# CrowdSurf: Real Time Crowd Monitoring Project for Indoor Spaces

## ECE 445 Design Document Spring 2026

Team #50

Johnathan Abraham, Tanvika Boyineni, Ananya Krishnan

Professor: Arne Fliflet

TA: Aniket Chatterjee

# 1 Introduction

## 1.1 Problems and Solutions

Indoor public spaces such as university libraries, study lounges, gyms, and student centers routinely experience congestion during peak hours. Students frequently arrive at these facilities only to find them overcrowded, while facility staff lack real-time, room-level data on occupancy and directional traffic flow. This mismatch between demand and available capacity reduces comfort, productivity, and, in emergency situations, safety. Existing occupancy monitoring solutions fall into two broad categories: camera-based systems, which raise serious privacy concerns, demand significant computational infrastructure, and may conflict with institutional policies; and manual or badge-based counting systems, which are labor-intensive, error-prone, and typically yield only coarse aggregate estimates rather than actionable, localized data. The absence of a low-cost, privacy-respecting, real-time monitoring solution represents a meaningful gap in campus infrastructure management. Crowdsurf addresses this gap by providing a privacy-preserving, real-time crowd monitoring system that estimates room occupancy and directional flow using distributed, non-imaging sensor nodes installed at doorways. Each node uses a pair of Adafruit 2168 infrared (IR) break-beam sensors spaced approximately 10–20 cm apart. When a person passes through the doorway, the two beams are interrupted in a specific temporal sequence, allowing the embedded ESP32-WROOM-32 microcontroller to infer entry versus exit direction without capturing any identifiable information. Only processed, aggregate event data (IN/OUT counts, timestamps, node health status, and sequence numbers) is published

over WiFi using the MQTT protocol to a Mosquitto broker running on a central Raspberry Pi gateway. The Raspberry Pi acts as both the MQTT broker and the application server, maintaining a live room occupancy estimate and serving a React-based web dashboard. To ensure robustness against temporary WiFi outages, each ESP32 node maintains a local RAM buffer of unacknowledged events. Upon reconnection, buffered events are flushed to the broker in sequence-numbered order, allowing the gateway to reconcile any missed counts without data loss. Unlike camera systems, Crowdsurf collects no images or biometric data, making it appropriate for deployment in privacy-sensitive academic and public environments.

## 1.2 Visual Aid

Figure 1 below illustrates the high-level deployment context of Crowdsurf. Two doorways in a single enclosed room are each equipped with a sensor node. Each node contains two Adafruit 2168 IR break-beam emitter/receiver pairs mounted at consistent heights across the doorframe, connected to an ESP32-WROOM-32 microcontroller on a custom PCB. The ESP32 connects to a local WiFi hotspot hosted by the Raspberry Pi and publishes telemetry packets via MQTT to a Mosquitto broker running on the Pi. The Pi aggregates occupancy from both nodes and serves a live dashboard accessible via web browser to students, staff, and facility managers. In the event of a temporary WiFi interruption, each node buffers events locally and retransmits them once connectivity is restored. No images or personal identifiers travel across this pipeline at any point.

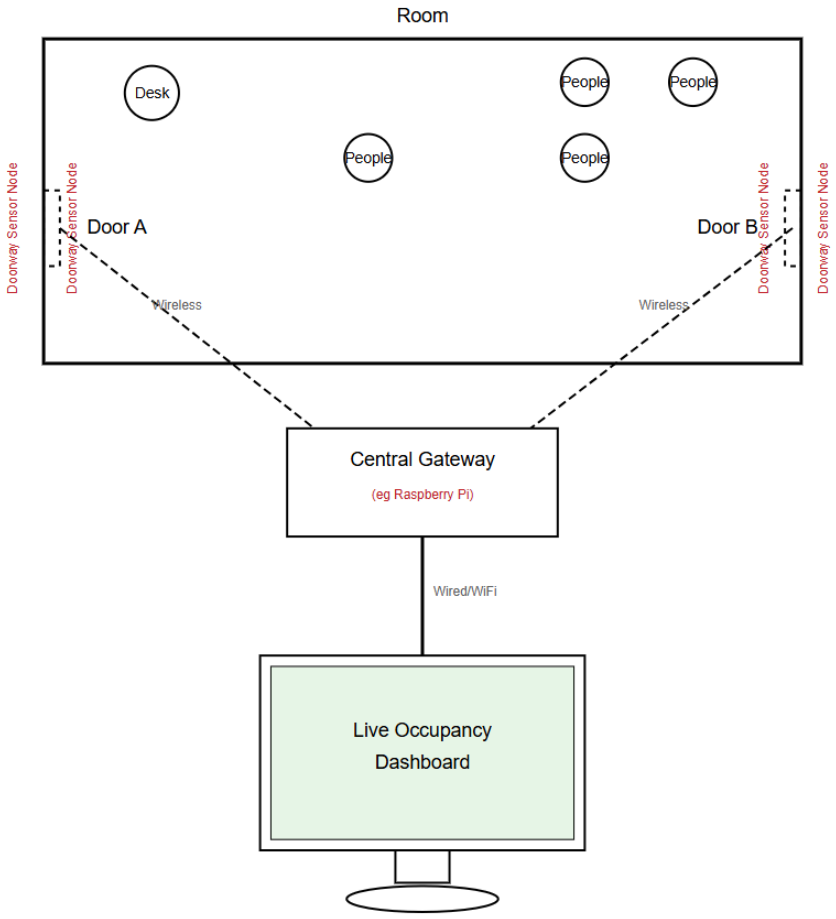


*Figure 1: Crowdsurf system deployment overview. Sensor nodes at each doorway communicate via WiFi/MQTT to a central Raspberry Pi gateway, which serves a live occupancy dashboard to end users.*

## 1.3 High-Level Requirements

To consider this project successful, the Crowdsurf system must satisfy all three of the following high-level requirements. These requirements are selected such that failure to meet any one of them would render the system unable to fulfill its core purpose as a reliable, real-time occupancy monitoring solution.

- The system shall achieve at least 90% correct IN/OUT directional classification accuracy at each monitored doorway under moderate, sequential pedestrian traffic

conditions. This threshold is required because an occupancy estimate built from systematically miscounted crossings degrades in accuracy over time and ceases to be useful for space management decisions.

- The live occupancy dashboard shall display an updated room occupancy estimate within 3 seconds of any doorway crossing event, under normal wireless operating conditions. This end-to-end latency requirement encompasses the full pipeline from sensor trigger through ESP32 processing, WiFi/MQTT transmission, Raspberry Pi aggregation, and browser display. A latency greater than 3 seconds would produce a dashboard that lags real conditions sufficiently to mislead users about current room state.

- The system shall remain continuously operational and automatically recover its internal occupancy state following temporary WiFi/MQTT packet loss, without requiring a manual reset or re-initialization. The system must sustain uninterrupted operation for a minimum of one hour while logging all occupancy and health data to persistent storage. This requirement is necessary because a system that loses its occupancy count on communication interruption would require constant human supervision, eliminating its value as an autonomous monitoring solution.

