Batch Forwarding in Wireless Sensor Networks

Turgay Korkmaz
Department of Computer Science
University of Texas at San Antonio
korkmaz@cs.utsa.edu

***Abstract*—Batch processing is a well-known concept in computer science and widely used to improve performance in many applications. Similarly, to improve the delivery ratio and end2end delay in wireless sensor networks (WSNs), we propose to use *batch forwarding* instead of *streaming*. Our main goal in this paper is to evaluate and compare the performance of batch forwarding and that of streaming. Accordingly, we first develop analytical models under simplified assumptions. We then conduct extensive actual experiments using TinyOS and IRIS motes from Crossbow. Both analytical and actual experimental results show that batch forwarding significantly improves the efficiency and delivery radio, particularly in heavily loaded network environments. Despite some delay due to batch formation, batch forwarding also improves the overall end2end delay performance due to its efficient use of resources. As a result, batch forwarding is a viable data gathering solution and must be used in WSN applications that generate significant amount of sensing data and require high delivery ratio and less delay.**

I. Introduction

In recent years, wireless sensor networks (WSNs) have been receiving significant attention due to their potential use in several different real-world applications [1], [2]. In many of these WSN applications, sensor nodes (i) monitor their environment, (ii) collect and/or aggregate sensory data, and (iii) send these data to a base station (or an actuator node). If the sensor nodes are not within the communication range of the final destination, then the other sensor nodes act as relay nodes (routers) and forward the messages from the originating sensor nodes to the final destination through a single path or multiple paths. Currently, the relay nodes simply try to forward each incoming message to the next node(s). We call this existing scheme as the *streaming* of sensory data. However, this is not an efficient way to utilize underlying resources.

Instead, being inspired from the well-known concept of batch processing, we propose to use a *batch forwarding* scheme where a relay node (i) collects a number of incoming messages destined to the base station, (ii) puts data parts of incoming messages into a larger container called a *batch message* and (iii) sends only this batch message to the next relay node toward the base station. Clearly, batch forwarding reduces the number of messages and their total sizes. In addition, batch forwarding tries to access the shared channel once while streaming tries to access the channel many times. In the next section, we elaborate on these advantages of batch forwarding and discuss some related work. However, in general, we can see that, due to reduced message size and one time access to the shared channel, batch forwarding is expected to utilize the underlying resources (e.g., bandwidth, energy) much more efficiently than streaming.

This research is supported by DoD Infrastructure Support Program for HBCU/MI, Grant: 54477-CI-ISP (UNCLASSIFIED).

The main goal of this paper is to verify the above claim and compare the performance of batch forwarding and streaming using analytical and experimental techniques. Accordingly, we first develop some analytical models to have better insights about how and why batch forwarding is useful and outperforms streaming. We then conduct actual experiments[1] to verify our analytical findings and more importantly to show that batch forwarding is an effective data gathering mechanism as it significantly improves the data delivery ratio and end2end delay performance.

To simplify our analysis and experiments, we use a single-hop WSN (or an extended star topology), where sensor nodes communicate with the base station via a single relay node as shown in Figure 1. Note that due to packet size limitations

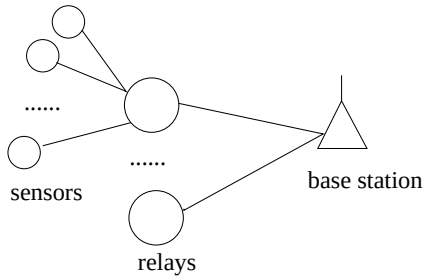

Fig. 1. Single-hop WSN or extended star topology.

in WSNs (e.g., 128 Bytes), the batch messages will mostly be created at the first relay node and forwarded to the next node. The subsequent relay nodes along the multi-hop path(s) will keep forwarding the same batch messages and thus benefit from the savings that we had in the first relay node. As a result, the network architecture that we are using is able to provide the bottom line performance gain of batch forwarding over streaming. Nevertheless, we plan to consider other topologies and multi-hop and multi-path forwarding scenarios in our future work.

The rest of this paper is organized as follows. In Section II we formally define batch forwarding, discuss its advantages,

[1]We implemented necessary modules using TinyOS 1.x [3] and nesC, and installed them on Crossbow IRIS motes [4]. The source code of our nesC programs and experimental results are available at www.cs.utsa.edu/∼korkmaz/research/batch.

and present some related work. We then analyze the efficiency and delay characteristics of both streaming and batch forwarding in Section III. We present the results of our experiments in Section IV. Finally, we conclude this paper and discuss some future work in Section V.

II. Batch Forwarding and Related Work

In batch forwarding, the idea is to simply collect a number of incoming messages destined to the base station, and send their data parts as a single batch message to the next node. We assume that each sensory data message consists of Header (H) and Data (D) Bytes, where Header contains all the physical, data link, and network layer headers. Now suppose we have N sensor nodes that send $m > 1$ messages to a relay node in a given time frame. In case of streaming, the relay node receives m messages with the total size of $m(H + D)$ and tries to forward the same number of messages with the same total size. In case of batch forwarding, the relay node again receives m messages with the total size of $m(H + D)$; but, it throws away all the headers and forwards *only one* message with the total size of $H + mD$ Bytes, where m is the batch size.

