A Tale of Two Synchronizing Clocks

Abstract

A specific application for wastewater monitoring and actuation, called SwimNet, deployed city-wide in a mid-sized US city, posed some challenges to a time synchronization protocol. The nodes in SwimNet have a low duty cycle (2% in current deployment) and use an external clock, called the Real Time Clock (RTC), for triggering the sleep and the wake-up. The RTC has a very low drift (2 ppm) over the wide range of temperature fluctuations that the SwimNet nodes have, while having a low power consumption (0.66 mW). However, these clocks will still have to be synchronized occasionally during the long lifetime of the SwimNet nodes and this was the problem we confronted with our time synchronization protocol. The RTC to fit within the power and the cost constraints makes the tradeoff of having a coarse time granularity of only 1 second. Therefore, it is not sufficient to synchronize the RTC itself—that would mean a synchronization error of up to 1 second would be possible even with a perfect synchronization protocol. This would be unacceptable for the low duty cycle operation—each node stays awake for only 6 seconds in a 5 minute time window. This was the first of three challenges for time synchronization. The second challenge is that the synchronization has to be extremely fast since ideally the entire network should be synchronized during the 6 second wake-up period. Third, the long range radio used for the metropolitan-scale SwimNet does not make its radio stack software available, as is seen with several other radios for long-range ISM band RF communication. Therefore, a common technique for time synchronization—MAC layer time-stamping—cannot be used. Additionally, MAC layer time-stamping is known to be problematic with high speed radios (even at 250 kbps).

We solve these challenges and design a synchronization protocol called HARMONIA. It has three design innovations. First, it uses the finely granular microcontroller clock to achieve synchronization of the RTC, such that the synchronization error, despite the coarse granularity of the RTC, is in the microsecond range. Second, HARMONIA pipelines the synchronization messages through the network resulting in fast synchronization of the entire network. Third, HARMONIA provides failure handling for transient node and link failures such that the network is not overburdened with synchronization messages and the recovery is done locally. We evaluate HARMONIA on SwimNet nodes and compare the two metrics of synchronization error and synchronization speed with FTSP. It performs slightly worse in the former and significantly better in the latter.

Keywords

Sleep and wake, synchronization, sensor, low duty cycle

1 Introduction

Wireless sensor actuator networks or WSANs consist of computer controlled sensors and actuators that communicate over a wireless (usually RF) communication network. WSAN's use sensed data to power actuators which can then effect the sensed environment. The resulting changes in that environment can then be sensed by the network. This forms a distributed feedback loop that has the potential for efficiently controlling geographically distributed processes at a scale that was previously unthinkable. A metropolitan scale (city wide) WSAN, called SwimNet[1], is currently being built by a partnership of private (Studebaker Inc.), public (City of South Bend), and academic (University of Andromeda) agencies[2]. The WSAN is being built to control the frequency of combined sewer overflow (CSO) events in a mid sized U.S. city (South Bend, Indiana). More than 700 cities in the U.S. have sewer systems that combine sanitary and storm water flows in the same system. During rain storms, wastewater flows can easily overload these combined sewer systems, thereby causing operators to dump the excess water into the nearest river or stream. The discharge is called a combined sewer overflow (CSO) event [11]. The problem addressed by SwimNet represents a major public health and environmental issue faced by many U.S. cities. At present, the system consists of 150 wireless sensor nodes monitoring 111 loca-

[1]The name is changed from the actual name to preserve anonymity.

[2]Names of the private and the academic institutions are fictional to preserve anonymity.

tions in the South Bend sewer system. Actuation nodes are scheduled to be completed in summer 2009.

The SwimNet deploys nodes in the sewage channels for sensing, on top of traffic poles for relaying, and at major traffic intersections to act as gateways to the cellular network, using which the sensed data is uploaded to a backend server. The nodes are called Chasqui nodes, which are based on the Crossbow mote design, but expand on it to add a longer range and faster radio, and significant to our problem, a Real Time Clock (RTC) with an extremely low drift of 2 ppm. The Chasqui nodes are meant for long-term operation without the need to change batteries. Therefore, a natural design point is to have low duty cycle operation of the network. In the current deployment, each node stays awake for 6 seconds in a 5 minute period, leading to a 2% duty cycle. This led us to the requirement of accurate time synchronization for the Chasqui nodes.

The distinctive challenges for synchronization in SwimNet were three-fold. First, the synchronization had to be fast since the network only stayed awake for 6 seconds at a time and the projected scale of the network is large, of the order of a few hundred nodes. Second, the Chasqui nodes used the RTC, which is external to the microcontroller chip, for the trigger for wake up. This is due to the RTC's low drift over the large temperature range to which the nodes are exposed—from -13°F to 122°F. However, crystals used for clocks have a tradeoff in three dimensions—drift, granularity, and power consumption. The power consumption has also to be kept very low and hence the RTC sacrifices the granularity that is exposed to the programmer—it has a coarse granularity of only 1 second. Thus, we have the situation that wake up is controlled by a clock whose granularity is so low that it is not sufficient to synchronize the clock, given that the duty cycle is low. Third, the high power, long range, and high speed radio used is a MaxStream 115.2 kbps radio where the firmware is not available for modification. Thus, we cannot use a common technique used in time synchronization protocols—MAC-layer time-stamping. Additionally, MAC layer time-stamping with high speed radios poses problems as documented in [15, 14]. While we have posed these challenges in the context of SwimNet, we believe they are more general than that. Abstracting out the details, these challenges to a time synchronization protocol will be posed by any WSAN that has large scale, low duty cycle operation, proprietary radio stack, and crystals that make a natural tradeoff between drift, granularity, and power consumption.

We found that no existing time synchronization protocol addressed these challenges motivating us to design and develop our protocol called HARMONIA. HARMONIA is designed and implemented in TinyOS and executes on the Chasqui nodes. It has *three primary design innovations*. *First*, it has an algorithm to use the high resolution microcontroller clock to synchronize the low resolution RTC. *Second*, the synchronization-related hand-shake between two adjacent nodes happens in two rounds through a single message in each round. However, HARMONIA pipelines the two rounds, with a node acting as a source of the first round message before it has itself received the second round message. This design is important in achieving a rapid synchronization of the entire network. *Third*, reliability is built into HARMONIA to handle transient node and link failures. The goal is to localize the effect of a failure and not overburden the network with synchronization-related messages.

To evaluate HARMONIA, we create small-scale linear and tree topologies with Chasqui nodes, with each node running the SwimNet application and having a low 2% duty cycle. We evaluate the time to synchronize the network and the synchronization error between any two pair of nodes. We compare this to FTSP running on Mica2 nodes. While a comparative evaluation on the same hardware platform would have been desirable, each protocol relies critically on some specific hardware feature. The results validate our design goal that HARMONIA is faster than FTSP, while sacrificing synchronization error. A representative result is that HARMONIA is 8.7X and 12.1X faster than FTSP for a 5 hop linear network depending on the setting of FTSP, and with a period of 300 ms for synchronization messages. The average one-hop synchronization error of FTSP is only 1.5 μs, while that of HARMONIA is 16.77 μs.

Next, we describe our target system. In Section 3, we motivate why we need a new synchronization protocol. Then we describe the design of HARMONIA. In Section 5, we present the experiments and results. Then we provide a discussion of extensions and issues with HARMONIA, followed by a survey of related work. Finally, we conclude the paper with an outline of ongoing work.

2 SwimNet

2.1 SwimNet Architecture

SwimNet's architecture was designed to be a set of local WSAN's that connect to an existing wide area network (WAN) through gateway devices. SwimNet can therefore be viewed as a heterogeneous sensor-actuator network. It consists of four types of devices: 1. Instrumentation Node or INode: these nodes are responsible for retrieving the measurement of a given environmental variable, processing that data and forwarding the data to the destination gateway through a radio transceiver. 2. Relay Node or RNode: these nodes aid in forwarding data collected by INodes that are more than one-hop away from the gateway node. The RNOdes only serve to enhance the connectivity in the wireless network. 3. Gateway Node or GNode: these nodes serve as gateways between the WSAN used to gather data from the INodes and a Wide Area Network (WAN) which allows remote users easy access to SwimNet's data. 4. Actuator Node or ANode: these nodes are connected to valves (actuators) that are used to hold back water in the sewer system.

To appreciate the challenges posed to a synchronization protocol, we first need to describe the system that is controlled by SwimNet. Figure 1 shows a sewer system in which combined sewer trunk lines (sanitation and storm water flows) feed into a large *interceptor sewer*. Prior to 1974, municipal combined sewer lines dumped directly into rivers and streams. Under the Clean Water Act, cities were forced to treat the water from these combined sewer lines before they were released into a river or stream. One common way to meet this regulatory burden was to build an interceptor

sewer along the river. This sewer would intercept the flow from the combined sewer trunk lines and convey that flow to a wastewater treatment plant (WWTP). Under dry weather conditions the flows were small enough to be handled by the WWTP. Under wet weather conditions (storms), the flows often overwhelmed the WWTP's capacity, thereby forcing operators to dump the excess directly into the river or stream. Such discharges constitute the CSO events described earlier.

