Application-Based Collision Avoidance in Wireless Sensor Networks

Thanos Stathopoulos[†] Rahul Kapur[†] Deborah Estrin[†]
John Heidemann[‡] Lixia Zhang[†]

[†] UCLA, Department of Computer Science
[‡]USC, Information Sciences Institute
[†]{thanos,rkapur,destrin,lixia}@cs.ucla.edu, [‡]johnh@isi.edu

Abstract

Wireless sensor networks are characterized by collections of small, low-power nodes that collect information about the physical world. Concurrent transmissions caused by the well-known hidden terminal problem result in collisions and packet corruption. Since corrupted packets must be retransmitted, collisions add an additional burden to the already energy constrained system. In this paper, we present an application-based *approach to collision avoidance. We propose two specific algorithms; the first one follows TCP's congestion avoidance algorithm and adjusts the transmission rate when a collision occurs, while the second one shifts packet transmission times to minimize collisions. We evaluated both algorithms through simulations and our results show that our approach can reduce the number of collision-induced retransmissions by a factor of 8 and the energy consumption by up to 50%.*

1. Introduction

Wireless sensor networks consist of collections of small, low-powered nodes that interface or interact with the physical environment. Once deployed sensor networks are expected to operate for extended periods of time without any human intervention. Substantial research effort has been directed toward increasing network lifetime by reducing radio communication, the largest source of energy drain.

In general, collisions on wireless networks can be a major source of increased latency and packet retransmission. When collisions occur on energy constrained wireless networks, extra latency and retransmissions equate to excess energy consumption. On the mote platform, energy is at a premium, therefore collision avoidance can increase the overall lifetime of the network.

Current sensor network solutions attempt to solve the collision problem in the MAC layer, by either using TDMA [11] or an RTS/CTS mechanism [13]. However, MAC-layer solutions cannot easily exploit application-specific knowledge—nor do they attempt to. Given a particular class of applications, a collision avoidance mechanism that takes advantage of domain-specific knowledge can supplement and augment the performance of the MAC layer.

In developing a collision avoidance scheme it is beneficial to partition wireless communication into two types: single-packet and multi-packet. In single-packet communication, previous packet arrivals provide no information about future ones; in essence, arrivals are random. In multi-packet communication the existence of a 'next' packet is inferred by definition. There is now a correlation between previous arrivals and future ones. It is in this communication space that a general heuristic can be applied for collision avoidance. In this paper we explore various approaches to application-layer collision avoidance. Our algorithms are specifically targeted for applications with multi-packet communication. We have used Multihop Over-the-Air Programming (MOAP) [10], a code distribution mechanism specifically targeted for Mica2 nodes [6], as a sample application on which to develop our algorithms.

Our design focuses on two methods. In the *informed TCP-like* approach, the receiver informs the source of a collision-induced loss. The source then reacts by multiplicatively decreasing its transmission rate. If no collisions are detected, then the source can increase its rate, thereby achieving lower latency over a static approach. In the *phase offset* approach, the source transmits at a constant rate. The receiver records packet arrivals and attempts to calculate the largest silent period

between two consecutive packet transmissions. When a collision occurs, the receiver passes that information to the source which then changes its transmission phase to match the largest silent period.

We compare and evaluate our algorithms using simulations. We show that, compared to a base case, in which sources transmit as fast as possible, our methods can decrease the number of collision-induced lost packets a factor of 8 and reduce additional energy consumption on the source by 50%.

2. Problem Description

In the MOAP dissemination protocol, code is first injected into the network from a specific node, called the *original source*, and then disseminated on a *neighborhood-by-neighborhood* basis. A neighborhood is a collection of nodes sharing a common broadcast domain. A publish-subscribe mechanism is used to initiate the code transfer. Once a receiver acquires the entire code image, it attempts to become a source by sending publish messages. Each node interested in receiving the new code will reply with a subscribe message. A node will not disseminate the code any further if it has no subscribers. This ripple-like dissemination guarantees that, in the absence of a permanent network partition, the new code will reach all the nodes.

