LogA: Concurrent Medium Access Control through Time Log Analysis in Sensor Networks

Zhaoliang Zhang†‡‡, Haiming Chen†, Tingpei Huang†‡‡, Li Cui†
†Institute of Computing Technology, Chinese Academy of Sciences
‡Graduate School of the Chinese Academy of Sciences, China
Email: {zhangzhaoliang, chenhaiming, huangtingpei, lcui}@ict.ac.cn

Abstract—This paper focuses on the design of high-throughput MAC for data-intensive sensor networks with unpredictable traffic. We propose LogA, a reactive concurrent MAC protocol that increases transmission concurrency based on time log analysis. Each node reactively and passively learns the interference relationship by analyzing the start transmission time and the end transmission time of packet blocks. Then, the learned interference relationship is exploited to improve the probability of beneficial concurrent transmissions of nodes that are within the interference range of each other. LogA has two salient features. First, it is passive and does not need network downtime to build interference relationship. Second, it is reactive and works only when traffic are generated. LogA has been implemented in Tinyos-2.1 and extensively evaluated in TOSSIM, the simulator of sensor networks. Experimental result shows that LogA outperforms the traditional CSMA protocol and an existing reactive concurrent MAC in terms of throughput, delivery latency, and energy consumption.

I. INTRODUCTION

Recent years have witnessed the deployment of many data-intensive Wireless Sensor Networks (WSNs) for various applications. Notable applications of this kind are environmental monitoring [1], fire detection [2], and surveillance applications using imaging/acoustic sensors. These applications require the transmission of bulk sensed data to the sink. For instance, volcano monitoring [1] needs sensors to sample seismoacoustic data at 100Hz in order to enable valid analysis of volcano activities. When lots of nodes transmit bulk data to the sink via multiple hops, the interference amongst nodes becomes severe. Such interference leads to persistent congestion and drastically reduces throughput, making the data transfer throughput a critical bottleneck in the system [1]. Hence, a high-throughput Media Access Control (MAC) protocol is crucial to address this challenge and enable such data-intensive applications.

Traditional CSMA-based protocols, e.g., S-MAC [3], B-MAC [4], resolve this problem by allowing only one of the interfering nodes to transmit, leading to low throughput. Recently, concurrent MAC protocols C-MAC [5] and OPC [6] are proposed to exploit transmission concurrency to improve system throughput. These protocols periodically measure the Received Signal Strength Indicator (RSSI) of each link. Based on RSSI, they then build the empirical model between Packet Reception Ratio (PRR) and Signal-to-Interference-plus-Noise-Ratio (SINR) to make transmission decisions. C-MAC [5] and OPC [6] are effective in applications with periodic or predictable traffic. However, in networks with unpredictable traffic, bulk traffic may be generated at any time. Nodes may have bulk data to transfer only when interested events happened [1], or users issued queries [2]. Existing concurrent MAC protocols C-MAC [5] and OPC [6] are not applicable in such networks with unpredictable traffic. First, they require the network to stop operation in order to measure the RSSI of links. This may affect the normal operation of WSNs, especially in time-critical networks. Second, they work in a proactive manner. They need to collect RSSI measurements and build the PRR-SINR model before traffic delivery. However, the relationship between PRR and SINR has shown to be varying with time, which may change in hours [7]. The empirical model should be repetitively updated even if there is no traffic. Repetitive proactive measurement incurs large measurement overhead and shorten the lifetime of WSNs.

In this paper, we propose a high-throughput concurrent MAC protocol, called LogA, for networks with unpredictable traffic. LogA overcomes the above drawbacks of existing protocols. It does not need network downtime and works in a reactive and passive manner. The key novelty of LogA lies in that it reactively and passively learns the interference relationship amongst nodes and utilizes this information to enable beneficial concurrent transmissions. Specifically, LogA transmits data packets in blocks that is composed of multiple packets of equal length. Each packet in blocks is transmitted without Clear Channel Assessment (CCA). The start time and end time of each block transmission, called time log, are recorded and exchanged. By analyzing the overlap of time logs, nodes infer the PRR of links under interference. In turn, the inferred PRR is used to estimate system throughput and make transmission decisions. We evaluate LogA extensively in TOSSIM. The experimental results show that LogA can improve system throughput by about 60% compared to the traditional CSMA protocol. The throughput gain is about 55% when LogA is compared with CMAP, which is a reactive concurrent MAC originally designed for 802.11.