# 2 Design

## 2.1 Physical Design

Each sensor node is physically mounted at a doorway using a pair of small 3D-printed or off-the-shelf brackets that position one IR emitter and one IR receiver on opposite sides of the doorframe. The Adafruit 2168 break-beam sensors operate at 5V and produce a clean digital LOW signal on their receiver output whenever the beam is interrupted. The two sensor pairs (Beam A and Beam B) are mounted vertically at approximately standing hip height (approximately 90–100 cm from the floor) with a horizontal separation of 15 cm along the direction of travel through the doorway. This 15 cm spacing is the critical physical parameter for direction inference: a person entering the room will interrupt Beam A before Beam B, while a person exiting will interrupt Beam B before Beam A. The custom PCB is housed in a small enclosure mounted to the doorframe above or beside the sensor brackets, with strain-relieved wiring running to each emitter/receiver pair. The 5V wall adapter plugs into a barrel jack on the PCB. The Raspberry Pi gateway is placed anywhere within WiFi range of both nodes, powered by its own 5V USB-C supply. Figure 2 is a rough illustration of the setup for a single doorway.

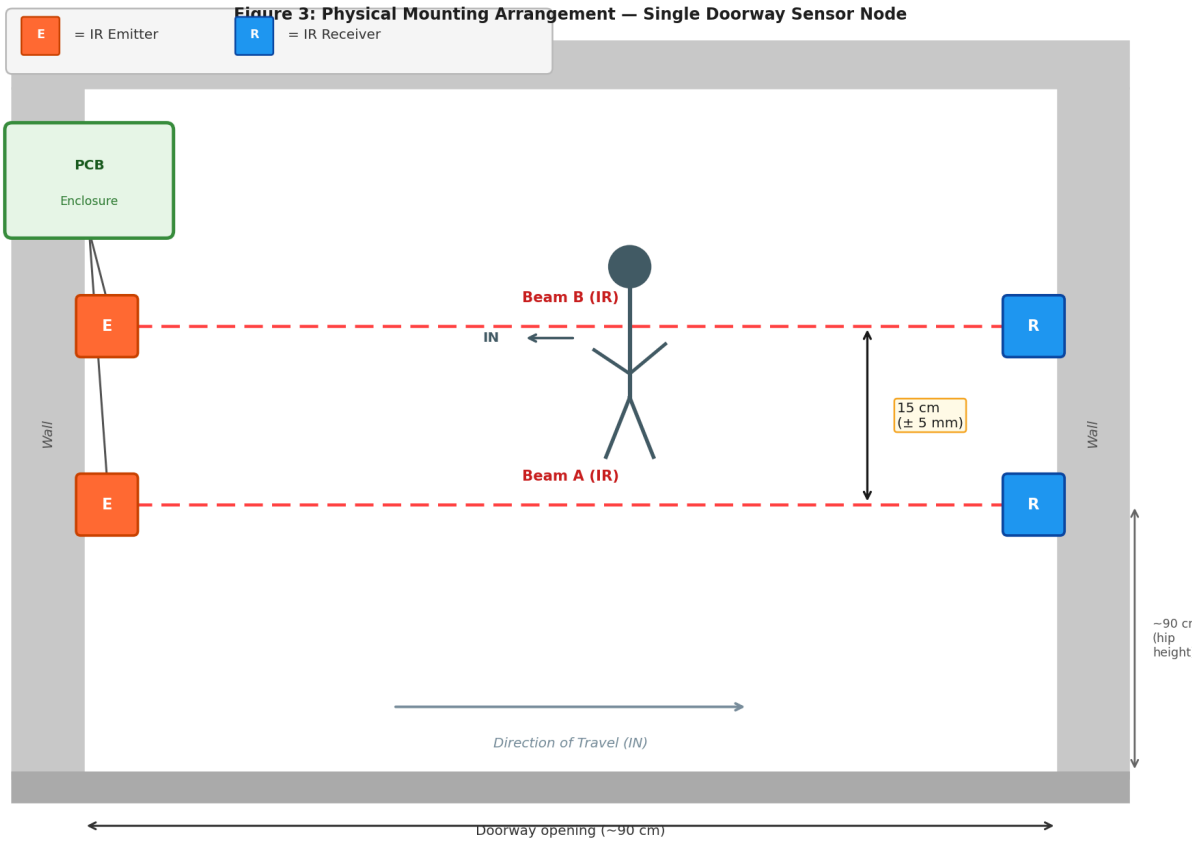


*Figure 2: Physical mounting arrangement for one sensor node. Beam A and Beam B are separated by 15 cm The custom PCB will be mounted above the doorframe.*

## 2.2 Block Diagram

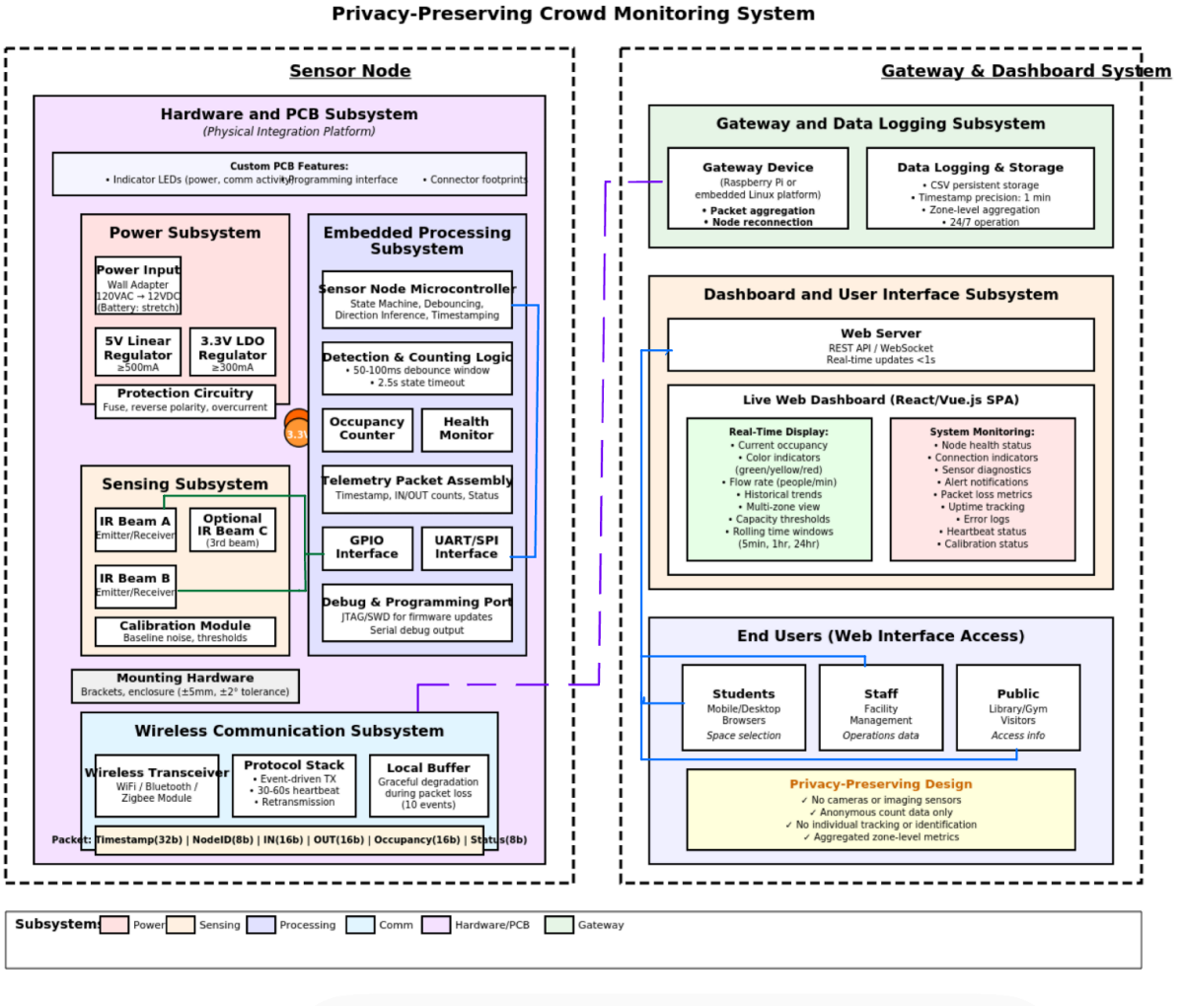


*Figure 3: Crowdsurf system block diagram showing all subsystems and their interfaces.*