Even though MOAP tries to minimize source overlap by requiring sources to have at least one subscriber in order to be active, some of the neighborhoods can still partially overlap in both space and time. The receivers located in the overlap area will receive data from their primary source while overhearing transmissions from other background sources. The primary and background sources for these receivers may not be in transmission range of each other, however they create packet collisions at the receivers. Consequently, these corrupted data packets lead to retransmissions.

In resource-constrained wireless sensor networks, packet transmissions are among the most energy-demanding operations (approximately 20uAh per packet [9]). Thus one effective means to reduce energy consumption is to reduce collisions. Collisions can be reduced by slowing down packet transmissions; however doing so will also extend the code dissemination completion time. Radio listening time, while not as energy consuming as packet transmissions, still incurs a non-negligible cost, except in special cases where an energy-aware MAC like S-MAC [13] is used. Generally speaking, however, reducing the completion time of the code transfer operation can also lead to energy savings. Furthermore, the network is for all practical purposes considered inoperable while the code update is in progress, since its primary function—collecting data from the physical world—cannot be accomplished. This adds another reason for reducing the completion time.

Traditionally, collision avoidance and recovery falls into the MAC protocols. TDMA-based MAC protocols avoid collisions imposing a deterministic transmission schedule; however they require communications among all the sources for coordination, which can induce both higher latency and energy consumption. Contention-based protocols which are currently used in sensor networks [5] [13] sense the channel for ongoing communication before transmitting. This can avoid certain collisions near the sender, but cannot avoid collisions at a receiver which can hear sources that are not in range of each other, i.e. the hidden terminal effect. An existing solution to the hidden terminal problem is the use of RTS/CTS messages to reserve the channel [13] [1]. The RTS/CTS approach assumes symmetric channels and is only applicable for point to point communications. Broadcast communication and the presence of asymmetric links—which are common in wireless sensor networks [12] [2]—renders this approach infeasible. Since MOAP uses broadcasts in order to reduce the number of transmissions, the RTS/CTS mechanism is not directly applicable for collision avoidance.

By exploiting domain specific knowledge that is only available to the application (and not to lower layers, like the MAC layer), we can design a collision avoidance scheme that can potentially be quite more efficient that a more generic, lower-layer mechanism. This design approach, although discouraged in larger networks like the Internet, is becoming the norm in wireless sensor networks. In the presence of resource constraints and unreliable links, it is considered acceptable to trade off generality for efficiency.

In the rest of this paper we develop application-specific solutions to reduce both energy consumption and code dissemination completion time through collision avoidance.

3. Collision Avoidance

A simple and straightforward approach to reduce collisions is to use a probabilistic scheme. Time is partitioned into large epochs, in which each source randomly chooses a time to send. Since the epoch is sufficiently large compared to the packet send time, the probability of any two sources choosing overlapping send times is small. MOAP currently uses such a scheme.

By definition, each source can send only one packet per epoch. Therefore, latency is directly proportional

to the size of the epoch. A sufficiently large epoch for a small network is much different than for a large network. In essence this scheme is not *adaptive* to the network environment.

An adaptive solution should try to decrease latency when possible but also react to collisions as they occur. Ideally, it should lower the probability of future collisions without significantly increasing latency or the number of retransmissions.

Adaptive collision schemes can be categorized into two types: source-based and receiver-based. In the source-based approach, there is no explicit feedback sent to the source from the receiver, except for that which is defined as a part of the protocol. An example of a source-based approach is Van Jacobson's TCP congestion avoidance [7]. TCP congestion avoidance uses the TCP protocol's duplicate ACKS and timeouts to infer network congestion and react to it.

The receiver-based approach allows the receiver to send feedback to the sender for the explicit purpose of reacting to collisions. As the receiver discovers pertinent information for collision avoidance, it transmits that information to the source. XCP [8], in which the network explicitly sends congestion information back to the source, is an example of a receiver-based approach.

3.1. Source-Based Collision Avoidance

Congestion avoidance on a packet switching network is similar to collision avoidance in a wireless network. In both cases the source throttles its data rate in the presence of an anomaly (congestion or collision). In collision avoidance, throttling the data rate changes the arrival period of the data packets, so any reoccurring collisions due to overlapping periods will be avoided. Also, in slowing down the source, heavy background traffic at the receiver is allowed to dissipate. This latter result implicitly assumes that all source traffic has a finite length and that after some time the background sources will eventually stop.

