BTP: a Block Transfer Protocol for Delay Tolerant Wireless Sensor Networks

Morten Tranberg Hansen
Department of Computer Science
Aarhus University
Email: mth@cs.au.dk

Edoardo Biagioni
Dept. of Information and Computer Sciences
University of Hawaii at Mānoa
Email: esb@hawaii.edu

***Abstract*—Wireless sensor networks that are energy-constrained must transmit and receive data as efficiently as possible. If the transmission is delay tolerant, transferring blocks of accumulated data can be more efficient than transferring each sensed measurement as soon as it is available. This paper proposes a Block Transfer Protocol (BTP) designed for efficient and reliable transmission in wireless sensor networks. BTP reduces the time it takes to reliably transfer a block of packets compared to conventional link layer protocols, by piggybacking in data packets information about the transfer, minimizing the number of acknowledgements needed for reliable transmission, and reducing the need for timeouts, which can substantially slow down communication when transmission is unreliable. In addition, BTP improves reliability by handling false positive acknowledgements[1] and by letting the receivers communicate to senders how many packets they are prepared to accept, providing a flow control mechanism to exert back-pressure on the senders.**

BTP has been evaluated on a real sensor node platform, as well as in simulation. BTP reduces the average time to transfer blocks of 60 packets, each 41 bytes long, by more than 20% over a perfect link with no packet loss, to 63% when 50% of the packets are lost. Furthermore, BTP ensures channel fairness and in our tests achieves a 100% delivery over multiple hops.

***Index Terms*—block transfer; hop-by-hop transport; reliability; energy efficiency; sensor network**

I. Introduction

Unattended battery-powered sensor networks must be as energy-efficient as possible. There are many real-world sensor network applications where this is a requirement, including long-term environmental monitoring, sensors embedded in buildings, bridges, and other structures, and any application where it is expensive to access the sensors, or where energy harvesting can only yield minimal amounts of power. In many cases, minimizing energy consumption is more important than delivering data promptly. Sensors may then be designed to sleep for long periods of time, and deliver the data when this can be done most efficiently.

Improvements in wireless communication and electronics have made low-power sensor nodes widely available. Many of today's sensor nodes consist of a low-power micro-processor, some sensors, and a wireless radio used to distribute the acquired data [3]. The power consumed by these nodes is typically dominated by the power needed to run the radio, both in receive and transmit mode—actually, some of today's popular radios consume slightly more power in receive than in transmit mode [1].

Since the effective energy cost is proportional to the amount of time the radio is on, power consumption is minimized by keeping to a minimum the time the radio is powered. This can be accomplished by sending less data, but for a given amount of data, the most significant factor is the communication protocols used for the data transmission. Such communication protocols include low-power link layer (MAC) protocols that provide access to the shared wireless communication medium [4], [20]. Most such protocols expend resources to acquire the channel, then only send a single packet. For a sensor network, this packet may be as small as 41 total bytes, including only 28 bytes of application data (default TinyOS payload size).

This paper improve the efficiency of link layer transmission by employing a standard link layer protocol to acquire a channel, then using that channel to transmit larger amounts of data. This is one example of the more general principle of *procrastination* to improve the lifetime of energy-limited sensor networks [6]. In a delay tolerant network, nodes can accumulate data and transmit it together so that the transmission is as efficient as possible. Instead of transferring one packet at a time between a pair of sensor nodes, we propose to transfer a block of multiple packets. Consequently, the channel only has to be reserved one time before the first packet is transfered, and acknowledgements can be concatenated in one packet after the last packet is transfered. This keeps the radios on for the shortest possible time, providing substantial energy savings.

The main contribution of this paper is the introduction and evaluation of BTP, a novel block transfer protocol designed for sensor networks that improves the transfer time and reliability for blocks of packets at the link layer while coping with the severe memory restrictions of sensor nodes. BTP works at the MAC layer and is described in Section III. To evaluate BTP, an implementation has been made in TinyOS 2.x running on the Tmote Sky platform [3] using the IEEE 802.15.4 compatible CC2420 radio [1]. The implementation is presented in Section IV, and experiments and simulations, described in Section V and VI, show that this implementation of BTP

[1] A *false positive acknowledgement* is an acknowledgement that is sent when the packet is received by the radio but not by the microcontroller. This accures when a packet is not fetched by the microcontroller from the radios buffer before a new packet arrives at the radio and overrides any older packets.

978-1-4244-8389-1/10/$26.00 ©2010 IEEE

improves the average transfer time for blocks containing more than 25 packets, under realistic channel conditions. Furthermore, we show that BTP improves the reliability by handling false positive acknowledgements and avoiding buffer overflow on the memory restricted sensor nodes.

II. Link Layer Acknowledgements

Techniques for reducing the energy cost of communication in sensor networks have been explored on all layers of the radio stack. Low power link layer protocols enable neighboring nodes to initiate communication [4], [20], network layer protocols enable energy efficient multi-hop data collection [9] and data dissemination [5], and transport layer protocols enable end-to-end reliability [19] and rate-control [18]. Despite the increasing complexity of such protocols, they all use simple link layer acknowledgements (LLA) to efficiently cope with the unreliable nature of the wireless channels when sending a packet from one sensor node to another.

