MCC: a High-Throughput Multi-Channel Data Collection Protocol for Wireless Sensor Networks

Ying Chen and Bhaskar Krishnamachari
Department of Electrical Engineering, Viterbi School of Engineering
University of Southern California, Los Angeles, CA, 90089
Email: {chen2 and bkrishna}@usc.edu

Abstract—**We present the design and implementation of MCC, the first high-rate multi-channel time-scheduled protocol for fair, real-time data collection in Wireless Sensor Networks. MCC incorporates sophisticated mechanisms for balanced routing tree formation, multiple frequency channel allocation and globally synchronized TDMA scheduling. Through systematic experiments with real WSN hardware(Tmote Sky), we first identify the maximum possible throughput for many-to-one (convergecast) data collection in sensor networks. We show through a comprehensive experimental evaluation on the Tutornet experimental testbed that MCC can achieve close to the maximum possible network throughput for collection in wireless sensor networks with very low synchronization overhead. Compared to the Collection Tree Protocol (CTP), the state of the art collection protocol for WSN, we show that MCC offers 25-100% improvement in throughput. Being a time-scheduled protocol, MCC can be easily configured for energy-efficient operation yielding energy savings of 85 to 95% compared to always-on operation.**

I. Introduction

Networks of wireless sensors, each capable of a combination of sensing, computation, radio communication, possibly even actuation are envisioned to form a key component of the emerging "Internet of Things". Wireless Sensor Networks (WSNs) have been developed and put into use for automated data collection in many different scenarios, like environment monitoring, surveillance system, traffic monitoring, building automation, etc. [1].

In many real implementations [2] [22], it has been found that even if each sensor generates low rate data individually, due to the density of deployment and the many-to-one hop-by-hop traffic pattern, the amount of traffic close to sink is still too high, leading to high loss rate and poor throughput. This is even true for WSNs with small size and light traffic as shown in [3], [22]. For example, as mentioned in [22], in their initial experiments on a 9-node WSN for a traffic monitoring application, the yield of the network was only around 73%, preventing a larger scale deployment. This is also observed in [8], in which experiments are performed on USC Tutornet testbed using Tmote Sky devices. It is shown that with 40-byte packets, the per-source rate in a 40-node WSN is only about 0.5pkts/sec; thus the network throughput is only $6.4kbps$, about 2.56% (!) of the nominal $250kbps$ data rate of the IEEE 802.15.4 radio on Tmotes. *These observations illustrate that sensor networks are fundamentally throughput limited.* Even for small-medium scale networks with 100 or fewer nodes, it is essential to design protocols that can improve network and per-source throughput, and have a more efficient link utilization.

In recognition of this limitation, significant effort has been expended in recent years in the literature on developing and experimentally evaluating high throughput routing and rate-control protocols [5], [7]–[9], [23]. Nearly all of these efforts have been single-channel, CSMA-based protocols. While single-channel CSMA mechanisms are relatively easy to implement, their data rate is significantly limited by co-channel interference, particularly in dense deployments.

Interference could be reduced by allocating different channels to different links, improving the efficiency of bandwidth usage. However, time synchronization is needed to efficiently coordinate the communication of nodes operating on different frequency channels. This motivates us to design and implement a time-scheduled multiple channel high rate data collection protocol, that we refer to as Multi-Channel Collection (MCC). There has been relatively little prior work in this direction in sensor networks; the most closely related work being the recently developed PIP [24], which too provides time-scheduled multiple channel collection, but in an end-to-end connection-oriented fashion for bulk data transfer from one or two sensors at a time. Our effort, in contrast, allows for real-time and fair data collection potentially from all sensors in the network.

Our work is inspired and informed by the theoretical study in [18], which presents frequency-time scheduling and routing algorithms for fast convergecast. In particular, that work shows that when sufficient channels are available to eliminate interference, the bottleneck for throughput becomes the routing topology, and proposes a balanced routing tree formation algorithm that we utilize here. However, that work considers an idealized network that is evaluated purely through simulations. This can be misleading, as under such an ideal evaluation packet size has no impact on throughput, and the maximum achievable throughput is assumed to be the link rate allowed by the radio's physical (PHY) layer.

In order to obtain a systematic understanding of throughput performance, currently lacking in the literature, we first undertake a series of experiments on a representative widely-used WSN platform, the Tmote Sky with CC2420 radio. We study how throughput performance is affected by the packet size, node function (leaf / relay / sink), and the use of acknowledgements for reliability. Our experiments show that the maximum achievable throughput varies significantly with these parameters, and is well below the radio link rate provided by the PHY layer. Using the lessons learned from these experiments, we present in this work the first-ever real protocol implementation of fair throughput-maximizing multichannel convergecast, and demonstrate that it is able to achieve almost the highest-possible rate given the hardware limitations of a real sensor node.

Our MCC protocol has many components, including network connectivity determination, routing tree formation, channel allocation, time synchronization and time scheduling. Although there is a plethora of existing work on each of these individual components treated independently in the literature, our effort in this work is to combine them intelligently to develop a cohesive, high-performance protocol. We bring together existing algorithms from the literature, with modifications where needed, to engineer our implementation of this protocol on the Tmote Sky platform. We then evaluate experimentally the various components of the protocol, to understand parameter settings and their impact on performance.

Finally, we present the overall performance of the protocol in terms of the network throughput under various settings (the use of acknowledgements (ACK), network topology) using the USC Tutornet testbed [26] and show that it achieves close to the estimated maximum throughput. Moreover, we compare MCC with the collection tree protocol (CTP) [4], a state of the art collection protocol for WSN, and show that it typically yields a 100% improvement in total throughput. We also show that if nodes were turned off during un-scheduled times, the average energy utilization could be reduced by about 85 to 95% compared to an always-on network. Thus MCC is not only throughput optimal, it is also extremely energy efficient.

