CONCURRENT TRANSMISSION WITH ADAPTIVE FORWARDER SET

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Qiang Li

December 2014

CONCURRENT TRANSMISSION WITH ADAPTIVE FORWARDER SET

Qiang Li

APPROVED:

Dr. Omprakash Gnawali, Chairman
Dept. of Computer Science, University of Houston

Dr. Jaspal Subhlok
Dept. of Computer Science, University of Houston

Dr. Rodrigo Fonseca
Dept. of Computer Science, Brown University

Dean, College of Natural Sciences and Mathematics

Acknowledgements

Special thanks to my academic advisor, Dr. Omprakash Gnawali, without whose guidance this work is unimaginable. Also sincerest appreciation for Dr. Jaspal Subhlok and Dr. Rodrigo Fonseca, for their time and valuable opinions, which make this paper more concrete.

This work was partially supported by the National Science Foundation under grant no. IIS-1111507.

CONCURRENT TRANSMISSION WITH ADAPTIVE FORWARDER SET

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Qiang Li

December 2014

Abstract

Precisely-timed concurrent transmission(CX) is relatively new in wireless sensor network (WSN) area, and protocols based on CX can implement reliable and conceptually simple flooding mechanism. These protocols work well; however, they are intrinsicly energy inefficient for non-flooding traffic models, because in such cases not every node needs those flooding packets, but all of them have to receive and then forward. Efforts have been spent in reducing forwarder set size for those cases, typically by removing those nodes which are not "between" the packet source and destination so that they are "useless". These efforts are effective, but they have to keep some unnecessary forwarders because they rely on topology information and this information is subject to network dynamics and then not precise. Here we present a new mechanism CXAFS to enhance Forwarder Selection(CXFS) protocol by further removing unnecessary forwarders. We implement 3 variants of CXAFS and evaluate them on our 27-node testbed. Results show that CXAFS can reduce forwarder set size by up to 40% and reduce duty cycle by 40%, at the expense of lower packet reception ratio(less than 10% lower).CXAFS is a feasible option when energy efficiency is top priority and packet reception ratio is not extremely important.

Contents

List of Figures

Chapter 1

Introduction

Fast network flooding based on precisely time-synchronized concurrent transmission (CX) and radio capture effect [10] is an effective way to spread packets among multiple nodes with high probability: every node will rebroadcasts exactly the same packet it receives at exactly the same time to leverage constructive interference. This scheme has several key benefits, including high Packet Reception Ratio (PRR) and good throughput. More important, this flooding-based mechanism is stateless - it does not need any routing state, so it is conceptually simple and very good for high-mobility wireless sensor network (WSN), for which common routing protocols would have to frequently re-discover routes and exchange routing information with high protocol overhead. WSN communication protocols such as Glossy [8], Low-power Wireless Bus (LWB) [7], Flash Flooding [12], Insteon [13], and Forwarder Selection (CXFS) [2] are examples of protocol based on CX and they perform well in multihop WSNs.

Despite all these benefits, CX is inherently not energy efficient for non-flooding traffic patterns, i.e., not all nodes need all packets. For example, point-to-point communication, in which only the destination node needs to receive the packets, plus necessary forwarders to make this communication possible, while CX flooding requires all nodes to be involved in packet receiving/rebroadcasting. An obvious solution to improve this scheme is to reduce forwarder set size so that those nodes not in forwarder set could sleep to save energy. In fact, this is exactly what CXFS does: it utilizes real-time topology information to choose nodes in "correct direction" as forwarders and make other nodes sleep.

Although CXFS works well in removing unnecessary forwarders and could reduce duty cycle by 30% compared with simple CX flooding, it relies on hop count as metric to infer topology, which is not accurate, especially when network dynamics exists. To offset, CXFS introduces tolerance in topology analysis and then unnecessary forwarding nodes.

To further reduce forwarder set size, we design and implement our new protocol based on CXFS, Concurrent Transmission with Adaptive Forwarder Set (CXAFS), which uses Packet-Loss-Feedback (PLF) technique to test and remove dispensable forwarders to save more energy while maintaining acceptable packet reception rate. Under our new protocol, we use CXFS algorithm to calculate forwarder set as first step, and then intentionaly introduce packet loss to detect which forwarders are *critical* and which are *dispensable*. After packet loss happens, we will revive critical forwarders and remove dispensables.

In this paper we make the following contributions:

- We propose and implement 3 variants of CXAFS on TinyOS 2.x;
- We evaluate these 3 variants and compare them with CXFS and simple flooding system.

Our evaluation result from 27-node testbed in an academic building in University of Houston shows that CXAFS can reduce forwarder set size by up to 40% and reduce ducy cycle by 40%, at the expense of lower packet reception ratio (less than 10% lower compared with CXFS).

Chapter 2

Related Work

Precisely synchronized concurrent transmission (CX), or transmission with constructive interference, is a relatively new topic while concurrent transmission itself has long history in WSN study. In fact, most previous study of concurrent transmission focuses on capture effect [10], a phenomenon associated with reception "in which only the stronger of two signals at, or near, the same frequency will be demodulated" [15], as long as the stronger signal is strong enough compared with the other signal. From this perspective, CX is totally different: signals are highly synchronized so that they will enhance each other instead of interfering destructively.