In the case of TCP congestion control, the design implicitly assumes that any packet loss is due to congestion. This assumption does not hold in wireless systems; data can be lost at the receiver from either collisions or link loss. Link loss encompasses a variety of reasons for packet corruption such as reflections, environmental changes, radio orientation, fading etc. Our assertion is that the source should not react to link losses since they are non-deterministic and outside the source's control. On the other hand collisions can be avoided if the source takes specific action. Since our source-based approach does not distinguish between link losses and collision losses, we expect this particular scheme to be very sensitive to link quality. We call this method *uninformed TCP-like collision avoidance*.

3.2. Receiver-Based Collision Avoidance

As discussed above, the MOAP protocol already sends NACKs to the source to inform it about lost packets. If the receiver can distinguish between collision-induced losses and link losses then the source can use this information to make a better decision on whether to decrease the data rate or not. We call this method *informed TCP-like collision avoidance* and note that its performance depends on the accuracy of the collision detection scheme.

If we assume that all sources transmit packets at the same rate $1/T$—that is, during the time frame T, every active source sends a packet—then the receiver can also monitor packet arrivals from all sources and detect large periods of inactivity within a given time frame. If a packet loss due to collision occurs, the receiver can send the information about the 'largest silent period' to its primary source. The source can then try to transmit at those specific times, so as to minimize the chances of further collisions. In essence, the source is trying to interleave its transmissions with background transmissions, while still keeping a constant data rate. We call this method *Phase-offset collision avoidance*.

4. Required Functionality

For both the *Informed TCP-like* and the *Phase-offset* methods, we need to add some extra functionality to both senders and receivers. Specifically, we need to be able to detect collisions, estimate the variation in the transmission times of packets and calculate the largest silent period between two consecutive packet arrivals.

4.1. Collision Detection

Determining collisions at the receiver is not as straightforward as just checking for corrupted packets, since corruption can be due to both link losses and collisions. Also since the packet is corrupted we cannot always inspect it to determine its origin. Therefore, the only thing we can infer is that a loss occurred.

Ideally, the receiver would know exactly when each source's packets arrived. If multiple packets were to arrive at the same time, the receiver could determine

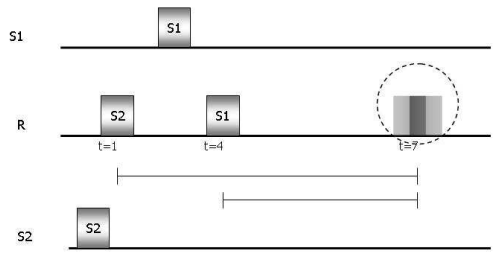

Figure 1. Collision detection at the receiver. Source S_2 transmits at $t = 1$ and informs the receiver that the next transmission will be at $t = 7$. Source S_1 transmits at $t = 4$ and informs the receiver that the next transmission will occur at $t = 7$. If at least one corrupted packet arrives at $t = 7$, the receiver infers that a collision occurred.

that a collision occurred and inform its source. The ideal solution can be approximated by adding a small amount of determinism to the protocol. In many applications, the source will send a packet, and then set a timer to schedule the next transmission. If we determine the next transmission time before sending the current packet, we can add this information in the payload, thus informing the receiver when to approximately expect the next packet. Since every source follows the same rules, the receiver can construct an *arrival timeline* for all the sources it can overhear. If the primary source and any background source(s) indicate that they will arrive at approximately the same time in the future, we can hypothesize that any corrupted packet that arrives at that time is the result of a collision. Figure 1 depicts this situation.

A CSMA MAC includes a backoff mechanism to avoid collisions. This mechanism can add a non-deterministic delay between the time at which the source passes the packet to the MAC layer and starts its timer and when the receiver receives the packet and builds its relative time offset. The source can send the *variance* together with the transmission interval to the receiver, to allow for a more accurate estimation of the next packet's arrival time. We discuss packet transmission time and variance calculations in section 4.2.