We summarize the key contributions of this work:

- The first systematic experimental analysis and modeling of the impact of packet size and node function on the maximum achievable throughput in real WSN hardware.
- Design, implementation, and testbed evaluation of MCC, the first high-rate multi-channel TDMA collection protocol for real-time data gathering in sensor networks. In engineering our protocol we build upon and integrate diverse mechanisms that have been developed and studied largely in isolation in prior work: network connectivity determination, balanced routing tree formation, channel allocation, time synchronization, and scheduling.
- The empirical demonstration that this protocol is able to provide close to the estimated maximum achievable network throughput.

The rest of the paper is organized as follows. First, we discuss prior work in section II. In Section III, we model, analyze and measure the maximum achievable throughput of convergecast. Based on the lessons we learned, in section IV, we present the design and implementation of MCC. In section V, we evaluate components of MCC on Tutornet testbed. Then we deploy MCC on a 30-node network and compare with CTP in section VI. Finally, we summarize our contributions in this work and present future work in section VII.

II. Related Work

Given more than a decade of research on wireless sensor networks, the number of papers written on convergecast and multi-channel protocols is too vast to enumerate comprehensively. Instead, we focus our review on key works, particularly those that have focused on real implementation and experimental validation over testbeds. Although it has not been designed for high-rate performance, the de-facto state of the art routing protocol for WSN today, which offers the best, reliable delivery performance, is the Collection Tree Protocol (CTP) [4]. We therefore use it as a baseline comparison for MCC. Alternate routing approaches that also show performance improvements are Arbutus [6], which offers improvements in load balancing and reliability, and the queue-aware dynamic routing mechanism BCP [5], which shows improvement in throughput as well as robustness to external interference.

Others have focused on avoiding congestion collapse and maintaining high rate delivery by using a rate control protocol on top of the routing mechanism. Examples include IFRC [7], and WRCP [8]. A centralized approach to rate control, RCRT [9], has been shown to yield even better rate performance. Flush [23] offers a robust, high-rate connection-oriented bulk transfer capability. All these protocols are single-channel protocols, and have been developed over CSMA, due to ease of implementation.

A comprehensive survey of multi-channel mechanisms for wireless networks (mostly for 802.11-based ad-hoc networks) can be found in [25]. MMSN [19] is the first multichannel MAC protocol especially designed for WSNs with devices having half-duplex single transceiver. There is a common broadcast channel, and nodes contend to access the channel on different unicast frequencies. The authors of [20] propose a multi-channel protocol with dynamic channel allocation by clustering a WSN. In each cluster, the header node needs to collect information and schedule for its member nodes. TMCP [16] is a tree-based channel allocation mechanism, in which the tree is partitioned into separate trees, each of which is allocated a separate channel (minimizing the need for synchronization). Other approaches to multi-channel MAC design include the cluster-based dynamic control-theoretic approach in [13], and MC-LMAC [14], which offers a distributed joint time and frequency scheduling mechanism. These schemes have all been evaluated primarily through simulations alone, or with limited testbed experiments (as in the case of TMCP). They are also not guaranteed to offer maximum throughput convergecast.

We now turn to practically implemented mechanisms for multichannel collection in wireless sensor networks that have been evaluated through extensive testbed experiments. Y-MAC [15] presents a multi-channel protocol for wireless sensor networks that is based on lightweight channel hopping. Nodes on a link hop to a new channel when traffic bursts occur, following a predetermined sequence. The focus of Y-MAC is to improve energy efficiency by reducing contention, rather than high rate performance, and it does not offer any guarantees in this regard. WRAP [10] uses multiple frequency channels with time synchronization. Data collection in WRAP is implemented using a token-passing scheme. Designed for highly dense deployments, it allows only one node in the network to be actively transmitting at any time. For this reason, while it does cut down drastically on interference and congestion, WRAP does not guarantee maximum rate performance in an arbitrary network.

The experimental work that is most relevant to our work is PIP [24]. This is a joint TDMA-FDMA based bulk-transfer protocol. In its basic form, it allows the sink to establish a connection with a particular sensor node and download data from that node at the highest rate possible. By keeping the sink occupied half the time (it must remain idle whenever the node one hop from it is in receive mode) it can achieve at least 50% of the maximum throughput. The authors of this work point out that to fully occupy the sink, with sufficient channels, it is possible to schedule two concurrent flows, keeping the sink fully occupied and thus achieving the maximum possible throughput, which is also our goal in this work. A key distinction, however, is that the MCC protocol we describe in this work is able to collect data fairly from all nodes in the network, rather than establishing connections only to one or two at a time. Thus, it is more suitable for real-time, fair data gathering applications. Also, MCC incorporates a balanced routing mechanism to ensure that the maximum rate is achievable. Moreover, we carefully evaluate the maximum possible network throughput with respect to packet size, network topology, use of ACK, etc., and demonstrate conclusively that this rate can be achieved in practice by our protocol.

III. Throughput Estimation

Although the maximum throughput may naively be considered to be simply the link rate of the underlying radio protocol, in practice the throughput may be even lower due to hardware limitations. In this section, we systematically study one widely used WSN hardware platform, the Tmote Sky, and understand its achievable rate performance with respect to different key parameters.

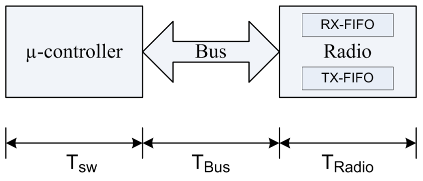

Fig. 1. A high level view of sensor node architecture

A. Basic Model

We show in Fig. 1 a high level view of a node's architecture (as our focus is on understanding throughput performance for the communication stack, we omit the sensor itself in this figure). The node has a micro-controller, a radio with 2 separate buffers (one for TX, another for RX), and a bus to copy data between them. The total time for transmission or reception of a single packet, T_{Packet}, can be written as a linear combination of a constant term and a term proportional to the packet size.

