribing to **fire_detected** events. These could include nearby forest ranger stations, a centralized forest fire monitoring station, or a sink node which could notify the police, local fire-fighting units, and public news services. These would also include any intermediate nodes which had to *forward* such events to interested nodes, even if they themselves were not interested per se. A different type of event could be a **low_battery** event, which would only be disseminated to a network maintenance facility, alerting personnel that a particular sensor node would soon fail if its battery was not quickly replaced. Another type of event could alert the same facility that the sensor equipment on a node was not functioning properly. Any number of event-types could be assigned to the system, depending upon the needs of the application.

Besides carrying distinguishing type semantics such as these (with which individual network nodes can decide whether or not to include them in their subscriptions), event-types may also be associated with network-specific physical characteristics, such as minimum and maximum event payload sizes (in bytes), latency constraints (e.g., maximum allowable propagation delay), and relative event priorities. TD-DES allows such event latency and priority values to be specified as quality of service (QoS) parameters input by the overlying application(s). Such parameters affect the ordering and eviction policies of the event *scheduling algorithm* (Section 4). However, for maximum interoperability with existing applications, the system also works in the absense of such parameters.

2.2 Publish/Subscribe tree network model

The TD-DES networking component organizes a subject wireless network into a shortest-path, publish/subscribe tree, where events of certain types are generated and disseminated to all interested parties. Event-types are enumerated by the system, so that a network with n different event types may publish event types $e_1, e_2, ..., e_n$.

The network overlay constructs and maintains this network tree, and also allows the tree topology to change dynamically in the face of node additions, failures, and mobility. The tree is shortest-path inasmuch as each node chooses, for its parent, the node which is closest to the root (as advertised by each parent node's downstream *control event*).

Each node maintains its own *event subscription*, which is the set of event-types that a node is interested in receiving. In addition, each node maintains its own *effective subscription*, which is the union of its own subscription and the subscriptions of all of its descendents. Each node must effectively subscribe to any event-type that it itself is interested in as well as any event-type that a descendent node is interested in. This is because each node is responsible for *forwarding* all relevant events to its descendants. We borrow this publish/subscribe subscription terminology from [10].

See Figure 2 for an example network tree of seven nodes and three event types e_1, e_2, and e_3. N_1 is the root node of the tree. The subscription of each node is given in parentheses to the upper left of the node. The effective subscription of each node is given to the upper right of each node in square brackets. Figure 3 shows the same tree, where an event of type e_2 is

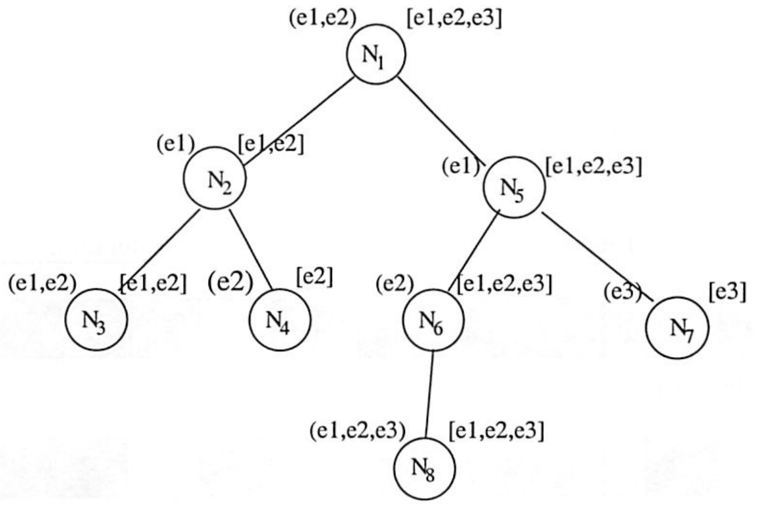

Figure 2: Publish/subscribe network tree of seven nodes $\{N_1, N_2, ..., N_6\}$ and three event types $\{e_1, e_2, \text{ and } e_3\}$. Subscriptions are given at the upper left of each node, effective subscriptions at the upper right.

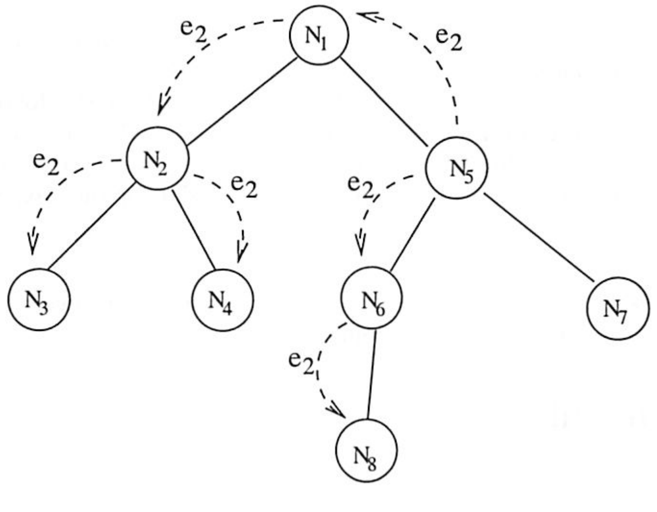

Figure 3: In the tree from Figure 2, an event of type e_2 is generated at node N_5 and disseminated throughout the tree. Arrows indicate the links over which the event is broadcast. Since N_7 is the only node which doesn't include e_2 in its effective subscription, it is the only node that doesn't receive the event.

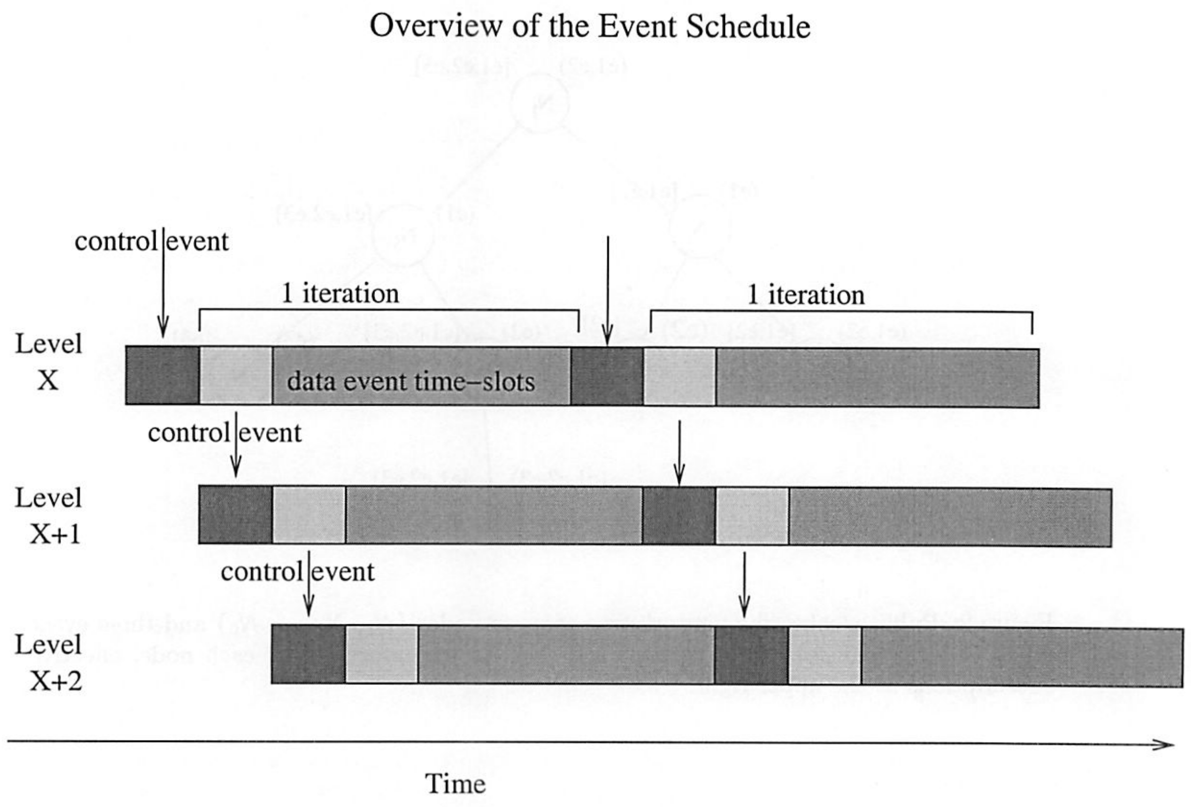

Figure 4: Overview of the event schedule. The control event is received by the node at level X in the first time-slot (marked in red). In the next time-slot (marked yellow), this node transmits the control event down to the next level, X+1. In the time-slot following this, the node at level X+1 passes the control event down to level X+2, and so on and so forth. The control event contains the schedule of data event time-slots in the current iteration, which are marked in green, and are followed by a receive control slot for the next iteration, also marked in red.

generated at node N_5. The arrows indicate the links across which the event is broadcast in order to disseminate the event to all subscribing nodes.

3 Event Schedule

3.1 Overview

The TD-DES event schedule determines the temporal partitioning of the RF medium for all of the event-types in the system. It does this by allocating *time-slots* for each event-type. Each time-slot provides roughly enough time for a single event of the corresponding type to be propagated one hop. However, a time-slot is wide enough to accomodate multiple transmissions in the case of contention (which is assumed to be handled by the underlying MAC layer). For our own implementation of TD-DES we used an 802.11 CSMA/CA MAC [2] layer with CTS / RTS (to effectively eliminate collisions) utilizing a randomized backoff policy to handle medium contention. A single event in

Tree	Time-slot #											
Level	0	1	2	3	4	5	6	7	8	9	10	11
0	r_c	s_c	Dr_1	Ds_1	Ur	Us	r_c	s_c	Dr_1	Ds_1	Ur	Us
1	*	r_c	s_c	Dr_1	Us	Ur	Ds_1	r_c	s_c	Dr_1	Us	Ur
2	*	*	r_c	s_c	Ur	Us	Dr_1	Ds_1	r_c	s_c	Ur	Us
3	*	*	*	r_c	Us	Ur	s_c	Dr_1	Ds_1	r_c	Us	Ur
4	*	*	*	*	*	*	r_c	s_c	Dr_1	Ds_1	Ur	Us

Tree	Time-slot #											
Level	12	13	14	15	16	17	18	19	20	21	22	23
0	r_c	s_c	Dr_1	Ds_1	Ur	Us	r_c	s_c	Dr_1	Ds_1	Ur	Us
1	Ds_1	r_c	s_c	Dr_1	Us	Ur	Ds_1	r_c	s_c	Dr_1	Us	Ur
2	Dr_1	Ds_1	r_c	s_c	Ur	Us	Dr_1	Ds_1	r_c	s_c	Ur	Us
3	s_c	Dr_1	Ds_1	r_c	Us	Ur	s_c	Dr_1	Ds_1	r_c	Us	Ur
4	r_c	s_c	Dr_1	Ds_1	Ur	Us	r_c	s_c	Dr_1	Ds_1	Ur	Us

Table 1: Simple downstream / upstream interleaved event schedule for a single event-type, e_1.

our system is encapsulated in one MAC frame. Events could also be longer than a MAC frame, however, it would be the responsibility of a higher network layer to perform fragmentation and reassembly.

Time-slots are allocated for each event based on the *determined* and *expected* bandwidth requirements needed to propagate all generated events reliably thoughout the network (see Section 4.1 on deterministic and speculative scheduling). Once the numbers of upstream and downstream time-slots for each event-type are determined, the ordering of the time-slots must then be determined. Ordering is done based on several criteria, including the *popularity* of each event-type (i.e., the number of nodes subscribing), the latency QoS constraints for events, as well as the application-defined priorities of events. In some cases, the number of generated events may exceed the storage or bandwidth capacities of the system, in which case some events may be dropped according to an eviction policy utilizing some of these criteria.

The principle idea is that the root node creates a schedule of time-slots. Each time-slot is designated as a send or receive slot, whether it is for upstream or downstream communication, and by the event-type which it should be used to propagate. The root creates the schedule one portion at a time – called an iteration – and passes it down through the tree inside a control event. Figure 4 presents the basic idea of creating a schedule and passing it down the levels of the network tree. The control event is received by the node at level X in the first time-slot (marked in red). In the next time-slot (marked yellow), this node transmits the control event down to the next level, X+1. In the time-slot following this, the node at level X+1 passes the control event down to level X+2, and so on and so forth. The control event contains the schedule of data event time-slots in the current iteration, which are marked in green, and are followed by a receive control slot for the next iteration, also marked in red.

The schedule is a sequence of atomic *send* and *receive time-slots*, each one for a specified event-type (including control event-types). Generally, for a particular event in a schedule, time-slots are allocated as a receive/send pair (which is not atomic, but is oftentimes allocated contiguously). At a particular node, if the schedule specifies a receive time-slot for event e_1, the node, if it subscribes to e_1, will listen to the RF medium in Rx mode during this time-slot to receive such an event. If the schedule specifies a send time-slot for e_1, the node can use this time-slot to transmit an event of this type (should it have one to send).

Furthermore, each slot is distinguished as either a *downstream* slot (for parent-to-child communication away from the root) or as an *upstream* slot (for child-to-parent communication toward the root). For downstream communication, a *send* slot for event-type e_i is denoted in shorthand as Ds_i

Tree	Time-slot #											
Level	0	1	2	3	4	5	6	7	8	9	10	11
0	r_c	s_c	Dr_1	Ds_1	Dr_2	Ds_2	Ur	Us	Ur	Us	r_c	s_r
1	*	r_c	s_c	Dr_1	Ds_1	Dr_2	Us	Ur	Us	Ur	Ds_2	r_c
2	*	*	r_c	s_c	Dr_1	Ds_1	Ur	Us	Ur	Us	Dr_2	Ds_2
3	*	*	*	r_c	s_c	Dr_1	Us	Ur	Us	Ur	Ds_1	Dr_2
4	*	*	*	*	r_c	s_c	Ur	Us	Ur	Us	Dr_1	Ds_1

Tree	Time-slot #											
Level	12	13	14	15	16	17	18	19	20	21	22	23
0	Dr_1	Ds_1	Dr_2	Ds_2	Ur	Us	Ur	Us	r_c	s_r	Dr_1	Ds_1
1	s_r	Dr_1	Ds_1	Dr_2	Us	Ur	Us	Ur	Ds_2	r_c	s_r	Dr_1
2	r_c	s_c	Dr_1	Ds_1	Ur	Us	Ur	Us	Dr_2	Ds_2	r_c	s_r
3	Ds_2	r_c	s_c	Dr_1	Us	Ur	Us	Ur	Ds_1	Dr_2	Ds_2	r_c
4	Dr_2	Ds_2	r_c	s_c	Ur	Us	Ur	Us	Dr_1	Ds_1	Dr_2	Ds_2

Table 2: Clustered downstream / upstream event schedule for two event-types, e_1 and e_2.

and a *receive* slot for the same event type as Dr_i. The equivalent upstream time slots are denoted Us and Ur, respectively. Upstream slots are not designated for event-types, as they are speculatively allocated – any event which is generated should be able to make use of the next upstream slot. In addition, nodes must always listen to upstream receive slots, as all events must be passed up to the root, regardless of event-type. Therefore, distinguishing upstream slots for particular event-types would serve no beneficial purpose.

Send and receive *control events*, which are almost always downstream (upsream control slots are explained later), are respectively denoted as s_c and r_c. The schedule of time-slots between two consecutive downstream control events is called a single *iteration* of the schedule. Upstream send and receive control events, when they occur, are denoted as Us_c and Ur_c, respectively. See Table 1 for a simple example schedule.

At each level of the tree, the schedule is divided into time-slots for downstream *control events* and for upstream and downstream send and receive slots for e_1, the only event-type in this example. The downstream control event, s_c, which is broadcast by each internal node to its children, indicates the schedule for the current iteration (terminated by a time-slot for the next downstream control event). Notice that the next iteration (after the second s_c) repeats the same schedule.

The downstream control event, s_c, also includes data used by the tree construction protocol, such as the number of hops to the root and the parent node's network-unique identifier. Notice that for each downstream *send* event, the simultaneous time-slot at the next level down is a *receive* time-slot for the same type of event. Similarly, for upstream *send* events, the concurrent time-slot at the next level up the tree is a corresponding *receive* time-slot for the same type of event.

