

PhD Proposal: An Adaptive Sensor Network Architecture for Multi-scale Communication

Santashil PalChaudhuri

Computer Science, Rice University

1 Introduction

Sensor networks have emerged as a promising tool for monitoring (and possibly actuating) the physical world, utilizing self-organizing networks of battery-powered wireless sensors that can sense, process and communicate. These sensor networks can be rapidly deployed and at low cost, thereby enabling large-scale, on-demand monitoring and tracking over a wide area. The sensors can be deployed where it is difficult to monitor otherwise, such as natural or man-made crises like severe weather, wild-fires, earthquakes, volcanic activities, chemical, biological or nuclear agents, structural and habitat monitoring. Generally the model of sensor networks is: the sensors measure physical properties at different spatial and temporal positions, signal processing and data assimilation is performed in-network, queried results are communicated with one or more data sinks, and the sinks monitor, compute or actuate based on the data.

I propose the design of an adaptive cross-layered Sensor Network Architecture for enabling multi-scale collaboration and communication. In sensor networks, energy and communication bandwidth are scarce resources, while applications exhibit a limited set of characteristics. Thus, there is both a need and an opportunity for adapting the network architecture to match the applications, in order to minimize the resource consumed and extend the life of the network. There are various requirements and limitations of sensor networks, which make their architecture and protocols both challenging and divergent from the needs of traditional internet architecture.

Many applications, such as large-scale collaborative sensing, distributed signal processing, and distributed data assimilation require the sensor data to be available at multiple resolutions, or allow fidelity to be traded-off for energy efficiency. The architecture I propose enables scalability, localization and resolution-tuning, while simplifying application design by providing communication abstractions. Examples of a few applications of this multi-scale approach are as follows:

- *Multi-resolution Data Extraction:* A monitoring application requires variable resolution of sampled data. It requires higher resolution during operation at specific times of day or during periods of large variation, but lower resolution at other times. Also, a finite energy budget might be given, and the resolution has to conform to this budget while providing the best possible resolution.
- *Requisite Resolution Drilling:* An application wants to receive higher resolution data from a specific region, but lower resolution data from other regions. The specific region might be a high security building thereby requiring better monitoring, or the region might be deemed of interest because of the nature of low-resolution data emanating from it.
- *Hotspot Localization:* Analyzing the data from a multi-scale monitoring framework, an application deems an area to be a hotspot, and requires to map the sensors to the geographical location. The hotspot might actually be a fire in a specific place, or maybe due to malfunctioning sensors in that region to produce anomalous data.

As the sensed data are measured quantities governed by laws of physics, there is considerable temporal and spatial locality which are present. For example, a sensor measuring temperature will have a high correlation with nearby sensors measuring temperature as well as with its previous sensed values. A simple hierarchical structure while providing multi-resolution will not be able to exploit such correlation effectively. A network hierarchy aligned to the communication flow can lead to efficiency in the amount and frequency of data to be transmitted. Thus a *self-organizing, self-adapting* sensor network architecture is formed, optimized by feedback from the application.

My architecture also takes advantages of the fact that sensor network applications have limited set of behavior, when compared to completely generalized network applications. The communication pattern — source-destination pairs as well as duration and periodicity — is known apriori in many applications. As opposed to multi-hop forwarding in ad hoc networks, in sensor networks, *fusion* of data takes place while being forwarded thereby possibly reducing communication. These application behavior provides opportunities to adapt the network protocols to match application requirements. These adaptations require cross-layer optimizations in the networking stack, which goes beyond the strict protocol layering.

The contributions of my thesis are as follows:

1. **SensorStack Architecture** — I propose a sensor network architecture in Section 2, which enables cross-layering optimizations and adaptive protocols. An Information Exchange Service (IES) provides a publish-subscribe interface for notification of events, facilitating protocol adaptation.
2. **Multi-scale Data Service** — I design a self-organizing adaptive hierarchical overlay for multi-scale communication, and provide communication primitives to simplify application design in Section 3.
3. **Hierarchical Medium Access Scheduling** — I design a medium access scheduling component tailored for the hierarchical data service layer in Section 4. The scheduling achieves time division multiplexing by taking advantage of the communication characteristics.
4. **Adaptive Synchronization** — In Section 5, I provide a clock synchronization service with an adaptive synchronization granularity. I convert service specifications to actual protocols parameters to provide synchronization with minimum overhead.

2 Adaptive Sensor Architecture

Wireless Ad hoc Sensor Network(WASN) [45] has recently attracted a considerable amount of attention due to its application potentials. Despite similarity to research issues in ad hoc networking and distributed systems, there are several requirements that are fairly unique to WASN leading to many recent innovations to the different layers of the traditional protocol stack. These unique characteristics of sensor networks which differentiate them from traditional networks are as follows:

- *Resource constraint:* Sensor nodes are resource-constrained in energy, computational and communication abilities. There are many applications where WASN will be deployed in remote areas, such that the nodes are either physically unreachable, or the cost of recharging them prohibitive. To increase the WASN application lifetime, there is a need to optimize energy consumption at all layers.
- *Data-centric routing:* Typically, the sensor applications do not care about the specific source providing the data; it cares more about the content provided. For example, an application wishes to find out the maximum temperature in a region, and the sensor nodes to be contacted for this purpose is not known apriori. This is very different from traditional internet routing and more aligned to content-based routing. To handle such needs, low-level naming of the nodes and interest-based data routing have been proposed by recent research [22].
- *Adaptability:* Sensor networks have a limited set of applications with specific set of requirements. So, the stack can be optimized for these applications, which is not possible in a completely generalized application scenario. This is an opportunity which should be taken advantage off, in order to better serve the application demands with limited resources.
- *Large scale:* WASN might consist of thousands of sensor nodes, much larger than typically realized in ad hoc networks. This makes the traditional way of stateful routing impractical because of the size and number of routes.
- *Data Fusion:* In sensor networks in-network data fusion is done at every hop by the application, instead of mere forwarding. This takes advantage of the correlation existing between nearby sensors. Called in-network processing, this technique is very useful for reducing communication cost. Support for this fusion needs to be inbuilt into the stack for performance reasons.
- *Stack Services:* Localization and synchronization are of paramount importance for sensor network applications. Stack should provide these service adequately and efficiently. Various other services provided in traditional stacks like flow control at transport layer and node fairness at medium access layer are not generic requirements for a large class of applications. Putting them in the stack incurs considerable performance penalty for applications not needing these services, and hence should be handled in the end-to-end fashion [8].

Traditionally, layering has been used as a design principle for networking stacks. This technique organized a network system into a succession of logically distinct entities, such that the service provided by one entity is solely based on the service provided by the lower level entity. This is important to deal with complex systems, as the explicit structure allows identification of the relationship between the pieces. This modularization eases maintenance, updating of the system components, and the change of the protocol in a layer is transparent to the rest of the system. However, the temptation and need to incorporate cross-layer adaptation is paramount in sensor network architectures. This is a manifestation of the ever present tension between architecture and performance, but I believe that for sensor networks, performance (and efficiency)

have a strong case because of the needs (resource-constraints) and opportunities (limited application behavior) presented. In providing a design for cross-layer interaction however, care should be taken so as not to lead to spaghetti design, which becomes difficult to evolve further [29].

In an effort to address these WASN characteristics, most research efforts to date in sensor networks have either completely ignored the traditional network stack and specialized the transport to suit the specific approach (e.g. data diffusion [27]), or have made localized modifications using the traditional *Open Systems Interconnection* (OSI) reference model [28] as the de facto standard (e.g. PicoNode [47]). The standard sensor node OS called TinyOS [23], provides the lowest layer of networking stack, MAC, giving a messaging service, and all routing and fusion logic is provided as an application on top of it. This is done due to the limited resources of the current sensor nodes. My proposed stack extends this to provide services like fusion, routing, filtering, and synchronization within the stack in more powerful future sensor nodes. There has also been considerable research effort directed at the different layers of the OSI model, but most of them have tended to treat the layers independently as encouraged by the OSI model. Though there has been some cross-layer optimization efforts, much more optimization is possible if the layers are treated together.

I believe that this is an appropriate time to think of a new design of the protocol stack for future sensors. I also believe that CPU and memory will not be constraining resources in future, but energy will remain to be. As sensor networks will become pervasive and powerful in the foreseeable future, it is counter-productive to design a customized stack for each application. On the other hand the traditional stack is inappropriate to WASN as explained above. With this motivation, I list the design goals of a stack for sensor networks and sketch the architecture.

2.1 WASN Stack Design Requirements

This section identifies a set of requirements for the SensorStack, a new protocol stack suitable for WASN environment.

Cross-layering : SensorStack should make the relevant information from one layer available to other layers such that the other layers can take more informed decision. The resource constraints of WASN demands the need to achieve optimization in an integrated way.

Adaptability : SensorStack should allow applications to tune the stack behavior and thus adapt itself to application-specific needs. Applications will have access to information about the user requirements, the network topology, etc, and these information can be used for improved decisions at the protocol stack : which route to take for data transmission, whether to do error checking or not, what duty cycle should be used for the low-power radio, etc.

Supporting application-specific adaptation can be taken in two extremes: *first*, stack is implemented for a particular application and the application directs the stack behavior without any intermediate abstractions, and *second*, the stack is implemented for a class of applications and applications direct the stack behavior via a broker service. SensorStack adaptation is not suitable for the first category, because this will need different protocols to be written for different sensor applications and these protocol codes to be loaded into network stack at run-time. Since the class of sensor applications have many common requirements, it seems plausible to support the application-specific adaptation via the broker service.

Network Programming Interface: SensorStack should provide an interface for addressing of the nodes via a programming interface. If the stack understands the concept of logical naming, then data-centric routing can be supported effectively. For example, if a node wants to send a data packet to gather temperature from the nodes in region X , the logical name of the destination becomes : “nodes in region X ”. This can be supported by flooding the information, as is done in directed diffusion. Here, every node who listens to the packet will compare its own location with X , and accordingly it will decide whether to reply with its temperature data or ignore the packet. This happens at application layer, which means the data packet has to travel to the application layer at every node in the network even for nodes not lying in the region X .

In-stack Data Fusion : SensorStack should provide mechanism to do data fusion in the protocol stack. If fusion logic is made part of the protocol stack, the data packets do not need to reach the application layer at the intermediate hops between source and destination nodes. This will improve the end to end latency of data packets. Also, by providing the common fusion requirement of sensor application as a separate module, application development can be made more modular. Thus, same fusion code can be used for multiple applications alleviating the load on resource constrained sensor nodes.

2.2 SensorStack Design

Based on the requirements discussed in previous section, I sketch the proposed SensorStack. Figure 1 presents this new proposed stack. The three main mandatory services needed to the sensor applications are Medium Access, Data Service, and Data Fusion. The services are situated logically in the outgoing data flow path in the shown order — 1. Data Fusion, 2. Data Service, and 3. Medium Access. The path from incoming packets might go all the way up to the application if the node itself is the destination, but can be handled in the Data Service or the Data Fusion layer if the node is an intermediate hop. The additional services like Clock Synchronization and Localization are in the control path and vertically integrated, providing the services to the application as well as to all the modules in the stack. Information Exchange Service (IES) acts as the broker among different service modules and the application to improve cross-layer optimizations. Below I describe the SensorStack architecture, of which I will build a few building blocks later in the thesis.

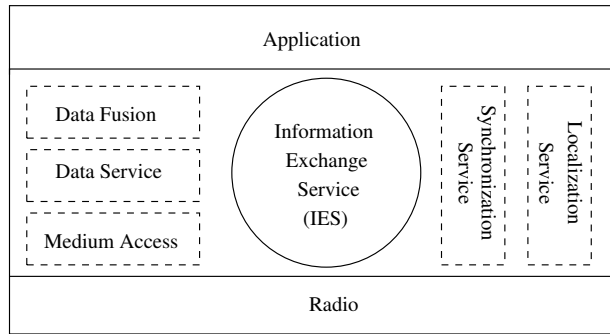


Figure 1: SensorStack

Information Exchange Service: This service acts as an information database and notification service. It serves two main purposes: allowing cross-layer optimization by making one service’s information available to another service, and making application requirements available to the different layers. By having a generic information exchange service, SensorStack can make the information available transparently, and it can control the information access and update rights in a centralized manner. This solves the problem of cross-layer design leading to very complicated dependent stack modules.

This service can be provided as a publish/subscribe API. Different protocol layers and the application layer can publish and subscribe to the relevant information. For example, some sample cross-layer interactions provided by IES include:

- A node monitoring service may be running at the application layer and it can publish the information to IES. This health information can be subscribed by different protocols in the stack. It can be used to do the next hop selection to choose the maximum energy path. Similarly, location information can be used to support geography-aware routing. It can also be used by MAC layer to do energy optimization, like adjusting the transmission power or to adjust the directional antenna.
- Clock synchronization between nodes is essential for many applications like tracking and monitoring, as also to build an efficient medium access protocol.
- The data communication schedule information can be published through the IES. This information can be used to schedule the radio wake-up and sleep, or to do energy-efficient medium access control. Application can directly publish this information if it knows its communication need, or a sub-service can be implemented that generates the communication schedule by looking at the traffic pattern.
- The data service layer and the medium access layer can use application data unit size to decrease the packet fragmentation.
- The medium access layer can provide the latency and congestion between itself and a neighboring node, and this information can be useful to do congestion control by the data service layer.

Medium Access : This layer provides the traditional TCP/IP’s MAC layer service, i.e. medium access for hop-to-hop data transfer and the channel error control. The medium access layer can use the following information for better efficiency: Fusion functions of the application which help it to determine the amount of data compression in a multi-hop route, the routing structure which provides efficient scheduling of the nodes, and periodicity of application data generation. In section 4, I show how this information obtained through the IES helps build a more efficient medium access layer.