One of the key issues is how to set the batch size m. If the system load level is known, then we may be able to fix the batch size based on our analysis in the next section. In practice, the system load level may not be known in advance or it may change over time, so the batch size needs to be determined dynamically based on system load levels. Accordingly, we propose to use a timeout-based *dynamic batch forwarding* mechanism where the relay node sets a timer and collects incoming sensory data messages. When the timer goes off or the maximum batch size is reached, the relay node constructs a batch from the already received messages and sends this batch to the next node. As a result, in the worst case, batch formation delay would be bounded by the given timeout value. Note that dynamic batch forwarding may set m to 1. In that case, batch forwarding will be the same as streaming. Our experimental results show that dynamic batch forwarding is able to find the best batch size under the given timeout values.

A. Advantages of Batch Forwarding

Batch forwarding has several advantages. First, since most packets in WSNs are destined to the base station, a relay node can easily concatenate the data portions of the received messages, and send them as a batch to the next node. By doing this, we avoid the re-transmission of several headers in physical, data link, and network layers. These savings allow us to make better use of the underlying scarce channel capacity (e.g., 250Kbps in current WSNs). Note that network layer contains some useful information in traditional IP networks such as the source address [5]. However, TOS_Msg structure [3] used in the network layer of WSNs does not have such useful information. So, by removing the header from each message, we do not lose any useful information. If there is any such information (e.g., the source address), it can be added to the data part of the message.

Second, let N be the number of sensor nodes served by a relay node. Since the sensor nodes and the relay node are within the same communication range, these $N + 1$ nodes compete for the same channel using a contention-based MAC protocol (e.g., IEEE 802.15.4 [1]). Suppose the underlying MAC protocol is able to provide a fair service, in which each node will have the same chance to access the channel. But, in case of streaming, this may cause significant queuing delays at the relay node because the relay node may get N packets while sending only one within a given time frame. The relay node will be able to send the remaining messages in the next time frame if no more packets come from the sensor nodes. So, streaming may work fine when there is not much load in the network. However, if the sensors generate more data messages in every time frame, then the relay node cannot be able to forward all the incoming messages, causing significant queuing delays. One solution could be to change the MAC protocol and give higher priority to the relay node. Fortunately, without changing the MAC protocol, batch forwarding can cope with that situation as follows. The relay node has the same chance of a sensor node to access the channel; but, when it accesses, it forwards more data and thus match the incoming and outgoing data rates. This results in significantly reducing queueing delays as verified in our actual experiments.

Since the relay node captures the shared wireless channel once instead of m times and avoids re-transmission of several headers, the underlying resources (e.g., energy, bandwidth) will be used much more efficiently in case of batch forwarding. However, depending on the system load and traffic characteristics, this performance gain may come at the cost of introducing some extra delay due to forming a batch at the relay node. Specifically, if the system is lightly loaded (i.e., if sensor nodes rarely generate data), then forming a larger size batch would introduce unnecessary delay. In this case, it might be better to use streaming. However, if the system gets highly loaded or many sensors report their data around the same time, then we specifically need dynamic batch forwarding as it determines the appropriate batch size based on the system load level and bounds the worst-case batch formation delay. If we do not deal with the excessive amount of traffic through batch forwarding, the incoming packets will either be dropped or experience huge amounts of queueing delay at the relay node.

B. Related Work

The general idea of batch forwarding is inspired from the well-known concept of batch processing. This concept has been widely used in many situations to improve performance (e.g., [6], [7]). In the context of WSNs, we can see data aggregation as some kind of batch processing and thus it is related to batch forwarding with some differences. The goal of data aggregation is to collect and process sensory data messages in the network [8] (e.g., find min, max, or average), and then send only the resulting information to the base station. In contrast, batch forwarding does not process data but simply combines the readings from different sensors and sends them to the base station as a batch.

As studied in many research papers (e.g., [9], [10], [11]), data aggregation is a great way to improve the energy efficiency and resource utilization by minimizing the amount of sensory data sent to the base station. However, in some cases, we may not want data aggregation in a network. For example, suppose we have an application that wants to track an object or see the temporal or spatial distribution of the sensory readings over time and/or field. In this case, we need to get every sensory reading to the base station.

Moreover, data aggregation works on one type of sensory data. But in many applications, different types of sensory data (e.g., light, temperature, vibration etc.) might be measured. The same types of measurements might be aggregated, but different types cannot be aggregated in one. Even the aggregated data need to be sent to the base station. So, we ultimately need an effective data *gathering mechanism* to transfer individual or aggregated sensory readings from sensor nodes to the base station. In this regard, the proposed batch forwarding mechanism is a viable solution and complementary to the data aggregation mechanisms as it significantly improves the delivery ratio and end2end delay performance.

III. Analytical Results

In general, it would be better for a relay node to use *larger* size batches to minimize the message overhead. On the other hand, since a message has to wait until a batch is formed, it would be better to use *smaller* size batches. Given this trade-off, we analyze how batch size impacts the overall utilization of network resources and delay performance.

A. Efficiency and Delivery Ratio

We first analyze how efficiently the underlying channel capacity (say R) is used when $m = 1$ (i.e., streaming is used) and $m > 1$ (i.e., batch forwarding is used). For this, we declare a goodput efficiency (ge) metric as follows

$$ge = \frac{Total\ size\ of\ sensor\ data\ sent/R}{Total\ size\ of\ messages\ sent/R + \chi}$$

where χ is the total time that is wasted because of collisions, bit errors, channel access delays due to underlying contention-based MAC scheme, etc. In our actual experiments, χ will be accounted for. However, to get a general idea about the benefits of batch forwarding over streaming from an analytical perspective, we assume that we have an idealized environment where χ is 0. In that case, based on the discussion about the message size of streaming and batch forwarding in the previous section, we can give $ge_{streaming}$ and $ge_{batching}$ as follows:

$$ge_{streaming} = \frac{mD}{2m(H+D)} = \frac{D}{2(H+D)}$$

and

$$ge_{batching} = \frac{mD}{m(H+D)+(H+mD)},$$

where m is the batch size, H is the header size and D is the sensory data size. According to the currently used operating system [3], MAC [12], and radio standards [13], [14] in WSNs, the header size H (including footer) would be 16 Bytes and the maximum data size that can be transferred is 128 Bytes. After excluding TinyOS headers, we have effectively 120 Bytes for application data. To consider different batch sizes in our actual experiments, we assume that individual sensors generate $D = 12$ Bytes sensory data. Using these values of H and D, Figure 2 illustrates the goodput efficiency of streaming and batch forwarding. Note that batch forwarding with $m = 1$ is the same as streaming.

