A Virtual Fence Based on Infra-Red Break Beams

18-748: Final Report

Alexei Colin and Nishant Parekh

05/05/2015

1 Abstract

Industrial automation often requires the motion of machines or robots to be confined to a designated area for safety of human workers that share the floor space. A virtual fence constructed out of wireless sensor nodes equipped with infra-red transmitters and receivers provides a cheap and flexible mechanism for monitoring crossings into and out of a designated region. A body that moves across the region boundary obstructs the line-of-sight path between sensor nodes and breaks the infra-red break beam. The boundary crossing event can be used to activate a fail-stop mechanism on the machine and to notify the operators. We present the design and implementation of the hardware and software components of a virtual fence system as an application of a wireless sensor networking platform based on the Firefly node and Nano-RK operating system.

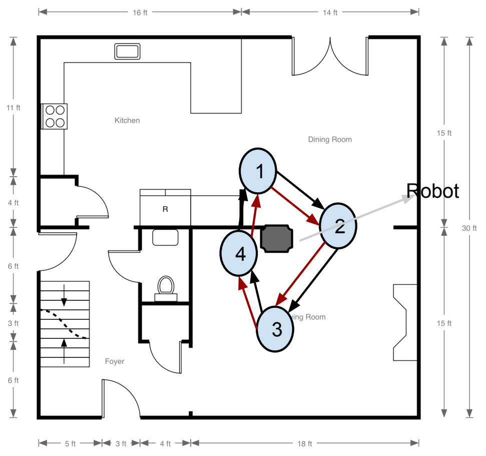


Figure 1: A virtual fence confining a robot to an area, implemented using wireless sensor nodes with infra-red beams.

2 Introduction

For safety and security it is often necessary to restrict the motion of machines or humans to the interior or the exterior of a designated area. For example, in large modern factories, the floor space is divided into lanes for vehicles and paths for pedestrians in order to avoid collisions while maintaining flexible mobility and reasonable throughput. Similarly, in areas where robots operate alongside human workers, the mobile robots must be confined to areas which are off limits to humans and vice-a-versa in order to prevent accidents. However, a demarcation alone without any enforcement mechanism provides limited protection for the same reason that traffic rules cannot eliminate road accidents. A driver's error or a malfunction in a mobile robot may lead to a transgression beyond a demarcated boundary and cause an accident. The objective of the present project is to design and prototype a boundary enforcement mechanism that is cheap, flexible, robust, and simple to deploy.

A baseline enforcement mechanism is a physical fence made out of a rigid material that can contain a vehicle or a machine by force. This mechanism provides high safety but is very inflexible and may constrict mobility inside the factory unnecessarily, decreasing production efficiency. Moreover, a physical fence is expensive to install and to modify, which is likely to be needed as new production lines are installed and old ones upgraded. A *virtual* fence with easily movable poles is an alternative that trades off the level of safety guarantee in return for flexibility and cost. The service provided by a virtual fence is to generate a notification in the form of an electrical signal in the event that the boundary has been crossed. This signal can then be relayed to the vehicles or machines near the breach to trigger an onboard safety stop mechanism as well as communicated to humans in the form of an audible or visible alert.

The goal of this project is to investigate a virtual fence implemented using a wireless sensor network in which nodes act as fence poles and infra-red (IR) break beams act as fence sections. Wireless sensor networks (WSN) are a synergy between the growing need to embed processing and communication into the environment and the scaling of electronic components to ever smaller size, energy usage, and cost. A virtual fence built using a WSN inherits the benefits of the WSN platform: low hardware cost with negligible incremental cost per pole (per node) and inexpensive deployment thanks to minimal wiring, when wall powered, or no wiring, when battery powered. Unfortunately, it also inherits the limitations of the WSN platform: the limited lifetime of the battery, which introduces a maintenance cost. In this project, we rely on energy-efficient communication protocol and hardware developed as a result of extensive research in the WSN domain. An unique advantage of a virtual fence is the versatility of the boundary crossing event: it can be relayed across the fence network and through a gateway to local and global networks for alert generation and coordinate response at a higher level, such as sending emergency personal to the location.

The following sections first describe how a virtual fence would be used in several operating modes, and then present the design and implementation of the system.

3 Operating Modes

3.1 Use case 1: Manually- specified fence

This use case involves using the topology of the system created by the nodes in IR proximity of one another.The nodes placed arbitarily from one another will form a basic relative topology according to the IR beams. This topology map can be displayed on the Graphical User Interface. This topology now shows possible connections to construct the fence.The user leverages this information and constructs the fence to secure the area In this use case there will be certain localization and the nodes are seen as a part of the external environment or room. The user now should be able to form the map without actually being in the room or observing the position of the nodes in the room. This usecase involves incorporating the localization information algorithm with the system.

3.2 Use case 2: Automatic maximum-area fence

The virtual fence system supports calculating and maintaining a fence that encloses the maximum area given the node distribution in space. The user can choose this mode so as to enclose the largest area possible. The fence thus formed may or may not contain all the nodes in the system.

4 System Architecture

4.1 Overview

The virtual fence system is a cooperation of a set of hardware and software components. The hardware layer consists of a set of Firefly wireless sensor network nodes each equipped with a custom daughter board. Each daughter board houses an array of infrared transmitters and receivers. The software layer consists of the modules that run on the target Firefly node and the modules that implement the control user interface on the host workstation. The system architecture is summarized in Figure 2.

One of the nodes in the network performs the function of a gateway in additional to its duties as a regular virtual fence node. This node is connected to a workstation via a USB interface and provides the user console, which is a part of the control center user interface into the system. The user interface exposes a set of commands for setting up a virtual fence as well as for diagnosing the system. The gateway node is responsible for communicating with all the other nodes to carry out the user's commands.

The hardware and software components are described in detail in the following sections.