The rest of the paper is organized as follow. In section II, we summarize the related work. In section III, we first give an overview of LogA and then discuss the design details of LogA. Experimental results are presented in section IV. In section V, we conclude the paper.

II. RELATED WORK

C-MAC [5] and OPC [6] are already designed to improve transmission concurrency. Both C-MAC and OPC are based
on empirical models between PRR and SINR. They are very effective in improving throughput when bulk traffic is periodic or predictable. However, the requirement of downtime together with proactive and periodic model building make them unsuitable for networks with unpredictable traffic. In contrast, LogA reacts to the interference relationship by time log analysis and does not need downtime measurement. Thus, it can work well in networks with unpredictable traffic.

Several bulk data streaming protocols are also proposed to improve system throughput. PIP [8] skillfully combines multi-channel, TDMA, and conditional immediate transmission techniques. However, PIP is designed for single path data transmission. Burst forwarding [9] optimizes the retransmission technique in lossy networks. Also, it can only work when at most several cross paths are active. Thus, all these protocols are not generic MAC protocols. LogA differs from these protocols in that it is generic and can work in scenarios of various traffic patterns.

Power control [10] and multi-channel communication [11] are also proposed to improve spatial reuse and throughput. But these techniques are orthogonal to LogA, and thus can be combined with LogA to further improve system throughput.

In 802.11 networks, CMAP [12] exploits the conflict relationship between link pairs to increase concurrent transmission. It is effective in 802.11 networks with rate adaptation. LogA is distinguished from CMAP. First, LogA considers the PRR of links under various interference, while CMAP only focuses on the binary conflict relationship between link pairs. Second, LogA infers the PRR of links by time log analysis, but CMAP identifies conflict link pairs by receiving the packet tailer of the interferer.

III. DESIGN OF LOGA

In this section, we present the design of LogA. We first give a brief overview of LogA, and then discuss each component in details. Since the time of most WSNs are synchronized, we assume all nodes have a common notion of time. To simplify the discussion, we assume all nodes transmit data packet of equal length. However, LogA can be extended to the cases of unsynchronized networks and unequal length packets.

A. Overview

The major goal of LogA is to achieve high system throughput. In order to achieve this goal, LogA reacts to the interference relationship amongst transmitting nodes, and allows interfering nodes to transmit concurrently.

LogA transmits data packets in blocks. A block is composed of multiple data packets. When a block is pending for transmission, the node assesses the channel and collects the information of the ongoing flows. If there are ongoing flows, the node then estimates the throughput gain of transmitting concurrently with the ongoing flows based on the interference vector (i-vector) table. If the throughput gain is not sufficiently large, the node retries after a delay. Otherwise, the node transmits the block immediately. The packets in the block are transmitted successively without CCA. In the process of transmission, the start transmission time of the first packet and end transmission time of the last packet are recorded as start time and end time of block transmission respectively. These two points of time are referred to as a time log. Nodes exchange time logs with its neighbors to infer interference relationship. Moreover, in the initial phase, nodes also rapidly estimate interference relationship based only on the collected information of the snooped flows and its own time log. In such a way, nodes reactively and passively infer the interference relationship by analyzing the time logs of block transmission. Then, this information is maintained in the i-vector table and used to estimate throughput gain.

As shown in Fig. 1, LogA consists of the follow components: 1) Interference Vector (i-vector) Table Maintenance which maintains the information that maps links to their PRR under certain interference; 2) Time Log Analysis which infers the interference relationship amongst concurrent transmitters; 3) Transmission Decision which estimates throughput of concurrently transmitting nodes and makes transmission decisions accordingly; 4) Transmission Control which coordinates the operation of other components.

B. Interference Vector Table Maintenance

The interference vector table is a data structure that maintains a list of self-inferred i-vectors and i-vectors received from neighbors. It is used to predict throughput and make transmission decisions. Each i-vector characterizes the PRR of a link when the sender of the link transmits concurrently with some interferers. Each i-vector has the form: \((IID, Link, PRR, N_{sample})\), where \(IID\) is the identity (ID) set of interferers that transmit concurrently with the sender of the link and \(N_{sample}\) is the number of samples used to estimate the PRR. For instance, Fig. 1 illustrates a situation consisting of three concurrent transmitters. An i-vector \(\langle s_0, r_0, 0.7, 15 \rangle\) means that the PRR of link \(<s_0, r_0>\) is 0.7 when \(s_0\) and \(r_0\) transmit concurrently. Moreover, the PRR is estimated based on 15 samples. To keep the table within proper size, we limit the cardinality of \(IID\), i.e. the size of the set, to be no larger than \(C_{max}\).