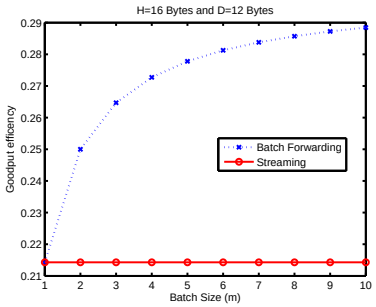

Fig. 2. Analytical: Goodput efficiency of streaming ($m = 1$) and batch forwarding ($m > 1$).

Naturally, $ge_{streaming}$ does not depend on m while $ge_{batching}$ increases with m until an upper bound. The upper bound on $ge_{batching}$ can be determined by taking the limit as m goes to ∞. Accordingly, we have

$$ge^{MAX}_{batching} = \lim_{m\to\infty} \frac{mD}{m(H+D)+(H+mD)}.$$

Using the L'Hospital rule, we get

$$ge^{MAX}_{batching} = \frac{D}{H+2D}.$$

In practice, due to the above mentioned limit on the maximum packet size in WSNs, the maximum batch size should be less than or equal to $\lfloor \frac{120}{D} \rfloor$. Fortunately, the goodput efficiency with these practically available batch sizes is very close to the maximum goodput efficiency, as depicted in Figure 3.

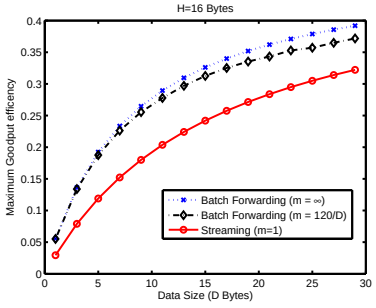

Fig. 3. Analytical: Maximum goodput efficiency of streaming ($m = 1$) and batch forwarding ($m > 1$).

To better understand the efficiency measure, we need to actually look at a practical performance metric known as delivery ratio (DR). We will formally define this metric and measure it through actual experiments in Section IV. We show that batch forwarding significantly improves DR over streaming as it increases goodput efficiency.

B. Delay Performance

In traditional packet networks, we have four delay components [5]: processing, queuing, transmission, and propagation delays. Sum of these delay components gives us the end2end delay. The propagation delay is the same for both streaming and batch forwarding. Moreover, since the distance between sensor nodes is very short, it is negligible. Thus, we will look at the other delay components.

1) Transmission Delay: This is the time required to send all the bits of a packet into the channel after the channel is captured. It is given by L/R, where L is the packet size and R is the channel capacity. Accordingly, the transmission delay under streaming will be $\frac{H+D}{R}$ for every message. Since the relay node needs to forward m such messages, the total transmission time will be $m\frac{H+D}{R}$. However, in case of batch forwarding, the relay will transmit only one packet with the size of $H+mD$. Accordingly, the total transmission time will be $\frac{H+mD}{R}$. Clearly, as the batch size increases, this will be significantly lower than that of streaming.

2) Processing Delay: This is the time to check the received packet, change some of its fields (if needed), and determine the next node. In general, these steps are the same for both streaming and batch forwarding. However, in case of batch forwarding, we have another step called *batch formation*. In case of streaming, there is no batch formation delay since the relay node tries to send the incoming sensory data messages as soon as possible. In batch forwarding, however, m messages need to be collected and thus the first message waits until the m^{th} message arrives. Actually, batch formation could be also seen as part of queueing delay discussed next. But we separate it from queueing delay and analyze it here.

Assume that a relay node receives λ messages per second from the sensor nodes, then the average inter-arrival time between the sensory data messages would be $\frac{1}{\lambda}$. Accordingly, we can compute the batch formation delay (bfd) for the first sensory data message as $bfd_1 = (m-1)\frac{1}{\lambda}$. In general, the batch formation delay for the i^{th} sensory data message in a m-size batch would be $bfd_i = (m-i)\frac{1}{\lambda}$ for $i = 1, 2, ..., m$. Clearly, the first sensory data message experiences the worst batch formation delay while the last one experiences no delay. So the average batch formation delay for an arbitrary sensory data message would be

$$avg\_bfd = \frac{\sum_{i=1}^{m} bfd_i}{m} = \frac{m^2 - m}{2\lambda}.$$

If λ increases (i.e., the system load increases), then the batch formation delay approaches to zero. However, in a lightly loaded system, batch formation delay could be significant if larger size batches are used. Therefore, as we proposed before, we can set an upper limit for the batch formation delay through *dynamic batch forwarding*. In essence, the relay sets a timer for Tms after receiving the first packet. When the timer goes off or the maximum batch size is reached, the relay makes a batch using the already received messages and sends this batch to the next node. So, in the worst case, the batch formation delay for the first packet will be Tms.

