Reliable Bursty Convergecast in Wireless Sensor Networks

Hongwei Zhang[†], Anish Arora[†], Young-ri Choi[‡], Mohamed G. Gouda[‡]

Abstract

We address the challenges of bursty convergecast in multi-hop wireless sensor networks, where a large burst of packets from different locations needs to be transported reliably and in real-time to a base station. Via experiments on a 49 MICA2 mote sensor network using a realistic traffic trace, we determine the primary issues in bursty convergecast, and accordingly design a protocol, RBC (for Reliable Bursty Convergecast), to address these issues: To improve channel utilization and to reduce ack-loss, we design a window-less block acknowledgment scheme that guarantees continuous packet forwarding and replicates the acknowledgment for a packet; to alleviate retransmission-incurred channel contention, we introduce differentiated contention control. Moreover, we design mechanisms to handle varying ack-delay and to reduce delay in timer-based retransmissions. We evaluate RBC, again via experiments, and show that compared to a commonly used implicit-ack scheme, RBC doubles packet delivery ratio and reduces end-to-end delay by an order of magnitude, as a result of which RBC achieves a close-to-optimal goodput.

***Keywords*—wireless sensor network, bursty convergecast, reliability, real-time, error control, contention control, System design, experimentation with real networks and testbeds**

1 Introduction

A typical application of wireless sensor networks is to monitor an environment (be it an agricultural field or a classified area) for events that are of interest to the users. Usually, the events are rare. Yet when an event occurs, a large burst of packets is often generated that needs to be transported reliably and in real-time to a base station.One exemplary event-driven application is demonstrated in the DARPA NEST field experiment "A Line in the Sand" (simply called *Lites* hereafter) [5]. In Lites, a typical event generates up to 100 packets within a few seconds and the packets need to be transported from different network locations to a base station, over multi-hop routes.

The high-volume bursty traffic in event-driven applications poses special challenges for reliable and real-time packet delivery. The large number of packets generated within a short period leads to high degree of channel contention and thus a high probability of packet collision. The situation is further exacerbated by the fact that packets travel over multi-hop routes: first, the total number of packets competing for channel access is increased by a factor of the average hop-count of network routes; second, the probability of packet collision increases in multi-hop networks due to problems such as hidden-terminals. Consequently, packets are lost with high probability in bursty convergecast. For example, with the default radio stack of TinyOS [3], around 50% of packets are lost for most events in Lites.

For real-time packet delivery, hop-by-hop packet recovery is usually preferred over end-to-end recovery [16, 18]; and this is especially the case when 100% packet delivery is not required (for instance, for bursty convergecast in sensor networks). Nevertheless, we find issues with existing hop-by-hop control mechanisms in bursty convergecast. Via experiments with a testbed of 49 MICA2 motes and with traffic traces of Lites, we observe that the commonly used link-layer error control mechanisms do not significantly improve and can even degenerate packet delivery reliability. For example, when packets are retransmitted up to twice at each hop, the overall packet delivery ratio increases by only 6.15%; and when the number of retranmissions increases, the packet delivery ratio actually decreases, by 11.33%.

One issue with existing hop-by-hop control mechanisms is that they do not schedule packet retransmis-

This work was sponsored by DARPA contract OSU-RF #F33615-01-C-1901.

[†] H. Zhang and A. Arora are with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210, U.S.A. E-mail: {zhangho, anish}@cse.ohio-state.edu.

[‡] Y. Choi and M. Gouda are with the Department of Computer Sciences, The University of Texas at Austin. Austin, TX 78712-0233, U.S.A. E-mail:{yrchoi, gouda}@cs.utexas.edu.

sions appropriately; as a result, retransmitted packets further increase the channel contention and cause more packet loss. Moreover, due to in-order packet delivery and conservative retransmission timers, packet delivery can be significantly delayed in existing hop-by-hop mechanisms, which leads to packet backlogging and reduction in network throughput. (We examine the details in Section 3.)

On the other hand, the new network and application models of bursty convergecast in sensor networks offer unique opportunities for reliable and real-time transport control:

- First, the broadcast nature of wireless channels enables a node to determine, by snooping the channel, whether its packets are received and forwarded by its neighbors.
- Second, high-precision time synchronization and the fact that data packets are timestamped relieve transport layer from the constraint of in-order packet delivery, since applications can determine the order of packets by their timestamps.

Therefore, techniques that take advantage of these opportunities and meet the challenges of reliable and real-time bursty convergecast are desired.

Contributions of the paper. We study the limitations of two commonly used hop-by-hop packet recovery schemes in bursty convergecast. We discover that the lack of retransmission scheduling in both schemes makes retransmission-based packet recovery ineffective in the case of bursty convergecast. Moreover, in-order packet delivery makes the communication channel under-utilized in the presence of packet- and ack-loss.

To address the challenges, we design protocol RBC (for Reliable Bursty Convergecast). Taking advantage of the unique sensor network models, RBC features the following mechanisms:

- To improve channel utilization, RBC uses a window-less block acknowledgment scheme that enables continuous packet forwarding in the presence of packet- and ack-loss. The block acknowledgment also reduces the probability of ack-loss, by replicating the acknowledgment for a received packet.
- To ameliorate retransmission-incurred channel contention, RBC introduces differentiated contention control, which ranks nodes by their queuing conditions as well as the number of times that the enqueued packets have been transmitted. A node ranked the highest within its neighborhood accesses the channel first.

In addition, we design techniques that address the challenges of timer-based retransmission control in bursty convergecast:

- To deal with continuously changing ack-delay, RBC uses adaptive retransmission timer which adjusts itself as network state changes.
- To reduce delay in timer-based retransmission and to expedite retransmission of lost packets, RBC uses block-NACK, retransmission timer reset, and channel utilization protection.

We evaluate RBC by experimenting with an outdoor testbed of 49 MICA2 motes and with realistic traffic trace from the field sensor network of Lites. Our experimental results show that, compared with a commonly used implicit-ack scheme, RBC increases the packet delivery ratio by a factor of 2.05 and reduces the packet delivery delay by a factor of 10.91. Moreover, RBC achieves a goodput of 6.37 packets/second for the traffic trace of Lites, almost reaching the optimal goodput — 6.66 packets/second — for the trace.

Organization of the paper. We describe our testbed and discuss the experiment design in Section 2. In Section 3, we study the limitations of existing hop-by-hop control mechanisms. We present the detailed design of RBC in Section 4, then we present the experimental results in Section 5. We discuss related works in Section 6, and we make concluding remarks in Section 7.

