CenWits: A Sensor-Based Loosely Coupled Search and Rescue System Using Witnesses

Jyh-How Huang
Department of Computer Science
University of Colorado,
Campus Box 0430
Boulder, CO 80309-0430
huangjh@cs.colorado.edu

Saqib Amjad
Department of Computer Science
University of Colorado,
Campus Box 0430
Boulder, CO 80309-0430
Saqib.Amjad@colorado.edu

Shivakant Mishra
Department of Computer Science
University of Colorado,
Campus Box 0430
Boulder, CO 80309-0430
mishras@cs.colorado.edu

ABSTRACT

This paper describes the design, implementation and evaluation of a search and rescue system called CenWits. CenWits uses several small, commonly-available RF-based sensors, and a small number of storage and processing devices. It is designed for search and rescue of people in emergency situations in wilderness areas. A key feature of CenWits is that it does not require a continuously connected sensor network for its operation. It is designed for an intermittently connected network that provides only occasional connectivity. It makes a judicious use of the combined storage capability of sensors to filter, organize and store important information, combined battery power of sensors to ensure that the system remains operational for longer time periods, and intermittent network connectivity to propagate information to a processing center. A prototype of CenWits has been implemented using Berkeley Mica2 motes. The paper describes this implementation and reports on the performance measured from it.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Design, Experimentation

Keywords

Search and rescue, sensor networks

1. INTRODUCTION

Search and rescue of people in emergency situation in a timely manner is an extremely important service. It has been difficult to provide such a service due to lack of timely information needed to determine the current location of a person who may be in an emergency situation. With the emergence of pervasive computing, several systems [12, 19, 1, 5, 6, 4, 11] have been developed over the last few years that make use of small devices such as cell phones, sensors, etc. All these systems require a connected network via satellites, GSM base stations, or mobile devices. This requirement severely limits their applicability, particularly in remote wilderness areas where maintaining a connected network is very difficult.

For example, a GSM transmitter has to be in the range of a base station to transmit. As a result, it cannot operate in most wilderness areas. While a satellite transmitter is the only viable solution in wilderness areas, it is typically expensive and cumbersome. Furthermore, a line of sight is required to transmit to satellite, and that makes it infeasible to stay connected in narrow canyons, large cities with skyscrapers, rain forests, or even when there is a roof or some other obstruction above the transmitter, e.g. in a car. An RF transmitter has a relatively smaller range of transmission. So, while an in-situ sensor is cheap as a single unit, it is expensive to build a large network that can provide connectivity over a large wilderness area. In a *mobile* environment where sensors are carried by moving people, power-efficient routing is difficult to implement and maintain over a large wilderness area. In fact, building an adhoc sensor network using only the sensors worn by hikers is nearly impossible due to a relatively small number of sensors spread over a large wilderness area.

In this paper, we describe the design, implementation and evaluation of a search and rescue system called CenWits (**C**onnection-less S**en**sor-Based Tracking System Using **Wit**nesses). CenWits is comprised of mobile, in-situ sensors that are worn by subjects (people, wild animals, or in-animate objects), access points (AP) that collect information from these sensors, and GPS receivers and location points (LP) that provide location information to the sensors. A subject uses GPS receivers (when it can connect to a satellite) and LPs to determine its current location. The key idea of CenWits is that it uses a concept of *witnesses* to convey a subject's movement and location information to the outside world. This averts a need for maintaining a connected network to transmit location information to the outside world. In particular, there is no need for expensive GSM or satellite transmitters, or maintaining an adhoc network of in-situ sensors in CenWits.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SenSys'05, November 2–4, 2005, San Diego, California, USA.
Copyright 2005 ACM 1-59593-054-X/05/0011 ...$5.00.

CenWits employs several important mechanisms to address the key problem of resource constraints (low signal strength, low power and limited memory) in sensors. In particular, it makes a judicious use of the combined storage capability of sensors to filter, organize and store important information, combined battery power of sensors to ensure that the system remains operational for longer time periods, and intermittent network connectivity to propagate information to a processing center.

The problem of low signal strengths (short range RF communication) is addressed by avoiding a need for maintaining a connected network. Instead, CenWits propagates the location information of sensors using the concept of witnesses through an intermittently connected network. As a result, this system can be deployed in remote wilderness areas, as well as in large urban areas with skyscrapers and other tall structures. Also, this makes CenWits cost-effective. A subject only needs to wear light-weight and low-cost sensors that have GPS receivers but no expensive GSM or satellite transmitters. Furthermore, since there is no need for a connected sensor network, there is no need to deploy sensors in very large numbers.

The problem of limited battery life and limited memory of a sensor is addressed by incorporating the concepts of *groups* and *partitions*. Groups and partitions allow sensors to stay in sleep or receive modes most of the time. Using groups and partitions, the location information collected by a sensor can be distributed among several sensors, thereby reducing the amount of memory needed in one sensor to store that information. In fact, CenWits provides an adaptive tradeoff between memory and power consumption of sensors. Each sensor can dynamically adjust its power and memory consumption based on its remaining power or available memory.

It has amply been noted that the strength of sensor networks comes from the fact that several sensor nodes can be distributed over a relatively large area to construct a multihop network. This paper demonstrates that important large-scale applications can be built using sensors by judiciously integrating the storage, communication and computation capabilities of sensors. The paper describes important techniques to combine memory, transmission and battery power of many sensors to address resource constraints in the context of a search and rescue application. However, these techniques are quite general. We discuss several other sensor-based applications that can employ these techniques.

While CenWits addresses the general location tracking and reporting problem in a wide-area network, there are two important differences from the earlier work done in this area. First, unlike earlier location tracking solutions, CenWits does not require a connected network. Second, unlike earlier location tracking solutions, CenWits does not aim for a very high accuracy of localization. Instead, the main goal is to provide an approximate, small area where search and rescue efforts can be concentrated.

The rest of this paper is organized as follows. In Section 2, we overview some of the recent projects and technologies related to movement and location tracking, and search and rescue systems. In Section 3, we describe the overall architecture of CenWits, and provide a high-level description of its functionality. In the next section, Section 4, we discuss power and memory management in CenWits. To simplify our presentation, we will focus on a specific application of tracking lost/injured hikers in all these sections. In Section 6, we describe a prototype implementation of CenWits and present performance measured from this implementation. We discuss how the ideas of CenWits can be used to build several other applications in Section 7. Finally, in Section 8, we discuss some related issues and conclude the paper.

2. RELATED WORK

A survey of location systems for ubiquitous computing is provided in [11]. A location tracking system for adhoc sensor networks using anchor sensors as reference to gain location information and spread it out to outer node is proposed in [17]. Most location tracking systems in adhoc sensor networks are for benefiting geographic-aware routing. They don't fit well for our purposes. The well-known active badge system [19] lets a user carry a badge around. An infrared sensor in the room can detect the presence of a badge and determine the location and identification of the person. This is a useful system for indoor environment, where GPS doesn't work. Locationing using 802.11 devices is probably the cheapest solution for indoor position tracking [8]. Because of the popularity and low cost of 802.11 devices, several business solutions based on this technology have been developed[1].

A system that combines two mature technologies and is viable in suburban area where a user can see clear sky and has GSM cellular reception at the same time is currently available[5]. This system receives GPS signal from a satellite and locates itself, draws location on a map, and sends location information through GSM network to the others who are interested in the user's location.

A very simple system to monitor children consists an RF transmitter and a receiver. The system alarms the holder of the receiver when the transmitter is about to run out of range [6].