Although CX has obvious advantages, the reseach community did not put much significance on this topic in the past. One of the main reasons is sensor motes typically have very limited computation resources, and CX was considered hard to implement on such simple platforms because it requires high-resolution time synchronization for interference to be constructive.

This situation lasted until Dutta et al. first utilized hardware-generated acknowledgements to implement transmission synchronization [6]. These acknowledgements are provided by IEEE 802.15.4-compliant radios, and have no extra costs. Dutta et al. uses this "free" mechanism to effectively wake up large network. Some receiver-initialized MAC protocols, such as A-MAC [5] and Flip-MAC [3], use the same mechanism for medium contention and arbitration.

After this, Ferrari et al. [8] use software-based method to implement CX on TelosB mote. Their implementation is on Contiki embedded OS [4], based on which they built an effective, highly synchronized and fast flooding protocol - Glossy. Later, Ferrari et al. extend Glossy to virtual wireless bus (LWB) [7] by adding a global scheduler, which generates and disseminates a TDMA schedule to coordinate the whole network. Insteon [13] is pretty much a commercial counterpart of Glossy, although it is not open-source and its implementation details are not revealed.

The limitation of Glossy is also investigated. Wang et al. in [14] found that time synchronization errors could accumulate until constructive interference downgrades to destructive, as the network diameter grows. They then proposed new flooding protocol to mitigate the time errors. According to their simulation, the new protocol is more scalable than Glossy.

From energy efficiency perspective, Carlson et al. [2] propose and implement CXFS, a Glossy-like flooding protocol with selected forwarder set, on TinyOS [11]. This protocol improves energy efficiency by muting those motes not between the source and the data sink, so that only necessary motes will be in forwarder set. However, there are still unnecessary forwarders in the final set, and our work is

focused on removing those motes while maintaining acceptable packet reception ratio.

Chapter 3

Adaptive Forwarder Set

In this section, we will illustrate the difference among concurrent transmission flooding, CXFS and our Adaptive Forwarder Set(CXAFS), and define CXAFS in formal terms. To make it clear, we define WSN as a directed acyclic graph, with each vertex a node and each edge a valid link. To simplify, we may omit some links that do not matter.

3.1 Two Ends of Forwarder Set: Single Path and Flooding

First of all, we will give the definition of forwarder set. As the name implies, we define the set of all nodes that will forward received packets from source Src to destination Dst as a forwarder set of F{Src, Dst}. According to this definition,

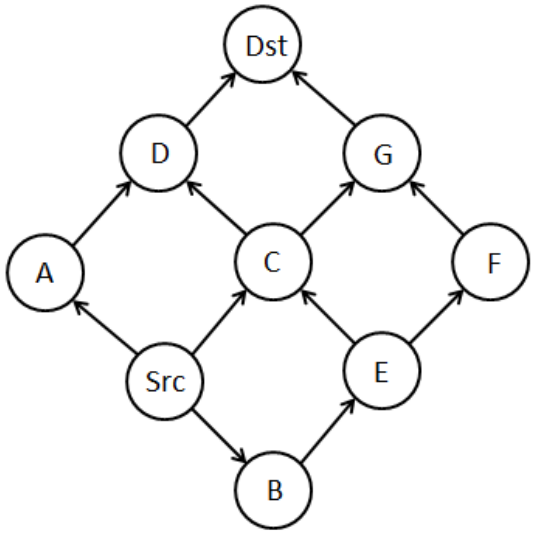


Figure 3.1: Topology of a sample network. Each edge is a link of cost 1.

there are obviously 2 extremities: single-path forwarder set, in which each node is indespensable for Dst to receive packet from Src and the total link cost is minimal; full network flooding, in which all nodes will involve in forwarding.

(1) Single Path: Single-path protocols will build a valid forwarder set for (source, destination) pair, which is of minimal total cost. Here cost could have different definitions relying on protocols themselves. For example, it could be hop-count, ETX, or anything else the protocol uses as metric. In figure 3.1, in which each link has cost 1, F{Src, Dst} could be {A, D}, {C, D} or {C,G}, depending on protocol implementation.

These kind of protocols, intuitively, are very energy efficient after establishing forwarder set, from the perspective of number of forwarders and less inter-nodes

interference. However, they are less reliable: they are vulnerable to network dynamic and external interferece, and will incur high overhead in rebuilding forwarder set in high-mobility network and/or under heavy interference.

(2) Flooding: When using flooding, every node in the network will forward the packets they receive, i.e., the network will try its best to transmit the packet from source to destination. In Glossy, for instance, all nodes except source and destination will transmit the same packets at highly-synchronized time spots to improve probability of packet reception. In fact, this scheme establishes the upper bound of PRR given same level external interference, at the expense of extra energy consumption.

In addition, flooding does not need maintain routing status or discover a specific route, so it is especially suitable for high-mobility networks and inherently resistant to external interference. In another word, flooding is designed for reliability.

According to the analysis above, we can see that those two extremities are pursuing two different aims to the extreme: single-path for energy efficiency and flooding for reliability. Naturally, we wonder if there is a in-between solution which is better balanced between energy consumption and reliability. CXFS is such a solution.