3) Queuing Delay: This is the time for a ready-to-go packet (or a batch) to wait in a queue until its transmission starts. There is a significant body of work on analyzing queueing delays under various assumptions [15], [16]. However, directly applying such results to wireless networks is a little bit complicated due to the shared nature of the underlying channel, collisions, bit errors, and random back-off algorithms etc. Nevertheless, researchers tried to incorporate these factors into queueing models and evaluated queuing delays and throughput in wireless networks (e.g., [17], [18], [19], [20]).

Most of these studies are done in the context of IEEE 802.11 and/or assume that N nodes independently generate data to send to others in a uniform manner. Actually, in case of streaming, we may utilize the analytical model provided in [17] by setting the absorption probability to 0 for the relay node since it forwards all the incoming messages, and to 1 for the base station. In case of batch forwarding, however, we cannot use the same model as is because the service distribution and the characteristics of the outgoing traffic will be different. In a future work, we will consider these differences and try to integrate them into the existing queuing models. For the time being, we develop a simplified analytical model next to see how batch forwarding improves the end2end delay that includes queueing delay, batch formation, and transmission delay. To account for all the other factors and show how our simplified model captures the general trend in practice, we mainly rely on the actual experimental results presented in Section IV.

4) End2end Delay: This is the total time between the time that a sensory data message is generated and the time that the sensory data message is received by the base station. It includes all the delay components discussed above. To get a general idea about end2end delay, we will consider an idealized environment where propagation delay is zero, and sensor nodes know when to access the channel so there are no channel access conflicts, backoff delays, or packet losses. Note that these factors affect streaming more than batch forwarding since batch forwarding is trying to access the shared channel less than streaming. So, ignoring them would give us the bottom line when comparing the delay performances of batch forwarding and streaming.

Suppose we have several sensor nodes and K of them have generated sensory data messages at the same time (say, $t = 0$). Each sensor node (say s_i) tries to send its own data to the relay node. The relay node collects m incoming messages and forwards them as an m-size batch message to the base station (we denote this behavior by $r(m)$). Now let us consider the following two extreme patterns for the channel access.

Pattern I: $s_1, r(1), s_2, r(1), ...$ when streaming is used, and $s_1, s_2, ..., s_m, r(m), s_{m+1}, s_{m+2}, ..., s_{2m}, r(m), ...$ when batch forwarding is used.

Pattern II: $s_1, s_2, ..., s_K, r(1), r(1), ...$ when streaming is used, and $s_1, s_2, ..., s_K, r(m), r(m), ...$ when batch forwarding is used.

Different patterns can also be considered. However, the delay performance for other patterns will be somewhat between these

two patterns because the sensory data messages are already generated and they need to wait either at the relay node or at their original source nodes.

Let d_i be the end2end delay for the sensory data message generated by sensor node i. For the above patterns, we can compute d_i as follows

For Pattern I:

$$d_i = \lceil \frac{i}{m} \rceil (m\frac{H+D}{R} + \frac{H+mD}{R})$$

For Pattern II:

$$d_i = K\frac{H+D}{R} + \lceil \frac{i}{m} \rceil \frac{H+mD}{R},$$

where $m = 1$ for streaming and $m > 1$ for batch forwarding. We draw the footprints of the end2end delay for the first and the last sensory data message in Figure 4. Both patterns show

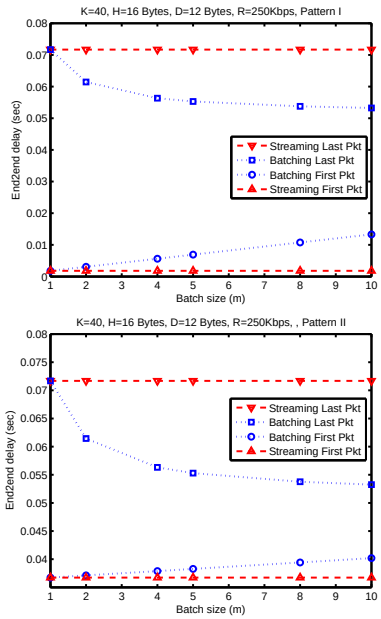

Fig. 4. Analytical: End2end delay for the first and the last message under streaming ($m = 1$) and batch forwarding ($m > 1$).

similar trends. Specifically, as expected in case of streaming, the end2end delay for the first message is the smallest while the end2end delay for the last message is the highest. In case of batch forwarding, the end2end delay for the first message gets worse than that of streaming as m increases due to batch formation. However, the end2end delay for the last message gets better than that of streaming since batch forwarding reduces the queueing delays.

Let us now analyze the average end2end delay for an arbitrary sensory data message. Clearly, we can compute the average end2end delay as follows

$$avg\_d = \frac{\sum_{i=1}^{K} d_i}{K}$$

Suppose K is multiple of m, then using the celing/floor sum techniques from [21], we can determine the average end2end delay for the above patterns as follows.