TABLE I
Parameters for the curve-fit of throughput estimation model

	α (ms)	β (ms/byte)
Sink, no ACK	1.79	0.062
Relay, no ACK	3.50	0.079
Leaf, no ACK	3.35	0.079
Sink, ACK	3.95	0.078
Relay, ACK	5.81	0.085
Leaf, ACK	5.52	0.079

$$T_{Packet} = \alpha + \beta \cdot Packet\_Size \tag{1}$$

The α pertains to per-packet software and radio latencies that are independent of the packet size. The rate term β depends on the CPU rate, the bus transfer rate, and the radio rate, and the degree of pipelining achieved between successive packets. While α is very software-specific, based on hardware specs for the Tmote Sky device, we can estimate that β lies between 0.04 and 0.09 ms / byte. The corresponding throughput can then be calculated as:

$$Throughput = \frac{Packet\_Size}{T_{Packet}} \tag{2}$$

B. Estimating the Maximum Convergecast Throughput

In practice, the measurement of T_{packet} and the corresponding throughput will be different depending on a nodes role and where it is being measured. Since we are concerned with a convergecast application, the three key node roles are sink, relay, and leaf. We therefore conduct experiments to estimate three kinds of throughput values: the receive rate of a sink node, the transmit rate of a leaf node, and the (transmit + receive) rate of a relay node. In these experiments we allocate time slots to nodes and use synchronized transmissions. To measure sink receive rate, we use a star-topology with the sink receiving packets from multiple nodes one hop away from it. To measure the leaf transmit rate, we use a simple two-node link with the transmitter continuously sending packets to the receiver. To measure the relay rate, we consider a linear topology with each node receiving on a different channel to avoid interference. We measure the relay rate of intermediate nodes as they alternate between sending and receiving packets. We vary the packet size from 10 bytes to 110 bytes and conduct the experiments with and without link-layer acknowledgement packets. For each setting we vary the slot length of packet transmissions to determine the maximum achievable rate and plot only those maximum rate points. The results we obtain are shown in Fig. 2 and Fig. 3. A striking observation is that the maximum achievable throughput observed in practice is at most about 100 kbps, well-below the ideal 250 kbps link rate provided by the CC2420 radio. *We find that the measurements show an excellent fit with the simple throughput model described in equations (1) and (2).* The corresponding parameters of the best-fit parameters for α and β are shown in table I. We see that the model fits the data very well, and that the β parameters are within the expected range.

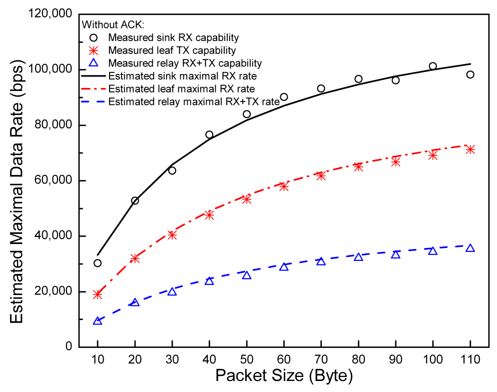

Fig. 2. Measured versus estimated throughput of Tmote Sky device for ACK-disabled case

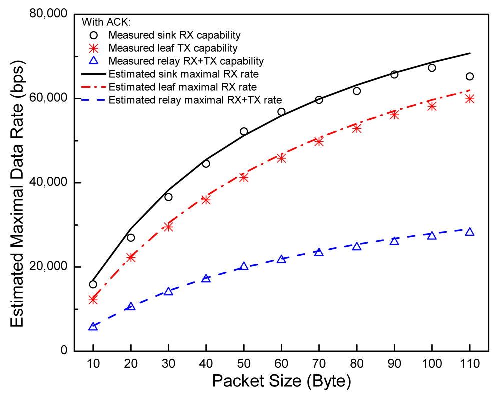

Fig. 3. Measured versus estimated throughput of Tmote Sky device for ACK-enabled case

C. Lessons

The modeling and throughput experiments summarized in Fig. 2 and Fig. 3 yield three key observations that inform our design of a high-throughput collection protocol:

- The throughput performance is very much a concave function of the packet size. To our knowledge, ours is the first work to systematically quantify using real WSN hardware how the throughput varies with the packet size.
- The throughput is higher without ACK than with ACK packets. This is intuitive, as there is additional overhead incurred in generating and waiting for acknowledgements. However, using ACK feedback will improve reliability.
- The throughput performance is clearly affected significantly by the role of a node. We find that the maximum sink receive rate is slightly higher than the maximum leaf transmission rate. This is due to the possibility of greater pipelining at the sink which is receiving data from multiple transmitters[1]. We also find that the relay rate, which is already halved as it must both receive and transmit, is further reduced due to the switching overhead between frequency channels. Thus, in a linear topology, the throughput bottleneck would be the relay node's maximum throughput. Note though that if we have even three different branches, assuming all transmissions are scheduled to avoid collisions, the sum rate of these branches exceeds the maximum sink receive rate. Thus the maximum collection rate of convergecast is bounded by the maximum sink receive rate. We will therefore compare the throughput achieved by our protocol with the maximum estimated sink throughput.

[1]At the transmitter, the application must wait to send the next packet from the microprocessor to the TX buffer on the radio until it is notified of the previous packet's transmission. From a receiver's perspective, the packet from one transmitter may overlap the time that the previous packet from a different transmitter is being copied from the RX buffer to the microprocessor, providing greater pipelining efficiency than in the transmitter.

IV. MCC Design and Implementation

Based on the observations of the previous section, we find that for convergecast the sink is generally the bottleneck. Conceptually, there are three possible sources of bottleneck: (1) interference, (2) relay node TX/RX rate and (3) sink RX rate. As argued in [18], we can mitigate (1) using multiple channels. We can address (2) by using suitably balanced routing topology. Therefore (3) becomes the fundamental limit on fair throughput. Our goal in this work is to develop a multi-channel collection (MCC) protocol for convergecast that can achieve this rate.