3.2 Forwarder Selection in Concurrent Transmission

CXFS is similar to full-network CX flooding, and the difference is CXFS tries to remove those unnecessary forwarders from forwarder set. Here by "unnecessary" we

refer to those nodes that are not on the shortest path (or more accurate, lowest-cost path) from source node to destination. In ideal environment where there is no external interference, those unnecessary nodes can be muted safely without affecting PRR because the destination node can still receive packets from source through shortest path(s).

In CXFS each node will detect whether it is "necessary", and this detection is based on 3 pieces of distance information: d_{sn}, shortest distance from self to source; d_{nd}, shortest distance from self to destination; and d_{sd}, shortest distance from source to destination. Since the distance is defined as hop count in CXFS, a node is on the shortest path from source to destination if and only if:

$$d_{sn} + d_{nd} = d_{sd}. \tag{3.1}$$

The distance information will be updated at the beginning of each round of transmission.

Take figure 3.1 as an example. For node A, its d_{sn} is 1, d_{nd} is 2, and d_{sd} is 3. According to formula (3.1), A is in forwarder set, and the same to node C, D and G. On the other hand, for node B, its d_{sn} is 1, d_{nd} is 4, and d_{sd} is 3, so B will go to sleep instead of forwarding packets. Node E and F are not in forwarder set for the same reason. So the forwarder set according to CXFS is {A, C, D, G}.

However, in real world things are different. Due to external interference and network dynamic, plus asymmetry of distance information [1] (i.e., distance from A to B could be different with distance from B to A), formula (3.1) could become

invalid in many cases. To mitigate, in practice CXFS uses the following formula:

$$d_{sn} + d_{nd} \leqslant d_{sd} + b \tag{3.2}$$

where b is constant tolerance. Given $b = 2$ (in practice this is the default value in CXFS implementation), we will find that in figure 3.1, forwarder set will include all nodes even if we use CXFS. So CXFS is still too conservative in energy saving.

3.3 CXAFS

Our protocol, Concurrent Transmission with Adaptive Forwarder Set (CXAFS), is based on CXFS. CXAFS uses CXFS as starting point. Before we can discuss CXAFS in detail, we will introduce several concepts and terms first.

(1) Slot and frame: CX and CXAFS uses these 2 concepts to coordinate transmission. It devides time into slots, in each slot there is only 1 source node, which will communicate to the data sink in this slot. Each slot is further divided into multiple frames, and packet transmission can only start at the beginning of a frame. Frame length is designed long enough so that frames will never overlap.

(2) Master, slot owner and leaf: In CX and CXAFS, there is only one data sink for a network, and it is also the coordinator of slot/frame. This node is called master; in each time slot, there is only 1 node allowed to send packets to master, and this is slot owner; other nodes are all leaf nodes, no matter they will forward packets or just sleep.

(3) CTS and status message: When a new slot starts, master will choose a node

as owner of this slot, and then broadcast a control message to all other nodes. This message contains slot owner's information and all nodes will use this message to update their distance to destination information. This packet is called CTS; after receiving CTS, slot owner will broadcast a packet as reply. This packet is called status message. It contains slot owner's status and the distance information from master to slot owner, and leaf nodes will also update their distance to source information according to status message. After status message slot owner will then start to send data to master.

CXAFS generates forwarder set exacly like CXFS as the first step. After that, it uses an algorithm we called Packet-Loss Feedback (PLF) to reduce forwarder set. In next section we will discuss in detail about the design and implementation of CXAFS.

3.4 Basic Packet-Loss Feedback Algorithm

From definition of CXFS we can find that in theory CXFS will keep all nodes on shortest paths in forwarder set; in addition, CXFS in practice has to have some margin when it judges whether one node should forward or not, so the final forwarder set will also have some unnecessary nodes that can be removed. These 2 factors make it possible and necessary to reduce forwarder set size further. We then designed the algorithm, Packet-Loss Feedback (PLF), for this purpose.

PLF consists of 2 consecutive phases, Leave-phase and Return-phase. Leave-phase starts with CXFS forwarder set. When a node receives a packet, it will check

if it is a CXFS forwarder. If it is not even in CXFS forwarder set, it will sleep until next slot starts, just like a regular CXFS node; otherwise it will forward this packet and then try to remove itself from forwarder set with a small probability. If it leaves, it will sleep until next slot starts, or it will wait for next packet. This Leave-phase will continue until packet loss happens (packet loss will happen because forwarder set is always shrinking in Leave-phase), and then Return-phase takes over.

In Return-phase, each node will judge if the packet loss is relevant to its leave. This can be done by checking if packet loss happened inside a window immediately after its leave. If relevant, it proves that this node is necessary for packet transmission and then this node should return to forwarder set, otherwise it will remain out. This smaller forwarder set will remain unchanged until master node decides to switch to next slot owner, and a new 2-phase PLF detection will be done for the new slot owner. More details of PLF can be found in Algorithm 1.