4.2 Daughter board

The daughter board consists of two boards mounted on top of each other and interconnected with four headers along the perimeter. The schematics for the top and bottom layers are shown in **Figure 4**. The top board consists of an array of eight IR transmitters and an array of four IR receivers. The arrangement of the IR LEDs and receivers is shown in **Figure 3**. The IR transmitters are placed in eight directions: North, North-East, East, South-East, West, South-West, South, and North-West. The IR receivers are mounted in four directions: North, East, West, and South. The bottom board contains transistors for driving the IR LEDs, a 3:8 decoder, and a digital compass IC. THe 3:8 decoder multiplexes three GPIO pins in order to drive eight IR LEDs. The number of

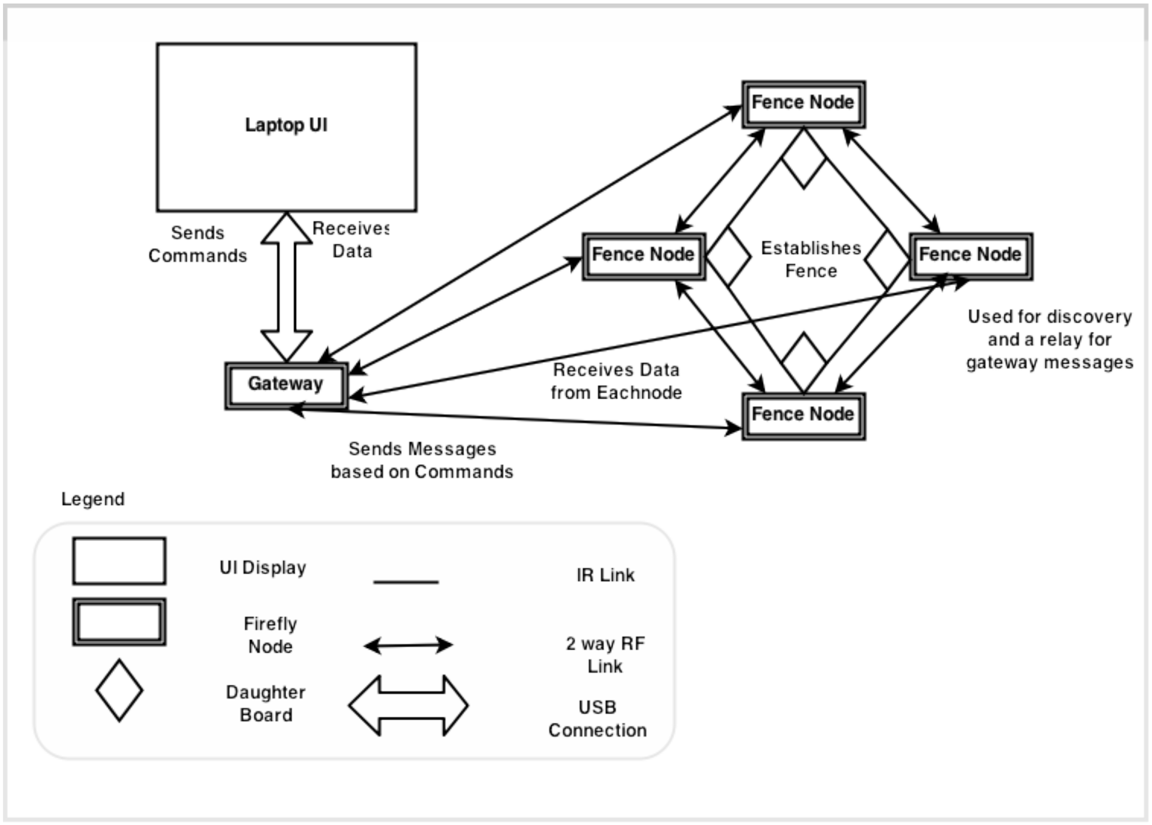


Figure 2: System architecture

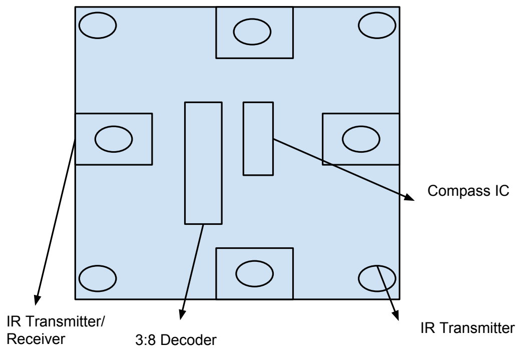


Figure 3: Daughter board model

GPIO pins available on the Firefly platform cannot accommodate eight lines without multiplexing. The compass IC interfaces with the Firefly microcontroller using a I2C interface.

4.3 Control Center user interface

During the exploitation of the virtual fence, the user's primary window into the system is the control center user interface. This software application runs on a host machine which has a direct connection to the gateway node via a USB wire. The primary functions of the user interface are:

- Display the state of the system:
  - Wireless link topology
  - Infra-red (IR) link topology
  - Location of the nodes on a map
  - Event log from the gateway node
- Accept specification of a fence from user
  - as a sequence of nodes (listed in as a CLI argument or selected on map)
  - as an area indicated on the map
- Notify of fence boundary crossing events

The virtual fence node network is self-sufficient, once setup and configured, the fence operates regardless