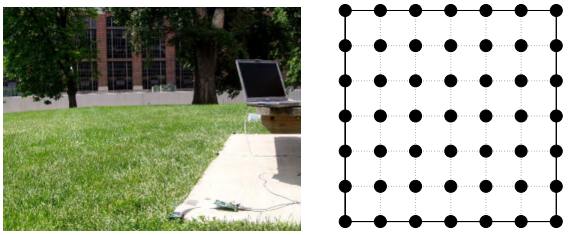
2 Testbed and experiment design

Towards characterizing the issues in making bursty convergecast both reliable and timely, we conduct an experimental study. We choose experimentation as opposed to simulation in order to gain higher fidelity and confidence in the observations. Before presenting our study, we first describe our testbed and the experiment design.

Testbed. We setup our testbed to reflect the field sensor network of Lites, and we use the traffic trace for a typical event in Lites as the basis of our experiments.

The testbed consists of 49 MICA2 motes deployed in a grass field, as shown in Figure 1(a), where the grass is 2-4 inches tall.The 49 motes form a 7×7 grid with a 5-feet separation between neighboring grid points, as shown in Figure 1(b) where each grid point represents a mote. The mote at the left-bottom corner of the grid is the base station to which all the other motes send packets. The 7×7 grid imitates a subgrid in the sensor network of Lites.

The traffic trace (simply called *Lites trace* hereafter)



(a) Environment (b) Grid topology

Figure 1: The testbed

corresponds to the packets generated in a 7×7 subgrid of the Lites network when a vehicle passes across the middle of the Lites network. When the vehicle passes by, each mote except for the base station detects the vehicle and generates two packets, which correspond to the start and the end of the event detection respectively and are separated 5-6 seconds on average. Overall, 96 packets are generated each time the vehicle passes by.[1] The cumulative distribution of the number of packets generated during the event is shown in Figure 2. (Interested readers can find the detailed description of the traffic trace in [4].)

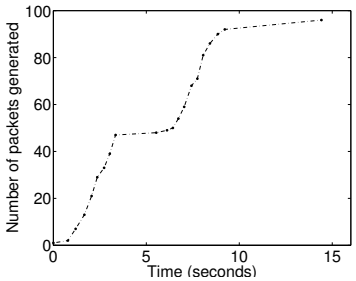


Figure 2: The distribution of packets generated in Lites trace

If we define the burst rate up to a moment in the event as the number of packets generated so far divided by the time since the first packet is generated, the highest burst rate in Lites trace is 14.07 packets/second. Given that the highest one-hop throughput is about 42.93 packets/second for MICA2 motes with B-MAC (the latest MAC component of TinyOS) and that, in multi-hop networks, even an ideal MAC can only achieve $\frac{1}{4}$ of the throughput that a single-hop transmission can achieve [11], the burst rate of Lites trace far exceeds the rate at which the motes can push packets to the base station. Therefore, it is a challenging task to deliver packets reliably and in real time in such a heavy-load bursty traffic scenario.

Experiment design. To reflect the multi-hop network of Lites, we let each mote transmit at the minimum power level by which two motes 10 feet apart are able to reliably communicate with each other, and the power level is 9 (out of a range between 1 and 255). We use the routing protocol LGR [7] in our testbed.[2] LGR uses links that are reliable in the presence of bursty traffic, and LGR spreads traffic uniformly across different paths to reduce wireless channel contention. Therefore, LGR provides a reliable and uniform packet delivery service in bursty convergecast [7]. In our testbed, the number of hops in a path is up to 6 and is 3.3 on average.

For each protocol we evaluate, we run the Lites trace 10 times and measure the average performance of the protocol by the following metrics:

- *Event reliability (ER)*: the number of unique packets received at the base station in an event divided by the number packets generated for the event.

  Event reliability reflects how well an event is reported to the base station.
- *Packet delivery delay (PD)*: the time taken for a packet to reach the base station from the node that generates it.
- *Event goodput (EG)*: the number of unique packets received at the base station divided by the interval between the moment the first packet is generated and the moment the base station receives any packet the last time.

  Event goodput reflects how fast the traffic of an event is pushed from the network to the base station. By definition, the optimal event goodput for Lites trace is 6.66 packets/second, which corresponds to the case where the packet delivery delay is 0 and all the packets are received by the base station.
- *Node reliability (NR)*: the number of unique packets that are generated by a node and received by the base station divided by the number of packets generated at the node.

(Remark: The study in this paper applies to cases where network topologies other than grid and routing protocols other than LGR are used, since the protocols studied are independent of the network topology and the routing protocol.)

[1] We could have chosen a traffic trace where fewer number packets are generated (e.g., when a soldier with a gun passes by), but that would not serve as well in showing the challenges posed by huge event traffic bursts.

[2] To focus on transport issues,we disable the "base-snooping" in LGR so that the base station does not accept packets snooped over the channel.

3 Limitations of two hop-by-hop packet recovery mechanisms

Two widely used hop-by-hop packet recovery mechanisms in sensor networks are synchronous explicit ack and stop-and-wait implicit ack. We study their performance in bursty convergecast as follows.

3.1 Synchronous explicit ack (SEA)

In SEA, a receiver switches to transmit-mode, immediately after receiving a packet, and sends back the acknowledgment without going through the procedure of channel access control (since the current transmission is preventing other nearby nodes from accessing the channel); the sender immediately retransmits a packet if the corresponding ack is not received after certain constant time. Using our testbed, we study the performance of SEA when used with B-MAC [13, 3] and S-MAC [21]. B-MAC uses the mechanism of CSMA/CA (carrier sense multiple access with collision avoidance) to control channel access; S-MAC uses CSMA/CA too, but it also employs RTS-CTS handshake to reduce the impact of hidden terminals.

SEA with B-MAC. The event reliability, the average packet delivery delay, as well as the event goodput is shown in Table 1, where RT stands for the maximum

Metrics	RT = 0	RT = 1	RT = 2
ER (%)	51.05	54.74	54.63
PD (seconds)	0.21	0.25	0.26
EG (packets/sec)	4.01	4.05	3.63

Table 1: SEA with B-MAC in Lites trace

number of retransmissions for each packet at each hop (e.g., RT = 0 means that packets are not retransmitted). The distribution of the number of unique packets received at the base station along time is shown in Figure 3.

Table 1 and Figure 3 show that when packets are retransmitted, the event reliability increases slightly (i.e., by up to 3.69%). Nevertheless, the maximum reliability is still only 54.74%, and, even worse, the event reliability as well as goodput decreases when the maximum number of retransmissions increases from 1 to 2. (The above data is for B-MAC with its default contention window size. We have conducted the experiment with different contention window size of B-MAC, and we found that the performance pattern remains the same.)