In general the cost of LLA is considered small as widely used radios, such as the CC2420 radio [1], have hardware support for sending IEEE 802.15.4 acknowledgements. This can either be done automatically on reception in the radio or after the micro-controller has read and approved the packet header. We will refer to these as hardware initiated acknowledgements (HWACK) and software initiated acknowledgments (SWACK), respectively. Table I shows the cost of using LLA on the popular Tmote Sky platform [3] with the default TinyOS 2.x CC2420 radio stack sending 41 byte packets. By default, the radio stack implements a simple Carrier Sense Multiple Access (CSMA) scheme with a linear back-off (random value between $305\mu s$ and $2.441ms$) that uses a couple of clear channel assessments (CCAs) to detect activity on the channel before transmission. On a lossless link, the time overhead of using the fast HWACK with and without CCA is at least 26% and 30%, respectively. Whenever packets are lost, this overhead will severely increase as one lost packet (or acknowledgement), using the TinyOS CC2420 radio stack, with and without CCA has an overhead of 157% and 183%, respectively.

Table I shows that HWACKs are faster than SWACK, but one advantage of SWACK is a decrease in the sending of false positive acknowledgments. These are acknowledgements sent when a packet is received at the radio but eventually not delivered to the micro-controller. In general, false positive acknowledgements decrease the reliability of LLA, and can only be eliminated by having the micro-controller send the acknowledgement after the packet is in main memory. This incurs an extra delay in the transmission of the acknowledgement, and is normally undesirable. Our tests use the HWACK and SWACK supported by TinyOS, even though with such acknowledgements, LLA is sometimes unable to detect that its data is discarded rather than placed in the receiver's buffer. In contrast, with BTP the data is only considered delivered if the protocol can confirm that the packet has been placed in the final buffer as we will see in Section VI-E.

When considering a packet as one unit of reliable transfer, one cannot send acknowledgements faster than with HWACK.

TABLE I
Cost of LLA on the Tmote Sky running TinyOS 2.x, with and without CCA before transmission. Since there is no activity on the channel, these overheads are independent of the back-off strategy used. Packet size is 41 bytes.

Approach	without CCA	with CCA
No ACK	$4319\mu s$	$4999\mu s$
Hardware initiated ACK	$5617\mu s$	$6305\mu s$
Software initiated ACK	$6537\mu s$	$7221\mu s$
Lost ACK	$12237\mu s$	$12837\mu s$

This overhead is inevitable when packets are transferred one at a time. In the next section we propose to avoid much of this overhead with a protocol for efficient hop-by-hop transfer of blocks of packets in sensor networks.

III. Block Transfer Protocol

Since the introduction of the TCP flow control window [16] it has been clear that sending a block of packets before waiting for an acknowledgement gives better performance than the stop-and-wait protocol, where each packet must be acknowledged before the next one can be sent. Although TCP is used worldwide and has been implemented in sensor networks [10], it is too heavyweight for many low-power sensor network deployments [2], even in specialized hop-by-hop variants [14].

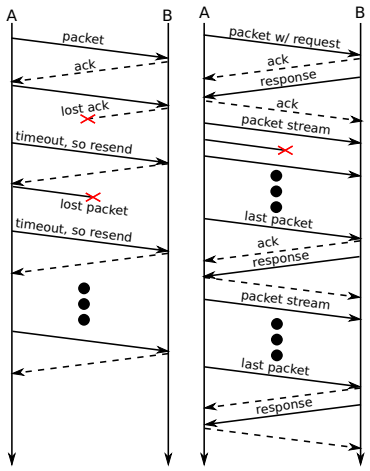
To the best of our knowledge, previous sensor network research on real sensor node platforms has only explored end-to-end reliable transport protocols, rather than link layer protocols, as alternatives to the use of the per packet LLA. Figure 1(a) shows how a block is transfered from a node, A, to another node, B, using LLA. In this protocol, each packet is sent repeatedly until acknowledged. Hence every time a packet is sent the sender must wait, with the radio on, for an acknowledgement, or for a timeout indicating that the packet or acknowledgement was lost. Using a per packet link layer acknowledgement provides low per packet delay, but creates the need for an explicit congestion back-off strategy to avoid buffer overflows and improve reliability [9].

However, many of today's sensor networks are delay tolerant, and energy efficiency and reliability are the primary concerns. In this section we present BTP, a block transfer protocol for delay tolerant sensor networks which decreases the per hop block transfer time while insuring per hop reliability. BTP includes an explicit hand-shake to grant permission to send the additional packets in a block.

A block in a sensor network may only consist of a few hundred bytes of data, for example 10 packets, each with a 28-byte payload. This is a good match for the main memories of sensor nodes, is about the size of the forwarding queue of many of today's popular sensor network routing protocols [8], and is the amount that can be sent between sensor nodes in a route over IPv6 based sensor networks [15].

A. Protocol Overview

In the Block Transfer Protocol (BTP), the sender and receiver agree on how many packets to transfer in a block,



(a) LLA (b) BTP

Fig. 1. Transfer of one block from node A to node B using traditional link layer acknowledgements (LLA) and the block transfer protocol (BTP).

then send the majority of the packets without explicit acknowledgements.