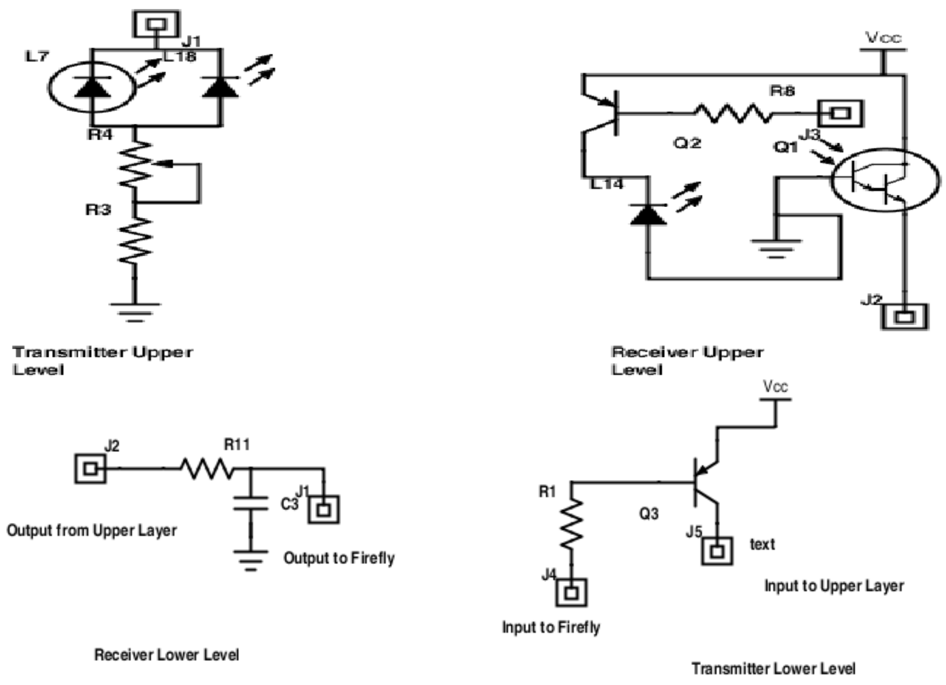


Figure 4: Daughter board schematic

of whether the control center is running on the host computer. However, if relaying of boundary crossing events to other networks (e.g. to a wireless or wired local area network) is desired, then this functionality would depend on the host computer.

The control center communicates with the virtual fence network, specifically, to the gateway node, through a text-based human-readable protocol over UART. A design goal is to be retain the ability to diagnose and monitor any node by connecting a terminal emulator directly to its UART. The control center would use the same interface as the human user at the console. The text-based interfaces consists of a set of commands, some of which are listed in Table 1. For example, issuing the command `graph` prints the wireless link topology in DOT format, such as `graph G { 1 -- 2; 2 -- 1; }`. A human user may read this output directly on the console, and the control center would parse it and display it in the graphical window.

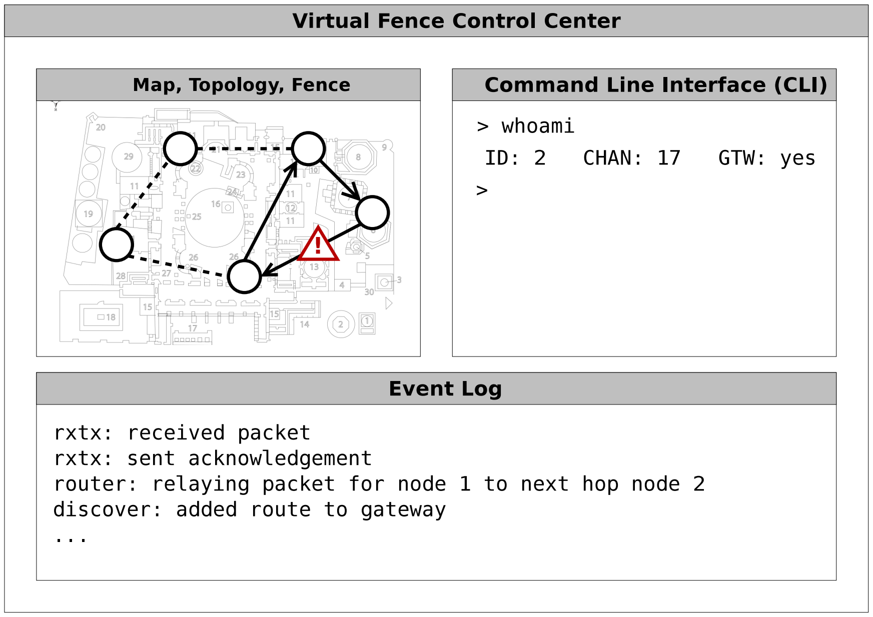
In order to enable fast development and tuning of the system, the node firmware features a persistent configuration module for manipulating parameters at runtime and saving/loading them from EEPROM. This eliminates the need to reprogram the node to change a parameter. For example, the number of packet retransmission attempts can be inspected, modified, and saved to persistent storage, by issuing the following commands:

```
> get num_retx
3
> set num_retx 5
> save
```

For accepting the fence specification from the user input, the control center would translate it into a set of commands. For example, if the user chooses the fence poles to be the sequence of nodes 3, 1, and 4, the corresponding commands to create and tear down a virtual fence would be `fence + 3 1 4` and `fence -`, respectively. Other commands and options, mainly for development purposes, allow to manually configure routes and to overlay a fake *topology mask* to "hide" links that would otherwise be visible when nodes are located close to each other on the bench.

4.4 IR neighbor discovery and localization

A virtual fence is composed of a sequence of juxtaposed *sections*. In the proposed IR fence, each section is a break beam between two nodes in one of the two possible directions. In order to construct and maintain a fence, it is necessary to know the IR link topology graph and the node locations in space. The link topology contains the information about which pairs of nodes can establish an IR break beam be-



```
> whoami
 ID: 2   CHAN: 17   GTW: yes
>
```

```
rxtx: received packet
rxtx: sent acknowledgement
router: relaying packet for node 1 to next hop node 2
discover: added route to gateway
...
```

Figure 5: Virtual fence control center user interface

Command	Description
`echo`	print the given args
`clrled`	turn off all LEDs
`setled`	turn on an LED for a duration
`set`	set option value
`get`	get option value
`save`	save options to eeprom
`load`	load options from eeprom
`whoami`	print node id and rf chan
`top`	configure topology mask
`link`	add link to topology mask
`unlink`	remove link from topology mask
`save-routes`	save routing table
`load-routes`	load routing table
`neighbors`	list neighbors
`routes`	show routing table
`route`	change a routing table entry
`ping`	send a ping
`graph`	print network topology graph
`bc-routes`	broadcast routes
`mping`	send a ping msg

Table 1: A subset of commands provided by the control interface

tween them and in which direction. The pseudo-code in Algorithm 1 defines the procedure by which this information is discovered autonomously by the nodes. This procedure needs to be triggered by the user once after deploying the fence and anytime after any node has been moved in space.