Data Service : This layer provides two main services, namely, provides an abstraction for delivery of data based on content and type rather than address, and efficient implementation of necessary information flow. In Section 3, I provide a clustering hierarchy which optimizes the communication flow based on interaction with the application. It also provides network programming interfaces which abstract the details of low level communication and provides a data-centric routing interface.

Data Fusion: This layer needs to support both types of fusion mechanisms: first, where the data packets are required to be fused at every hop, second, where the data packets are to be considered for fusion only at the (application-specified) selected nodes or the destination before the data is delivered to the application. The first kind of fusion mechanism is meant for hop-to-hop data transfer, and is useful for application scenarios where every node is sensing some useful data that needs the in-network fusion [35]. The second kind is useful for supporting more traditional way of doing fusion at the end-points, when only some nodes are contributing towards the information that is being sought [32]. Creating a specific instance of a Data Fusion layer is outside the purview of my thesis.

Helper Service: Helper services are responsible for supporting data capturing, data presentation at the sink, and providing other services that can be used to adapt the SensorStack for the application-specific demand. Localization and synchronization are two important services that need to be part of SensorStack, and are placed in the helper service. The localization and synchronization service is configured and publishes location and time information through the IES. For example, the accuracy and granularity of clock synchronization is controlled by the various modules requiring synchronization, and clock synchronization module provides the requisite service. This is elaborated and a solution presented in Section 5.

2.3 Related Work

OSI reference model provides the basic framework for development of standards for interconnecting two or more systems. TCP/IP stack has similar motivation and structure as the OSI model. Each layer in OSI provides a set of *services* to the subsystems in the layer above. In providing these services, a layer implements a set of *functionalities* using the services made available by the layer(s) below.

The layered model of the TCP/IP stack and the OSI reference model encapsulate the issues appropriate to the related tasks within each layer. In a standardized way, this layering allows transparent access to all lower-level functions, and makes it possible to upgrade any given layer without the redesign of other layers. The strength of layered approach lies in its modularity. Every layer now can be supported by different vendors, or implemented for different hardware platforms, and yet the overall stack can be realized by simply combining the appropriate protocols for every layer. This modularity leads to simplicity by hiding the complexities of the lower layers.

While the TCP/IP stack has been in common use on general purpose machines, they have also been adapted for more specialized networks or application needs. Specifically, research projects have looked at the scope of cross-layer optimizations in TCP or UDP stack for wireless networks. For example, there have been proposals [44, 20, 62] for sharing the physical and MAC layer knowledge of wireless medium with the higher layers for efficient allocation of network resources or for congestion detection. Similarly, research for providing QoS-based services has sought for cross-layer collaboration in network stack [5]. But, these cross-layering efforts have only increased the complexity of the protocol stack, with the layers being much more dependent upon one other, and less modular than the original stack.

Application-specific network protocol design has been explored by some projects. Plexus [16] allows applications to achieve high performance with customized protocols, where the application-specific protocols are installed dynamically into the operating system kernel. But, Plexus needs the protocols to be written in Modula-2, a type safe language, and it runs only in the context of the SPIN extensible operating system [3]. Exokernel [15] presents an architecture to permit the application-specific customization of operating system abstractions, including the network resources. While Exokernel provides a suitable architecture for supporting application-specific protocol stack, it is unclear what set of network abstractions will be suitable and/or sufficient for the evolving WASN environment and WASN applications.

Berkeley motes use TinyOS [23] operating system. TinyOS uses active message based networking and provides only the minimal networking support to the applications because of the extreme hardware constraints of the motes. This limits the types of applications that can be efficiently supported. The network stack used by PicoNodes at Berkeley [47] identifies the need for many WASN specific situations (such as keeping the transport layer off the stack), but it still does not support application-specific adaptation and other WASN design goals.

3 Data Service

In this section, I propose an *adaptive network architecture* that matches the communication characteristics of different applications by optimization based on application feedback. I design a hierarchical overlay to handle aggregation, dissemination, and multiple resolution, and I leverage the Abstract Region [55] architecture to handle local collaboration between nodes. These overlays coexist and serve different purposes; together they efficiently support a wide range of different application data communication patterns. To simplify the application design, I provide a set of Network Programming Interfaces to abstract the details of low-level communication. Applications specify their communication needs through these interfaces; the architecture then uses this information to optimize the communication data flow.

Many applications, such as large-scale collaborative sensing, distributed signal processing, and data assimilation, require the sensor data to be available at multiple resolutions, or allow fidelity to be traded-off for energy efficiency. I form a self-organizing network hierarchy that can scale to very large numbers of nodes and enables multi-scale data communication. This multi-scale hierarchical overlay adapts to form clusters such that data communication becomes efficient. While a self-organized hierarchy has been known to scale well, this is the first proposal to align the network hierarchy with the application data flow.

After overviewing related work in Section 3.1, I describe the hierarchical overlay in Section 3.2. Section 3.3 details the initial design evaluation.

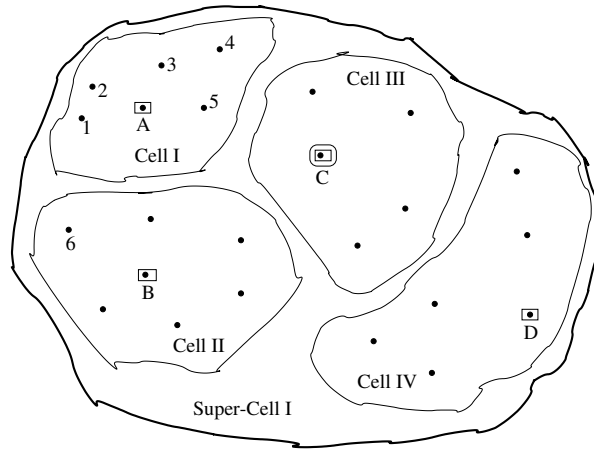


Figure 2: Cluster hierarchy

3.1 Related Work

Various protocols and architectures have been proposed over the years for sensor networks. Earliest were the diffusion class [27] of algorithms, which are effective in aggregation and dissemination communication abstractions but cannot support multi-resolution communication required by many applications. There are various ways of implementing these diffusion algorithms depending on the application behavior, such as two-phase pull, one-phase pull, and push. The specific diffusion behavior can be chosen by the application to match its requirements [21]. GARUDA [42] is an architecture that handles reliable delivery of downstream data under various notions of reliability.

Fractional Cascading [18] and DIMENSIONS [17] have recently been proposed to handle multi-scale data communication. I extend their design, which is essentially for storage and retrieval of sensor data, to have more general applicability. Their architecture is based on a regular grid structure and assumes regular data sampling. However, practical multi-scale transforms need to accommodate networks with arbitrary irregular placement of sensors; I achieve this using the self-organizing hierarchy. My architecture adapts to the node collaboration requirements to make the networking more efficient, which is not addressed in these two previous approaches.

SDIMS [60] is another hierarchical approach that is more targeted toward wired networks but can be adapted for wireless sensor networks. This approach provides a flexible API for configuration but does not address the adaptability to application requirements. It does have a tunable interface for a tradeoff between latency and overhead for added flexibility.

The goal of Abstract Regions [55] is to simplify the application design by providing abstract interfaces to hide the details of low-level communication. It proposes the concept of neighborhood as a programming unit, and shows how various applications can be efficiently written using it. Many applications need multi-resolution data though, and my architecture addresses the abstraction requirement for such applications. The Abstract Region concept of neighborhood is thus complementary with my approach, and both together encompass a much wider range of application requirements. Hood [56] is another approach to providing a programming abstraction, but it is also targeted toward neighborhood-based programming models only.

3.2 Hierarchical Overlay

I design the hierarchical overlay for efficient aggregation, dissemination, and multiple resolution of application data. In this overlay, a self-organizing hierarchical clustering is formed, inspired by the self-organization component of protocols like Safari [12] and L+ [6].

The hierarchy is a recursive organization of nodes into cells, cells into supercells, and so on, based on an autonomous self-election of a subset of the nodes into *drums*, and iteratively drums self-electing to become higher level drums, and so on. The drum is also called the *parent* for all nodes within its cell. Figure 2 shows an example cell hierarchy. In the figure, nodes 1, 2, 3, 4, 5, and A group together to form a cell (called *fundamental cell*) with node A being the drum for the cell. Each of the drums in the network, namely nodes A, B, C, and D form a higher level cell with node C being the drum for that higher-level cell (called a *super-cell*). This hierarchy formation goes on iteratively until all the nodes come under one highest level cell. The self-selected drums aid in this hierarchy formation by sending periodic beacon packets during initialization or during topology change.

Each node can be thought of as a level 0 cell and a level 0 drum (level 0 drums do not send beacon packets). Every level k drum is at the same time also a level i drum, for all $i < k$. This hierarchy formation algorithm is distributed, with no central coordination. Drums of the same level are roughly uniformly spaced, with higher level drums more sparse

Algorithm 1. Cluster Formation

```
1: repeat
2:    $Drum_i$  wait for a random time up to  $T_{max}$ 
3:   if  $Drum_i$  does not hear a higher level drum beacon, or is not the highest level drum then
4:     Steps up to  $level_{i+1}$  and starts sending periodic beacons with hop limit of  $\mathcal{D}_{i+1} = \alpha \times \mathcal{D}_i = \alpha^i \times \mathcal{D}_1$ 
5:   else
6:     Associates with the nearest  $drum_{i+1}$ , randomly choosing between equivalent forwarding paths.
7:   end if
8:   if  $Drum_i$  hears any non-duplicate beacon by a drum in its super-cell then
9:      $Drum_i$  forwards the beacon.
10:  end if
11:  if  $Drum_i$  hears any  $beacon_i < \mathcal{D}_i$  hops away then
12:    The drum with the lower id steps down to  $level_{i-1}$ 
13:  end if
14:  if  $Drum_i$  hears any  $beacon_{i+1}$ , which is closer than current  $beacon_{i+1}$  then
15:     $Drum_i$  associates itself with the closer drum
16:  end if
17: until All nodes are assigned stable coordinates
```

than lower level drums. A *coordinate* of any drum at level i is the concatenation of the coordinate of the level $i + 1$ drum with which it associates, along with a unique identifier. This unique identifier can be any random string large enough to avoid collisions. I use an address assignment technique proposed in TreeCast [39], whereby the nodes are given compact addresses minimizing the length of address strings. Various sensor applications require the identifiers of nodes along with their values, and this technique leads to efficient encoding of the identity (and location) of the nodes.

In the initial startup period, each drum sends beacon packets (*beacons*) periodically, containing the beacon sequence number, the drum level, and a hop count, which aids in the hierarchy formation. These beacons are forwarded by all nodes within the hop count limit or those within the cell defined by the parent of the drum sending the beacon. In Figure 2, the beacons from node B reach all the nodes in super-cell I. The beacons sent during this initial phase also give the shortest path from any drum of level i to its associated higher level drums and the drums in the same level within its super-cell.

Algorithm 1 shows the *Cluster Formation* algorithm. A drum of level i is denoted by $drum_i$, and $drum_0$ denotes the nodes. Beacons sent by $drum_i$ are denoted by $beacon_i$. \mathcal{D}_1 is the fundamental cell diameter and is dependent of the cluster size required by the application.

The drums wait a random time between 0 and T_{max} , before deciding whether to become a higher level drum or associate with another drum. When a node has more than one equivalent drums to choose from, it chooses any one randomly. Also for the same drum, there might be multiple paths through different forwarding nodes, one of which is also chosen randomly. This leads to balanced forwarding trees being formed, thereby evenly distributing the load amongst nodes in the network. The association scope of the drums, determined by the hop limit of the beacons, increase geometrically with the level of drum. Decreasing α increases the number of levels in the hierarchy while reducing the number of nodes in each level, whereas increasing the α has the opposite effect. Lines 8-10 of the algorithm ensure that the beacons of any drum reach all nodes within its super-cell. Lines 11-13 ensure that no drums of same level form too close to each other, as that reduces the efficiency of the clustering. Lines 14-16 make the hierarchy evolve such that the drums are always associated with closest higher level drum. The number of levels formed in the hierarchy is of $O(\log(N))$, where N is the number of nodes in the network. As a simple analysis of the cost of the algorithm, if instantaneous propagation is assumed, then all $level_i$ drums are separated by \mathcal{D}_i hops, with very high probability, and so the total startup phase is of $O(T_{max} \log(N))$. Therefore the latency can be made smaller by reducing T_{max} to an optimal value.

The hierarchy formed can be analyzed using the Random Sequential Adsorption (RSA) model, since under the instantaneous propagation approximation, the hierarchy formation conforms to the RSA model [12]. The average number of level 1 drums formed is

$$n = 0.547 \left(\frac{N}{\pi \frac{\mathcal{D}_1^2}{4} \rho} \right) \quad (1)$$

where N is the number of nodes in the network, ρ is the node density in hop-metric sense, its value depending on the transmission range and spatial density of nodes. When applied to multiple levels of the hierarchy, this gives us

$$\frac{n_i}{n_{i+1}} = \left(\frac{\mathcal{D}_{i+1}}{\mathcal{D}_i} \right)^2 \quad (2)$$

where n_i is the number of level i drums.