Personal Locater Beacons (PLB) has been used for avalanche rescuing for years. A skier carries an RF transmitter that emits beacons periodically, so that a rescue team can find his/her location based on the strength of the RF signal. Luxury version of PLB combines a GPS receiver and a COSPAS-SARSAT satellite transmitter that can transmit user's location in latitude and longitude to the rescue team whenever an accident happens [4]. However, the device either is turned on all the time resulting in fast battery drain, or must be turned on after the accident to function.

Another related technology in widespread use today is the ONSTAR system [3], typically used in several luxury cars. In this system, a GPS unit provides position information, and a powerful transmitter relays that information via satellite to a customer service center. Designed for emergencies, the system can be triggered either by the user with the push of a button, or by a catastrophic accident. Once the system has been triggered, a human representative attempts to gain communication with the user via a cell phone built as an in-car device. If contact cannot be made, emergency services are dispatched to the location provided by GPS. Like PLBs, this system has several limitations. First, it is heavy and expensive. It requires a satellite transmitter and a connected network. If connectivity with either the GPS network or a communication satellite cannot be maintained, the system fails. Unfortunately, these are common obstacles encountered in deep canyons, narrow streets in large cities, parking garages, and a number of other places.

The Lifetch system uses GPS receiver board combined with a GSM/GPRS transmitter and an RF transmitter in one wireless sensor node called Intelligent Communication Unit (ICU). An ICU first attempts to transmit its location to a control center through GSM/GPRS network. If that fails, it connects with other ICUs (adhoc network) to forward its location information until the information reaches an ICU that has GSM/GPRS reception. This ICU then transmits the location information of the original ICU via the GSM/GPRS network.

ZebraNet is a system designed to study the moving patterns of zebras [13]. It utilizes two protocols: History-based protocol and flooding protocol. History-based protocol is used when the zebras are grazing and not moving around too much. While this might be useful for tracking zebras, it's not suitable for tracking hikers because two hikers are most likely to meet each other only once on a trail. In the flooding protocol, a node dumps its data to a neighbor whenever it finds one and doesn't delete its own copy until it finds a base station. Without considering routing loops, packet filtering and grouping, the size of data on a node will grow exponentially and drain the power and memory of a sensor node with in a short time. Instead, Cenwits uses a four-phase hand-shake protocol to ensure that a node transmits only as much information as the other node is willing to receive. While ZebraNet is designed for a big group of sensors moving together in the same direction with same speed, Cenwits is designed to be used in the scenario where sensors move in different directions at different speeds.

Delay tolerant network architecture addresses some important problems in challenged (resource-constrained) networks [9]. While this work is mainly concerned with interoperability of challenged networks, some problems related to occasionally-connected networks are similar to the ones we have addressed in CenWits.

Among all these systems, luxury PLB and Lifetch are designed for location tracking in wilderness areas. However, both of these systems require a connected network. Luxury PLB requires the user to transmit a signal to a satellite, while Lifetch requires connection to GSM/GPRS network. Luxury PLB transmits location information, only when an accident happens. However, if the user is buried in the snow or falls into a deep canyon, there is almost no chance for the signal to go through and be relayed to the rescue team. This is because satellite transmission needs line of sight. Furthermore, since there is no known history of user's location, it is not possible for the rescue team to infer the current location of the user. Another disadvantage of luxury PLB is that a satellite transmitter is very expensive, costing in the range of $750. Lifetch attempts to transmit the location information by GSM/GPRS and adhoc sensor network that uses AODV as the routing protocol. However, having a cellular reception in remote areas in wilderness areas, e.g. American national parks is unlikely. Furthermore, it is extremely unlikely that ICUs worn by hikers will be able to form an adhoc network in a large wilderness area. This is because the hikers are mobile and it is very unlikely to have several ICUs placed dense enough to forward packets even on a very popular hike route.

CenWits is designed to address the limitations of systems such as luxury PLB and Lifetch. It is designed to provide hikers, skiers, and climbers who have their activities mainly in wilderness areas a much higher chance to convey their location information to a control center. It is not reliant upon constant connectivity with any communication medium. Rather, it communicates information along from user to user, finally arriving at a control center. Unlike several of the systems discussed so far, it does not require that a user's unit is constantly turned on. In fact, it can discover a victim's location, even if the victim's sensor was off at the time of accident and has remained off since then. CenWits solves one of the greatest problems plaguing modern search and rescue systems: it has an inherent on-site storage capability. This means someone within the network will have access to the last-known-location information of a victim, and perhaps his bearing and speed information as well.

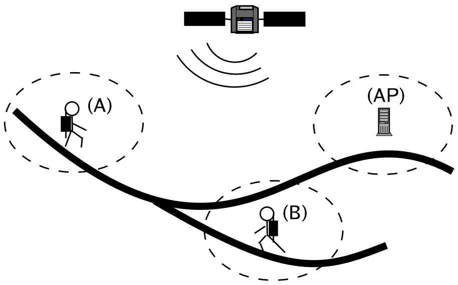

Figure 1: Hiker A and Hiker B are are not in the range of each other

3. CENWITS

We describe CenWits in the context of locating lost/injured hikers in wilderness areas. Each hiker wears a sensor (MICA2 motes in our prototype) equipped with a GPS receiver and an RF transmitter. Each sensor is assigned a unique ID and maintains its current location based on the signal received by its GPS receiver. It also emits beacons periodically. When any two sensors are in the range of one another, they record the presence of each other (*witness information*), and also exchange the witness information they recorded earlier. The key idea here is that if two sensors come with in range of each other at any time, they become each other's witnesses. Later on, if the hiker wearing one of these sensors is lost, the other sensor can convey the last known (witnessed) location of the lost hiker. Furthermore, by exchanging the witness information that each sensor recorded earlier, the witness information is propagated beyond a direct contact between two sensors.

To convey witness information to a processing center or to a rescue team, access points are established at well-known locations that the hikers are expected to pass through, e.g. at the trail heads, trail ends, intersection of different trails, scenic view points, resting areas, and so on. Whenever a sensor node is in the vicinity of an access point, all witness information stored in that sensor is automatically dumped to the access point. Access points are connected to a processing center via satellite or some other network[1]. The witness information is downloaded to the processing center from various access points at regular intervals. In case, connection to an access point is lost, the information from that

[1] A connection is needed only between access points and a processing center. There is no need for any connection between different access points.

access point can be downloaded manually, e.g. by UAVs. To estimate the speed, location and direction of a hiker at any point in time, all witness information of that hiker that has been collected from various access points is processed.

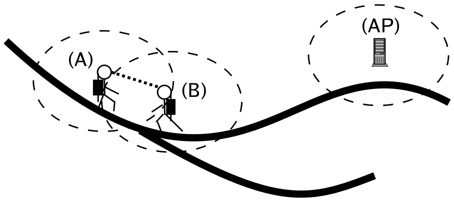

Figure 2: Hiker A and Hiker B are in the range of each other. A records the presence of B and B records the presence of A. A and B become each other's witnesses.

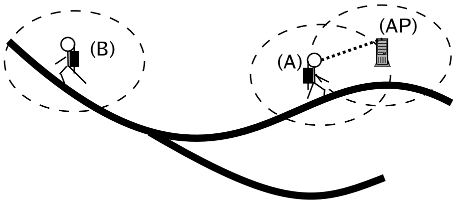

Figure 3: Hiker A is in the range of an access point. It uploads its recorded witness information and clears its memory.