PLF will remove forwarders randomly from CXFS forwarder set. According to previous analysis, we know that in CXFS forwarder set, there are both redundant and unnecessary nodes. Our goal is to remove unnecessary nodes, but PLF can not guarantee. Instead, it will remove them with equal probability. So PLF could hurt the redundancy of CXFS while removing unnecessary nodes for energy saving;

PLF is not free: it intentionally induces packet loss, and since packet loss could be detected only at the beginning of a slot, i.e., it can not respond immediately after packet loss happens, it inevitably lowers down the PRR. However, the cost of PLF for a given slot owner is relatively constant, and is independent with session length. So to make the cost relatively low, PLF is suitable only for those long sessions.

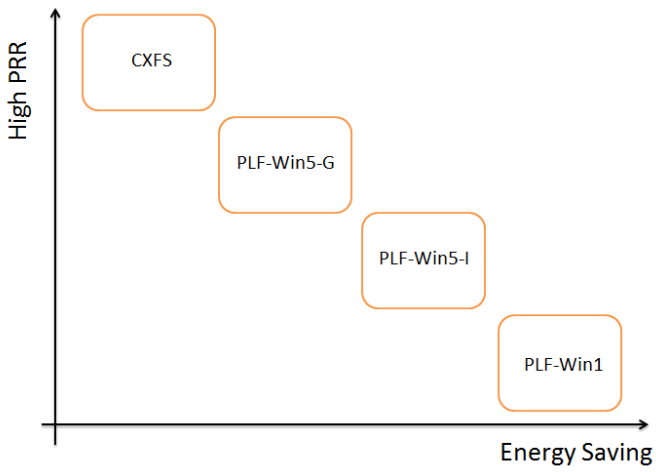


Figure 3.2: Design goal of 3 CXAFS variants compared with CXFS

3.5 3 Variants of PLF

PLF essentially is an algorithm saving energy at reasonable expense of PRR. To find out the best balance point between energy efficiency and PRR, we design 3 variants of PLF, each of which has different design preference as shown in figure 3.2. These 3 variants have exactly the same Leave-phase but they are different in Return-phase in how to respond to packet loss.

(1) PLF-Win1: A node running this variant cares about only packet-loss that happens immediately after this node leaves forwarder set. Other packet losses will not be taken into account. For example, a node leaves forwarder set after it forwards the #25 packet, and master successfully receives #25 and #26 packets but misses #27, then this node will not return; only if master misses #26 packet will this node comes back. This variant is the most aggressive in energy saving, because in fact it could be responsible for loss of #27 packet due to network dynamic but we do not

take that into account;

(2) PLF-Win5-I: "win5" here means a node running this variant will take action to repond to the packet loss happenning within 5 consecutive packets after this node leaves forwarder set. If PRR in those 5-packet-window is lower than a threshold, in our implementation 60%, this node will think it is necessary for current slot owner so it will return to the forwarder set. "I" here means this "individual" node will return and it will not trigger any global reaction;

(3) PLF-Win5-G: As the name implies this variant only considers the 5 consecutive packets immediately after the node leaves forwarder set. If PRR in this 5-packet-window is higher than or equal to a threshold (in current implementation 80%), this node will keep out of forwarder set; if PRR is lower than 80% but higher than or equal to 60%, this node will return to forwarder set; if things are even worse, i.e., PRR is less than 60%, the whole network will revert to CXFS to help keep PRR at reasonable level. So "G" here means that packet loss could trigger network-wide reaction to respond. This variant is designed most conservative in energy saving but most aggressive in maintaining PRR level.

3.6 Implementation Details of CXAFS

We implement CXAFS on "Bacon" mote, which is based on TI CC430 SoC and MSP430MCU [9]. Here we will focus on high-level considerations for CXAFS.

(1) How to detect selfs role: In CXAFS, there are 3 roles: master, slot owner,

and (potential) forwarder. A node can use a special interface to detect if it is master or not because this role is burnt statically, and non-master nodes can check CTS message to know who is the slot owner;

(2) How to create long sessions: Through our previous discussion, we know long session is indispensable for PLF to amortize the cost of setup of this pattern. In original CXFS implementation, master will communicate with all nodes in polling manner for data download. Thats to say, in this slot it will communicate with node A, then in next slot it will communicate with node B even if node A tells master it still has data pending. We change this pattern so that the master will contact the same node until that node says through status message it has no more data pending. This way user can conviniently create long session;

(3) How to detect and notify packet loss: For master node, it is very hard to detect if packet loss has happened and how many packets are lost during a slot, because the phenomenon it can observe is just no packets is arriving. Using timeout could be a solution, but it is not reliable, because it is possible that the slot owner has no more data. Even worse, master has no direct way to notify forwarders the packet loss has happened. The reason is in every slot there are only 3 control messages, CTS, status and EOS, among which only CTS is sent by master and it is only sent at slot beginning; in beginning of every frame, slot owner and forwarders will send/forward packets, and then fall asleep. So master has no chance to send a notification while at the same time forwarders can receive. Since packet loss detection and immediate notification from master are not practical, we make use of existing framework to implement detection directly on every forwarders: instead of detecting if packet loss

has happened, master and slot owner will only pack the serial numbers of their received/sent packets into CTS/status message, respectively, at the beginning of every slot. Then forwarders can compare these 2 groups of numbers to get to know all information about packet loss in last slot;

(4) Misc problems: There is a potential problem in PLF detection: EOS message could be lost also, which is a control message sent from slot owner to master to notify a slot should end. Fortunately this message is not indispensable: it just carries the data pending status of slot owner. Even if it gets lost, master will get this information in next time slot.