The procedure in Algorithm 1 also records the information necessary for localization of the nodes. The map L maps each edge in the IR topology graph to the LED identifier by which the beam was generated and the RF signal strength (RSSI) between the sending and the receiving node. This two pieces of information are sufficient to localize the nodes up to a translation of the whole map. First, the LED identifier is mapped to an *absolute* angle of the beam by reading the compass sensor and using the known layout of the IR LED transmitters on the daughter board (Section 4.2). Second, the RF signal strength (RSSI) is mapped to a distance by using a pre-calibrated table. [1] Figure 6 illustrates how (relative) location is calculated from the absolute angle of the beam, α, and distance to the receiver, d. Algorithm 2 outlines how this information is used to assign coordinates to all nodes in the graph relative to a reference node. This basic version of the algorithm traverses each

[1]The measurement error in the distance is expected to be high. The imprecision in node location does not fundamentally break the virtual fence, but it does degrade the user experience since the control center will have an inaccurate representation of the fence poles on the map.

Algorithm 1 IR Link Discovery

```
 1: function IR-DISCOVER
 2:     $s \leftarrow$ ID of this node
 3:     $G \leftarrow (E, V) \leftarrow (\emptyset, \emptyset)$                ▷ IR topology graph
 4:     $L[e \in E] \leftarrow$ nil                ▷ Raw data for later localization
 5:     for led $l$ in IR LED array do
 6:         turn on IR LED $l$
 7:         bounded wait for RF packet of type Beam-Detected
 8:         if packet Beam-Detected received from node $i$ then
 9:             $V \leftarrow V \cup i$                ▷ add node to IR topology graph
10:             $E \leftarrow E \cup (s, i)$                ▷ add edge to IR topology graph
11:             $L[(s, i)] \leftarrow (l,$ RSSI$)$                ▷ save localization data
12:         turn off IR LED $l$
13:     return $G, L$
14: function IR-LISTEN
15:     for receiver $r$ in IR receiver array do
16:         activate IR receiver $r$
17:     while True do
18:         wait for beam detected event from any IR receiver
19:         broadcast Beam-Detected packet
```

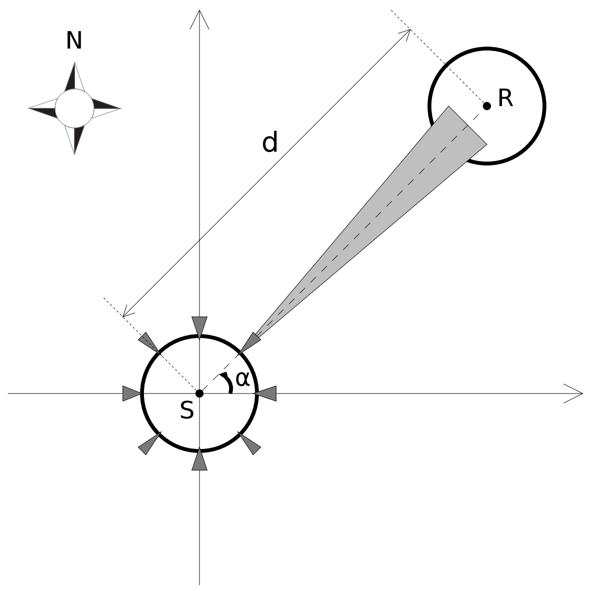


Figure 6: Fence pole (node) localization based on IR LED transmitter ID, compass reading, and RF signal strength

node only once, however a data point exists per edge, which yields multiple location estimates per node. An enhanced version might traverse all edges and average the estimates.

Algorithm 2 Node Localization

```
 1: function LOCALIZE($G = (V, E)$, $\alpha$, $d$, $(x_r, y_r)$)
 2:     $Q \leftarrow \{r\}$          ▷ Start at the reference node
 3:     while $Q \neq \emptyset$ do
 4:         $i \leftarrow$ POP($Q$)
 5:         $W \leftarrow W \cup i$                ▷ Mark the node
 6:         for $j \in (i, j) \in E$ do   ▷ IR-neighbors of $i$
 7:             $x_j \leftarrow x_i + d_{i,j} \cos \alpha_{i,j}$
 8:             $y_j \leftarrow y_i + d_{i,j} \sin \alpha_{i,j}$
 9:             if $j \neq W$ then
10:                 PUSH($Q$, $j$)
```

4.5 Fence construction

In Use Case #1, the fence is explicitly specified by the user as a sequence of poles, i.e. nodes. The job of the system is then to instruct the relevant nodes to shine IR break beams into the correct direction. This is accomplished by sending each node a packet with the direction of the beam, and a later packet to verify the state of each node to confirm.

In Use Case #2, the system automatically constructs the maximum area fence without any input from the user. This is accomplished by localizing the nodes (see Section 4.4) and finding the nodes that lie on a "hull" that encloses all the nodes. The resulting hull might or might not be convex, since IR links do not exist between all nodes. The algorithm that computes the maximum-area fence is listed in Algorithm 3. [2] The procedure starts with a node s on the hull. The specific node on the hull is determined by the reference axis relative to which angle α is defined. For a horizontal axis (east-west), the starting node must be the node with the lowest y coordinate: $s = \arg\min_{i \in V} y_i$. An illustration of the algorithm steps in action on example node locations and IR links of five nodes is shown in Figure 7. A distributed version of the same algorithm might be possible and might be investigated within this project.