An example of how CenWits operates is illustrated in Figures 1, 2 and 3. First, hikers A and B are on two close trails, but out of range of each other (Figure 1). This is a very common scenario during a hike. For example, on a popular four-hour hike, a hiker might run into as many as 20 other hikers. This accounts for one encounter every 12 minutes on average. A slow hiker can go 1 mile (5,280 feet) per hour. Thus in 12 minutes a slow hiker can go as far as 1056 feet. This implies that if we were to put 20 hikers on a 4-hour, one-way hike evenly, the range of each sensor node should be at least 1056 feet for them to communicate with one another continuously. The signal strength starts dropping rapidly for two Mica2 nodes to communicate with each other when they are 180 feet away, and is completely lost when they are 230 feet away from each other[7]. So, for the sensors to form a sensor network on a 4-hour hiking trail, there should be at least 120 hikers scattered evenly. Clearly, this is extremely unlikely. In fact, in a 4-hour, less-popular hiking trail, one might only run into say five other hikers.

CenWits takes advantage of the fact that sensors can communicate with one another and record their presence. Given a walking speed of one mile per hour (88 feet per minute) and Mica2 range of about 150 feet for non-line-of-sight radio transmission, two hikers have about 150/88 = 1.7 minutes to discover the presence of each other and exchange their witness information. We therefore design our system to have each sensor emit a beacon every one-and-a-half minute. In Figure 2, hiker B's sensor emits a beacon when A is in range, this triggers A to exchange data with B. A communicates the following information to B: "My ID is A; I saw C at 1:23 PM at (39°49.3277655', 105°39.1126776'), I saw E at 3:09 PM at (40°49.2234879', 105°20.3290168')". B then replies with "My ID is B; I saw K at 11:20 AM at (39°51.4531655', 105°41.6776223')". In addition, A records "I saw B at 4:17 PM at (41°29.3177354', 105°04.9106211')" and B records "I saw A at 4:17 PM at (41°29.3177354', 105°04.9106211')".

B goes on his way to overnight camping while A heads back to trail head where there is an AP, which emits beacon every 5 seconds to avoid missing any hiker. A dumps all witness information it has collected to the access point. This is shown in Figure 3.

3.1 Witness Information: Storage

A critical concern is that there is limited amount of memory available on motes (4 KB SDRAM memory, 128 KB flash memory, and 4-512 KB EEPROM). So, it is important to organize witness information efficiently. CenWits stores witness information at each node as a set of *witness records* (Format is shown in Figure 4.

Node ID	Record Time	X, Y	Location Time	Hop Count
1 B	3 B	8 B	3 B	1 B

Figure 4: Format of a witness record.

When two nodes i and j encounter each other, each node generates a new witness record. In the witness record generated by i, Node ID is j, Record Time is the current time in i's clock, (X,Y) are the coordinates of the location of i that i recorded most recently (either from satellite or an LP), Location Time is the time when the this location was recorded, and Hop Count is 0.

Each node is assigned a unique Node Id when it enters a trail. In our current prototype, we have allocated one byte for Node Id, although this can be increased to two or more bytes if a large number of hikers are expected to be present at the same time. We can represent time in 17 bits to a second precision. So, we have allocated 3 bytes each for Record Time and Location Time. The circumference of the Earth is approximately 40,075 KM. If we use a 32-bit number to represent both longitude and latitude, the precision we get is $40{,}075{,}000/2^{32} = 0.0093$ meter $= 0.37$ inches, which is quite precise for our needs. So, we have allocated 4 bytes each for X and Y coordinates of the location of a node. In fact, a foot precision can be achieved by using only 27 bits.

3.2 Location Point and Location Inference

Although a GPS receiver provides an accurate location information, it has it's limitation. In canyons and rainy forests, a GPS receiver does not work. When there is a heavy cloud cover, GPS users have experienced inaccuracy in the reported location as well. Unfortunately, a lot of hiking trails are in dense forests and canyons, and it's not that uncommon to rain after hikers start hiking. To address this, CenWits incorporates the idea of location points (LP). A location point can update a sensor node with its current location whenever the node is near that LP. LPs are placed at different locations in a wilderness area where GPS receivers don't work. An LP is a very simple device that emits prerecorded location information at some regular time interval. It can be placed in difficult-to-reach places such as deep canyons and dense rain forests by simply dropping them from an airplane. LPs allow a sensor node to determine its current location more accurately. However, they are not an essential requirement of CenWits. If an LP runs out of power, the CenWits will continue to work correctly.

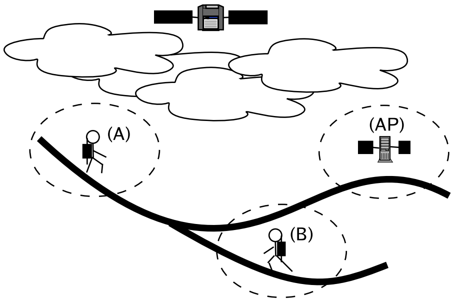

Figure 5: GPS receiver not working correctly. Sensors then have to rely on LP to provide coordination

In Figure 5, B cannot get GPS reception due to bad weather. It then runs into A on the trail who doesn't have GPS reception either. Their sensors record the presence of each other. After 10 minutes, A is in range of an LP that provides an accurate location information to A. When A returns to trail head and uploads its data (Figure 6), the system can draw a circle centered at the LP from which A fetched location information for the range of encounter location of A and B. By Overlapping this circle with the trail map, two or three possible location of encounter can be inferred. Thus when a rescue is required, the possible location of B can be better inferred (See Figures 7 and 8).

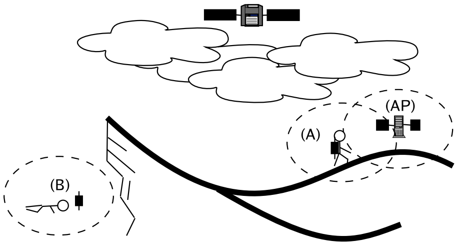

Figure 6: A is back to trail head, It reports the time of encounter with B to AP, but no location information to AP

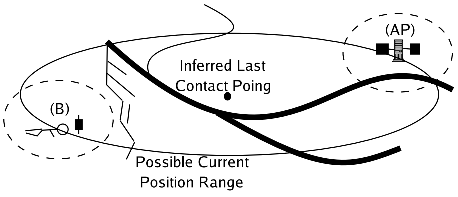

Figure 7: B is still missing after sunset. CenWits infers the last contact point and draws the circle of possible current locations based on average hiking speed

CenWits requires that the clocks of different sensor nodes be loosely synchronized with one another. Such a synchronization is trivial when GPS coverage is available. In addition, sensor nodes in CenWits synchronize their clocks whenever they are in the range of an AP or an LP. The synchronization accuracy Cenwits needs is of the order of a second or so. As long as the clocks are synchronized with in one second range, whether A met B at 12:37'45" or 12:37'46" doesn't matter in the ordering of witness events and inferring the path.

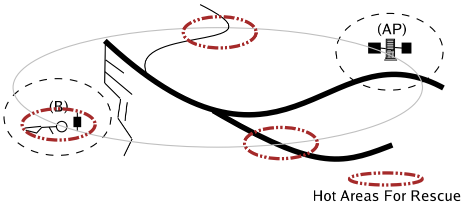

Figure 8: Based on overlapping landscape, B might have hiked to wrong branch and fallen off a cliff. Hot rescue areas can thus be determined

4. MEMORY AND POWER MANAGEMENT

CenWits employs several important mechanisms to conserve power and memory. It is important to note while current sensor nodes have limited amount of memory, future sensor nodes are expected to have much more memory. With this in mind, the main focus in our design is to provide a tradeoff between the amount of memory available and amount of power consumption.

4.1 Memory Management

The size of witness information stored at a node can get very large. This is because the node may come across several other nodes during a hike, and may end up accumulating a large amount of witness information over time. To address this problem, CenWits allows a node to pro-actively free up some parts of its memory periodically. This raises an interesting question of when and which witness record should be deleted from the memory of a node? CenWits uses three criteria to determine this: *record count*, *hop count*, and *record gap*.