Figure 1(b) shows how a block is transfered from a node, A, to another node, B, using BTP. BTP uses LLAs, shown as dashed lines, as a synchronization mechanism. Since an LLA only carries information about the last packet received, BTP also uses data acknowledgements, marked "response" in Figure 1(b). Responses convey information about the state of the receiver, and specifically, which packets have not yet been received and how many more packets the receiver is able to buffer.

Node A initiates the transfer by sending a request to send n packets to node B. The request is embedded in the first data packet. Node B responds to this request with a response containing a grant to send m out of the n packets requested ($m \leq n$). To make sure that the request and response are received by the opposite side, they are sent reliably with LLA.

Each data packet in BTP contains a request field and a sequence number. The request field is embedded in all data packets, though only the request carried by the first data packet is actually used, and the sequence number is used to identify received packets within a block. Responses in BTP consist of a grant field granting a number of packets to send, a bit vector identifying which of the previously granted packets have been successfully received, and a start sequence number identifying the block of packets. The grant field is only valid in the first response and the bit vector is only used in subsequent responses.

After receiving the response with a grant to send $m > 0$ packets, node A sends to B a stream of the remaining $m - 1$ granted packets. Once node B receives the last packet in the stream, it responds by sending a compound acknowledgement with a bit vector indicating which packets were received. To make sure that B receives the last packet, it is sent reliably using LLA. Assuming LLA is reliable, this avoids the use of an expensive timeout mechanism at B to handle the case where it stays awake to listen for a lost final packet.

If any packets have been lost due to an unreliable channel, node A repeats the cycle by re-sending the lost packets in a new packet stream as seen in Figure 1(b).

B. Link Layer Acknowledgements

The use of LLA in BTP is not as costly as depicted in Section II, but can cause interference between nodes A and B, and result in false positive acknowledgements.

BTP uses LLA to reliably send the initial request, all responses, and the last packet in each packet stream. These LLAs are "free" in the sense that the sender will be awake anyway waiting for the response from the receiver, or for a timeout, and in the sense that the radio at the receiving node will be able to send the link layer acknowledgement while the microcontroller prepares the response packet. However, with the use of LLA in BTP there is a risk for the sender and receiver to interfere. After sending the initial data packet, the sender will expect an acknowledgement. If this acknowledgement is lost, and the receiver tries to send the corresponding response at the same time as the sender retransmits the data there is a risk for interference between the two packets. Likewise, this could happen for all but the last response, and for the last packet in each packet sequence. To minimize such self-interference, BTP sends all packets using LLA with CCA enabled. Furthermore, as explained in Section II, with the use of LLA there is a risk of false positive acknowledgements. As BTP relies on LLA as a synchronization mechanism between the sender and the receiver these cannot be ignored and must be handled by a timeout. BTP does this at the sending side.

C. Variable Block Size

The explicit hand-shake in the beginning of the block transfer protocol enables the use of variable block sizes which is an important feature in resource restricted sensor networks. A large block size improves the performance of BTP, but in practice this block size is limited by the storage capabilities of the sensor nodes and the fact that a sensor node might receive packets from multiple senders. Variable block sizes let the receiver dynamically determine what block size to accept from each sender, based on its remaining available storage.

D. Sharing the Channel

In many delay tolerant sensor networks, transmissions are very infrequent and hence sharing of the wireless channels is not an issue. In more data intensive networks, multiple senders might interfere with each other and hence a collision avoidance mechanism is needed. As described in Section II, LLA supports a simple per packet CCA, an interval prior to transmission during which the radio listens for other transmissions. If such transmissions are detected, the current transmission is delayed. CCA decreases the risk of collision, but increases the minimum and average latency of transmission. In the following we will differentiate between two versions BTP: one "without" CCA, where CCA is only used for the reliably transmitted packets, and one "with" CCA, where all packets are transmitted with CCA.

IV. Implementation

We have implemented both BTP and a version of LLA in TinyOS 2.x, as a layer on top of the TinyOS active message layer. In accordance with most data collection protocols for sensor networks, our implementation enables a sensor node to send one block of packets at a time, while being able to receive blocks from multiple senders. Thus, the amount of state needed at the sending side is limited and dominated by the length of the packet buffer to send, which is allocated by the application. The receiving side needs to keep a state for all incoming blocks, store the packets in a common packet buffer, and deliver them to the upper layers whenever a complete block has been transfered.

With the application allocating the packets to send and the states for all incoming blocks stored in a neighbor table with limited entries, our BTP implementation has a RAM overhead of 176 bytes plus the size of the receive buffer, and a ROM (program memory) overhead of 4448 bytes. As the maximum amount of RAM available on the Tmote Sky is 10kB, it does not seem viable to have a receive buffer larger than 80 packets (consisting of 41 bytes plus meta-data per packet), which will take up about half the RAM.

In addition to the RAM and ROM overheads, the BTP implementation adds to all data packets a 1 byte request and a 1 byte sequence number, as described in Section III-A. The response consists of a 1 byte grant, a received bit vector with 64 bits, one bit for each possible packet in a block, and a 1 byte start sequence number.