Algorithm 1 Basic Packet-Loss Feedback for Leaf Node

```
1: function SlotStarted()                                   ▷ Invoked when slot starts
2:     WaitForCTS();
3:     WaitForStatusMsg();
4:     if SlotOwner_Changed() then
5:         PLF_Left ← false;
6:         PLF_Returned ← false;
7:         return
8:     else if !Packet_Loss_Happened_Last_Slot() then
9:         return
10:    else
11:        if PLF_Left && PL is immediately after PLF_Left then
12:            Return to Forwarder Set;
13:            PLF_Returned ← true;
14:        end if
15:    end if
16: end function
17:
18: function PacketReceived(msg)                            ▷ Invoked when get a packet
19:    if !In_CXFS_Forwarder_Set() then
20:        Sleep until next slot starts;
21:        return
22:    else if !PLF_Left then
23:        Forward msg;
24:        Randomly choose to leave forwarder set;
25:        if Choose to leave then
26:            PLF_Left ← true;
27:        end if
28:    else if PLF_Returned then
29:        Forward msg;
30:    else
31:        Sleep until next slot starts;
32:    end if
33: end function
```

Chapter 4

Evaluation

We evaluate CXAFS on our 27-node Bacon testbed deployed in an academic building (PGH building) on UH campus. The map of this testbed is shown as figure 4.1. This testbed covers physically 9 offices, about 20m x 55m area. When we set mote transmission power to -6 dBm, its diameter is about 7∼9 hops, depending on environmental interference.

In all our experiments, we set node 0 as master, and in every round master collects 100 consecutive packets from each leaf node respectively. We repeat each experiment at least three times and calculate the average of the results. Every node records its radio activity information with a 6.5MHz timer on radio chip and sends it to the serial port.

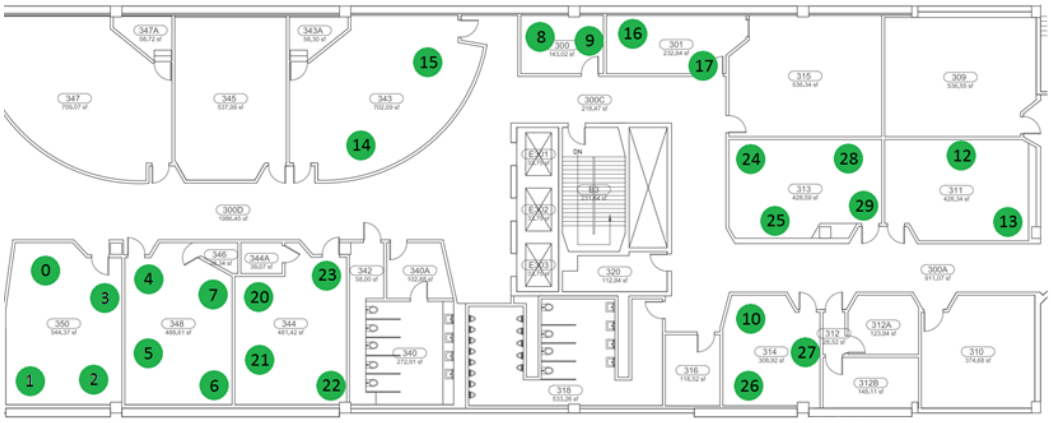
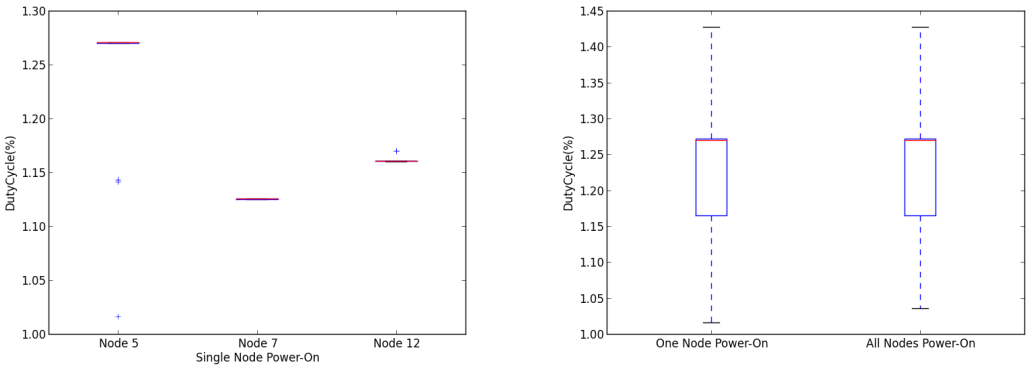


Figure 4.1: Deployment of Bacon Testbed in 3rd floor in PGH building on UH campus.

4.1 Baseline: Duty Cycle without Workload

Before we can evaluate the CXAFS performance, we need to master the baseline of our testbed without any valid workload. Here by "baseline" we refer to CXAFS's base duty cycle when there is no master node at all. Since CXAFS, CXFS and CX flooding are exactly the same with null workload, the baseline is a property more of the testbed than of a specific protocol.