The Crowdsurf system is divided into two major physical units: the Sensor Node and the Gateway & Dashboard System. The Sensor Node, deployed at each doorway, contains five subsystems: a Sensing Subsystem (two Adafruit 2168 IR break-beam pairs), a Power Subsystem (5V wall input regulated to 3.3V via LDO), an Embedded Processing Subsystem (ESP32-WROOM-32 running the direction-inference FSM), a Wireless Communication Subsystem (ESP32 built-in 802.11b/g/n WiFi with MQTT), and a Hardware/PCB Subsystem

(custom fabricated PCB integrating all node electronics). The Gateway & Dashboard System, running on a Raspberry Pi, contains a Gateway and Data Logging Subsystem (Mosquitto MQTT broker, CSV logging, occupancy aggregation) and a Dashboard and User Interface Subsystem (Node.js/Express REST API with WebSocket, React SPA).

## 2.3 Functional Overview & Block Diagram Requirements

### 2.3.1 Sensing Subsystem

The Sensing Subsystem consists of two Adafruit 2168 IR break-beam sensor pairs (emitter + receiver) mounted across each doorway with a 15 cm spatial separation along the pedestrian travel axis. The Adafruit 2168 operates at 5V with a beam range up to 10 cm and produces a digital output (LOW when beam is broken, HIGH when unobstructed) suitable for direct connection to ESP32 GPIO pins. A calibration procedure performed at installation measures ambient IR interference and sets the baseline. The subsystem contributes to HLR1 by producing reliable, low-noise digital transitions that the embedded FSM can sequence into directional counts, and supports HLR2 by generating near-instantaneous triggers upon beam interruption.

| Requirement | Verification |
|---|---|
| Beam A and Beam B must each generate a stable digital LOW transition within 5 ms of physical beam interruption under normal indoor ambient lighting (fluorescent or LED overhead, no direct sunlight on receiver). | Equipment: Logic analyzer (Saleae Logic 8 or equivalent, 10 MHz sample rate), 5V bench power supply (current limit 500 mA), opaque rod (~1 cm diameter). Procedure: (1) Power PCB from bench supply at 5.0V. (2) Connect logic analyzer CH0 to test point TP_BEAM_A on the PCB (Beam A receiver GPIO signal line). (3) Configure trigger on falling edge, 10 ms capture window. (4) Block Beam A with the rod at sensor height. (5) Capture the falling-edge transition and record the elapsed time from rod contact to LOW signal. (6) Repeat steps 3-5 five times and record all five values. (7) Repeat for Beam B using TP_BEAM_B on CH1. Results: Record all 10 transition delay values (5 per beam) in a table in the lab notebook. Pass criterion: all 10 values <= 5 ms. |
| Both sensor pairs must produce zero spurious LOW transitions over any 60-second window when no object is in the beam path, under typical indoor fluorescent or LED ambient lighting. | Equipment: USB-to-UART adapter (3.3V logic), laptop with serial terminal (115200 baud, 8N1), 5V wall adapter. Procedure: (1) Flash firmware build with debug logging enabled (prints a timestamped line to UART on every GPIO falling-edge interrupt for GPIO34 and GPIO35). (2) Power node from wall adapter. (3) Open serial terminal on laptop connected to PCB UART debug header (TX, RX, GND pins). (4) Ensure both beams are fully unobstructed and stable. (5) Run for 60 seconds under overhead fluorescent lights with no objects near the doorway. (6) Count the number of spurious-interrupt lines printed to serial terminal. Results: Record the total spurious event count and the full terminal log in the lab |

| | |
|---|---|
| | notebook. Pass criterion: 0 spurious transitions in 60 seconds for each beam. |
| Beam A and Beam B must be physically mounted with a center-to-center separation of 15 cm +/- 5 mm along the axis of pedestrian travel through the doorway. | Equipment: Steel ruler or digital calipers (resolution <= 1 mm). Procedure: (1) With both sensor brackets fully mounted and secured to the doorframe, measure the horizontal distance between the center of the Beam A receiver lens and the center of the Beam B receiver lens along the direction of pedestrian travel using the ruler or calipers. (2) Record the measurement. (3) Repeat on the emitter side and record. Results: Record both measurements (receiver side and emitter side) in the lab notebook. Pass criterion: both measurements between 140 mm and 160 mm. |
| The voltage at each ESP32 GPIO input pin connected to an IR receiver output must not exceed 3.3V +/- 0.1V when the beam is unobstructed, and must fall to <= 0.4V when the beam is interrupted, ensuring safe logic-level compatibility with the ESP32-WROOM-32 (absolute maximum GPIO input voltage: 3.6V). | Equipment: Digital multimeter (DMM, Fluke 87V or equivalent, resolution <= 0.01V). Procedure: (1) Power PCB from 5V bench supply, current limit 500 mA. (2) Ensure Beam A is unobstructed. Probe the Beam A GPIO input at TP_GPIO34 on the PCB with the DMM positive lead; GND lead to PCB ground. Record voltage. (3) Block Beam A with an opaque rod. Measure and record voltage at TP_GPIO34. (4) Repeat steps 2-3 for Beam B at TP_GPIO35. Results: Record all four voltage measurements in a table (beam unobstructed / interrupted for each of Beam A and Beam B) in the lab notebook. Pass criterion: unobstructed voltage <= 3.3V on both pins; interrupted voltage <= 0.4V on both pins; no reading exceeds 3.6V. |

### 2.3.2 Embedded Processing Subsystem

The Embedded Processing Subsystem is implemented on the ESP32-WROOM-32, which runs the Arduino framework firmware implementing a finite state machine (FSM) for direction inference. The FSM monitors interrupt-driven GPIO inputs from Beam A and Beam B and uses a configurable 500 ms timeout window to classify each crossing as a valid IN event, a valid OUT event, or an ambiguous event (incremented as a diagnostic counter). The firmware also implements 20 ms hardware debouncing, monotonically incrementing 16-bit sequence numbers on each transmitted packet, a local RAM event buffer (up to 10 unacknowledged events) for WiFi outage tolerance, and a periodic 2-second heartbeat MQTT publish. This subsystem directly enables HLR1 through deterministic direction logic, HLR2 by converting sensor events to MQTT packets within 200 ms of a valid crossing, and HLR3 by continuing to count locally during outages and reporting health status. The FSM operates as follows. In the IDLE state, both beams are unobstructed. When Beam A is interrupted first, the FSM transitions to BEAM_A_TRIGGERED and starts a 500 ms timer; if Beam B is subsequently interrupted before the timer expires, the crossing is classified as IN (entry) and the IN counter increments. If Beam B is triggered first, the FSM transitions to BEAM_B_TRIGGERED and the subsequent Beam A interruption classifies the event as OUT (exit). If neither second beam is interrupted before the 500 ms timeout, the event is logged as ambiguous and the FSM returns to IDLE.