BTP uses LLA as a synchronization mechanism, and this makes it vulnerable to false positive acknowledgements. These are handled by a timeout at the sending side in BTP. In our implementation this timeout is over-estimated to $10ms$ times the maximum number of retransmissions. Furthermore, the variable block sizes of BTP lets the receiver dynamically determine what block size to accept from multiple senders. Our BTP implementation currently allocated the remaining available storage at a receiver to multiple senders on a simple first-come-first-served basis.

To enable sharing of the channel, both BTP and LLA can be run either with and without CCA. Likewise, both BTP and LLA can be run with either SWACKs or HWACKs. HWACK is faster because the micro-processor does not have to initiate the generation of the HWACK, but the probability of a false positive acknowledgement is higher. The choice of using SWACK or HWACK, or CCA or not, is application dependent, so in the following experimental evaluation we will consider all four versions of BTP and LLA.

V. Simulation

To verify the performance benefits of BTP compared to LLA, and to verify our BTP implementation, we did a Matlab simulation of the two protocols, where we 1) use a simplified link model with fixed packet reception ratio (PRR) and acknowledgement reception ratio (ARR) set to be the same, 2) let the cost of all transmissions be the same even though responses tend to be shorter and therefore cheaper than data packets, 3) ignore potential parallelism in BTP where an acknowledgement can be sent at the same time as a packet is prepared or transfered to the radio, 4) ignore any processing time, and 5) let the number of per packet retransmissions be set to infinite when transferring a packet reliably with link layer acknowledgements.

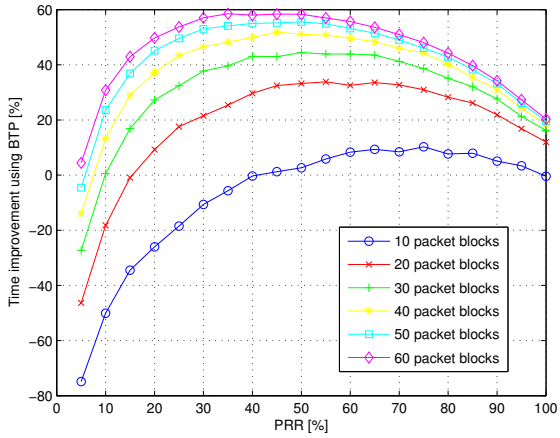
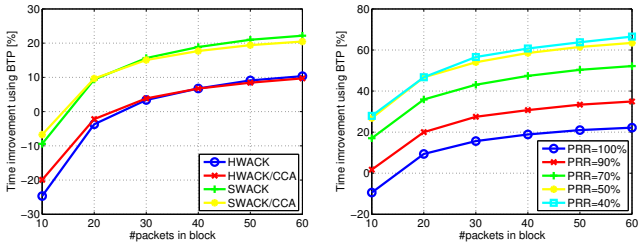


Fig. 2. Matlab simulation showing the time improvement of using BTP compared to LLA to transfer blocks of different sizes over links with different Packet Reception Ratio (PRR) using HWACKs and CCA.

Figure 2 shows the time improvements of using BTP compared to LLA to transfer blocks of different sizes over links with different PRR in the simulation. The figure is based on packet transmission costs from Table I when using HWACK with CCA as the time improvement of BTP will be higher in all other cases. The figure shows that over a perfect link, with no packet loss, BTP is favorable whenever 20 or more packets are transferred in a block, and that the benefits of using BTP over a perfect link increases with the block size. When the link quality decreases, the time improvement of using BTP increases until a certain point, depending on the block size, where the cost of the responses in BTP is more than the cost of acknowledgements in LLA. After this point the time improvement of BTP decreases and equals zero, for the first time, with a 10 packet block and PRR and ARR being 40%. We argue that any reasonable routing protocol for sensor networks would never route packets over a link with bi-directional link quality of 16% or less, and hence we will not consider these cases in the following evaluation.

VI. Experimental Evaluation

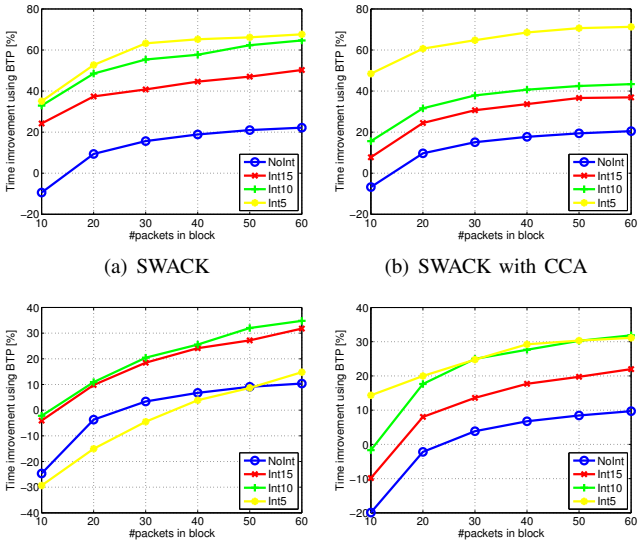
In the following evaluation we use our TinyOS implementation of BTP to verify the improvements of using BTP on a real sensor platform. We deploy BTP and the LLA equivalent implementation on a number of Tmote Sky [3] sensor nodes and let the senders send blocks of packets to the corresponding receivers as fast as they can. To make sure that debug information sent over the serial connection does not interfere with the protocols it is kept to a minimum and only sent between block transfers. Unless otherwise specified, all times are averages over a 2 minute experiment. All experiments are done with hardware address recognition enabled, full transmission power



(a) Time improvement over a perfect link.