The transmission interval and the transmission variance define a range over which the next packet may arrive. Collisions are now determined by the overlap of multiple time ranges. Since packets can arrive at any time in the range, it is possible that though ranges overlap a collision may not occur. Thus, the collision detection model cannot be used to accurately predict *future* collisions. It can only be used for *post-facto* collision detection.

Based on the above, a lost packet is classified as a collision-induced loss if, by the end of expected arrival time plus the transmission variance, the following has occurred:

- No data packet from the primary source has arrived.
- The arrival timeline showed an overlap between the primary source's arrival time and a background source's arrival time.
- At least one corrupted packet was received.

We note that our collision detection mechanism is an *estimator*—it doesn't always guarantee a correct reply. If a source's packet does not arrive when expected, we can only say that it collided with some probability, since we are dealing with ranges as well as incomplete information. It may be possible that a detected collision was predicted to collide with another source, but was actually corrupted due to increased noise levels (link loss). Our estimator would then produce a *false positive*.

If no packet is received at the expected time and no other source was known to transmit at that time, then the packet is assumed to have been lost due to radio propagation. However, if other sources were expected to transmit at that time, the packet's fate is unknown: it could have been lost due to radio propagation or a collision. If the collision occurs in the *preamble* or the *start symbol*, then the receiver will not be able to lock on to the sender and no packet will be received. This would lead to a *false negative*.

Finally, the estimator is only as good as the arrival timeline built at the receiver. If any source's packets are lost then the information those packets hold about future arrival times will not be visible to the receiver. In this case the receiver may incorrectly mark a collision as a link loss. The correctness of our collision detection scheme is directly proportional to the link quality between the receiver and its surrounding sources.

4.2. Transmission Time

When a packet arrives at the receiver, the transmission time (which includes propagation delay and MAC transmission delay) is implicitly included. If we assume that there are no variations in the transmission time, then packets originating from a source every T_p seconds would be received at the receiver at $T_p + T_{tx}$ sec-

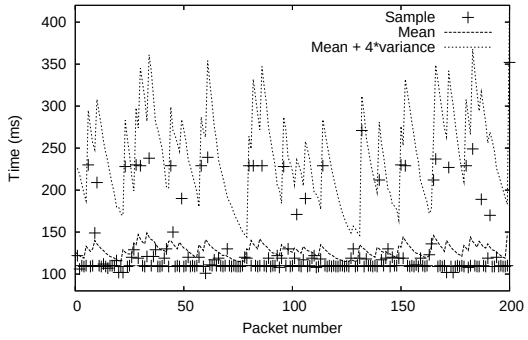

Figure 2. Transmission time variations at the source. The experiment consisted of two nodes. The primary source (node 1) transmits as fast as possible, while the background source (node 2) transmits every 530ms. The link quality between the two nodes was approximately 60%.

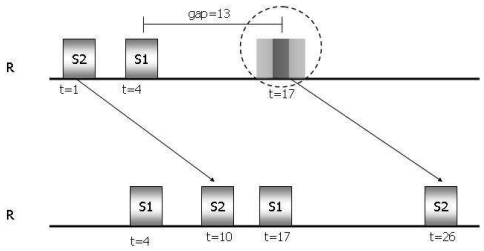

Figure 3. Changing the phase of a transmission while keeping the rate constant can lead to collision avoidance.

onds, where T_{tx} is the transmission time (mote radios have very small ranges, therefore propagation time is negligible).

In reality, the transmission time is not constant. The receiver can wait for a small amount of time after the expected reception time expires, in order to allow for transmission time variations. This additional time, or 'guardband' can be pre-configured based on experimental observations. But a statically assigned value cannot adapt to arbitrary network loads [7]. We thus turn again to TCP for a solution.

The source calculates the transmission time of each packet it sends, by subtracting the transmission start timestamp from the transmission end timestamp. This sample is then used to update the *variance* of the transmission time. The calculation is the same as the one used to estimate the TCP retransmission timeout (RTO) [7]. The value of the variance is sent to the receiver together with the expected time of the next transmission.