Algorithm 3 Maximum Area Fence

```
1: function MAX-FENCE($G = (V, E)$, $s \in V$)
2:     $F \leftarrow \emptyset$                    ▷ Ordered set of fence nodes
3:     $i \leftarrow s$
4:     repeat
5:         $F \leftarrow F \cup \{i\}$                 ▷ Add node to fence
6:         $i \leftarrow \arg\min_{k \in (i,k) \in E} \alpha_{i,k}$    ▷ "Rightmost" node in CCW direction
7:     until $i == r$
8:     return $F$
```

4.6 Networking stack

For the proposed virtual fence communication over RF is an indispensable component of importance equal to that of the IR component. Figure 8 outlines the communication software stack. At the lowest layer data transmission capability is provided by widely used 802.15.4 radio and the B-MAC stack. On top of BMAC is a custom RX/TX layer which implements reliable transmission of packets over one link. The RX/TX layer uses a receive queue to decouple packet processing from listening in order to keep the radio in listen mode whenever it is not transmitting, which minimizes the risk of losing a packet because the node was busy doing something else. The router provides an API for sending relayed messages between any two nodes. The API is designed such that multiple independent applications can use the router simultaneously. For example, there may be a simple ping-pong application task that sends ping messages and handles pong messages running alongside the virtual fence applications. It is the responsibility of the router to discover and maintain routes, which is described in Section 4.6.2. Also the router maintains a hop path of a bounded length and a hop count in the packet as it relays it.

[2] The algorithm is inspired by Graham's Scan algorithm for computing the convex hull of a set of points. As opposed to the convex hull, the fence problem lends itself to a greedy algorithm.

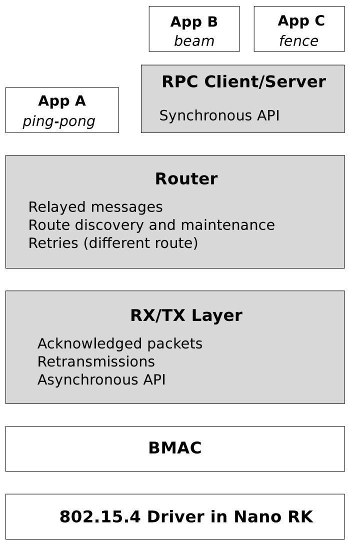


Figure 8: Networking stack (shaded areas indicate custom components designed and implemented for this project)

The two design goals that a communication stack over the RF link must fulfill to successfully support a virtual fence application are *low latency* and *reliability*. Latency is relevant for the virtual fence in order to detect a fence crossing and notify the safety monitor before the misbehaving robot goes too far out of the fenced region. Reliability is important for a virtual fence because the fence crossing event delivery must happen even if some intermediate nodes malfunctioned. Our design is optimized for low latency thanks to routing according to the next hop routing tables, which contain the next hop along the shortest path to destination (by hop count). See Section 4.6.2 for info on computing the shortest-path routing tables. Our design is optimized for reliability by acknowledging packets upon each hop at the lowest RX/TX layer, by attempting to relay messages multiple times (over different routes), and by healing the routing tables when routes are determined to be broken.

4.6.1 Router Interface

Our network supports routed messages between any two nodes. Multiple applications (Nano-RK tasks) can use the routing layer at the same time. The API into our router consists of

- `send_message(recipient, payload, length)`
- `receive_messages(sender, handler)`
- message received signal

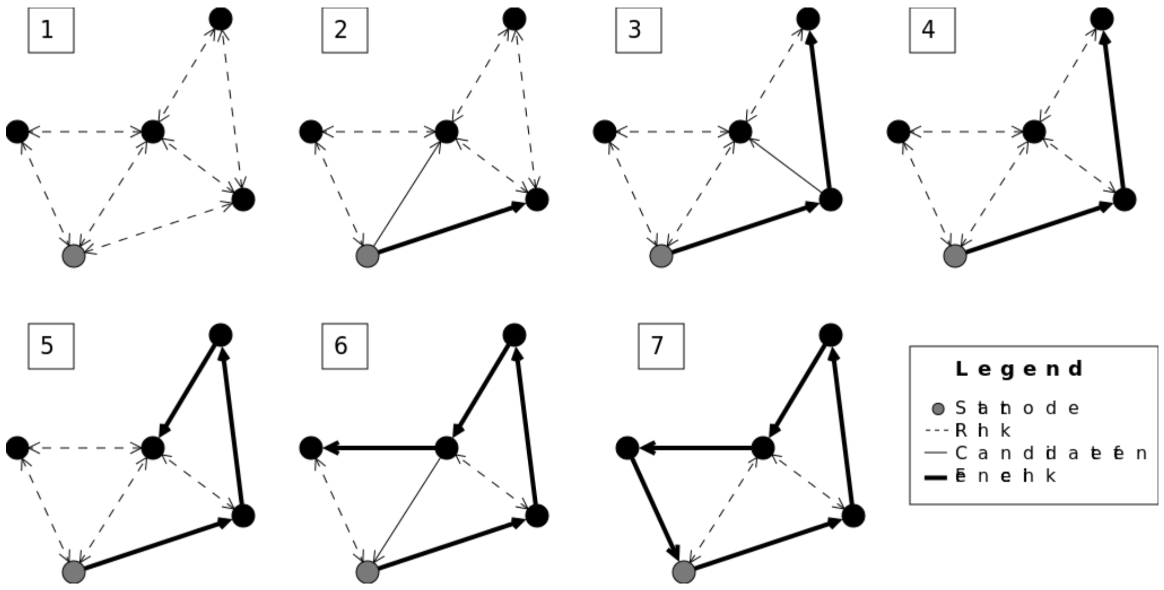


Figure 7: Fence construction algorithm steps in action on an example node locations and IR links

The task that uses the router API is responsible for calling `receive_messages` upon receiving the signal. Within this call, the handler callback provided in the second argument will be invoked for each message. The callback receives arguments `payload` and `length`. The `send_message` is designed to be invoked from tasks other than the `router`, and enqueues the messages onto a queue, which is asynchronously processed by the router task.