Record count refers to the number of witness records with same node id that a node has stored in its memory. A node maintains an integer parameter MAX_RECORD_COUNT. It stores at most MAX_RECORD_COUNT witness records of any node.

Every witness record has a hop count field that stores the number times (hops) this record has been transferred since being created. Initially this field is set to 0. Whenever a node receives a witness record from another node, it increments the hop count of that record by 1. A node maintains an integer parameter called MAX_HOP_COUNT. It keeps only those witness records in its memory, whose hop count is less than MAX_HOP_COUNT. The MAX_HOP_COUNT parameter provides a balance between two conflicting goals: (1) To ensure that a witness record has been propagated to and thus stored at as many nodes as possible, so that it has a high probability of being dumped at some AP as quickly as possible; and (2) To ensure that a witness record is stored only at a few nodes, so that it does not clog up too much of the combined memory of all sensor nodes. We chose to use hop count instead of time-to-live to decide when to drop a packet. The main reason for this is that the probability of a packet reaching an AP goes up as the hop count adds up. For example, when the hop count if 5 for a specific record, the record is in at least 5 sensor nodes. On the other hand, if we discard old records, without considering hop count, there is no guarantee that the record is present in any other sensor node.

Record gap refers to the time difference between the record times of two witness records with the same node id. To save memory, a node n ensures the the record gap between any two witness records with the same node id is at least MIN_RECORD_GAP. For each node id i, n stores the witness record with the most recent record time rt_i, the witness with most recent record time that is at least MIN_RECORD_GAP time units before rt_i, and so on until the record count limit (MAX_RECORD_COUNT) is reached.

When a node is tight in memory, it adjusts the three parameters, MAX_RECORD_COUNT, MAX_HOP_COUNT and MIN_RECORD_GAP to free up some memory. It decrements MAX_RECORD_COUNT and MAX_HOP_COUNT, and increments MIN_RECORD_GAP. It then first erases all witness records whose hop count exceeds the reduced MAX_HOP_COUNT value, and then erases witness records to satisfy the record gap criteria. Also, when a node has extra memory space available, e.g. after dumping its witness information at an access point, it resets MAX_RECORD_COUNT, MAX_HOP_COUNT and MIN_RECORD_GAP to some predefined values.

4.2 Power Management

An important advantage of using sensors for tracking purposes is that we can regulate the behavior of a sensor node based on current conditions. For example, we mentioned earlier that a sensor should emit a beacon every 1.7 minute, given a hiking speed of 1 mile/hour. However, if a user is moving 10 feet/sec, a beacon should be emitted every 10 seconds. If a user is not moving at all, a beacon can be emitted every 10 minutes. In the night, a sensor can be put into sleep mode to save energy, when a user is not likely to move at all for a relatively longer period of time. If a user is active for only eight hours in a day, we can put the sensor into sleep mode for the other 16 hours and thus save $2/3^{rd}$ of the energy.

In addition, a sensor node can choose to not send any beacons during some time intervals. For example, suppose hiker A has communicated its witness information to three other hikers in the last five minutes. If it is running low on power, it can go to receive mode or sleep mode for the next ten minutes. It goes to receive mode if it is still willing to receive additional witness information from hikers that it encounters in the next ten minutes. It goes to sleep mode if it is extremely low on power.

The bandwidth and energy limitations of sensor nodes require that the amount of data transferred among the nodes be reduced to minimum. It has been observed that in some scenarios 3000 instructions could be executed for the same energy cost of sending a bit 100m by radio [15]. To reduce the amount of data transfer, CenWits employs a *handshake protocol* that two nodes execute when they encounter one another. The goal of this protocol is to ensure that a node transmits only as much witness information as the other node is willing to receive. This protocol is initiated when a node i receives a beacon containing the node ID of the sender node j and i has not exchanged witness information with j in the last δ time units. Assume that $i < j$. The protocol consists of four phases (See Figure 9):

1. **Phase I:** Node i sends its *receive constraints* and the number of witness records it has in its memory.

2. **Phase II:** On receiving this message from i, j sends its receive constraints and the number of witness records it has in its memory.

3. **Phase III:** On receiving the above message from j, i sends its witness information (filtered based on receive constraints received in phase II).

4. **Phase IV:** After receiving the witness records from i, j sends its witness information (filtered based on receive constraints received in phase I).

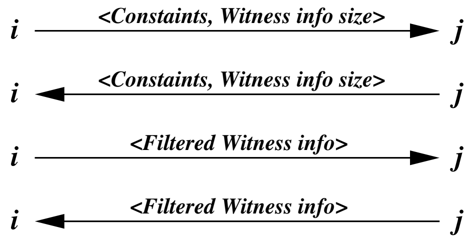

Figure 9: Four-Phase Hand Shake Protocol ($i < j$)

Receive constraints are a function of memory and power. In the most general case, they are comprised of the three parameters (record count, hop count and record gap) used for memory management. If i is low on memory, it specifies the maximum number of records it is willing to accept from j. Similarly, i can ask j to send only those records that have hop count value less than MAX_HOP_COUNT − 1. Finally, i can include its MIN_RECORD_GAP value in its receive constraints. Note that the handshake protocol is beneficial to both i and j. They save memory by receiving only as much information as they are willing to accept and conserve energy by sending only as many witness records as needed.

It turns out that filtering witness records based on MIN_RECORD_GAP is complex. It requires that the witness records of any given node be arranged in an order sorted by their record time values. Maintaining this sorted order is complex in memory, because new witness records with the same node id can arrive later that may have to be inserted in between to preserve the sorted order. For this reason, the receive constraints in the current CenWits prototype do not include record gap.

Suppose i specifies a hop count value of 3. In this case, j checks the hop count field of every witness record before sending them. If the hop count value is greater than 3, the record is not transmitted.

4.3 Groups and Partitions

To further reduce communication and increase the lifetime of our system, we introduce the notion of *groups*. The idea is based on the concept of abstract regions presented in [20]. A group is a set of n nodes that can be defined in terms of radio connectivity, geographic location, or other properties of nodes. All nodes within a group can communicate directly with one another and they share information to maintain their view of the external world. At any point in time, a group has exactly one leader that communicates with external nodes on behalf of the entire group. A group can be static, meaning that the group membership does not change over the period of time, or it could be dynamic in which case nodes can leave or join the group. To make our analysis simple and to explain the advantages of group, we first discuss static groups.

A static group is formed at the start of a hiking trail or ski slope. Suppose there are five family members who want to go for a hike in the Rocky Mountain National Park. Before these members start their hike, each one of them is given a sensor node and the information is entered in the system that the five nodes form a group. Each group member is given a unique id and every group member knows about other members of the group. The group, as a whole, is also assigned an id to distinguish it from other groups in the system.

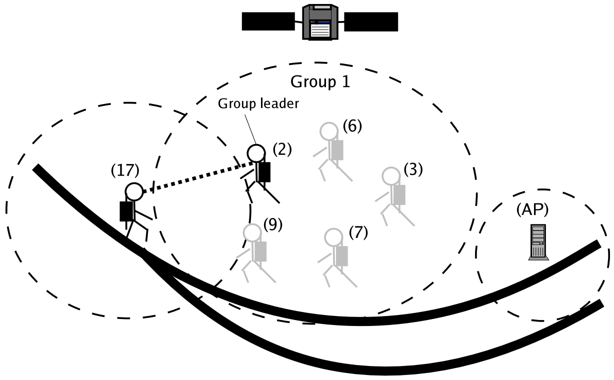

Figure 10: A group of five people. Node 2 is the group leader and it is communicating on behalf of the group with an external node 17. All other (shown in a lighter shade) are in sleep mode.