Notice also how the downstream propagation of events in the schedule is *tightly pipelined.* That is, events are sent down the tree in a series of temporally contiguous time-slots to minimize the propagation latency. Additionally, the root schedule (level 0) interleaves each downstream receive/send pair of time-slots with an upstream receive/send pair. We call this an *interleaved* schedule. The interleaving changes somewhat as we move down the tree, as columns for downstream and upstream events are fixed while the root sequence is shifted. It is important to understand that a column is either completely downstream or completely upstream. A schedule cannot have an upstream time-slot at one level of the tree concurrent with a downstream time-slot at another level. Because of this, the smooth downstream pipelining is interrupted by the upstream time-slots (and vice-versa).

Tree	Time-slot #																							
Level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	r_c	s_c	Dr_1	Ds_1	Dr_1	Ds_1	Ur	Us	Cru_1	Csu_1	Cru_2	Csu_2	Cru_3	Cru_3	Crj_1	Csj_1	Crj_2	Csj_2	r_c	s_c	$*$	$*$	$*$	$*$
1	$*$	r_c	s_c	Dr_1	Ds_1	Dr_1	Us	Cru_1	Csu_1	Cru_2	Csu_2	Cru_3	Cru_3	Crj_1	Csj_1	Crj_2	Csj_2	Ur	Ds	r_c	s_c	$*$	$*$	$*$
2	$*$	$*$	r_c	s_c	Dr_1	Ds_1	Cru_1	Csu_1	Cru_2	Csu_2	Cru_3	Csu_3	Crj_1	Csj_1	Crj_2	Csj_2	Ur	Us	Dr_1	Ds_1	r_c	s_c	$*$	$*$
3	$*$	$*$	$*$	r_c	s_c	Dr_1	Csu_1	Cru_2	Csu_2	Cru_3	Csu_3	Crj_1	Csj_1	Crj_2	Csj_2	Ur	Us	Cru_1	Ds_1	Dr_1	Ds_1	r_c	s_c	$*$
4	$*$	$*$	$*$	$*$	r_c	s_c	Cru_2	Csu_2	Cru_3	Csu_3	Crj_1	Csj_1	Crj_2	Csj_2	Ur	Us	Cru_1	Csu_1	Dr_1	Ds_1	Dr_1	Ds_1	r_c	s_c

Table 3: Example of an *upstream iteration*, with time-slots for 3 `child_update` and 2 `child_join` events, as well as up- and downstream time-slots for a single event-type, e_1.

3.2 Clustered versus interleaved schedules

Table 2 illustrates a more complicated schedule, with two event types: e_1 and e_2. At the root, all of the downstream time-slots are clustered together, separately from the upstream time-slots, which are also clustered together. We call this a *clustered* schedule. Whether or not a schedule is interleaved or clustered can have an effect on the expected average and worst-case dissemination latencies of an event (Section 4.3), and this is taken into consideration by the scheduling algorithm (particularly when latency QoS parameters are present in the system).

This example also illustrates how upstream data flow is also pipelined. As we saw in Table 1, to pipeline downstream data, the downstream portion of the schedule is shifted right by one time-slot for every level moved down the tree from the root. However, the downstream slots skip over the upstream columns as they are shifted into the next downstream column.

Similarly, the upstream schedule is pipelined by shifting the schedule *left* one slot for every level moved away from the root. The upstream schedule also skips the downstream columns and is shifted into the next upstream column to the left. Because the schedule alternates these upstream and downstream clusters, the upstream and downstream pipelining is interrupted once per iteration.

The examples in Tables 1 and 2 present a repeating schedule – that is, each successive iteration is the same. However, our system provides for a dynamic scheduling, and each iteration can be dramatically different from the previous one (as specified by the downstream control event at the beginning of each iteration). Section 4 explains how the schedule is determined for each iteration, how it is adjusted dynamically to changing application and network conditions, and how it is algorithmically generated by the root and (potentially) extended as it is passed down the tree.

3.3 Upstream control events

Upstream control information must also be disseminated, from child to parent. A child needs to inform a parent node that it is accepting it as its parent. In addition, information such as the child's event subscription also must be passed to the parent (so that the parent can modify its own effective subscription). For this reason, upstream control event time-slots are allocated by the parent node occasionally – the frequency with which they are allocated is sytem dependent, and depends upon two things:

1. how often nodes should be allowed to connect to a parent

2. how often subscription and other information should be passed up the tree

Table 3 presents an example of a schedule iteration where such upstream control events are allocated. Such an interation is called an *upstream iteration*, and in addition to downstream and upstream time-slots for the regular data events and downstream control events, it contains time-slots for upstream control events. There are two types of upstream control events: a `child_join` event, and a `child_update` event.

During the `child_join` event slot, any disconnected node can send a `child_join` event to accept the advertising node as its parent. Included with the event are the child's effective subscription, network-unique identifier (such as a MAC address), and other information necessary for constructing the tree and executing the scheduling algorithm.

A `child_update` event differs in that there is one `child_update` time-slot allocated for each current child of a particular parent node. Each child must send a `child_update` event during its allocated slot. A `child_update` event contains the same information about the child that a `child_join` event contains, so that each child can update its subscription information to its parent, should it change. It also serves as a heartbeat message from child to parent, letting the parent know that its child is still connected.

Subsequent schedule iterations can be of different length, but any particular iteration must be the same length at every level of the tree. Because each internal node may have a different number of children, the root node will allocate a system-dependent maximum number of `child_update` time-slots during an upstream iteration. This is to accomodate internal nodes which may have a large number of children. Each internal node may schedule the available `child_update` time-slots for its children in any order it chooses, and will not use the extra slots as `child_updates` (but may use them for something else, see Section 4). In addition, during an upstream iteration, the root will allocate an arbitrary number of `child_join` event slots, during which any unjoined node can transmit such an event to choose the node as its parent.

In the example upstream iteration given by Table 3, the root specifies 3 `child_update` time-slots and 2 `child_join` time-slots. Each `child_update` time-slot is denoted Cru_i for a receive slot from child i and Csu_i for a send slot from child i, where i equals 1, 2 or 3. Each child_join time-slot is denoted Crj_i for a receive slot and Csj_i for a send slot, where i equals 1 or 2. In addition, two downstream time-slots and one upstream time-slot are allocated for e_1. The iteration is clustered inasmuch as all of the the downstream slots are scheduled in contiguous blocks (columns 0-5 and 18-23), as are the upstream slots (columns 6-17).

When an occasional upstream iteration is needed, TD-DES typically generates several consecutive upstream iterations (the number being dependent on the depth of the tree) to ensure that all of the upstream control information reaches the root, including that of the farthest leaf. This is required because the schedule iteration's pipelining is disrupted as it is circularly shifted down the tree.

4 Scheduler

4.1 Deterministic and speculative scheduling

The TD-DES scheduling alorithm makes use of two methods of scheduling: deterministic and speculative.

By *deterministic scheduling*, we mean that the scheduler allocates exactly the number of time-slots required for a set of events which have been generated. For example, if, in the space of one iteration, the root generates 3 events of types e_1, e_2, and e_3, then it will specify in the control event (s_c) for the next iteration exactly when it is going to broadcast these events. Aside from listening to its own upstream time-slots, each child of the root need only power up its antenna during the subset of these time-slots which matches its subscription.

To illustrate how events generated at the root are scheduled deterministically, refer to Figure 5. Here is shown a series of two back-to-back iterations of a root schedule. In the first iteration, event e_1 is generated by the root node and added to a queue of events to send during the next iteration.

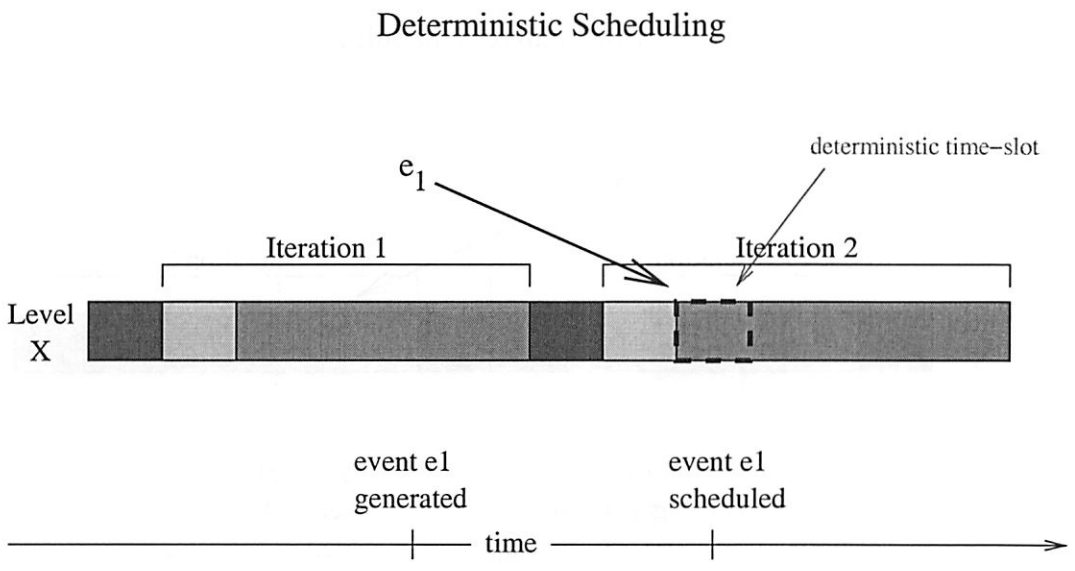

Figure 5: Example of deterministic scheduling

When the next iteration is scheduled, a send slot for that event is allocated. In this way, a send slot has been *deterministically* allocated for a specific generated event.

By *speculative scheduling*, we mean that a fixed number of event time-slots are allocated to each event per iteration. A speculative schedule is determined based on expected event generation frequencies (similar to the multiplexing of asynchronously broadcast data used broadcast disks[3][4]), and is meant to be much less dynamic than a deterministic schedule (which can change every iteration). To see an example of speculative scheduling, refer to Figure 6. Again, two back-to-back schedule iteration at the root are given. However, in this case, a speculative time-slot is allocated at the beginning of the data portion of the schedule in each iteration. In the first iteration, the speculative slot is unused (nothing is propagated). However, towards the end of the first iteration, event e_1 is generated. In this case, rather than allocated a specific slot for the generated event in the next iteration, the event can be scheduled for the already allocated speculative slot.

The obvious downside to speculative scheduling is that many slots are allocated but do not get used to propagate events, while nodes must always listen to speculative receive slots – a waste of radio energy spent in Rx mode. Alternatively, it may be the case that more events are generated than can be propagated because too few time-slots were speculatively allocated to the schedule. If the generating node exceeds its storage capacity before it can propagate all the events, events may be discarded.

The scheduler uses deterministic scheduling for downstream communication (i.e., from the root to the leaves) and speculative scheduling for upstream communication (i.e., toward the root). As mentioned, deterministic scheduling affords the advantage of allowing the network to allocate exactly the amount of bandwidth that is needed to propagate a set of events, but requires that control information indicating the schedule be transmitted downstream frequently. In TD-DES, such a control event is sent from a parent to its children at the beginning of every iteration.

It is not feasible to allow as much control information to flow upstream, as a parent with a large number of children would have to listen to an excessive amount of control information (one event from each child) during every iteration. Control information from child to parent in TD-DES therefore occurs much less frequently, and, rather than containing scheduling information, indicates other information necessary for maintaining the tree topology and calculating event popularities and expected dissemination latencies (see Sections 4.3 - 4.5).

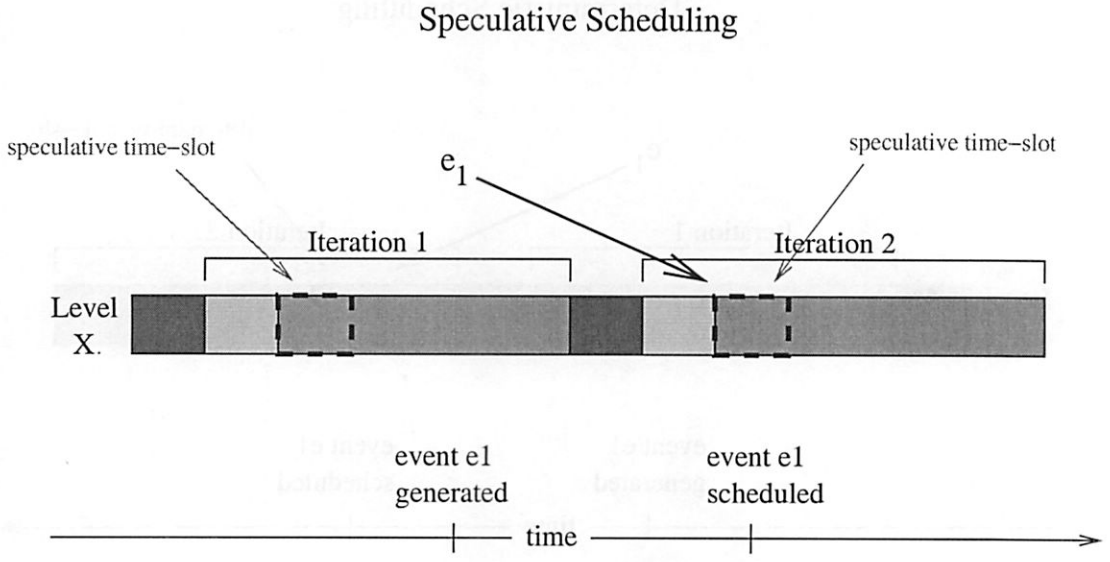

Figure 6: Example of speculative scheduling

The scheduler uses speculative scheduling for upstream communication, to obviate the need for frequent upstream control information. This division – deterministic for downstream, speculative for upstream – makes sense in a tree-based network topology. Far more data is sent downstream than upstream. When an event is generated by the root, it is sent only downstream. When an event is generated elsewhere (see Section 4.2 below), it is sent upstream and downstream, but the upstream communication is a single direct path to the root, whereas downstream communication is used to propagate the event to every branch of the tree. The ratio of the amount of downstream to upstream data sent over time (assuming uniform temporal event generation probabilities at any node in the tree) increases exponentially with the degree of the tree.

4.2 Extending the root schedule

So far we have considered a deterministic schedule created by the root node and passed down the tree. The schedule contains time-slots for the events generated by the root. Typically, such events are scheduled in the iteration following the iteration during which they are generated. In this way, events generated by the root are scheduled deterministically.

But what about events which are generated by internal nodes other than the root? As interal nodes must adhere to the root's schedule, how do they allocate time-slots for their own events? The answer to this is that the root always leaves *blank* time-slots in every iteration. When an internal node (other than the root) generates an event, it simply modifies one of the blank time-slots of the schedule iteration it received from its parent into a downstream time-slot for the appropriate event-type. It then passes this *extended* schedule iteration to its children so they are prepared to receive the newly generated event. In this way, events generated by internal nodes other than the root are deterministically allocated downstream time-slots.

Events generated by non-root internal nodes must also be passed upstream, in order to disseminate the events to all interested nodes in the tree. In wired, routed networks, this is analogous to using a shared multicast tree which, although having a single root node, allows for multiple senders. Such events are passed up to the root in a direct path, and each internal node along this path (after forwarding the event to the root) also propagates the event to the downstream branches of its own subtree.