4.6.2 Network Discovery and Routing Table Calculation

The router relays messages based on a routing tables that it maintains on each node. The routing table is a map from the destination node ID to the next hop neighbor node ID. There is an entry per each node ID in the network. Although this is not scalable in general, it is sufficient for small networks. The routing tables for all nodes are computed centrally from the complete network topology graph and then distributed to the nodes. This computation consists of running Dijkstra's shortest path algorithm with unit weights once per node and each time setting the routing table entry to the first hop from the shortest path. The advantage of the centralized approach is its extensibility to more complicated routing algorithms which operate on the full topology graph, such as energy-aware algorithms which balance the traffic across nodes. To make use of another routing algorithm, only the algorithm routing itself needs to be substituted without any change to the router and route distribution code. A disadvantage is that the complete graph must be gathered on one node before routes can be computed, which is a time consuming and potentially costly process. We mitigate this problem through a local route healing mechanism that can update the routing table locally, eliminating the need for frequent global route re-computations. The route computation happens on the gateway node according to Algorithm 4.

4.6.3 Route Healing

For robustness, each router maintains its routing table by "healing" broken paths. This happens each time a message relay attempt fails at the router layer, which implies that a packet transmission had failed to receive an ack despite several re-transmission attempts. For example, if a node A gets a message to C from D, and A's routing table says that the next hop to C is B, then A sends the packet to B, but B is dead. If A does not succeed in relaying the packet to B (no ack), then A updates its routing table to no longer list B as the next hop for C. Instead, A picks a neighbor X from its neighbor list (other than B and other than the packet source node D) and sets the routing table entry for C to this neighbor X. [3] On next message transmission attempt, A will not forward it to B but to the different neighbor X.

[3]Currently, the choice of neighbor is random, but in the future it could be based on last heard time and RSSI, both of which are tracked.

Algorithm 4 Route Discovery

```
function MAINTAIN-ROUTES
    $v \leftarrow 0$                                          ▷ Distinguish runs of the discovery procedure
    while True do
        $v \leftarrow v + 1$ DISCOVER-ROUTES($v$)
        sleep for the rest of the period
function DISCOVER-ROUTES($v$)
    $G \leftarrow (V, E) = (\emptyset, \emptyset)$                                    ▷ Topology graph
    broadcast Discover($v$, this node ID) packet
    while elapsed time < route discovery timeout do
        wait for Discovered packets
        $P \leftarrow$ hop path extracted from the packet
        for node $i$ in path $P$ do
            $V \leftarrow V \cup i$
            $E \leftarrow E \cup (i, i+1)$
    $R \leftarrow$ COMPUTE-ROUTING-TABLES($G$)                    ▷ Invoke Dijkstra's algorithm
    broadcast Routes($R$) packet
function LISTEN-FOR-DISCOVERY
    discovered $\leftarrow 0$                     ▷ Reset the mark (will store discovery version)
    while True do
        wait for Discover($v, g$) packets from any node $s$
        RoutingTable[$g$] $\leftarrow s$                                  ▷ Needed for relaying
        send a Discovered response back to $s$
        broadcast Discover($v$)                                    ▷ Propagate to neighbors
        discovered $\leftarrow v$                                ▷ Mark the node for this iteration
function RELAY-DISCOVERY
    while True do
        wait for Discovered packets from any node $s$
        relay the packet to $g$ by sending it to RoutingTable[$g$]
function LISTEN-FOR-ROUTES
    $V \leftarrow 0$                                ▷ At boot time route table version is invalid
    route table $\leftarrow \emptyset$
    while True do
        wait for Routes($v, R$) packets
        if $v > V$ then
            RoutingTable $\leftarrow R$[node ID of self]
            $V \leftarrow v$                                         ▷ Bump route table version
```

5 Challenges

Our implementation and testing of the virtual fence prototype system has uncovered several design and implementation challenges that an engineer charged with a similar task might find useful. These challenges and the relevant design choices are documented in this section.

5.1 Sensitivity of the IR sensors

The infra-red sensors as explained earlier reach a peak sensitivity for a 38 kHz carrier wave frequency. Initially we designed circuit and software to achieve this peak frequency for best performance. However, the highest sensitivity turned out to not be desirable for the virtual fence application. At their highest sensitivity, the receivers detect a signal from the IR LEDs mounted on the node which point away from the receivers. This false-positive beam detection error is undesirable in the fence formation logic. Furthermore, the receiver detection logic appears to be based on the change in ambient IR intensity. The change is highest when an IR LED (including one on the same board as the receiver) is first turned on. This is favorable for detecting an incoming IR beam, but exacerbates the false-positives described above.

To mitigate this issue, we have changed the carrier frequency to 50KHz, which reduced the receiver sensitivity to 20% of the peak. In addition, we enclosed the receivers in a plastic heat-shrink tube with an opening in order to attenuate signals coming from anywhere but outside of the board.

5.2 Magnetic interference

To localize the nodes on a map, a digital magnetometer (compass) was mounted on each daughter board. The compass provides a heading value, which indicates the orientation of the node relative to the magnetic north of the Earth. The compass is affected significantly by stray magnetic fields and proximity to ferro-magnetic objects, including batteries attached to the Firefly node at the bottom side of the board. This interference manifests in highly inaccurate readings that are hardly correlated with the orientation of the node.

To reduce the amount of interference, we have removed the batteries from the holder and placed them in a holder that could be placed a few centimeters away from the node and connected by wires. Nevertheless, the accuracy of the readings as compared to a digital compass in a smart phone had an error on the order of 20-30 degrees in the best-case locations, and was unusable in some locations.

6 Potential Improvements

6.1 Fault-tolerant fence

A virtual fence may be made robust against node failures and persist in monitoring a region at least within the originally specified region should a failure occur. Consider a “fence section,” i.e. a beam, from node A, which actuates an IR transmitter, to node B, which senses the IR signal. Node A can monitor the state of node B using RF connectivity, and choose a different peer automatically should B fail. With a large network of nodes spread throughout an area, the same bounded region may be enclosed by many potential sets of beams. In this case, the monitored region would remain roughly unchanged despite the reconfiguration of the individual beams that make up the fence.