As the group moves through the trail, it exchanges information with other nodes or groups that it comes across. At any point in time, only one group member, called the leader, sends and receives information on behalf of the group and all other $n-1$ group members are put in the sleep mode (See Figure 10). It is this property of groups that saves us energy. The group leadership is time-multiplexed among the group members. This is done to make sure that a single node does not run out of battery due to continuous exchange of information. Thus after every t seconds, the leadership is passed on to another node, called the *successor*, and the leader (now an ordinary member) is put to sleep. Since energy is dear, we do not implement an extensive election algorithm for choosing the successor. Instead, we choose the successor on the basis of node id. The node with the next highest id in the group is chosen as the successor. The last node, of course, chooses the node with the lowest id as its successor.

We now discuss the data storage schemes for groups. Memory is a scarce resource in sensor nodes and it is therefore important that witness information be stored efficiently among group members. Efficient data storage is not a trivial task when it comes to groups. The tradeoff is between simplicity of the scheme and memory savings. A simpler scheme incurs lesser energy cost as compared to a more sophisticated scheme, but offers lesser memory savings as well. This is because in a more complicated scheme, the group members have to coordinate to update and store information. After considering a number of different schemes, we have come to a conclusion that there is no optimal storage scheme for groups. The system should be able to adapt according to its requirements. If group members are low on battery, then the group can adapt a scheme that is more energy efficient. Similarly, if the group members are running out of memory, they can adapt a scheme that is more memory efficient. We first present a simple scheme that is very energy efficient but does not offer significant memory savings. We then present an alternate scheme that is much more memory efficient.

As already mentioned a group can receive information only through the group leader. Whenever the leader comes across an external node e, it receives information from that node and saves it. In our first scheme, when the timeslot for the leader expires, the leader passes this new information it received from e to its successor. This is important because during the next time slot, if the new leader comes across another external node, it should be able to pass information about all the external nodes this group has witnessed so far. Thus the information is fully replicated on all nodes to maintain the correct view of the world.

Our first scheme does not offer any memory savings but is highly energy efficient and may be a good choice when the group members are running low on battery. Except for the time when the leadership is switched, all $n-1$ members are asleep at any given time. This means that a single member is up for t seconds once every $n*t$ seconds and therefore has to spend approximately only $1/n^{th}$ of its energy. Thus, if there are 5 members in a group, we save 80% energy, which is huge. More energy can be saved by increasing the group size.

We now present an alternate data storage scheme that aims at saving memory at the cost of energy. In this scheme we divide the group into what we call *partitions*. Partitions can be thought of as subgroups within a group. Each partition must have at least two nodes in it. The nodes within a partition are called *peers*. Each partition has one peer designated as *partition leader*. The partition leader stays in *receive* mode at all times, while all others peers a partition stay in the sleep mode. Partition leadership is time-multiplexed among the peers to make sure that a single node does not run out of battery. Like before, a group has exactly one leader and the leadership is time-multiplexed among partitions. The group leader also serves as the partition leader for the partition it belongs to (See Figure 11).

In this scheme, all partition leaders participate in information exchange. Whenever a group comes across an external node e, every partition leader receives all witness information, but it only stores a subset of that information after filtering. Information is filtered in such a way that each partition leader has to store only B/K bytes of data, where K is the number of partitions and B is the total number of bytes received from e. Similarly when a group wants to send witness information to e, each partition leader sends only B/K bytes that are stored in the partition it belongs to. However, before a partition leader can send information, it must switch from receive mode to send mode. Also, partition leaders must coordinate with one another to ensure that they do not send their witness information at the same time, i.e. their message do not collide. All this is achieved by having the group leader send a signal to every partition leader in turn.

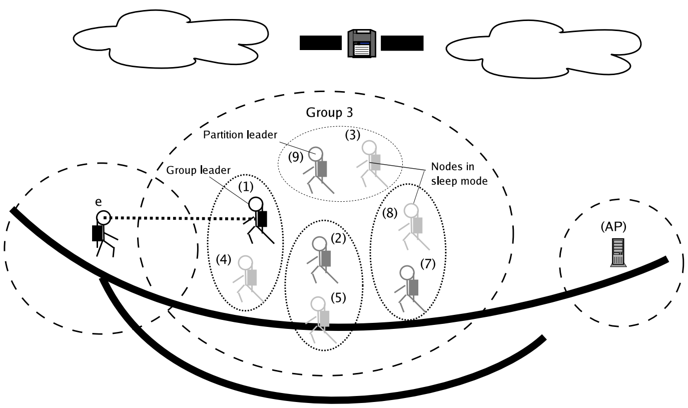

Figure 11: The figure shows a group of eight nodes divided into four partitions of 2 nodes each. Node 1 is the group leader whereas nodes 2, 9, and 7 are partition leaders. All other nodes are in the sleep mode.

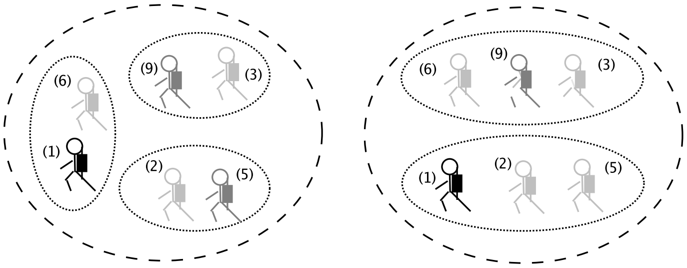

Figure 12: The figure shows two different ways of partitioning a group of six nodes. In (a), a group is divided into three partitions of two nodes. Node 1 is the group leader, nodes 9 and 5 are partition leaders, and nodes 2, 3, and 6 are in sleep mode. In (b) the group is divided into two partitions of three nodes. Node 1 is the group leader, node 9 is the partition leader and nodes 2, 3, 5, and 6 are in sleep mode.

Since the partition leadership is time-multiplexed, it is important that any information received by the partition leader, $p1$, be passed on to the next leader, $p2$. This has to be done to make sure that $p2$ has all the information that it might need to send when it comes across another external node during its timeslot. One way of achieving this is to wake $p2$ up just before $p1$'s timeslot expires and then have $p1$ transfer information only to $p2$. An alternate is to wake all the peers up at the time of leadership change, and then have $p1$ broadcast the information to all peers. Each peer saves the information sent by $p1$ and then goes back to sleep. In both cases, the peers send acknowledgement to the partition leader after receiving the information. In the former method, only one node needs to wake up at the time of leadership change, but the amount of information that has to be transmitted between the nodes increases as time passes. In the latter case, all nodes have to be woken up at the time of leadership change, but small piece of information has to be transmitted each time among the peers. Since communication is much more expensive than bringing the nodes up, we prefer the second method over the first one.

A group can be divided into partitions in more than one way. For example, suppose we have a group of six members. We can divide this group into three partitions of two peers each, or two partitions with three peers each. The choice once again depends on the requirements of the system. A few big partitions will make the system more energy efficient. This is because in this configuration, a greater number of nodes will stay in sleep mode at any given point in time. On the other hand, several small partitions will make the system memory efficient, since each node will have to store lesser information (See Figure 12).

A group that is divided into partitions must be able to readjust itself when a node leaves or runs out of battery. This is crucial because a partition must have at least two nodes at any point in time to tolerate failure of one node. For example, in figure 3 (a), if node 2 or node 5 dies, the partition is left with only one node. Later on, if that single node in the partition dies, all witness information stored in that partition will be lost. We have devised a very simple protocol to solve this problem. We first explain how partitions are adjusted when a peer dies, and then explain what happens if a partition leader dies.