(b) Time improvement with programmed packet loss using SWACK without CCA.

Fig. 3. Time improvement of BTP compared to LLA running on a pair of Tmote Skys over a perfect link with no packet loss and where packet loss are programmed at the low levels of the radio stack. The lower time improvements of using BTP without CCA for low block sizes is due to the fact that BTP always uses CCA for reliable transfers.



(a) SWACK

(b) SWACK with CCA

(c) HWACK

(d) HWACK with CCA

Fig. 4. Time improvement of BTP under real interference from a pair of neighboring nodes sending a packet periodically. The interference level is increased by decreasing the interference interval from never (NoInt) to $15ms$ (Int15), $10ms$ (Int10), and $5ms$ (Int5).

so that all sensor nodes are in range of each other, a packet size of 41 bytes (default TinyOS CC2420 packet size), and a receive packet buffer size of 60 packets. The maximum number of retransmissions when transferring a packet reliably using LLA is set to 30.

A. Perfect Link

Our simulation in the last section showed that BTP is favorable to LLA whenever more than 20 packets are transferred in a block. In this subsection we verify this on real sensor node hardware by deploying our BTP and LLA implementations on two sensor nodes next to each other so that no packets will be lost. Figure 3(a) shows the time improvement of BTP when transferring blocks of different sizes over a perfect link. Using SWACK (HWACK), BTP becomes beneficial when more than 14 (24) packets are transfered in a block with CCA, and 15 (25) packets without CCA. The time improvement of using BTP with SWACK (HWACK) for a 60 packet block is 20.5% (9.5%) and 22% (10.5%) with and without CCA, respectively. Considering the extra time it takes to wait for an acknowledgement in Section II, this is a very good result for BTP, since over a lossless link LLA will not experience any expensive timeouts.

B. Programmed Packet Loss

The previous section showed that the time improvements of using BTP over a perfect link increases with the block size and the simulation showed that this improvement will be even greater with a decrease in channel quality. In this section we show that the improvement also increases with a decrease in channel quality on real sensor node hardware. Due to the uncertainty of the changing wireless channel, it is not possible to predict packet loss in such a setup by adjusting the distance between the nodes. So to evaluate BTP under different channel conditions we make use of programmed packet loss at the low level of the radio stack by randomly discarding packets, and acknowledgements, according to some fixed PRR set at compile time. This is only possible using SWACK.

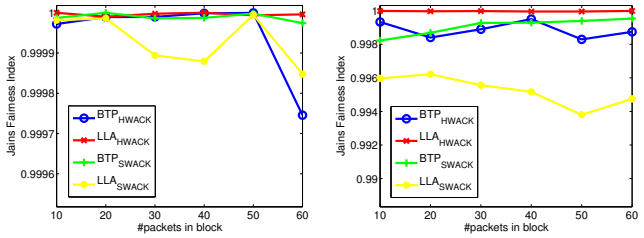
Figure 3(b) shows the time improvement of using BTP to transfer blocks of different sizes between a pair of sensor nodes with our programmed packet loss. Both BTP and LLA uses SWACK and no CCA. The figure shows that when the PRR decreases to 90% it is beneficial to use BTP with block sizes down to 10 packets and that the time improvement of using BTP with 60 packets increases from 22% with $PRR = 100\%$ to 35%, 52%, and 63% for $PRR = 90\%$, $PRR = 70\%$, and $PRR = 50\%$, respectively.

C. Packet Interference

The experiments with programmed packets loss show that BTP improves the transfer time of a block of packets on a real sensor node platform under different channel conditions. In this section we evaluate how BTP performs under actual interference from another pair of nodes, which not only creates natural packet losses but also stresses the radio as it automatically receives the headers of these other packets.

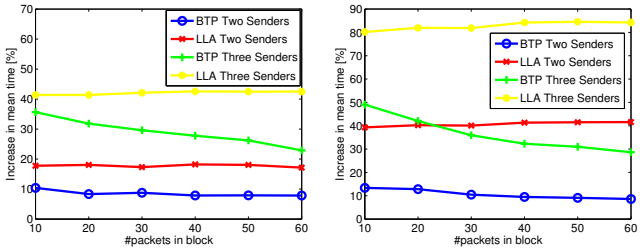
We created natural interference by placing four sensor nodes in a diamond, where two of the opposing nodes use BTP and LLA to transfer blocks of packets, and the other two opposing nodes, the interferers, send a packet at a regular interval without using either acknowledgements or CCA. To avoid synchronization between the block transfer and the interferers, we use a 50% random offset on the interferer interval.