The receiver now sets its expected reception timer to $T_p + 4v$, where v is the variance reported by the source, and T_p is the expected arrival time of the next packet.

4.3. Largest Silent Period Detection

Consider a scenario where two sources are transmitting at constant periods T_1 and T_2. We assume that channel utilization is low, so the aggregate rate is not high enough to saturate the channel. Hidden terminal collisions happen when the actual transmission times T_{tx_1} and T_{tx_2} are overlapping. Since the utilization is low, we can avoid the overlap (and thus collisions) while keeping the rates constant, by adjusting the *phase* at which the source's transmissions happen.

The receiver can provide the source with the necessary information by calculating the largest time difference between two consecutive packet arrivals, within a given time period. Since the entire system is clocked off of the source's transmission period, this is also the period used for the calculation of the time difference. We also take into account corrupted packets. Even though those packets are not useful from the application's perspective, they are still an indication that the channel is busy at that particular point in time, so they shouldn't be ignored. Figure 3 shows an example of the phase offset calculation.

5. Implementation

We implemented our designs using TinyOS [5], since motes are MOAP's target platform. Our application sends a series of packets from the source to the receiver in sequential order. Each packet contains a sequence number, the next transmission time and the calculated variance.

The receiver uses a bitmap to store successful packet receptions. Every time a packet arrives from the primary source, the receiver sets a timer based on the next expected transmission time plus any variation defined by the guardband. A packet loss is discovered if no packet arrives by the end of transmission time. At this point the receiver scans the bitmap for the first missing packet. It then sends a NACK to its source. The source will retransmit requested packets before sending any new packets.

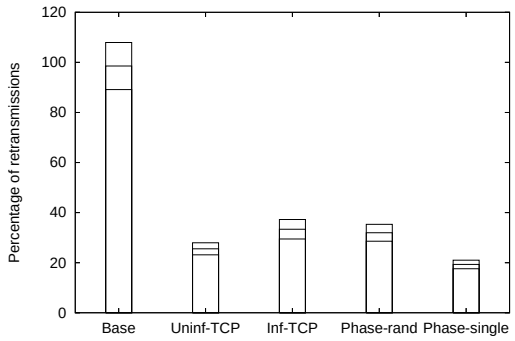

Figure 4. Percentage of retransmissions for each method, with two sources transmitting 200 packets. The single frequency phase offset method was very close to the theoretical minimum of 20%. The base case was the worst, incurring a penalty of almost 100%.

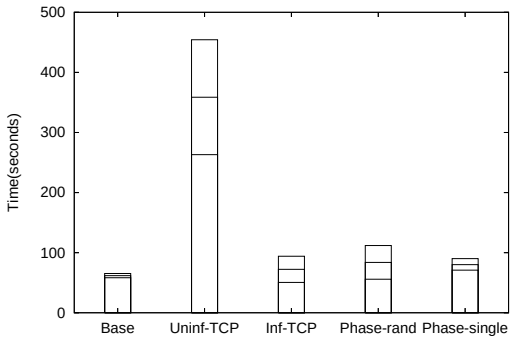

Figure 5. Time required for the primary source to complete its transmission, with two sources sending 200 packets. The base case is the fastest. The phase offset methods and informed-TCP are evenly matched, the latter slightly faster in some cases, due to its adaptive properties. The uninformed-TCP is approximately five times slower than the base case.

5.1. Uninformed TCP-like Collision Avoidance

Our implementation of source-based collision avoidance follows the general design of TCP congestion avoidance. The initial transmission period of the primary source is 500ms. For each data packet that is successfully received, the source will additively increase its data rate, using the rate-based formula described in [7] with $a = 10$.

For each data packet that is lost at the receiver, the source will multiplicatively decrease its data rate. Since MOAP uses NACKs to notify the source of losses, they are used to trigger the multiplicative decrease. To avoid cases where synchronized sources stay synchronized even after the multiplicative decrease, a random floating-point number between $1.5 - 1.9$ is used.