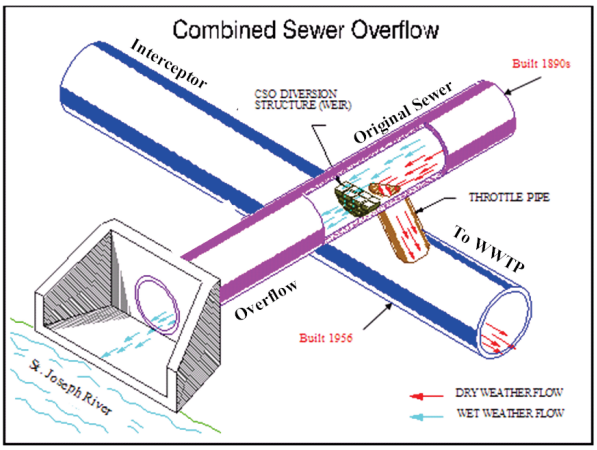


Figure 1. South Bend Interceptor Sewer and CSO Diversion Structure.

From Figure 1 we can see that the combined sewer trunk lines and interceptor sewer connect at a *CSO diversion structure*. This is the point where we can apply control. This means that the natural place to place ANodes is at the CSO diversion points. These ANodes would then adjust the amount of water diverted into the interceptor sewer line based on an adaptive threshold that is a function of the current flows into the system. The GNode serves as a gateway between this particular WSAN and neighboring WSAN's up and down the interceptor line. Figure 2 illustrates this system architecture with 2 different WSAN's controlling the two diversion structures into the interceptor line. GNodes at these diversion structures and the WWTP are used to exchange control information in a way that allows coordinated flow control across the city's entire sewer system.

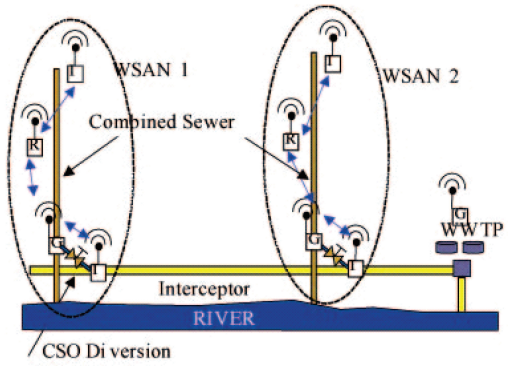


Figure 2. SwimNet's Hierarchical Architecture.

2.2 SwimNet Hardware

The basic building block of SwimNet's WSAN is a more rugged version of the MICA2 processor module called the Chasqui wireless sensor node. The Chasqui node started with the original embedded node designs developed by U.C. Berkeley. Studebaker Inc. enhanced the radio subsystem and sensor/actuator interface subsystems of this earlier design. The Chasqui node uses a 115 kbps MaxStream radio operating at 900 MHz. It uses frequency hopping spread spectrum (FHSS) signaling to reduce the radio's sensitivity to interference. The radio has a larger maximum transmission power (1 watt) than the conventional Chipcon radio. Consequently, the Chasqui node has a range of over 700 meters in urban environments and up to a 5 km range for line-of-sight connections. The longer range of the Chasqui processor fits well with the distances required by the SwimNet application. The MAC layer of the radio is implemented in proprietary firmware that is closed source. However, a feature significant to our synchronization protocol, is that the radio sends a signal a fixed offset time after the first bit being sent out on the wireless channel and also a signal when the first bit is received from the wireless channel. This signal is used to trigger an interrupt followed by executing part of HARMONIA's algorithm.

To give a sense of the deployment for which our HARMONIA is targeted, we provide in Figure 3 an overlaid map view of the largest of the 36 CSO areas in South Bend, which covers an area of 3758 acres. It has 7 RNodes, 3 INodes, 2 GNodes and 1 ANode, that controls an automated valve at the basin. Notice that the network of Rnodes is almost linear. Due to the requirements of the application that the network needs to span a large geographical area, the Rnodes provide relaying functionality, and the radio has a long range, the network in most parts is almost linear. This is a driver for some design decisions in HARMONIA, which we will discuss in Section 6.

In spite of the higher transmission power required by the MaxStream module, careful design of the SwimNet middleware and hardware allows the WSAN's based on the Chasqui node to operate for several years before changing batteries. The Chasqui node consumes up to 5W when fully active and drops down to 0.14 mW in sleep mode. Long battery life can be effectively achieved by using low duty cycles. All the nodes in SwimNet wake up at the beginning of every T_w seconds defined a slot, and stay awake only for the first T_a seconds of each slot based on the RTC. Here, T_a is much smaller than T_w to save battery power. In current deployment, those are set to $T_a = 6$ seconds and $T_w = 300$ seconds, resiting in 2% duty cycle.

These values are possible due to the nature of the phenomenon that the WSAN is meant to monitor—such events last for more than 5 minutes. The biggest limitation to efficient communication in low duty cycle systems is precise synchronization. Typical crystal tolerances such as the one used in the Mica2 platform are on the order of 40 ppm yielding drifts of up to 3.456 seconds per day. Extreme temperature differentials can be seen in the SwimNet application: nodes inside the sewer system are at a relatively constant temperature of around 10°C year round while nodes mounted

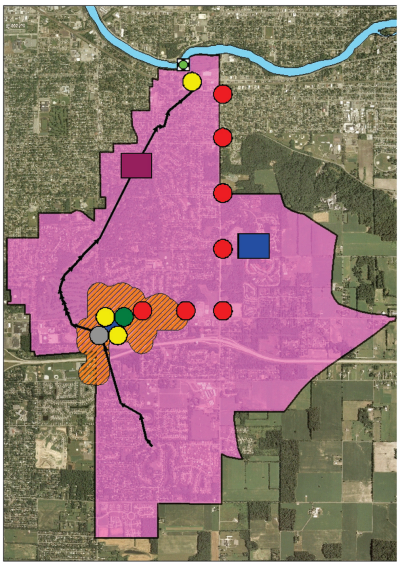


Figure 3. Overlaid map view of the largest of the 36 CSO areas in South Bend. It shows the four different kinds of nodes - Instrumentation node (INode in yellow), Relay node (RNode in red), Gateway node (GNode in green), and Actuator node (Anode in gray). The blue box is a unit with a RNode and a GNode.

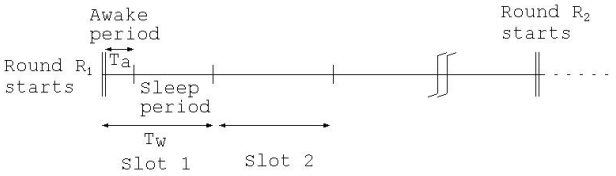


Figure 4. The duty cycle of a Chasqui node showing the awake period (T_a = 6 seconds in the deployment) and the sleep period, which together constitute a slot (T_w = 5 minutes in the deployment). The beginning of a round is marked by the BS initiating a new synchronization process.

on traffic poles can experience temperatures ranging between -20°C to 50°C. Experiments at these temperatures showed drifts of up to 3 seconds per day using regular crystals. While synchronization algorithms can periodically reset the drift error between nodes, they also consume precious energy resources. Therefore, the Chasqui node uses a precision real time clock (RTC) [6]. Using this, the nodes can coordinate their active and sleep cycles with sufficient precision to reliably function at a 2% duty cycle. The Chasqui node implements a precision RTC with a typical drift of only 2 ppm giving SwimNet tight synchronism between synchronization updates. Experiments showed that the Chasqui nodes can reliably function with periodic synchronization updates in HARMONIA every 7 days. With such a duty cycle, the SwimNet applications based on the Chasqui processor node have a service life in excess of three years with a 4 cell lithium battery pack.

3 Can FTSP be Used to Synchronize SwimNet?

The FTSP protocol [5] represents the state-of-the-art in synchronization protocols and compensates for most sources of time variability, thus achieving highly accurate synchronization. It does not rely on any network topology. A root is elected, based on node IDs, and it initiates the synchronization by periodically broadcasting a synchronization message. After some initial startup time when the caches are being populated, each node periodically broadcasts to its neighbors its local estimate of the time at the root node. FTSP uses a single broadcast message, rather than a two-way handshake, to establish synchronization points between the sender and the receiver. FTSP's design eliminates many sources of synchronization error, notably the interrupt handling time and the encoding/decoding time. It also uses MAC-layer timestamping. Each node uses a linear regression table to estimate the offsets between the local clock and that of the root node. The performance of the protocol—the synchronization error and the time to synchronize the network—is dependent on the number of points that are used to create the regression line. This technique enables each node to estimate its drift with respect to another node and compensate for it.