Figure 4 shows the time improvement of using the different versions of BTP when we increase the level of interference by decreasing the interferer interval. Figure 4(a) shows how the time improvement of BTP increases with the level of interference when using SWACKs. The figure shows that the maximum improvement of using BTP is under 70% with a block size of 60 packets which correspond to the findings in the last section with programmed packet loss. Furthermore, the improvement of BTP with a block size of 10 has reached its maximum with an interference interval of $5ms$ (the time



(a) Two senders (b) Three senders

Fig. 5. Jain's Fairness Index for LLA and BTP taken on the number of blocks sent during a 4 minute experiment with two and three multiple independent senders.



(a) HWACK (b) SWACK

Fig. 6. The increase in mean transfer time for BTP and LLA when sharing the channel with one or two other senders.

improvement does not increase much compared to an interference interval of $10ms$). Figure 4(b) shows how the use of CCA limits the effect of the interferer and thereby reduces the improvement of BTP for each interference interval.

Figure 4(c) shows how the time improvement of BTP using HWACKs is less than when using SWACKs. This is caused by false positive acknowledgements and high ARR—the latter being the most significant in these experiments. As explained in Section II, false positive acknowledgements are more likely to happen with HWACK than SWACK, and as BTP relies on LLA as a synchronization mechanism, it has no other alternative than to handle these cases with an expensive timeout. Furthermore, the probability of the channel being free right after successful reception is high which makes the ARR of the quick HWACKs larger than the ARR of the slower SWACKs and BTP responses. The high ARR does not make BTP perform worse. Instead, it allows LLA using HWACK to apparently perform better, as LLA is less influenced by interference—hence the time improvement of BTP is decreased. However, this apparent good performance of LLA is misleading, since some of the data that is acknowledged is not delivered to the application. This is particularly obvious in the high-interference regime of Int5. In contrast, BTP actually delivers all its data, and to do so has to work proportionally harder in the presence of false positive acknowledgements. We discuss this more in Section VI-E and VI-F, where we argue that BTP is still preferable to LLA in these situations. As with SWACKs, Figure 4(d) shows how the use of CCA with HWACKs limits the effect of the interferer and thereby reduces the improvement of BTP under the different interference intervals.

D. Fairness

The design of BTP, where multiple packets are sent in a stream without intermediate responses from the receiver, could lead one to assume that the use of BTP compromises MAC layer channel fairness, causing one sender to back-off for a longer period of time due to constantly detecting activity on the channel. But even though BTP sends packets as fast as it can there will still be quiet times, caused by the time it takes to transfer a packet from the micro-controller to the radio and other processing, which let other nodes acquire and utilize the channel. In this section we evaluate the fairness of BTP with CCA and compare it to LLA with CCA. We measure the level of fairness in two steps: first we show that multiple senders are able to acquire a fair share of the channel, and then we show how the sharing of the channel affect the mean time to transfer blocks of different sizes for the two protocols.

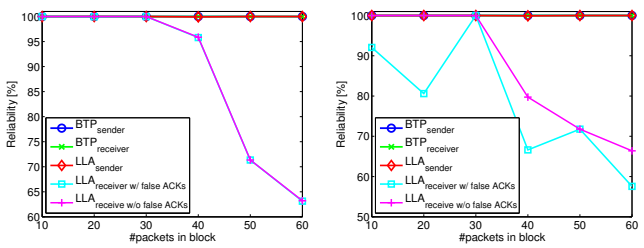
We placed multiple independent sender nodes in a line, with a receiver across from each sender, making sure that the distance each sender and its receiver was the same. Then we let the senders send blocks of packets as fast as they could for 4 minutes, and counted the number of blocks they were able to send and the respective transfer times of these blocks. Due to hardware limitations we were only able to do this experiment with 1, 2 and 3 senders.

We use Jain's Fairness Index [11] (JFI) [2] with the number of packets each node was able to send during the experiment as input. The transmission is fair if each node was able to acquire its share of the channel. Figure 5 shows the fairness of BTP and LLA using HWACK and SWACK. Figure 5(a) shows that with only two senders, the fairness index is close to 1 for both protocols with HWACK and SWACK, which means that both senders are able to send the same amount of blocks throughout the experiment. The only outlier is LLA with SWACK which has a hint of unfairness. Figure 5(b) show that with 3 senders, LLA using SWACK is still an outlier and divides the channel less equally among the senders than BTP with HWACK and SWACK.

To show the effect of sharing the channel for the two protocols, we compare their mean transfer times when sharing the channel to the mean transfer time when not sharing. Figure 6 shows the increase in mean transfer time when sharing the channel. Both with SWACK, Figure 6(b), and HWACK, Figure 6(a), we see that the overhead of transferring a block of packets does not increase more than with LLA. Thus, transmissions of blocks from multiple senders, using BTP, can overlap at least as well as they can using LLA.