5.2. Informed TCP-like Collision Avoidance

The source will react much like it did in the uninformed TCP-like model. The initial transmission period is again 500ms. For each successful packet sent, the source will again additively increase its data rate. For each packet lost the source will now distinguish if the loss was due to link loss or collision. The source will multiplicatively decrease its data rate *only* when the receiver indicates the loss is due to collision but will take no action on link losses. To inform the source about a possible collision, we have extended the NACK packets to include a 'collision' flag.

5.3. Phase-Offset Collision Avoidance

We again use the NACK mechanism to pass the information back to the source from the receiver. In the absence of NACK packets, the source implicitly assumes that everything is working properly, since no receivers complain. An arriving NACK will now include a time offset indicating the largest silent period time. When the source receives the NACK, it will set a timer for this particular offset. After that timer expires, it will resume its regular transmission, at its constant rate.

6. Evaluation

In order to evaluate the performance of our algorithms, we used the EmStar framework [3] and EmTOS [4]. EmTOS allows development of NesC applications in the resource-rich environment of a 32-bit platform. EmStar includes a packet-level simulator which provides different channel models. For our simulations, we used an empirical channel model, collected from the ceiling array in our lab [2].

6.1. Two Sources, Single Receiver

We initially compared the performance of our algorithms using a simple, three-node setup. The first node was the primary source, the second was the receiver and the third one was the background source. The nodes were arranged in a line. The link quality between nodes 1 and 2 (primary source and receiver) and

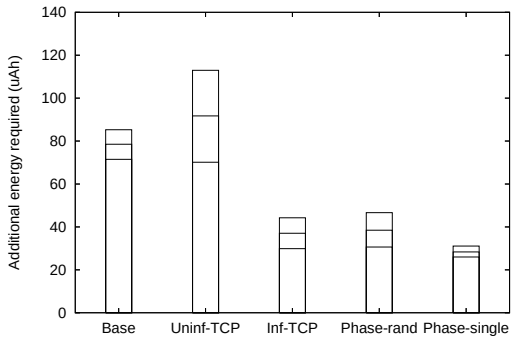

Figure 6. Additional energy requirements, on the primary source, for each method. Single-period phase offset was the best since it had few retransmissions and was also among the fastest to complete. Uninformed-TCP was the worst, due to its very long completion time.

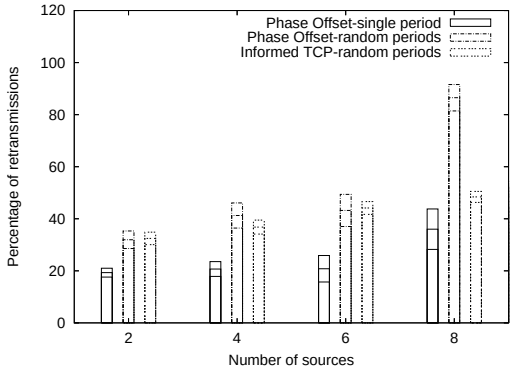

Figure 7. Percentage of retransmissions at the primary source versus number of total sources. Single-period phase offset gives the best results, while random-periods phase offset has the worst performance.

nodes 2 and 3 was approximately 80%. The link quality between the two sources was approximately 50%. The sources in all cases sent 200 packets. The packet size was 150 bytes. All results were averaged over 10 experimental runs. Error bars on the figures represent 95% confidence intervals.

In both TCP-like cases, the background source transmits at a constant period that is initially randomly selected from 500-2500ms. In the phase offset case, we used two different setups. The first one (phase-rand) was equivalent to the TCP-like cases. The second one (phase-single) had the background source transmit at the same rate as the primary source (500ms), but with a random initial starting time. Intuitively, the second case would provide better results. The reason is that if the frequencies of the two sources are not integer multiples of each other, the largest silent period will not be constant; rather, it will keep shifting as one frequency drifts into another. If a collision occurs and is detected, there is little guarantee that the new phase offset will be collision-free.

For this particular set of experiments, we also implemented a 'base' case, in which each source sent packets as fast as possible (back-to-back). We expect this to produce the fastest completion time, but also have the largest number of collision-induced retransmissions.