Our design of MCC includes network connectivity determination, routing tree formation, channel allocation, time synchronization, time scheduling and collection. We do not claim to have innovated each of these components indvidually, as these are difficult problems that have been well studied in the literature. Our contribution is instead in identifying, building upon, and integrating state of the art algorithms for each, modifying them as needed, to engineer a single cohesive high performance protocol. In the following, we present our implementation of MCC on the Tmote Sky platform with the TinyOS 2.x operating system.

A. Network Connectivity Determination

Initially, all nodes in the network start in the same channel. For a certain power level, each node broadcasts 100 messages. Whenever a node receives a message, it logs the sender's ID. Node i is j's neighbor only if j receives more than 90 messages from i, which implies a Packet Reception Ratio (PRR) $\geq 90\%$. Other links with $90\% > \text{PRR} > 0\%$ are considered as interfering links. Every node maintains a neighbor list by this blacklisting approach. After this step, we get a connectivity graph $G(V, E)$ and an interference graph $I(V, E')$ of the network for use in routing and scheduling.

B. Balanced Routing Tree Formation using CMS

Motivated by [18], we assume that interference can be eliminated by allocating multiple channels in a TDMA network. The available schedule length is lower-bounded by $max(2n_k - 1, N)$, where n_k is the maximum number of nodes on any subtree and N is the number of nodes in the network. If it is possible to have $2n_k - 1 < N$ in the tree construction, we can achieve N as the lower bound for time scheduling. But for an arbitrary graph G, can we construct a tree T on G such that $n_k < (N-1)/2$? This is defined as the "Capacitated Minimal Spanning Tree Problem" and is proven to be NP-complete [21]. For MCC, we use the Capacitated Minimal

Spanning (*CMS*) Tree heuristic presented in [18] to build a balanced routing tree, using the connectivity graph $G(V, E)$ obtained in the previous step.

C. Lightweight Time Synchronization (LTSP)

Synchronization is crucial for a multi-channel scheduling protocol. We implement a lightweight time synchronization protocol (LTSP) that is similar to FTSP [11] in that one-way synchronization is performed between every child and its parent on the tree, using MAC-layer time-stamps and linear regression over multiple packets. The reason we do not use FTSP code from TinyOS directly is that we piggy-back other configuration and control information for scheduling and channel allocation sent from the sink to the network nodes on the time-sync packets. All nodes are in the same channel at first. The root (sink) initiates the lightweight synchronization by broadcasting a group of LightTimeSync packets, which contains time information, to its children. After a child gets synced with its parent, it needs to propagate time synchronization and time scheduling information to its children if it is not a leaf node. Then the node gets ready for data collection.

D. Channel Allocation

In IEEE 802.15.4 radios such as the CC2420 radio on the Tmote Sky platform, the 2.4Ghz frequency is divided into 16 channels. From the routing tree topology, we know the intended transmissions. In a wireless network, interference would happen if an intended receiver is within the transmission range of a sender intended for other receiver. We define node i interferes with node j if i is in the transmission range of any j's child. Therefore, from the network connectivity and interference information we have obtained, a conflict graph C can be generated. An edge (i, j) in conflict graph C represents node i and j interfere with each other. We adopt receiver-based channel allocation, ensuring that different channels are allocated to nodes that have a link in C. Since graph coloring is NP-hard, we use the Welsh-Powell algorithm [27], a greedy heuristic that we find does well in practice.

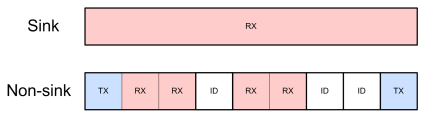

Fig. 4. TX and ID mode have one slot, RX mode has two slots

E. Time Scheduling

Once enough channels have been allocated so that interference can be completely eliminated, the time scheduling problem can be simplified to only consider the tree topology. We modify the algorithm indicated in [12] for our time scheduling. The key difference between our modified scheduling and that described in [12] is that while the original algorithm assumes same slot length for TX and RX, for MCC we double the length of RX slots in order to introduce a guard-time for continuous transmissions. The building block in our algorithm is a slot. A non-sink node has 3 modes: TX, RX and ID (idle). The TX and ID mode have duration of one slot-length and the RX mode has duration of 2*slot-length. The sink node is scheduled to remain in RX mode at all times. This is illustrated in Fig. 4. Nodes follow the same schedule pattern in each frame (corresponding to one round of data collection from all nodes).

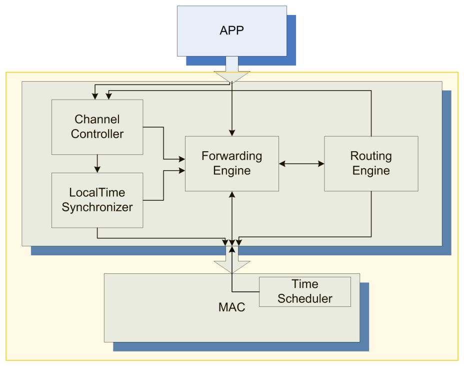

Fig. 5. Software Architecture for distributed part of MCC

F. Collection

At this point, a node has all its configuration information, e.g., next-hop, number of children, channel assignment and time schedule. It starts to collect data from its application layer or the network.

There are 5 major components in the distributed part of MCC as shown in Fig. 5:

- **Routing Engine**: Routing Engine obtains routing information, which is generated centrally, from sink through LTSP packets.
- **Forwarding Engine**: This is responsible for maintaining a queue of packets to transmit. The packets could be generated by the node's own application or received from its children. When ACK is enabled, it also provides retransmission mechanism.
- **Channel Controller**: This determines which channel a node uses for transmission and reception. This is also currently obtained from the sink via LTSP packets.
- **Lightweight Time Synchronizer**: This component implements LTSP.
- **Time Scheduler**: it maintains the time schedule, and performs the time calculations needed to inform the node when to transmit and when to be in receive mode.