To get accurate baseline information, we conducted 2 experiments. In the first experiment, we turn on 27 motes one by one to disable any interaction among nodes, and then measure their base duty cycle. We randomly choose 3 motes and their results are shown in figure 4.2(a). We can find that different motes have their own ducy cycle patterns, but these patterns are very stable. Other motes have very similar results. Then we turn on all leaf nodes to check how interactions will affect node's base duty cycle. Figure 4.2(b) shows the aggregated results from these 2



(a) Duty cycle of 3 random nodes out of 27 motes.

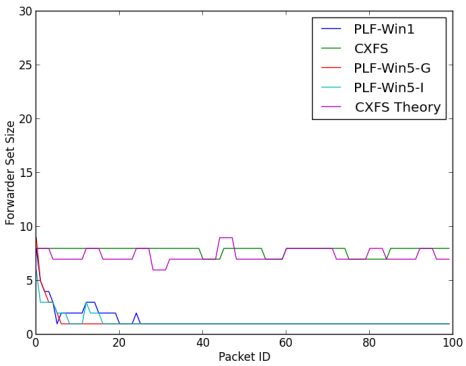
(b) Duty cycle distribution of all 27 motes.

Figure 4.2: Baseline of Bacon testbed: baseline is irrelevant to leaf interactions, but different motes have distinct pattern.

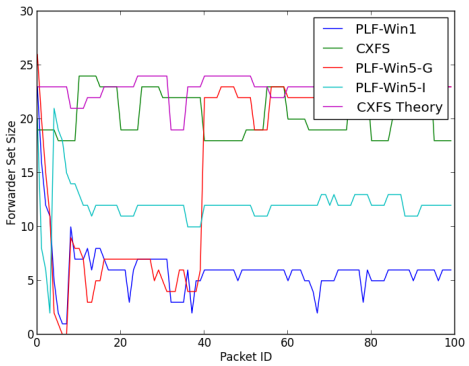
experiments, through which we can see that interaction among leaf nodes has very limited impact on base duty cycles. So we conclude that duty cycle increment is caused purely by workload and protocol overhead.

4.2 Forwarder Set Size

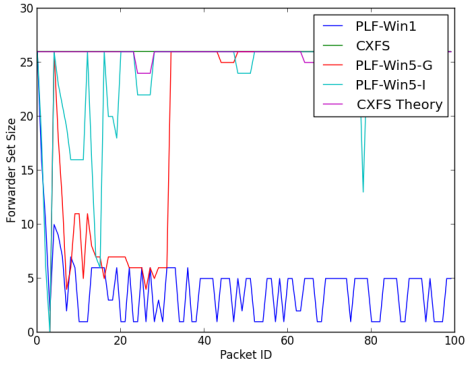
CXAFS is designed to save more energy by reducing the size of forwarder set due to lower down workload compared to CXFS. To evaluate the effectiveness of CXAFS, we conducted a serie of experiments to see how the forwarder set size changes across time, and all those experiments are done during 1pm to 5pm so that they will be under similar level of interference. During these experiments, each mote has in total 100 packets ready, each packet 66 bytes long. For each packet, every mote receiving it will record whether it will forward it or not. In addition, variant PLF-Win1 will



(a) Typical close-to-master node.



(b) Typical far-from-master node.



(c) Node that suffers from severe interference.

Figure 4.3: Forwarder set size comparison among 3 typical types of nodes.

also record its forwarding decision if it is running CXFS. We call this record CXFS-theory result. If this CXFS-theory value is in same level with CXFS result, we can make sure these experiments are in similar environment.

Figure 4.3(a) shows the results from a typical close-to-master node. We can see that CXFS-theory line is very near to CXFS line, indicating that these results from different protocols are comparable. For all the 3 variants of CXAFS, there is a reduction of number of forwarders at the very beginning, indicating that forwarders

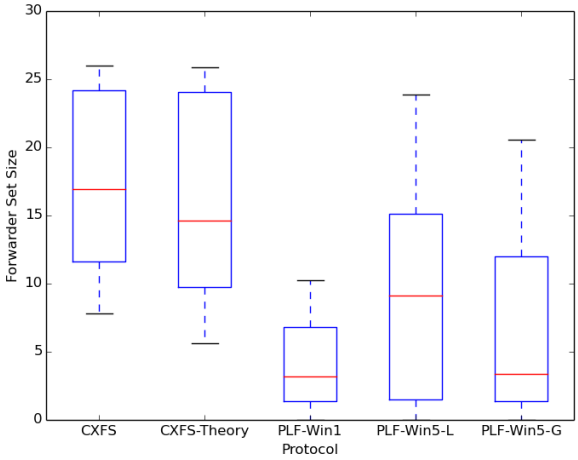


Figure 4.4: Aggregate forwarder set size across 5 protocols.

are in Leave-phase and are leaving. Very soon (within 5 packets) the forwarder set size reaches 1, compared with CXFS set size 8. Another fact is 3 variants all have only 1 forwarder after Return-phase, so that the 3 lines overlap, suggesting PRR should be very good.