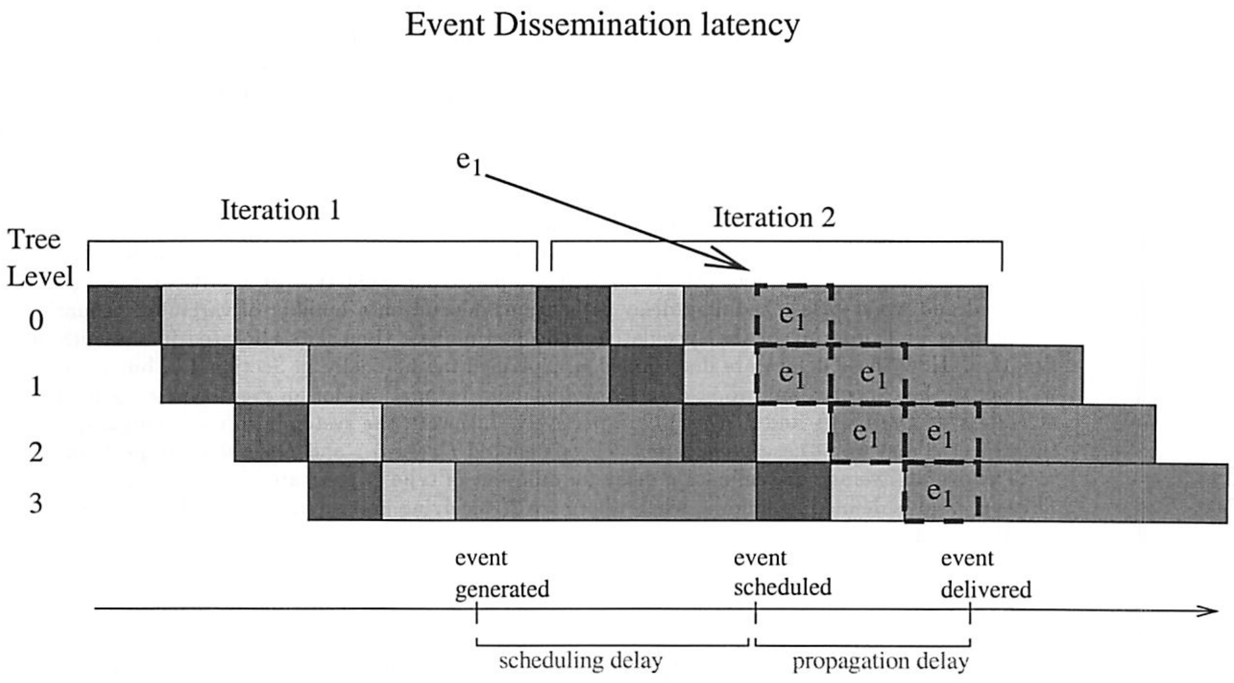

Figure 7: Dissemination Latency: scheduling and propagation delay.

For such upstream event propagations, the internal nodes use the speculative time-slots that the root node has allocated for each iteration. In TD-DES, until this allocation is changed by the scheduler (based on changes in expected event generation frequencies), the number and sequence of these time-slots will be the same in each successive iteration.

4.3 Event dissemination latency

Event dissemination latency is defined as the time between when an event is generated and when it is received by an interested node. For a particular network tree and its event subscriptions, we are concerned with predicting the average and worst-case event dissemination latencies for all types of events.

Dissemination latency for a particular event is divided into two components:

- *Scheduling delay* – the time between when an event is generated by a node and when the node can first propagate the event on the wireless channel.
- *Propagation delay* – the time between when an event is first propagated on the wireless channel by its generating node and when it is received by a subscribing node.

Figure 7 illustrates scheduling and propagation delay for a sample event which is generated. The figure represents a two-iteration schedule for a 4 level network tree. Event e_1 is generated at the end of the first iteration by the root, and is scheduled for propagation during the second time-slot of the second iteration. The interested node is at level 3. The diagram divides the total dissemination delay by showing on a timeline exactly which portion is the scheduling delay and which is the propagation delay.

For an event of type e_i which is generated by node k, its scheduling delay is denoted by Ds_{ik}. The propagation delay of the same event with regard to a particular receiving node, j, is denoted by Dp_{ijk}. Note that the scheduling delay is not affected by the identity of the receiving node, hence the k parameter is only used by the propagation delay (which can vary substantially with k). The total delay for an event of type e_i, being sent from node k to node j, is denoted D_{ikj} and is equal to the sum of both components:

$$D_{ijk} = Ds_{ik} + Dp_{ijk} \tag{1}$$

The TD-DES system is concerned with estimating the average and worst-case dissemination delays at every node for every type of event. It uses these estimations when determining event schedule orderings and evictions (Section 4.5). For a particular event-type at a particular node, the expected and worst-case scheduling delay estimations depend on a number of variables, primarily the number and ordering of the speculative and deterministic time-slots allocated in the current iteration. How these delays are determined is explained methodically in Section 4.5, but for now we refer to the expected and worst-case *scheduling* delay estimations for an event-type e_i generated at node k as $Ds(exp)_{ik}$ and $Ds(wst)_{ik}$, respectively. Likewise, the system also uses estimations of the expected and worst-case *propagation* delays, denoted $Dp(exp)_{ik}$ and $Dp(wst)_{ik}$, respectively.

The overall average dissemination delay for an event of type e_i generated by node k is denoted $D(exp)_{ik}$ and is defined as follows, where there are N receiving nodes (each referred to by variable j in the equation):

$$D(exp)_{ik} = Ds(exp)_{ik} + (1/N)\sum_j Dp(exp)_{ikj} \tag{2}$$

The worst-case dissemination delay for the same event is denoted $D(wst)_{ik}$ and is defined:

$$D(wst)_{ik} = Ds(wst)_{ik} + \max_j(Dp(wst)_{ikj}) \tag{3}$$

In addition, we need to consider the expected and and worst-case scheduling and propagation delays for each event-type e_i regardless of what node is generating the event. These latencies are denoted exactly as they are in the above two equations ((2) and (3)), however the subscript k (for the generating node) is omitted from each term.

4.4 Ordering the events

The root node of the tree is responsible for determining a base event schedule that the entire network tree must adhere to. The schedule must allocate enough bandwidth for all of the events that are generated to be properly disseminated, and must schedule them in a way that observes any application-specific, relative event priorities and which also meets each event's latency constraint (if any exists). In all, the scheduler uses three basic criteria, alone or in conjunction with each other, to determine how to order a set of newly generated events for an upcoming iteration. These criteria are:

- *popularity* - the number of network nodes subscribing to an event-type
- *latency constraint* - the maxmium allowable dissemination delay for an event-type
- *priority* - the relative priority of an event-type over other event-types

Ordering by popularity is the standard method used by the scheduler, and is used in the absense of any application-provided QoS parameters such as latency constraints or priorities. In essense, this method assumes that the event-types that are most subscribed to are considered to be most important by the system, and so they are scheduled first in the upcoming iteration(s). Section 4.5 explains how the tree-construction and maintanance layer of TD-DES gathers the popularity of each event-type (and more generally *all* of the network subscription data).

If Quality of Service (QoS) latency constraints are specified by the application layer for event-types, TD-DES will make use of the average- and worst-case latency dissemination estimates when scheduling events. A node can reduce the overall dissemination latency of an event by scheduling it as early as possible – this reduces the scheduling delay component of the latency. When latency is the primary sorting criterion, the scheduler will attempt to schedule events such that their average and worst-case latencies remain within the given maximum bounds. Latency constraints are specified in millisecond units (ms). A default latency bound (if none is specified) is defined (denoted L_{max}) by the system, and should be fairly long (our implementation uses $L_{max} = 200$ milliseconds). While an event is being disseminated, if the event exceeds its bound (as measured by its *time-of-flight counter* (see Section 4.11), it will considered worthless by a forwarding or receiving node, and will be dropped. The latency constraint for an event-type e_i is denoted Le_i.

Priorities can also be specified by the application-layer for event-types. Priorities are given as integer values between 1 and 20, inclusive. A priority of 1 indicates least priority, and 20 indicates maximum priority. When priority is the primary sorting criterion, events will be scheduled in order of decreasing priority. If an event-type is not assigned a priority, it's default value is 10. The priority of an event-type e_i is denoted Pe_i.

An application can not only specify latency bounds and priorities for event-types as it sees fit, but it also supplies an ordered list of these three criteria in order of importance. That is, if it wants priority to be the primary sorting criterion, followed by latency and then popularity, it should pass the list (*priority, latency, popularity*) to the TD-DES network layer. If no list is sent, the default of the system is (*popularity, latency, priority*). Furthermore, if there are particular criteria that it does not want used at all by the scheduler, it simply omits them from the passed list (e.g., (*popularity*) omits latency and priority). The ordered criteria list is denoted *sort_criteria*.

As in typicial sorting algorithms using multiple criteria, the primary criterion is used to the sort the set of events, with sorting ties broken by a second sorting using the next criterion in the ordered list, and so forth.

4.5 Subscription profiles and latency estimates

Determining the ordering of a set of events, whether it is by popularity, latency, or priority, requires the nodes of the tree to gather data for each event-type. For popularity, each node simply needs to know how many nodes subscribe to each event-type. For latency, nodes need to be aware of the average and maximum number of hops from the root for the set of nodes subscribing to each event-type (hop-distance is used as part of the propagation latency estimate). In addition, the priorities of event-types must also be known by all nodes.

The easiest of these values to disseminate are the priorities of the event-types. These are specified by the application at the root node, and are passed down the tree in the body of the *downstream control event*. If the priorities are relatively fixed, they need only be included in the control event on occasion (as often as new event-types are added or the priorities change).

The number of nodes subscribing to an event-type e_i is called the *count* of e_i and is denoted $count(e_i)$. The number of hops from the root of a node k subscribing to event-type e_i is called the *distance* of e_i at node k. The average distance for all nodes subscribing to event-type e_i is denoted $distance_{avg}(e_i)$. The worst-case distance is denoted $distance_{wst}(e_i)$.

In order to determine these event-type metrics, the tree gathers data in a bottom-up fashion by having each internal node maintain partial values for its own subtree and pass these values up to its parent node in its *upstream control event*. Each node j maintains the following metrics, which is passes to its parent:

- $count_j(e_i)$ – the number of descendents of the current node j subscribing to event e_i (including node j).
- $cost_j(e_i)$ – the total number of hops that an event e_i must be propagated to the entire subtree rooted at the current node j.
- $avg\_cost_j(e_i)$ – the average number of hops that an event e_i must be propagated per interested node inside the subtree rooted at the current node j.
- $max\_cost_j(e_i)$ – the maximum number of hops that an event e_i must be propagated to an interested node inside the subtree rooted at the current node j.

Each node j passes its $count_j(e_i)$, $cost_j(e_i)$, and $max\_cost_j(e_i)$ values for each event-type e_i to the parent as parameters of its *upstream control event*. For each child j of a particular internal node k, the parent node calculates its own values recursively as follows:

$$count_k(e_i) = \sum_j count_j(e_i) + (0/1) \tag{4}$$

The last term (0/1) equals 0 if the internal node k does not subscribe to the event, and 1 if it does. Similarly, the $cost_k(e_i)$ at k is calculated in terms of each child as follows:

$$cost_k(e_i) = \sum_j count_j(e_i) + \sum_j cost_j(e_i) + (0/1) \tag{5}$$

The maximum cost value is simply the maximum of the maxima of its children plus 1:

$$max\_cost_k(e_i) = \max_j(max\_cost_j(e_i)) + 1 \tag{6}$$

At each node, the $avg\_cost_k(e_i)$ is a derived value of $count_k(e_i)$ and $cost_k(e_i)$:

$$avg\_cost_k(e_i) = count_k(e_i)/cost_k(e_i) \tag{7}$$

The root node, denoted r, then defines, for each event-type e_i, the system-wide count and distance values in the following way:

$$count(e_i) = count_r(e_i) \tag{8}$$

$$distance_{avg}(e_i) = avg\_cost_r(e_i) \tag{9}$$

$$distance_{wst}(e_i) = max\_cost_r(e_i) \tag{10}$$

Since all internal nodes are interested in knowing these three values (for scheduling by latency or popularity), the root node disseminates these values in a *downstream control event* as often as they change. It is worth noting here that when an internal node besides the root generates an event, it must also consider, in its latency calculations for the event, its own distance from the root as the event will first be passed upstream to the root and *then* downstream to the other subtrees of the network.

4.6 Impact of interleaving on dissemination latency

As described the Section 3.2, the root can create schedule iterations which are interleaved (i.e., downstream time-slot pairs alternated with upstream time-slot pairs) or clustered (i.e., all of the downstream slots in a contiguous block followed by the upstream slots). There are trade-offs to choosing either one over the other:

- *interleaved* – Pro: minimizes the expected scheduling delay for upstream events. Con: interrupts pipelining, increasing expected propagation delay for events.
- *clustered* – Pro: pipelining is not interrupted (except at end of cluster), minimizing propagation delay. Con: scheduling delay is worse, often must wait half an iteration (or more) before event can be scheduled.

Each style negatively affects a different component of the dissemination latency, either the *scheduling* or the *propagation* delay. Which method actually minimizes the average dissemination delay depends largely upon the depth and the subscription profile of the tree. If the distance (from the root) for event-types is, on the average, high, then the propagation delay may be the largest component of the overall delay. In this case, using a clustered schedule may make more sense in minimizing delay. If, on the other hand, the tree is relatively shallow and the distance of event-types is small, then the scheduling delay may make the greatest impact on overall delay. A clustered approach may be best for this situation.

Having the system automatically and introspectively determine the best style based on its size and subscription profile may be a subject of future work. For initial design simplicity, TD-DES takes a boolean value, *clustered*, as an application parameter at the root, and schedules iterations accordingly. This value is passed down through the downstream control event.

4.7 Choosing minimum iteration length

The TD-DES system must choose an appropriate minimum iteration length, denoted $T_{min\_iteration}$, for the root schedule. Choosing an iteration length which is too short creates an unneccessary abundance of downstream control events (one per iteration), and results in a higher amount of power consumption at the radio tranceivers of the network nodes. Conversely, choosing an iteration length which is too long results in a longer scheduling delay for downstream data events (as the events must wait for the next iteration to be scheduled).

$T_{min\_iteration}$ is the effective minimum iteration length inasmuch as iterations are not restricted by this value. If the root has generated an abundance of data events to be propagated downstream, it will attempt to schedule all of these in a single iteration, stretching the length of the iteration arbitrarily. Another purpose of $T_{min\_iteration}$ is to ensure that a certain number of *blank* time-slots (Section 4.2) are present in every iteration, so that internal nodes other than the root can schedule their own events.

In determining $T_{min\_iteration}$, the length of an event time-slot must be used. As mentioned previously, in our initial model, we consider time-slots for all event-types to be of uniform size. This constant length for event time-slots, measured in microseconds, is denoted $T_{time-slot}$. In the simulations of Section 6, $T_{time-slot}$ was varied from 2 to 8 ms.

$T_{min\_iteration}$, when chosen, effectively becomes the worst-case scheduling delay value for the deterministically-scheduled downstream data events. This worst-case scheduling delay must be small enough such that the following holds for each event-type e_i:

$$T_{min\_iteration} + Dp(wst)_i \leq Le_i \tag{11}$$

This indicates that $T_{min\_iteration}$, added to the worst-case progation delay for event-type e_i, must be less than the latency constraint Le_i for each event-type. Recall from before that if Le_i is not defined by the application, it defaults to the system-dependent maximum value, L_{max} (Section 4.4). To satisfy these constraints, $T_{min\_iteration}$ is therefore defined as:

$$T_{min\_iteration} = \max((2 * T_{time-slot}), \min_i(Le_i - Dp(wst)_i)) \tag{12}$$

Note that $T_{min\_iteration}$ cannot be less than 2 time-slots in duration – one for the s_c time-slot, and one for r_c (for the receipt and forwarding of the downstream control events); hence the first argument to the max function in the above equation.

$Dp(wst)_i$, the worst-case propagation delay for event-type e_i, is difficult at best to precisely determine (as it depends on both upstream and downstream scheduling characteristics), and so is best estimated heuristically using $distance_{wst}(e_i)$ and $T_{time-slot}$:

$$Dp(wst)_i = distance_{wst}(e_i) * T_{time-slot} * (2 + k) \tag{13}$$

k here is a real number greater than or equal to 0. If k is zero, then we consider the worst-case propagation delay to be the minimum time it takes to propagate an event to the root and back again, where the event is generated by the furthest subscribing node (that is, furthest from the root). This assumes that there is no delay between hops (i.e., each successive time-slot at the next level occurs immediately). In reality, this is hardly the case, particularly because a large portion of the time-slots are for propagation in the opposite direction. Therefore, the k parameter increases this *best* worst-case propagation delay estimation by an appropriate factor. For simulation purposes, values of 2 and 3 for k provided a good latency estimation. Varying k dynamically in accordance to changing network conditions, event subscription profiles, and scheduling styles may be the subject of future work.

A derived metric which is useful later is the number of time-slots in the minimum-length iteration, which is denoted $Nmin_{iteration}$ and defined as:

$$Nmin_{iteration} = T_{min\_iteration}/T_{time-slot} \tag{14}$$

4.8 Allocation of speculative upstream time-slots

Since time-slots for events that are propagated upstream cannot be allocated when they are needed, a certain number of upstream time-slots must be speculatively allocated for each event-type to the schedule during every iteration. This is done based on expected upstream event-propagation frequencies. The root node, since it receives every event that is propagated upstream, keeps an (event/time) measurement for each event-type e_i.This value is called the *upstreamfrequency* of e_i, is denoted Fu_i, and is measured in milliHertz (events/ millisecond). The root measures this by keeping track of how many events of each type it has received in the last N_{freq} milliseconds.