MCC is compatible with TEP119 *Collection* [17], which defines interfaces, components, and semantics used by collection protocol in TinyOS 2.x.

V. Parameter Evaluation

In this section, we evaluate each building block in MCC: time synchronization, routing, channel allocation, and scheduling. All our experiments are conducted on the Tutornet testbed, on either all or some of a set of 30 nodes. Fig. 6 shows the testbed and the visualized network connectivity in terms of PRR for all pairwise links between the nodes that we use (numbered 1-30), at power levels 15 (medium) and 31 (high).

A. Overhead of Time Synchronization

LTSP is responsible for the time synchronization. It uses MAC layer timestamping and linear regression to provide precision of jiffy-level (1 jiffy $\sim 30.5\mu s$).

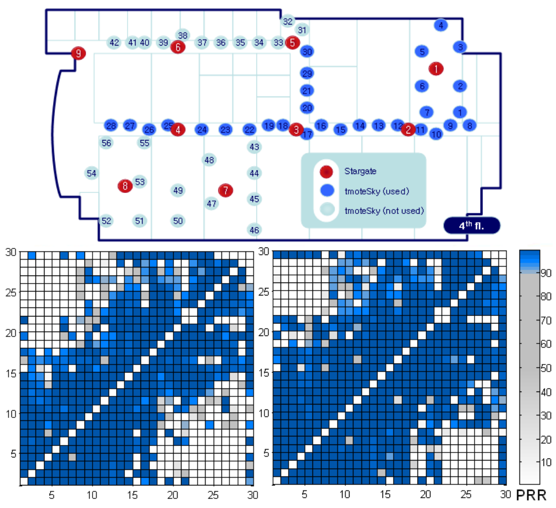
Fig. 6. Tutornet Testbed and Network Connectivity

To evaluate the performance of LTSP, we first test it on a star-topology. Fig. 7 shows the difference between real time on the parent and the approximate time calculated by the children. As shown in Fig. 7, when the sample set size (over which regression is performed to determine the skew) is 20, the error of time synchronization is less than 0.00854 jiffy/second. The frequency of synchronization needed can be estimated based on this number and the slot length. Because we provide an additional slot-length worth of guard-time in MCC, in principle, the protocol can handle synchronization error up to half that time. For example, if we consider using 40 bytes without ACK, the slot length would be 160 jiffies. If the sync error is less than 0.00854 jiffy/second, then it should take about 156 minutes before the error exceeds 80 jiffies, at which point the lack of synchronization would start to deteriorate protocol performance.

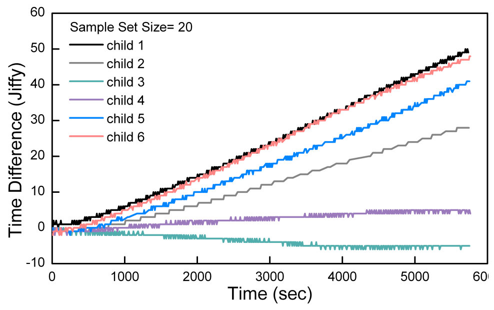

Fig. 7. LTSP: Time Synchronization Error

To verify this, we conduct a testbed experiment with a simple 5-node line topology, where we let the protocol run with a single synchronization event before the start of the data collection at time 0. The result is shown in Fig. 8. We see that, setting aside times when there is heavy external interference (which happens twice, around 90 minutes and 260 minutes into the experiments, identifiable through the reduction in the delivery ratio at those times), there is a consistent deterioration in the performance at 146 minutes. This is in remarkably close agreement with our prediction of 156 minutes. Even with a more conservative setting, we see that synchronization, need only be invoked once every two hours. As the synchronization process takes about 3 minutes, the overhead due to synchronization will be less than 3%.

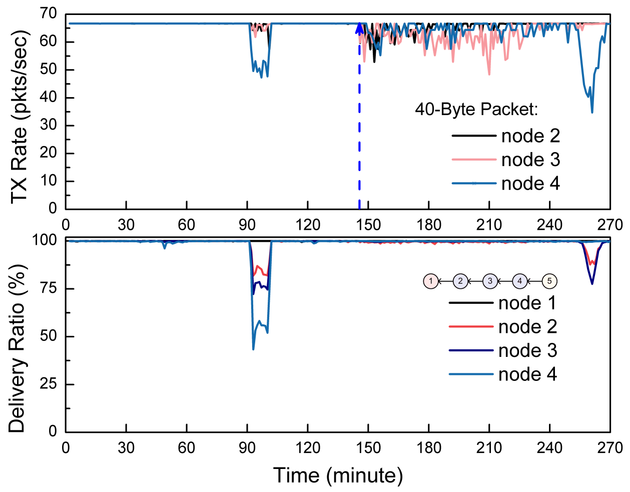

Fig. 8. 5-node line topology with 40-byte packets

B. Generation of a Balanced Routing Tree

Fig. 9 shows the maximum subtree size obtained using the CMS tree algorithm on the 30 node testbed for different power settings. We see that result satisfies the desired requirement of $n_k < (N-1)/2$ in all cases, so that the sink will be the bottle-neck. An ancillary benefit of this algorithm is that it yields a relatively shallow tree, with small maximum hop count, which is beneficial for reducing the synchronization time, delay, packet loss (in the case of no ACK). This is also illustrated in Fig. 9, which shows that the hop-count is always less than 5, and often just 2 hops at the medium to high transmit power settings.