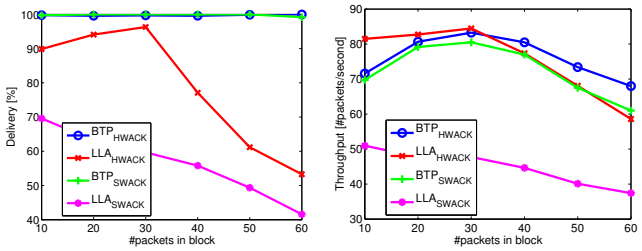
To summarize, BTP transmission is no less fair than LLA transmission, and the performance is more robust in the presence of interfering traffic. The fairness of BTP is very important when considering that one node with a small block of packets should not have to wait for another node, with a

[2] $JFI = \frac{\left(\sum x_i\right)^2}{n\sum x_i^2}$, where n is the number of elements, $1/n$ the worst fairness, and 1 the best fairness.



(a) Reliability from node 1 (b) Reliability from node 2

Fig. 7. Reliability of BTP and LLA. The reliability seen by the sender and the receiver are shown for two different sensor nodes sending packets to the same receiver. The receiving side of LLA is shown with and without the detected false positive acknowledgements included in the graph.



(a) Delivery (b) Throughput

Fig. 8. Delivery and throughput of BTP and LLA with HWACK and SWACK over two hops.

large block of packets, to finish its transmission.

E. Single-hop Reliability

BTP improves reliability by handling false positive acknowledgements and access to the limited buffer space on the receivers. In this section we explore these functionalities and compare BTP to the similar LLA implementation based on reliability.

We evaluate the reliability by placing three sensor nodes on a row, with the middle node being a receiver and the two other sensor nodes simultaneously sending data to this node. As explained in Section II false positive acknowledgments can happen with both HWACKs and SWACKs, but are more likely to happen with HWACK, and hence we will focus on LLA and BTP using HWACK in this section. All experiments are done with CCA which is a natural choice in a multi-sender environment.

Figure 7 shows the reliability of BTP and LLA in this one-hop setup where the senders send blocks of packets to the receiver as fast as they can. Figure 7(a) shows how BTP achieves 100% reliability between sender one and the receiver with all packet sizes, and how the actual reliability at the receiver drops for LLA with packet sizes larger than 30 due to limited buffer space at the receiver. The same trend is shown in Figure 7(b) which shows the reliability between sender two and the receiver. In addition to the packets lost due to limited buffer space, we also see that some packets are lost due to false positive acknowledgements between sender 2 and the receiver. This also goes unnoticed by the sender when using LLA.

F. Multi-hop Delivery

The last section showed how BTP improves reliability over a single hop compared to LLA by handling false positive acknowledgements and limited buffer space at the receiver. In this section we take this a step further, and look at how BTP improves delivery, and possibly throughput, over multiple hops by automatically providing back-pressure on the senders; leaving time for the receiver to forward its packets.

We do this by placing three sensor nodes on a line, with the middle node being a forwarder from one of the outer ones, the sender, to the other outer one, the receiver. The forwarder forwards incoming packets from the receive buffer and only frees the packets from the receive buffer once the entire block is forwarded. To simplify things, we only let the sender generate packets.

We have not implemented any back-off strategy for BTP, so whenever the receiver rejects a block, BTP tries to immediately resend it instead of giving the receiver time to clear out its own buffer being retransmitting. We leave back-off strategies for future work. Figure 8 shows the delivery rate and the throughput of LLA and BTP over two hops.

Comparing BTP and LLA using SWACK, Figure 8(a) shows that BTP achieves a 100% delivery rate over all block sizes. Instead, the delivery rate of LLA decreases from only 70% with a block size of 10 packets to about 40% with a block size of 60 packets—thus, with any block size tested, the forwarder using LLA is not able to keep up to speed with the sender.

Similarly the throughput over two hops for LLA using SWACK, shown in Figure 8(b), decreases with the block size. Using SWACK, BTP's throughput over two hops is always better than LLA's, and reaches its maximum with a block size of 30 packets. After this, it decreases with the block size due to an increasing number BTP rejects at the sender which interfere with communication without improving throughput. BTP's variable block size keeps the throughput high for block sizes larger than 30 packets: instead of the sender waiting to transfer another full size block, it just sends as many packets as the receiver is able to store.

Figure 8(a) also show that BTP achieves a 100% delivery rate over all block sizes using HWACK. Again, LLA is not able to do this, but it does have a higher delivery rate with HWACK than SWACK. This is due to the forwarder being better at keeping up with the sender. Looking at the throughput for LLA with HWACK in Figure 8(b), we see that the greedy transmission of LLA, which does not take into account the limited buffer space at the receiver, pays off with a slightly better throughput than BTP for small packet sizes. Although this throughput is good, it comes with a corresponding decrease in delivery rate.

VII. Related Work

The BTP protocol effectively embeds an RTS/CTS exchange [12] within the first data packet and the first response. Any node overhearing the first data packet will avoid transmissions for the time needed to send the entire requested

block, whereas nodes overhearing the response keep quiet for the time needed to transmit the block size granted by the receiver. Subsequent transmissions and responses update this information. One benefit of BTP over RTS/CTS is that the request and reply are carried as part of the normal data transfer mechanism, lessening the overhead of RTS/CTS.