If N_{freq} is very high (say 2 x 10^{10} ms), then the value of Fu_i for each event-type e_i will not be highly adaptive to changes in upstream frequencies. Alternatively, if N_{freq} is too small, the system will be too senstive, and the values of Fu_i will fluctuate too wildly. It is desireable that the upstream speculative schedule be relatively stable, but at the same time representative of the actual bandwidth needs of the events being generated.

Given that we know the minimum iteration length, $T_{min\_iteration}$, and the upstream frequencies for each event-type, Fu_i, the number of upstream slots for event-type e_i, denoted Nu_i, to be allocated to in a particular iteration is defined as:

$$Nu_i = \max(Nu_{min}, \lceil (T_{min\_iteration} * Fu_i) \rceil) \tag{15}$$

In the above equation, Nu_{min} represents a system-dependent value indicating the minimum number of upstream slots which should be allocated to any event-type in any iteration. In our system, this number is 2 (one receive and one send slot). Even if the expected upstream frequency of an event-type is very low, the system will still allow for it to be propagated upstream at least one hop per iteration.

The total number of upstream time-slots needed per iteration is denoted Nu_{total} and is defined as the sum of numbers for each event-type, Nu_i:

$$Nu_{total} = \sum_i Nu_i \tag{16}$$

4.9 Allocation of deterministic downstream time-slots

When an event is generated, either by the root or an internal node, the node must wait until the next iteration before it can schedule the event downstream. First, consider that the root generates several events of event-type e_i during a particular iteration. The cardinality of the set of events generated of this type is denoted Nd_i, and is equal to the number of downstream time-slots that must be allocated for event-type e_i in the next iteration. The total number of downstream time-slots which needs to be allocated in the next iteration is denoted Nd_{total} and is defined as the sum of the cardinalities, Nd_i, for each event-type to be scheduled:

$$Nd_{total} = \sum_i Nd_i \tag{17}$$

Having values for both Nu_{total} and Nd_{total}, the root can also calculate the number of *blank* time-slots, denoted Nb_{total}, to be allocated for the next iteration as:

$$Nb_{total} = \max(Nb_{min}, Nmin_{iteration} - (Nu_{total} + Nd_{total})) \tag{18}$$

Nb_{min} specifies a system-dependent minimum number of blank time-slots to be allocated for each iteration. In our implementation, we set Nb_{min} to between 3 and 10, but this is wholly dependent on the size of the network, the event generation rates, and other factors. This is neccessary, for otherwise, in the case that the root schedules a burst of events, downstream nodes would have no bandwidth with which to schedule their own events (i.e., there would be no blank time-slots in the schedule).

4.10 Frequency of upstream iterations

Itermittently, the system must schedule a consecutive cluster of one or more *upstream iterations*. These are simply extensions of a regular iteration. That is, when the system decides to schedule an upstream iteration cluster, it takes the set of time-slots already allocated so far (i.e., the set defined by Nb_{total}, Nu_{total}, and Nd_{total}) and extends it with a set of *child_join* and *child_update* time-slots. The number of both types of slots is system dependent, and is equal to the fixed values Ncj and Ncu. Ncu is set to a value indicating the maximum number of children any parent node could have for a particular network. Ncj is set to a value high enough to allow disconnected nodes to join the network quickly (without contention from other disconnected nodes).

In our implementation, we set Ncu and Ncj based on the density of the network. With higher density, the expected number of children for a parent node grows, as does the expected number of disconnected nodes trying to join a particular parent node.

The *periodicity* and the *length* of each upstream iteration cluster also must be determined. The periodicity indicates how often upstream control events should be sent. How often should children send heartbeats to their parents, resend their (possibly updated) subscription and control information? In addition, how long must a disconnected node wait before it can send a $child_join$ event? This upstream periodicity is denoted $P_{upstream}$ and in our implementation is set to two seconds for a static network. However, this value should decrease depending upon the level of the mobility of the network.

How many iterations should a cluster be composed of? In a deep network, requiring many hops to the root, a single iteration may not be enough to propagate a leaf's upstream control information all the way to the root (due to the fact that the pipelining is disrupted over several hops). Therefore, the length of an upstream cluster, denoted $N_{upstream}$, represents the number of back-to-back upstream iterations in a cluster. A simple heuristic to define $N_{upstream}$ based on $tree - depth$ is:

$$N_{upstream} = \lceil tree\_depth/j \rceil \tag{19}$$

j here is another sysem-dependent constant, which we set to 4 in our implementation.

4.11 Sorting the event-types

A sorted order of the event-types must be maintained at all times. Re-sorting need only occur when either (1) the event-subscription profile of the network changes or (2) the *sort_criteria* list specified by the application layer changes. The *sort_order* is an ordered list of all the event-types in the system. If there are n event-types in the system, enumerated e_1, e_2, ..., e_n:

$$sort\_order = (e_{x1}, e_{x2}, e_{x3}, ..., e_{xn}) | e_{xi} = e_j \wedge i, j \in \{1, 2, ...n\} \tag{20}$$

That is, the *sort_order* provides a permutation of the event-types sorted in accordance with the *sort_criteria* list provided by the application layer (or the default list). Again, this sorting is not meant to take place every iteration – only when a change from the application or network layer requires it.

Sorting the event-types by popularity or priority is very straightforward, as these values are easily obtained by the system. Sorting by latency is different inasmuch the requirement is that all events are sorted in a way such that they all meet their latency requirements. If this is not possible, then the scheduler will schedule them such that the maximum number of events meet their latency requirements.

4.12 Sequencing the iteration

The last stage of the TD-DES scheduling algorithm is to determine the actual sequence of the time-slots allocated for the next iteration. This is dependent upon the *clustered* boolean and the *sort_criteria* ordered list (sections 4.6 and 4.4). At this point, the allocation is complete, and the time-slots have been grouped into unordered sets (defined by Nb_{total}, Nd_i and Nu_i for each event-type e_i). From these sets, the sequencer must derive a set of ordered sequences as follows:

- $Seq_{downstream}$ - the ordered sequence of time-slots for downstream data events.
- $Seq_{upstream}$ - the ordered sequence of time-slots for upstream data events.
- $Seq_{iteration}$ - the final ordered sequence of time-slots for the next iteration.

The first two sequences are both derived using the *sort_order* list to determine if one time-slot is greater than, equal to, or less than another time-slot in the set. Each sorting is an $n * \log n$ time-complexity operation, where n equals Nd_{total} for sorting $Seq_{downstream}$ and Nu_{total} for sorting $Seq_{upstream}$.

If the *clustered* boolean is true, then the $Seq_{downstream}$ is placed unbroken at the start of $Seq_{iteration}$, followed immediately by $Seq_{upstream}$. If the iteration is an upstream iteration, then the Ncu `child_update` time-slots are placed next in $Seq_{iteration}$ followed by the Ncj `child_join` time-slots. These are lastly followed by the Nb_{total} `blank` time-slots.

Additionally, $Seq_{iteration}$ always starts with a *send* downstream control event, s_c, and ends with a *receive* downstream control event, r_c, both added by the sequencer.

If the *clustered* boolean is false, then the Nd_{total} downstream and Nu_{total} upstream time-slots must be interleaved as evenly as possible. There are three possibilities, each handled differently:

1. $Nd_{total} = Nu_{total}$ When building $Seq_{iteration}$, we simply follow each downstream send/receive time-slot pair (dequeued from the front of $Seq_{downstream}$) with an upstream send/receive pair (from the front of $Seq_{upstream}$).
2. $Nd_{total} > Nu_{total}$, assume $Nd_{total} = Nu_{total} * q$ + r $(0 \leq r < Nu_{total})$. For every q downstream receive/send pairs we dequeue from the front of $Seq_{downstream}$ and place into $Seq_{iteration}$, we then dequeue and insert 1 upstream pair (from $Seq_{upstream}$) into $Seq_{iteration}$, and repeat this n times. If $r \neq 0$, we finish building $Seq_{iteration}$ by adding the remaining r downstream pairs.
3. $Nu_{total} > Nd_{total}$, assume $Nu_{total} = Nd_{total} * q$ + r $(0 \leq r < Nd_{total})$. For every q upstream receive/send pairs we dequeue from the front of $Seq_{upstream}$ and place into $Seq_{iteration}$, we then dequeue and insert 1 downstream pair (from $Seq_{downstream}$) into $Seq_{iteration}$, and repeat this n times. If $r \neq 0$, we finish building $Seq_{iteration}$ by adding the remaining r upstream pairs.

Lastly, after the interleaved portion of $Seq_{iteration}$ has been built, we add Nb_{total} blank time-slots to the end of the sequence.

4.13 Shifting the root schedule at lower levels of the tree

When the iteration sequence, denoted $Seq_{iteration}$, is received by a child node, the node must adjust (or shift) the iteration accordingly, and send it down to its children during the downstream control event time-slot, s_c, which marks the beginning of the iteration.

The child takes the parent's schedule during the receive control time-slot, r_c, indicated as the last slot of the current iteration. The s_c time-slot of its new iteration schedule will be correspond to the next time-slot in real-time. However, to shift the schedule, each upstream event-slot is shifted left in the schedule. If a downstream slot is displaced by this process, the downstream slot is moved to the right until it can be placed in a vacant time-slot (vacated by a left-shifted upstream slot). In other words, the displaced downstream slot slides to the right over the immediate upstream cluster until it finds the vacated spot made by the rightmost upstream event-slot of the cluster. Refer to Tables 1-3 of this document for examples of schedule shifting.

Upstream slots cannot be left-shifted into the two downstream control slots at beginning of the iteration. Instead, they are shifted to the end of the iteration (but before the final r_c slot marking the receipt of the next downstream control event).

At this point, events which have been generated at the child node during the past iteration can now be scheduled. This is done by the node locating the leftmost pair of blank time-slots and converting them into a send/receive pair for the event-type in question. Generated events should be sorted and scheduled in the same way as they are at the root, by whichever sorting criteria is decided upon.

After this adjustment of the parent's schedule has taken place, the node can send it to its own children and adhere to it until the next downstream control event, containing the next $Seq_{iteration}$, is received from its parent.

4.14 Event eviction policy

When a node receives or generates more events than its storage facilities can cache before such events are able to be propagated on the wireless channel, events must be evicted from storage based on some ordering in terms of their relative importance. In simulation, this is done simply by evicting events in the reverse order specified by the *sort_order* sequence (section 4.11). Ties (that is, multiple events of the same event-type or with the same position in the *sort_order*) are broken randomly.

4.15 Algorithmic complexity

Assuming a static topology and event-subscription profile (where a one-time cost of constructing the network tree and gathering subscription data is incurred by the network), the overall complexity of the system is dominated by the event ordering, which is $O(n \log(n))$, where n is the number of events generated for a particular iteration. This ordering is incurred mainly on the root, however, recall that internal nodes which generate events to extend the root's schedule must also order such events.

Although the ordering dominates the time complexity, another source of computation, which is encountered by all nodes, is the shifting of the parent's schedule by the child, an operation which costs $O(n)$, where n is simply the number of time-slots in the iteration.

Assume now a network where the topology, event-subscriptions and sorting criteria are dynamic. Every time the *sort_criteria* ordered list changes, the *sort_order* sequence of event-types must be recomputed, an $O(n \log(n))$ operation, where n is the number of event-types in the system. However, this operation is presumably done infrequently. When the event-subscriptions change, upstream iterations pass the new information up the tree toward the root. In addition, topology changes also require a recalculation of the $count_j(e_i)$, $cost_j(e_i)$, $avg\_cost_j(e_i)$, and $max\_cost_j(e_i)$ metrics for the subtree where the change takes place. Any topology change could require an $O(nk)$ operation (at each node) to recompute these values, where n is the number of nodes in the system, and k is the number of event-types. However, in general, n in this complexity is really the maximum degree of each node, which although bounded by $n-1$ theoretically, is not expected to be nearly as high unless the topology is very dense and uneven.

5 Analytical Performance Analysis

Given a certain set of network, subscription, and event parameters, it is of interest to be able to predict the average dissemination latencies for the event-types of the system. In addition, we would also like to be able to predict the average power-consumption at each node. To do so for the purposes of this paper, we make a few simplying assumptions (also used by the simulation model in the following section):

- The network tree topology is static and therefore parent/child associations are fixed.

- Although upstream event time-slots are allocated, we only estimate latencies for downstream events started at the root.

5.1 Latency

Given a particular network topology with n nodes and a particular event subscription of k different event-types, we want to estimate the average dissemination delay for each event-type e_i, denoted $D(exp)_i$ (from section 4.4). This can be broken into the sum of both the scheduling and propagation delays:

$$D(exp)_i = Ds(exp)_i + Dp(exp)_i \tag{21}$$

The scheduling delay can further be broken into two components:

$$Ds(exp)_i = Ds_{pre-iteration}(exp)_i + Ds_{sorting}(exp)_i \tag{22}$$

$Ds_{pre-iteration}(exp)_i$ is the remaining time left in the current iteration at the time the event is generated. This depends first upon $T_{iteration}$, the minimum iteration length. If an event is generated at a random point in time, then the event must first wait for the current iteration to end. This time, on average will be $(T_{iteration}/2)$. Therefore, if we expand the previous equation using equation (12), we derive:

$$Ds_{pre-iteration}(exp)_i = (\max((2 + Nu_{total} + Nd_{total} + Nb_{min}) * T_{time-slot}, \min_j(Le_j - Dp(wst)_j))/2) \tag{23}$$

Here we are assuming that Nu_{total} is the sum of the minimum values for upstream time-slots for all event-types, and that $Nb_{total} = Nb_{min}$. Nd_{total} here represents the number of events we are generating in the course of one iteration. We choose j to be the variable that ranges across all event-types such that event-type e_j satisfies the minimum function above. Using equation (13), we can expand this further:

$$Ds_{pre-iteration}(exp)_i = (\max((2 + Nu_{total} + Nd_{total} + Nb_{min}) * T_{time-slot}, \min_j(Le_j - (distance_{wst}(e_j) * T_{time-slot} * (2 + k))))/2) \tag{24}$$

The second part of the scheduling delay, $Ds_{sorting}(exp)_i$, depends on Nd_{total} for the particular iteration. Let pos_i be the position of an event of type e_i in the $Seq_{downstream}$ sequence, such that $pos_i \in \{1, 2, ..., Nd_{total}\}$. Then if *clustered*, the event must wait for pos_i time-slots before it is scheduled (this includes one time-slot for the downstream control event, s_c). If *!clustered*, the sorting delay is lengthened by the interleaving of upstream time-slots. The number of upstream time-slot pairs that occur before the event of type e_i is allocated a downstream slot is roughly equal to (from section 4.12) $\lfloor (Nu_{total} \ / \ Nd_{total}) * pos_i \rfloor$:

if (*clustered*)

$$Ds_{sorting}(exp)_i = pos_i * T_{time-slot} \tag{25}$$

else

$$Ds_{sorting}(exp)_i = (pos_i + \lfloor (Nu_{total}/Nd_{total}) * pos_i \rfloor) * T_{time-slot} \tag{26}$$

Expected propagation delay, $Dp(exp)_i$, can be expressed as:

if (*clustered*)

$$Dp(exp)_i = T_{time-slot} * [distance_{avg}(e_1) * 2 + (\lfloor (distance_{avg}(e_1) + pos_i)/(Nd_{total}) \rfloor * (Nu_{total})] \tag{27}$$

else

$$Dp(exp)_i = T_{time-slot} * [distance_{avg}(e_1) * 2 + \\ (\lfloor (Nu_{total}/Nd_{total}) * (distance_{avg}(e_i)) \rfloor)] \tag{28}$$

The above two formulas allocate 2 time-slots worth of propagation time per hop, in addition to allocating time for upstream time-slots which cause further delay in the propagation. Note that in a clustered approach, for short paths (less than Nd_{total} hops from the root), no upstream time-slots will be encountered. However, for longer paths (greater than or equal to Nd_{total} hops from the root), upstream time-slots will be encountered in at least one large cluster during the propagation. For interleaved scheduling, an upstream time-slot can cause delay at every hop along the path. As paths become longer, the effect that interleaving or clustering has on the overal propagation delay becomes less and less.