SEA with S-MAC. Unlike B-MAC, S-MAC uses

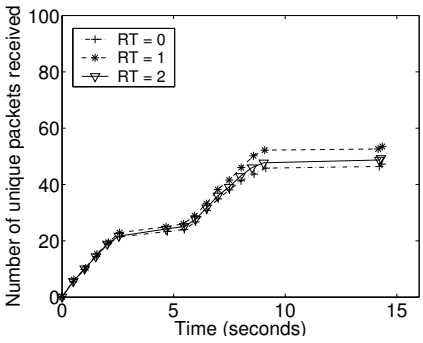


Figure 3: The distribution of packet reception in SEA with B-MAC

RTS-CTS handshake for unicast transmissions, which reduces packet collisions. We evaluate SEA when it is used with S-MAC, and the performance data is shown in Table 2 and Figure 4.

Metrics	RT = 0	RT = 1	RT = 2
ER (%)	72.6	74.79	70.1
PD (seconds)	0.17	0.183	0.182
EG (packets/sec)	5.01	4.68	4.37

Table 2: SEA with S-MAC in Lites trace

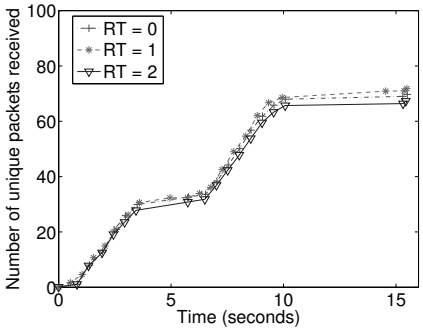


Figure 4: The distribution of packet reception in SEA with S-MAC

Compared with B-MAC, RTS-CTS handshake improves the event reliability by about 20% in S-MAC. Yet packet retransmissions still do not significantly improve the event reliability and can even decrease the reliability.

Analysis. We find that the reason why retransmission does not significantly improve — and can even degenerate — communication reliability is that, in SEA, lost packets are retransmitted while new packets are gen-

erated and forwarded, thus retransmissions, when not scheduled appropriately, only increase channel contention and cause more packet collision.[3] The situation is further exacerbated by ack-loss (with a probability as high as 10.29%), since ack-loss causes unnecessary retransmission of packets that have been received. To make retransmission effective in improving reliability, therefore, we need a retransmission scheduling mechanism that ameliorates retransmission-incurred channel contention.

3.2 Stop-and-wait implicit ack (SWIA)

SWIA takes advantage of the fact that every node, except for the base station, forwards the packet it receives and the forwarded packet can act as the acknowledgment to the sender at the previous hop [12]. In SWIA, the sender of a packet snoops the channel to check whether the packet is forwarded within certain constant threshold time; the sender regards the packet as received if it is forwarded within the threshold time, otherwise the packet is regarded as lost. The advantage of SWIA is that acknowledgment comes for free except for the limited control information piggybacked in data packets.

We evaluate SWIA only with B-MAC, given that the implementation of S-MAC is not readily applicable for packet snooping. The performance results are shown in Table 3 and in Figure 5.

Metrics	RT = 0	RT = 1	RT = 2
ER (%)	43.09	31.76	46.5
PD (seconds)	0.35	8.81	18.77
EG (packets/sec)	3.48	2.58	1.41

Table 3: SWIA with B-MAC in Lites trace

We see that the maximum event reliability in SWIA is only 46.5%, and that the reliability decreases significantly when packets are retransmitted at most once at each hop. When packets are retransmitted up to twice at each hop, the packet delivery delay increases, and the event goodput decreases significantly despite the slightly increased reliability.

Analysis. We find that the above phenomena are due to the following reasons. First, the length of data packets is increased by the piggybacked control information in SWIA, thus the ack-loss probability increases (as high as 18.39% in our experiments), which in turn increases unnecessary retransmissions. Second,

[3]This is not the case in wireline networks and is due to the nature of wireless communications.

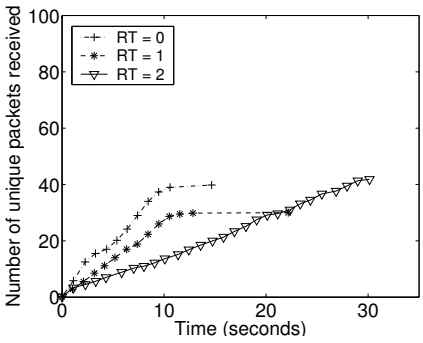


Figure 5: The distribution of packet reception in SWIA with B-MAC

most packets are queued upon reception and thus their forwarding is delayed. As a result, the piggybacked acknowledgments are delayed and the corresponding packets are retransmitted unnecessarily. Third, once a packet is waiting to be acknowledged, all the packets arriving later cannot be forwarded even if the communication channel is free. Therefore, channel utilization as well as system throughput decreases, and network queuing as well as packet delivery delay increases. Fourth, as in SEA, lack of retransmission scheduling allows retransmissions, be it necessary or unnecessary, to cause more channel contention and packet loss.

4 Protocol RBC

To address the limitations of SEA and SWIA in bursty convergecast, we design protocol RBC. In RBC, we design a window-less block acknowledgment scheme to increase channel utilization and to reduce the probability of ack-loss. We also design a distributed contention control scheme that schedules packet retransmissions and reduces the contention between newly generated and retransmitted packets. Moreover, we design mechanisms to address the challenges of bursty convergecast on timer-based retransmission (such as varying ack-delay and timer-incurred delay).

Given that the number of packets competing for channel access is less in implicit-ack based schemes than in explicit-ack based schemes, we design RBC based on the paradigm of implicit-ack (i.e., piggybacking control information in data packets). We elaborate on RBC as follows. (Even though the mechanisms used in RBC can be applied in the explicit-ack paradigm, we relegate the detailed study as our future work.)

4.1 Window-less block acknowledgment

In traditional block acknowledgment [6], a sliding-window is used for both duplicate detection and in-order packet delivery.[4] The sliding-window reduces network throughput once a packet is sent but remains unacknowledged (since the sender can only send up to its window size once a packet is unacknowledged), and in-order delivery increases packet delivery delay once a packet is lost (since the lost packet delays the delivery of every packet behind it). Therefore, the sliding-window based block acknowledgment scheme does not apply to bursty convergecast, given the real-time requirement of the latter.

To address the constraints of traditional block acknowledgment in the presence of unreliable links, we take advantage of the fact that in-order delivery is not required in bursty convergecast. Without considering the order of packet delivery, by which we only need to detect whether a sequence of packets are received without loss in the middle and whether a received packet is a duplicate of a previously received one. To this end, we design, as follows, a *window-less block acknowledgment* scheme which guarantees continuous packet forwarding irrespective of the underlying link unreliability as well as the resulting packet- and ack-loss. For clarity of presentation, we consider an arbitrary pair of nodes S and R where S is the sender and R is the receiver.