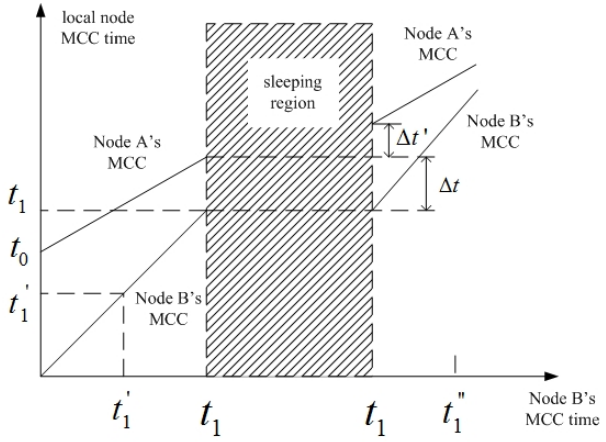
If we say that the synchronization packet flooding period is P, the number of points needed to draw the regression line is k (k = 8 by default), and the maximum number of hops in the network from the root is N, then FTSP takes approximately kPN time to synchronize the whole network [5]. This is because only after a node finishes the linear regression by receiving the k synchronization packets, it can start to flood the estimate of the global time through a local broadcast. Moreover, if the root fails and a new root needs to be reelected, this takes $PN/2$ time on an average. This is for the average case where the new root is at a distance $N/2$ from the old root. Such time requirements of the FTSP make it challenging to apply it to SwimNet synchronization since nodes in the SwimNet stays awake only for 6 seconds every wakeup and many parts of the network are in effect connected in a linear topology. For example, even if we set the value of P quite short (compared to values used in the experiments in [5]) as $P = 300$ms and sacrifice the performance of linear regression by taking the minimum two points, we can synchronize at most a 20-hop network from the root within the 6 seconds. Practically this number will be smaller because nodes can communicate with each other for even less than 6 seconds due to the drift in RTC when a synchronization protocol is initiated.

Let us consider two straw man proposals to adapt FTSP to our problem. First, we use FTSP to synchronize the microcontroller clock (MCC) since it has a fine granularity (0.125 μs for the Mica2) and can benefit from the small synchronization error achievable with FTSP. However, the MCC does not run during the time the Chasqui node is asleep and the sleep-wake is guided by the RTC. Therefore, synchronizing the MCC will not serve our purpose.

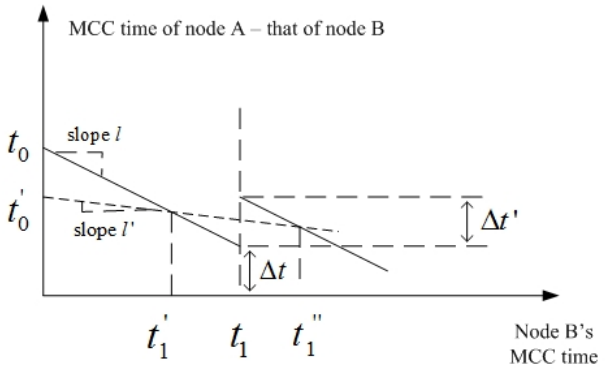
Second (and alternately to the first), we synchronize the RTC since the RTC continues to tick through the microcontroller's sleep period. Then we can relax the requirement that the entire network needs to be synchronized within the

awake period of one slot. Rather the synchronization packets needed for regression can be collected over multiple slots and the RTC synchronized with them. However, the RTC has a coarse granularity of only 1 sec and therefore, despite the small synchronization error of FTSP, the clock may differ by up to 1 sec. This would be unsuitable for the low duty cycle SwimNet.

The two straw man proposals suggest the approach that we take in design of HARMONIA. The approach simply put is to synchronize the MCC first and then change the RTC to a globally determined value at the same time based on the synchronized MCC. This achieves both finely granular value for the time used in synchronization algorithm and synchronism of RTC for sleep-wake. Therefore, applying FTSP to this model boils down to first synchronizing the MCC using FTSP and then adjusting the RTC based on this synchronized MCC. However, this runs in to the slow network-wide synchronization problem of the FTSP explained at the beginning of this section. This argument crucially depends on the following observation—the synchronization of the MCC has to happen within *one awake period of one slot*. This cannot be staggered over multiple slots.



(a) MCC drift between a pair of nodes.



(b) MCC difference between the pair.

Figure 5. FTSP's problem with linear regression when working with sleep-wake operation.

The reason is explained by Figure 5. Consider that node B is trying to synchronize itself to the clock of node A. In Figure 5(a), we see two lines one corresponds to node B's MCC measured with respect to node B's MCC — obviously this is a 45° line from the origin; the second corresponds to node A's MCC again measured with respect to node B's MCC. The two clocks have different frequencies and hence the difference in slope between the two lines. Node A's clock also has an offset—time t_0 in the figure. In Figure 5(b), we see the MCC difference between A and B with respect to node B's MCC. Ideally, node B should be able to estimate the difference in drift between A's MCC and its own MCC. Thus, in Figure 5(b), it should be able to estimate the slope l. According to FTSP, if FTSP had completed within the wake period, it would indeed have been able to estimate the slope. However, since the synchronization does not complete within the wake period, node B hits against the onset of sleep, time t_1 in Figure 5(a). At this time the offset that A's MCC has over B's MCC is Δt. However, nodes A and B wake up after their sleep based on a trigger from their respective RTCs. The RTCs also have different frequencies. Therefore, nodes A and B wake up at slightly different times, say node A wakes up before node B. Then the offset at node B's MCC time t_1 suddenly jumps from Δt to $\Delta t + \Delta t\prime$. In other words, the curve in Figure 5(b) has a discontinuity at time t_1. Now, consider what happens if node B had staggered its regression points across the two awake periods. Node B would then have estimated, using FTSP, that the slope of the relative MCC difference is $l\prime$ (Figure 5(b)), rather than the correct slope of l. There is no fixed relation between $l\prime$ and l—it depends on the arbitrary order and difference in time between the wake-up of nodes A and B.

The nub of the argument then is that the linear regression should be finished within each awake period. For networks of the size of SwimNet, FTSP out-of-the-box cannot achieve this as we will show in the experiments section. Hence, the need for a new synchronization protocol, hence HARMONIA.

4 Proposed Protocol

4.1 Operational Scenario

The SwimNet is connected for data dissemination and collection in a tree topology whose root is a base station (BS). The topology is created by stateless gradient-based routing [7]. Each node in the network has a gradient number that is an indication of how close the node is to the destination. Since there might be several destinations, each node stores one gradient number per destination in the network. HARMONIA will also use the tree topology.

Recollect that all nodes in the SwimNet wake up at the beginning of every T_w seconds defined as a slot, and stay awake only for the first T_a seconds of each slot. The BS initiates the synchronization procedure in certain slots. We say a new *round* of synchronization is started when that happens. The BS may decide when to initiate this based on a fixed period, for example, through calculation of the worst case drift of the RTC, or some indication that the network has gone out of synch, for example, inferring from a drop in the received data rate.

We first provide a conceptual view of how HARMONIA works, hiding the technical details. Note that there are two clocks in the picture - a microcontroller clock (MCC) and the

Real Time Clock (RTC). The MCC has a high drift but high resolution, and it also does not tick when the node is sleeping. The RTC has a low drift but low resolution, low enough that synchronizing the RTC alone will have a large synchronization error (up to 1 sec) and thus will not be sufficient for our requirements. Our goal is to synchronize RTCs of all nodes accurately enough to ensure all nodes in the network wake up at the same time.

The BS initiates the synchronization once a certain time has elapsed since waking up. Synchronization happens in cascaded stages where the synchronization proceeds along the tree topology with the BS acting as the root node. The interaction between a node and its children happens in two phases. A pipelining effect is achieved between multiple levels of the tree by having a node perform the first phase of the synchronization with its children even though it has not *completed* its own synchronization, i.e., it is yet to complete its second phase. After a node has received the two phases from its parent, a node is considered synchronized with respect to its parent. It then sets an alarm using its MCC. The drift in the MCC during this alarm interval contributes to the synchronization error in HARMONIA, in addition to other factors. The synchronization achieves the effect that the alarms of all the nodes in the network will go off at the same time, modulo the synchronization error. When the alarm goes off, a node sets its RTC's second hand to a value determined by globally known parameters. Since all the nodes do this at the same time and since sleep-wake happens according to the RTC value, this implies that the entire network is synchronized for its sleep-wake.

Once the BS decides to synchronize a network, it begins the protocol T_s seconds after it wakes up as depicted in Figure 6. The value of T_s should be chosen to ensure that all the children of the BS have already woken up so as not to miss any synchronization-related messages from the BS. In addition, since all nodes adjust their RTC at $T_{alarm} = T_s + T_{interval}$ they have to stay awake until the RTC is adjusted. Thus the values for T_s and $T_{interval}$ must be taken to satisfy the following conditions:

$$T_s > T_d \qquad \text{and} \qquad T_s + T_{interval} < T_a - T_d, \qquad (1)$$

where T_d denotes the maximum offest in RTC that has built up between a parent and a child node since the previous sychornization. However the second condition is not as critical if we enforce the design that a node delays going to sleep till its alarm has expired. Additionally, the value $T_{interval}$ used at the BS should be large enough that all the nodes in the network have gone through both synchronization phases and are ready to set their RTCs. However, there is a desire to keep $T_{interval}$ small since the drift in the MCC during this interval contributes to the synchronization error.