Adding scheduling and propagation delay together, the total expected dissemination delay for event-type e_i is now:

if (*clustered*)

$$D(exp)_i = (\max((2 + Nu_{total} + Nd_{total} + Nb_{total}) * T_{time-slot}, \min_j(Le_j - \\ (distance_{wst}(e_j) * T_{time-slot} * (2 + k))))/2) + pos_i * T_{time-slot} + \\ T_{time-slot} * [distance_{avg}(e_i) * 2 + (\lfloor (distance_{avg}(e_1) + pos_i)/(Nd_{total}) \rfloor * Nu_{total})] \tag{29}$$

else

$$D(exp)_i = (\max((2 + Nu_{total} + Nd_{total} + Nb_{total}) * T_{time-slot}, \min_j(Le_j - \\ (distance_{wst}(e_j) * T_{time-slot} * (2 + k))))/2) + \\ (pos_i + \lfloor (Nu_{total}/Nd_{total}) * pos_i \rfloor) * T_{time-slot} + \\ T_{time-slot} * [distance_{avg}(e_i) * 2 + (\lfloor (Nu_{total}/Nd_{total}) * distance_{avg}(e_i) \rfloor)] \tag{30}$$

Figures 8, 9 and 10 plot the expected latencies in TD-DES, where Nd_{total} and Nu_{total} are fixed, but pos_i varies from 1 to Nd_{total}. Two networks are shown, one using clustered scheduling, and the other using interleaved scheduling. The average distance that the events must travel (hops from the root) is kept fixed at 5 hops. Nd_{total} is 20 and Nu_{total} is 10, and Nb_{min} is set to 2. $T_{time-slot}$ is set to 2 ms. It is assumed also that no latency constraints for events are specified, and that the default maximum latency, Le_j, is 200 ms, and that the farthest distance from the root is 10 hops. Furthermore, k is set to 2 (see section 4.7).

Figure 8 plots the scheduling delay for both the clustered and the interleaved scheduling formats as a function of pos_i. The interleaved scheduling delay is intuitively higher than the clustered scheduling delay, as upstream time-slots are interleaved with the downstream events to be scheduled. The jaggedness of the interleaved line accounts for these jumps in delay from upstream time-slots. In the clustered network, all of the downstream events are scheduled contiguously, and the upstream time-slots follow this contiguous block. Therefore, the only scheduling delay for clustered TD-DES is in waiting for the $(pos_i - 1)$ preceeding events in the sequence to be scheduled.

Figure 9 plots the expected propagation delay as a function of pos_i. With the interleaved schedule, as we expect, pos_i has no effect on the propagation delay, as the number of interleaved upstream time-slots that increase the delay is a function of the hop-distance, which is fixed at 5 hops. However, for the clustered scheduled, we see that at first the propagation delay is less than the interleaved schedule. This is because no upstream time-slots are encountered when pos_i is less than 15 – that is, the event is tightly pipelined for all 5 hops. When pos_i is greater than 14, however, the event encounters a block of upstream time-slots, which causes a large jump in delay.

Figure 10 simply shows the total dissemination delay for both a clustered and interleaved schedule as a function of pos_i, and suggests that the clustered schedule will give at least as good latency

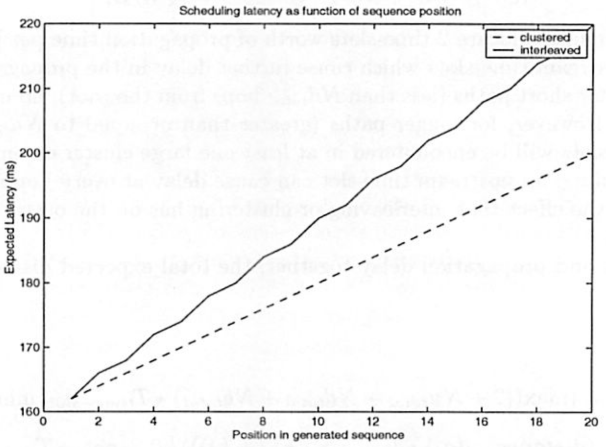

Figure 8: Scheduling delay: clustered schedule versus interleaved schedule. Scheduling delay for an event e_i as a function of its position in the sequence of downstream events to be scheduled in the next iteration.

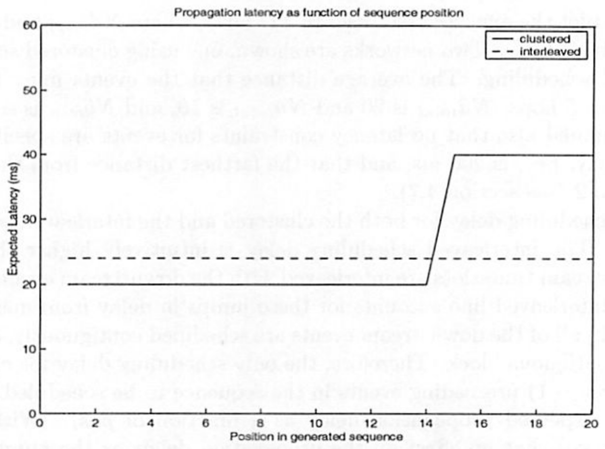

Figure 9: Propagation delay: clustered schedule versus interleaved schedule. Propagation delay for an event e_i as a function of its position in the sequence of downstream events to be scheduled in the next iteration.

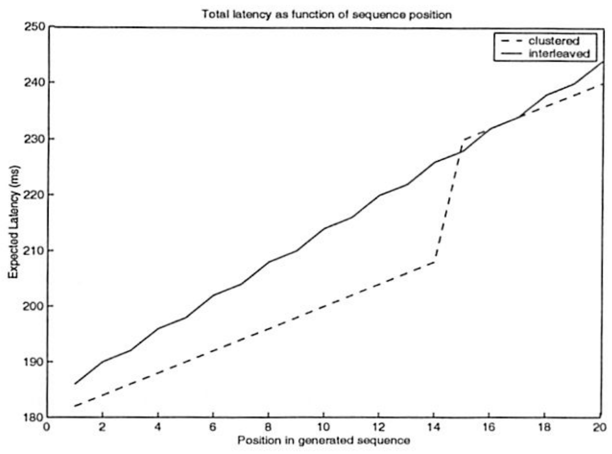

Figure 10: Total delay: clustered schedule versus interleaved schedule. Total dissemination delay for an event e_i as a function of its position in the sequence of downstream events to be scheduled in the next iteration.

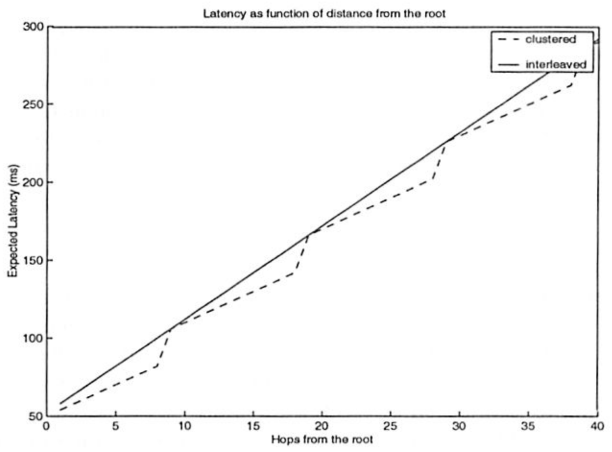

Figure 11: Latency: clustered schedule versus interleaved schedule. Dissemination delay for an event e_i as a function of the distance (in hops) it is propagated from the root.

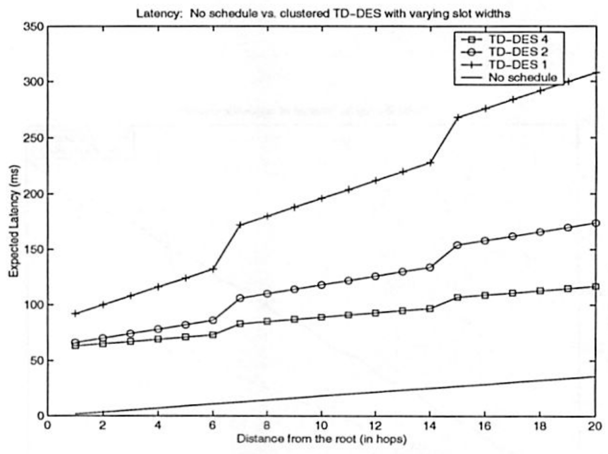

Figure 12: Latency: No schedule vs. clustered TD-DES with varying slot widths.

as the interleaved schedule. It would be interesting to see how delay is affected (for both scheduling methods) as a function of the number of hops to be propagated.

Figure 11 displays latency as a function of distance propagated from the root (in hops). Clearly, this diplays the same phenomenon we saw before in Figure 6 – for downstream event propagation at the root, clustered latency is always equal to or lower than interleaved latency. This is naturally because when clustering, TD-DES schedules the downstream events first before the upstream time-slots. If it scheduled the upstream time-slots first, we would undoubtably see a similar graph, except that the steps on the clustered line would be *over* the interleaved line, rather than under it. The advantage of improved downstream event latency comes at the cost of increased delay for propagated upstream events (although this isn't shown). However, this advantage becomes more and more marginal as the distance from the root increases.

How would latency using TD-DES compare to a non-scheduled CSMA/CA layer which broadcasts events immediately without any scheduling delay or per-hop buffering? For the non-TD-DES network, we want to use a general metric, denoted $hop\_delay_{avg}(e_i)$, which is the average expected delay per hop for a transmitted message. The delay per hop will be a fraction of the per-hop delay of TD-DES, since TD-DES allows for time-slots several times longer than the actual propagation delay for an event-frame. Let us consider the ratio of $T_{time-slot}$ to the actual propagation delay for an event/MAC frame to be denoted $Ratio_{prop}$. For the non-TD-DES model, without any collisions, $hop\_delay_{avg}(e_i)$ will be:

$$hop\_delay_{avg}(e_i) = (T_{time-slot}/Ratio_{prop}) \tag{31}$$

However, we also want to account for the delay which occurs due to medium contention and collision handling. We can set this parameter, $T_{contention}$, as an additional per-hop delay based on the rates at which downstream and upstream events are generated. To compare with TD-DES, we can take the sum of Nd_{total} and Nu_{total} and divide this by $T_{time-slot}$ to get the rate of event generation, which gives a fair estimation of the total number of events currently in flight. It is reasonable to assume that the delay per hop caused by medium contetion will go up linearly with the number of events in flight. $T_{contention}$ can be estimated using a linear formula as follows:

$$T_{contention} = 1/u * ((Nd_{total} + Nu_{total})/T_{time-slot}) \tag{32}$$

u should be chosen carefully for the particular network in question. We can then estimate $D(exp)_i$ for the non-TD-DES network model to be:

$$D(exp)_i = distance_{avg}(e_i) * ((T_{time-slot}/Ratio_{prop}) + T_{contention}) \tag{33}$$

Combining (32) and (33), this expands to:

$$D(exp)_i = distance_{avg}(e_i) * ((T_{time-slot}/Ratio_{prop}) + 1/u * ((Nd_{total} + Nu_{total})/T_{time-slot})) \quad (34)$$

Figure 12 plots the expected latency of four networks as the propagation distance from the root is varied from 1 to 20 hops. 4 lines are plotted, each for a different network. The first three networks use clustered TD-DES with 3 different $Ratio_{prop}$ values: 4, 2 and 1. The 4th network is the non-TD-DES network, which has no scheduling delay or further per-hop buffering delays (except for medium contention delays). TD-DES 4 clearly has the worst scheduling and propagation delays. TD-DES 2 has rougly more than half the scheduling delay as TD-DES 4, and its propagation delay per hop is apparently half that of TD-DES 4's. TD-DES 1 has roughly the same scheduling delay as TD-DES 2, but its propagation delay is on par with the non-scheduled network. This is intuitive, for if the time-slots are exactly as wide as they need to be to propagate a single event/MAC frame, one would expect propagation delay to be near-optimal (barring any other delays). However, this leaves no room for collision handling. Extending TD-DES to incorporate collision handling directly as part of the protocol (while maintaining time-slots at their minimum width) was thought to be too much of an added complication at this early developmental stage – this would require incorporating the wireless MAC protocol into TD-DES (perhaps by allocating extra slots for retransmissions after collisions). As the subject of future work, this extention could potentially make TD-DES not only extremely power-efficient, but allow it to provide near-optimal dissemination latency as well.

5.2 Power consumption

Where we saw somewhat sub-par latency performance for TD-DES (although it was still well within acceptable bounds for many applications), we expect the gain in power efficiency to be dramatic over its non-scheduled counterpart network. We define the average, or expected, energy consumption of a node for a particular iteration as:

$$E_{iteration} = T_{time-slot} * [(N_{Tx} * P_{Tx}) + (N_{Rx} * P_{Rx}) + (N_{standby} * P_{standby})] \quad (35)$$

N_{Tx}, N_{Rx}, and $N_{standby}$ are the number of time-slots that a node spends in Tx, Rx, and standby modes, respectively, for a particular iteration. These can be determined using the event generation and subscription profiles of the nodes of the network. For a particular radio platform, the values P_{Tx}, P_{Rx}, and $P_{standby}$ represent the power consumption level of each node in Tx, Rx, and standby modes, respectively. These values depend upon the particuar radio platform being used. For example, the Proxim RangeLAN2 2.4 GHz 1.6 Mbps PCMCIA card expends 1.82W in transmit mode, 1.80W in receive mode, and 0.18W in standby mode [14]. For the sake of this analysis, we assume these power consumption levels. We also assume a homogeneous network – that is, each node uses the same radio hardware.

For a particular network, we need to know the number of nodes effectively subscribing to each event-type, denoted the *effective count* of each event-type. The effective count of event-type e_i is denoted $effective\_count(e_i)$. For each of the event-types e_i in the system, if we know, for a particular iteration, Nd_i and Nu_i, and we know $effective\_count(e_i)$, we can calculate N_{Tx}, N_{Rx}, and $N_{standby}$ for each node in the system (for a particular iteration).

We first need to determine the total number of time-slots in the iteration in question. Borrowing from equations (23) and (14), we can calculate the number of time-slots in a particular iteration, N_{slots}, to be:

$$N_{slots} = (\max((2 + Nu_{total} + Nd_{total} + Nb_{min}) * T_{time-slot}, \min_j(Le_j - Dp(wst)_j))/T_{time-slot}) \quad (36)$$

If we make the simplifying assumption that the speculative upsteam send time-slots are always used to transmit data, then the number of time-slots spent Rx and Tx mode for an iteration are equal (since the same number of send and receive slots are allocated for each event-type). For a particular node, the number of slots spent in Rx and Tx mode, denoted N_{Rx} and N_{Tx}, respectively, can be expressed as:

$$N_{Rx} = N_{Tx} = 1 + \sum_{s=1}^{N_{event-types}} Sub_s * ((Nd_s + Nu_s)/2) \tag{37}$$

Here s ranges over the event-types, of which there are $N_{event-types}$ in the system. Sub_s is a boolean which indicates whether the current event-type s is effectively subscribed to by the node in question. We add one to each number, for the send and receive downstream control events, s_c and r_c.