Window-less queue management. The sender S organizes its packet queue as $(M + 2)$ linked lists, as shown in Figure 6, where M is the maximum number of retransmissions at each hop. For convenience, we call the linked lists *virtual queues*, denoted as $Q_0, \ldots, Q_{M+1}$. The virtual queues are ranked such that a virtual queue Q_k ranks higher than Q_j if $k < j$.

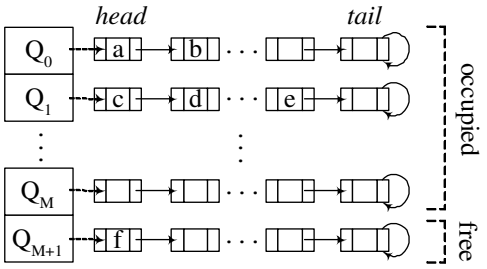


Figure 6: Virtual queues at a node

Virtual queues $Q_0, Q_1, \ldots$, and Q_M buffer packets waiting to be sent or to be acknowledged, and Q_{M+1} collects the list of free queue buffers. The virtual queues are maintained as follows:

- When a new packet arrives at S to be sent, S detaches the head buffer of Q_{M+1}, if any, stores the packet into the queue buffer, and attaches the queue buffer to the tail of Q_0.
- Packets stored in a virtual queue Q_k $(k > 0)$ will not be sent unless Q_{k-1} is empty; packets in the same virtual queue are sent in FIFO order.
- After a packet in a virtual queue Q_k $(k \geq 0)$ is sent, the corresponding queue buffer is moved to the tail of Q_{k+1}, unless the packet has been retransmitted M times[5] in which case the queue buffer is moved to the tail of Q_{M+1}.
- When a packet is acknowledged to have been received, the buffer holding the packet is released and moved to the tail of Q_{M+1}.

Therefore, the order in which unacknowledged packets have been sent is maintained in the virtual queues without any window-based control, providing the basis for window-less block acknowledgment. Moreover, newly arrived packets can be sent immediately without waiting for the previously sent packets to be acknowledged, which enables continuous packet forwarding in the presence of packet- and ack-loss.

Block acknowledgment & reduced ack-loss. Each queue buffer at S has an ID that is unique at S. When S sends a packet to the receiver R, S attaches the ID of the buffer holding the packet as well as the ID of the buffer holding the packet to be sent next. In Figure 6, for example, when S sends the packet in buffer a, S attaches the values a and b. Given the queue maintenance procedure, if the buffer holding the packet being sent is the tail of Q_0 or the head of a virtual queue other than Q_0, S also attaches the ID of the head buffer of Q_{M+1}, if any, since one or more new packets may arrive before the next enqueued packet is sent in which case the newly arrived packet(s) will be sent first. For example, when the packet in buffer c of Figure 6 is sent, S attaches the values c, d, and f.

When the receiver R receives a packet p_0 from S, R learns the ID n' of the buffer holding the next packet to be sent by S. When R receives a packet p_n from S next time, R checks whether p_n is from buffer n' at S: if p_n is from buffer n', R knows that there is no packet loss between receiving p_0 and p_n from S; otherwise, R detects that some packets are lost between p_0 and p_n.

For each maximal sequence of packets $p_k, \ldots, p_{k'}$ from S that are received at R without any loss in the

[4]Note that SWIA is a special type of block acknowledgment where the window size is 1.

[5]Due to block-NACK, to be discussed in Section 4.3.2, a packet having been retransmitted M times may be in a virtual queue other than Q_M.

middle, R attaches to packet $p_{k'}$ the 2-tuple $\langle q_k, q_{k'} \rangle$, where q_k and $q_{k'}$ are the IDs of the buffers storing p_k and $p_{k'}$ at S. We call $\langle q_k, q_{k'} \rangle$ the *block acknowledgment* for packets $p_k, \ldots, p_{k'}$. When S snoops the forwarded packet $p_{k'}$ later, S learns that all the packets sent between p_k and $p_{k'}$ have been received by R. Then S releases the buffers holding these packets. For example, if S snoops a block acknowledgment $\langle c, e \rangle$ when its queue state is as shown in Figure 6, S knows that all the packets in buffers between c and e in Q_1 have been received, and S releases buffers between c and e, including c and e.

One delicate detail in processing the block acknowledgment $\langle q_k, q_{k'} \rangle$ is that after releasing buffer q_k, S will maintain a mapping $q_k \leftrightarrow q_{k''}$, where $q_{k''}$ is the buffer holding the packet sent (or to be sent next) after that in $q_{k'}$. When S snoops another block acknowledgment $\langle q_k, q_n \rangle$ later, S knows, by $q_k \leftrightarrow q_{k''}$, that packets sent between those in buffers $q_{k''}$ and q_n have been received by R; then S releases the buffers holding these packets, and S resets the mapping to $q_k \leftrightarrow q_{n''}$, where $q_{n''}$ is the buffer holding the packet sent (or to be sent next) after that in q_n. S maintains the mapping for q_k until S receives a block-NACK (to be discussed in Section 4.3.2) or a block acknowledgment $\langle q, q' \rangle$ where $q \neq q_k$.

In the above block acknowledgment scheme, the acknowledgment for a received packet is piggybacked onto the packet itself as well as the packets that are received consecutively after the packet without any loss in the middle. Therefore, the acknowledgment is replicated and the probability for it to be lost decreases significantly, by a factor of 2.07 in Lites trace as analyzed below.

Analysis of ack-loss probability. For convenience, we define the following notations:

p	:	the probability of losing a single (data) packet;
N	:	the number of packets received in succession without any loss in the middle;
N'	:	the number of packets lost in succession;
B	:	the number of packets received in succession without any loss in the middle, after a packet is already received;
A	:	the number of times that the acknowledgment for a packet is received at the sender.

Assuming that packet losses are independent of one another, we have the probability mass functions for random variables N and N' as follows.

$$\begin{aligned} P[N = k] &= p(1-p)^k \\ P[N' = k] &= (1-p)p^k \end{aligned}$$

In RBC, when a packet m is received at a receiver R, the acknowledgment for m can reach back to the sender S in two ways: S snoops m when it is forwarded by R later, with probability P_{self}; or S does not snoop m but snoops a packet whose block acknowledgment acknowledges the reception of m, with probability P_{ba}. Therefore, the probability P_{rbc} of S receiving the acknowledgment for m can be derived as follows:

$$\begin{aligned} P_{self} &= 1-p \\ P_{ba} &= p\sum_{k=0}^{\infty} P[B=k]P[A \geq 1 | B=k] \\ &= p\sum_{k=0}^{\infty} P[B=k](1 - P[A=0|B=k]) \\ &= p\sum_{k=0}^{\infty} P[N=k+1](1 - P[N'=k]) \\ &= \frac{p(1-3p+4p^2-2p^3)}{1-p+p^2} \\ P_{rbc} &= P_{self} + P_{ba} \\ &= 1 - p + \frac{p(1-3p+4p^2-2p^3)}{1-p+p^2} \end{aligned}$$

Then, the probability P'_{rbc} of losing the acknowledgment for a packet in RBC is $1 - P_{rbc}$.