Nodes exchange its self-inferred i-vectors with neighbors. There are three types of i-vectors. Type I i-vectors are inferred by the node via precise analysis of time log. Type II i-vectors are the results of rapid time log analysis, the \(N_{sample}\) of which is 0. Type III i-vectors are received from neighbors, the \(N_{sample}\) of which is also 0.
Initially, the i-vector table is empty. Each node maintains the table in the following way. For a new i-vector \((iid, link, prr_i, n_{sample_i})\), if there exists no i-vector in the table that matches iid and link, LogA simply adds the new i-vector to the table. Otherwise, LogA updates the existing i-vector using different rules. Suppose the PRR in the existing i-vector is \(P_{RR_{i-1}}\) and the number of sample is \(N_{sample_{i-1}}\). For a new i-vector of type I and II, the new \(P_{RR}\) is set to

\[
P_{RR_i} = \frac{P_{RR_{i-1}} \cdot N_{sample_{i-1}} + prr_i \cdot n_{sample_i}}{N_{sample_{i-1}} + n_{sample_i}}.
\]

And the new \(N_{sample_i}\) is set to

\[
N_{sample_i} = N_{sample_{i-1}} + n_{sample_i}.
\]

For a new i-vector of type III, LogA simply sets \(P_{RR_i}\) to \(prr_i\), and \(N_{sample_i}\) to \(n_{sample_i}\) that is 0.

By exchanging i-vectors, the i-vector table gradually accumulates i-vectors of links originated from other nodes. Moreover, i-vector table periodically times out stale i-vectors to accommodate changing channel conditions. In our experiments, an i-vector can be removed at 60s after its last update.

### C. Time Log Analysis

We define a time log as a pair \(TL=<t_0, t_1>\), where \(t_0\) and \(t_1\) are the start time and end time of a block transmission respectively. Time log analysis aims at inferring i-vectors.

LogA transmits data packets in blocks and data packets have equal length. Except the first packet, all packets in a block are transmitted immediately after the previous packet without CCA. In this way, nodes have little variation in the interval from transmitting the first bit of a packet to transmitting the first bit of the next packet in a block, referred to as packet transmission interval. This interval is measured and used as a parameter of LogA. Moreover, the time from transmitting the last bit of a packet to transmitting the first bit of next packet in the block is small compared to transmission time of a packet of adequate length. This is because the radio in WSN adopts fix low data rate, e.g., 19,2kpbs for CC1000 and 250kpbs for CC2420, thus the transmission time of a packet is fix. Besides, the handling time of sending related interrupt is mostly small, e.g., less than a few microsecond (\(\mu\)s) for Telosb motes.

Each sender \(s_0\) in LogA records the time log of each block transmission. It also gets time logs of concurrent transmitters by exchanging time logs in an efficient manner or by approximation that will be explained later. When some nodes transmit concurrently with the sender \(s_0\), the transmission time of their blocks overlaps. Hence, by comparing these time logs, the sender \(s_0\) can calculate the start overlap time and end overlap time of its block with other overlapping blocks. Since the packet transmission interval is known, the sender \(s_0\) can precisely convert the overlap time to packet indices in a block. Thus, the sender \(s_0\) learns that whether a packet in its blocks is interfered and which nodes interfere the packet. In addition, LogA requires the receiver \(r_0\) to send a block-level ACK with a bitmap that indicates whether a packet in the block is received. With the interferer’s ID of packets and the bitmap in ACK, the sender \(s_0\) can easily estimate the PRR of the receiver \(r_0\) when different interferers transmit concurrently with \(s_0\).