Suppose node 2 in figure 3 (a) dies. When node 5, the partition leader, sends information to node 2, it does not receive an acknowledgement from it and concludes that node 2 has died[2]. At this point, node 5 contacts other partition leaders (nodes 1 ... 9) using a broadcast message and informs them that one of its peers has died. Upon hearing this, each partition leader informs node 5 (i) the number of nodes in its partition, (ii) a candidate node that node 5 can take if the number of nodes in its partition is greater than 2, and (iii) the amount of witness information stored in its partition. Upon hearing from every leader, node 5 chooses the candidate node from the partition with maximum number (must be greater than 2) of peers, and sends a message back to all leaders. Node 5 then sends data to its new peer to make sure that the information is replicated within the partition.

However, if all partitions have exactly two nodes, then node 5 must join another partition. It chooses the partition that has the least amount of witness information to join. It sends its witness information to the new partition leader. Witness information and membership update is propagated to all peers during the next partition leadership change.

We now consider the case where the partition leader dies. If this happens, then we wait for the partition leadership to change and for the new partition leader to eventually find out that a peer has died. Once the new partition leader finds out that it needs more peers, it proceeds with the protocol explained above. However, in this case, we do lose information that the previous partition leader might have received just before it died. This problem can be solved by implementing a more rigorous protocol, but we have decided to give up on accuracy to save energy.

Our current design uses time-division multiplexing to schedule wakeup and sleep modes in the sensor nodes. However, recent work on radio wakeup sensors [10] can be used to do this scheduling more efficiently. we plan to incorporate radio wakeup sensors in CenWits when the hardware is mature.

[2] The algorithm to conclude that a node has died can be made more rigorous by having the partition leader query the suspected node a few times.

5. SYSTEM EVALUATION

A sensor is constrained in the amount of memory and power. In general, the amount of memory needed and power consumption depends on a variety of factors such as node density, number of hiker encounters, and the number of access points. In this Section, we provide an estimate of how long the power of a MICA2 mote will last under certain assumtions.

First, we assume that each sensor node carries about 100 witness records. On encountering another hiker, a sensor node transmits 50 witness records and receives 50 new witness records. Since, each record is 16 bytes long, it will take 0.34 seconds to transmit 50 records and another 0.34 seconds to receive 50 records over a 19200 bps link. The power consumption of MICA2 due to CPU processing, transmission and reception are approximately 8.0 mA, 7.0 mA and 8.5 mA per hour respectively [18], and the capacity of an alkaline battery is 2500mAh.

Since the radio module of Mica2 is half-duplex and assuming that the CPU is always active when a node is awake, power consumption due to transmission is $8 + 8.5 = 16.5$ mA per hour and due to reception is $8 + 7 = 15$mA per hour. So, average power consumtion due to transmission and reception is $(16.5 + 15)/2 = 15.75$ mA per hour.

Given that the capacity of an alkaline battery is 2500 mAh, a battery should last for $2500/15.75 = 159$ hours of transmission and reception. An encounter between two hikers results in exchange of about 50 witness records that takes about 0.68 seconds as calculated above. Thus, a single alkaline battery can last for $(159 * 60 * 60)/0.68 = 841764$ hiker encounters.

Assuming that a node emits a beacon every 90 seconds and a hiker encounter occurs everytime a beacon is emitted (worst-case scenario), a single alkaline battery will last for $(841764 * 90)/(30 * 24 * 60 * 60) = 29$ days. Since, a Mica2 is equipped with two batteries, a Mica2 sensor can remain operation for about two months. Notice that this calculation is preliminary, because it assumes that hikers are active 24 hours of the day and a hiker encounter occurs every 90 seconds. In a more realistic scenario, power is expected to last for a much longer time period. Also, this time period will significantly increase when groups of hikers are moving together.

Finally, the lifetime of a sensor running on two batteries can definitely be increased significantly by using energy scavenging techniques and energy harvesting techniques [16, 14].

6. PROTOTYPE IMPLEMENTATION

We have implemented a prototype of CenWits on MICA2 sensor 900MHz running Mantis OS 0.9.1b. One of the sensor is equipped with MTS420CA GPS module, which is capable of barometric pressure and two-axis acceleration sensing in addition to GPS location tracking. We use SiRF, the serial communication protocol, to control GPS module. SiRF has a rich command set, but we record only X and Y coordinates. A witness record is 16 bytes long. When a node starts up, it stores its current location and emits a beacon periodically—in the prototype, a node emits a beacon every minute.

We have conducted a number of experiments with this prototype. A detailed report on these experiments with the raw data collected and photographs of hikers, access points etc. is available at *http://csel.cs.colorado.edu/∼huangjh/Cenwits.index.htm*. Here we report results from three of them. In all these experiments, there are three access points (A, B and C) where nodes dump their witness information. These access points also provide location information to the nodes that come with in their range. We first show how CenWits can be used to determine the hiking trail a hiker is most likely on and the speed at which he is hiking, and identify hot search areas in case he is reported missing. Next, we show the results of power and memory management techniques of CenWits in conserving power and memory of a sensor node in one of our experiments.

6.1 Locating Lost Hikers

The first experiment is called *Direct Contact*. It is a very simple experiment in which a single hiker starts from A, goes to B and then C, and finally returns to A (See Figure 13). The goal of this experiment is to illustrate that CenWits can deduce the trail a hiker takes by processing witness information.

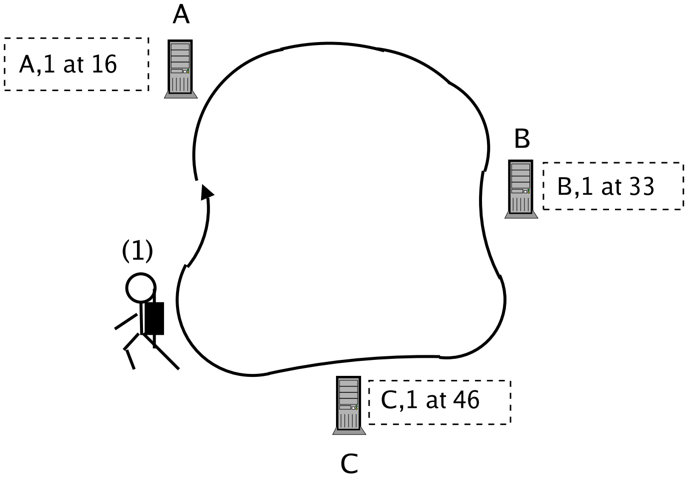

Figure 13: Direct Contact Experiment

Node Id	Record Time	(X,Y)	Location Time	Hop Count
1	15	(12,7)	15	0
1	33	(31,17)	33	0
1	46	(12,23)	46	0
1	10	(12,7)	10	0
1	48	(12,23)	48	0
1	16	(12,7)	16	0
1	34	(31,17)	34	0

Table 1: Witness information collected in the direct contact experiment.

The witness information dumped at the three access points was then collected and processed at a control center. Part of the witness information collected at the control center is shown in Table 1. The X,Y locations in this table correspond to the location information provided by access points A, B, and C. A is located at (12,7), B is located at (31,17) and C is located at (12,23). Three encounter points (between hiker 1 and the three access points) extracted from this witness information are shown in Figure 13 (shown in rectangular boxes). For example, A,1 at 16 means 1 came in contact with A at time 16. Using this information, we can infer the direction in which hiker 1 was moving and speed at which he was moving. Furthermore, given a map of hiking trails in this area, it is clearly possible to identify the hiking trail that hiker 1 took.