3.2.1 Adaptive Hierarchy

During startup, each drum sends beacon packets periodically, which are forwarded by all nodes based on a policy of geographical (hop count) proximity. These beacons then induce a cell hierarchy that is proximity based. In various sensor

Algorithm 2. Recluster (*Selectors*)

- 1: *Drum sends out beacons with Selectors*
 - 2: *Nodes matching the Selectors (re)select the Drum as their parent*
 - 3: **if** Any node become orphan or cell is sub-optimal **then**
 - 4: *It sends Solicit Beacons with specified Selectors*
 - 5: *Nodes matching the Selectors respond with Beacons*
 - 6: *Choose the drum most suitable with respect to the Selectors, and become a child of the drum*
 - 7: **end if**
-

network applications, proximity is a good measure of correlation and hence of data interchange. So, the above approach of cell structure formation matches the collaboration between and nodes.

In many other applications though, the collaboration and communication take place between nodes based on various other constraints. For example, nodes with similar magnetic field or temperature readings within a neighboring scope might need to communicate more often. So, if the clustering hierarchy matches the collaborative set of nodes, communication can be efficiently abstracted and implemented. I use some application specified filters, called *Selectors*, to align this hierarchy to match the collaboration and communication sets of nodes.

I define a *Selector* as a tuple of $\langle \textit{attribute}, \textit{value}, \textit{operator} \rangle$, where *attribute* is any application specified variable, and *value* is a valid element from the range of the attribute. The *operator* is a binary operation (such as $>$, $<$, or $=$) with *value* being one of the operands. I extend the definition of a *Selector* to form:

$$\begin{array}{lcl} \textit{Selectors} & = & \textit{Selectors} \wedge \textit{Selectors} \\ & | & \textit{Selectors} \vee \textit{Selectors} \\ & | & (\textit{Selectors}) \\ & | & \textit{Selector} \\ & | & \text{null} \end{array}$$

The values for the attributes are shared between the application, sensor hardware, and the networking layer through the IES, which enables the *Selectors* to be evaluated at the networking layer. An operator needing a time-series of previous attribute values might entail the sharing of whole data structures of application computed values. Currently only scalar operators are supported; more complex operators could have application-defined call-back functions to enable them to be evaluated by the application.

The beacons of drums are forwarded by a node if the hop count in the beacon packet is less than a specific value and the specified *Selectors* evaluates to true for the attribute values in that node. For an empty *Selectors*, the effect is to forward the beacons based only on hop count.

Reclustering of the network hierarchy to adapt it closely to the communication flow is initiated by the application locally in cells where adaptation is needed. This is best judged by the application, as the networking layer does not have any knowledge of the application logic or how the sensed values influence the communication. Algorithm 2 shows the *Recluster Algorithm*. The parameters for cluster formation are changed locally to reflect current communication patterns. The *Selectors* encode the criterion for the new cluster formation in the *Split Phase* of the algorithm. After reclustering, some nodes might become *orphans* (nodes without parent) or some cells might be smaller than optimal. These nodes or cells then merge with neighboring cells meeting the criterion in the *Merge Phase* of the algorithm. This reclustering is triggered by the application locally, and only occurs in the cells needing it for efficiency. The rest of the clusters in different areas are not changed. Hence, this reclustering takes place only locally where necessary, and invokes no long range messaging.

3.2.2 Network Programming Interfaces

I design a set of address-free Network Programming Interfaces (NPIs) to adhere to the paradigm that communication for the typical sensor network applications should be expressed without referencing specific nodes [11]. The interest is in data over space and time, rather than individual node values. The subject of direct one-to-one communication has been extensively studied in the literature, and I will not propose any new protocol for this but will leverage the existing body of work. I provide primitives to ease the programmability in a sensor network, by capturing the interfaces that are needed by sensor applications in general. Abstract Regions [55] proposed a flexible means of node addressing, by supporting data sharing using a tuple-space-like programming model. Their approach is similar to the MPI approach for parallel machines, by hiding the details of the sharing primitives. I support similar primitives and extend them for my multi-scale architecture, although sharing is explicit using put and get primitives, to provide a more efficient implementation of the programming model. The two groups of interfaces I support are *discovery* and *communication*.

Each of the interfaces can be implemented in either blocking or non-blocking fashion. In non-blocking mode, the operation is invoked through a command, and when the operation is complete, a call-back is invoked on the original requesting component. In blocking mode, the operation may block and then resume on an interrupt either by a timer or message arrival. The blocking mode is significantly easier to program, as in the non-blocking mode the programmer has

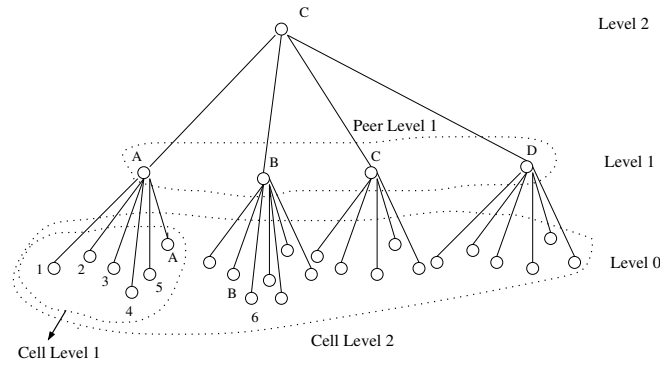


Figure 3: Virtual hierarchy

to handle the synchronization and call-back explicitly. TinyOS [23] supports the non-blocking concurrency model, but a lightweight thread-like abstraction called Fiber [55] has been implemented recently as a blocking model.

Figure 3 shows the *virtual* hierarchy schematic of Figure 2, which will help explain the interfaces. All the levels above level 0 are virtual, and the nodes only exist at the lowest level.

Discovery Interfaces

Discovery interfaces can be invoked by any node to give itself information about related nodes. This information is gathered periodically, with a period as specified by the application, and the application is informed of any changes in the information. This procedure is continuous, either triggered by node failures or additions. The information might contain the identifiers of the set of nodes, their locations, link quality and number of hops to each of them, and resource (e.g., remaining battery or available sensors) present in each of them. The information learned from these discovery interfaces can be used to configure the Selectors with any specified criterion. For example, the Selectors might be specified to filter nodes within a specified geographical distance or with higher than a specific link quality. There are three possible interfaces supported. As a node can be simultaneously at different levels, a level is specified for each interface to specify the level for which the information is required. The format for representing the gathered information is dependent on the implementation.

- *Parent Information (Level)*: Returns information about the parent of the node at the specified level. For example in Figure 3, this interface when invoked on node 6 with level 1 returns information about node B, and with level 2 returns information about node C.
- *Peer Information (Level)*: Returns information about the peers of the node at the specified level. For example in Figure 3, this interface when invoked on node B at level 1, returns information on the set of nodes A, B, C and D.
- *Cell Information (Level)*: Returns information about the cell of the node at the specified level. For example in Figure 3, this interface when invoked on node A, returns information on the set of nodes 1, 2, 3, 4, 5 and A.

Communication Interfaces

In the multi-scale architecture, I support both kinds of communication models: *Put* and *Get*. In *Put*, a node sends data to its cell, parent, or peers, whereas in *Get*, a node solicits data from its cell, parent, or peers. The *Put* interfaces correspond to the push paradigm, and the *Get* interfaces correspond to the pull paradigm that has been proposed by the diffusion type of algorithms [27]. I support both types, as different applications might be optimized using different paradigms, as pointed out by Heidemann et. al [21]. In some situations where the data generation rate is infrequent and unknown, polling using *Get* will be inefficient; using *Put* by the source of the data when the data is generated will be optimal. In another scenario, where the data generation rate is high and consumption rate is low, pushing data using *Put* will entail redundant data communication; using *Get* by the consumer of the data when the data is required is optimal in this case. The *Put* Interface can be implemented in multiple ways: stored locally, sent immediately to the designated scope, or cached at different intermediate locations. Similarly, the *Get* Interface implementation might involve either fetching remote data or local retrieval. The specific implementation depends on the application characteristics.

I also support *Reduction* interfaces that use an associative operator (such as *sum*, *max*, or *min*) to reduce an attribute across all the nodes in a specified region. This *Reduction* interface can be implemented using *Get* and *Put*, but efficient implementations can take advantage of local reductions while propagating the values. This abstraction also provides ease of programmability.

There are three groups of primitives a node might address: its parent, its peers, or its cell. This leads to six different interfaces for *Put* and *Get*. *Reduction* interfaces are done on either cells or peers. All examples below refer to Figure 3. For

node A, the parent is node C; the peers at level 1 are B, C and D; and the peers at level 0 as well as the cell at level 1 are nodes 1, 2, 3, 4, and 5.

- *PutParent (Attribute, Value)*: The value of the attribute is sent to the parent node. When called on Node A, the data is sent to node C.
- *PutCell (Level, Selectors, Attribute, Value)*: Level can be at most one level higher than the node using this interface. So, a node of $level_0$ can send message to the fundamental cell. In general, a $drum_i$ can send message to all nodes in the same $level_{i+1}$ cell. For Node A, PutCell called with level 1 delivers data to nodes 1, 2, 3, 4 and 5, while called with level 2 delivers data to all the nodes marked by Cell level 2. The targeted nodes can filter the receipt of the Data using the *Selectors*.
- *PutPeer (Level, Selectors, Attribute, Value)*: The level can be at most same as the level of the node using the interface. This interface provides the same functionality as PutCell for $level_0$ nodes. For node A, level 1 delivers the data to nodes B, C and D (Peer Level 1).
- *GetParent (Attribute)*: The value of the attribute is solicited from the parent node.
- *GetCell (Level, Selectors, Attribute)*: Level can be at most one level higher than the node using this interface. In this interfaces, Data is received from the cell nodes matching the *Selectors*.
- *GetPeer (Level, Selectors, Attribute)*: The level can be at most same as the level of the node using the interface. This interface provides same functionality as GetCell for $level_0$ nodes.
- *ReduceCell (Level, Selectors, Attribute, Operator)*: This interface is applied on the attribute for all nodes in the cell specified by level and *Selectors*. And the reduced attribute value is stored locally. For example for operator *max*, the maximum attribute value within the cell is returned by this interface.
- *ReducePeer (Level, Selectors, Attribute, Operator)*: Similar interface where the scope is all the peer nodes at the specified level.

3.2.3 Efficient Communication Operations

The Network Programming Interface in the previous section is used by the applications to form a clustered hierarchy and to adapt it for efficient communication. In this section, I describe mechanisms for efficiently supporting the different communication interfaces. I assume the existence of bidirectional wireless links, which is true for most commonly used wireless MAC protocols.

- *With the parent node*: The drum beacons that are used to form the cluster hierarchy is utilized to route from and to parent node, by following the path or reverse path of the beacons respectively. If the path breaks, due to nodes in the path moving away or dying, then local route repair is done to find a new route. The drum whose path breaks, sends out beacons for a short interval to repair the broken path.
- *With the peer nodes*: At any level, the peer nodes need to be able to communicate with each other efficiently. This is achieved by expanding the scope of the beacons for the drums. The beacon packet of a level n drum is also forwarded by all nodes in the level $n + 1$ cell of the originating drum. The reverse path is followed to reach each peer. This is only done in the startup period or when a path breakage is detected. Multicasting at network or MAC layer (if possible) is done to prevent duplicate packets along common part of the paths. For example in Figure 2, beacon packets from node A in cell I is flooded to the whole super-cell I. And hence B, C and D know of the shortest path to A.

The above technique leads to minimum latency communication from any node to the rest of its peers, using shortest path. But, it also leads to higher cost in terms of number of forwards. Alternatively a Minimum Spanning Tree can be formed between the peer nodes. This leads to lesser number of forwards, but also leads to higher maximum latency. Figure 4 shows an example topology to illustrate this. This tradeoff can be exposed for the application to choose from.

- *Within the cell*: Communication from any node to the whole cell can be achieved by simple flooding within the scope of the cell, whereby each node forwards a packet exactly once. However, this is very sub-optimal and leads to many redundant broadcasts specially in a dense network. I describe next a strategy for optimal cell flooding.

Optimal Cell Flooding

Typical broadcasting using simple flooding, where each node forwards a packet exactly once, leads to broadcast storm problems [38] and is very energy inefficient. To form a more efficient flooding algorithm, there has been substantial work

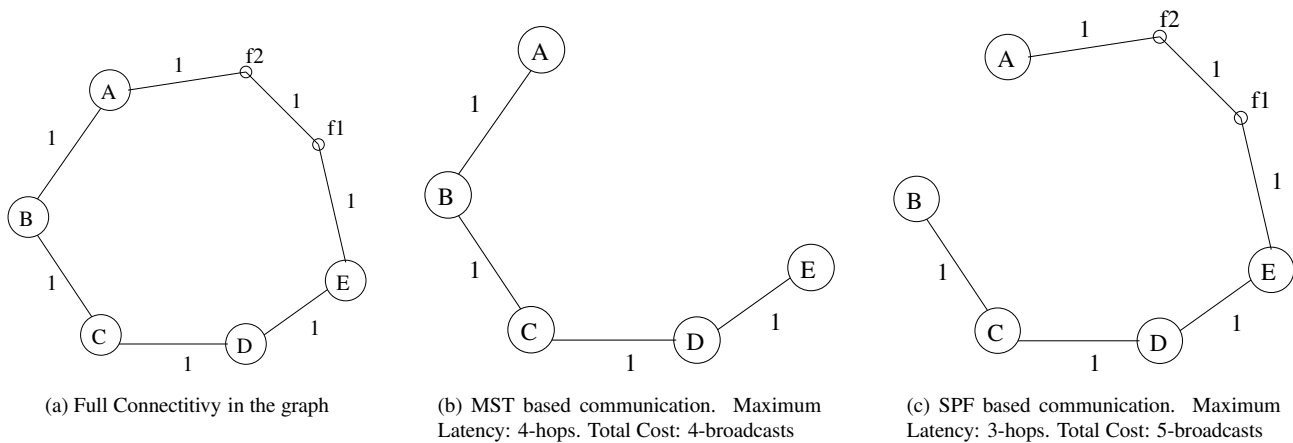


Figure 4: Example topology connecting peer nodes (A, B, C, D and E) and forwarding nodes (f1 and f2)

with regard to carefully choosing the forwarding nodes to reduce the number of forwards without reducing its effectiveness. Williams and Camp [57] categorized the techniques recently. In probability-based methods, nodes forward with some variable probability parameter chosen randomly or depending on the number of broadcasts heard. In area-based methods, distance or location of the nodes are taken into account by the node before deciding on forwarding. Both of these methods are completely localized without the need for any coordination. In neighbor knowledge methods, a distributed algorithm forms a Connected Dominating Set (CDS) to choose a subset of nodes to be forwarders. This has more overhead than the previous methods, but leads to more optimal flooding due to better knowledge about the neighborhood. Ideally, a Minimal Connected Dominating Set (MCDS) will give the most efficient set of nodes to forward packets such that all nodes are reached. Building a MCDS is an *NP-Complete* problem however, and the problem gets harder in sensor networks in the absence of global knowledge. There has been a significant body of work on approximating the MCDS using heuristics [4].