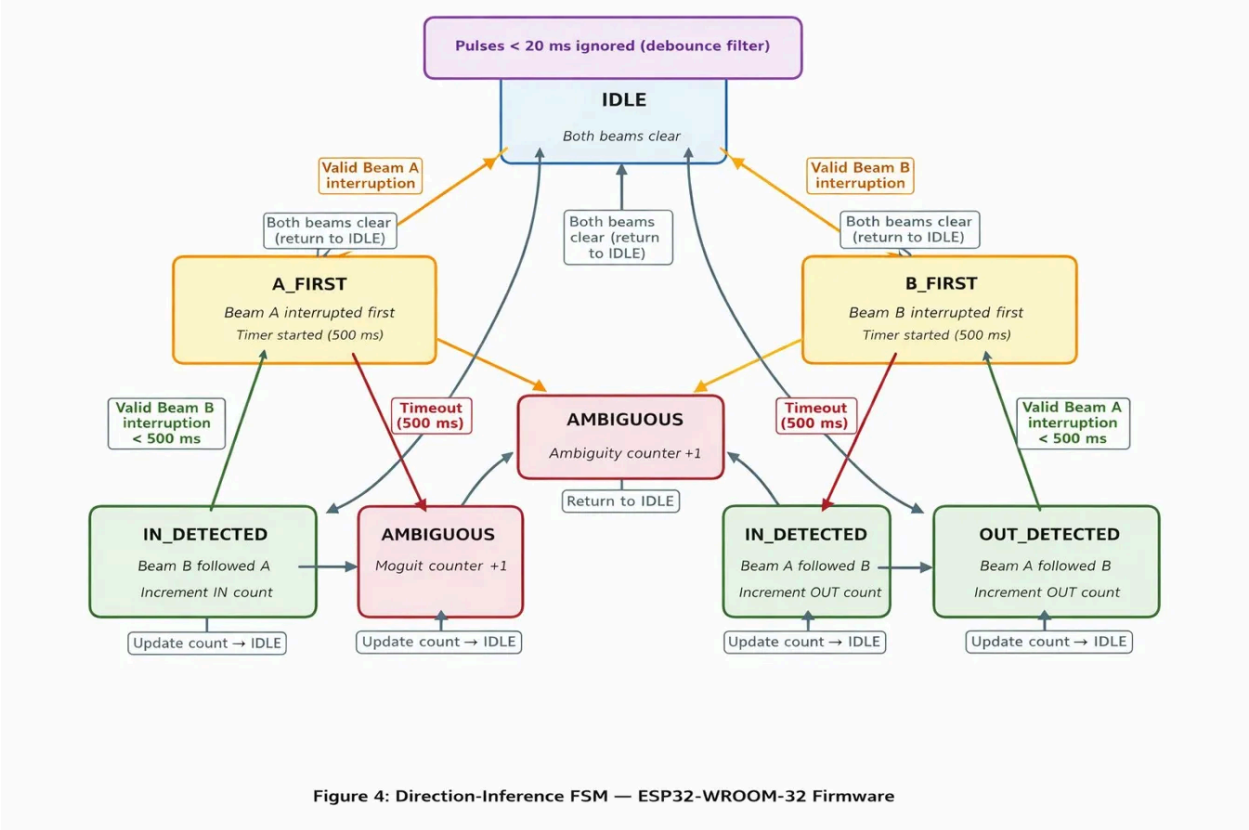


**Figure 4: Direction-Inference FSM — ESP32-WROOM-32 Firmware**

*Figure 4: Direction-inference finite state machine implemented on the ESP32-WROOM-32.*

| Requirement | Verification |
|---|---|
| The FSM must correctly classify a crossing as IN when Beam A GPIO (GPIO34) is interrupted before Beam B GPIO (GPIO35) within a 500 ms window, and as OUT when Beam B is interrupted before Beam A within a 500 ms window. Classification accuracy must be >= 90% (>= 18 of 20 trials correct) in each direction. | Equipment: USB-to-UART adapter, laptop serial terminal (115200 baud), fully mounted sensor node, opaque rod. Procedure: (1) Open serial terminal; enable firmware log mode that prints [IN], [OUT], or [AMBIGUOUS] on each classified event. (2) Slowly pass rod through doorway in the IN direction (crossing Beam A first, then Beam B) at approximately normal walking speed (~1.4 m/s). (3) Confirm [IN] is printed. (4) Repeat 20 times. Record each result (IN / OUT / AMBIGUOUS) in a tally table in the lab notebook. (5) Repeat steps 2-4 in the OUT direction (Beam B first). Results: Record a 20-row tally table for each direction in lab notebook. Pass criterion: >= 18/20 correct classifications for each direction. |
| The firmware debouncing must reject any Beam A or Beam B GPIO pulse shorter than 20 ms in duration, preventing false crossing events from vibration or electrical noise. Pulses >= 25 ms must be registered as valid beam interruptions. | Equipment: Function generator (e.g., Rigol DG1022Z) set to produce a square-wave pulse at 3.3V amplitude, USB-to-UART adapter, serial terminal. Procedure: (1) Connect function generator output to GPIO34 (Beam A input) through a 100-ohm series resistor for protection. Set generator to produce a single 10 ms LOW pulse at 3.3V high level. (2) Trigger the pulse and observe serial terminal. Confirm no [IN] or [OUT] event is logged. (3) Change pulse width to 25 ms. Trigger the pulse. Confirm a crossing event IS logged. (4) Repeat steps 2-3 three times each and record all results. Results: Record pulse widths and corresponding event/no-event outcomes in a table in the lab notebook. Pass criterion: 0/3 events logged for 10 ms pulses; 3/3 events logged for 25 ms pulses. |

| | |
|---|---|
| The firmware must transmit an MQTT heartbeat packet to topic node/[id]/heartbeat at least once every 2 seconds, continuously, containing node_id, sequence number, uptime_s, and a status byte. The inter-heartbeat interval must not exceed 2500 ms. | Equipment: Raspberry Pi running Mosquitto broker, laptop running mosquitto_sub CLI tool, stopwatch or terminal timestamp logging. Procedure: (1) Start Raspberry Pi hotspot and Mosquitto broker. (2) Power sensor node and allow it to connect to WiFi. (3) On the Raspberry Pi, run: mosquitto_sub -h localhost -t node/+/heartbeat -v \| ts '[%H:%M:%.S]' (using the 'ts' timestamp utility). (4) Record 30 consecutive heartbeat arrival times for each node. (5) Compute inter-arrival intervals from the timestamps. Results: Record all 30 timestamps and 29 inter-arrival intervals per node in a table in the lab notebook. Pass criterion: all inter-arrival intervals <= 2500 ms. |
| During a simulated WiFi outage of up to 30 seconds, the firmware must buffer crossing events locally (up to 10 events) and retransmit all buffered events to the MQTT broker in monotonically increasing sequence-number order within 5 seconds of WiFi reconnection, with zero events lost. | Equipment: Raspberry Pi (hotspot + Mosquitto broker + gateway CSV logger), USB-to-UART adapter and serial terminal for event logging on ESP32, stopwatch. Procedure: (1) Run the full system with both nodes connected. (2) Disable the Raspberry Pi WiFi hotspot (sudo nmcli radio wifi off or equivalent). Start stopwatch. (3) Manually trigger exactly 5 crossing events through the doorway during the outage, recording each crossing direction and time in the lab notebook. (4) After 30 seconds, re-enable the hotspot. Start a 5-second timer. (5) After 5 seconds, open the gateway CSV log and identify the 5 events by their timestamps. Results: Record the expected 5-event sequence and compare to what appears in the CSV log (sequence numbers, IN/OUT classification, order). Pass criterion: all 5 events present in CSV log, correct IN/OUT classification, sequence numbers in order, all entries appear within 5 seconds of hotspot re-enable. |