In the case of Lites trace and implicit-ack, $p = 22.7\%$. Therefore $P'_{rbc} = 8.89\%$, reducing the ack-loss probability of SWIA by a factor of 2.07.

□

Duplicate detection & obsolete-ack filtering. Since it is impossible to completely prevent ack-loss in lossy communication channels, packets whose acknowledgments are lost will be retransmitted unnecessarily. Therefore, it is necessary that duplicate packets be detected and dropped.

To enable duplicate detection, the sender S maintains a counter for each queue buffer, whose value is incremented by one each time a new packet is stored in the buffer. When S sends a packet, it attaches the current value of the corresponding buffer counter. For each buffer q at S, the receiver R maintains the counter value c_q piggybacked in the last packet from the buffer. When R receives another packet from the buffer q later, R checks whether the counter value piggybacked in the packet equals to c_q: if they are equal, R knows that the packet is a duplicate and drops it; otherwise R regards the packet as a new one and accepts it. The duplicate detection is local in the sense that it only requires information local to each queue buffer instead of imposing any rule involving different buffers (such as in sliding-window) that can degenerate system performance.

For the correctness of the above duplicate detection mechanism, we only need to choose the domain size C for the counter value such that the probability of losing C packets in succession is negligible. For example, for the high per-hop packet loss probability 22.7% in the case of Lites trace, C could still be as small as 7, since the probability of losing 7 packets in succession is only 0.003%.[6]

[6]Given the small domain size for the counter value as well as the usually small queue size at each node, the duplicate detection mechanism does not

In addition to duplicate detection, we also use buffer counter to filter out obsolete acknowledgment. Despite the low probability, packet forwarding at R may be severely delayed, such that the queue buffers signified in a block acknowledgment have been reused by S to hold packets arriving later. To deal with this, R attaches to each forwarded packet the ID as well as the counter value of the buffer holding the packet at S originally; when S snoops a packet forwarded by R, S checks whether the piggybacked counter value equals to the current value of the corresponding buffer: if they are equal, S regards as valid the piggybacked block acknowledgment; otherwise, S regards the block acknowledgment as obsolete and ignores it.

Aggregated-ack at the base station. In sensor networks, the base station usually forwards all the packets it receives to an external network. As a result, the children of the base station (i.e., the nodes that forward packets directly to the base station) are unable to snoop the packets the base station forwards, and the base station has to explicitly acknowledge the packets it receives. To reduce channel contention, the base station aggregates several acknowledgments, for packets received consecutively in a short period of time, into a single packet and broadcasts the packet to its children. Accordingly, the children of the base station adapt their control parameters to the way the base station handles acknowledgments.

4.2 Differentiated contention control

In wireless sensor networks where per-hop connectivity is reliable, most packet losses are due to collision in the presence of severe channel contention. To enable reliable packet delivery, lost packets need to be retransmitted. Nevertheless, packet retransmission may cause more channel contention and packet loss, thus degenerating communication reliability. Also, there exist unnecessary retransmissions due to ack-loss, which only increase channel contention and reduce communication reliability. Therefore, it is desirable to schedule packet retransmissions such that they do not interfere with transmissions of other packets.

The way the virtual queues are maintained in our window-less block acknowledgment scheme facilitates the retransmission scheduling, since packets are automatically grouped together by different virtual queues. Packets in higher-ranked virtual queues have been transmitted less number of times, and the probability that the receiver has already received the packets in higher-ranked virtual queues is lower (e.g., 0 for packets in Q_0). Therefore, we rank packets by the rank of the virtual queues holding the packets, and higher-ranked packets have higher-priority in accessing the communication channel. By this rule, packets that have been transmitted less number of times will be (re)transmitted earlier than those that have been transmitted more, and interference between packets of different ranks is reduced.

Window-less block acknowledgment already handles packet differentiation and scheduling within a node, thus we only need a mechanism that schedules packet transmission across different nodes. To reduce interference between packets of the same rank and to balance network queuing as well as channel contention across nodes, inter-node packet scheduling also takes into account the number of packets of a certain rank so that nodes having more such packets transmit earlier.

To implement the above concepts, we define the rank $rank(j)$ of a node j as $\langle M - k, |Q_k|, ID(j)\rangle$, where Q_k is the highest-ranked non-empty virtual queue at j, $|Q_k|$ is the number of packets in Q_k, and $ID(j)$ is the ID of j. A node with a larger rank value ranks higher.[7] Then, the distributed transmission scheduling works as follows:

- Each node piggybacks its rank to the data packets it sends out.
- Upon snooping or receiving a packet, a node j compares its rank with that of the packet sender k. j will change its behavior only if k ranks higher than j, in which case j will not send any packet in the following $w(j,k) \times T_{pkt}$ time. T_{pkt} is the time taken to transmit a packet at the MAC layer, and $w(j,k) = 4 - i$, when $rank(j)$ and $rank(k)$ differ at the i-th element of the 3-tuple ranks. $w(j,k)$ is defined such that the probability of all waiting nodes starting their transmissions simultaneously is reduced, and that higher-ranked nodes tend to wait for shorter time. T_{pkt} is estimated by the method of *Exponentially Weighted Moving Average (EWMA)*.
- If a sending node j detects that it will not send its next packet within T_{pkt} time (i.e., when j knows that, after the current packet transmission, it will rank lower than another node), j signifies this by marking the packet being sent, so that the

consume much memory. For example, it only takes 36 bytes in the case of Lites.

[7] The concept of "rank" is defined such that the first field guarantees that packets having been transmitted less number of times will be (re)transmitted earlier, the second field ensures that nodes having more packets enqueued get chances to transmit earlier, and the third field is to break ties in the first two fields.

nodes overhearing the packet will skip j in the contention control. (This mechanism reduces the probability of idle waiting, where the channel is free but no packet is sent.)

4.3 Timer management in window-less block acknowledgment

In window-less block acknowledgment, a sender S starts a retransmission timer after sending a packet, and S retransmits the packet if S has not received the corresponding ack when the timer times out. Retransmission timers directly affect the reliability and the delay in packet delivery: large timeout values of timers tend to increase packet delivery delay, whereas small timeout values tend to cause unnecessary retransmissions and thus decrease packet delivery reliability. To provide reliable and real-time packet delivery, we design mechanisms to manage the timers in window-less block acknowledgment as follows. (Again, we consider a sender S and a receiver R.)