There are two types of neighbor knowledge methods proposed: *Relaying* in which a node determines the forwarding status of its neighbors, and *Pruning* in which a node makes a local decision on its forwarding status. Multi-point relaying is an efficient approach for relaying, and has been used in the OLSR ad hoc network routing protocol [9]. The re-broadcasting nodes are explicitly chosen by the upstream nodes, either via “hello” packets, or within the header of each broadcast packet. In relaying, there is either additional overhead in each packet to designate the forwarding nodes, or relevant state that has to be maintained by each node. The statelessness of protocols has prime importance in an unreliable network of sensor nodes, as a stateless protocol never operates on out of date state. In the pruning methods, nodes decide on their own locally whether to forward or not, leading to better reliability in the face of failure. There is automatic correction for small changes and robustness to big changes. Nodes can go to sleep independently in pruning methods, but there needs to be additional coordination in relaying methods so that no delegated forwarder is sleeping. In the absence of MAC layer multicast for pruning, nodes have to broadcast every packet for which all the neighbors have to process the packet before knowing that they are not designated forwarders. And finally, a comparison paper [57] showed very similar performance for both of the methods. I chose to use pruning method for the above reasons.

Various approaches for pruning based broadcasting use knowledge of k -hop neighborhood information, m -hop last visited nodes information for each packet, and priority between nodes. Larger value of k leads to more optimal forwarding set, but also entails higher cost for maintaining this neighborhood information. Larger value of m is also useful, but entails packet overhead. Wu and Dai [59] have proposed a generic scheme to cover all pruning based approaches. A node determines its forward status by finding existing replacement paths between all pairs of its k -neighbors. If all the replacement path nodes have higher priority values than itself or is already visited, then the node chooses to be a non-forwarder; else it forwards.

In designing an efficient flooding algorithm, the factors I chose were the following. The computational complexity is $O(k^2)$ for such algorithms, thereby dictating a small value of k . Also, recent proposals for sensor network MAC protocols [2, 48] maintain a 2-hop neighborhood to deal with efficient assignment of conflict-free slots. I choose value of k to be equal to 2. The information for m equal to 1 is available at no cost when the packet is received, by looking at the source. A greedy approach is taken to prioritize the nodes based on their degree of connectivity. My solution is similar to the approach taken in the Scalable Broadcast Algorithm [43]. In a more or less static network, the neighborhood information is invariant. On detection of neighborhood change, this two-hop neighborhood is recalculated by all nodes broadcasting their neighborhood nodes. My *Cell Flood Algorithm* is shown in Algorithm 3.

Algorithm 3. Cell Flood Algorithm

Require: $Node_B$ receives a broadcast packet from a neighbor $node_A$

Require: \mathcal{N}_B be the set of 1-hop neighbors of $node_B$.

Require: \mathcal{S}_B is a set of nodes in the 1-hop neighborhood which have not yet received the packet

```
1: if  $Node_B$  has handled the same broadcast packet before then
2:    $Node_B$  silently drops the packet
3: else
4:    $Node_B$  calculates a timer proportional to the ratio  $\left(\frac{\text{maximum}(|\mathcal{N}_X - \mathcal{N}_A| \text{ for } X \in \mathcal{N}_A \cap \mathcal{N}_B)}{|\mathcal{N}_B - \mathcal{N}_A|}\right)$ 
5:   Till timer expires,  $node_B$  updates  $\mathcal{S}_B$  when it hears any other neighbor broadcast, using the neighborhood information
6:   if  $\mathcal{S}_B = \emptyset$  then
7:      $Node_B$  silently drops the packet
8:   else
9:      $Node_B$  rebroadcasts the packet with some jitter
10:  end if
11: end if
```

$\text{maximum}(|\mathcal{N}_X - \mathcal{N}_A| \text{ for } X \in \mathcal{N}_A \cap \mathcal{N}_B)$ is the maximum number of additional nodes that can be covered by any node which has received a broadcast from $node_A$ and is in the neighborhood of $node_B$. Greedy approach is in choosing the broadcasting node in line 5 of the algorithm, by favoring the node with maximum additional coverage. In Line 6, \mathcal{S} is updated every time the node receives a broadcast of the packet from any neighboring node. Elements of \mathcal{S} which are present in the \mathcal{N} of the neighboring node are removed. If \mathcal{S} becomes empty, the packet is dropped, else the packet is forwarded.

Proof of full coverage: Every node in the network checks its neighborhood to determine whether all neighbors have received a packet, and forwards the packet if there is any uncovered node in the neighborhood. Hence, all the neighbors of any node in the network receives the packet. As the full network is an union of the neighborhood of all the nodes, hence all the nodes in the network are covered.

Optimality of coverage: The proposed greedy heuristic leads to a approximate MCDS for the graph. Analysis of the approximation bound gives $\log(n)$ times the cardinality of the optimal MCDS solution, where n is the number of nodes in the graph. The proof is similar to the optimality proof for Multi-point forwarding [46].

Optimal Selectors implementation

Selectors filter can be specified in some of the communication interfaces. If the filter is *null*, then the communication operations are efficiently implemented as elaborated previously. If this filter is not *null* but has a *Selectors*, then this scope (termed *Selectors* scope) is a subset of the scope with *null* selector (termed *null* scope). The communication operation can be done assuming *null* selector, and filtered at each node. This is not most efficient, specially is the *Selector* scope is significantly less than *null* scope. I implement the scoping of the *Get* and *Put* operations using the following technique.

If operations with any *Selectors* filter is invoked the first time, the communication has to be delivered in the *null* scope as there is no knowledge of the location of the nodes matching the *Selectors*. But, for frequently invoked *Selectors* filters, an optimized implementation is done to cover the matching nodes only. If any particular *Selectors* is invoked frequently at any particular node, a trigger is set. The first packet being delivered after this trigger is set includes a *DoReinforcement* flag. All the nodes matching this *Selectors* filter, sends a *Reinforced* message back to the source, specifying the *Selectors*. All the nodes through which this *Reinforced* message passes back are part of the forward set of nodes for the specified *Selectors*. All the subsequent packets with this *Selectors*, has a *OnlyReinforced* flag and is forwarded by the forward set of nodes only. This is remembered for a specific interval, greater than the period specified in the communication interface, if present. Periodically the *DoReinforcement* flag is set again and forwarded by all nodes to account for any change in topology or interest.

3.2.4 Multi-Scale Application

To demonstrate the effectiveness of the network programming interfaces, I describe a distributed wavelet compression algorithm which can effectively use the interfaces to optimize the communication. Multi-resolution data analysis, processing and compression is useful for various sensor network applications. A lot of previous work on wavelet-based processing in sensor networks have assumed regularly-spaced data.

Recently, Wagner et. al. [54] have proposed a haar wavelet based multi-scale data analysis which enables irregular wavelet transform. In the bottom-up approach of that algorithm, all the sensors in the fundamental cell sends their sensor readings to the drum for that cell using *PutParent*. The locations and identifiers of the nodes are also available to the drum through the *Discovery* interface. The drum calculates the *scaling coefficient* describing the average reading of the cluster and the *wavelet coefficient* encoding the deviations from the average readings. These scaling coefficients are then passed up the hierarchy using the *Parent* interfaces, and similarly computed. In the top-down approach of the algorithm, querying is from the top-level and drilling down until the requisite resolution of the data is obtained. This is achieved using the *GetPeer* interfaces. Finest resolution is obtained if query goes down until the fundamental cell level.

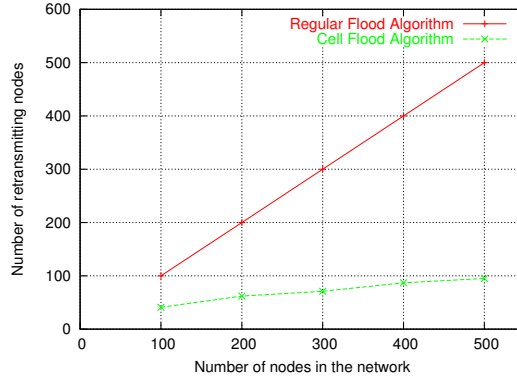


Figure 5: Number of retransmitting nodes with increasing network size

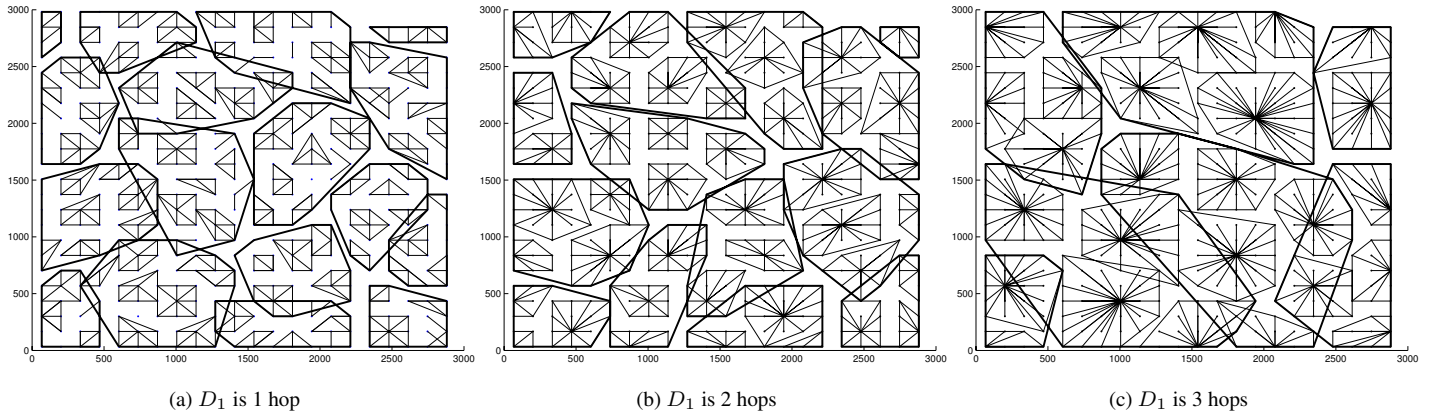


Figure 6: Example topology with different values of D_1

For any compression, maximum efficiency with minimal loss is achieved when many data points are similar enough to be represented with one data point. When compressing a field of sensor data, various regions in space might have similar readings. So, if there is one cluster for each region, the region might be efficiently represented by a single value. But, if one cluster encompasses two different regions with divergent values, compression is not so efficient. The reclustering technique is used here such that each cluster has similar values and hence efficient representation. Locally near the region boundaries clusters are aligned with the regions.

I plan to build a few toy applications which use these interfaces provided, to show their effectiveness and expressibility.

3.3 Initial Design Evaluation

I have performed an initial evaluation of the design of the architecture by simulating the adaptive overlay formation in the *ns-2* simulator.

3.3.1 Cluster Operations

In this section, I evaluate communication operations. In particular, I show the effectiveness of the Cell Flood Algorithm. To flood a particular cell with or without Selectors, this algorithm builds an approximate Minimum Connected Dominating Set. Thereby, the number of forwarding nodes is reduced without compromising the quality of the flood. Figure 5 shows the number of retransmitting nodes with increasing network size. The area of the network is kept constant while increasing the number of nodes, thereby increasing the node density. For a regular flooding algorithm, where each node forwards a packet exactly once, the number of forwarders is exactly equal to the number of nodes. In the Cell Flood Algorithm, the number of retransmitting nodes grows very slowly with increasing network size. The percentage of retransmitting nodes decreases with increasing network size, thereby showing good scalability.