Figure 4.3(b) is from a typical far-end node. As discussed in [14], the time synchronization errors could accumulate as distance grows. We can find this effect obviously in this figure: there are lots of sharp angles on PLF-Win1 line, which means for that specific packet, some forwarders failed to receive it. An obvious fact is for those far-end motes, CXFS forwarder set will be very big because for each hop CXFS will introduce some redundant/unnecessary motes. On this mote the CXFS forwarder set size is always between 18 to 25 out of 26, showing that CXFS is not so energy efficient.

Some nodes suffer from low PRR, which could be caused by network dynamics

or interference. Figure 4.3(c) is one of them. We can find that for PLF-Win5-G and PLF-Win5-I, in Return-phase their forwarder set size will soon increase to 26 because of bad PRR, while for PLF-Win1, the size never exceeds 5. This illustrates the limitation of PLF-Win1, whose window size is only 1 so it is vulnerable to network dynamic and interference.

Figure 4.4 shows the forwarder set size aggregated from all the nodes across the four protocols. The CXFS-theory shows the CXFS forwarder set size computed during the initialization of the CXAFS variants. If the experiments with CXAFS and CXFS experiments are done under similar wireless environment, the CXFS and CXFS-theory values would overlap. In our case, the difference between CXFS and CXFS-Theory median is less than 1.4 (out of 28 nodes), so we can be reasonably confident that all the protocols were tested under similar wireless environment.

4.3 Duty Cycle

In theory CXAFS should have lower duty cycle than CXFS, because even in worst case, CXAFS will not have larger forwarder set than CXFS, so on average every mote will be less involved in forwarding when running CXAFS. Since different motes have different baseline duty cycle pattern in our testbed, we aggregate duty cycle data by motes. Figure 4.5 is from a random mote and it proves our expectation: CXAFS does have less duty cycle than CXFS. Other motes have similar results.

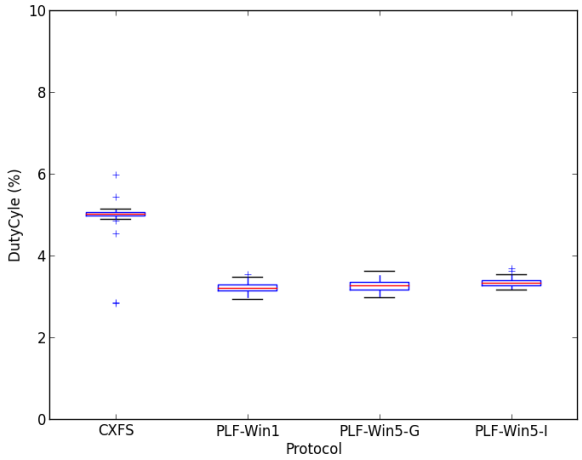


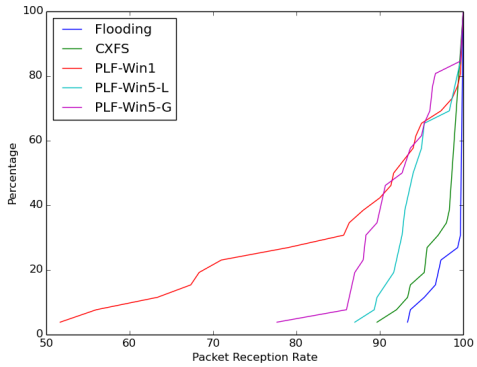
Figure 4.5: Aggregate duty cycle distribution for a random node.

4.4 Packet Reception Ratio

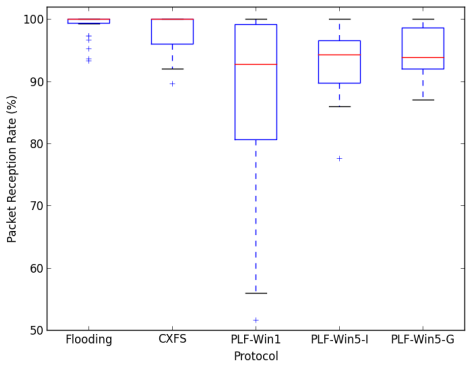
Compared with CXFS, CXAFS has smaller forwarder set, lower duty cycle, and in turn better energy efficiency. But these advantages come at a cost. As we discussed in previous section, CXAFS could remove redundant motes as well as unnecessary motes from forwarder set because it has no way to tell them apart. As a result, CXAFS will loss some reliability and then will suffer lower PRR.

Figure 4.6 shows comparison among flooding, CXFS and 3 variants of CXAFS. From figure 4.6(a) we can see that CXFS has very good PRR even compared with flooding, with no node below 90%. 3 variants of CXAFS show different results: PLF-Win5-G have best PRR, PLF-Win1 worst, complying with our design goal.

In addition, nodes running PLF-Win5-G and PLF-Win5-I have more coherent PRR, while PLF-Win1 are more divergent (figure 4.6(b)).



(a) CDF of PRR for 5 Protocols.