We first compare the different methods in terms of number of required retransmissions. As Figure 4 shows, the single-period phase offset was very close to 20%—the expected number of retransmission in the absence of collisions, since the link quality was 80%. The base case was the worst, since operating at or close to the channel capacity leads to a large number of hidden terminal collisions. Collision avoidance schemes were thus able to reduce the number of retransmissions by almost a factor of 8.

The reason why uninformed-TCP performed so well in reducing retransmissions can be seen in Figure 5. The completion time of uninformed-TCP was so large, that the background source, which transmits at a constant rate, had already finished its transmissions. The uninformed-TCP method, unable to distinguish between collisions or link losses, was backing off on *each* NACK received. Consequently, for a significantly large period of time, only the primary source was (slowly) transmitting. All losses for that time were due to radio propagation, not collisions. The other methods had almost equivalent results, with the exception of the base case which was the fastest.

By combining the results for latency and number of retransmissions, we can obtain the additional energy requirements that each method incurs. In our calculations, we used the values reported in [9] (20.000nAh to transmit a packet and 1.250nAh per millisecond for idle listening). From Figure 2 we see that the minimum transmission time per packet is approximately 120ms. We therefore used that value for our calculations and assumed that transmission times are constant. We also assumed that when the node was not transmitting, it was idle and thus used the 'idle listening' value for those periods of time. This however ignores the cost of packet reception which is not equal to the idle listen cost. In that sense, our results are more optimistic.

We calculated the total energy usage on the primary source node and then subtracted the energy needed to

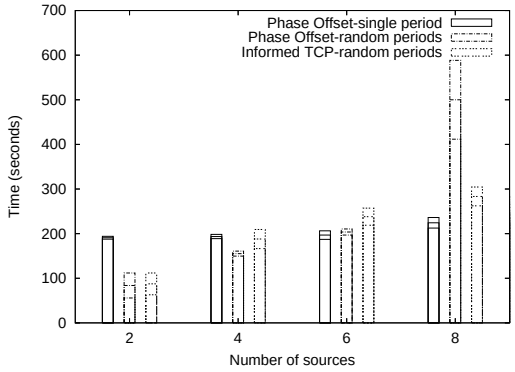

Figure 8. Time required for the primary source to complete its transmission, versus number of total sources. Initially, informed-TCP and random-periods phase offset have the best performance. However, random-periods phase offset scales poorly as the number of sources increases. The other two methods have good scaling properties.

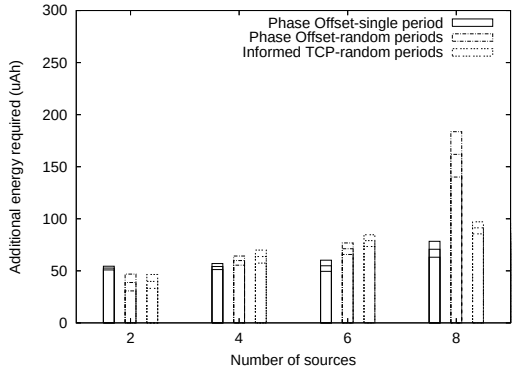

Figure 9. Additional energy requirements on the primary source, versus number of total sources. Single-period phase offset again gives the best performance, followed by informed TCP.

transmit 200 packets *back-to-back*—the optimal energy usage for all cases. The results are shown in Figure 6. The single-period phase offset had the best performance, since it had the smallest number of retransmissions and was one of the fastest methods. Uninformed-TCP had the worst performance. Even though it was among the best in terms of additional retransmissions, it took so long to complete that the idle listen energy consumption had a large negative impact. Phase offset and informed-TCP were able to reduce the additional energy penalty by almost 50% compared to the base case.

6.2. Multiple Sources, Single Receiver

The second set of experiments involved studying the behavior of the three better-performing methods—informed TCP and the two phase-offset variants—as the number of background sources increases. The placement of the initial three nodes, used in the previous set of the experiments remained the same. Extra sources were added in random locations within a 10-by-10 meter square. We run experiments with 2, 4, 6 and 8 sources.