The number of slots then spent in standby mode, denoted $N_{standby}$, can then be calculated as:

$$N_{standby} = N_{slots} - (N_{Tx} + N_{Rx}) \tag{38}$$

The energy consumption at a particular node, this brings us to the previous energy equation, here reproduced:

$$E_{iteration} = T_{time-slot} * [(N_{Rx} * P_{Rx}) + (N_{Tx} * P_{Tx}) + (N_{standby} * P_{standby})] \tag{39}$$

The power consumption level at this node, for the duration of this iteration, is therefore:

$$P_{iteration} = E_{iteration}/(N_{slots} * T_{time-slot}) \tag{40}$$

However, rather than iterating over equations (37) through (40) for each node in the network, an easier way to calculate power consumption is by using the $effective\_count(e_i)$ metric for each event-type e_i. Based on these values, the total number of Tx, Rx, and standby slots spent during an iteration for the entire network, denoted $N(net)_{Tx}$, $N(net)_{Rx}$, and $N(net)_{standby}$, respectively, can be calculated. These can be expressed as:

$$N(net)_{Rx} = N(net)_{Tx} = N_{nodes} + \sum_{s=1}^{N_{event-types}} effective\_count(e_i) * ((Nd_s + Nu_s)/2) \tag{41}$$

$$N(net)_{standby} = (N_{nodes} * N_{slots}) - (N(net)_{Tx} + N(net)_{Rx}) \tag{42}$$

The energy consumption, for the iteration, for the entire network, is then:

$$E(net)_{iteration} = T_{time-slot} * [(N(net)_{Rx} * P_{Rx}) + (N(net)_{Tx} * P_{Tx}) + (N(net)_{standby} * P_{standby})] \tag{43}$$

Power consumption for the entire network is then, by equation (40):

$$P(net)_{iteration} = E(net)_{iteration}/(N_{slots} * T_{time-slot}) \tag{44}$$

How would we estimate power-consumption for the non-scheduled network? If we assume that the same event generation rates, given by Nd_{total}, Nu_{total}, and Nb_{min} for a particular iteration, then we can determine the iteration length for this set of parameters, and, with the $effective\_count(e_i)$ metrics, determine what portion of time each node spends in Tx and Rx modes (there is no switching to standby mode for nodes of this network). First, the length of the time period we are measuring power consumption for is equal to the length of the counterpart TD-DES network's iteration, given by multiplying equation (36) by $T_{time-slot}$ (we will denote this period $T_{iteration}$ as it is equal to this value). We then determine the aggregate time for all network nodes spent in Tx and Rx modes during $T_{iteration}$, denoted $T(net)_{Rx}$ and $T(net)_{Tx}$, as follows:

$$T(net)_{Tx} = (T_{time-slot}/Ratio_{prop}) * \sum_{s=1}^{N_{event-types}} effective\_count(e_i) * ((Nd_s + Nu_s)/2) \tag{45}$$

$$T(net)_{Rx} = T_{iteration} - T(net)_{Tx} \tag{46}$$

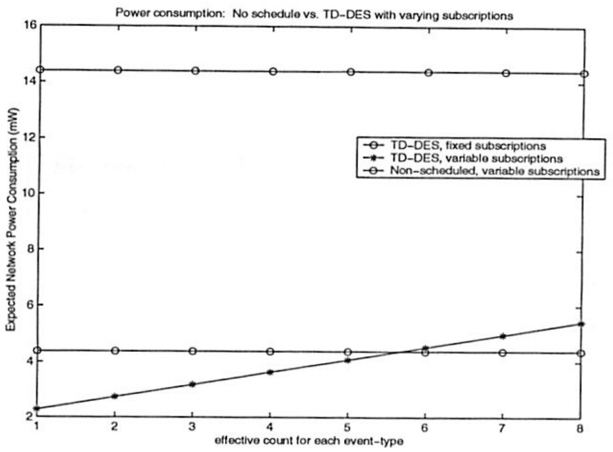

Figure 13: Power consumption: No schedule vs. TD-DES with varying subscriptions. Light traffic, Tx power equals Rx power consumption.

The energy expended by the network during this period is expressed as:

$$E(net)_{iteration} = (T(net)_{Rx} * P_{Rx}) + (T(net)_{Tx} * P_{Tx}) \tag{47}$$

The power consumption rating is therefore:

$$P(net)_{iteration} = E(net)_{iteration}/T_{iteration} \tag{48}$$

Refer to the publish/subscribe tree given in Figure 2, on page 5. For this network, let us assume that we have a fixed downstream event generation rate at the root, such that Nd_i is 4 per iteration for each of the three event-types, e_i, e_2, and e_3. Nd_{total} is therefore 12. We assume also that the Nu_i is 2 for each event-type, so that Nu_{total} is 6. As before, we assume that these upstream time-slots are always being used to propagate events, as our power model assumes this (in this sense, the power formula makes a *worst-case* estimation for the speculative portion of the schedule). Nb_{min} is also set to 2 and we use the power ratings from the Proxim RangeLAN2 2.4 GHz 1.6 Mbps PCMCIA card, given before.

From Figure 2, it is clear that $effective\_count(e_1)$ is 6, $effective\_count(e_2)$ is 6, and $effective\_count(e_3)$ is 5. Figure 10 plots 3 power-consumption lines. The first is the steady consumption rating given by TD-DES for this subscription profile. The second line plots TD-DES as a function of $effective\_count(e_i)$, where this value for all three event-types are equal and range from 1 to the maximum, which is 8. The third line is the expected power consumption for the non-scheduled network which doesn't use TD-DES, where the $effective\_count(e_i)$ follows the same pattern as the second line.

The power consumption for TD-DES for the fixed subscription (according to Figure 2) intuitively does not vary, as all of the parameters stay the same. The power consumption for TD-DES with the ranging subscription density (from light to maximum), varies dramatically, and demonstrates the effectiveness of the protocol, especially for lightly subscribed networks. Where each $effective\_count(e_i)$ equal 1, the power consumption is only 52% of that for the fixed-subscription TD-DES network, and only 42% of that for the fully-subscribed network.

Keep in mind that for the non-scheduled network, each network node is constantly in either Rx or Tx mode. The power consumption of this typical CSMA/CA network is much greater here than it is for any configuration of TD-DES. The most lightly-subscribed TD-DES network consumes only 16% as much energy as the non-scheduled network, resulting in 6 times the power savings (and potentially 6 times the network longevity). At its worst, the fully subscribed TD-DES network consumes only 37% as much as the non-scheduled network – still a remarkable savings.

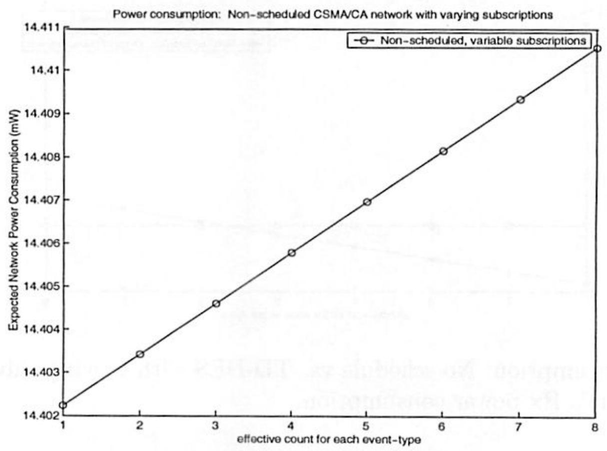

Figure 14: Power consumption: From Figure 10, close-up of non-scheduled CSMA/CA network with varying subscription.

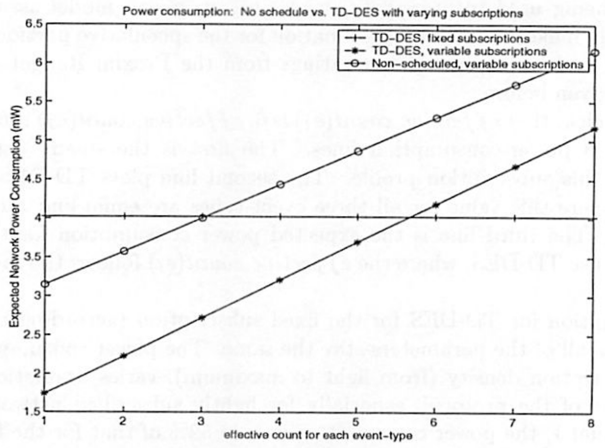

Figure 15: Power consumption: No schedule vs. TD-DES with varying subscriptions. Heavy traffic, Tx power much greater than Rx power.

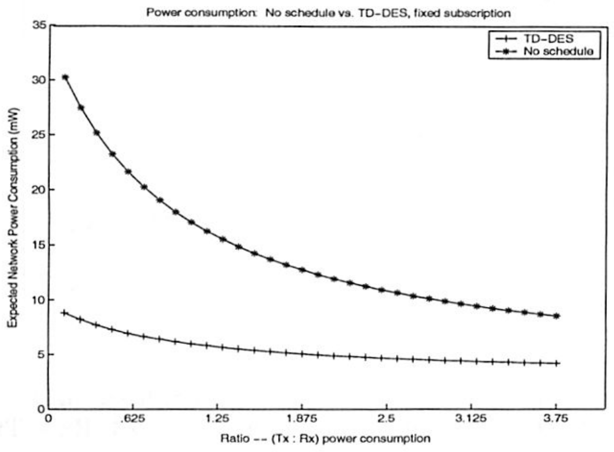

Figure 16: Power consumption: No schedule vs. TD-DES, fixed, moderate subscription, heavy traffic. Ratio of Tx : Rx power consumption varies from 1/8 to 4. Rx + Tx power = 4.5W.

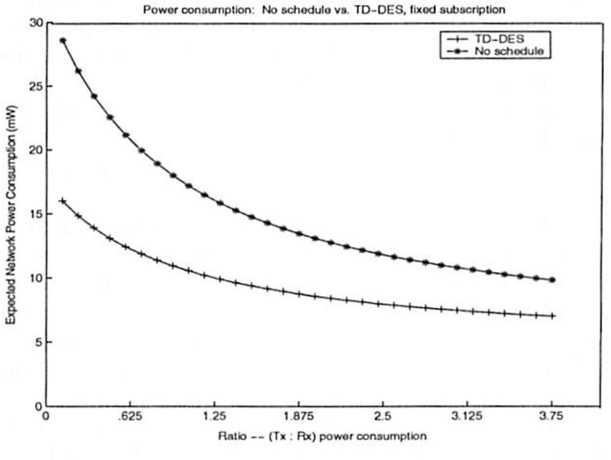

Figure 17: Power consumption: No schedule vs. TD-DES, fixed, full subscription, heavy traffic. Ratio of Tx : Rx power consumption varies from 1/8 to 4. Tx + Rx power = 4.5W.

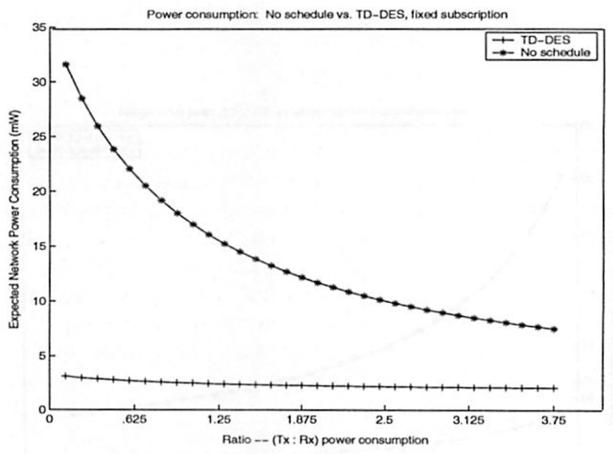

Figure 18: Power consumption: No schedule vs. TD-DES, fixed, light subscription, light traffic. Ratio of Tx : Rx power consumption varies from 1/8 to 4. Rx + Tx power = 4.5W.

Although it appears as if the non-scheduled network (whose subscription also varies along the x-axis) does not experience a change in power consumption, this is not so – it is very small, however. Figure 11 demonstrates this change, as the subscription density increases. What this simply means is that the ratio of time spent in Tx mode to Rx mode increases. But since our radio model specifies roughly the same values for both Rx and Tx power ratings (1.82W and 1.80W), this shift hardly makes an impact on the overall power consumption (the minimum and maximum power consumption values differ by only 0.008289 mW). This shift would be much more marked if the Tx and Rx power ratings were much different.

Let us consider the same network topology and variable subscription profiles as before. Before, however, we considered an event generation frequency which was fairly light. This of course allowed TD-DES to leverage it's ability to allow nodes to power-down to standby mode. What if the event generation frequencies were heavy, almost saturating the bandwidth capacity of the network? In addition, what if the Tx and Rx power ratings for the radio hardware were much different? For example, the Ritron SST-144D 2-way radio tranceiver [1] operates at 4W, .14W, and .325W in Tx, standby, and Rx modes, respectively. Figure 15 plots the same network configurations and variable subscription profiles as did Figure 13, except with the new radio model and a much higher event generation frequencies. Nd_i and Nu_i are both 10 for each event-type, making Nd_{total} and Nu_{total} both 30 each. Nb_{min} is still held at 2.

Again, we note that the fixed subscription TD-DES network has a steady power consumption, and the variable subscription TD-DES network sees an increase in power consumption as $effective\_count(e_i)$ increases. Also, it is interesting that for the lowest subscription profile, where $effective\_count(e_i)$ equals 1, the non-scheduled network has lower power-consumption than the fixed-subscription TD-DES network (although the fixed-subscription is much higher than where $effective\_count(e_i)$ equals 1). This graph shows that the effectiveness of using TD-DES over a non-scheduled network varies as a function of the ratio of Tx power to Rx power consumption.

Figure 16 plots the power consumption of TD-DES versus the non-scheduled network a function of this ratio, while keeping all other variables fixed at the level they were at for the previous figure. In addition, $effective\_count(e_i)$ for all event-types is fixed at 4.

Intuitively, when the Rx power rating is high, the CSMA/CA layer will do poorly inasmuch as it is almost always in this radio mode (especially with light traffic). A comparably configured TD-DES network will see a much lower power consumption levels over the non-scheduled network when Rx power is high, as it spends much less time in this mode. Figure 16 plots a network with moderate traffic levels (as $effective\_count(e_i)$ was 4). Figure 17 plots the same analysis except that the subscription density is maximum ($effective\_count(e_i)$ is 8).

As expected, TD-DES expends more energy with the maximal subscription density, since every node receives and forwards every generated event, and less time is spent in standby mode. In this case, TD-DES has less of an advantage over the non-scheduled network.

Figure 18 demonstrates the wide disparity of power consumption for a network configuration that allows TD-DES to fully leverage its efficiency: a light subscription profile, low event generation frequencies, and high Rx transceiver power consumption. It is important to emphasize that this configuration exactly describes the characteristics of many low-bandwidth, wireless sensor network applications, such as intruder detection (or the forest fire scenario of Section 2.1), where data is only infrequently passed along the network (e.g., when there is an intruder or a forest fire).

6 Simulation and Implementation

6.1 Simulation

An ad-hoc networking simulator was written in C++ to compare latency, power-consumption, and frame drop-rate statistics between a set of randomly-generated network pairs. The first network in each pair implemented a CSMA/CA MAC layer (with exponential random backoff) along with with the TD-DES scheduling overlay. The second network implemented the same MAC layer but without TD-DES (i.e., the non-scheduled network).

For each pair, both networks were identical in their topologies and event subscription profiles. Also, for each pair the same poisson event generation distribution was used to simulate events being generated and disseminated. Topologies were static, all nodes had omnidirectional, uniform broadcast ranges (i.e., links were symmetric), and no probabilistic simulation of link breakages and other random network conditions were modeled. In other words, both networks were modeled under optimal conditions to test optimal performance statistics given a particular set of network parameters. In addition, only event generation at the root (with subsequent downstream propagation) was modeled in the first simulation phase.

For the first simulation, a wireless network of 25 nodes covering a 150 square meter area was randomly generated. Each node had a 30 meter transmission range, and a tranceiver with Rx, Tx, and standby power output ratings of 1.80W, 1.82W, and 0.18W, respectively (according to the Proxim RangeLAN2 specification). In addition, there were 3 event-types in the system: e_1, e_2, and e_3. Each node had a 0.4 probability of subscribing to e_1, a 0.5 probability of subscribing to e_2, and a 0.6 probability of subscribing to e_3. In all, this was a fairly *moderate* subscription profile (a maximally dense profile would give a 1.0 probability that each node subscribed to *all* event-types).

Because some nodes in the network topology had up to 3 children, a time-slot width of 3 times the actual time needed to propagate an event-frame was used by the TD-DES scheduler. Each event-frame was 8 bytes in size and required 2 ms to propagate (given the bandwidth of the hypothetical radio hardware). L_{max} was set to 300 ms and each node could buffer up to 10 frames. TD-DES was configured to sort generated events in order of *popularity*. Events were generated using a poisson distribution.

Figure 19 plots the power consumption of a TD-DES network and its counterpart, non-scheduled network as a function of increasing rate of event generation (from an average generation rate of roughly 10 Hz to 80 Hz – however, the frequency is given by the average distance in time-slots between generated events, used by the poisson distribution). As expected, given the moderate subscription profile (allowing nodes to power-down during unsubscribed slots) and with light event generation, TD-DES exhibits markedly less power consumption than its non-scheduled counterpart network. TD-DES' RF output is between 20% to 55% of that of the non-scheduled network. Since the Rx power is virtually the same as the Tx power rating, the non-scheduled network experiences virtually no power output reduction, even when the event generation is light (as nodes of that network are always in Rx mode when they are not transmitting events).