4.2 Synchronization Protocol

Our goal is to synchronize RTCs of all nodes to ensure all nodes in the network wake up at the same time. However, since RTC has only 1-second resolution, if we adjust any node's RTC on the basis of another node's, there could be at worst a 1-second synchronization error between the two nodes. In order to reduce this kind of uncontrollable synchronization error, we adopt another timer in our protocol, which uses a MCC provided by the Atmel Atmega128L, a microcontoller used in Chasqui motes. The MCC provides much finer resolution than the RTC, operating at the frequency of 8MHz. However it cannot be used directly as a system clock since it does not run when a node is sleeping. Therefore the core part of HARMONIA is about how to use the MCC to set the RTC to the same value, at the same time. Here "same time" must be defined within a high resolution, identical to that of the MCC. From now on, the value of the MCC is expressed using lowercase t not to be confused with the value of the RTC, which is being expressed using uppercase T.

When the BS initiates the protocol, it sets an alarm to go off after $T_{interval}$. To achieve this, it sets the MCC timer that goes off at t_{alarm}. For example, for $T_{interval} = 2$s and for a 8 MHz MCC, it will set the timer to expire after 16×10^6 ticks. When the alarm fires, the RTC's second hand is set to the value $T_{alarm} = T_s + T_{interval}$. Right after setting the alarm, the BS gets to be the first to do the following two-phase message transmission. This is repeated recursively by each node with its children through the network.

Phase 1**:** SYNC packet transmission and reception
- Transmission: A parent sends to its children a syncrhonziation initiator packet called SYNC that carries t_{alarm}. The parent records t_p the local time at which its radio chip starts to transmit the first bit of the SYNC through an antenna.
- Reception: Each child records t_c the local time at which its radio chip starts to receive the first bit of the SYNC.

Phase 2**:** SYNCD packet transmission and reception
- Transmission: A parent sends to its children a synchronization data packet called SYNCD carrying the t_p and t_{dif}, where the t_{dif} is the offset between its MCC and the BS's. For the BS, the t_{dif} is always set to zero.
- Reception: After receiving the SYNCD, each child of the parent updates its t_{dif} as $t_{dif} = t_{dif}^{rcv} + t_c - t_p$ (the "rcv" indicates it is the value received by the node), and sets an alarm that goes off at $(t_{alarm} + t_{dif})$.

Here each SYNC(SYNCD) packet is sent after a backoff time taken randomly from a uniform distribution over $[0, t_{bf}]$, where t_{bf} denotes the maximum backoff time. This is to avoid contention in the synchronization packets among neighbors.

Figure 7 depicts the above two-phase synchroization packet transmissions and receptions performed from the BS to two-level lower hierarchy. Every node in the network becomes aware of t_{alarm}, the time at which the BS expires its alarm by receiving SYNC packet from its parent. However, since all nodes' MCC may not be synchronized, each node needs to figure out the offset in the MCC between itself and BS to make its alarm go off at the same physical time as at the BS. This is done by the SYNCD packet propagation: When a node receives the SYNCD from its parent, the SYNCD lets it know the offset between the parent and the BS, that is, t_{dif}. Thus the node can calculate the offset between itself and the BS by adding the offset between itself and its parent to the received t_{dif}. It would be obvious from the above description that HARMONIA does not compensate for the difference in drifts in the MCCs or the RTCs of two nodes, nor for the

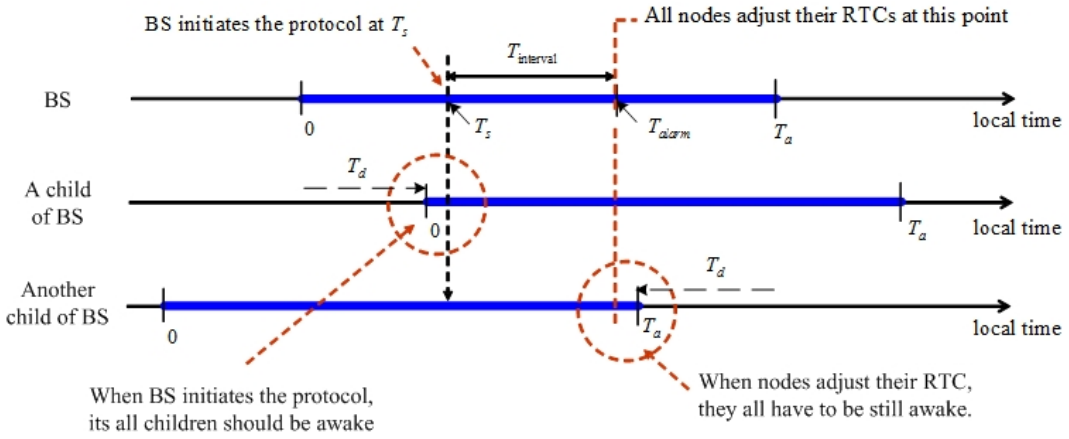


Figure 6. Sleep-wake operation and its relationship to the synchronization protocol.

jitter in the interrupt handling times for the interrupts arising from the MaxStream signals.

Note that the t_p and t_c in the Phase 1 are recorded in a similar way that MAC layer time-stamping technique gets timestamps, but unlike in the MAC layer time-stamping, the value of t_p is transmitted in a different packet—SYNCD, not SYNC. This is because the MaxStream radio MAC firmware is not modifiable and we cannot embed the t_p into the SYNC.

MaxStream Signaling on Bit Transmission and Reception

In our description above, we simplified the issue of signaling from the MaxStream radio to the microcontroller. In reality, what happens is depicted in Figure 8. On the transmitter side, the radio generates a pulse of width T_{TL} and on the receiver side, the radio generates a pulse of width T_{RL}. Trigger to the Chasqui microcontroller happen respectively on the rising edge and the falling edge. There is a time difference between when the event is time-stamped at the transmitter and at the receiver end, since $T_{TL} > T_{RL}$. According to the MaxStream 9XTend OEM RF Module specification [1], this difference is 190 μs, while experimentally we found this to be 250 μs. We capture this using a parameter t_{con} and thus t_{dif} in phase 2 reception is updated as $t_{dif} = t_{dif}^{rcv} + t_c - t_p + t_{con}$. We explain in Section 5 how t_{con} is experimentally measured.

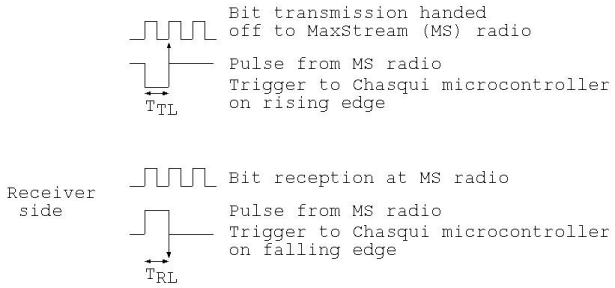


Figure 8. The signaling from the MaxStream radio to the microcontroller. The signal on the transmit and on the receive side are used to take timestamps which are used in HARMONIA.

4.3 Failure Handling

In this section, we discuss how HARMONIA can handle transient failures in either links or nodes. A node needs to detect the loss of any synchronization packet. For this it uses overhearing of its child's synchronization packet as an implicit acknowledgement (ACK).

After a node sends SYNC to its children, it sets a timer which goes off after t_{out} time within which it expects to overhear all its children sending SYNC to their own children. If the node does not overhear the SYNC packet(s) from one or more children, this is taken as an indication of failure and it sends the SYNC again. The protocol has a bound N_{max} for which the process will be tried, within a slot, before declaring failure.

A parent node begins sending the SYNCD packet to its children only after it has been assured that all its children have received the Sync packet, or that there has been a failure. The same technique is used by the node to detect and to handle failure in SYNCD.

4.4 Packet Sequences and State Management

In HARMONIA, since retransmissions can occur, we need a way to allow the nodes which have already received a packet to disregard the same type of packet subsequently. Practically those state variables are managed in the following manner. Every node has two different kinds of round sequences: One is round sequence as a parent denoted by N_{rp} and the other is round sequence as a child which is N_{rc}. Those are both initially set to zero. Whenever BS initiates a new round of synchronization procedure, it increases the N_{rp} by 1. All SYNC and SYNCD packets carry the node's N_{rp} value. A child accepts a synchronization packet with N_{rp} greater than or equal to its current N_{rc} if the packet is from its parent. When a node receives a N_{rp} value from its parent, it updates its own N_{rp} value to be the received one. It updates its N_{rc} value as $N_{rc} = N_{rp} + 1$ after it sends SYNCD. This will help the node disregard re-sends of the SYNC from its parent.