### 2.3.3 Wireless Communication Subsystem

The Wireless Communication Subsystem uses the ESP32-WROOM-32's integrated 802.11b/g/n WiFi radio and the MQTT protocol (via the PubSubClient Arduino library) to communicate with a Mosquitto broker running on the Raspberry Pi. The Raspberry Pi hosts a local WiFi hotspot (2.4 GHz, WPA2) requiring no external network infrastructure. Each transmitted telemetry packet includes: 32-bit timestamp (ESP32 millis()), 8-bit node ID, 16-bit sequence number, 8-bit IN delta, 8-bit OUT delta, 16-bit local occupancy, and 8-bit status flags. Event packets publish to node/[id]/events within 200 ms of a validated crossing; heartbeats publish to node/[id]/heartbeat every 2 seconds. This subsystem supports HLR2 by delivering events to the gateway within the latency budget, and enables HLR3 through reconnection and buffering.

| Requirement | Verification |
|---|---|
| An MQTT event packet must be received by the Raspberry Pi MQTT broker within 200 ms of a validated doorway crossing event, measured end-to-end from ESP32 beam-interrupt timestamp to Mosquitto broker receipt timestamp, under normal indoor WiFi conditions at a separation of up to 10 m with at least one interior wall obstruction. | Equipment: Raspberry Pi (Mosquitto broker with timestamps enabled in log), USB-to-UART adapter and serial terminal on ESP32 (timestamp of beam interrupt logged at millisecond resolution). Procedure: (1) Position sensor node 10 m from Raspberry Pi with one interior wall between them. (2) Enable Mosquitto verbose logging (log_timestamp true in mosquitto.conf). (3) Trigger a crossing event. (4) Record T1 = ESP32 serial log timestamp of beam-A interrupt (ms). Record T2 = Mosquitto log timestamp of event packet receipt (ms). Compute latency = T2 - T1. (5) Repeat 10 times. Results: Record all 10 latency values in a table in the lab notebook. Pass criterion: all 10 values <= 200 ms. |

| | |
|---|---|
| MQTT event packets must include a 16-bit sequence number that increments monotonically by 1 for each successive event published by a given node. The gateway must detect and log any gap in sequence numbers as a missed-packet warning entry in the CSV log. | Equipment: Raspberry Pi gateway CSV logger, mosquitto_sub CLI. Procedure: (1) Subscribe to node/+/events on the Raspberry Pi. (2) Trigger 50 crossing events in sequence. (3) Open the gateway CSV log and extract the sequence number column for all 50 events. Verify all 50 are consecutive integers (no gaps). (4) To test gap detection: temporarily disable MQTT publishing in the firmware for exactly 3 events (via a debug flag), then re-enable. Trigger 3 more events. (5) Inspect CSV log for missed-packet warning entries. Results: Record the full sequence number list from step 3 and the warning entries from step 5 in the lab notebook. Pass criterion: 50 consecutive sequence numbers in step 3; >= 3 missed-packet warnings logged in step 5. |
| The WiFi/MQTT subsystem must maintain stable connectivity (no MQTT disconnection events lasting >= 10 seconds) during 60 minutes of continuous operation under simulated 10% packet loss. | Equipment: Raspberry Pi with tc netem installed, USB-to-UART adapter and serial terminal monitoring ESP32 MQTT connection state. Procedure: (1) On the Raspberry Pi hotspot interface, apply 10% random packet loss: sudo tc qdisc add dev wlan0 root netem loss 10%. (2) Power both sensor nodes and confirm MQTT connection on serial terminal. (3) Run system for 60 minutes, triggering at least 1 crossing event per minute. (4) Monitor serial terminal output for any MQTT disconnect messages. Log each disconnect event with timestamp and duration. (5) After 60 minutes, remove the netem rule and inspect the serial log. Results: Record all disconnect events (if any) with timestamps and durations in the lab notebook. Pass criterion: zero disconnection events lasting >= 10 seconds over the 60-minute run. |

### 2.3.4 Gateway and Data Logging Subsystem

The Gateway and Data Logging Subsystem runs on the Raspberry Pi and consists of a Mosquitto MQTT broker receiving telemetry from both sensor nodes, a Python gateway application that subscribes to all node topics, aggregates IN/OUT deltas into a shared room occupancy estimate, detects node disconnections via missed heartbeats, and logs all telemetry and occupancy updates to a persistent CSV file. Occupancy is computed as Occ(t+) = Occ(t) + sum(in_delta - out_delta) across both doorways. The gateway exposes a REST API and WebSocket feed via Flask-SocketIO at approximately 1 Hz. This subsystem enables HLR1 through correct delta aggregation and sequence-number reconciliation, HLR2 by providing updated occupancy within 1 second of event receipt, and HLR3 through continuous logging and graceful node dropout handling.

| Requirement | Verification |
|---|---|
| The gateway must correctly compute room occupancy as Occ(t+) = Occ(t) + sum(in_delta - out_delta) across both nodes, with zero accumulation error after any sequence of IN and OUT events. The occupancy value exposed by the REST API must match the expected ground-truth count within +/- 0 (exact integer match) after a 30-event controlled test sequence. | Equipment: Raspberry Pi running gateway application, laptop with curl or Postman for REST API queries, lab notebook for recording expected vs. actual counts. Procedure: (1) Reset gateway occupancy to 0 via the API (POST /reset or restart the gateway). (2) Trigger a pre-defined sequence of exactly 15 IN events and 10 OUT events across both nodes in a known order (record each event direction and which doorway in the lab notebook as ground truth). (3) After all 25 events are processed, query GET /occupancy from the laptop. Record the returned value. (4) Open the CSV log and count the number of logged IN and OUT entries. Results: Record expected occupancy (15-10=5), actual API value, and CSV event counts in the lab notebook. Pass criterion: API returns exactly 5; CSV contains exactly 25 entries with correct IN/OUT classifications. |
| The gateway must log all telemetry events and occupancy updates continuously to a CSV file for at least 60 minutes without file corruption, missing entries, or process crash. The CSV must contain at minimum: timestamp, node_id, event_type (IN/OUT/HEARTBEAT), sequence_number, and occupancy fields. | Equipment: Raspberry Pi, laptop with SSH access, file integrity tools (wc -l, python csv.reader). Procedure: (1) Start the gateway application on the Raspberry Pi. Note the start time. (2) Run both sensor nodes for 60 minutes, triggering at least 1 crossing event per minute (>= 60 events total). (3) After 60 minutes, SSH into the Raspberry Pi and open the CSV log. (4) Run: python3 -c "import csv; rows=list(csv.reader(open('log.csv'))); print(len(rows))" to count rows. (5) Run: python3 -c "import csv; [print(r) for r in csv.reader(open('log.csv')) if len(r)<5]" to check for malformed rows. (6) Confirm the gateway process is still running (ps aux \| grep gateway). |

| | |
|---|---|
| | Results: Record row count, number of malformed rows (if any), and gateway process status in the lab notebook. Pass criterion: row count >= 61 (header + >= 60 events), 0 malformed rows, gateway process still running. |
| The gateway must mark a node as offline in the REST API /status endpoint within 6-8 seconds of the last received heartbeat from that node, and must mark it back as online within 5 seconds of the next received heartbeat after reconnection. | Equipment: Raspberry Pi running gateway, laptop with curl, stopwatch. Procedure: (1) Confirm both nodes show as online via GET /status on the laptop. (2) Power off Node 1 completely. Start stopwatch immediately. (3) Poll GET /status every 1 second from the laptop using a loop: while true; do curl -s localhost:5000/status; sleep 1; done. Record the timestamp at which Node 1 status first changes to offline. (4) Note elapsed time from power-off to offline status. (5) Power Node 1 back on. Start stopwatch. Poll GET /status every 1 second. Record timestamp at which Node 1 status returns to online. Results: Record power-off timestamp, offline-detection timestamp (and elapsed time), power-on timestamp, and online-recovery timestamp (and elapsed time) in the lab notebook. Pass criterion: offline detected within 6-8 seconds of power-off; online detected within 5 seconds of node reconnection. |