For Pattern I:

$$avg\_d = \frac{(K+m)}{2m}\left(m\frac{H+D}{R} + \frac{H+mD}{R}\right)$$

For Pattern II:

$$avg\_d = K\frac{H+D}{R} + \frac{(K+m)}{2m}\frac{(H+mD)}{R},$$

where $m = 1$ for streaming and $m > 1$ for batch forwarding. We draw the footprints of the average end2end delay behavior of streaming and batch forwarding in Figure 5.[2] In contrast

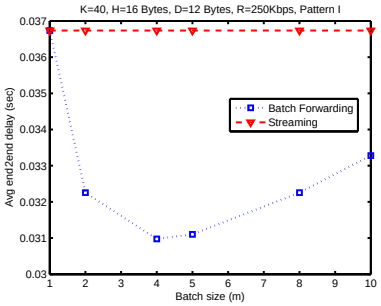

Fig. 5. Analytical: Average end2end delay for streaming ($m = 1$) and batch forwarding ($m > 1$).

to common sense, batch forwarding actually decreases the average end2end delay when the appropriate batch size is selected. Even though we ignored several important factors (e.g., backoff times, collisions etc.), this simple model is able to capture the general end2end delay characteristics that we observe through our experiments in the next section. The reason is that the ignored factors negatively impact streaming more than batch forwarding. As a matter of fact, as our experiments show in the next section, when the load increases, the end2end delay of streaming gets significantly worse than that of batch forwarding. The reason is that the relay node has the same chance to access the channel; but in case of streaming, this is not enough to send all the incoming messages, causing significant queuing delay. In case of batch forwarding, the relay node is able to send more data and reduce the queueing delay.

IV. Actual Experiments

In the previous section, we showed that batch forwarding is clearly using underlying channel much more efficiently than streaming and also improves the end2end delay. In this section, we would like to verify if that is really the case in practice. Accordingly, using TinyOS 1.x [3] and nesC, we implemented the necessary modules and installed them on Crossbow IRIS motes [4] for the following nodes.[3]

For the **sensor nodes**, we implemented a module that periodically generates a $D = 12$-Byte application message.

[2]Since similar trends are observed, we exclude the figure for the second pattern to save some space.

[3]The source code of our nesC programs and experimental results are available at www.cs.utsa.edu/∼korkmaz/research/batch.

Instead of actual sensory data, this message contains node ID, batch size, packet ID, and time stamp that are used for our performance measurements. Since we have a small number of actual sensor nodes (currently 5), we use them to generate data more often so that we can mimic over loaded networks. Specifically, each sensor node generates a new message when it receives SendMsg.sendDone event for the previous message. Inspecting individual traces shows that this way a sensor node generates one message every 6ms, on average.[4]

For the **relay node**, we implemented streaming and both fixed size and dynamic batch forwarding modules. Streaming module simply tries to forward an incoming message to the base station. Fixed size batch forwarding module collects m messages. We gave m as an experiment parameter in the range $[1, 9]$. Dynamic bach forwarding module sets a timer for Tms when the first message is received, and then collects the incoming messages until the timer goes off or the maximum batch size is reached. In our experiments, we set T to 16ms, 32ms, and 64ms. The relay node then tries to send the data parts of the collected messages as a single batch message. To queue the incoming messages that cannot be sent right away, the relay node maintains a FIFO buffer that can hold B sensory data messages. In our experiments, we considered two different buffer sizes of 15 and 30. Increasing buffer size mainly impacted the end2end delay for streaming, while the other performance measures had the same trend. Thus, to save some space, we mainly report the results for $B = 15$ and also include end2end delay results for $B = 30$. More results can be obtained from the web page mentioned before.

For the **base station**, we slightly modified the Xsniffer module provided by Crossbow MoteWorks Software Platform [4]. Originally, this module used to send the received packets to the host computer and save them into a log file. In essence, we made two modifications (i) to synchronize the base station with the sensor nodes, and (ii) to determine the end2end delay by subtracting the time stamp of a sensory data message from the current time at the base station. Finally, the modified module puts the end2end delay into each message, and sends it to the host computer to be saved into a log file.

Synchronization of the base station with sensor nodes is necessary to exactly measure one way delay [22]. To perform synchronization, we implemented a special node called **starter node**. This node simply broadcasts a message to all nodes to achieve the following tasks: (i) synchronize the sensor nodes and the base station so that we can measure end2end delay, (ii) set some parameters of the relay node (e.g., fixed batch size or use dynamic batch forwarding, buffer size etc.), and (iii) start the experiment.

Using five sensor nodes along with one relay node and one base station, we conducted several actual experiments with different parameters. To create different workloads in the network, we did not activate all the sensor nodes at once. Instead, we ran our experiments with $N = 1, 2, 3, 4$, and 5 sensor nodes. So an experiment with $N = 1$ represents a lightly loaded network while an experiment with $N = 5$ represents a highly loaded network. In addition to N, we set other parameters m, T, and B, and repeated each experiment 3 times. We conducted our experiments indoors and used channel 26 which is not overlapping with 802.11. In each experiment, sensor nodes generated several thousand sensory data messages, and the relay node put the received messages into batches. In each experiment, we tried to get at least 1000 batches and saved them into log files. We later processed the log files and took the average of three experiments to determine our performance measures.

[4]Note that, individual sensors may not generate data that often in a practical WSN application. However, in a large scale WSN, since many sensors are involved, the relay node could frequently receive messages and try to forward them to the base station, as we try to mimic here.

A. *Efficiency and Delivery Ratio with Fixed Batch Sizes*

We first evaluate the efficiency and delivery ratio under fixed batch size. This will allow us to verify our analytical findings and show how different batch sizes impact the overall efficiency and delivery ratio in practice. For each value of N, m, T, and B, we conducted our experiments as discussed above. We then examined the log files and determined (i) Δ which is the total size of received sensory data, and (ii) Γ which is the total time to receive Δ amount of data for each experiment. Note that Γ includes all types of delays and factors that we ignored during our analysis in Section III-A. We then averaged the results of three experiments and computed the actual goodput efficiency (ge) as follows

$$ge_{actual} = \frac{\Delta/R}{\Gamma}$$

where $R = 250Kbps$ according to the current Radio standard [13] used in our experiments.