In addition to the round sequences, the SYNC packet should carry another type of sequence number for the ARQ operation denoted by N_{trial} which represents how many times

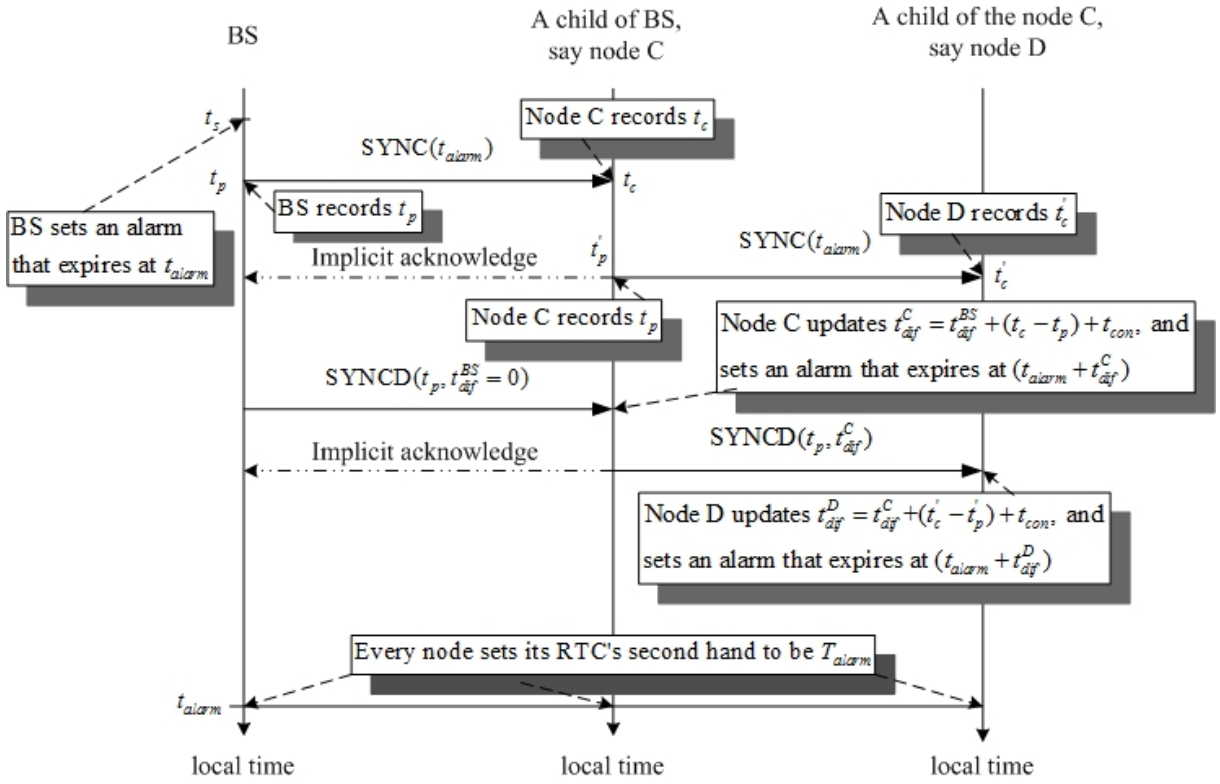


Figure 7. Illustration of the synchronization protocol.

the SYNC transmission has been tried so far including the current one. A parent does not know at what value of N_{trial} the SYNC packet will be received at each child, Therefore, it has to record all the instants at which the SYNC is sent with the corresponding value of N_{trial}, and send all these information in the SYNCD packet. Each child remembers the value of N_{trial} in the SYNC it received, and finds the correspoding time of sending the SYNC when it receives the SYNCD. It uses this time to calculate t_{dif} in reception step of the Phase 2. For example, if a node A had to send two SYNCs to satisfy its two children C_1 and C_2. The times corresponding to these two sends are t_{p1} and t_{p2}. Then when A sends the SYNCD, it has fields: *Trial 1*: t_{p1}; *Trial 2*: t_{p2}.

4.5 Fast Recovery

In spite of trying N_{max} number of times within a slot, a node may be unable to synchronize all its children. For this case, we introduce the feature of fast recovery. A node can then proactively initiate fast recovery for synchronizing its descendant sub-tree in the next slot, and does not have to wait for the BS to initiate the next synchronization round. Let us say a node A is in this situation. Let us consider one of its child nodes C_1. This situation can happen because of any of the three reasons: (i) C_1 did not even receive the SYNC; (ii) C_1 received the SYNC but for some reason did not finish getting synchronized in the slot; (iii) C_1 is synchronized, but the implicit acknowledgment has been lost to A, or A is trying to synchronize a sibling of C_1. For case (i), no special treatment of the state variables N_{rp} or N_{rc} is needed since these had not been incremented (the SYNC was not even received). For case (ii), C_1 decrements its N_{rc} before going to sleep so that it will accept the SYNC in the next slot. For case (iii), C_1 disregards the synchronization message and sends an explicit acknowledgment to A by sending a message called SYNCA. A node tries fast recovery for a maximum $N_{maxtrial}$ times.

The fast recovery concept is powerful enough to handle the situation that a large network cannot all be synchronized in one slot. Rather the synchronization proceeds with as much of the network being synchronized initially as possible, and the unsynchronized parts of the network being handled through fast recovery.

4.6 Choice of Important Parameters

Here we discuss the tradeoffs in choosing the most important parameters in HARMONIA.

1. T_s: This is the time the BS waits after waking up to initiate the synchronization messages. This value has to be large enough to accommodate clock drifts that have built up between a parent and its child node. This is to ensure that the child node is awake to receive the synchronization message. But, it must be small enough that the synchronization can complete in the awake period of one slot. We find for the SwimNet a value of 2 s is reasonable.
2. N_{max}: This is the maximum number of times a node tries to synchronize its children nodes within a slot. A larger value will increase the reliability of the synchronization process, within one slot. However, it cannot be so large that the node arrives at the time to sleep within the slot before it has exhausted all N_{max} tries. Also there is a resource consumption that goes up with increasing values of N_{max}. This depends

upon the frequency of transient failures in the network. We find a value of 3 works well for us.

3. $T_{interval}$: This is the time after which an alarm will be triggered to set the RTC, all together all through the network. This value should be large enough to give time for the entire network to be synchronized. But, the drift in the MCC in this time contributes to the synchronization error; therefore, it should be kept small. The value will depend on the scale of the network and we should set it to the smallest possible value that meets the above condition.

4. t_{bf},t_{out}: The first is the backoff before sending a SYNC or SYNCD, the second is the time between the two phases. We have the condition $t_{out} > t_{bf} + t_{proc}$, where t_{proc} is the small time used in processing the synchronization message. This condition is required since otherwise a node may mistake that its SYNC message to its child has been lost when in reality the child was backing off before sending it along. The parameter t_{bf} should be chosen based on the network density, a higher density requiring a larger value. The smaller the value of t_{bf} is, the faster will be the synchronization time of HARMONIA. For our case with a network density of 6 neighbors, we find $t_{bf} = 100ms$ does not cause appreciable collisions. However, we are yet to do thorough experimentation to determine its setting.

5 Experiments

5.1 Experimental Methodology

We tested HARMONIA focusing on network-wide synchronization time and synchronization error with three different network topologies shown in Figure 9. However, in our experiments, in all three topologies, the nodes are actually placed within a short distance. This is to make the experiments feasible from a logistic standpoint. Therefore, we use software topology control to define the neighbor relations between the nodes. Thus, if node i is not connected to node j in the topology, it disregards all packets it receives from j and vice-versa. Note that this still causes contention that would not be present in the actual network.

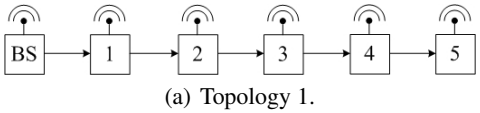
Metrics

We define the *synchronization error* for a node as the difference in its estimate of the BS's local time from the actual local time at the BS. For HARMONIA, synchronization error for node i corresponds to the difference in time when the BS and the node adjusts its RTC. We measure this error right after i has been synchronized. For FTSP, a polling node queries the network nodes with a fixed period (3 s in our experiments). On being polled, a node i responds with its estimate of the BS's local time and at that instant the BS's own local time is also measured. The difference gives the synchronization error. Thus, for FTSP, there can be a delay of up to 3 s from the synchronization to the measurement.

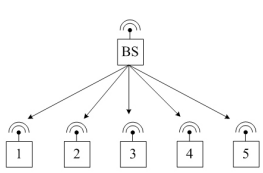
We define the network-wide *synchronization time* as the time from when a round of the synchronization protocol begins to when all nodes in the network get all the packets required to make an estimate of the BS's local time and then have finished the processing of the packets. In HARMONIA, the time ends when the last node has received SYNCD and done the processing (update its t_{dif}) based on SYNCD.

For the experiments with HARMONIA, the microcontroller is programmed to generate a rectangular pulse at Pin 7 and Pin 10 on the Chasqui board shown in Figure 10 at the instants when we have to pinpoint to calculate the synchronization error and the network-wide synchronization time. These two pins are connected to an oscilloscope. Specifically, a node generates the pulse at Pin 7 when it receives SYNCD for the first time in a round of synchronization procedure and has completed the attendant processing. A node generates a pulse at Pin 10 when it adjusts its RTC to get synchrony back. In case of BS, it generates a pulse at Pin 7 whenever it initiates a round of the synchronization procedure. Therefore, the synchronization error between a pair of nodes is the time gap in the rising edge of the pulse generated at Pin 10 and the network-wide synchronization time is measured by taking the time gap at Pin 7 between the BS and the last node to generate the pulse.