4.3.1 Dealing with varying ack-delay

When the receiver R receives a packet m from the sender S, R first buffers m in Q_0. The delay in R forwarding m depends on the number of packets in front of m in Q_0. Since the number of packets enqueued in Q_0 keeps changing, the delay in forwarding a received packet by R keeps changing, which leads to varying delay in packet acknowledgment. Therefore, the retransmission timer at the sender S should adapt to the queuing condition at R; otherwise, either lost packets are unnecessarily delayed in retransmission (when the retransmission timer is too large) or packets are unnecessarily retransmitted even if they are received (when the retransmission timer is too small).

To adaptively setting the retransmission timer for a packet, the sender S keeps track of, by snooping packets forwarded by R, the length s_r of Q_0 at R, the average delay d_r in R forwarding a packet after the packet becomes the head of Q_0, and the deviation d'_r of d_r. When S sends a packet to R, S sets the retransmission timer of the packet as

$$(s_r + C_0)(d_r + 4d'_r)$$

where C_0 is a constant denoting the number of new packets that R may have received since S learned s_r the last time (C_0 depends on the application as well as the link reliability, and C_0 is 3 in our experiments). The reason why we use the deviation d'_r in the above formula is that d_r varies a lot in wireless networks in the presence of bursty traffic, in which case the deviation improves estimation quality [10].

At a node, each local parameter α (such as d_r for node R) and its deviation α' are estimated by the method of EWMA as follows:

$$\alpha \leftarrow (1-\gamma)\alpha + \gamma\alpha''$$
$$\alpha' \leftarrow (1-\gamma')\alpha' + \gamma'|\alpha'' - \alpha|$$

where γ and γ' are weight-factors, and α'' is the latest observation of α. Empirically, we set $\gamma = \frac{1}{8}$ and $\gamma' = \frac{1}{4}$ in RBC.

4.3.2 Alleviating timer-incurred delay

The packet retransmission timer calculated as above is conservative in the sense that it is usually greater than the actual ack-delay [10]. This is important for reducing the probability of unnecessary retransmissions, but it introduces extra delay and makes network resources under-utilized [22].

To alleviate timer-incurred delay, we design the following mechanisms to expedite necessary packet retransmissions and to improve channel utilization:

- Whenever the receiver R receives a packet m from buffer n of the sender S while R is expecting (in the absence of packet loss) to receive a packet from buffer n' of S, R learns that packets sent between those in buffers n' and n at S, including the one in n', are lost. In this case, R piggybacks a block-NACK $[n', n)$ onto the next packet it forwards, by which the block-NACK can be snooped by S immediately.

  When S learns the block-NACK $[n', n)$ from R, S resets the retransmission timers to 0 for the packets sent between those in n' and n (including the one in n'), and for each of these packets, S moves the corresponding buffer to the tail of Q_{k-1} if the buffer is currently at Q_k. Therefore, packets that need to be retransmitted are put into higher-ranked virtual queues and are retransmitted quickly.[8]
- Whenever S learns that the virtual queue Q_0 of R becomes empty, S knows that R has forwarded all the packets it has received. In this case, S resets the retransmission timers to 0 for those packets still waiting to be acknowledged, since they will not be (due to either packet-loss or ack-loss).

  Similarly, when S snoops the acknowledgment for a packet m, S resets the retransmission timer to 0 for those packets that are sent before m but are still waiting to be acknowledged.

[8] Note that the movement of NACKed packets do not disrupt the buffering order required by block-acknowledgment.

- When a network channel is fully utilized, it should be busy all the time. Therefore, if the sender S has packets to send, and if S notices that no packet is sent by any neighboring node in a period of $C_1 \times T_{pkt}$ time, S sends out the packet at the head of its highest-ranked non-empty virtual queue, without considering the retransmission timer even if the packet is to be acknowledged.[9] C_1 is a constant reflecting the degree of channel utilization we want and T_{pkt} is the time taken to transmit a packet at the MAC layer.

5 Experimental results

We have implemented protocol RBC in TinyOS using B-MAC,[10] and RBC has been successfully applied in a field sensor network of about 1,000 XSM motes (an enhanced version of MICA2) to support reliable and real-time convergecast [2]. We have also evaluated RBC in our testbed, and the performance results are shown in Table 4. Figure 7 shows the distribution of packet reception in RBC. From Table 4 and Figure 7, we observe the following properties of RBC:

Metrics	RT = 0	RT = 1	RT = 2
ER (%)	56.21	83.16	95.26
PD (seconds)	0.21	1.18	1.72
EG (packets/sec)	4.28	5.72	6.37

Table 4: RBC in Lites trace

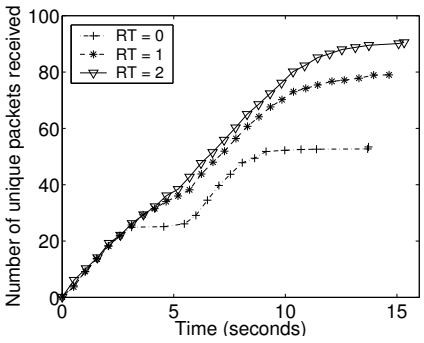


Figure 7: The distribution of packet reception in RBC

- The event reliability keeps increasing, in a significant manner, as the number of retransmissions increases. The increased reliability mainly attributes to reduced unnecessary retransmissions (by reduced ack loss and adaptive retransmission timer) and retransmission scheduling.
- Compared with SWIA which is also based on implicit-ack, RBC reduces packet delivery delay significantly. This mainly attributes to the ability of continuous packet forwarding in the presence of packet- and ack-loss and the reduction in timer-incurred delay.
- The rate of packet reception at the base station and the event goodput keep increasing as the number of retransmissions increases. When packets are retransmitted up to twice at each hop, the event goodput reaches 6.37 packets/second, quite close to the optimal goodput — 6.66 packets/second — for Lites trace.

Compared with SWIA, RBC improves reliability by a factor of 2.05 and reduces average packet delivery delay by a factor of 10.91. Compared to SEA with B-MAC (simply referred to as SEA hereafter), RBC improves reliability by a factor of 1.74, but the average packet delivery delay increases by a factor of 6.61 in RBC. Interestingly, however, RBC still improves the event goodput by a factor of 1.75 when compared with SEA. The reason is that, in RBC, lost packets are retransmitted and delivered after those packets that are generated later but transmitted less number of times. Therefore, the delivery delay for lost packets increases, which increases the average packet delivery delay, without degenerating the system goodput. The observation shows that, due to the unique application models in sensor networks, metrics evaluating aggregate system behaviors (such as the event goodput) tend to be of more relevance than metrics evaluating unit behaviors (such as the delay in delivering each individual packet).

To further understand protocol behaviors in the presence of packet retransmissions, we conduct, as follows, a comparative study of RBC, SWIA, and SEA for the case where packets are retransmitted up to twice at each hop.