#### 2.3.5 Dashboard and User Interface Subsystem

The Dashboard and User Interface Subsystem is a React single-page application served by the Raspberry Pi. It receives real-time occupancy and node status updates via WebSocket (Socket.IO) from the gateway backend and refreshes displayed data at 1 Hz. The dashboard displays current room occupancy as a numeric count, directional flow rate in people per minute over a rolling 5-minute window, a color-coded congestion indicator (green/yellow/red based on

configurable capacity thresholds), and per-node health status (online/offline, last heartbeat time, error flags). All metrics are aggregate and anonymous. This subsystem reinforces HLR1 by surfacing accuracy indicators, demonstrates HLR2 with <= 1 second display latency, and supports HLR3 by explicitly indicating node fault states.

| Requirement | Verification |
|---|---|
| The dashboard occupancy display must update within 1 second of the gateway emitting a WebSocket push following a validated doorway crossing event. This is measured from the WebSocket message arrival time in the browser to the time the React component re-renders with the updated value. | Equipment: Laptop browser (Chrome) with DevTools open (Network tab, WS filter enabled), sensor node and gateway running. Procedure: (1) Open the dashboard on the laptop browser connected to the Raspberry Pi hotspot. Open Chrome DevTools > Network > WS and filter to WebSocket frames. (2) Trigger a single crossing event at a sensor node. (3) In the DevTools WS log, record T1 = timestamp of the incoming WebSocket frame containing the updated occupancy. (4) In the React component, add a console.log with Date.now() at the point where occupancy state updates (or use React DevTools Profiler to record the render timestamp). Record T2. Compute latency = T2 - T1. (5) Repeat 10 times. Results: Record all 10 latency values in a table in the lab notebook. Pass criterion: all 10 values <= 1000 ms. |
| The dashboard node health indicator for each node must transition from Online (green) to Offline (red) within 8 seconds of that node losing power, and must return to Online (green) within 10 seconds of the node reconnecting. | Equipment: Laptop browser with dashboard open, stopwatch, power switch for Node 1. Procedure: (1) Confirm both node health indicators show green on the dashboard. (2) Disconnect Node 1 power. Start stopwatch. (3) Observe the dashboard and record the elapsed time from power disconnection to the Node 1 indicator turning red. (4) Reconnect Node 1 power. Restart stopwatch. Record elapsed time from power reconnection to Node 1 indicator turning green. (5) Repeat steps 2-4 three times. Results: Record all 6 timing measurements (3 offline transitions + 3 online recovery transitions) in a table in the lab notebook. |

| | |
|---|---|
| | Pass criterion: all offline transitions <= 8 seconds; all online recoveries <= 10 seconds. |
| The dashboard crowded-state threshold indicator must change from the uncrowded state (green) to the crowded state (yellow or red) within 2 seconds of occupancy crossing the configured capacity threshold, and must return to green within 2 seconds of occupancy falling below the threshold. | Equipment: Laptop browser with dashboard open, sensor node for triggering events, stopwatch. Procedure: (1) Set the dashboard capacity threshold to 5 via the configuration panel. Confirm occupancy is currently 0 (or reset to 0). (2) Trigger 4 IN events (occupancy = 4). Confirm indicator remains green. (3) Trigger 1 more IN event (occupancy = 5, threshold reached). Start stopwatch. Record elapsed time until indicator changes from green. (4) Trigger 1 OUT event (occupancy = 4, below threshold). Start stopwatch. Record elapsed time until indicator returns to green. (5) Repeat the full sequence (steps 2-4) three times. Results: Record all 6 timing measurements (3 threshold-crossing transitions + 3 below-threshold recoveries) in a table in the lab notebook. Pass criterion: all transitions <= 2 seconds. |

### 2.3.6 Power Subsystem

Each sensor node is powered by a 5V/2A wall adapter connected via a 5.5 mm barrel jack on the custom PCB. An onboard AMS1117-3.3 LDO voltage regulator steps the 5V rail down to 3.3V for the ESP32-WROOM-32 module. The 5V rail also powers the Adafruit 2168 IR emitters and the 5V side of the voltage divider at the receiver outputs. Protection circuitry includes a 500 mA polyfuse on the 5V input rail and a reverse-polarity protection diode. The power subsystem supports HLR1 by preventing brownout-induced ESP32 resets that cause missed or phantom counts, supports HLR2 by ensuring the node remains continuously responsive, and enables HLR3 by sustaining at least 1 hour of continuous operation.

| Requirement | Verification |
|---|---|
| The AMS1117-3.3 LDO output voltage at test point TP_3V3 on the PCB must remain within 3.135V and 3.465V (3.3V +/- 5%) under load conditions ranging from 0 mA to 300 mA, with the 5V input provided by the wall adapter. | Equipment: Bench DC power supply (0-10V, 0-2A), programmable DC electronic load (e.g., BK Precision 8500), DMM (Fluke 87V, resolution <= 0.01V). Procedure: (1) Connect bench supply set to 5.00V to the PCB barrel jack (current limit 1A). (2) Connect the DC electronic load to TP_3V3 and GND on the PCB. (3) Set electronic load to 0 mA (no load). Measure voltage at TP_3V3 with DMM. Record. (4) Set electronic load to 150 mA (typical ESP32 WiFi operating current). Measure and record TP_3V3. (5) Set electronic load to 300 mA (ESP32 peak WiFi transmit current). Measure and record TP_3V3. Results: Record all three voltage measurements in a table in the lab notebook. Pass criterion: all three measurements within 3.135V - 3.465V. |
| The 5V input rail polyfuse must interrupt current flow (trip) when current exceeds 500 mA, protecting the PCB from short-circuit damage. The node must resume normal operation after the polyfuse cools and resets (within 60 seconds of short removal). | Equipment: Bench power supply with current display (0-10V, 0-2A, current limiting enabled), DMM, test wire. Procedure: (1) Connect bench supply set to 5.00V with current limit set to 600 mA to the PCB barrel jack. Note the baseline current reading. (2) With the PCB powered, use a test wire to short the 5V test point (TP_5V) to GND (TP_GND) momentarily (< 1 second). Observe the bench supply current meter: it should spike then drop to near 0 mA as the polyfuse trips. (3) Remove the test wire. Wait 60 seconds for polyfuse to cool and reset. (4) Confirm normal operation resumes by checking that the power LED illuminates and the ESP32 boots (serial monitor output). Results: Record the peak current observed, the current after polyfuse trip, and whether normal operation resumed in the lab notebook. Pass criterion: current drops to <= 50 mA within 1 second of short; normal operation resumes within 60 seconds of short removal. |

| | |
|---|---|
| The node must operate continuously from the 5V wall adapter for at least 60 minutes without any ESP32 firmware reset, brownout event, or MQTT disconnection lasting >= 10 seconds. | Equipment: 5V wall adapter, USB-to-UART adapter and serial terminal (monitoring ESP32 boot messages and MQTT connection state at 115200 baud), Raspberry Pi MQTT broker with heartbeat monitoring. Procedure: (1) Power the node from the wall adapter. Open serial terminal. Confirm the ESP32 boots and connects to MQTT (log the boot timestamp). (2) Run the full firmware for 60 minutes. (3) Monitor the serial terminal for any lines containing 'Brownout', 'rst:', or 'MQTT DISCONNECT'. Log any occurrences with timestamp. (4) On the Raspberry Pi, run mosquitto_sub for the node heartbeat topic and confirm heartbeats arrive continuously throughout the 60 minutes. Results: Record the total run duration, number of ESP32 reset events (if any), number of MQTT disconnect events >= 10 seconds (if any), and a sample of serial terminal output in the lab notebook. Pass criterion: 0 ESP32 resets, 0 brownout events, 0 MQTT disconnections >= 10 seconds over 60 minutes. |