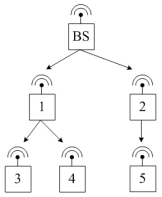
All the experimental results are statistics calculated from at least 10 points—in many cases, it is more; the 10 runs are used when experimental errors caused us to reject other runs. We have run experiments for HARMONIA for four different values for t_{out} ($t_{out} = 150, 200, 250$, and 300 (ms)) choosing other parameters as in Table 1. Regarding how to measure the value of t_{con}, we need to think about what the potential sources are for the synchronization error in HARMONIA: (i) the propagation delay; (ii) the frequency difference in MCC of each node, accumulated between the time the alarm is set to when the alarm fires, and (iii) the handling time for the interrupts that the radio chip signals to record t_p and t_c with the SYNC packet. For this calculation, we neglect (i) and (iii) because those factors are normally in a range of a few microseconds. Let us use the uncorrected equation for synchronization: $t_{dif} = t_{dif} + t_c - t_p$. Then, the absolute value of the synchronization error between a sender (node i) and a receiver (node j) E can be expressed as $E = t_{con} + F$, where F is the error due to (ii). We then measure the absolute value E' of the synchronization error with node j as sender and i as receiver. Then $E' = t_{con} - F$. Hence, we can obtain t_{con} as $t_{con} = (E + E')/2$. Averaging over a number of experiments, we select t_{con} as 250 μs.



(a) Topology 1.



(b) Topology 2.



(c) Topology 3.

Figure 9. Network topologies used for our experiments.

5.2 Network-wide Synchronization Time

Our main objective is to synchronize a network of sensor nodes running on a very low duty-cycle *quickly*—within the

Table 1. Values of parameters in HARMONIA used in the experiments.

T_a	T_w	T_s	T_{alarm}	t_{bf}	t_{con}	N_{max}
6s	5min	2s	4s	100ms	250μs	3

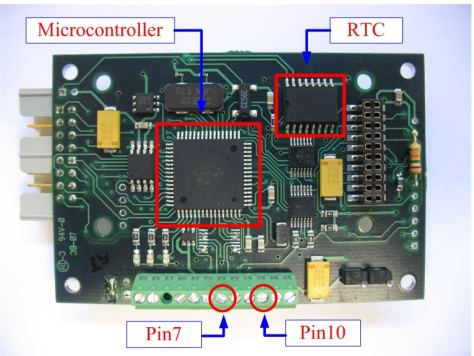


Figure 10. Chasqui board.

time period for which they remain awake—keeping the synchronization error among the nodes within a tolerable limit. Hence synchronization time is the primary metric for us.

For the experiment with HARMONIA, we vary the time between a SYNC and a SYNCD message, denoted as t_{gap}. This time is given by a time-out at the sender side (t_{out}) followed by a back-off at the sender side (chosen in a random uniform manner from $[0, t_{bf}]$). Therefore, the expected value of $t_{gap} = t_{out} + t_{bf}/2$. This calculation of t_{gap} assumes there is no retransmission. In our experiments, there are collisions and retransmissions, and the synchronization time value for HARMONIA is measured in the presence of such events. It is only that the average value of t_{gap} would be higher in that case from what is plotted.

Figure 11 shows that HARMONIA can synchronize the three networks within several hundred microseconds for all the chosen parameters. Since HARMONIA pipelines the SYNC and SYNCD transmissions, the network-wide synchronization time is kept small. Thus, a node does not have to finish getting synchronized before it can act as a source of synchronization messages. When there is no retransmission, increment in the network-wide synchronization time at each hop is due to the backoff, not the timeout. The total synchronization time can be modeled as $c + h \times b$, where c is the constant cost due to the timeout at the BS and b is the variable cost which depends on the back-off and is multiplied by the number of hops h. We can see from the figure that even considering retransmissions, which occur in the experiments, the synchronization time increases quite slowly with hops.

We ran some testbed experiments using Mica2 motes for the topologies shown in Figure 9 using FTSP to see if it can achieve our goal and also to compare FTSP with HARMONIA. In FTSP, each node periodically broadcasts the synchronization packet (say with a period P) containing the MAC layer time-stamp of the instant when the packet is sent. A node needs to receive N_R (8 by default) number of such packets to apply linear regression (to account for the clock drift) and get synchronized with the *root* node. Since the network synchronization time, say T_N, is directly proportional to P and N_R, we reduced the values of these parameters as much as we could to see how fast FTSP can synchronize the network. For linear regression, N_R has to be at least 2. We found that the TinyOS timer count not fire when we reduced P below 10 ms and therefore the minimum value for which we have the reading is P = 10 ms.

Table 2. Slopes of the linear relationship between network synchronization time and synchronization period observed in our experiments. N_{reg} is the number of regression points used by FTSP.

	Topology-1		Topology-2		Topology-3	
	N_{reg}=2	N_{reg}=8	N_{reg}=2	N_{reg}=8	N_{reg}=2	N_{reg}=8
slope	0.005442	0.011031	0.001486	0.005951	0.003056	0.005194
y-intercept	4.003156	4.639860	0.020137	0.086050	1.890372	2.296942

Figure 12 shows the network synchronization time for the three topologies as a function of N_R and P. As expected, T_N decreases as N_R and P are reduced. However, except for one-hop network of topology-2, the network synchronization time is quite large for our purpose because we need to synchronize the network within 6 s when the nodes are awake. Furthermore, these figures also show that T_N increases with the increase in the number of hops in the network. Thus FTSP out-of-the-box would not be suitable for deployment in SwimNet due to its performance in terms of network synchronization time.

Although we do not provide the network synchronization time for larger values of the synchronization period, note that Figure 12 shows that it increases linearly with the synchronization period. The slope of this linear relationship depends upon various factors like network topology, link reliabilities among the nodes, etc. Table 2 shows the slope of these lines along with the y-intercept value using linear regression.

First off, comparison between HARMONIA and FTSP would ideally have been done on the same platform. However, critical features of the protocols are dependent on the features of the specific hardware. Thus, HARMONIA depends on the signals from the MaxStream radio while FTSP depends on MAC layer time-stamping available in the Mica2 radio stack. Nevertheless, we see that the network synchronization time for HARMONIA is of the order of a few seconds in FTSP and it is in the order of a few hundreds of milliseconds in HARMONIA. For example, with topology-1, which most closely resembles SwimNet topology, with t_{gap}=200 ms and equivalently, P=200 ms, FTSP is 7.4X and 9.8X slower than HARMONIA, for number of regression points 2 and 8 respectively. The improvement of HARMONIA increases with increasing values of the period. The improvement is 8.7X and 12.1X for t_{gap}=P=300 ms. Note that the equivalence between t_{gap} and P is not perfect. In FTSP, P denotes a fixed period; in HARMONIA, t_{gap} is an expected value and this represents the gap between SYNC and SYNCD messages and not a period.

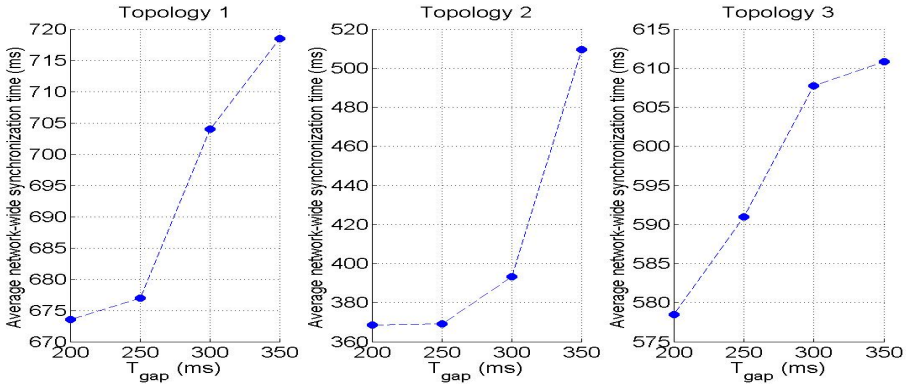


Figure 11. Average network-wide synchronization time of HARMONIA.

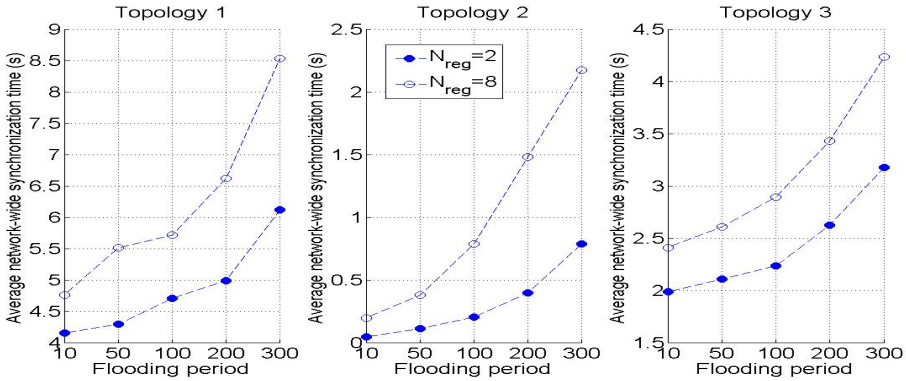


Figure 12. Average network-wide synchronization time of FTSP.

5.3 Synchronization Error

First, we measure the synchronization error in HARMONIA and FTSP in a single-hop network. In this, there are only two nodes. For this, HARMONIA results in an average synchronization error of 16.77μs, while FTSP results in 1.5μs as shown in Table 3.