As a second test of TD-DES' power efficacy, a 100-node pair of networks was simulated over a 400 square meter range, each node with a 50 foot wireless range. The same radio model and set

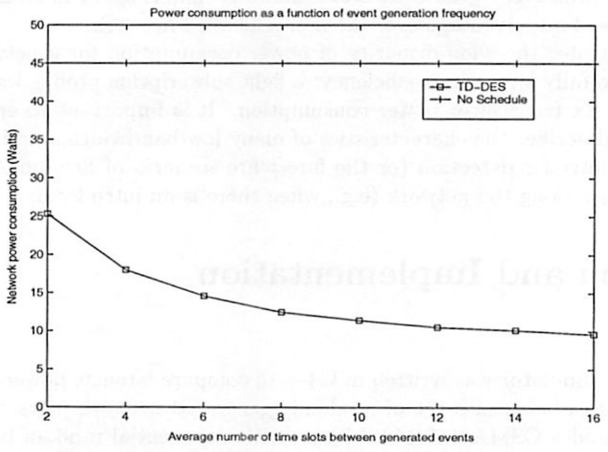

Figure 19: Power consumption: No schedule vs. TD-DES. Both simulated networks had the same 25 node topology with the same moderate subscription profiles and stochastic event generation model. Each network's total RF power consumption is given as a function of decreasing event generation frequency (given in average number of time-slots between events).

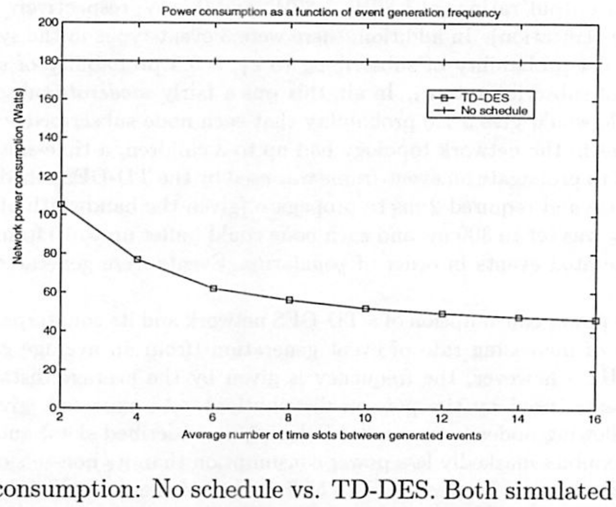

Figure 20: Power consumption: No schedule vs. TD-DES. Both simulated networks had the same 100 node topology with the same light subscription profiles and stochastic event generation model. Each network's total RF power consumption is given as a function of decreasing event generation frequency (given in average number of time-slots between events).

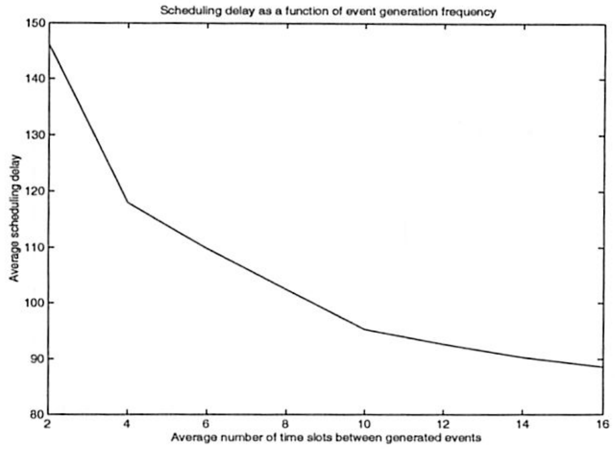

Figure 21: Scheduling latency: TD-DES. For the 25 node network, scheduling latency decreases as the event generation frequency decreases due to shorter queueing delays for newly generated events.

of event-types as before were used. However, a very light subscription profile was used – allowing TD-DES to maximize its efficiency. In this case, each node had a 0.1 probability of subscribing to each event-type. Figure 20 plots the power output of TD-DES and the non-scheduled network as functions of increasing event generation frequency, as before. In this case as well, the power savings of TD-DES is dramatic, with TD-DES expending between 25% to 58% as much power as the non-scheduled network.

For the 25 node network, the average message latency for both networks agreed with the previous analytical results. For the highest event generation frequency, the average propagation latency for the non-scheduled network was roughly 7.4 milliseconds (with an average distance of 3 hops), and for the TD-DES network, the average was 3 times this amount – in accordance with the time-slot width of 3 times the actual needed propagation time – at 22.153 milliseconds. By far, the greatest source of latency was derived from the the scheduling component of the delay – which was anywhere between 90 and 140 milliseconds. The maximum scheduling delay never exceeded 300 ms (the system set maximum). As expected, with increased event generation frequency, the scheduling delay increases, as the scheduling queue is larger for each iteration and events toward the end of the queue must wait (in some cases) up to two iteration lengths before they can be propagated to the wireless channel. Figure 21 plots the scheduling latency for the 25-node network as a function of event generation frequency. The latency results for the 100-node network showed the same trend.

A further test was run to determine the effect of popularity on an event-type's effective scheduling latency. A 100 node network, with 10 different event-types, was simulated. The 10 event-types were given a range of subscription probabilties from 0.1, 0.2, and so on, sequentially up to 1.0. The popularities (how many nodes subscribed to each event-type) were recorded and the scheduling latency for each compared. An average event generation frequency was used (where the average number of time-slots between generated events was 8). Figure 22 plots the results of the experiment. Clearly, as expected, the average scheduling delay was inversely proportional to the popularity of the event-type. As a result, the usefulness of ordering newly generated events by popularity has been verified. Events that are considered important (using their popularity as a gauge) will be disseminated promptly and be less likely to exceed an application-defined latency contraint than will a relatively unpopular event. In addition, because popular events are disseminated first and are received by more nodes, fewer events will exceed their latency bounds than if random or reverse-popularity ordering were used.

However, what if popularity isn't an accurate measure of the importance that an application

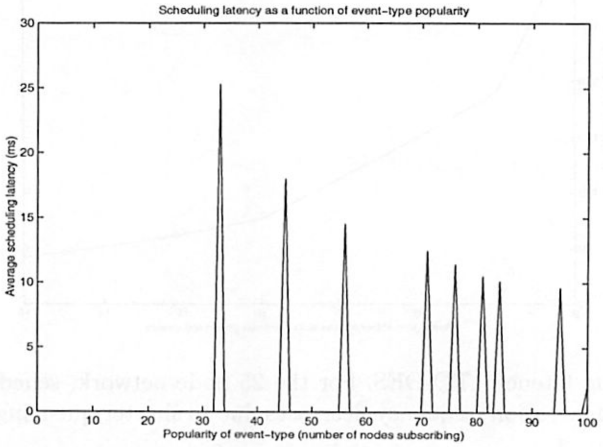

Figure 22: Scheduling latency as a function of event-type popularity. For the 100 node network, scheduling latency decreases with increasing popularity (as the scheduler gives preference to popular events). The spikes represent the popularities of the 10 event-types in the system.

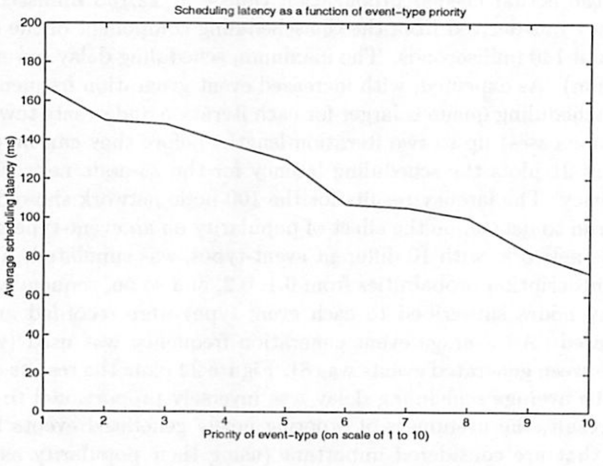

Figure 23: Scheduling latency as a function of event-type priority. For the 100 node network, scheduling latency decreases with increasing priority.

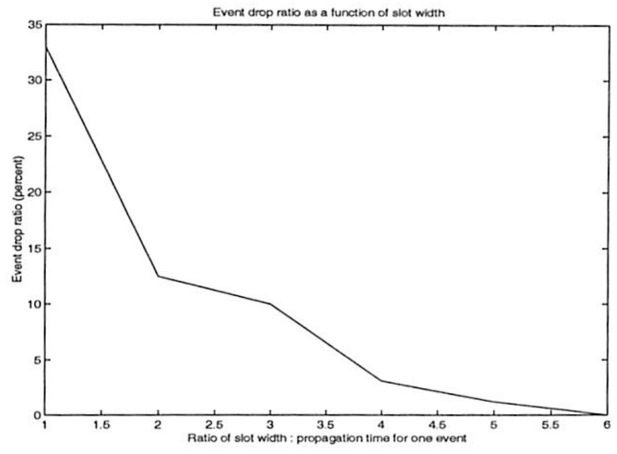

Figure 24: Event drop ratio (from medium contention) as a function of time-slot width. For the 100 node, fully subscribed graph, which has a maximum node degree of 5 (and an averge of 1.86), the drop ratio varies from 33.0 to 0 percent as the time-slot width increases.

places on a particular event-type? There may, for example, be a specific event-type which is subscribed to by only a handful of nodes, but who's timely dissemination is of utmost importance to a network application. In this case, priorities could be assigned to each event-type, and the scheduler would order newly generated events using these values. In general, there may be only one such event-type in the system, in which case priority and popularity could be used together, as the first and second sorting criteria, respectively (the special event-type could have a high priority, and every other event-type could have lower but equal priority, meaning that they would be scheduled against each other by popularity).

The next test simulated another 100 node network with 10 event-types as before, but where scheduling was done on the basis of priority. Each event-type had a subscription probability of 0.5 (and therefore roughly the same popularity in the network), but the priorities of the event-types ranged sequentially from 1, 2, ..., to 10. Other than scheduling by priority, the network was run under the same conditions as before. Figure 23 plots the scheduling latency results of the simulation, which verifies the expected trend that average scheduling latency decreases with increasing priority.

One of the greatest drawbacks to the protocol as it is currently implemented is the fact that time-slot widths must be several times longer than the actual one-hop transmission time for an event/MAC-frame. This is to allow for wireless channel medium contention, where multiple nodes (which are within range of each other) need to transmit during the same time-slot. Through random backoff and the use of RTS/CTS, collisions are kept to a minimum (although this extends the time-slot width even further). The number of nodes that can actually transmit successfully during a slot depends upon the ratio of the time of the time-slot and the propagation time of an event frame (this is referred to as $Ratio_{prop}$ in the previous analytical modeling section). How large this ratio must be be depends upon the network connectivity model of the graph. Generally, the average degree of the internal nodes gives a good indication what this ratio should be.

To illustrate, Figure 24 plots the ratio of "evicted" events (due to non-delivery from medium contention) as a function of this ratio for the previously simulated graph of 100 nodes. In this graph, the average degree of internal nodes is 1.86, and the maximum is 5. This graph is also fully subscribed (that is, every node subscribes to every event), so that medium contention will be maximized. Where the slot-width ratio is 1 (i.e., the time-slot width is the exact one-hop propagation width needed for one event), roughly 33% of all events are dropped in the system. This amount quickly drops to zero when the time-slot width expands to 6 times the minimum needed to propagate one event. With a time-slot ratio of 6, the propagation and scheduling latencies for

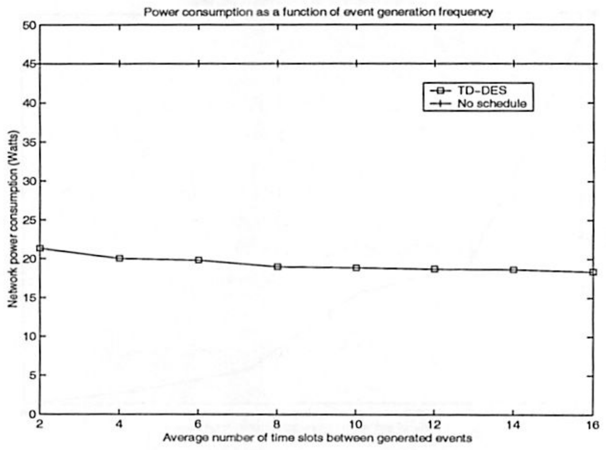

Figure 25: Power consumption: No schedule vs. TD-DES. Both simulated networks had the same 25 node topology with the same subscription profiles and stochastic event generation model. This time, speculative upstream time-slots were allocated and events could be generated at any node. Each network's total RF power consumption is given as a function of decreasing event generation frequency (given in average number of time-slots between events).

events will be severely effected.

For the initial implementation of the protocol, using wider time-slots was a stopgap solution to the problem of medium contention. In networks where node density is light, and the number of links per node is likewise small, using a time-slot ratio of perhaps 3 times the actual needed propagation width would be feasible. But clearly, for denser networks, where the degrees of internal nodes are higher, one cannot simply extend the time-slot width enough to eliminate all dropping of events – the effect on latency would be too severe. The next phase of this project will include incorporating a different approach to the problem of medium contention. Briefly, time-slot widths will be kept at the actual propagation time needed for an event/MAC frame, however, iterations will include unused time-slots at their ends to be used for extending the schedule as a result of failures to transmit events during their originally designated time-slots. As the protocol already allocates extra blank time-slots at the ends of iterations (to extend the schedule for events generated elsewhere besides the root), this approach should not be difficult to implement. However, it will certainly extend the length of the iteration, and determining the number of such slots to handle worst-case medium contention difficulties will be the problem to solve. The section ahead on future work (section 8) addresses this and other extensions to the protocol in more detail.

The next phase of the simulation involved extending the simulator to include speculative upstream time-slots for upstream event dissemination. Now, events could be generated at any node in the tree. If they were generated at nodes other than the root, they would be disseminated upstream as well as downstream. According to the random poisson event generation model, enough upstream slots were allocated per iteration to accomodate the expected number of generated events. Unlike downstream time-slots, upstream slots can be used for any event-type.

As before, a pair of 25-node networks was simulated, each network identical to the other in every way except for the fact that one used a TD-DES scheduler and the other one did not. Three event-types were used (each with a 0.5 subscription probability at each node), and the power consumption of both networks was again modeled as a function of decreasing event generation frequency (given as the poisson average distance in time-slots between generated events). Figure 25 plots the power consumption of both networks.

Refer back to Figure 19, which plotted the power consumption for both 25-node networks where

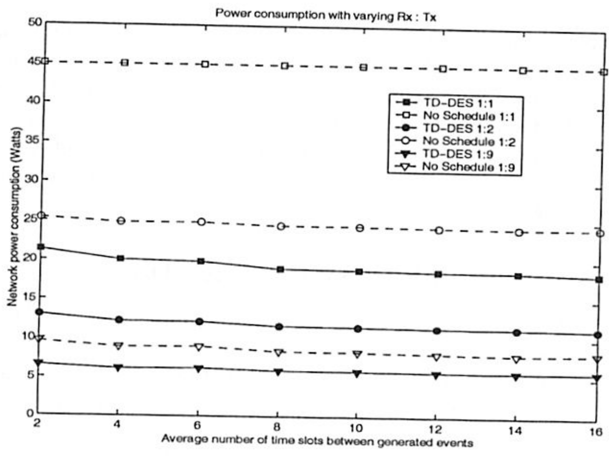

Figure 26: Power consumption: No schedule vs. TD-DES. Both simulated networks had the same 25 node topology with the same subscription profiles and stochastic event generation model. This time, speculative upstream time-slots were allocated and events could be generated at any node. Each network's total RF power consumption is given as a function of decreasing event generation frequency (given in average number of time-slots between events).

only downstream event dissemination was allowed. In Figure 22, the non-scheduled network still expends virtually the same power as before (roughly 45 milliWatts). However, at the highest event generation frequency, TD-DES spends roughly the same as it did in Figure 19, but the consumption does not drop rapidly with decreasing event generation as it did with only downstream dissemination. This is the intuitive result, as one would expect the newly allocated upstream time-slots to allow less opportunity for power savings. The reason for this is that speculative upstream receive slots must always be listened to, and therefore nodes waste energy in Rx mode during unused upstream slots. Nevertheless, TD-DES in this example expends only between 40 and 47% as much as the non-scheduled network.