3.3.2 Hierarchy Formation

In this section, I show the effectiveness of the hierarchy formation. Figure 6 shows the hierarchy formed for an example topology. 500 nodes are evenly distributed in an area of 3000 meters by 3000 meters. The radio range is taken to be 250

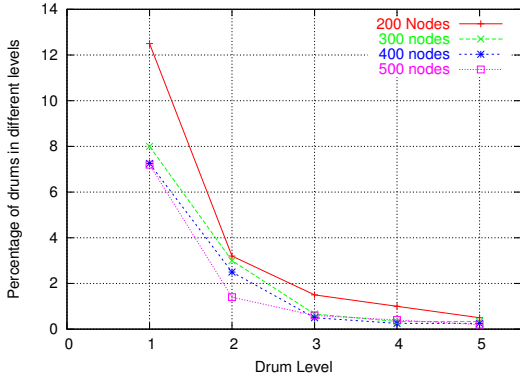


Figure 7: Percentage of nodes which are drums are various levels

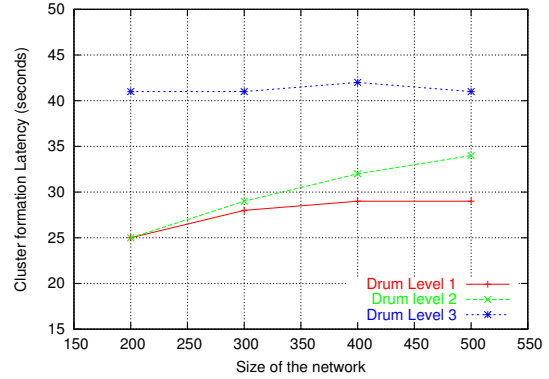


Figure 8: Cluster formation latency with increasing size of network

eters. Each part of Figure 6 shows the first and second level clusters formed, along with rays connecting each node to its parent. Figure 6(a) shows the clustering for \mathcal{D}_1 equal to 1 hop, Figure 6(b) for \mathcal{D}_1 equal to 2 hops, and Figure 6(c) for \mathcal{D}_1 equal to 3 hops. As the number of hops allowed in the fundamental cell increases, the size of it increases along with decrease in the number of levels in the hierarchy.

Figure 7 illustrates the number of drums at each level of the hierarchy as a percentage of the total number of nodes in the network. This is shown for increasing networks sizes, with the node density kept constant. As shown in the analysis in Equation 2, the number of drums in increasing levels decreases quadratically. Larger the network, lesser is the percentage of drums at each level. In this scenario, the value of α , which is the ratio between the beacon hop limits for consecutive drum levels, is taken as 2.

Figure 8 shows the latency for cluster formation with increasing network size. Here again, in all the networks sizes, the density is kept constant. The higher the level of drum, the longer it takes to stabilize. This is because the higher level drums are further spread apart and have larger beaconing intervals, which makes any change in higher levels propagate more slowly. The startup latency increases slowly with increasing network size. For the scenario sizes experimented with, the drum level 3 for all network sizes get stable at the same time. The cost of cluster formation arises from the beaconing during this phase, and hence is directly proportional to the length of time it takes to stabilize.

This also illustrates the effect of local change for adapting the hierarchy to communication requirements. The local clustering changes take place at a lower latency as shown in the figure. Therefore adaptivity of the clustering can be achieved with low latency, and hence also with low cost.

4 Medium Access Scheduling

In this section, I propose a Medium Access Scheduling protocol for the multi-scale architecture which is energy efficient and has strict latency bound. I take advantage of the sensor network characteristics — periodic nature of communication, limited communication abstractions, and fusion function characteristics. I design a Medium Access Control protocol which is adaptive to the application requirements and optimized for energy usage as much as possible. Since, no one single MAC protocol will be well-suited for all sensor applications [21], the network services need to adapt themselves to the application-specific requirements, presented through the IES module. I design a Time Division scheduling such that nodes are guaranteed to have a collision-free 2-hop neighborhood during their allocated slot. Time is divided into Contention Free Period for periodic a priori known traffic and Contention Access Period for event driven traffic.

4.1 Background

I explore the design space for the adaptation and optimization of the proposed MAC layer. In shared medium networks, one of the fundamental tasks of a MAC layer is to avoid collisions between two interfering nodes. It allocates the channel to the nodes efficiently, so that each node can communicate with a bounded waiting time and with as little overhead as possible. The important attributes for traditional MAC are fairness, latency, throughput and bandwidth utilization. In contrast, the important attributes of a MAC protocol for WASN are energy efficiency and scalability towards size and topology change. The major sources of energy wastage as elaborated in [61] are:

- *Collision*: Collision results in corruption of a packet and subsequent retransmission leading to increased energy consumption as well as latency.

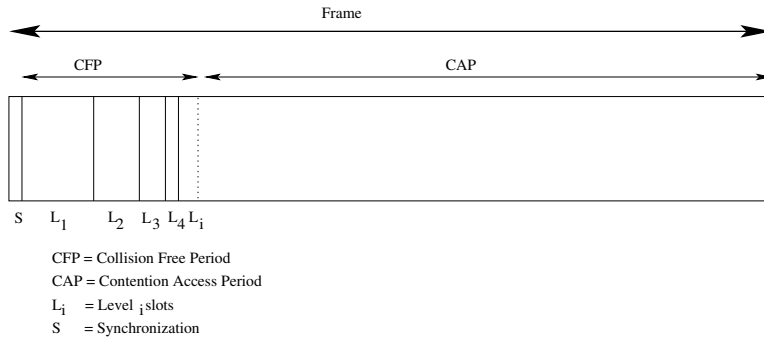


Figure 9: Frame Diagram

- *Idle listening & overhearing*: Listening for either possible packets or packets destined for other nodes leads to wastage of energy. Idle listening consumes significant energy comparable to actually receiving a packet.
- *Control packets overhead*: Increased control overhead leads to increased energy usage in direct proportion.

Though the proposed MAC protocols [52, 61] for WASN have identified and addressed many of the WASN *environment* requirements, they have not taken advantage of the nature of WASN *applications*. The following properties of WASN applications can be exploited for the MAC protocol design:

- Communication requirements may be *periodic* and known beforehand such as collecting temperature statistics at regular intervals. This information can be used to schedule the medium access by the nodes and thus minimize collisions, as well as aid the radio interfaces sleep/wake-up decisions thus decreasing the idle listening and overhearing.
- A contention based medium access will also be necessary to support *event-driven* applications like intrusion or fire detection. The forwarding node can be woken up in time to process event-driven data, by making it application aware. Real-time constraints can be communicated from the application to adapt the MAC to meet its requirements.
- Often the sensed data packet may have fixed size. This simple information will help minimize doing packet fragmentation and assembly at MAC layer, thus improving the throughput [30].

An application-adaptable energy-aware MAC protocol can be build upon the previous sensor MAC protocols, that exploits the known application requirement. The MAC protocol should have two modes to support the two different communication requirements of sensor applications, namely periodic and event-driven. The need to support these two kinds of modes was recently proposed in the IEEE standard for low-power sensors [26]. The relative proportion of the two modes in a super-frame structure is dynamically determined by the applications depending on their current needs.

Periodic Contention Free Period: Medium access in this mode is of the class of Spatial Time Division Multiple Access (STDMA) [37] protocols. The application’s deterministic traffic distribution during the periodic communication can be used to compute an efficient slot allocation policy [7, 19]. The length of the slot is enough to send a complete data packet of fixed size, allowing TAG like applications to send periodic sensor readings.

Event-Driven Contention Access Period: During this mode, a sensor will be in sleep-mode except when necessary to communicate. For sending event-driven data like the image scan of a specific location, this mode of communication is used to send the data to the next fusion node. The mode is based on IEEE 802.11 protocol [25] with carrier sense and RTS-CTS. Techniques such as overhearing of neighbor’s NAV vector [50] to save energy, and sacrificing per-node fairness for lesser collision [61] is utilized in this mode.

4.2 Medium Access Scheduling in a hierarchy

I sketch the design of the medium access protocol for the hierarchically clustered network - COMPASS [40]. My Medium access protocol is specifically designed for supporting the communication abstractions in COMPASS. There is a conflict-free period for periodic apriori known traffic, and there is a contention-based period for nodes to send event-driven traffic, as well as introduce new periodic traffic. It is shown in Figure 9.

Assumptions:

- I assume that a link is bi-directional. It can be used for both sending or receiving packet, or no communication at all.
- I assume that the interference range is same as transmission and reception range. I can relax the assumption using techniques such as RID, where instead of 2-hop communication to reach interfering nodes, multi-hop forwarding is used to reach the interfering nodes. But, my theory of conflict free slot allocation still remains valid.

Algorithm 4. Centralized Algorithm for Intra-Cluster Scheduling

Require: k slots necessary per node known; Depends on fusion function and number of children

- 1: Traverse tree in Depth First fashion from cluster root
 - 2: Visit finish time forms a topological sort to give a partial order
 - 3: For each visited node, assign a minimum possible slot number s , with the following constraints
 - 4: **repeat**
 - 5: A. $s >$ maximum (slot number allocated to any child)
 - 6: B. $s \neq$ slot already allocated to any node within 2 hops
 - 7: **until** k consecutive slots are found
-

- I assume the presence of clock synchronization. Providing clock synchronization of necessary accuracy is solved in the next section.

The scheduling protocol I propose is specifically for multi-scale data retrieval applications, where data flows from the sensors to the sink. This is the common case which I am optimizing. The medium access scheduling for the contention-free period is divided into two parts: For periodic and apriori known traffic, we design a TDMA-style protocol; and leverage the existing literature on energy-efficient CSMA protocols [58, 61, 52] for the contention-access period.

4.2.1 Intra-Cluster Scheduling

In this scheduling, I allocate spatially conflict-free slots for each node within a cluster to gather data in the lowest level drums. I assume the presence of a tree-like hierarchy within each lowest-level cluster for routing purposes. Trees like these can be achieved with proposals like TreeCast [39]. As the data flows upwards from the children towards their parent, the parents are allocated slots after the slots allocated to their children. This allows data from the lowest level sensor to flow to the top-level sink within one frame. This also produces a partial ordering of the nodes which I need to maintain. The slots allocated to the children should be close together, such that the parent can wake up for one contiguous time to receive data packet from its children, and then switch off the radio to save power. As switching between on and off state taken time and energy, contiguous allocation is beneficial. Treecast ensures that the trees rooted at each level are balanced, thereby ensuring all nodes have to handle roughly equivalent amount of data. The fusion functions at each node is known apriori through the IES module from the application. Fusion function characteristics is used by the protocol to determine the number of slots to be allocated to slots at each level. For example, for a fusion function like average sensed value, the parent needs to average its own value with the value received from its children and can send it upwards in a single slot. The number of slots to be allocated to a node per frame also depends on the application-determined periodic rate which is also known through the IES.

I describe next the centralized scheduling algorithm 4 to allocate conflict-free slots. Within the lowest level cluster, there is a cluster-head which coordinates the traffic from that cluster. It does a Depth-First Search(DFS) of the tree routed at itself, and sorts the nodes according to the finish times (of a DFS visit) to give a topological sort of the nodes within the cluster. The topological sort has the nice property that the parent has a finish time right after the finish times of its children in the tree. The slot times are assigned according to that given by the topological sort, thereby giving the desired partial order.

This procedure gives a collision-free slot allocation within a single cluster, but there can be a lot of parallelism of communication between the nodes which is not allowed, due to topological sort assigning a unique slot per node within the cluster. We use a distributed token-passing protocol 5 to compact the slot allocation. Each node holding the token determines whether it can choose an earlier slot without conflicting with any node in its two-hop neighborhood. Nodes maintain a two-hop neighborhood for this purpose.

This problem can be mapped to a graph coloring problem. The network is taken as a graph (G), and a graph G^2 is produced. G^2 has the same vertex set as G , and all the pairs of vertices which are 2-edge reachable in G have an edge in G^2 . Thus, G^2 gives the interference graph exactly similar to our node scheduling problem. Coloring a graph is a NP-Complete problem even in centralized setting, and lots of approximate coloring solutions have been proposed in graph theory literature. However, due to the constraints on slot allocation that partial ordering induces, none of these solutions are directly application to my scheduling scenario. Also, my graph being that of a wireless network is an Unit Disk Graph (UDG) having special properties. In a UDG, links between nodes exist if the distance between the nodes is less than the radio range. Competitive ratio is the ratio between the chromaticity given by approximate algorithm, when compared to the optimal solution. I have proved that the lower bound of this competitive ratio is 1.2 for my algorithm, using an example worst-case graph. I have also showed an upper bound on the competitive ratio of 6, using properties of an UDG. These small constant bounds on the competitive ratio is a good indication of the optimality of my solution. The time and cost for the scheduling protocol are both $O(N)$ where N is the number of nodes in the smallest level cluster. The clustering hierarchy ensures that the number of nodes in a cluster is constant, depending on the density of nodes. So, the algorithm is both constant in time and cost with the number of nodes in the network.

Algorithm 5. Distributed Protocol for Intra-Cluster Scheduling

Require: k slots necessary per node known; Depends on fusion function and number of children

Require: Each node knows its 2-hop neighborhood

Require: Each node knows the cluster it belongs

Require: Each node knows its parent and its children

```
1: For all token packets overheard, a node remembers slots its neighbor is sending or receiving
2: while All children of a node not visited do
3:   Send token to one of the children not visited
4:   Block till token comes back
5:   After unblocking, remember the slot allocation of the child
6:   Put into the token packet any slot the node's reception will be interfered with due to a nearby cluster communication
7: end while
8: All nodes in the subtree rooted at the node has been visited
9: while Assign  $k$  slot numbers starting from  $s$ , with constraints do
10:   A.  $s >$  maximum (slot number allocated to any child)
11:   B.  $s \neq$  slot already allocated in token packet
12:   C.  $s \neq$  slot overheard being allocated by another node in 1-hop neighborhood
13: end while
14: Append the node id, its chosen sending slot and its send mode (parent or all) to the token
15: Append the node id and receiving slots to the token
16: Send token to its parent with flag stating SUBTREE_DONE
17: while Token not received from parent with flag SUBTREE_DONE do
18:   Block
19: end while
20: Broadcast the final token packet locally
```

4.2.2 Inter-Cluster Scheduling

Inter-cluster scheduling occurs after the data has reached the cluster-heads at the lowest cluster. Each level of communication is separated into time, and hence does not conflict with each other. This is shown in Figure 9, where the CFP is divided into multiple periods for each level.