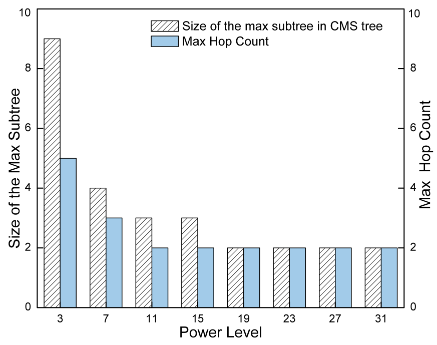

Fig. 9. Maximum subtree size and max hop count obtained by CMS Algorithm in a 30-node network

C. Channel Allocation

After the routing tree being constructed, we use a greedy algorithm to allocate channels. Fig. 10 shows the chromatic number, i.e., the number of frequency channels needed in order to completely remove the interference. We show it with different settings of power level for the full 30-node topology, after the CMS routing tree algorithm has been applied. We see that the chromatic number initially increases with the power level because of increasing interference. But when power level is high, and most nodes use one or two hops to reach the sink, the chromatic number decreases. We see that in all cases the number of channels required is less than the maximum of 16 that are available with the IEEE 802.15.4 radio that we use. In other tests that we do not present here, we find that even a network size of 55 nodes can be supported with less than 16 channels. We believe that the number of frequency channels used can be even further reduced by jointly optimizing the frequency allocation with the time-scheduling (this is part of our ongoing/future work).

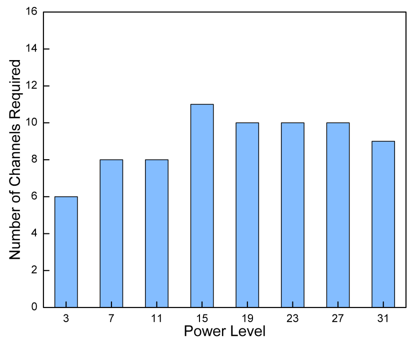

Fig. 10. Number of channels required to eliminate interference

D. Time Scheduling

The main parameter for time scheduling is the length of slots, which should be a function of the packet size. We identify the optimal schedule length for each packet size by conducting experiments where we vary the slot-length and observe its impact on the throughput. We conduct two sets of experiments; one to measure the impact of slot length for a star-topology, to understand the best slot-length for maximizing the rate of sink reception, and another to measure the impact of slot length for a linear topology (with each receiver on a different frequency channel to eliminate interference), to understand the best slot-length for maximizing the relay transmission rate (subject to the MCC design constraint that the reception slots are twice as long as the transmission slots).

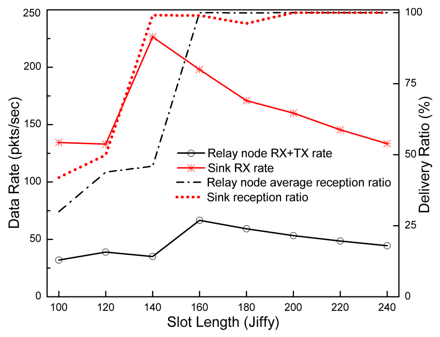

Fig. 11. The impact of time slot length on relay transmission and sink reception (packet size = 40 bytes)

A typical set of results, for the 40-byte, no-ACK case is shown in Fig. 11. We see that in both cases, a smaller slot-length causes packet losses, indicated by the delivery rate falling significantly below 100%, a larger slot-length results in poor utilization. Interestingly, we find that the maximum sink reception rate in the star-topology is achieved at a slot-length of 140 jiffies, while the maximum relay transmission rate in the linear topology achieves a maximum at a slot-length of 160 jiffies. This can be attributed to the additional channel-switching overhead incurred in the linear topology.

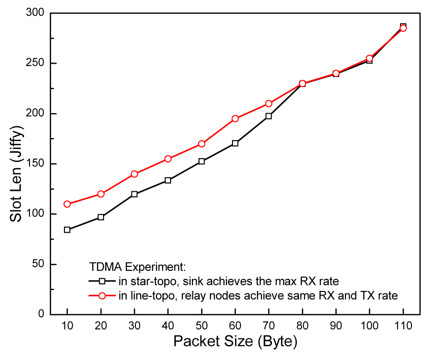

Fig. 12. Optimal slot lengths for sink and relay nodes in MCC

Fig. 12 shows the value of the best slot-length as the packet size is varied, for both the sink and relay nodes. Again, we see that the best slot-length for the relay node is generally higher until a packet size of about 80B. After this point, it appears that the overhead due to channel switching is negligible and the two cases require the same slot-length. In the MCC protocol implementation, we use the higher of the two curves, i.e. the curve for the relay nodes, to set the slot-length for all nodes in the network according to the corresponding packet size[2]. One implication of this plot is that for small packet sizes, the sink will not be fully utilized, resulting in some reduction of the throughput compared to the maximum possible sink reception rate. Note that these plots are both for the case without ACK. Similar curves are obtained in the case ACK is used, however, as expected, the best slot lengths in that case are higher to accommodate the transmission and reception of the acknowledgement packet.

VI. MCC Performance Evaluation

We have already evaluated all building blocks in MCC. Now we focus on measuring the collection throughput of MCC. From section III, we already have an estimation of the maximum possible throughput in the network. But the performance of MCC collection can be affected by packet size, power level, packet queue management, retransmission mechanism (ACK or no ACK) and channel allocation. In the following, we will evaluate MCC collection under different settings to see how these factors impact the performance, and compare it with the estimated maximum achievable throughput.

A. Backlogged MCC

MCC collection maintains a queue of packets, from both its own applications and the network. When a node is scheduled to transmit, if this queue if not empty, it will send the first packet in the queue. If the queue is empty, then the current TX slot is wasted. We first test MCC collection throughput when the queue is always backlogged so that all TX slots are busy and utilization of the time schedule is 100%.

[2]We assume in our evaluations, as in most WSN applications, that packets are all of the same length; in the rare case of an application where multiple packet sizes are utilized, the slot-length should be chosen to correspond to the maximum packet size, though this may result in lower utilization.