	HARMONIA	FTSP ($N_{reg} = 8$)
average	16.77μs	1.5μs
max	38μs	3μs

Table 3. One-hop synchronization error.

Thus FTSP outperforms HARMONIA in terms of synchronization error. There are two primary contributory factors. First, we do not compensate for the differential drifts in the MCCs of two nodes. Note however that we are exposed to this effect only during the period $T_{interval}$. Second, we do not account for the jitter in interrupt handling that occurs when the MaxStream radio gives a signal on message transmission and on message reception. However, since the RTC has a high precision oscillator (with a drift of only 2ppm), the synchronization error achieved by HARMONIA still means SwimNet can operate for extended periods of time between synchronizations. A simple computation for this can be formulated as follows. Consider that for a safety margin, we do not want a parent and a child node to go out of synchronization by more than 2 s. Right after synchronization, the two clocks differ by 38 μs (the maximum value from our experiments). Therefore, the nodes can run for $(2 \times 10^6 - 38)/2$ s or 11.5 days before requiring another round of synchronization. Note that for data flow in SwimNet it is important for the parent and its children nodes to be synchronized to one another.

How HARMONIA will work in multi-hop networks can be seen from Figure 13, where the synchronization error at node i is the absolute value of the synchronization error between node i and the BS. We use the default value of t_{out} = 200 ms. The synchronization error decreases from node 1 to node 2. This can be explained by the fact that the synchronization error at node 1 has the opposite sign to the synchronization error between node 1 and node 2. The sign of the synchronization error between a pair of nodes depends on the relative frequencies of the clocks of the two nodes and could be either positive or negative. Thus, the synchronization error in HARMONIA will not continously build up as the number of hops from the BS increases. We can also see from Figure 14 that the time gap between SYNC and SYNCD does not have a strong impact on the synchronization error. This is expected — the synchronization error will go up with $T_{interval}$ and with message load that would cause a higher rate of interrupts at a node. With a really small value of T_{gap}, the second effect could be seen, but this was not observed during the experiments.

From Figures 15 and 16, we see that the synchronization error in FTSP is very small (the results for Topology 3 are

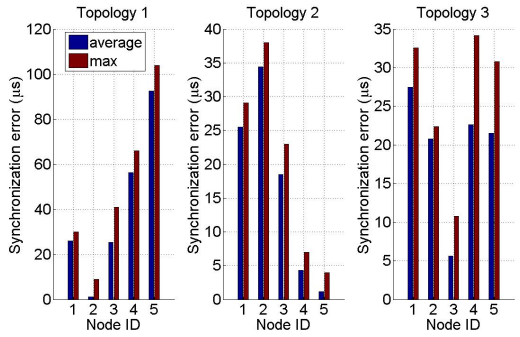


Figure 13. Synchronization error of HARMONIA for the different nodes in Topology 1.

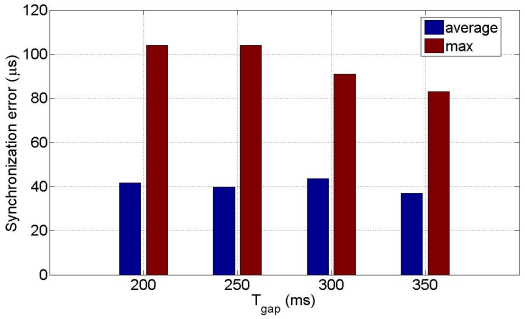


Figure 14. Synchronization errors of HARMONIA with different values of T_{gap} in Topology 1.

omitted since they are similar to that of Topology 2). The error tends to increase when the synchronization messages are sent too quickly (faster than 100 ms) except for the one-hop network (Topology 2). However, the error is always within the tolerable limit for SwimNet. Also as the number of regression points is increased, the synchronization error decreases, as expected.

6 Discussion

On-demand synchronization

HARMONIA can be easily extended to handle on-demand synchronization in which a node requests its parent for initiating synchronization. It sends a SYNC_REQ packet which causes the parent to send the SYNC packet thereby initiating the first phase of the two phase protocol. The child will send the request if it has not been synchronized for greater than some multiple of the duration of a round. This threshold time is such that if the node does not get synchronized then *complete asynchrony* may result, meaning the node's T_a wake period completely misses the wake period of a neighbor. With the on-demand synchronization, the node initiating the request and the sub-tree rooted at that node will be synchronized. This function would be important when a node or a link recovers from a failure and the synchronization process had occurred during the failure duration.

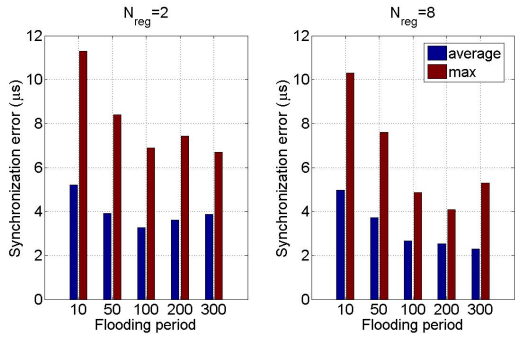


Figure 15. Synchronization error of FTSP in Topology 1.

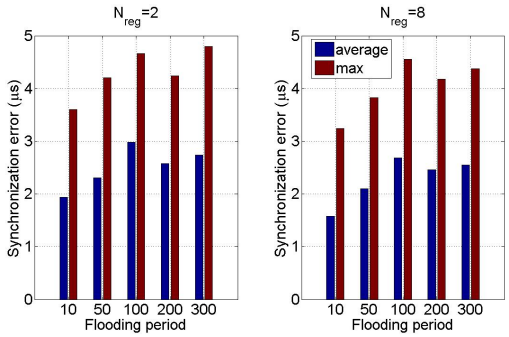


Figure 16. Synchronization error of FTSP in Topology 2.

Issues with MAC-layer time-stamping

MAC-layer time-stamping is quite widely used in synchronization protocols, e.g., TPSN and FTSP. In SwimNet's Chasqui node, MAC-layer time-stamping was not possible due to the proprietary closed-source nature of the MAC protocol. However, even if it had been possible, there are some cautions to using the technique. On the receiving side, as soon as a (synchronization) packet comes in, it is time-stamped at the MAC layer and put in a queue. A queue is required since for fast radios, more than one packet may come in before being consumed by the synchronization protocol. However, the packet itself may be discarded by the receiver if it fails the CRC check. Then, in the absence of identifying information attached to the time-stamp, the receiver has no way of discarding the timestamp that corresponds to the discarded packet. This issue was hinted at in [15] and in subsequent postings on the TinyOS help forum [14].

Synchronization in sparse (almost) linear networks

The SwimNet is almost linear in most parts when we consider the Rnodes as the network nodes. This means that HARMONIA can have a low back-off time since a parent has one or only a few children nodes. However, in our experiments, we have the Chasqui nodes placed on a table close by to each other and use software topology control. However, this increases the likelihood of collisions. Additionally, MaxStream radio typically sends larger-sized packets than the Chipcon radios making the packets more susceptible to collisions. The maximum packet size in MaxStream is 2048 Bytes while in CC2420 it is 128 Bytes. Therefore, in actual deployment, we expect that HARMONIA will have a lower synchronization time since it will incur smaller back-off times.

Reliance on topology

HARMONIA relies on some other middleware service (like

the stateless gradient-based routing in the case of SwimNet) to get the knowledge about the tree structure used for communication. FTSP does not require this knowledge. Although at first glance this may seem as a drawback of HARMONIA, we believe this prerequisite about the knowledge of the topological structure is essential to tradeoff generality for synchronization speed. It is because of this knowledge of the topological structure that a node n_1 can quickly start synchronizing its children after receiving the synchronization packet from its parent. It just needs to backoff a short random time depending upon the number of nodes present at the same depth of the tree as n_1 to prevent collision. Without such knowledge, after a node receives a synchronization packet, it has to conservatively estimate the backoff time or wait for a timer with a sufficiently long interval to fire (as in FTSP) before starting to broadcast its own synchronization packet. Although HARMONIA relies on the knowledge of the tree structure, it works with any such structure as long as it gets this information from some other middleware application. During the network operation, if the tree structure is changed due to node/link failure, HARMONIA will work with the new topology by adjusting the backoff period of a node.

Handling permanent failures

If the external service that creates the topology is run relatively infrequently and a node fails permanently or for a long time, the subtree rooted at the failed node may lose synchrony. However, HARMONIA can be adjusted such that a node does not need to wait for the topology service to reconstruct the tree if its parent has a permanent failure or a failure that persists for a long time. In such a situation, the children of the failed node can select a new parent by broadcasting the *ReqToChangeParent* packet containing the information about the depth of this node in the tree. Since many nodes at different depths of the tree can receive this request, they may concurrently try to be the parent of the requesting node. To avoid this, each node replies to this request if its depth is smaller than that of the requesting node (i.e. if it is higher up in the tree than the requesting node) after a random interval proportional to the difference between its depth and that of the requesting node. This causes the nodes which are in the closest tier *above* the requesting node in the tree to respond to the request first and become the parent. If another node overhears this response, it will suppress its response. This will allow the sub-tree to be synchronized before the topology has been repaired.