For inter-cluster communication between peers, the medium access protocol utilizes knowledge of the routing path. Beacons transmitted by cluster-heads at all levels of the hierarchy by the routing protocol gives the routing path for communication between peers at different levels. A packet reserving the channel is sent for the periodic traffic along the routing path given by the beacons. Each node which is used allocates slots in a conflict-free fashion such that two different peer-paths do not conflict in time, while ensuring maximum parallelism in the communication paths.

5 Clock Synchronization

In this section, I provide a clock synchronization service which is necessary for TDMA scheduling to work, as well as for a majority of sensor applications which need synchronization. The need for accuracy of synchronization however varies from application to application and with time. My adaptive clock synchronization provides the synchronization exactly required for the applications, and does not attempt to provide higher than necessary granularity. The synchronization needs of the application and other protocols are communicated to the IES module. The IES module configures the synchronization service to provide the necessary synchronization at any specific time. In the work, I derive expressions to convert service specification (maximum clock synchronization error and confidence probability) to actual protocol parameters (minimum number of messages and synchronization interval).

As in any distributing computer system, clock synchronization is an important service in sensor networks. Sensor network applications can use synchronization for data integration and sensor reading fusion. A sensor network may also use synchronization for TDMA medium access scheduling, power mode energy savings, and scheduling for directional antenna reception. Sensor networks show some unique characteristics that make it difficult to directly apply traditional network clock synchronization approaches.

The error in clock synchronization comes mainly from the non-deterministic random time delay for a message transfer between two nodes. Kopetz et al. [31] first characterized this message delay. In some recent work, this non-determinism has been reduced to provide tighter bounds on the clock synchronization error [53, 13]. My work, which is based on the Reference Broadcast Synchronization (RBS) [13], provides an analytical way to convert service specifications to protocol parameters.

The need in sensor networks is to provide the best possible clock synchronization under existing circumstances, given the limited resources of the nodes and the network in the system. Highly accurate clock synchronization typically requires more message transfers and processing as part of the synchronization protocol. In situations where the system energy is extremely low, it might not be possible to provide high accuracy. The accuracy needed in clock synchronization is variable, depending on the higher layer application requirements. If the need for determinism can be relaxed, probabilistic

guarantees often suffice for the needs of an application, while allowing for optimal use of resources. Probabilistic guarantees provide better accuracy in *most* cases as compared to deterministic accuracy in *all* cases. Also the failure probability of achieving a certain accuracy can be bounded. Quality of Service (QoS) in networks make extensive use of this concept to give probabilistic guarantees by allowing resources to be allocated on the basis of expected aggregate demand. Previous work [1, 10] uses this concept for providing probabilistic clock synchronization in distributed systems.

Clock synchronization might not be necessary at all times, except during sensor reading integration. In such a case, providing clock synchronization all the time will be a waste on the limited available resources of sensors. The sensor clocks can be allowed to go out of sync, and then re-synchronize only when there is a need for synchronization, thereby saving resources.

I design the probabilistic clock synchronization protocol keeping in mind the concepts discussed above. The remainder of this paper is organized as follows. In Section 5.1, I describe the various concepts of clock synchronization in detail and motivate the design principles used in building the algorithm. In Section 5.2, I survey the related work in this area. Section 5.3 my protocol. Section 5.4 evaluates the performance of the protocol. Section 5.5 extends this protocol for multihop sensor networks.

5.1 Design Principles

In the paper describing the Network Time Protocol (NTP), Mills [36] defines the various terms used in clock synchronization. The *stability* of a clock is how well the physical clock can maintain a constant frequency. *Accuracy* refers to how well the maintained time is true to the standard time. The *offset* of two clocks is the actual time difference between them, and the *skew* is the frequency difference between them. To *synchronize frequency* means to adjust the clocks to run at the same frequency, and to *synchronize time* means to set their time at a particular epoch to be exactly the same. To *synchronize clock* means to synchronize the clocks in both frequency and time. In this paper, our algorithm will *synchronize clocks*, i.e., it will synchronize both frequency and time.

5.1.1 Traditional versus Sensor Network Synchronization

Elson and Romer [14] point out the basic features that are common to clock synchronization protocols in general. These are as follows:

- Synchronization has to be provided over a connectionless messaging protocol between nodes.
- Synchronization has to be achieved with the exchange of clock information between clients and one (or a few) servers.
- The protocol has methods to reduce or predict the non-determinism in message delivery and processing.
- The protocol assumes the presence of an algorithm to update local clock at client based on received value.

NTP is scalable, robust to failures, self-configuring, and has various properties that are needed in the sensor network world. However, wireless sensor networks pose a number of challenges beyond traditional network systems. Elson and Romer describe the differences quite exhaustively. I reiterate these differences, as they are important from the design point of view:

- **Energy Constraint:** Energy efficiency is very important for sensor networks as opposed to traditional networks. The basic assumptions of the nodes having steady power supply, free listening to network traffic, and network transmissions being inexpensive, do not hold for sensor networks. In sensor networks, listening to the network interface and sending packets take significant energy. Also, the CPU is powered down in sensor networks for much of the time, breaking the traditional assumption of the availability of CPU whenever necessary. Synchronization is not required to be maintained at all times, rather only when required by the application. Hence, doing traditional synchronization in a sensor network would require the CPU and network interface to be powered up for significant amounts of time, thereby having a large resource overhead.
- **Tunable Accuracy:** Traditional synchronization protocols try to achieve the highest degree of accuracy possible. The higher the level of accuracy required, the higher the resource requirement. The accuracy of required synchronization depends on the application requirement. Using resources for high accuracy even though lower accuracy is enough for the application wastes the limited resources of sensor networks. Therefore, there is a need for a trade-off between resource requirements and accuracy, depending on the need of the application and resource availability of the system.
- **Non-determinism:** Sensor networks are dynamic systems with considerably higher rate of failures of the individual nodes than in tradition networks. These nodes can also be mobile. This gives rise to a greater non-determinism

for communication delays than that present in traditional networks. Typically, NTP has the need for some manual configuration of peers and upstream nodes.

Thus the synchronization protocol needs to be more robust to failures and also to the greater variability in communication delay.

- **Multihop:** Most of the typical synchronization protocols have a highly accurate clock present in the LAN, such that all the nodes in the system can directly exchange messages. Sensor networks span many hops, with higher jitter. So the algorithms for sensor network clock synchronization need to have some sort of localization to reduce this error, as well as some other means of achieving multihop synchronization even in the presence of high jitter.
- **Server-less:** Traditional protocols have specified servers, with multiple accuracy levels which are sources of accurate time. Sensor networks do not have any external infrastructure present and can be large in scale. Maintaining global time scale in this network is thus harder, if no external broadcast source of global time such as GPS is present. Elson et al. [14] proposed that each node maintain an undisciplined clock, augmented with the relative frequency and phase information of its neighbors. I also use this approach in my work.

5.1.2 Theoretical Bounds on Clock Synchronization

There have been various theoretical results that have been proven regarding clock synchronization. These analytical results and their consequences are useful when designing a clock synchronization protocol:

From the causality property in a system, the ordering of events can be formally stated. If an event a occurs before another event b , then a should happen at an earlier time than b . Let $C_i(a)$ be the clock value of process i when event a occurs. Then it can be formally stated that:

If a and b are events in process i , and event a occurs before event b , then $C_i(a) < C_i(b)$.

Lamport [33] showed that, when the value of a clock needs to be adjusted, it always has to be set forward and never back. Setting the clock back could cause the above condition to be violated. Hence, in an ideal system, the slower clocks needs to be adjusted to the value of the fastest clock, for all clocks to be synchronized. This restriction will also maintain the partial ordering of the events.

It is useful to have a bound on the best accuracy achievable in any system, such that no bound lower than that is specified. Srikanth et al. [51] have shown that for any synchronization algorithm, even in the absence of faults, the bound on the rate of drift of logical clocks from real time is greater than the bound on the rate of drift of physical clocks. In the presence of faults — such as message losses and node failures — the accuracy of logical clocks becomes even worse.

Most clock synchronization algorithms proposed in literature try to guarantee a deterministic upper bound on the clock skew. Lundelius et al. [34] derived a theoretical limit on the best achievable clock skew that deterministic algorithms can guarantee. They show that the upper bound on clock skew that can be deterministically guaranteed by any clock synchronization algorithm can be no smaller than $(d_{max} - d_{min})(1 - 1/n)$, where n is the number of nodes in the system, and d_{max} and d_{min} are the maximum and minimum value of message delays in the system, respectively. Significantly smaller bounds on the upper bound of clock skew can be achieved if the condition of determinism is loosened. If the deterministic guarantee is replaced with a probabilistic guarantee, where the probability of failing can be bounded, it might suffice for many applications.

This probabilistic guarantee is specially significant for sensor networks, as sensor networks inherently assume a level of redundancy present in the number of nodes and are fairly robust to failures. This approach might entail saving a significant amount of resources in the sensor network, while giving some guarantees. I thus develop a probabilistic clock synchronization protocol for sensor networks, basing my work done previously by Christian [10] and Arvind [1].

5.1.3 Sources of Clock Synchronization Error

The biggest source of error in synchronization algorithms stems from non-determinism in message delivery latency. In an effort to reduce this non-determinism, I review the sources of this latency. Kopetz et al. [31] have characterized the message delivery latency into four distinct components:

- **Send Time:** The time spent at the sender to build the message.
- **Access Time:** This is the delay incurred when waiting in the network interface for access to the transmission channel.
- **Propagation Time:** This is the time needed for the message to propagate from sender to receiver over the wireless medium.
- **Receive Time:** This is the time needed for processing at the receiver's network interface.

Most clock synchronization algorithms go to great lengths to reduce the above non-determinism in message delivery. A significantly different approach has been taken in the CesiumSpray system [53]. CesiumSpray takes advantage of the inherent broadcast nature of the wireless medium. The Send Time and Access Time are unknown and highly variable. However, for a set of receivers listening to a common sender, those times are identical for all of the receivers. The only variable time is the Propagation Time and Receive Time, which are much smaller in value. This approach entails synchronizing a set of receivers with each other, in contrast to synchronizing with the sender. I use this idea to significantly reduce the sources of error in my clock synchronization protocol.

5.1.4 Models of Clock Synchronization

There are many different types of clock synchronization. Each type has its usage. They are as follows:

- **Global clock:** There is a precise global time, UTC, which is maintained by atomic clocks in standard laboratories. Traditional internet clock synchronization algorithms try to maintain this global time in all computer systems. Maintaining this time in sensor networks is significantly harder. Sensor networks also do not typically need this strict clock synchronization. In this work, I can provide this service with a certain degraded accuracy if a GPS is available. But, maintaining this time is not the thrust of this work.
- **Relative clock:** This is the relative notion of time within the sensor network. Each node is synchronized with every other node with a time which might be totally different from UTC. This suffices for most of the applications of clock synchronization that I have described earlier. In this work, I provide this synchronization with an accuracy which is bounded with a tunable confidence probability.
- **Relative notion of time:** Time can also be maintained between nodes, not with real time, but with some logical notion. This logical notion need not match with physical clock. For example, two nodes might need to trigger a alarm 10 time units after some event has occurred (time unit might not be seconds). This sort of time is very often enough for a variety of applications. But, providing the previous type of clock synchronization also automatically provides this type of clock synchronization. So, I indeed provide this notion of clock synchronization.
- **Physical ordering:** In many cases precise times might not be important, but what is important is the ordering of events. If the system can state whether an event occurred before or after another, that is enough. This type of synchronization simply involves ordering of events in some partial or total order. Having previous clock synchronization types automatically provide this type. However, I note that this might be significantly cheaper in terms of resources to achieve. So, if application need only this type of synchronization, alternate means should be used to preserve scarce resources.

Another classification is in terms of the initiator of synchronization procedure. In this work, I provide all three types of synchronization below, depending on the system need. They are as follows:

- **Always On:** In this model, clock synchronization between nodes is always present. Many applications like TDMA scheduling might need this model. This model is the model of traditional clock synchronization protocols. But, maintaining this model might present a significantly higher overhead, if the applications require clock synchronization rarely.
- **Sensor Initiated:** In this model the sensor nodes decide whether to have synchronization or not. They synchronize between themselves or a subset of the nodes, whenever necessary. This is useful when the need for synchronization is not required frequently. But it might entail a certain degree of latency before synchronization can be achieved. So the applications will need to tolerate this latency.
- **Outsider Initiated:** This is similar to the previous model. But here the initiator of clock synchronization is somewhere outside the sensor network, for example a control center. This model degrades to the previous model though, once the message to start synchronization has reached from the outsider to the necessary sensor nodes. So, this is essentially same as the previous one and I will not treat this as different.

5.2 Related Work

5.2.1 Traditional Clock Synchronization Protocols

As mentioned earlier, most traditional clock synchronization protocols share the same basic design: a connection-less messaging protocol, exchange of clock information between client and server(s), methods to reduce the effects of random non-deterministic communication delay, and a method to upgrade the client time based on the information from the server. NTP is widely deployed in the internet, since it is scalable, robust and has good performance. It consists of various levels

(or stratum) of servers in a hierarchy providing synchronization to the clients which are leaves in a hierarchical tree. These protocols cannot be applied directly to sensor networks because of the differences pointed out in Section 5.1.1.

5.2.2 Wireless Clock Synchronization Protocols

There have been a few synchronization protocols which are specifically for wireless or ad hoc networks. Romer [49] proposed a scheme for sparse ad hoc networks. The algorithm does not synchronize the computer clocks of the nodes, but generates time-stamps which is used by the unsynchronized clocks to transform the message time-stamp. As a message moves from hop to hop, each node transforms the message time-stamp to its local time-stamp with some introduced error. This error increases with the number of hops. Also, the protocol uses round trip delays, the calculation of which will have the typical errors associated with estimating round trip time.