Algorithm 1 gives the pseudocode for time log analysis. The input parameter \(TL_0\) is a time log of the node itself and \(TL_i (1 \leq i \leq M)\) are those time logs of neighbors that overlap time of its block with other overlapping blocks. Since the packet transmission interval is known, the sender \(s_0\) can precisely convert the overlap time to packet indices in a block. Thus, the sender \(s_0\) learns that whether a packet in its blocks is interfered and which nodes interfere the packet. In addition, LogA requires the receiver \(r_0\) to send a block-level ACK with a bitmap that indicates whether a packet in the block is received. With the interferer’s ID of packets and the bitmap in ACK, the sender \(s_0\) can easily estimate the PRR of the receiver \(r_0\) when different interferers transmit concurrently with \(s_0\).
of the algorithm is interference and update the i-vector table. The time complexity per retries after a random delay: 1) Zero or more than conditions is satisfied, the node gives up transmission and 2) The intended receiver is active in the overhears the k active links, \(<s_i, r_i>\), where 1\( \leq \)k. Let \(T = \{s_i | 1 \leq i \leq k\}\). For 1\( \leq \)k, node s0 estimates the current PRR of the link \(<s_i, r_i>\), denoted by \(P_{\text{RR}}(T \setminus \{s_i\}, <s_i, r_i>)\), by looking up an i-vector \((T \setminus \{s_i\}, <s_i, r_i>, *, *)\) in the i-vector table. If an i-vector matches, \(P_{\text{RR}}(T \setminus \{s_i\}, <s_i, r_i>)\) is estimated to be the PRR element in the matched i-vector. If no i-vector matches, \(P_{\text{RR}}(T \setminus \{s_i\}, <s_i, r_i>)\) is estimated to be 1.0. In this case, the nodes in T may have not transmitted concurrently recently, and thus no i-vector about \(P_{\text{RR}}\) of \(<s_i, r_i>\) under the interference of T is estimated. However, since the concurrency occurred before s0’s decision, there is a high chance that i-vectors about this concurrency will be derived soon. With the estimated \(P_{\text{RR}}(T \setminus \{s_i\}, <s_i, r_i>)\), node s0 estimates the throughput achieved by node set T to be

\[
TH_T = \sum_{s_i \in T} P_{\text{RR}}(T \setminus \{s_i\}, <s_i, r_i>)
\]

If s0 transmits, the PRR of the k active links will change. We denote the PRR of link \(<s_i, r_i>\) when s0 transmits concurrently with T by \(P_{\text{RR}}(T \cup \{s_0\} \setminus \{s_i\}, <s_i, r_i>)\), where 0\( \leq \)i\( \leq \)k. The PRR of links \(<s_i, r_i>\) is also estimated by looking up the i-vector table. But when no i-vector matches, \(P_{\text{RR}}(T \cup \{s_0\} \setminus \{s_i\}, <s_i, r_i>)\) is estimated to be \(1/(k+1)\) if any link of \(<s_i, r_i>\) (0\( \leq \)i\( \leq \)k) has matched i-vector. Otherwise, \(P_{\text{RR}}(T \cup \{s_0\} \setminus \{s_i\}, <s_i, r_i>)\) is also estimated to be 1. A PRR estimation of 1 for unmatched i-vector can encourage unknown concurrent transmissions and derive the corresponding i-vectors. Based on the estimated \(P_{\text{RR}}(T \cup \{s_0\} \setminus \{s_i\}, <s_i, r_i>)\) after s0’s transmission, the new throughput can be estimated as

\[
TH_{T \cup \{s_0\}} = \sum_{s_i \in T \cup \{s_0\}} P_{\text{RR}}(T \cup \{s_0\} \setminus \{s_i\}, <s_i, r_i>)
\]

If \(TH_{T \cup \{s_0\}} \geq (1+\alpha)TH_T\) in which \(\alpha(>0)\) is a constant, s0 transmits immediately; otherwise, it retries transmission when the ongoing flow with the earliest end time finishes.

**E. Transmission control**

LogA adds to the header of each data packet a blockNo field that carries a sequence number for each transmitted block. After transmission of a block, the sender waits for a block ACK from the receiver for up to a duration \(T_{\text{wait ack}}\). If the ACK is lost, the sender does not immediately retransmit all packets of the unacknowledged block. Instead, it transmits the next block in the buffer. It invokes a cluster backoff when the previous \(N_{\text{unack}}\) ACKs are all lost, and then retransmits unacknowledged packets. This prevents wasteful retransmissions due to loss of ACK. The ACK are accumulative and carries \(N_{\text{unack}}\) bitmaps for the previous \(N_{\text{unack}}\) blocks. Each bit in the bitmap indicates whether a packet in the block is received. In our experiments, we set \(N_{\text{unack}} = 4\).