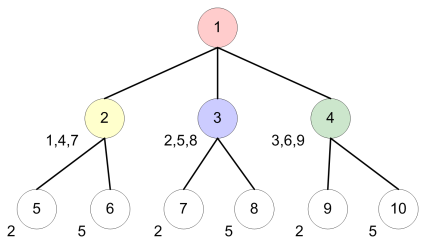

Fig. 13. A balanced routing tree with 10 nodes

We perform the test on a 10-node balanced tree on the testbed, with the topology as shown in Fig. 13. Full power is used to have the best link quality. In order to eliminate interference, each receiver has a different RX channel. We also label the scheduled TX slots of each node. Note that all these slot numbers are relative to its parent's first TX slot. The total frame length is 9. In this setting, we test MCC collection throughput with packet sizes of 40 bytes and 100 bytes. For each packet size, we measure the throughput with ACK enabled and disabled.

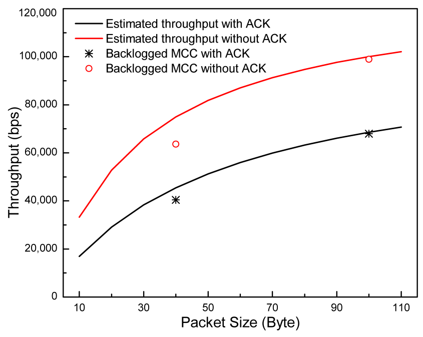

Fig. 14. Throughput of backlogged MCC compared with the expected throughput with or without ACK

The result is shown in Fig. 14. We compare MCC collection throughput with the expected throughput (obtained in section III) with and without ACK. We see that for the 100-byte packet without ACK, the throughput of MCC collection is $\sim 99kbps$, almost achieving the sink RX capability we measured in Sec. III. However, for the 40-byte packet, we see that the throughput is a bit lower than the estimated maximum sink RX rate. As mentioned in section V-D, this is due to the channel-switching overhead which causes the relay nodes to require a longer slot-length than the sink-rate-maximizing slot-length. A similar trend is observed for backlogged MCC with ACK. This figure shows that MCC is able to achieve close to the maximum achievable throughput so long as nodes always have data to send at each slot.

B. MCC Performance

In Sec. VI-A, we bypassed the queue management. Now we assume the application generates a constant data rate equal to the maximum that can be transported by the MCC protocol.

In the same 10-node balanced tree setting described in the previous sub-section, we compare MCC collection with backlogged MCC and CTP, the state of the art single-channel collection protocol for wireless sensor networks. For CTP, we empirically determine and present the maximum achievable fair rate. Again, full power is used, and we test 40-byte and

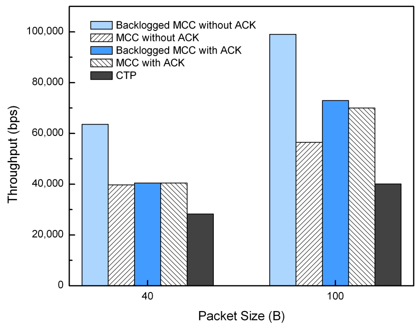

Fig. 15. Network throughput comparison of backlogged MCC, MCC and CTP in a 10-node network

100-byte packets, with and without ACK. When we enable ACK, if a packet is not acknowledged, MCC is configured to retransmit it up to a maximum of 3 times.

Fig. 15 shows the result. The performance of backlogged MCC provides an upper bound of the throughput performance. We find that MCC with ACK enabled can achieve almost the same throughput as backlogged one. It has a substantial improvement over CTP. But for MCC collection without ACK, throughput is much lower than backlogged MCC without ACK. This can be attributed to the fact that without ACK, there is a greater loss of packets sourced from the rear of the network. When these packets do not make it to an intermediate node, the corresponding transmission opportunities have to be wasted, resulting in a reduced throughput. In future work, we plan to examine whether a redundant transmission policy can be used for ACK-disable case to compensate for the link losses, albeit at the expense of greater energy expenditure.

Finally, we deploy MCC collection on the full 30-node testbed. We consider two power levels, one medium, and one high. In these experiments, for a fair comparison with CTP, we enable ACK and test for packet size of 40 bytes and 100 bytes. This throughput comparison is shown in Fig. 16.

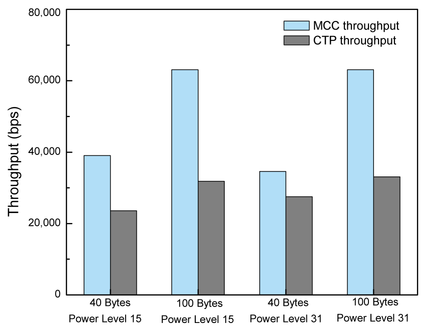

Fig. 16. Network throughput comparison of MCC and CTP in a 30-node network

In this medium-sized network, we see that MCC collection can achieve between 25 to as much as 100% (2X) improvement over CTP in terms of throughput, with the gains being close to 100% for large-size packets (where, as we have shown, MCC is able to achieve the maximum possible sink throughput).

Due to the TDMA feature, each node is aware of its TX/RX time schedule. Although we have not implemented this in code yet, MCC can be easily configured to turn off the radio

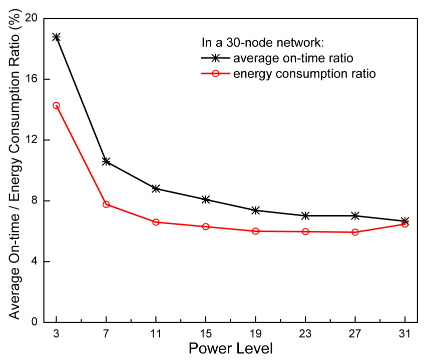

Fig. 17. Average on-time of a 30-node network

when a node is idle. Using energy consumption numbers for reception and transmission of the CC2420 radio, we perform an analysis of the energy savings that can be obtained in MCC by turning off the radio during idle times. In Fig. 17, we show how the average on-time ratio (ratio of number of on-times compared to total time in each frame) and the average energy consumption ratio vary with the power level (the ratio of the energy consumed with MCC to the energy consumed when all nodes are always one). Note that because we take ratios in this plot, the curves are independent of packet size. Overall, we find that MCC can yield an energy reduction ranging from 85 to 95%. We see that the total energy is minimized at a relatively high power level of 27, this is because initially the higher power level results in more nodes talking directly to the sink and thus not needing to be on to receive or relay packets, but eventually the additional cost of increased transmit power catches up.