Figure 6 shows the goodput efficiency under different values of N and m. Clearly, batch forwarding $(m > 1)$ significantly

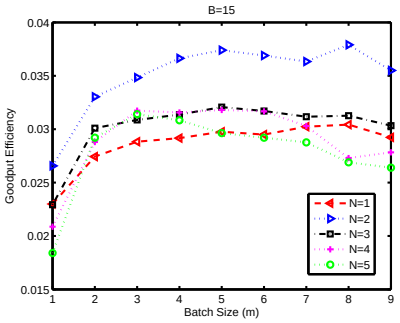

Fig. 6. Experimental: Goodput efficiency of streaming $(m = 1)$ and batch forwarding $(m > 1)$.

improves the goodput efficiency over streaming $(m = 1)$ under any N (i.e., load level). Although there are some ups and downs due to packet losses, the general trend is consistent with our analytical findings in Section III-A. Another observation here is that when we increase the number of nodes N from 1 to 2, the ge improves a little bit for streaming while it improves significantly for batch forwarding. If we keep increasing N, then the ge starts decreasing for both schemes due to excessive

loads and packet losses. However, the ge provided by batch forwarding is still significantly better than that of streaming under any load level. This indicates that batch forwarding is able to better deal with the increased load and efficiently utilize resources.

To better understand how efficiently the underlying channel is used, it would be better to look at a practical performance metric known as delivery ratio (DR), which can be defined as follows:

$$DR = \frac{TotalReceived}{TotalSent}$$

where $TotalSent$ is the total number of sensory data messages sent by all sensors and $TotalReceived$ is the total number of sensory data messages received by the base station. Figure 7 shows the actual DR under different parameters. As

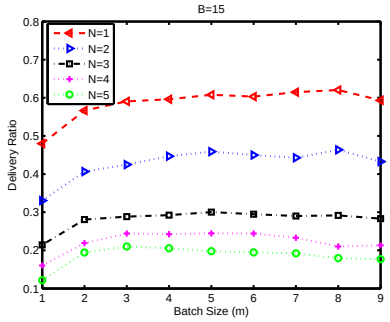

Fig. 7. Experimental: Delivery ratio of streaming ($m = 1$) and batch forwarding ($m > 1$).

the number of sensor nodes N increases, the delivery ratio is going down since all nodes are not able to send their messages to the relay node due to the increased load on the channel. Under any load level, however, the delivery ratio of batch forwarding is getting significantly better than that of streaming as m increases. To better see the improvement, we computed the delivery ratio improvement percentage of batch forwarding over streaming as shown in Figure 8. When the

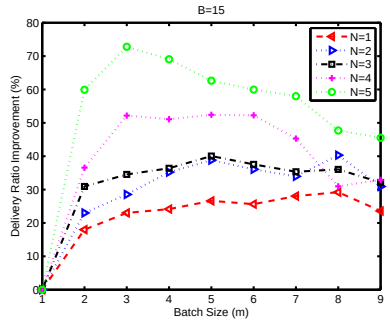

Fig. 8. Experimental: Delivery ratio improvement of batch forwarding ($m > 1$) over streaming ($m = 1$).

load significantly increases (N=5), batch forwarding improves the delivery ratio up to 70% over streaming. This is due to the fact that batch forwarding is using the underlying channel much more efficiently than streaming, and that batch forwarding sends more data to avoid buffer overflows when it gets its chance to access the channel.

Actual experimental results reveal an important fact about batch sizes in practice. In theory, as we studied in Section III-A, the performance metric ge (and, thus DR) is expected to continuously increase with m while approaching a limit. In practice, however, this happens until batch size is 4 or 5. After batch size 5, we see some ups and downs. The reason for that might be the fact that a large packet is more likely to be corrupted than a smaller packet under the same bit error rate [23]. In other words, since the packets get larger with increased batch size, the probability of losing such packets increases and causes some performance degradation. So, we recommend using batch sizes no more than 4 or 5 in practice. Actually, as shown by our analytical results in the previous section and our experiments in the next sub-section, a batch size of around 4 also minimizes the average end2end delay for sensory data packets. Note that this recommendation is valid when the sensory data size D is 12. If D increases or decreases, then the recommended batch size should be decreased or increased, respectively based on the analysis in the previous section. Actually, as we show later, dynamic batch forwarding would be better since it determines an appropriate batch size under any load level while setting an upper bound on the worst-case batch formation delay.

B. End2end Delay with Fixed Batch Sizes

To measure end2end delay, the starter node broadcasts a sync message to synchronize the sensor nodes and the base station. Then the sensor nodes time stamp their sensory data messages when they are generated. Time stamp is included in the $D = 12$-Byte sensory data message sent to the base station. When the base station receives a sensory data message, it subtracts the time stamp of this message from the current time at the base station and determines the one way end2end delay. Figure 9 shows the averages of these end2end delays when $B = 15$, and 30. When $N = 1$, the inter-arrival time between the packets is relatively large, and thus the batch formation delay becomes the dominant component of the end2end delay. Accordingly, the end2end delay increases as m increases. However, when N increases, more nodes try to send messages and thus the inter-arrival time becomes relatively small. So, the relay node quickly constructs batches with given sizes. As a result, batch formation delay becomes negligible while other components of end2end delay increase due to various factors including channel access delays, back-off delays, and queueing delays.