### 2.3.7 Hardware and PCB Subsystem

The Hardware and PCB Subsystem integrates all sensor node electronics onto a custom-designed two-layer PCB fabricated via JLCPCB or OSH Park. The board integrates: the ESP32-WROOM-32 module via through-hole pin headers, the AMS1117-3.3 LDO regulator with 10 uF decoupling capacitors on input and output, a 5.5 mm barrel jack for 5V input, two 3-pin JST-PH headers for IR receiver signal and 5V/GND connections, two 2-pin JST-PH headers for IR emitter 5V/GND connections, a 500 mA polyfuse and reverse-polarity diode on the 5V input, a 3-pin UART debug header (TX, RX, GND) for firmware flashing and serial debugging, and two status LEDs (power-on indicator and MQTT heartbeat blink) with

current-limiting resistors. The board is designed in KiCad. Two identical PCBs are fabricated, one per doorway node.

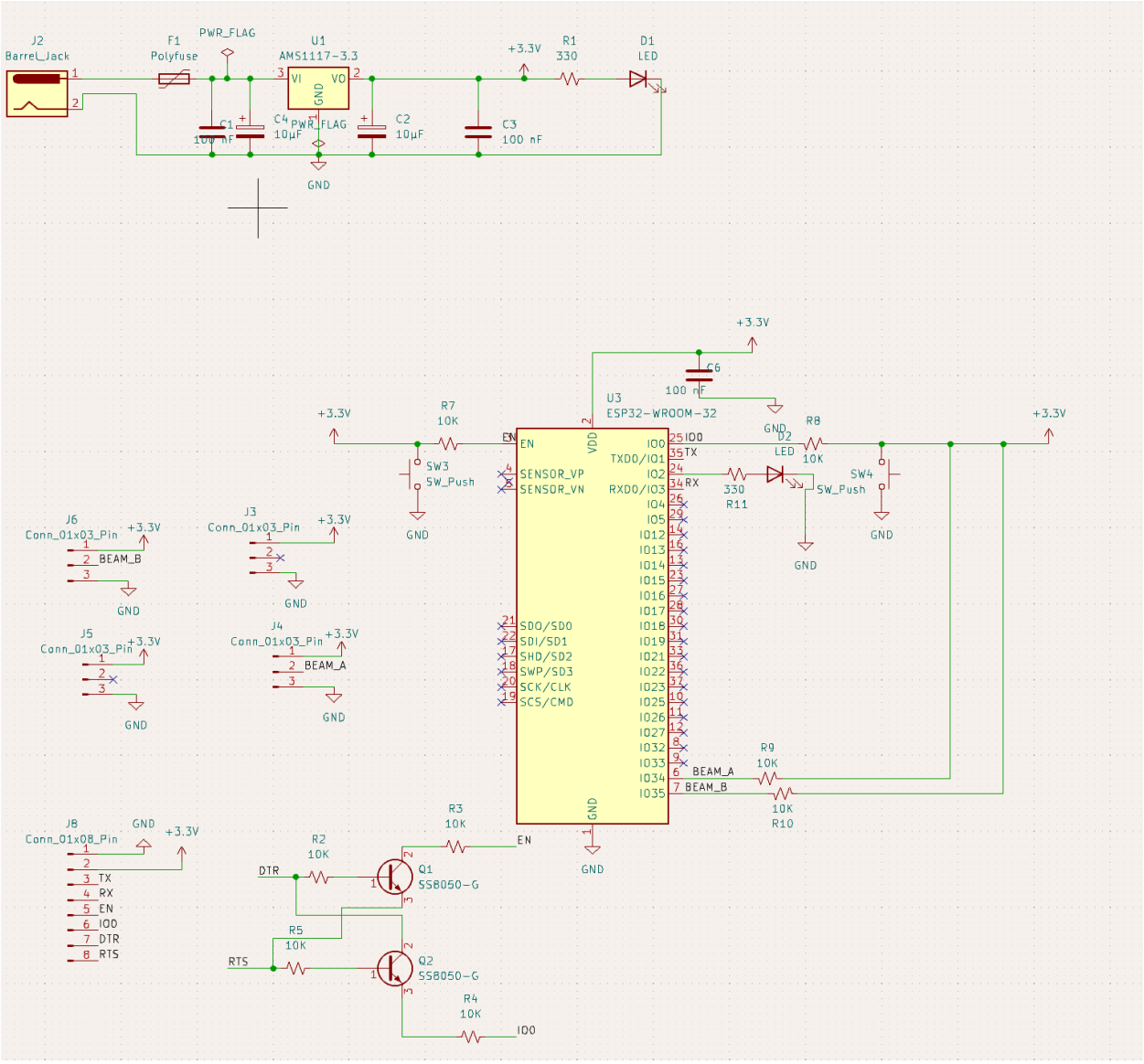


*Figure 5: Sensor node PCB schematic showing ESP32-WROOM-32, AMS1117-3.3 LDO, IR sensor connectors, protection circuitry, debug header, and status LEDs.*

| Requirement | Verification |
|---|---|
| At least one custom-fabricated PCB sensor node must be used in the final demonstration. The PCB must integrate the ESP32-WROOM-32, AMS1117-3.3 LDO regulator, IR sensor JST-PH connectors, 5V barrel jack, polyfuse, and 3-pin UART debug header on a single two-layer board. | Equipment: Fabricated PCB with all components soldered, 5V wall adapter, USB-to-UART adapter, serial terminal. Procedure: (1) Visually inspect the PCB and confirm the presence of all required components (ESP32 module, LDO SOT-223 package, barrel jack, JST-PH headers, polyfuse, debug header, LEDs). Record observations in lab notebook. (2) Connect wall adapter to barrel jack. Confirm power LED illuminates. (3) Connect USB-to-UART adapter to UART debug header (TX-RX, RX-TX, GND-GND). Open serial terminal at 115200 baud. Confirm ESP32 boot log appears. (4) Connect Adafruit 2168 sensors to JST-PH headers. Trigger a crossing event. Confirm event appears in serial terminal output. Results: Record visual inspection checklist, power LED status, boot log output (copy first 5 lines), and crossing event log entry in the lab notebook. Pass criterion: all components present, power LED on, ESP32 boots, crossing event logged. |
| The UART debug header must support firmware flashing to the ESP32 via esptool.py without removing the ESP32 module from the PCB, and must output serial debug messages at 115200 baud readable on a laptop. | Equipment: USB-to-UART adapter (CH340 or CP2102, 3.3V logic levels), laptop with Python 3 and esptool.py installed, USB cable. Procedure: (1) Connect USB-to-UART adapter to the UART header (TX to RX, RX to TX, GND to GND). On the PCB, hold the ESP32 BOOT button (or short the BOOT pin to GND) and apply power to enter bootloader mode. (2) Run: python3 -m esptool --port /dev/ttyUSB0 --baud 115200 flash_id to confirm esptool can communicate with the ESP32. Record the chip description printed by esptool. (3) Flash a test firmware binary: python3 -m esptool --port /dev/ttyUSB0 write_flash 0x0 test_firmware.bin. Confirm flash completes without errors. (4) Release BOOT pin and reset the ESP32. Open serial terminal. Confirm the test firmware prints its startup message. Results: Record esptool chip ID output, |