Reliable per-hop block transfer for 802.11 networks has been explored in the Hop protocol, designed to improve the throughput on multi-hop 802.11 networks [13]. Besides the difference in application area and underlying link layer technology, BTP and Hop differ in a number of ways: 1) Hop does not specify how many packets will be sent in a block, whereas, to accommodate possible scarcity of buffer space and also provide back-pressure, BTP specifies the number of packets in the request and in the grant, 2) because of this, and because the last data packet is sent reliably, BTP has no need for the BSYN packet from the Hop protocol, 3) Hop handles hidden terminals with acknowledgment withholding, as opposed to the BTP scheme which mostly handles hidden terminals as RTS/CTS does, and 4) Hop does not detect a broken link until a whole block has been transfered, whereas BTP detects failed links as soon as the initial data packet is not acknowledged.

Hop-by-hop reliability, provided by BTP, is not sufficient to insure end-to-end reliability [17]. TCP [16] provides end-to-end reliability for many applications, but is known to result in very inefficient transmission on most multi-hop wireless networks [7]. In addition, an end-to-end mechanism requires continuous end-to-end connectivity [13], which may be unacceptable in delay tolerant networks where the goal of end-to-end reliability for any given transmission may be less important than increasing the overall lifetime of the network by transmitting more efficiently.

VIII. Conclusion and Future Work

Given the low efficiency and reliability of conventional link layer protocols for data transfer in delay tolerant wireless sensor networks, we have designed a block transfer protocol, BTP, to transmit blocks of data more efficiently than by sending individual packets. Although block transfers similar to that employed for BTP have been used before, applying this to wireless sensor networks requires careful consideration of what must be optimized. For wireless sensor networks, minimizing the total transfer time allows radios to be shut down sooner, saving the most energy. Rather than minimizing the bit count, where possible the design of BTP schedules transmission for times when the medium is available anyway, and optimizes the time to finish transmitting the data.

In our evaluation we have found that BTP provides substantial speedups. For block sizes of 60 packets, on links with no interference to links with 50% packet loss, BTP outperforms traditional link layer acknowledgements by factors ranging from 20% to 63%. Furthermore, in our experiments BTP showed 100% reliability, thanks to correct handling of false positive acknowledgements and to not sending more packets than a receiver is able to buffer.

Future work includes exploring exactly what MAC protocols are amenable to BTP-adaption, ways of preventing false positive acknowledgements, alternatives to the first-come-first-served-sender strategy for managing buffer space at the receiving node, and exploring back-off strategies at the sender that is better than the currently immediately retransmit strategy.

References

[1] Cc2420: 2.4 ghz ieee 802.15.4/zigbee rf transceiver. http://www.ti.com/lit/gpn/cc2420.

[2] The sensobyg project. http://sensobyg.dk/english.

[3] Tmote sky low power wireless sensor module. http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf.

[4] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320, New York, NY, USA, 2006. ACM.

[5] T. Dang, N. Bulusu, W. C. Feng, and S. Park. Dhv: A code consistency maintenance protocol for multi-hop wireless sensor networks. In *EWSN '09: Proceedings of the 6th European Conference on Wireless Sensor Networks*, pages 327–342, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] P. Dutta, D. Culler, and S. Shenker. Procrastination might lead to a longer and more useful life. In *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets-VI)*, November 2007.

[7] Z. Fu, X. Meng, and S. Lu. How bad tcp can perform in mobile ad hoc networks. In *ISCC '02: Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*, pages 298+, Washington, DC, USA, 2002. IEEE Computer Society.

[8] O. Gnawali, R. Fonseca, K. Jamieson, and P. Levis. Ctp: Robust and efficient collection through control and data plane integration. Technical report, Stanford, 2008.

[9] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys '09: Proceedings of the 7th ACM conference on Embedded network sensor systems*, New York, NY, USA, 2009. ACM.

[10] J. W. Hui and D. E. Culler. Ip is dead, long live ip for wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 15–28, New York, NY, USA, 2008. ACM.

[11] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, Digital Equipment Corporation, September 1984.

[12] P. Karn. MACA – a new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, September 1990.

[13] M. Li, D. Agrawal, D. Ganesan, and A. Venkataramani. Block-switched networks: a new paradigm for wireless transport. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 423–436, Berkeley, CA, USA, 2009. USENIX Association.

[14] Y.-N. Lien. Hop-by-hop tcp for sensor networks. *International Journal of Computer Networks & Communications*, 1(1), April 2009.

[15] G. Mulligan. The 6lowpan architecture. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, pages 78–82, New York, NY, USA, 2007. ACM.

[16] J. Postel. Rfc 793: Transmission control protocol. Technical report, September 1981.

[17] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984.

[18] A. Sridharan and B. Krishnamachari. Explicit and precise rate control for wireless sensor networks. In *SenSys '09: Proceedings of the 7th ACM conference on Embedded network sensor systems*, New York, NY, USA, 2009. ACM.

[19] F. Stann and J. Heidemann. Rmst: reliable data transport in sensor networks. pages 102–112, Anchorage, Alaska, USA, April 2003. IEEE.

[20] Y. Sun, O. Gurewitz, and D. B. Johnson. Ri-mac: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 1–14, New York, NY, USA, 2008. ACM.