These factors specifically increase the end2end delay of streaming ($m = 1$) as the load (i.e., N) increases. Moreover, increasing buffer size B makes the end2end delay of streaming worse. The major reason for this is the queueing delay at the relay node. As we explained before, due to the underlying "fair" MAC protocol, the sensor nodes access the channel N times while the relay node access only once. So, as N increases, the relay node is not able to get enough chance to forward all the packets coming from N sensor nodes, and

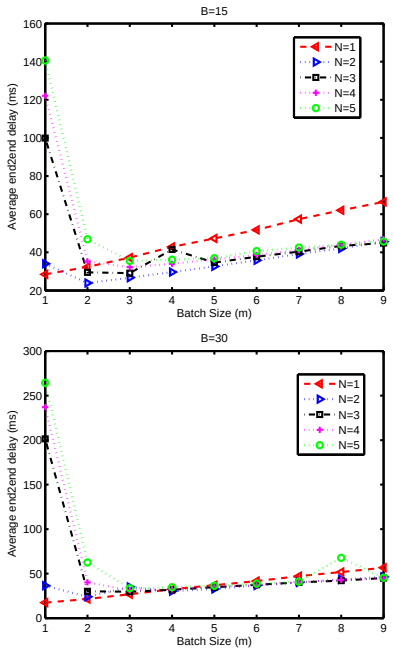

Fig. 9. Experimental: Average end2end delay of streaming ($m = 1$) and batch forwarding ($m > 1$).

thus it keeps buffering incoming packets and causes significant queueing delay. This is specifically true when we increase the buffer size from 15 to 30 because received packets are buffered for a long time instead of being discarded.

Fortunately, batch forwarding copes with that situation by allowing the relay node to send more data when it gets its fair chance to access the channel. So, when batch forwarding ($m > 1$) is used, the average end2end delay gets significantly lower than that of streaming. But, for larger values of m, the end2end delay starts increasing again due to batch formation delay and other factors. This suggests that we should use batch forwarding with carefully selected batch sizes around 3 and 4. Actually, as we discussed in previous sub-section, limiting batch size to 4 or 5 was also necessary to get better performance regarding goodput efficiency and delivery ratio. In general, these results are consistent with our analytical results in the previous section and show that we get similar trends with some differences in scale.

C. Dynamic Batch Forwarding

In dynamic batch forwarding, the relay node sets a timer for Tms when the first message is received. In our experiments, we used $T = 16$ms, 32ms, and 64ms. The relay then collects the incoming messages until the timer goes off or the predefined maximum batch size (9 in our experiments) is reached. Dynamic batch forwarding allows us to determine the appropriate batch size on the fly while setting an upper bound on the worst-case batch formation delay.

We first look at the average batch size. As shown in Figure 10, the average batch size changes depending on the system load level and the given timeout value. As expected, the average batch size increases either when N (i.e., load level)

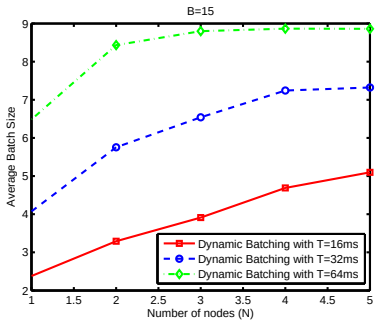

Fig. 10. Experimental: Average batch size under dynamic batch forwarding ($1 \leq m \leq 9$).

increases under the same value of T or when T increases under the same value of N. In both cases, the relay node gets more messages and constructs larger batches. Otherwise, it uses smaller size batches so that it can satisfy the constraint on the batch formation delay.

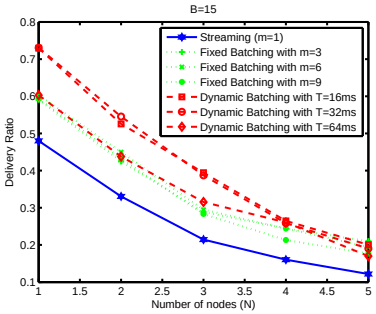

Fig. 11. Experimental: Delivery ratio of streaming ($m = 1$) and dynamic batch forwarding ($1 \leq m \leq 9$).

We next look at the delivery ratio and end2end delay performance under dynamic batch forwarding. Figure 11 compares the delivery ratio of streaming, fixed size batch forwarding, and dynamic batch forwarding. In general, since dynamic batch forwarding determines the appropriate batch size under any load level, it gives better delivery ratio than streaming and fixed size batch forwarding. However, when T is large like 64ms in our experiments, we reach the maximum batch size before the timer goes off, and thus dynamic batch forwarding performs as if we used fixed size batch forwarding with maximum batch size. However, if we made T too small, then dynamic batch forwarding would perform as if we use streaming. So, for good delivery ratio performance, we recommend setting the value of T to half of the time that is needed to reach the maximum batch size, e.g. 32ms in our experiments.

Figure 12 compares the end2end delay performance of streaming, fixed size batch forwarding, and dynamic batch forwarding. Depending on the given value of T, the end2end delay of dynamic batch forwarding is slightly less than or greater than that of fixed size batch forwarding. When the load (i.e., N) is small, the larger values of T increase batch formation delay and thus make the end2end delay worse than that of streaming. However, as mentioned above, selecting very small values for T would cause dynamic batch forwarding

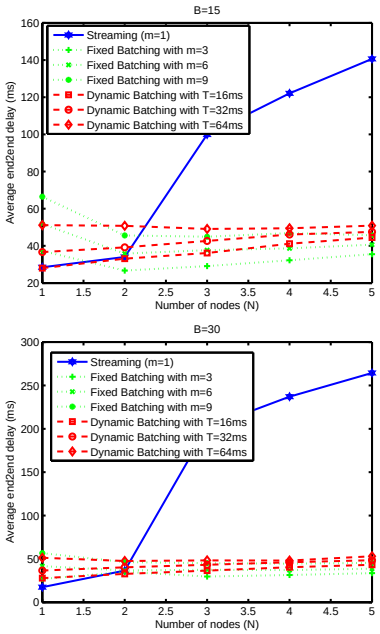

Fig. 12. Experimental: Average end2end delay of streaming ($m = 1$) and dynamic batch forwarding ($1 \leq m \leq 9$).