Figure 8 compares the distribution of packet generation in Lites trace with the distributions of packet reception in SEA, SWIA, and RBC. We see that the curve for packet reception in RBC smooths out and almost matches that of packet generation. In contrast, many packets are lost in SEA despite the fact that the rate of packet reception in SEA is close to that in RBC; packet delivery is significantly delayed in SWIA, in addition

[9] This mechanism improves channel utilization without introducing unnecessary retransmissions because of the "differentiated contention control" in RBC.

[10] In the implementation, the control logic takes 185 bytes of RAM when each node maintains a buffer capable of holding 16 packets, and the control information piggybacked in data packets takes 14 bytes. We are currently studying methods of reducing the memory consumption and the length of the piggybacked control information, for instance, using explicit acknowledgment.

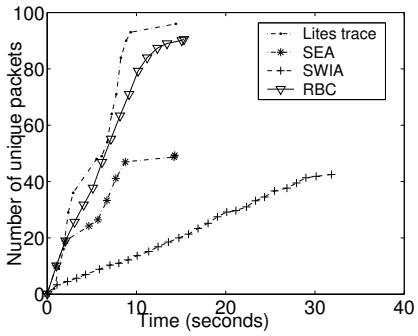


Figure 8: The distributions of packet generation and reception

to the high degree of packet loss.

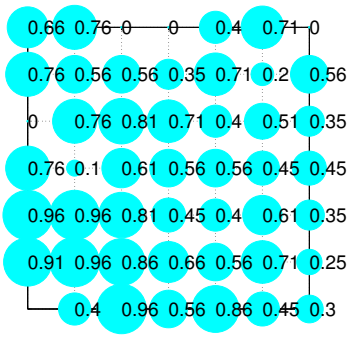
Based on the grid topology as shown in Figure 1(b), Figures 9(a)-(c) show the node reliability in SEA, SWIA, and RBC respectively. Figure 10 shows the cumulative distribution of node reliability in SEA, SWIA, and RBC. We see that node reliability improves significantly in RBC: only 4.17% of nodes have a node reliability less than 80% in RBC; yet in SEA and SWIA, above 80% of nodes have a node reliability less than 80%.

Figure 11 shows the average node reliability in SEA, SWIA, and RBC as the number of routing hops (to the base station) increases. We see that the node reliability in RBC is much higher than that in SEA and SWIA at every routing hop, and that the reliability at the farthest hop in RBC is even greater than that at the closest hop in SEA and SWIA. (Note that, in RBC, the reason why nodes 5 hops away from the base station have lower average delivery rate than nodes 6 hops away can be due to the specific traffic pattern and the difference among nodes' hardware.)

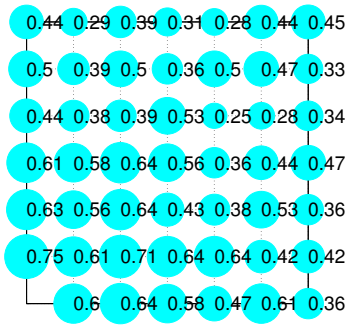
Remark. In this paper, we focus on scenarios where packets are timestamped and thus we do not need to precisely preserve the relative timing between packets as it is when they are generated. Nevertheless, to characterize how RBC affects the relative timing of packets, we measure the *timing-shift* of packet delivery as follows. Given a packet $P1$ received at the base station, the timing-shift for $P1$ is calculated as

$$|(R_1 - R_0) - (S_1 - S_0)|$$

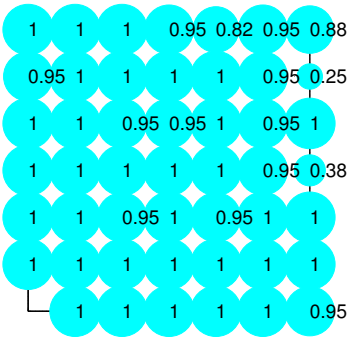
where R_1 denotes the time when $P1$ is received at the base station, R_0 denotes the time when a packet $P0$ is received at the base station immediately before $P1$, S_1 denotes the time when $P1$ is generated at some node in the network, and S_0 denotes the time when $P0$ is



(a) SEA



(b) SWIA



(c) RBC

Figure 9: Node reliability

generated at some node in the network.[11] For convenience, we set the timing-shift to 0 for the first packet received at the base station. Based on this definition,

[11] In this definition, we do not consider the packets that are lost, only calculating the relative timing-shift between packets that are received at the base station.

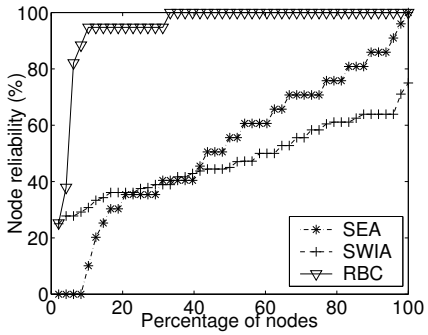


Figure 10: Distribution of node reliability

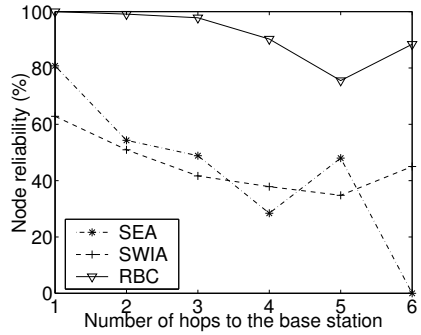


Figure 11: Node reliability as a function of routing hops

we find that the average timing-shift is 1.0035 second for the packets received in the case of RBC, and that the average timing-shift is around 0.1698 second in both SEA and SWIA. Even though the timing-shift in RBC is predictably greater than those in SEA and SWIA, it is still small enough for real-time event-driven applications such as Lites and ExScal [2] where high-level decisions are based on data in the order of seconds. Note that to reduce the timing-shift in packet delivery, we can modify the queue management policy in window-less block acknowledgment, but we relegate a study of this to our future work.

6 Related work

The performance of packet delivery in dense sensor networks has been studied in [23]. The results show that, in the presence of heavy channel load, a commonly used loss recovery scheme at link layer (i.e., lost packets are retransmitted up to 3 times) does not mask packet loss, and more than 50% of the links observe 50% packet loss. The observation shows the challenge of reliable communication over multi-hop routes, since the reliability decreases exponentially as the number of hops increases.

The limitations of timers in TCP retransmission control have been studied in [22]. The author analyzes the intrinsic difficulties in using timers to achieve optimal performance and argues that additional mechanisms should be used. Despite its focus on TCP, the study also applies to retransmission control in sensor networks.