LogA adopts two backoff policies: block backoff and cluster backoff. Block backoff is mainly used to prevent transient
losses when hidden terminals exist. Each sender maintains a block contention window \( CW \) that is 0 initially. Before the channel assess period of a block transmission, the sender waits for a random backoff time between 0 and \( CW \). The sender updates \( CW \) only when an ACK is received. The sender calculates the average PRR, denoted by \( PRR' \), of the link based on the bitmaps in the ACK. Then, the sender updates \( CW \) using the following rules:

\[
CW_i = \begin{cases} 
0, & \text{if } PRR' > \eta_{cw} \\
CW_{min}, & \text{if } PRR' \leq \eta_{cw} \text{ and } CW_{i-1} = 0 \\
2 \cdot CW_{i-1}, & \text{if } PRR' \leq \eta_{cw} \text{ and } 2 \cdot CW_{i-1} < CW_{max}, \\
CW_{max}, & \text{if } PRR' \leq \eta_{cw} \text{ and } 2 \cdot CW_{i-1} \geq CW_{max}, 
\end{cases}
\]

in which \( \eta_{cw} \) is a constant, \( CW_{min} \) and \( CW_{max} \) are the minimal and maximal value of \( CW \) respectively. \( CW_{max} \) shouldn’t be smaller than the transmission time of a block.

Cluster backoff is used to avoid continuous ACK losses that may indicate severe interference. When the sender has lost \( N_{\text{unack}} \) ACKs continuously, it stops transmission and waits for a random duration chosen between \( \theta_{min} \) and \( \theta_{max} \), where \( \theta_{min} \) and \( \theta_{max} \) are two constants. \( \theta_{max} \) should be large enough for completing \( N_{\text{unack}} \) blocks transmission.

Nodes exchange i-vectors and time logs by periodically or opportunistically broadcasting separate control packets or by piggy-backing them onto routing beacons or other control packets. In order to mitigate the effect of control packet losses, i-vectors are transmitted repetitively. Besides, each control packet carries the time logs of the previous \( N_{\text{rep}} \) blocks, where \( N_{\text{rep}} \) is a constant. In our implementation, nodes use dedicated control packets. It broadcasts control packets after the ACK waiting period if the channel is clear or if the node skips the previous control packet. The granularity of time log is milliseconds (ms). We use 3 bytes to carry a base time, then each time log can be represented by only 3 bytes, i.e., 2 bytes for the value \( t_0 \) - base time and 1 byte for the value \( t_1 - t_0 \).

IV. EVALUATION

A. Methodology and Settings

We implement LogA in TinyOS-2.1 and evaluate its performance in TOSSIM. The closest-pattern matching model used in TOSSIM 2.1 is derived from real-world experimental traces on radio TI CC2420 [13]. It has been shown that this model accurately simulates packet delivery behavior of CC2420.

Currently, there exists no concurrent MAC designed to provide high throughput for sensor networks with unpredictable traffic. We compare LogA with a B-MAC [4] like CSMA protocol. We also use a reactive concurrent MAC CMAP [12] for comparison, which is designed for 802.11. This is because CMAP can be an alternative approach to the problem addressed in this paper. CMAP is also based on interference relationship. CMAP transmits packets in a bursty manner. A receiver identifies conflicting links by snooping the IDs in the tailer of the interfering burst. To adapt CMAP to WSN, we allow a receiver to identify conflicting links by not only the IDs in the tailer but also IDs in each data packet.

This improves the performance of CMAP by 8% - 15% in our experiments. To differentiate between the modified CMAP and the original one, we use CMAP+ to refer the improved CMAP.

In our implementation of LogA, we set \( N_{\text{unack}} \) to 4, \( N_{\text{rep}} \) to 5, \( \alpha \) to 0.1, \( \eta_{cw} \) to 0.5, and \( C_{max} \) to 3. Moreover, we use \( T_s = 12\text{ms}, T_{\text{wait ack}} = 4\text{ms}, \) and \( CW_{min} = 4\text{ms} \). \( CW_{max} \) is configured to a block transmission time in each simulation. The parameter of cluster backoff \( \theta_{max} \) is set to \( \frac{1}{2} \cdot N_{\text{unack}} \cdot CW_{max} \), and \( \theta_{min} \) is set to \( \frac{\theta_{max}}{2} \). In addition, all nodes transmit data at full rate unless mentioned otherwise.