perform like streaming and increase queueing delay. As we see from the figure, the end2end delay for streaming gets significantly large as the load (i.e., N) increases. So, to avoid such queueing delays, we need to use dynamic batch forwarding with appropriate values of T. As recommended above, it seems appropriate to set the value of T to half of the time that is needed to reach the maximum batch size in a given WSN. This can be further halved to strictly bind the batch formation delay. In future, we will further investigate the relationships between batch formation delay, queuing delay, and delivery ratio so that we can also dynamically adjust the values of T for the best performance.

V. Conclusions and Future Work

We analyzed the efficiency and delay performance of batch forwarding in WSNs. We also verified our findings using actual experiments. Our results show that batch forwarding significantly improves efficiency and delivery ratio while minimizing end2end delay, particularly in heavily loaded environments. So, when the load increases, it would be necessary to activate a batch forwarding mechanism in WSNs. Specifically, dynamic batch forwarding with appropriate constraints on batch formation delay will give the best performance as it determines the appropriate batch sizes on the fly.

As future work, we will enhance our analytical modeling by analyzing the queuing delay performance under batch forwarding and relaxing some of the simplifying assumptions. In our actual experiments, we will further investigate the key relationships and tradeoffs between different parameters to better utilize batch forwarding in practice. We also plan to conduct large scale experiments involving multi-hop and multi-path forwarding. In addition to goodput efficiency, delivery radio and end2end delay improvements studied in this paper, we plan to investigate how batch forwarding can decrease energy usage and prolong the lifetime of WSNs.

References

[1] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 2006.

[2] I. Stojmenovic, *Handbook of Sensor Networks: Algorithms and Architectures*. Wiley-InterScience, 2005.

[3] "TinyOS," http://www.tinyos.net/tinyos-1.x/.

[4] "Crossbow technology, MoteWorks software platform," http://www.xbow.com, 2008.

[5] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 4th ed. Addison Wesley, 2008.

[6] C. Pavlovski and C. Boyd, "Efficient batch signature generation using tree structures," in *CrypTEC'99*. City University of Hong Kong, 1999, pp. 70–77.

[7] T.-Y. Youn, Y.-H. Park, T. Kwon, S. Kwon, and J. Jongin Lim, "Efficient flexible batch signing techniques for imbalanced communication applications," *IEICE TRANS. INF. & SYST.*, pp. 1481–1484, May 2008.

[8] R. Rajagopalan and P. Varshney, "Data-aggregation techniques in sensor networks: a survey," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 4, pp. 48–63, 2006.

[9] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 457.

[10] M. X. Cheng and L. Yin, "Energy efficient data gathering algorithm in sensor networks with partial aggregation," *Int. J. Sen. Netw.*, vol. 4, no. 1/2, pp. 48–54, 2008.

[11] W. Liang and Y. Liu, "Online data gathering for maximizing network lifetime in sensor networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 1, pp. 2–11, 2007.

[12] "IEEE std. 802.15.4 - 2003: Wireless medium access control (mac) and physical layer (PHY) specifications for low rate wireless personal area networks (LR-WPANs)," http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf.

[13] "Chipcon products, 2.4ghz IEEE 802.15.4 ZigBee-ready RF transceiver," http://www.stanford.edu/class/cs244e/papers/cc2420.pdf.

[14] "TEP: 126: CC2420 radio stack," http://www.tinyos.net/tinyos-2.x/doc/html/tep126.html.

[15] H. Kobayashi and B. L. Mark, *System Modeling and Analysis: Foundation of System Performance Evaluation*. Prentice Hall, 2009.

[16] A. O. Allen, *Probability, Statistics, and Queueing Theory with Computer Science Applications*, 2nd ed. Academic Press, 1990.

[17] N. Bisnik and A. Abouzeid, "Queuing delay and achievable throughput in random access wireless ad hoc networks," in *The 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON)*, vol. 3, Sept. 2006, pp. 874 – 880.

[18] J. Vardakas, I. Papapanagiotou, M. Logothetis, and S. Kotsopoulos, "On the end-to-end delay analysis of the ieee 802.11 distributed coordination function," in *Second International Conference on Internet Monitoring and Protection (ICIMP)*, July 2007.

[19] K. Ghaboosi, M. Latva-aho, Y. Xiao, and B. Khalaj, "Ieee 802.11 distributed coordination function service time and queuing delay analysis using parallel space - time markov chain," in *IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sept. 2008, pp. 1–5.

[20] O. Tickoo and O. Sikdar, "Modeling queueing and channel access delay in unsaturated ieee 802.11 random access mac based wireless networks," *IEEE/ACM Transactions on Networking*, vol. 16, no. 4, pp. 878 – 891, Aug. 2008.

[21] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathmematics: A Foundation for Computer Science*. Addison-Wesley, Dec. 2006, Twentieth Printing.

[22] L. De Vito, S. Rapuano, and L. Tomaciello, "One-way delay measurement: State of the art," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 12, pp. 2742–2750, Dec. 2008.

[23] A. Kopke, A. Willig, and H. Karl, "Chaotic maps as parsimonious bit error models of wireless channels," in *Proceedings of the INFOCOM 2003 Conference*, vol. 1, 30 March-3 April 2003, pp. 513 – 523.