For this particular experimental setup, we changed the single-period phase offset slightly; the transmission period of all sources was now 800ms. With each packet transmission taking approximately 120ms, this variant of phase offset can theoretically accommodate up to seven non-overlapping sources. In practice, since we don't care about background source overlap (only overlaps with the primary source can cause important data loss on the receiver), more sources can be accommodated. The random-period variant and the informed-TCP case had a transmission period of 500ms.

The results, in terms of number of retransmissions, completion time and energy cost are shown in Figures 7, 8 and 9 respectively. Again, as in the two-source case, the single-period phase offset gives the best results. However, the random-periods phase offset now performs much worse, especially as the number of sources increases to eight. Even though the channel isn't necessarily over-utilized (since sources transmit at random periods), the 'frequency drift' effect is very pronounced. In effect, phase-offset cannot find a solution that will be valid for a long period of time.

Even though the single-period phase offset performed the best in all cases, we noticed a significant increase in the number of retransmissions when the number of sources increases to eight from six. This indicates that the channel is reaching capacity and that a further increase in the number of sources will be detrimental to the performance of the phase-offset method. Informed-TCP performed adequately, degrading gracefully as the number of sources increases.

Therefore, if one can design the system to pick the right frequency and make every node take the same frequency constant, phase-offset is the best method to use. If a system does not have a commonly agreed upon frequency, or is highly dynamic, then the informed-TCP method is preferred.

7. Conclusions And Future Work

In resource-constrained wireless sensor networks, collisions can be a major cause of packet retransmissions and hence increased energy consumption. Traditional MAC-layer solutions solve the problem only to certain extent and have rather limited applicability. In this paper we take advantage of inherent multi-packet communication in our specific application of code dissemination to develop two application-based solutions to collision avoidance: a TCP-like method that uses collision detection as an input signal to adjust transmission rate, and a 'phase-offset' method that shifts packet transmission time within a given time frame to minimize collisions. Simulation results show that both of our solutions can significantly reduce the number of retransmissions and energy usage.

Our results indicate that if the system characteristics, such as the maximum number of concurrent sources, are known, one can pick a common transmission period to accommodate all the sources. In this case, the phase offset method is very effective at reducing collisions. In cases where sources transmit using different periods, the informed-TCP method, which can adapt to network dynamics, can be used to effectively reduce collisions.

As part of our ongoing effort, we plan to do several additional experiments that would provide further insight and confidence on the performance of our algorithms:

Running experiments with multiple sources and multiple receivers. Multiple receivers may lead to different feedbacks to each source, this is another interesting aspect that we need to investigate.

Running experiments with real hardware. Every simulator model, no matter how accurate can still not match reality. We therefore plan to run experiments with real hardware, using our ceiling array.

Further exploration of the design space. We are looking into combining the phase-offset method with the informed-TCP method by using the latter to adjust the common time frame period while using the former to shift each source's transmission time within the time frame.

Integrating the most appropriate solution in MOAP. Since MOAP was the driving application for our work, we plan on integrating our final collision avoidance solution into it to improve its performance.

References

[1] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A Media Access Protocol for Wireless LAN's. In *ACM SIGCOMM 1994*, pages 212–225, London, UK, August 31–September 2 1994.

[2] A. Cerpa, N. Busek, and D. Estrin. SCALE: A tool for Simple Connectivity Assessment in Lossy Environments. Technical report, CENS-TR-21, September 2003.

[3] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *Proceedings of the 2004 USENIX Technical Conference*, Boston, MA, 2004. USENIX Association.

[4] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. Tools for deployment and simulation of heterogeneous sensor networks. In *Proceedings of SenSys 2004*, November 2004.

[5] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Arhitectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.

[6] C. T. Inc. Mica2 wireless measurement system datasheet, http://www.xbow.com/products/product_pdf_files /datasheets /wireless/6020-0042-03_a_mica2.pdf.

[7] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18, 4:314–329, 1988.

[8] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for future high bandwidth-delay product environments, 2002.

[9] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM Press, 2002.

[10] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

[11] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 171–180. ACM Press, 2003.

[12] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 14–27. ACM Press, 2003.

[13] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of IEEE INFOCOM*, 2002.