Metrics used in the evaluation are throughput, delivery latency, and energy consumption. Moreover, we also evaluate the detection rate and estimation accuracy of time log analysis.

B. Micro-evaluation of Time Log Analysis

The performance of time log analysis is very crucial to the performance of LogA. In this subsection, we evaluate detection rate and estimation accuracy of LogA.

We define detection rate as the ratio of the number of successfully estimated i-vectors to the total number of i-vectors that should be estimated. This includes all interference patterns of one-hop interferers of a sender that have ever transmitted concurrently with the sender. We compare detection rate of LogA with that of receiver-based interferer detection of CMAP+. We randomly generate different number of one-hop flows. However, we require that the PRR of the flows should be above 0.4 since routing layer will not choose a link with too low PRR. Moreover, the PRR of links between these senders is above 0.2. This represents scenarios of heavy interference. For each flow density, we randomly generate 20 topologies.

Fig. 4 shows the average detection rate of both schemes. We can see that the detection rate of LogA is very high, above 80% for all considered flow density. It also has much higher detection rate than CMAP+ especially for high flow density. This is because CMAP+ can only detect those interferers that are within the communication range of both the sender and the receiver. However, in many cases, the receiver is not within the communications range of the one-hop interferers of the sender. For LogA, the cause of missing detection is the loss of time log and ACK. But repetitive transmission of time logs and ACKs effectively reduces the chance of loss of time log.

In addition, we investigate the estimation accuracy of LogA. We fix the flow density to 8. The requirements of PRR of each flow and the links between senders are the same as the previous settings of experiments. We vary the maximum synchronization error from 10 microsecond (\( \mu s \)) to 10 millisecond.
(ms). For each level of synchronization error, we generate 20 random topologies. To gain the ground-truth, we first control all combinations of 2−4 senders to transmit packets concurrently and keep other senders silent. The measured PRR of each receiver is used as the “true” PRR. We then compare the estimated PRR of randomly selected links to its ground truth. The absolute estimation error is calculated as the results.

Fig. 5 shows the estimation error. In this figure, bsize and psize represent block size and payload size respectively. We consider 6 combination of payload size and block size. We can see that LogA achieves quite accurate estimation of PRR under interference, mostly below 0.15. Moreover, LogA is tolerant to synchronization error. This is because LogA uses a sequence of overlapped packets to estimated the PRR. Synchronization errors below 5ms only cause the overlap of time log to drift 1 or 2 packets, having small effects on PRR estimation. When the synchronization error is 10ms, the estimation error increases distinctly for cases with bsize≤16 and psize≤72. In such conditions, we advocate the use of large packets or blocks since the estimation error still remains small when bsize or psize is large as shown in Fig. 5. Remember that the state-of-art time synchronization protocols can keep synchronization error below 10s (us) with very small cost.

C. Performance with Different Flow Density

In this subsection, we compare LogA with CSMA, CMAP+ in terms of system throughput, latency, and energy consumption. The system throughput is defined as the average sum of data delivery rate of all flows. Since LogA is designed to achieve high throughput, a CSMA protocol working in the duty-cycle mode cannot achieve its highest throughput. To give a fair comparison, the B-MAC like protocol works without sleep. The network size is chosen to be 50m×50m and the transmit power is set to 0dBm. To simulate unpredictable traffic, we deliver 10 bursts of bulk traffic in each run and randomize the start time of each bulk traffic. Once a burst of bulk traffic appears, senders of all flows transmit at the maximum rate for 20 seconds. We measure the three metrics in networks with different flow density, i.e., the number of flows. For each flow density, we generate 20 random topologies and run three schemes once on each topology.

Fig. 6 plots the system throughput of three schemes. We can see that LogA achieves a system throughput of 280−480kbps, which corresponds to 30%−60% performance gain over CSMA and 11%−30% performance gain over CMAP+. With the increase in flow density, the throughput of all schemes increases. But when the flow number is larger than 11, the throughput of CSMA and CMAP+ almost stops increasing while LogA still has large throughput increase. This is because as flow density increases, more and more senders are within the interference range of each other. CSMA only allows one of these sender to access the channel. On the contrary, CMAP+ cannot detect all interferers when the number of flows becomes large as shown in Fig. 4. So, senders in CMAP+ tend to transmit aggressively when its transmission will severely reduce the throughput. However, the detection rate and estimation accuracy of LogA are still high even when flow density is high. So, the senders can make correct transmission decision in most cases. Moreover, the senders in LogA transmit packets of a block without CCA. This saves a lot of backoff time compared with CSMA.