VII. Conclusion and Future Work

We have proposed and implemented MCC, a new high-rate, multi-channel time-scheduled protocol for convergecast data collection in wireless sensor networks. We first evaluated the maximum sink receive rate, which is the best possible in a single-sink network, and showed empirically through testbed experiments that MCC can achieve a rate close to this. We also observe that the overhead for time-synchronization is quite small (the time required for synchronization is less than 3% of the stable running time of the protocol). We showed that compared to CTP, the state of the art single channel collection protocol, MCC is able to provide between 25-100% improvement in throughput. Finally, we also showed that because it is a time-scheduled protocol, nodes can be turned off during unscheduled idle times to save energy. Our calculations show that it is able to save between 85-95% energy consumption compared to the always-on case.

There are a number of directions for future work. Straightforward extensions include considering multiple sinks and sinks equipped with multiple transceivers, which would both further increase the network throughput of MCC. We are interested in developing a joint time-frequency scheduling mechanism to further reduce the number of channels required in MCC. We are also interested in exploring distributed implementations of the balanced routing as well as channel-time scheduling components of MCC. It would also be of interest to consider extensions of MCC that can handle bursty, dynamic traffic patterns.

References

[1] B. Krishnamachari, *Networking Wireless Sensors*, Cambridge University Press, 2005.

[2] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, “Deploying a Wireless Sensor Network on an Active Volcano”, *IEEE Internet Computing, Mar/Apr 2006*.

[3] G. S. Ahn, E. Miluzzo, A. T. Campbell, S. G. Hong, and F. Cuomo, “Funneling-MAC: A Localized, Sink-Oriented MAC For Boosting Fidelity in Sensor Networks”, *ACM SenSys 2006*.

[4] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, “Collection Tree Protocol”, *ACM SenSys 2009*.

[5] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, “Routing Without Routes: The Backpressure Collection Protocol”, *ACM/IEEE IPSN 2010*.

[6] D. Puccinelli and M. Haenggi, “Reliable Data Delivery in Large-Scale Low-Power Sensor Networks”, *ACM Transactions on Sensor Networks, July 2010*.

[7] S. Rangwala, R. Gummadi, R. Govindan and K. Psounis, “Interference-Aware Fair Rate Control in Wireless Sensor Networks”, *ACM SIGCOMM 2006*.

[8] A. Sridharan and B. Krishnamachari, “Explicit and Precise Rate Control for Wireless Sensor Networks”, *ACM SenSys 2009*.

[9] J. Paek, R. Govindan, “RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks”, *ACM SenSys 2007*.

[10] C. J. Liang, J. Liu, and L.Q. Luo, A. Terzis, and F. Zhao, “RACNet: A High-Fidelity Data Center Sensing Network”, *ACM SenSys 2009*.

[11] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, “The flooding time synchronization protocol”, *ACM SenSys 2004*.

[12] C. Florens, M. Franceschetti, and R.J. McEliece, “Lower bounds on data collection time in sensory networks”, *IEEE Journal on Selected Areas in Communications, 22 (6) 2004*.

[13] H. Le, D. Henriksson, and T. Abdelzaher, “A Practical Multi-Channel Medium Access Control Protocol for Wireless Sensor Networks”, *ACM/IEEE IPSN 2008*.

[14] O. D. Incel, L. van Hoesel, P. Jansen and P. Havinga, “MC-LMAC: A Multi-Channel MAC Protocol for Wireless Sensor Networks”, *Elsevier Ad Hoc Networks Journal, 2010*.

[15] Y. Kim, H. Shin, and H. Cha, “Y-MAC: An Energy-Efficient Multi-channel MAC Protocol for Dense Wireless Sensor Networks”, *ACM/IEEE IPSN 2008*.

[16] Y. Wu, J. A. Stankovic, T. He, J. Lu, and S. Lin, “Realistic and Efficient Multi-Channel Communications in Wireless Sensor Networks”, *IEEE INFOCOM 2008*.

[17] http://www.tinyos.net/tinyos-2.x/doc/html/tep119.html

[18] O. D. Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi, “Fast Data Collection in Tree-Based Wireless Sensor Networks”, *IEEE Transactions on Mobile Computing, 2011*.

[19] G. Zhou, C. Huang, T. Yan, T. He, and J. A. Stankovic, “MMSN: Multi-Frequency Media Access Control for Wireless Sensor Networks”, *IEEE INFOCOM 2006*.

[20] X. Chen, P. Han, Q. He, S. Tu, and Z. Chen, “A Multi-Channel MAC Protocol for Wireless Sensor Networks”, *IEEE CIT 2006*.

[21] C. H. Papadimitriou, “The complexity of the capacitated tree problem”, *Networks, vol. 8, no. 3, 1978*.

[22] M. Bathula, M. Ramezanali, I. Pradhan, N. Patel, J. Gotschall, and N. Sridhar, “A sensor network system for measuring traffic in short-term construction work zones”, *IEEE DCOSS 2009*.

[23] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica, “Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks”, *ACM SenSys 2007*.

[24] B. Raman, K. Chebrolu, S. Bijwe, V. Gabale, “PIP: A Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer”, *ACM Sensys 2010*.

[25] O. D. Incel, *Multi-Channel Wireless Sensor Networks: Protocols, Design and Evaluation*, Ph.D. Thesis, University of Twente, Netherlands, 2009.

[26] Embedded Networks Laboratory. http://testbed.usc.edu.

[27] D.J.A. Welsh, M.B. Powell, “An upper bound for the chromatic number of a graph and its application to timetabling problems”, *The Computer Journal,10(1967)*.