To avoid dropping events due to medium contention, a time-slot ratio of 4 – $Ratio_{prop}$ – was used in the TD-DES network. Naturally, the propagation latency was roughly 4 times that of the non-scheduled network, and scheduling latency varied between approximately 210 and 380 milliseconds – essentially doubling what it was without the upstream time-slots. As before, scheduling latency decreased with decreasing event generation frequency. Clearly, the latency for TD-DES with full downstream and speculative upstream dissemination was suboptimal. However, for many sensor applications, the latency still falls well within acceptable bounds. It is our estimation, also, that this latency can be reduced by well over 50% when the revised method for dealing with medium contention is implemented in future versions of the protocol (see Section 8).

Figure 26 demonstrates the effect that varying the radio power output model can have on the efficiency of TD-DES compared to the non-scheduled network. Three radio models were used for both networks. As before, Tx and standby power outputs were held at 1.82 and 0.18 Watts, respectively. However, Rx power output varied from 1.80, to 0.90, to 0.20 Watts, for Rx:Tx ratios of 1:1, 1:2, and 1:9. The same 25-node network and subscription profile was used as before. For each network and radio model, power consumption was again measured as a function of decreasing event generation frequency.

TD-DES, in all cases, exhibits a decrease in power consumption as the event generation frequency decreases (although again, this decrease is not as marked as with the downstream-only network). For both networks, with decreasing Rx power output, the overall power output of the network also decreases. Where the Rx:Tx ratio is 1:1, the non-scheduled network has a stable network power consumption, unaffected by event generation frequency. Only when the ratio decreases does the non-scheduled network show a trend of decreasing power consumption as event generation frequency decreases. Of great interest is the fact that TD-DES loses its advantage over the non-scheduled network as Rx power decreases. When Rx power is 0.20 Watts (nearly equal to standby power), TD-DES expends 80% as much energy as the non-scheduled network. Clearly, as we also learned from the analytical modeling section, TD-DES is most effective when the differential between Rx and standby power output is significant.

6.2 Implementation and clock synchronization

Another phase in the protocol's development is to actually implement TD-DES on a number of wireless devices and empirically test the effectiveness of the protocol (measuring power-consumption, dissemination latency, and general network longevity statistics as before). One of the challenges to actually doing this is achieving correct clock synchronization, a subject that this paper has heretofore glossed over.

In our network tree, TD-DES calls for each child to have its clock synchronized with its parent's, and the level of precision attainable depends both on the method of synchronization and the hardware radio platform being used. For the TD-DES scheduler to work, no external synchronization with real-time (such as with a WWV radio signal) is neccessary, although particular applications running on the system may require a real-time/logical-time synchronization. As mentioned, children and parents need only have their internal clocks synchronized.

Current research has shown that clock synchronization methods for wireless networks, such as NTP [18] and *post-facto synchronization* [8], have achieved well under 100 microseconds of precision between synchronizing nodes. In fact, both of these methods, when used together, have demonostrated roughly a 1 microsecond level of precision though preliminary tests using a set of wired nodes [8].

For our purposes, we will assume that our nodes use the same model as defined by the 802.11 standard in infrastructure mode [2]. Clock synchronization here is achieved by the central node (access point) including a local time-stamp in its beacon frame. Upon receiving the beacon frame, all slave nodes adjust their internal 64-bit microsecond counters to match the time-stamp. Our system adapts this method by incorporating the time-stamp into an occasional *downstream control event.*

Using this method, imprecision between the clock counters of the master and slave nodes can occur from two sources: the variance of the time-critical path and in the respective clock speeds of parent and child (which causes drift). The time-critical path [19] is the time between when the master takes its time-stamp and the slave adjusts its local clock.

Recent protocols using the frame-level time-stamp method [19] have achieved maximal master/slave clock deviation of 150 microseconds (with the average being much lower). This assumes a minimum synchronization rate of one sync per second. For the analytical modeling and simulation, we safely assumed this lower bound for our protocol, giving an effective schedule granularity of 400 microseconds. Using the frame beacon method, time-slots could be partitioned on 400+ microsecond intervals across the broadcast schedule, and to compensate for clock deviation, devices could begin listening for events at least 150 microseconds prior to the actual beginning of a subscribed-to time-slot (when switching from either standby or Tx to Rx mode). For the simulation and analytical modeling, time-slots were actually milliseconds in length, which safely stayed within the expected maximum deviation between synchronized nodes.

7 Related Work

This project was inspired partly by the *broadcast disks* architecture [3], which proposes an architecture for asymmetric systems in which a server broadcasts a rotating schedule, or "disk," of data downstream to receiving clients. This is similar in TD-DES to the dissemination of events through a schedule from the root of the network tree downstream to all subscribing nodes. In broadcast disks, the schedule allocates actual data, where in TD-DES, time-slots for the transmission of data are allocated by the schedule. TD-DES also assumes a multi-hop environment, where the rotating schedule is further used by downstream nodes other than the root. Furthermore, TD-DES is designed for more-or-less symmetric data flow, up and down the tree.

In broadcast disks, the access patterns of the data to be broadcast (i.e., the client request rates for particular data items) govern how the schedule is created. For example, an item with a high access pattern will be multiplexed onto the schedule with a high frequency. This is similar to the speculative allocation of upstream time-slots in TD-DES, which is done based on network introspection of the upstream propagation frequencies for each event-type (section 4.8).

Even more directly applicable to this work is "broadcasting on air" paper by Imielinski et al. [11], where a wireless server broadcasts data items according to a temporal *directory*. The directory tells clients when to listen for particular data items. This is analagous to the downstream control event in TD-DES, which indicates the schedule for the upcoming iteration. A notable difference with their approach is the concern not with wireless devices wasting energy in Rx mode, but rather the CPUs of such devices wasting time in active mode. The work presented could also make use of the power-savings involved with switching the CPU of a wireless device in accordance with the TD-DES schedule (along with the radio tranceiver) – however, since TD-DES requires computation to determine the schedule of each iteration, and internal nodes in the network must shift the iteration schedules they receive from their parents, the answer to when and how often a node can toggle its CPU is another open research question entirely.

The publish/subscribe tree literature [10] for networking was another inspiration, and led to the initial design of TD-DES using a publish/subscribe network tree for data dissemination. This design is attractive for low-power wireless networks because it requires very little state and exchange of control information to maintain (compared to traditional wireless routing protocols). Additionally, it provides suitable means of disseminating information effectively for many real-world wireless networking applications, such as the *forest-fire-detecting* example given (section 2.1).

Directed diffusion [12], TAG [21], and other projects related to low-power sensor network design are of course research projects with the very same goals in mind: reducing the power-consumption of wireless networks.

Bluetooth [16], EC-MAC [14], other TDMA MAC protocols have provided a basis from which the idea of event-based time-division multiplexing arose, and these protocols, combined with aspects of asymmetric, scheduled data broadcasting and publish-subscribe network tree design, helped lead to the initial incarnation of TD-DES.

8 Conclusions and Future Work

Analytical modeling and simulation has shown the inherent benefit of using TD-DES as a datalink/ networking protocol for certain types of wireless networks, especially low-power sensor networks where each node is a small, battery-powered mote. It is important to note, however, that TD-DES in its present incarnation is unsuitable for the high-bandwidth, wireless applications which 802.11b, and especially 802.11g and 802.11a (both with 54 Mbps link capacities), have been designed. As time-slots are several times longer than the raw propagation time for an event-frame, it is impossible to saturate the bandwidth capacity of the physical radio links. However, many wireless applications, such as those *not* involving transmission of streaming audio or video, do not require large amounts of bandwidth, nor do they require constant monitoring of the wireless channel. In addition, any application which did need to saturate link capacity would be inherently power inefficient and not likely to be employed on energy-constrained hardware platforms.

TD-DES is efficient from a power-consumption perspective, but clearly suffers (in its first incarnation) generally from worse multi-hop dissemination latencies for generated events than does its non-scheduled counterpart network (which provides optimal latency and maximal power consumption). This is naturally because the latter lacks any sort of scheduling delay when propagating events. The only sort of delay that can occur is through node backoff from medium contention or re-transmission after collision, which are also problems with TD-DES. In addition, TD-DES uses slot-widths which are several times longer than the actual single-hop propagation delay for an event – depending on the the number of hops an event must travel, this can have a higher impact on the overall delay than the scheduling delay (although simulations have generally shown scheduling delay to be the largest contributor, except in extremely deep networks).

8.1 Mobility and reliability

We simulated static topologies in our first performance analysis of TD-DES. However, the tree-construction protocol allows nodes to detach from and join other nodes in the network through the transmission of *child_join* events (section 3.3). The ability of the protocol to adapt to topology changes should not be a problem, but it brings into question another concern: reliability. What if events destined for a particular node were never received because the node was briefly disconnected?

Reliability would largely involve the caching of events at subscribing nodes. Applying monotonically increasing sequence numbers to each event-type would be one way to handle the problem. If a node received an event with a sequence number, but hadn't received the previous sequence number, it might request its parent to resend the event. The idea would be to implement reliability in such a way that the power-efficiency of the system was not compromised. Mobility and reliability both will be the subject of future simulation and empirical study of the devloping TD-DES architecture.

8.2 Further simulation and empirical testing

A further comparison study would be of TD-DES with its counterpart node-based TDMA MAC protocols (such as Bluetooth), to determine any performance gain involved. We suspect that TD-DES will prove comparatively more efficient inasmuch as identical data is broadcast only *once* to all interested parties, whereas in Bluetooth or EC-MAC, the master node (or access point) potentially sends data redundantly to each slave node. Granted, for many applications (such as cellular communications), such dedicated individual access-point-to-end-node data streams are neccessary. It is our belief, however, that for the types of applications where common data is desired by multiple nodes that TD-DES will prove a more energy-efficient architecture.

Another step in the development will involve implementing TD-DES on a number of sensor motes which implement the TinyOS sensor node operating system. We currently have the motes, and plan to program them and use them for empirical performance studies in the near future.

8.3 Integrating collision avoidance

Currently, TD-DES relies on an underlying CSMA/CA, random backoff collision-avoidance and retransmission policy. In doing so, time-slots must be wider than the actual single-hop propagation delay for an event – in our implementation, they are two to six times wider. This, of course, is a waste of resources. Ideally, time-slots would be the same width as the propagation delay for maximum efficiency. However, if this were the case, the collision handling would have to be incorporated into the actual schedule itself – that is, extra slots would need to be allocated for retransmission. Presumably, extra slots could be allocated at the end of each iteration for the retransmission of events. This would complicate TD-DES, but would be worth the power-savings (and latency improvements) achieved from reducing slot widths.

Nodes which contended for a particular time-slot (such as sibling nodes using the same iteration schedule) would use the random backoff method of the MAC layer to decide who actually got to send during the time-slot. All others wishing to send would then wait until the end of the iteration, and transmit during these special "contention" slots (likewise using random backoff to determine the ordering). This would entail that downstream (or upstream) nodes listen to such "contention" slots, in some cases blindly (much in the same way that speculative uptream receive slots must always be listened to).

Optimizations here could be utilized. In the case that no event was transmitted during a determistic downstream slot, the receiving node would then know that the sending node had lost the backoff battle to a sibling (or some other node), and in this case would expect transmission during the contention slots. In addition, nodes listening to the contention slots might stop listening after the maximum backoff period expired (per the MAC protocol specification). When this period expired, this would imply that no other node had any more events to send.

Despite such techniques which allow nodes to selectively listen to such "contention" slots at the ends of iterations, the addition of such extra slots will have an affect on latency inasmuch as the iterations will be longer (in terms of number of slots). However, since time-slots will be greatly reduced in length, this approach should provide TD-DES with a significant overall improvement in terms of both latency and power.

8.4 Variable-length events and network scalability

Our current implementation assumes that events are actually single data frames – that is, each event fits into a single frame and requires one time-slot for propagation. What if events are longer in length than a single frame? This could easily be handled by a layer above TD-DES – perhaps the application itself – with a simple fragmentation and reassembly protocol. Experimenting with variable lengths of events will also be the subject of future work.

Finally, the idea of using TD-DES in a peer-to-peer network topology, rather than a tree-based topology, may also have its advantages. One of the disadvantages to the tree-based approach is that events must always be passed through the root – therefore, events certainly do not take shortest paths from the source to the destination nodes. It would be interesting if some sort of peer-to-peer, publish/subscribe event dissemination mechanism could be employed to further improve latency and power-consumption by reducing the path lengths of events.

Wireless mesh and grid topologies [5], and the data dissemination protocols which enable them have gained acceptance in recent years as alternative approaches to providing routing scalability in wireless networks. For the realization of extremeley dense wireless networks, potentially comprising thousands of motes, today's data dissemination techniques and protocols are largely insufficient. Projects such as SmartDust [15] have begun to address this future vision of pervasively distributed wireless devices, but much work has yet to be done.

TD-DES in its present form provides a novel approach to wireless networking and a basic working framework for power-efficient multi-hop wireless data dissemination. It is currently suitable for use with moderately dense networks, comprising up to hundreds of nodes. Extending the model for extremely high density networks, clearly the direction in which wireless technologies are heading, will be one of the primary goals in the future development of this ongoing project.

References

[1] http://www.ritron.com/sst_d.pdf.

[2] P 802.11; draft wireless LAN medium access control (MAC) and physical layer (PHY) specifications. Technical report, New York, 1997.

[3] Swarup Acharya, Rafael Alonso, Michael J. Franklin, and Stanley B. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 199–210, San Jose, California, 22–25 May 1995.

[4] Swarup Acharya, Michael J. Franklin, and Stanley B. Zdonik. Disseminating updates on broadcast disks. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, pages 354–365, Mumbai (Bombay), India, 3–6 September 1996. Morgan Kaufmann.

[5] Sang Ho Bae, Sung-Ju Lee, William Su, and Mario Gerla. The design, implementation, and performance evaluation of the on-demand multicast routing protocol in multihop wireless networks. *IEEE Network*, 14(1):70–77, January / February 2000.

[6] Kenneth C. Barr. Energy aware lossless data compression. Master's thesis, Massachusetts Institute of Technology, September 2002.

[7] Pravin Bhagwat and Charles Perkins. Highly dynamic destination-sequenced distance vector routing for mobile computers. *ACM SIGCOMM 1994*. http://www.cs.umd.edu/projects/mcml/papers.html.

[8] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*, pages 186–186, Los Alamitos, CA, April 23–27 2001. IEEE Computer Society.

[9] Chane L. Fullmer and J. J. Garcia-Luna-Aceves. FAMA-PJ: a channel access protocol for wireless LANs. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, pages 76–85, 1995.

[10] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. Technical report, Stanford University, Department of Computer Science, 2001.

[11] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. pages 25–36, 1994.

[12] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM-00)*, pages 56–67, N. Y., August 6–11 2000. ACM Press.

[13] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353 of *THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE*. Kluwer Academic Publishers, 1996.
http://athos.rutgers.edu/ imielins/book.html.

[14] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J-C. Chen. A survey of energy efficient network protocols for wireless networks. Technical report, School of EECS, Washington State University, Pullman, WA 99164, 2001.

[15] J. Kahn, R. Katz, and K. Pister. Next century challenges: mobile networking for smart dust. In *International Conference on Mobile Computing and Networking (MOBICOM '99)*, pages 271–278, August 1999.

[16] James Kardach. Bluetooth architecture overview. *Intel Technology Journal*, (Q2):7, May 2000.

[17] Phil Karn. Maca – a new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–40, 1990.

[18] Dave L. Mills. Network time protocol (version 1) specification and implementation. Network Working Group Request for Comments: 1059, July 1988.

[19] Michael Mock, Reiner Frings, Edgar Nett, and Spiro Trikaliotis. Clock synchronization for wireless local area networks. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 183–189, Stockholm, June 2000. IEEE Computer Society.

[20] Charles Perkins. Ad hoc on demand distance vector (aodv) routing. Technical report, Internet-Draft, draft-ietf-manet-aodv-00.txt. Work in Progress, November 1997.

[21] Joseph M. Hellerstein Samuel Madden, Michael J. Franklin. Tag: a tiny agregation service for ad-hoc sensor networks. In *5th Annual Symposium on Operating Systems design and Implementation (OSDI)*, December 2002.

[22] Marvin K. Simon, Jim K. Omura, Scholtz Robert A., and Barry K. Levitt. *Spread Spectrum Communications Handbook*. McGraw-Hill, New York, NY, USA, 1994. ISBN 0-07-057629-7.

[23] Amit Sinha and Anantha P. Chandrakasan. Operating system and algorithmic techniques for energy scalable wireless sensor networks. *Lecture Notes in Computer Science*, 1987:199–??, 2001.

[24] C.-K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems*. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2002.