Huang et al. [24] showed that the 802.11 MAC time synchronization protocol is not scalable for large number of nodes. They proposed a simple modification to the MAC protocol which maintains synchronization among nodes in a single broadcast region.

5.2.3 Receiver-Receiver Synchronization

A couple of previous works use a completely different approach than the traditional approaches. They synchronize a set of receivers among themselves. This reduces much of the message delivery latency non-determinism associated with traditional protocols. CesiumSpray [53] was the first to use this idea. It is a hybrid external/internal synchronization protocol. It uses a two-level hierarchy to improve scalability.

My work is based on the Reference Broadcast Synchronization (RBS) [13]. Least-squares linear regression is used to find the relative frequency of the clocks. RBS uses post-facto synchronization to synchronize two nodes' clocks by extrapolating backwards to estimate the phase offset at any previous time. It extends the work to do multihop clock synchronization. My protocol is similar to this work in terms of protocol in the single-hop case, other than some minor improvements. However, RBS does not contain any analysis on the number of reference broadcasts necessary, or the frequency of reference broadcasts. I analyse these issues and provide a probabilistic bound on the maximum error. I also relate this probability with the number of reference broadcasts required. RBS also has more overhead in terms of exchanging information between the receivers. It assumes a single broadcast region for the sender and all the receivers, which is not the case. Two receivers, lying in the broadcast range of a sender, might not be able to exchange messages since they might be out of range of each other. For multihop synchronization, unlike RBS, my protocol does not require all sensor nodes to be within one hop of at least one sender.

5.2.4 Probabilistic Clock Synchronization

There are a few synchronization protocols that are probabilistic in nature. The clock skew that a probabilistic protocol guarantees, has a probability of invalidity associated with the guarantee. However, the probability of invalidity can be bounded. All the probabilistic algorithms proposed are for server-client architecture, where the clients try to synchronize with the clock of the server. This is fundamentally different from my approach where we synchronize amongst receivers.

However, I adapt the idea of the probability analysis techniques proposed in the following two papers. The idea of probabilistic protocol was proposed by Christian [10]. He realized that guarantee cannot be provided when unbounded message delays are possible or messages can be lost. Hence, his principle is to retry sufficient number of times to read the clock of another process with a given precision with probability as close as desired. However, there are some fundamental limitations to the accuracy that can be achieved. His algorithm reads the remote server clock within the specified precision. It repeats these attempts till a reply comes back within a certain desired interval of time. The lower the round trip time for a reading attempt and its reply to come back, the higher the accuracy achieved in reading the clock of the remote server. The average number of messages to reach synchronization is $2/(1-p)$, where p is the probability of failure of message delivery within a fixed period of time. This process is repeated k times, such that the probability of reaching synchronization is $1-p^k$. By choosing a large enough k , the probability can be made arbitrarily close to 1.

Arvind [1] proposed another probabilistic synchronization protocol. It has two main parts—the Time Transmission Protocol (TTP), by which the clock value of the sender is read by the receiver, and Probabilistic Clock Synchronization (PCS), which uses this value to adjust the receivers clock. In TTP, the sender sends a sequence of n synchronization messages, each having the sender timestamp. The receiver adjusts the clock value using these n timestamps. The synchronization procedure is repeated every interval of time. He analytically showed the minimum number of messages necessary for achieving a given maximum synchronization error. This paper has less overhead than Christian's paper, if the physical medium is considered to be broadcast.

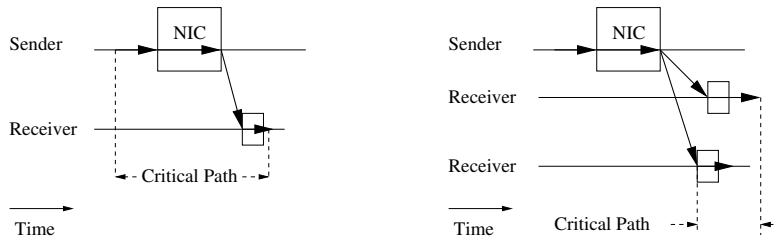


Figure 10: Types of Clock Synchronization [13]: Sender-Receiver Synchronization and Receiver-Receiver Synchronization

5.3 Single-Hop Clock Synchronization

In this section, I describe the technique of receiver-receiver synchronization used in my protocol. Next, I present the extension of this technique to probabilistic synchronization for use within a single broadcast region. I then analyse it mathematically to find the probabilistic bounds.

5.3.1 Receiver-Receiver Synchronization Error

The sources of non-deterministic error in message latency have been described in Section 5.1.3. In RBS and CesiumSpray, this non-deterministic error was significantly reduced by synchronizing among the receivers, instead of synchronizing between the sender and receivers. In receiver-receiver synchronization, the only remaining sources of non-determinism are the Propagation Time and Receive Time. Propagation Time is very small, considering the range of the sensors and the speed of radio waves. Receive Time is much more deterministic and can be bounded by time-stamping the receiving time at the hardware level [31, 13]. If the total error previously was ϵ_{sr} , for sender-receiver synchronization, and the error in this case is ϵ_{rr} , for receiver-receiver synchronization, then $\epsilon_{sr} \gg \epsilon_{rr}$. These two different ways of synchronization can be seen in Figure 10, showing the reduction of error in receiver-receiver synchronization.

The receiver-receiver synchronization works in the following manner. The sender sends a reference pulse at any time. Each receiver marks when it received the reference pulse according to its local clock. All the receivers exchange with each other the time of reception of the reference pulse. Since each receiver assumes that the pulse should have been received by all other receivers at approximately the same real time, a receiver *A* is able to estimate the offset of its clock with respect to another receiver *B* which has exchanged information with *A*. The number of reference packets may be increased in order to get better synchronization.

Elson et al. [13] have found the distribution of the synchronization error among a receiver set. Multiple pulses are sent from the sender to the set of receivers. The difference in actual reception time at the receivers is plotted. As each of these pulses are independently distributed, the difference in reception times gives a Gaussian (or normal) distribution with zero mean.

Given a Gaussian probability distribution for the synchronization error, it is possible to calculate the relation between a given maximum error in synchronization and probability of actually synchronizing with an error less than the maximum error. If the maximum error that I allow between two synchronizing nodes is ϵ_{max} , then the probability of synchronizing with an error $\epsilon \leq \epsilon_{max}$ is given from Gaussian distribution property.

$$P(|\epsilon| \leq \epsilon_{max}) = \frac{\int_{-\epsilon_{max}}^{\epsilon_{max}} e^{-\frac{x^2}{2}} dx}{\sqrt{2\pi}}$$

So, as the ϵ_{max} limit is increased, the probability of failure ($1 - P(|\epsilon| \leq \epsilon_{max})$) decreases exponentially. I use this observation in the analysis below.

5.3.2 Description of the Protocol

In this section, I present my protocol, extending the deterministic RBS protocol to provide probabilistic clock synchronization. The frameworks for providing external synchronization with UTC and for providing relative synchronization among the nodes are different. In this work, I concentrate on providing relative synchronization, as it is sufficient for most sensor network applications.

- For external synchronization, I assume the availability of GPS in a subset of the nodes. These nodes will be senders of synchronization messages. The sensor nodes will synchronize with these GPS receivers using any of the many sender-receiver probabilistic algorithms proposed previously [1, 10]. Obviously, the synchronization error will be more than receiver-receiver synchronization because of the reasons pointed out in the Section 5.3.1.

- For relative synchronization, I use receiver-receiver algorithm. The basic framework for providing the *Always On* or *Sensor Initiated* model is the same. So, I present and analyse them together, pointing out the differences as and when necessary. The steps in this protocol are described next. The subset of nodes chosen as senders is random among the set of sensor nodes. If the relative density of sensor sender nodes and all sensor nodes is above a threshold, and if I assume uniform distribution of the sensor nodes, then the presence of sender sensors in every broadcast region can be assumed. But, this assumption is not necessary, as we will see in section 5.5, when I extend this protocol for multihop synchronization.

The following happens for every sender sensor in a single-hop broadcast region. A particular sensor being in the broadcast region of two senders will do all of the steps below separately for each sender. When synchronization is necessary in a sensor-initiated model, the sensors needing synchronization send out a *REQUEST*. This request is broadcasted till it reaches a sender sensor, which starts a cycle of the algorithm. In the *Always On* model, each sender sensor starts the cycle and repeats it periodically (period is determined analytically in Section 5.3.3).

1. A sender broadcasts n reference packets to its neighbors. Each packet contains two counters, one showing a cycle number, and another the reference packet number in the current cycle. The interval between each packet is fixed and greater than some minimum, such that they are independent of each other.
2. Each receiver records the time according to its own local clock, when each of these reference packets are received. Using these time-stamps, the receiver uses linear regression to fit a line on these data. The slope of the line will approximate the relative clock skew between the receiver and the sender.
3. Each receiver sends back to the sender, a packet containing the slope of the line and one point on that line. The sending back of these packets are jittered over an interval so that the packets sent back by different receivers have less chance of colliding with each other.
4. The sender composes all these slopes together, and broadcasts a packet containing its relative clock skew slope to all the receivers who have replied back.
5. Each receiver after receiving this packet, can now calculate its own slope relative to all the receivers in the broadcast region of a particular sender. So, for every pair of receivers, within the broadcast region of the sender, the clock skew and clock offset are now known with some synchronization error. The Send Time and Access Time errors are factored out when calculating this relative slope, as that error is the same for any two receivers. The only error present will be that due to propagation time and receive time.

5.3.3 Mathematical Analysis

The preceding section shows how this algorithm keeps the clocks of sensor nodes synchronized. In this section I will analyse the probabilistic guarantee of achieving the desired synchronization skew. I will derive how to convert the service specifications (maximum clock synchronization error and confidence probability) to actual protocol parameters (minimum number of messages and synchronization interval).

Synchronization overhead The error among the receivers is a normal distribution, as described in Section 5.3.1. Normal distribution makes it easier to study a distribution statistically. The following theorem gives a relation between the synchronization error and its associated probability, with the message overhead.

For n synchronization pulses from the sender, the receivers exchange their observations via the sender. As explained earlier, the slope of the skew between the receivers is found by a least square linear estimation using the n data points. The calculated slope of the skew has an associated error in it. This error is the difference in phase between the calculated slope and the actual slope. As the points have a Gaussian distribution, this error can be calculated. As the number of data samples, n , increases, this error reduces.

Using the Gaussian distribution $N(\mu, \frac{\sigma^2}{n})$ of Z , the synchronization error function can be found. Theorem 1 gives the relation between the synchronization bound with associated probability, and the minimum number of messages necessary. As the number of messages is the overhead of the protocol, this also gives the resource requirement to achieve a specified synchronization bound. This is the error in synchronization at the time when synchronization is done.

$$\textbf{Theorem 1: } P(|\epsilon| \leq \epsilon_{max}) = 2 \operatorname{erf} \left(\frac{\sqrt{n} \epsilon_{max}}{\sigma} \right)$$

where ϵ is the clock skew at synchronization, ϵ_{max} is the maximum specified clock skew at synchronization point, n is the minimum number of synchronization messages to guarantee the specified error, and σ is the variation of the distribution.

Proof: The proof can be found here [41].

Synchronization interval The previous theorem specified the minimum synchronization error, given the number of messages. That was the error in synchronization precisely when synchronization was done. In the next theorem, given a maximum specified clock skew, a time period within which re-synchronization has to be done is derived.

$\frac{\epsilon_{max}}{\sigma}$	Probability	Number of messages
0.5	0.95	16
0.5	0.99	28
0.5	0.999	44
1.0	0.95	4
1.0	0.99	7
1.0	0.999	11
2.0	0.95	1
2.0	0.99	2
2.0	0.999	3

Table 1: Variation in probability and number of messages for different values of $\frac{\epsilon_{max}}{\sigma}$

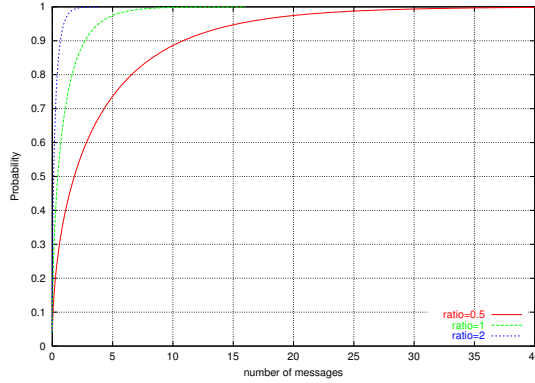


Figure 11: Probability of achieved error being less than a maximum

Theorem 2: $\gamma_{max} = \epsilon_{max} + (T_{sync} + \sigma_{max})\rho$

where γ_{max} is the maximum allowable synchronization at any point in time, T_{sync} is the time period between synchronization points for the Always On model (time period of validity for Sensor Initiated model), ρ is the maximum drift of the clock rate, and σ_{max} is the maximum delay (after the synchronization procedure has been started) in the time values of one receiver reaching another receiver.

Proof: The proof can be found here [41].

5.4 Evaluation

Theorem 1 gives the *probability of achieved error being less than a maximum specified error* as

$$P(|\epsilon| \leq \epsilon_{max}) = 2 \operatorname{erf} \left(\frac{\sqrt{n} \epsilon_{max}}{\sigma} \right)$$

I can derive a relationship between n i.e. the number of messages and the achieved probability P . This relationship is plotted in Figure 11. Each of the three different curves corresponds to the $\frac{\epsilon_{max}}{\sigma}$ ratio of 0.5, 1.0, and 2.0. Thus, given an ϵ_{max} and the probability P of achieving an error $\epsilon \leq \epsilon_{max}$ I can find the required number of messages. As expected, if the ratio increases lesser number of messages are needed in order to achieve a desired probability i.e. $n \propto \left(\frac{\sigma}{\epsilon_{max}} \right)^2$. This proportionality suggests that once the ϵ_{max} has been specified, the number of messages required is very sensitive to the standard deviation. From Table 1, it is clear that for lower values for the ratio of $\frac{\epsilon_{max}}{\sigma}$ the number of messages required gets quite large. For example, to achieve a probability of 0.99 with $\frac{\epsilon_{max}}{\sigma} = 1$, the algorithm requires 7 messages whereas for $\frac{\epsilon_{max}}{\sigma} = 2$, the algorithm requires only 2 messages.