6.2 Two-way fence

A two way fence calls for two break beams formed by infra-red LEDs and receivers on the two nodes. They could indicate a sense of direction of the boundary crossing: into the region or out of the region. This information could help in choosing the most appropriate corrective action. The redundancy that comes with two beams also improves the robustness of the fence, which can continue operating despite a failure in one of the two beams.

6.3 Ultra-sonic ranging for localization

We used a combination of RSSI and compass to localize the nodes on a map. We observed in an indoor environment the RSSI readings were very imprecise. An ultra-sonic transceiver pair on each node would provide an accurate measurement of the distance between two nodes and yield better localization. Although the ultra-sound consumes a relatively large amount of power, it would only be used once per a pair of nodes. These types of transceivers are commonly used in small mobile applications for obstacle detection, which makes them suitable for the proposed virtual fence application.

6.4 Digital control of IR intensity

In our implementation an analog potentiometer is used as a knob to control the intensity of the IR LED. An alternative design based on a digital POT IC could enable to control intensity on all nodes in

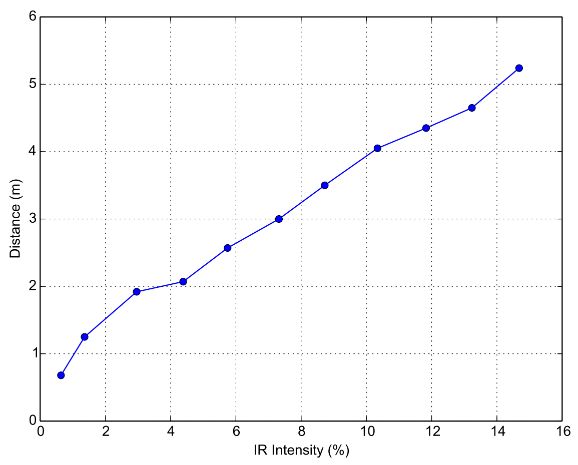


Figure 9: Maximum distance between nodes forming an infra-red beam as a function of intensity of the infra-red transmitter.

the network from the control center user interface via the gateway node. This would improve usability and reduce deployment time.

7 Results

A sequence of experiments were conducted in order to quantitatively assess the quality and limitations of the virtual fence system. The results are presented in this section.

7.1 Node separation distance

The maximum separation distance between two "fence poles," i.e. nodes, is constrained by the distance at which the IR signal from the transmitting LED can be detected. This, in turn, is determined by the intensity of the IR transmission and the sensitivity of the IR receiver. This section presents results from two experiments that quantify both effects.

7.1.1 IR transmitter intensity

The IR transmission distance is a function of the intensity of the transmitter (IR LED), which is determined by the current sourced through the LED. Our design allows the user to control the intensity by a potentiometer mounted on the top side of the daughter board. Figure 9 shows the relationship between the intensity of the LED set by the potentiometer and the measured maximum distance of the receiving node, measured indoors.

7.1.2 IR receiver sensitivity

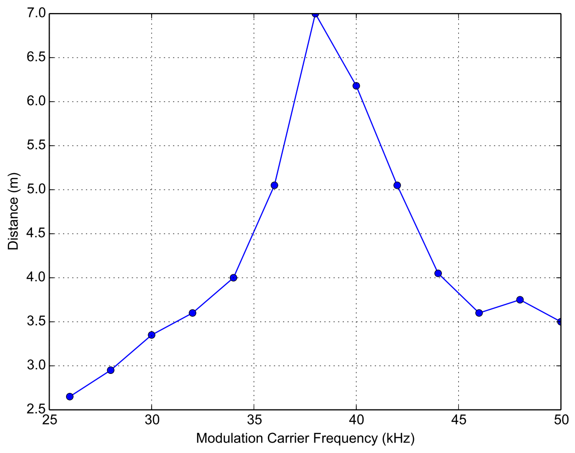
The distance at which the IR signal can be detected depends on the sensitivity of the IR receiver IC, which is a function of the frequency of the signal. The sensors used in our prototype reach their peak sensitivity for a carrier wave at 38kHz. Our software and hardware design permits the user to control the carrier frequency by setting a parameter in the control center console. This parameter sets the timer period for the timer that drives the transistor that switches the LED. In the experiment presented in Figure 10 we varied the frequency and measured maximum distance at which the signal is received, while holding the transmission intensity constant. For comparison, the figure also reproduces the relationship between carrier frequency and sensor sensitivity provided in the datasheet for the receiver IC.

7.2 Compass heading precision

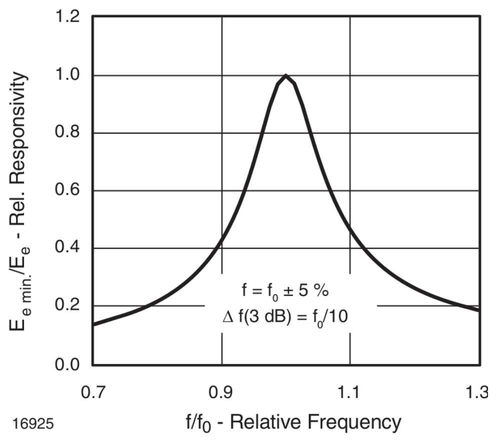
To localize the nodes on a map, the system requires an absolute orientation of each node. Our design relies the heading calculated from magnetic intensity readings performed by a compass IC, which is mounted on each daughter board. The accuracy of localization directly depends on the accuracy of the heading. Figure 11 quantifies the precision and accuracy in the heading as compared to a reference digital compass in a smart-phone. In the figure, the data from an ideal compass would be on the line $x = y$. Deviations from that line indicate an error in the readings. The observed error can approach 50 degrees. Although not shown in the figure, the observed readings are precise: low variability across multiple readings while the node is held stationary. Our observations of the performance of the compass cast doubt on whether a commodity digital compass is an appropriate choice for this indoors application.

7.3 Power consumption

The longer a virtual fence can be active without a replenishment of batteries, the more practical is the system. To quantify the energy requirements of our prototype, we measured the current drawn by the Firefly node and the daughter board with a multimeter. Table 2 lists the total power consumption in the idle state without any IR beams and in active state in which the node transmits a beam and receives another beam. At an average node separation of 2 meters, this corresponds to about one week of active operation on two AA batteries (2500 mAh).