The second experiment is called *Indirect Inference.* This experiment is designed to illustrate that the location, direction and speed of a hiker can be inferred by CenWits, even if the hiker never comes in the range of any access point. It illustrates the importance of witness information in search and rescue applications. In this experiment, there are three hikers, 1, 2 and 3. Hiker 1 takes a trail that goes along access points A and B, while hiker 3 takes trail that goes along access points C and B. Hiker 2 takes a trail that does not come in the range of any access points. However, this hiker meets hiker 1 and 3 during his hike. This is illustrated in Figure 14.

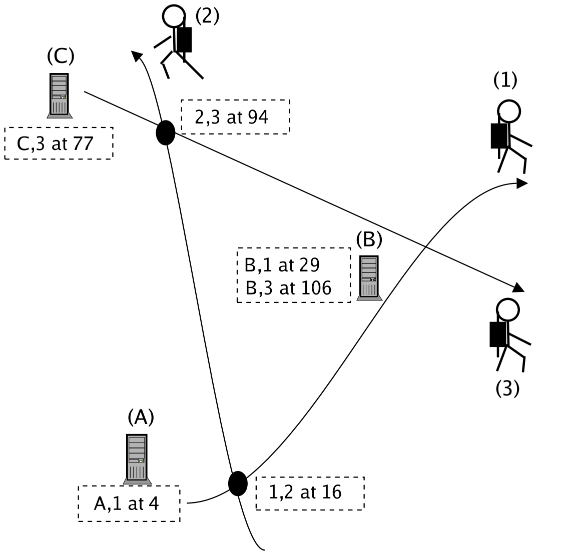

Figure 14: Indirect Inference Experiment

Node Id	Record Time	(X,Y)	Location Time	Hop Count
2	16	(12,7)	6	0
2	15	(12,7)	6	0
1	4	(12,7)	4	0
1	6	(12,7)	6	0
1	29	(31,17)	29	0
1	31	(31,17)	31	0

Table 2: Witness information collected from hiker 1 in indirect inference experiment.

Node Id	Record Time	(X,Y)	Location Time	Hop Count
3	78	(12,23)	78	0
3	107	(31,17)	107	0
3	106	(31,17)	106	0
3	76	(12,23)	76	0
3	79	(12,23)	79	0
2	94	(12,23)	79	0
1	16	(?,?)	?	1
1	15	(?,?)	?	1

Table 3: Witness information collected from hiker 3 in indirect inference experiment.

Part of the witness information collected at the control center from access points A, B and C is shown in Tables 2 and 3. There are some interesting data in these tables. For example, the location time in some witness records is not the same as the record time. This means that the node that generated that record did not have its most up-to-date location at the encounter time. For example, when hikers 1 and 2 meet at time 16, the last recorded location time of hiker 1 is (12,7) recorded at time 6. So, node 1 generates a witness record with record time 16, location (12,7) and location time 6. In fact, the last two records in Table 3 have (?,?) as their location. This has happened because these witness records were generate by hiker 2 during his encounter with 1 at time 15 and 16. Until this time, hiker 2 hadn't come in contact with any location points.

Interestingly, a more accurate location information of 1 and 2 encounter or 2 and 3 encounter can be computed by process the witness information at the control center. It took 25 units of time for hiker 1 to go from A (12,7) to B (31,17). Assuming a constant hiking speed and a relatively straight-line hike, it can be computed that at time 16, hiker 1 must have been at location (18,10). Thus (18,10) is a more accurate location of encounter between 1 and 2.

Finally, our third experiment called *Identifying Hot Search Areas* is designed to determine the trail a hiker has taken and identify hot search areas for rescue after he is reported missing. There are six hikers (1, 2, 3, 4, 5 and 6) in this experiment. Figure 15 shows the trails that hikers 1, 2, 3, 4 and 5 took, along with the encounter points obtained from witness records collected at the control center. For brevity, we have not shown the entire witness information collected at the control center. This information is available at *http://csel.cs.colorado.edu/∼huangjh/Cenwits/index.htm.*

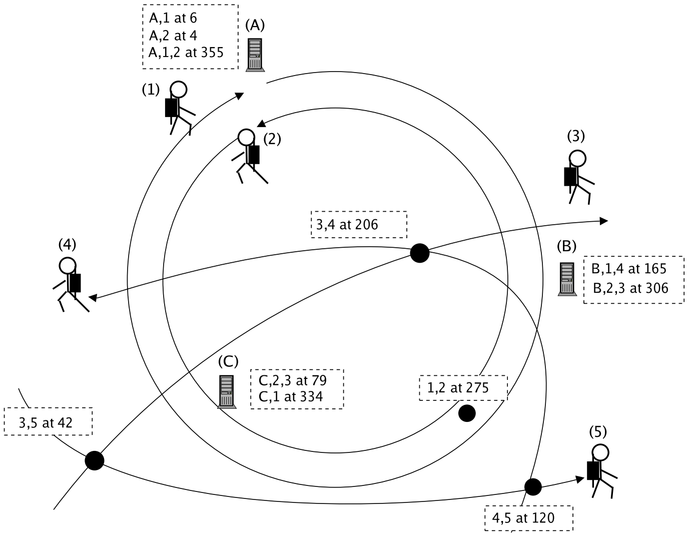

Figure 15: Identifying Hot Search Area Experiment (without hiker 6)

Now suppose hiker 6 is reported missing at time 260. To determine the hot search areas, the witness records of hiker 6 are processed to determine the trail he is most likely on, the speed at which he had been moving, direction in which he had been moving, and his last known location. Based on this information and the hiking trail map, hot search areas are identified. The hiking trail taken by hiker 6 as inferred by CenWits is shown by a dotted line and the hot search areas identified by CenWits are shown by dark lines inside the dotted circle in Figure 16.

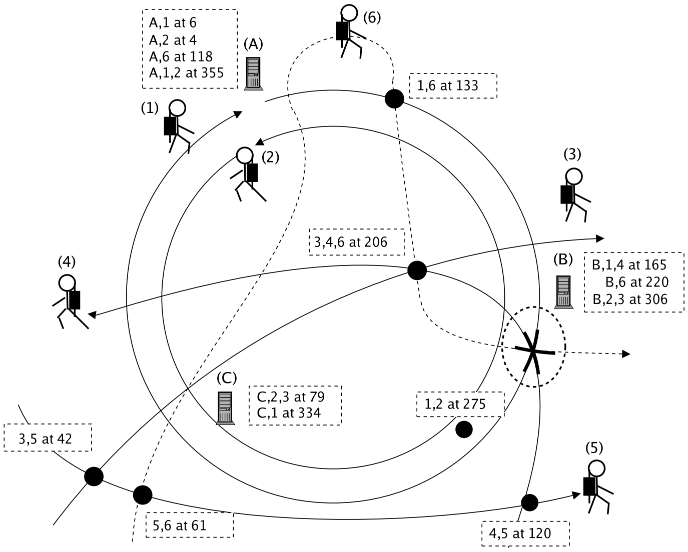

Figure 16: Identifying Hot Search Area Experiment (with hiker 6)

6.2 Results of Power and Memory Management

The witness information shown in Tables 1, 2 and 3 has not been filtered using the three criteria described in Section 4.1. For example, the witness records generated by 3 at record times 76, 78 and 79 (see Table 3) have all been generated due a single contact between access point C and node 3. By applying the record gap criteria, two of these three records will be erased. Similarly, the witness records generated by 1 at record times 10, 15 and 16 (see Table 1) have all been generated due a single contact between access point A and node 1. Again, by applying the record gap criteria, two of these three records will be erased. Our experiments did not generate enough data to test the impact of record count or hop count criteria.