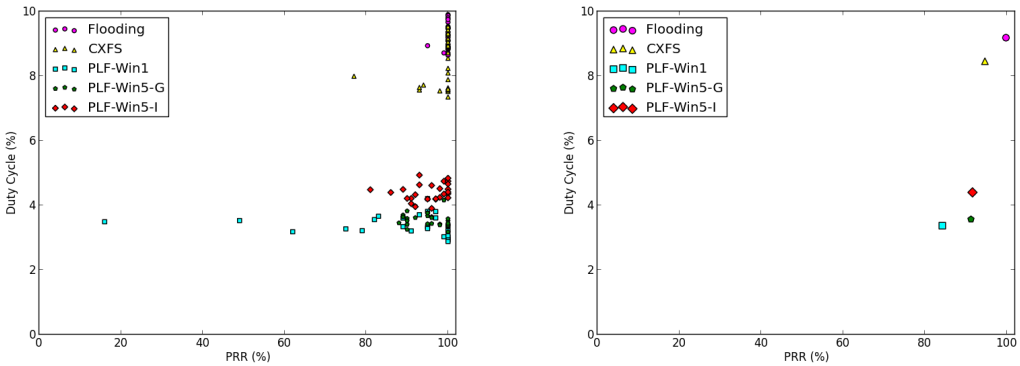
(b) Distribution of PRR across the whole testbed.

Figure 4.6: PRR comparison among 5 protocols.

4.5 PRR vs. Duty Cycle

CXAFS is designed to trade energy efficiency with reliability, and its 3 variants offer different design tradeoff in between duty cycle and PRR. To evaluate if this tradeoff is worthwhile, we plot all nodes in PRR-DutyCycle domain, as shown in figure 4.7(a). We can see that 3 CXAFS variants are far under Flooding and CXFS, indicating duty cycle of CXAFS could be up to 50% lower than CXFS; also these variants are on the left of CXFS and Flooding dots, suggesting lower and more divergent PRR. But according to figure 4.7(b), which shows average of all dots, average PRR of CXAFS is less than 10% lower than flooding and CXFS. These results proves that this tradeoff is reasonable and has its value, especially suitable for those WSNs in which energy conservation is critical.

Another notable fact is distribution of PRR for PLF-Win1 is too divergent, implying that this variant is probably not a pratical option for real-world WSN although



(a) Distribution of all nodes running different protocols.

(b) Distribution of characteristics of 5 protocols.

Figure 4.7: PRR vs. Duty Cycle of 5 protocols

the average PRR is acceptable. However, the other 2 variants, PLF-Win5-I and PLF-Win5-G, show coherent results and are better balanced between reliability and energy efficiency.

Chapter 5

Conclusion

Compared with traditional routing protocols for WSN, concurrent-transmission based protocols, such as Glossy, Low-Power Wireless Bus, etc., have shown their advantages, including but not limited to good throughput, and high reliability. However, this flooding-based protocol family is inherently energy inefficient, because flooding will involve some nodes whose forwarding is not helpful for PRR.

CXFS improves energy efficiency, but its design priority is still to keep very high PRR. For those cases where PRR is not of highest importance while energy conservation is critical, our CXAFS is a feasible option: it reduces up to 40% power consumption compared with CXFS at reasonable expense of PRR. In addition, CXAFS provides 3 variants that hit different balance points between energy conservation and high yield, suitable for wide range of applications.

Bibliography

[1] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Trans. Sen. Netw.*, 8(4):34:1–34:33, Sept. 2012.

[2] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali. Forwarder selection in multi-transmitter networks. In *DCOSS*, pages 1–10. IEEE, 2013.

[3] D. Carlson and A. Terzis. Flip-mac: A density-adaptive contention-reduction protocol for efficient any-to-one communication. In *in Proceedings of the 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS). IEEE*, pages 1–8.

[4] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[5] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 1–14, New York, NY, USA, 2010. ACM.

[6] P. Dutta, R. Musloiu-e, I. Stoica, and A. Terzis. Wireless ack collisions not considered harmful. In *In HotNets-VII: The Seventh Workshop on Hot Topics in Networks*, 2008.

[7] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *IEEE Transactions on Communications*, volume 24, pages 531–539, 1976.

[8] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In X. D. Koutsoukos, K. Langendoen, G. J. Pottie, and V. Raghunathan, editors, *IPSN*, pages 73–84. IEEE, 2011.

[9] T. Instruments. Msp430 soc with rf core. http://www.ti.com/product/cc430f5137, 2010.

[10] J. F. K. Leentvaar. The capture effect in fm receivers. In *Proceedings of the 10th ACM conference on Embedded Networked Sensor Systems(SenSys 12)*, 2012.

[11] P. A. Levis. Tinyos: An open operating system for wireless sensor networks (invited seminar). In *MDM*, page 63. IEEE Computer Society, 2006.

[12] J. Lu and K. Whitehouse. Exploiting the capture effect for low-latency flooding in wireless sensor networks. In T. F. Abdelzaher, M. Martonosi, and A. Wolisz, editors, *SenSys*, pages 409–410. ACM, 2008.

[13] I. SmartLabs. Insteon: The details. http://www.insteon.net/pdf/insteondetails.pdf, 2005.

[14] Y. Wang, Y. He, X. Mao, Y. Liu, Z. Huang, and X. Li. Exploiting constructive interference for scalable flooding in wireless networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2104–2112, March 2012.

[15] Wikipedia. Capture effect. http://en.wikipedia.org/wiki/Capture_effect, 2012.