Fig. 7 shows the average delivery latency which is the interval from the time a packet is put into the sender’s transmission buffer to the time the receiver correctly receives the packet. From the figure, we can see that LogA has smaller latency than CSMA and CMAP+. For example, when there are 8 active flows, the average latency of CSMA and CMAP+ is 125.4ms and 118.6ms respectively, while that of LogA is 102.9ms. LogA reduces the latency by 13% as compared to CMAP+ and by 17% as compared to CSMA.

Fig. 8 plots the proportion of energy consumption per delivered byte. To measure the energy consumption, we add an energy management module to TOSSIM, which records separated time of three radio states: transmitting, receiving, and idle. We multiplies the total time in each state by the power consumption in the state as listed on the CC2420 data sheet. Then, we divide the total energy consumed by the total number of bytes of payload delivered. Finally, a normalization is conducted. We can see that when the flow density is very low, the three schemes consume almost the same energy. But as the number of flow increases, the energy consumption of LogA rapidly decreases. On average, LogA reduces the energy consumption by about 30% as compared to CSMA, and by about 8.5% as compared to CMAP+.

D. Performance with Different Block Sizes

In this subsection, we investigate the effect of block size on the performance of LogA. The compared CSMA protocol
works under its default block size of 1, i.e., it transmits only one packet once it gets access to the channel. We vary the block size from 16 to 64. Moreover, two payload sizes, i.e., 48 and 72, are considered. The transmit power and network size are the same as in the previous subsection. The method of unpredictable traffic generation is also the same as in the previous subsection. We fix the flow number to 12 and calculate the average results of 20 random topologies.

Fig. 9 shows the system throughput of the three schemes. This figure shows that LogA always outperforms CSMA and CMAP+. Compared with CMAP+, LogA achieves about 19%~49% throughput gain when the payload size is 48 bytes. The throughput gain is about 22%~55% in the case of 72 bytes payload size. In addition, the system throughput of LogA increases as the block size increases. This is because LogA encourages concurrent transmissions and hence larger block size increases the efficiency of concurrency. On the other hand, the throughput of CMAP+ drops when the block size is larger than some threshold. The reasons are two-fold. First, the binary interference relationship considered by CMAP+ prevents the throughput from increasing. Second, with the increase in the block size, the interferer detection capability of CMAP+ deteriorates.

Fig. 10 presents the latency under different block sizes. Here, the delivery latency is also measured in a one-hop sense. We can see that the delivery latency of LogA is lower than CSMA and CMAP+. For instance, when the block size is 32 packets and the payload size is 72 bytes, LogA reduces the latency by about 10% as compared to CSMA and by about 7% as compared to CMAP+. Both LogA and CMAP+ have larger latency with the increase in block size. Moreover, larger packets increase latency since the transmission time of each packet increases.

Fig. 11 plots the energy consumption for different block sizes. Here, we take the energy consumption of CSMA in the case of 48 bytes payload size as baseline. We can see that LogA consumes less energy than CSMA and CMAP+, especially when the block size is larger than 40 packets. Compared to CMAP+, LogA improves the energy efficiency by about 18% on average in the case of 48 bytes payload, and by about 20% in the case of 72 bytes payload. The result shows that although LogA has larger throughput when block size increases, the throughput gain is not achieved at the cost of more energy consumption than CSMA and CMAP+.

V. CONCLUSION

In this paper, we present LogA, a reactive concurrent MAC protocol for data-intensive sensor networks with unpredictable traffic. LogA exploits the start time and end time of block transmission to reactively and passively infer interference relationship amongst nodes. The inferred interference relationship is then utilized to improve the probability of beneficial concurrent transmission. We evaluate the performance of LogA extensively in TOSSIM and compare it with CSMA and CMAP. The experimental results show that LogA outperforms CSMA and CMAP in terms of system throughput, delivery latency, and energy consumption.

REFERENCES