To evaluate the impact of these criteria, we simulated CenWits to generate a significantly large number of records for a given number of hikers and access points. We generated witness records by having the hikers walk randomly. We applied the three criteria to measure the amount of memory savings in a sensor node. The results are shown in Table 4. The number of hikers in this simulation was 10 and the number of access points was 5. The number of witness records reported in this table is an average number of witness records a sensor node stored at the time of dump to an access point.

These results show that the three memory management criteria significantly reduces the memory consumption of sensor nodes in CenWits. For example, they can reduce the memory consumption by up to 75%. However, these results are premature at present for two reasons: (1) They are generated via simulation of hikers walking at random; and (2) It is not clear what impact the erasing of witness records has on the accuracy of inferred location/hot search areas of lost hikers. In our future work, we plan to undertake a major study to address these two concerns.

MAX_ RECORD _COUNT	MIN_ RECORD _GAP	MAX_ HOP_ COUNT	# of Witness Records
5	5	5	628
4	5	5	421
3	5	5	316
5	10	5	311
5	20	5	207
5	5	4	462
5	5	3	341
3	20	3	161

Table 4: Impact of memory management techniques.

7. OTHER APPLICATIONS

In addition to the hiking in wilderness areas, CenWits can be used in several other applications, e.g. skiing, climbing, wild life monitoring, and person tracking. Since CenWits relies only on intermittent connectivity, it can take advantage of the existing cheap and mature technologies, and thereby make tracking cheaper and fairly accurate. Since CenWits doesn't rely on keeping track of a sensor holder all time, but relies on maintaining witnesses, the system is relatively cheaper and widely applicable. For example, there are some dangerous cliffs in most ski resorts. But it is too expensive for a ski resort to deploy a connected wireless sensor network through out the mountain. Using CenWits, we can deploy some sensors at the cliff boundaries. These boundary sensors emit beacons quite frequently, e.g. every second, and so can record presence of skiers who cross the boundary and fall off the cliff. Ski patrols can cruise the mountains every hour, and automatically query the boundary sensor when in range using PDAs. If a PDA shows that a skier has been close to the boundary sensor, the ski patrol can use a long range walkie-talkie to query control center at the resort base to check the witness record of the skier. If there is no witness record after the recorded time in the boundary sensor, there is a high chance that a rescue is needed.

In wildlife monitoring, a very popular method is to attach a GPS receiver on the animals. To collect data, either a satellite transmitter is used, or the data collector has to wait until the GPS receiver brace falls off (after a year or so) and then search for the GPS receiver. GPS transmitters are very expensive, e.g. the one used in geese tracking is $3,000 each [2]. Also, it is not yet known if continuous radio signal is harmful to the birds. Furthermore, a GPS transmitter is quite bulky and uncomfortable, and as a result, birds always try to get rid of it. Using CenWits, not only can we record the presence of wildlife, we can also record the behavior of wild animals, e.g. lions might follow the migration of deers. CenWits does nor require any bulky and expensive satellite transmitters, nor is there a need to wait for a year and search for the braces. CenWits provides a very simple and cost-effective solution in this case. Also, access points can be strategically located, e.g. near a water source, to increase chances of collecting up-to-date data. In fact, the access points need not be statically located. They can be placed in a low-altitude plane (e.g a UAV) and be flown over a wilderness area to collect data from wildlife.

In large cities, CenWits can be used to complement GPS, since GPS doesn't work indoor and near skyscrapers. If a person A is reported missing, and from the witness records we find that his last contacts were C and D, we can trace an approximate location quickly and quite efficiently.

8. DISCUSSION AND FUTURE WORK

This paper presents a new search and rescue system called CenWits that has several advantages over the current search and rescue systems. These advantages include a loosely-coupled system that relies only on intermittent network connectivity, power and storage efficiency, and low cost. It solves one of the greatest problems plaguing modern search and rescue systems: it has an inherent on-site storage capability. This means someone within the network will have access to the last-known-location information of a victim, and perhaps his bearing and speed information as well. It utilizes the concept of witnesses to propagate information, infer current possible location and speed of a subject, and identify hot search and rescue areas in case of emergencies.

A large part of CenWits design focuses on addressing the power and memory limitations of current sensor nodes. In fact, power and memory constraints depend on how much weight (of sensor node) a hiker is willing to carry and the cost of these sensors. An important goal of CenWits is build small chips that can be implanted in hiking boots or ski jackets. This goal is similar to the avalanche beacons that are currently implanted in ski jackets. We anticipate that power and memory will continue to be constrained in such an environment.

While the paper focuses on the development of a search and rescue system, it also provides some innovative, system-level ideas for information processing in a sensor network system.

We have developed and experimented with a basic prototype of CenWits at present. Future work includes developing a more mature prototype addressing important issues such as security, privacy, and high availability. There are several pressing concerns regarding security, privacy, and high availability in CenWits. For example, an adversary can sniff the witness information to locate endangered animals, females, children, etc. He may inject false information in the system. An individual may not be comfortable with providing his/her location and movement information, even though he/she is definitely interested in being located in a timely manner at the time of emergency. In general, people in hiking community are friendly and usually trustworthy. So, a bullet-proof security is not really required. However, when CenWits is used in the context of other applications, security requirements may change. Since the sensor nodes used in CenWits are fragile, they can fail. In fact, the nature and level of security, privacy and high availability support needed in CenWits strongly depends on the application for which it is being used and the individual subjects involved. Accordingly, we plan to design a multi-level support for security, privacy and high availability in CenWits.

So far, we have experimented with CenWits in a very restricted environment with a small number of sensors. Our next goal is to deploy this system in a much larger and more realistic environment. In particular, discussions are currenly underway to deploy CenWits in the Rocky Mountain and Yosemite National Parks.

9. REFERENCES

[1] 802.11-based tracking system. *http://www.pangonetworks.com/locator.htm.*

[2] Brent geese 2002. *http://www.wwt.org.uk/brent/.*

[3] The onstar system. *http://www.onstar.com.*

[4] Personal locator beacons with GPS receiver and satellite transmitter. *http://www.aeromedix.com/.*

[5] Personal tracking using GPS and GSM system. *http://www.ulocate.com/trimtrac.html.*

[6] Rf based kid tracking system. *http://www.ion-kids.com/.*

[7] F. Alessio. Performance measurements with motes technology. *MSWiM'04*, 2004.

[8] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. *IEEE Infocom*, 2000.

[9] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM*, 2003.

[10] L. Gu and J. Stankovic. Radio triggered wake-up capability for sensor networks. In *Real-Time Applications Symposium*, 2004.

[11] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 2001.

[12] W. Jaskowski, K. Jedrzejek, B. Nyczkowski, and S. Skowronek. Lifetch life saving system. *CSIDC*, 2004.

[13] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. In *ASPLOS*, 2002.

[14] K. Kansal and M. Srivastava. Energy harvesting aware power management. In *Wireless Sensor Networks: A Systems Perspective*, 2005.

[15] G. J. Pottie and W. J. Kaiser. Embedding the internet: wireless integrated network sensors. *Communications of the ACM*, 43(5), May 2000.

[16] S. Roundy, P. K. Wright, and J. Rabaey. A study of low-level vibrations as a power source for wireless sensor networks. *Computer Communications*, 26(11), 2003.

[17] C. Savarese, J. M. Rabaey, and J. Beutel. Locationing in distributed ad-hoc wireless sensor networks. *ICASSP*, 2001.

[18] V. Shnayder, M. Hempstead, B. Chen, G. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Sensys*, 2004.

[19] R. Want and A. Hopper. Active badges and personal interactive computing objects. *IEEE Transactions of Consumer Electronics*, 1992.

[20] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. *First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, 2004.