7 Related Work

Clock synchronization has long been a subject of study in the wireline universe. NTP and GPS-based receivers are popularly used for synchronization. However, there are significant challenges in applying them to wireless sensor networks, such as high power consumption, accuracy of only milliseconds, and unavailability of synchronization signals indoors. We also need to consider that the hardware clocks on the individual nodes may experience significant drifts. This happens chiefly due to manufacturing variations in the different crystals, the temperature fluctuations the nodes (and consequently the crystals) are exposed to, and aging of the crystals. Tight budget concerns in the design of the sensor nodes rule out the use of the highly accurate oven controlled crystal oscillator (OCXO) or high-end temperature compensated crystal oscillator (TCXO). Also, the multi-hop nature of the sensor network precludes the use of client-server solutions, which most of the solutions from the wireline world fall in.

Therefore, there has been active research in time synchronization in the sensor network community. We refer the reader to [12] for a good coverage of the early work in this field and here we focus on the more recent work. At the high level, HARMONIA is motivated by the unmet need for synchronizing networks that are sleep-wake enabled and that have low duty cycle. The real-world constraints of the Chasqui node introduce the additional challenge for HARMONIA to synchronize a low resolution real time clock. These challenges are orthogonal to those addressed by the existing work that we survey here.

The Timing-sync Protocol for Sensor Networks (TPSN) [2] aims to provide network-wide time synchronization. The TPSN algorithm elects a root node and builds a spanning tree of the network during an initial discovery phase. The synchronization phase proceeds in rounds with the children node in the tree being synchronized to their parents through a two-way message handshake in each round. Each node embeds its local clock's readings in the two-way message handshake and through it the child node can calculate the propagation delay and its clock offset relative to its parent's. TPSN introduced the idea of MAC-layer time-stamping. However, TPSN does not compensate for clock drifts which makes frequent resynchronization necessary. In addition, TPSN requires the two-way handshake to complete between a parent-child pair before the synchronization can propagate further in the network.

The Flooding Time Synchronization Protocol (FTSP) [5] has already covered it in some detail in Section 3.

The Reachback Firefly Algorithm (RFA) for clock synchronization [16] is inspired by the way neurons and fireflies spontaneously synchronize. Each node periodically generates a pulse (message) and observes pulses from other nodes to adjust its own firing phase. RFA only provides synchronicity—nodes agree on the firing phases but do not have a common notion of time. RFA is likely to take a long time to get all the nodes to be firing synchronously and therefore will likely not be suitable for our application.

In [9], the authors propose a way to estimate the drifts in the clocks of two nodes caused by the environment-dependent variations. The authors introduce the notion of a software compensated crystal oscillator (SCXO). In an SCXO, the differential drift between the crystals of two nearby nodes is used to estimate the drift in the crystal of one of the nodes. The solution comprises of a one time calibration phase and a runtime measurement and compensation phase. SCXO achieves mean effective clock stability of 1.6 ppm over a temperature range of -40°C to 75°C. This would allow us to increase the period between synchronizations of SwimNet. The authors provide a practical implementation of the SCXO work in [10] and describe a Crystal Compensated Crystal based Timer (XCXT), a new way of compensating

a pair of crystals which achieves a 1.2ppm precision over a temperature range of -10 to 60°C while using only 1.27mW. The solution relies on a node having two crystal inputs and two timer units (TMote Sky is their demonstration platform). To improve the power consumption the authors describe two approaches. The first is to simply duty cycle one of the crystals. The second approach is to use two crystals, one fast and the other slow. The fast crystal (8 MHz crystal of the MSP430 microcontroller in their demonstration) is used if fine granularity time is needed. The second slower crystal (32 kHz in their demonstration) is used while the system is in sleep. Both crystals compensate for each other's drift and together form a highly stable timer unit. This last hardware design feature, in a context quite different from that of the Chasqui nodes, shares a similarity with the Chasqui design of two clocks. However, this is used by the authors for achieving power savings.

A recent development in the field is gradient based clock synchronization [13]. In this the authors present the design to minimize the clock offset between neighboring nodes. The motivation is that other time synchronization protocols synchronize clocks based on some topology, whether assumed or created as part of the synchronization protocol. Two geographically nearby nodes may be distant in this topology. Therefore, existing protocols, while trying to ensure a small synchronization error globally in the network, may cause the synchronization error in a local neighborhood to be appreciable. Therefore, the authors design the protocol to have very low synchronization error in local neighborhoods. The gradient property of a clock synchronization algorithm requires that the synchronization error between any two nodes is bounded by the distance (uncertainty in the message delay) between the two nodes. For example, Lenzen *et al.* [3] proposed a distributed clock synchronization algorithm guaranteeing clock skew O(logD) between neighboring nodes while the global skew between any two nodes is bounded by O(D). In GTSP, the nodes periodically broadcast synchronization beacons to their neighbors. Using a simple update algorithm, they try to converge to a common logical clock value and a common logical clock rate with their neighbors.

8 Conclusions

We have presented a synchronization protocol called HARMONIA targeted to low duty cycle multi-hop wireless networks. The requirements for the synchronization protocol come from a wastewater monitoring and actuation application called SwimNet, deployed city-wide in a mid- sized city. The nodes in SwimNet use an external clock called the Real Time Clock (RTC) which has a low drift, but a coarse 1 second resolution. The RTC is used for driving the sleep-wake periods on the nodes. The radio used on the nodes does not allow MAC layer time-stamping, a technique commonly used in synchronization protocols. The innovation in HARMONIA is to use a finely granular microcontroller clock with a relatively high drift to achieve synchronization of the RTC, such that the synchronization error is in the microsecond range despite the coarse granularity of the RTC. HARMONIA pipelines the synchronization packets enabling the entire network to be synchronized within an awake period. Also, failure handling for transient node and link failures is excuted locally, not overburdening the network with synchronization-related messages. Experiment results show that HARMONIA's synchronization error is higher than that of FTSP, but is still acceptable for SwimNet, being in the range of tens of microseconds. However, HARMONIA is significantly faster than FTSP with respect to the network synchronization time making it a good fit for low duty cycle networks.

In ongoing work, we are deploying and measuring the performance of HARMONIA in the real deployment. We expect to find interesting insights by subjecting our protocol to the different interference and collision environment. On the design side, we are laying out the failure handling functionality of HARMONIA for long-lasting failures. We are also looking to incorporate techniques to compensate for the difference in drifts without sacrificing the speed of HARMONIA.

9 References

[1] Digi International Inc. 9xtend oem rf module. http://www.digi.com/products/wireless/long-range-multipoint/xtend-module.jsp, Retrieved: 4/8/2009.

[2] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync protocol for sensor networks. In *Proc. of 1st intl. conf. on Embedded networked sensor systems*, pages 138–149, 2003.

[3] C. Lenzen, T. Locher, and R. Wattenhofer. Clock Synchronization with Bounded Global and Local Skew. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.

[4] M. Lukac, P. Davis, R. Clayton, and D. Estrin. Recovering Temporal Integrity with Data Driven Time Synchronization. In *Information Processing in Sensor Networks, 2009. IPSN'0.9 Intl. Conf. on (To Appear)*, pages 1–13, 2009.

[5] M. Maroti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd intl. conf. on Embedded networked sensor systems*, pages 39–49, 2004.

[6] Maxim Inc. Ds3231 extremely accurate i2c-integrated rtc/tcxo/crystal. http://www.maxim-ic.com/quick_view2.cfm/qv_pk/4627, Retrieved: 4/8/2009.

[7] L. Montestruque and M. Lemmon. Csonet: a metropolitan scale wireless sensor-actuator network. In *MODUS '08: International Workshop on Mobile Device and Urban Sensing*, 2008.

[8] J. Sallai, B. Kusy, A. Ledeczi, and P. Dutta. On the scalability of routing integrated time synchronization. *Lecture Notes in Computer Science*, 3868:115, 2006.

[9] T. Schmid, Z. Charbiwala, J. Friedman, Y. H. Cho, and M. B. Srivastava. Exploiting manufacturing variations for compensating environment-induced clock drift in time synchronization. *SIGMETRICS Perform. Eval. Rev.*, 36(1):97–108, 2008.

[10] T. Schmid, J. Friedman, Z. Charbiwala, Y. H. Cho, and M. B. Srivastava. Low-power high-accuracy timing systems for efficient duty cycling. In *ISLPED '08: Proceeding of the thirteenth intl. symposium on Low power electronics and design*, pages 75–80, 2008.

[11] M. Schütze, A. Campisano, H. Colas, W. Schilling, and P. Vanrolleghem. Real time control of urban wastewater systemswhere do we stand today? *Journal of Hydrology*, 299(3-4):335–348, 2004.

[12] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: A survey. *IEEE network*, 18(4):45–50, 2004.

[13] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Information Processing in Sensor Networks, 2009. IPSN'09. Intl. Conf. on (To Appear)*, pages 1–12, 2009.

[14] Tinyos-help. Ftsp on tmotes. http://www.mail-archive.com/tinyos-help@millennium.berkeley.edu/msg07079.html, Retrieved: 4/8/2009.

[15] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. *7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 381–396, 2006.

[16] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd intl. conf. on Embedded networked sensor systems*, pages 142–153, 2005.