5.5 Multihop Synchronization

In the previous section, I presented the algorithm for achieving probabilistic clock synchronization among all receivers that are within a single wireless hop of a sender. In this section, I extend the algorithm to achieve clock synchronization between receivers that may be multiple hops away from a sender. This multihop extension is in contrast to the multihop extension proposed by Elson et al. [13] that assumes that all sensor nodes are always within a single hop of at least one

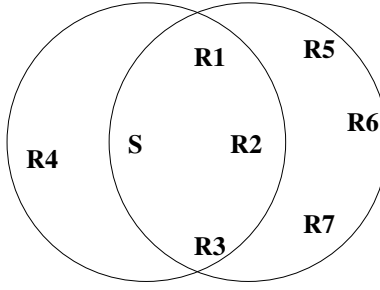


Figure 12: Multihop clock synchronization

sender. Also, in order for two sensor nodes (present in the broadcast regions of two different senders) to be synchronized, their algorithm requires the existence of a node that is within the broadcast region of both the senders. My algorithm does not make any such assumptions, and sensor nodes in this algorithm are allowed to be multiple hops away from a sender and still be synchronized with all other nodes within the nodes transmission range.

5.5.1 Protocol Description

For this protocol, I consider senders at various levels. A sender which does not need any synchronization (like the sender in Section 5.3) is called a sender at *level 0*. A sensor node which is within the broadcast region of a sender at level 0 can behave as a sender in order to synchronize sensor nodes which are two hops away from the sender at level 0. Such a sender is called a sender at *level 1*. This can be extended for multiple hops from the sender at level 0.

The receivers which are within the broadcast region of the sender at level 0 get synchronized in the same way as described in Section 5.3. Once these receivers get synchronized among themselves, each receiver starts behaving as a sender at level 1 and starts sending n reference broadcast packets. In order to avoid collision of reference broadcast packets, a sender at level 1 delays the transmission of its n reference packets until it does not hear the reference packets of any other sender at level 1. These packets are received by all sensor nodes which are within the broadcast region of a sender at level 1.

Consider the scenario presented in Figure 12. Nodes R1, R2, R3 and R4 are within the broadcast region of the sender S. Using the single hop synchronization protocol nodes R1, R2, R3 and R4 are synchronized among themselves. Suppose R2 gets to be the first node to end the reference broadcast, that is R2 starts behaving as a sender at level 1. By a similar synchronization procedure, R1, R3, R5, R6 and R7 get synchronized among themselves. Now suppose R6 needs to send a message to R4. The message would have to be routed through a node which is synchronized with R6, say R3. The assumption here is that due to the relative high density of sensor nodes, a node, such as R3 as shown in Figure 12, will exist in the broadcast region of two senders; the two senders might be at the same level or they might be separated by a single level. Now since R3 is synchronized with R4, R3 can transform the time reported by R6. Finally, since R3 is synchronized with R4, R4 can transform the time reported by R3. Hence, all along the routing path of the message suitable time transformations can be performed.

However, from the description of the protocol it seems that the protocol will have a very high overhead since the protocol essentially floods the entire network with reference packets. This can be easily fixed by causing time synchronization to be sensor initiated so that a sender (at any level) broadcasts reference packets only if the sender receives a request for synchronization from a sensor node in the local broadcast region of the sender. This ensures that a node does not broadcast reference packets if there is no sensor node in its local broadcast region that can listen to the reference packets.

5.5.2 Mathematical Analysis

The mathematical analysis of this protocol is similar to the mathematical analysis presented in Section 5.3.3. If ϵ_{max} is the maximum error between two receivers present in the broadcast region of a sender then the maximum error possible between two sensor nodes which are k hops apart is $k \cdot \epsilon_{max}$, and the average error is $\epsilon_{avg} = \epsilon_{avg} \cdot \sqrt{k}$.

Proof: The proof can be found here [41].

6 Research Plan

Following is the proposed schedule that leans towards optimism.

1. **May....** Complete the design description and graph evaluation of the scheduling protocol. Submit it to POMC for publication on May 30.
2. **June....** Implement the scheduling protocol into ns-2 to evaluate together with the multi-scale architecture.

3. **June-July....** Build a few toy applications using the multi-scale communication primitives provided to show it's effectiveness. Evaluate the performance of the architecture for these applications.
4. **July-August....** Final thesis write-up. Major thing extra from the proposal writeup are: the complete scheduling design specification, evaluation of the distributed scheduling protocol, and evaluation of the multi-scale architecture for a few applications with and without the TDMA scheduling.
5. **August-end....** Defend PhD thesis.

References

- [1] K. Arvind. Probabilistic Clock Synchronization in Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):474–487, May 1994.
- [2] Lichun Bao and J. J. Garcia-Luna-Aceves. A new approach to channel access scheduling for ad hoc networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 210–221. ACM Press, 2001.
- [3] Brian N. Bershad, Craig Chambers, Susan J. Eggers, Chris Maeda, Dylan McNamee, Przemyslaw Paradyak, Stefan Savage, and Emin Gun Sirer. SPIN - an extensible microkernel for application-specific operating system services. In *ACM SIGOPS European Workshop*, pages 68–71, 1994.
- [4] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected Dominating Set in Sensor Networks and MANETs. In *Handbook of Combinatorial Optimization (Editors D.-Z. Du and P. Pardalos)*, pages 329–369. Kluwer Academic Publisher, 2004.
- [5] Andrew Campbell, Geoff Coulson, Francisco Garcia, David Hutchison, and Helmut Leopold. Integrated quality of service for multimedia communications. In *INFOCOM (2)*, pages 732–739, 1993.
- [6] Benjie Chen and Robert Morris. L+:scalable landmark routing and address lookup for multi-hop wireless network. Technical Report MIT-LCS-TR-837, Laboratory for Computer Science Massachusetts Institute for Technology, 2002.
- [7] A.-M. Chou and V.O.K Li. Slot allocation strategies for TDMA protocols in multihop packet radio networks. In *Proceedings of INFOCOM 1992*, pages 710–716, 1992.
- [8] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures & protocols*, pages 200–208. ACM Press, 1990.
- [9] T. Clausen, P. Jacquet (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot. Optimized link state routing protocol (olsr). RFC 3626, October 2003. Network Working Group.
- [10] Flaviu Cristian. Probabilistic Clock Synchronization. *Distributed Computing*, 3:146–158, 1989.
- [11] David Culler, Scott Shenker, and Ion Stoica. Creating an architecture for wireless sensor networks. In <http://today.cs.berkeley.edu/SNA/>, 2004.
- [12] Shu Du, Muhammed Khan, Santashil PalChaudhuri, Ansley Post, Amit Saha, Peter Druschel, David B. Johnson, and Rudolf Riedi. Self-organizing hierarchical routing for scalable ad hoc networking. Technical report, Rice, March 2004.
- [13] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, December 2002.
- [14] Jeremy Elson and Kay Romer. Wireless Sensor Networks: A New Regime for Time Synchronization. In *First Workshop on Hot Topics in Networks*, Princeton, New Jersey, October 2002.
- [15] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole. Exokernel: An operating system architecture for application-level resource management. In *Symposium on Operating Systems Principles*, pages 251–266, 1995.
- [16] Marc E. Fiuczynski and Brian N. Bershad. An extensible protocol architecture for application-specific networking. In *USENIX Annual Technical Conference*, pages 55–64, 1996.
- [17] Deepak Ganesan, Ben Greenstein, Denis Perelyubskiy, Deborah Estrin, and John Heidemann. An evaluation of multi-resolution storage for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 89–102. ACM Press, 2003.
- [18] Jie Gao, Leonidas J. Guibas, John Hershberger, and Li Zhang. Fractionally cascaded information in a sensor network. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 311–319. ACM Press, 2004.
- [19] J. Gronkvist. Traffic controlled spatial reuse TDMA in multi-hop radio networks. In *Proceedings of 9th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1203–1207, 1998.
- [20] Tian He, John A. Stankovic, Chenyang Lu, and Tarek Abdelzaher. Speed: A stateless protocol for real-time communication in sensor networks. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 46, Washington, DC, USA, 2003. IEEE Computer Society.
- [21] John Heidemann, Fabio Silva, and Deborah Estrin. Matching data dissemination algorithms to application requirements. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 218–229. ACM Press, 2003.
- [22] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 146–159. ACM Press, 2001.
- [23] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104. ACM Press, 2000.
- [24] Lifei Huang and Ten-Hwang Lai. On the Scalability of IEEE 802.11 Ad Hoc Networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Lausanne, Switzerland, June 2002.

- [25] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.
- [26] IEEE Computer Society LAN MAN Standards Committee. *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std 802.15.4. The Institute of Electrical and Electronics Engineers, New York, New York, 2003.
- [27] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings, Sixth Annual Int. Conf. on Mobile Computing and Networking (MobiCOM '00)*, pages 56–67, Boston, Massachusetts, USA, 2000.
- [28] International Standard Organization. OSI - Basic Reference Model. ISO 7498, 1984.
- [29] Vikas Kawadia and P. R. Kumar. A cautionary perspective on cross-layer design. *IEEE Wireless Communications*, pages 3–11, february 2005.
- [30] C. A. Kent and J. C. Mogul. Fragmentation considered harmful. *WRL Technical Report 87/3*, 1987.
- [31] Hermann Kopetz and wilhelm Ochsenreiter. Global time in distributed real-time systems. Technical Report 15/89, Technische Universitat Wien, Wien Austria, October 1989.
- [32] Rajnish Kumar, Matthews Wolenz, Bikash Agarwalla, Jun Suk Sin, Phil W. Hutto, Arnab Paul, and Kishore Ramachandran. DFuse: Framework for Distributed Data Fusion. In *SenSys 2003*, Nov 2003.
- [33] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.
- [34] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the Third annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, Vancouver, Canada, 1984.
- [35] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Operating System Design and Implementation(OSDI)*, Boston,MA, Dec 2002.
- [36] David Mills. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.
- [37] R. Nelson and L. Kleinrock. Spatial-TDMA: A collision-free multihop channel access control. *IEEE Transactions on Computers*, 33:934–944, 1985.
- [38] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162. ACM Press, 1999.
- [39] Santashil PalChaudhuri, Shu Du, Amit Saha, and David Johnson. Treecast: A stateless addressing and routing architecture for sensor networks. In *Proceedings of the 4th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, April 2004.
- [40] Santashil PalChaudhuri, Rajnish Kumar, Richard Baraniuk, and David Johnson. Design of adaptive overlays for multi-scale communication in sensor networks. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2005.
- [41] Santashil PalChaudhuri, Amit Saha, and David B. Johnson. Adaptive clock synchronization in sensor networks. In *Proceeding of the Information Processing in Sensor Networks(IPSAN)*, Berkeley, CA, April 2004.
- [42] Seung-Jong Park, Ramanuja Vedantham, Raghupathy Sivakumar, and Ian F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 78–89. ACM Press, 2004.
- [43] Wei Peng and Xi-Cheng Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Poster at MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 129–130. IEEE Press, 2000.
- [44] K. Pentikousis. Tcp in wired-cum-wireless environments. *IEEE Communications Surveys & Tutorials*, vol. 3, no. 4, Fourth Quarter 2000.
- [45] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.
- [46] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, page 298. IEEE Computer Society, 2002.
- [47] Jan M. Rabaey and Robert W. Brodersen. PicoRadio: Communication/Computation PicoNodes for Sensor Networks. DARPA Final Report, Dec 2002.
- [48] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *Sensys '03: Proceedings of the first international conference on Embedded networked sensor systems*, pages 181–192. ACM Press, 2003.
- [49] Kay Romer. Time Synchronization in Ad Hoc Networks. In *Proceedings of the Second ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, California, October 2001.
- [50] Suresh Singh and C.S. Raghavendra. PAMAS: Power Aware Multi-Access Protocol with Signalling for Ad Hoc Networks. *SIGCOMM Computer Communication Review*, 28(3), July 1998.
- [51] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.
- [52] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 171–180. ACM Press, 2003.
- [53] P. Verissimo, L. Rodrigues, and A. Casimiro. CesiumSpray: A Precise and Accurate Global Time Service for Large-Scale Systems. *Journal of Real-Time Systems*, 12(3):243–294, May 1997.

- [54] R. Wagner, S. Sarvotham, and R. Baraniuk. A multiscale data representation for distributed sensor networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Philadelphia, March 2005.
- [55] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *NSDI*, pages 29–42, 2004.
- [56] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110. ACM Press, 2004.
- [57] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205. ACM Press, 2002.
- [58] Alec Woo and David E. Culler. A transmission control scheme for media access in sensor networks. In *Mobile Computing and Networking*, pages 221–235, 2001.
- [59] Jie Wu and Fei Dai. Broadcasting in ad hoc networks based on self-pruning. In *Proceedings of Infocom '03*, 2003.
- [60] Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 379–390. ACM Press, 2004.
- [61] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE.
- [62] W. Yuen, H. Lee, and T. Andersen. A simple and effective cross layer networking system for mobile ad hoc networks. In *Proceedings of PIMRC*, 2002.