(a) Measured



(b) Datasheet

Figure 10: Maximum distance between nodes as a function of carrier frequency of the modulated infra-red signal

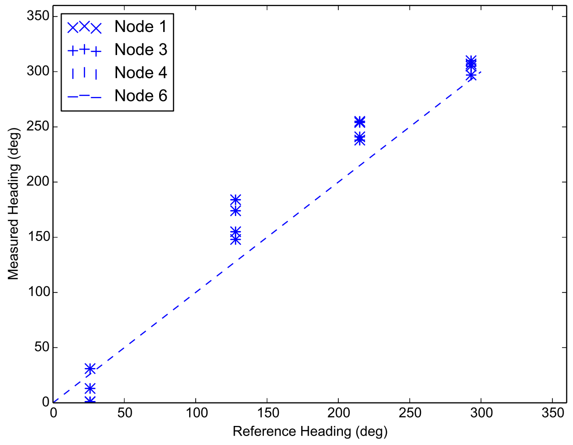


Figure 11: Accuracy and precision of the node orientation given by the digital compass plotted against the reference from the compass in a smart phone (Motorola Droid Mini).

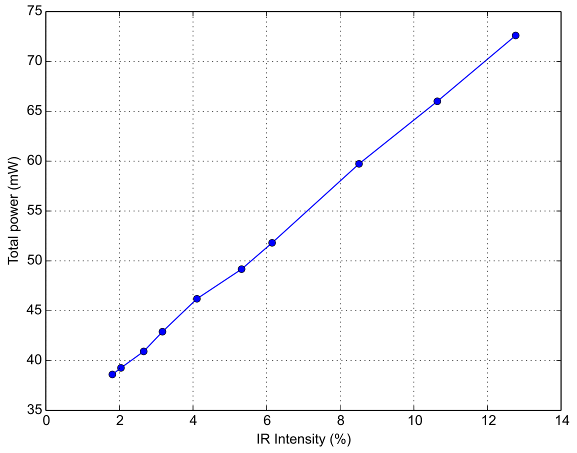


Figure 12: Total power consumption of the device as a function of intensity of the infra-red transmitter

State	Power (mW)
Idle state	15
Active state	40-75

Table 2: Total measured power consumption of the device in different operating modes.

7.4 Computational resources

To be useful the virtual fence application must be able to run on small platforms with limited memory and processing power. Table 3 lists the ROM and RAM memory required by each of the major components of the application. The total of 121KB of ROM and 12KB of RAM uses 95% and 75% of the flash memory and SRAM, respectively, available on the Atmega128RFA1 microcontroller in the Firefly platform. The ROM memory demand may be significantly reduced by eliminating diagnostic log output code and message strings. Processing capacity is not a bottleneck for the virtual fence application. The Atmega128RFA1 microcontroller running at 8MHz is sufficient to support 5 application tasks, 3 infrastructure tasks, 5 networking tasks, and 2 MAC-layer tasks.

8 Demonstrations

We would be having two demos during the course of the project. An intermediate demo at the end of march and a Final demo at the end of the semester.

8.1 Intermediate demo

The intermediate project report demonstrated the discovery of radio connectivity topology and one section of the virtual fence formed by two nodes in action. The control center user interface displayed the graph of the radio connectivity between all nodes in the network and the state of a beam formed between two nodes.

8.2 Final demo

The final demo included the following scenarios:

Scenario 1:

1. The user would place the nodes in a room.
2. The connection between the nodes is displayed on the UI to the user.
3. The user then chooses a sequence of fence poles in the UI.
4. The resultant fence is formed.
5. The fence detects an intrusion and generates a notification.

Scenario 2:

1. The first two steps will be the same as in Scenario 1.
2. In this scenario the max area mode is set and fence is computed.
3. The fence is displayed to the user on the UI.
4. The fence is able to detect an intrusion and generate a notification.

9 Schedule

The project schedule and work distribution are listed in Table 4.

10 Conclusion

A system for constraining industrial machines or robots to a safe designated area can be constructed using a wireless sensor network of nodes that form a virtual fence out of infra-red break beams. The prototyped system consists of five Firefly nodes each equipped with a custom daughter board with eight infra-red transmitters and four receivers. The user can specify a virtual fence in the control center user interface as a sequence of "poles" (nodes) or as an automatically determined maximum area enclosure and monitor the node and beam state on a graphical display. Once formed, the virtual fence monitors the region boundaries and generates an alert in the user interface. The system design choices, challenges, and potential improvements were documented in this report. The prototype built was used to experimentally evaluate key aspects of the virtual fence application, including maximum node separation and power consumptions. The system was demonstrated in action during a project showcase.

Component	**Size (lines)**	**ROM (KB)**	**RAM (KB)**
Nano-RK OS, drivers, BMAC (*)	23000	23.134	0.971
Networking stack	3008	70.43	6.255
I2C driver (*)	751	0.420	0.019
Infrastructure	3239	45.886	2.604
Fence application	3112	31.642	3632
Total	33110	121	12

Table 3: Code size and memory requirements of the virtual fence application. Infrastructure includes console, persistant parameter store, led indicator pulsing, ping-pong diagnostics, and basic data structures. All components without a (*) have been designed and implemented within this project.

Start	**Completion**	**Activity Description**	**Responsible**
02/01	02/10	Initial understanding and feasibility of the project	Both
02/11	03/15	Hardware daughter board design	Nishant
02/11	03/15	Routing layer and simulator for development purposes	Alexei
03/15	03/31	Daughter board assembly	Both
04/01	04/15	Discovery of IR topology and localization	Alexei
03/15	04/15	Algorithms for constructing fences for all use cases	Both
04/15	04/30	Experimental evaluation	Nishant
04/30	05/05	Testing and tuning	Both

Table 4: Schedule and work Division