Block acknowledgment [6] has been proposed for error as well as flow control in the Internet. It considers the problem of in-order packet delivery. Therefore, a lost packet blocks the delivery of all the packets that are behind the lost one but have reached the receiver, as a result of which packet delivery delay is increased. Moreover, a sender can send packets at most up to its window size once a packet is sent but unacknowledged, thus the channel resource may be under-utilized. Block acknowledgment also uses timers without addressing their limitations, which can render additional delay in packet delivery.

For packet-loss detection and retransmission control, DFRF [12] uses stop-and-wait implicit ack (SWIA). Yet DFRF does not address the issue of retransmission-incurred channel contention. Moreover, the retransmission timers in DFRF do not adapt to the varying ack-delay, which can introduce more retransmission or delay than necessary. To reduce the number of packet transmissions, DFRF uses raw data aggregation where multiple short packets are concatenated to form a longer packet. In the type of bursty convergecast as experienced by Lites and ExScal [2], it is more difficult to perform data aggregation at the mote level because motes detecting an event can be multiple hops away from one another and the length of a single sensor data entry is more than half of the packet length. Therefore, the current implementation of RBC is based on the paradigm of implicit-ack to reduce the number of packets competing for channel access. On the other hand, we believe that the methodologies developed in RBC (e.g., window-less block acknowledgment and differentiated contention control) can also be applied when there is data aggregation, in which case we can use explicit-ack packets to send out control information. The detailed study on this is a part of our future work.

RMST [16] and PSFQ [18] have shown the importance of hop-by-hop packet recovery in sensor networks. Yet RMST and PSFQ do not focus on bursty convergecast. Therefore, they do not cope with retransmission-incurred channel contention, they do not design mechanisms to alleviate delay incurred by

retransmission timers (whose timeout values are conservatively chosen to reduce unnecessary retransmissions), and they do not design mechanisms to reduce the probability of ack-loss. Our work complements theirs by identifying issues with existing hop-by-hop mechanisms in bursty convergecast and proposing approaches to address the issues.

CODA [19] and ESRT [14] have studied congestion control in sensor networks. They consider a traffic model where multiple sources are continuously or periodically generating packets. Therefore, they do not focus on real-time packet delivery in bursty convergecast, and they do not consider retransmission-incurred delay as well as channel contention. Recent work in [9] and [8] has studied different techniques for mitigating congestion and guaranteeing fairness in wireless sensor networks. Our work complements theirs by focusing on retransmission-based error control and retransmission scheduling. Several transport protocols, such as ATP [17] and WTCP [15], are also proposed for wireless ad hoc networks. Again, they do not face the challenges of reliable bursty convergecast.

7 Concluding remarks

Unlike most existing literature on reliable transport in sensor networks that focuses on periodic traffic, we have focused on bursty convergecast where the key challenges are reliable and real-time error control and the resulting contention control. To address the unique challenges, we have proposed the window-less block acknowledgment scheme which improves channel utilization and reduces ack-loss as well as packet delivery delay; we have also designed mechanisms to schedule packet retransmissions and to reduce timer-incurred delay, which are critical for reliable and real-time transport of bursty traffic. With its well-tested support for reliable and real-time transport of bursty traffic, RBC has been used in a field sensor network experiment where about 1,000 XSM motes are deployed [2].

From protocol RBC, we see that bursty convergecast not only poses challenges for reliable and real-time transport control, it also provides unique opportunities for protocol design. Tolerance of out-of-order packet delivery enables the window-less block acknowledgment, which not only guarantees continuous packet delivery in the presence of packet- and ack-loss but also facilitates retransmission scheduling. Overall, the unique network as well as application models in sensor networks offer opportunities for new methodologies in protocol engineering and are interesting areas for further exploration.

In designing RBC, we have focused on reliable bursty convergecast in event-driven applications. As a part of our future work, we plan to evolve the design of RBC to support reliable and real-time transport of periodic data.

Acknowledgment

We thank Hui Cao for his help with the initial data analysis. We thank other members of our lab for their support in the experimentation. We also appreciate the comments from the anonymous referees.

References

[1] Crossbow technology inc., *http://www.xbow.com*.

[2] Exscal. *http://www.cse.ohio-state.edu/exscal*.

[3] Wireless embedded systems. *http://webs.cs.berkeley.edu*.

[4] A Lites event traffic trace. http://www.cse.ohio-state.edu/∼zhangho/publications/Lites-trace.txt, 2003.

[5] A. Arora and et al. A Line in the Sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks (Elsevier)*, 46(5), 2004.

[6] G. Brown, M. Gouda, and R. Miller. Block acknowledgment: Redesigning the window protocol. In *ACM SIGCOMM*, pages 128–134, 1989.

[7] Y.-R. Choi, M. G. Gouda, H. Zhang, and A. Arora. Routing on a Logical Grid in Sensor Networks. Technical Report TR04-49, Department of Computer Sciences, The University of Texas at Austin *(http://www.cse.ohio-state.edu/∼zhangho/publications/lgr.pdf)*, 2004.

[8] C. T. Ee and R. Bajcsy. Congestion control and fairness for many-to-one routing in sensor networks. In *ACM SenSys*, pages 134–147, 2004.

[9] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *ACM SenSys*, pages 134–147, 2004.

[10] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314–329, 1988.

[11] J. Li, C. Blake, D. D. Couto, H. Lee, and R. Morris. Capacity of ad hoc wireless networks. In *ACM MobiCom*, pages 61–69, 2001.

[12] M. Maroti. The directed flood routing framework. In *Technical report, Vanderbilt University, ISIS-04-502*, 2004.

[13] J. Polatre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *ACM SenSys*, 2004.

[14] Y. Sankarasubramaniam, O. Akan, and I. Akyildiz. ESRT: Event-to-sink reliable transport in wireless sensor networks. In *ACM MobiHoc*, pages 177–188, 2003.

[15] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: a reliable transport protocol for wireless wide-area networks. In *ACM MobiCom*, pages 231–241, 1999.

[16] F. Stann and J. Heidemann. RMST: Reliable data transport in sensor networks. In *IEEE SNPA*, pages 102–112, 2003.

[17] K. Sundaresan, V. Anantharaman, H. Hsieh, and R. Sivakumar. ATP: A reliable transport protocol for ad-hoc networks. In *ACM MobiHoc*, pages 64–75, 2003.

[18] C. Wan, A. Campbell, and L. Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. In *ACM WSNA*, pages 1–11, 2002.

[19] C. Wan, S. Eisenman, and A. Campbell. CODA: Congestion detection and avoidance in sensor networks. In *ACM SenSys*, pages 266–279, 2003.

[20] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multi-hop routing in sensor networks. In *ACM SenSys*, pages 14–27, 2003.

[21] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *IEEE InfoCom*, pages 1567–1576, 2002.

[22] L. Zhang. Why TCP timers don't work well. In *ACM SIGCOMM*, pages 397–405, 1986.

[23] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *ACM SenSys*, pages 1–13, 2003.