| | |
|---|---|
| | flash completion status, and serial terminal startup message in the lab notebook. Pass criterion: esptool detects ESP32, flash completes without errors, test firmware boots and outputs expected serial message. |
| The power status LED must illuminate within 1 second of 5V power being applied to the barrel jack and extinguish within 1 second of power removal. The LED operating current must be between 5 mA and 20 mA. | Equipment: 5V bench power supply, DMM (mA measurement mode), stopwatch. Procedure: (1) Connect DMM in series between the power supply positive terminal and the PCB barrel jack positive input (to measure current through the LED circuit). Set DMM to mA range. (2) Apply 5V from bench supply. Start stopwatch. Record elapsed time until LED illuminates. Record DMM current reading while LED is on. (3) Remove power from bench supply. Start stopwatch. Record elapsed time until LED extinguishes. Results: Record power-on delay, LED current reading, and power-off delay in the lab notebook. Pass criterion: LED on within 1 second of power applied; LED current between 5 mA and 20 mA; LED off within 1 second of power removed. |

## 2.4 Hardware Design

### 2.4.1 Voltage Regulation

The sensor node is powered from a 5V/2A wall adapter. The ESP32-WROOM-32 requires a 3.3V supply at up to 500 mA peak (during WiFi transmission bursts). The AMS1117-3.3 is selected as the LDO regulator for this purpose. It is available in SOT-223 package, supports up to 1A output current, and has a dropout voltage of approximately 1.3V at 1A — well within our headroom since the input is a stable 5V rail (headroom = 5V − 3.3V = 1.7V). The LDO requires a 10 µF output capacitor for stability and a 10 µF input capacitor for transient suppression. Power dissipation in the LDO at peak load is P = (Vin − Vout) × Iload = (5 − 3.3) × 0.5 = 0.85W, which

is within the AMS1117 SOT-223 package thermal limit of approximately 1.5W at room temperature. The 5V rail also directly powers the Adafruit 2168 IR emitters (nominally drawing ≈20 mA each, so 40 mA total for both beams) and the pull-up resistors at the receiver outputs. The receiver output of the Adafruit 2168 is an open-collector NPN transistor pulled up to 5V via a 10kΩ resistor on the sensor board. Since the ESP32 GPIO maximum input voltage is 3.6V, a voltage divider (10kΩ / 20kΩ) is placed between the receiver output and the ESP32 GPIO pin to bring the high-state voltage from 5V down to approximately 3.33V, within safe GPIO input range.

### 2.4.2 Protection Circuitry

A 500 mA polyfuse (resettable fuse) is placed in series with the 5V input rail before the barrel jack to protect the PCB from overcurrent in the event of a short circuit. A 1N4007 diode is placed in series with the positive 5V input line (after the barrel jack, before the polyfuse) in the reverse-polarity orientation to protect against accidental reverse connection of the power supply. Under normal operation this diode has a forward voltage drop of approximately 0.7V, reducing the effective 5V supply to approximately 4.3V at the input to the LDO which is still well above the AMS1117-3.3 minimum input voltage of 4.6V. To avoid this marginal headroom, an alternative implementation using a P-channel MOSFET for reverse polarity protection (zero forward-voltage-drop) is noted as a design option during PCB review.

## 2.5 Software Design

### 2.5.1 Direction Inference FSM

The core firmware logic on the ESP32-WROOM-32 implements the direction inference FSM described in Section 2.3.2. GPIO interrupts are configured on both Beam A and Beam B receiver pins to trigger on the falling edge (beam broken). The FSM is implemented in the Arduino framework using interrupt service routines (ISRs) that set volatile state flags and record millisecond timestamps via millis(). The main loop polls these flags and drives FSM transitions. The 500 ms crossing window timeout is enforced using a non-blocking timer check in the main loop. Debouncing is implemented by ignoring interrupts on a pin within 20 ms of the previous interrupt on that same pin (tracked per-pin via a lastInterruptTime array).

### 2.5.2 MQTT Packet Structure and Buffering

Each MQTT event packet published to node/[id]/events is a JSON object containing the following fields: node_id (uint8), seq (uint16, monotonically incrementing), ts_ms (uint32, ESP32 millis() timestamp), in_delta (uint8, 0 or 1), out_delta (uint8, 0 or 1), local_occ (int16, node-local running count), and status (uint8, bit flags for sensor faults and WiFi state). Each MQTT heartbeat packet published to node/[id]/heartbeat contains: node_id, seq, ts_ms, uptime_s (uint32), and status. The local RAM event buffer is a circular array of up to 10 event structs. On WiFi reconnection, the buffer is drained in FIFO order before new events are published, ensuring sequence-number ordering is preserved at the gateway.

### 2.5.3 Gateway Application

The Raspberry Pi gateway runs a Python application using the paho-mqtt library to subscribe to node/+/events and node/+/heartbeat topics. On each event packet receipt, the gateway: validates the sequence number (logs a warning if a gap is detected), applies the in_delta and out_delta to the shared room occupancy counter, appends a row to the CSV log file, and emits a WebSocket push to all connected dashboard clients via Flask-SocketIO. Heartbeat packets update a per-node last_seen timestamp; the gateway marks a node as offline if last_seen exceeds 6 seconds in the past. The REST API exposes GET /occupancy (current count), GET /flow (rolling 5-min people/min), and GET /status (per-node health) endpoints.

## 2.6 Tolerance Analysis

The most critical feature of Crowdsurf is achieving at least 90% correct IN/OUT directional classification accuracy (HLR1). This requirement is not a single-component specification but an emergent property of three physical parameters: the beam spacing d, the walking speed v, and the FSM timeout window T. The tolerance analysis below identifies the accuracy-limiting component, enumerates the tolerances of each parameter, computes worst-case classification boundaries, and confirms that the chosen design values satisfy the 90% accuracy requirement across the full range of expected operating conditions.

The direction inference FSM correctly classifies a crossing as IN when Beam A is triggered before Beam B within the timeout window T, or as OUT when the order is reversed. A misclassification occurs under two conditions: (1) the inter-beam delay Delta_t for a single crossing falls below the debounce threshold t_db = 20 ms, causing one beam trigger to be rejected; or (2) the FSM has not returned to IDLE before the next person begins a crossing, causing two crossings to overlap in the FSM state machine. Both failure modes are governed by the same underlying quantity: the inter-beam delay Delta_t.

For a person of shoulder width w_s crossing at speed v, the inter-beam delay for a single valid crossing is:

$$\Delta t = d / v \quad ...(1)$$

where d is the physical center-to-center beam spacing in meters and v is the pedestrian walking speed in m/s. The three sources of uncertainty affecting Delta_t are described below, with their tolerances.

Beam Spacing Uncertainty (Delta_d). The beams are positioned using mounting brackets with a ruler tolerance of +/- 5 mm (the mounting requirement specified in the Sensing Subsystem R&V table). The nominal beam spacing is d_nom = 150 mm, so the beam spacing d falls in the range:

$$d_{min} = 140 \text{ mm}, \quad d_{nom} = 150 \text{ mm}, \quad d_{max} = 160 \text{ mm} \quad ...(2)$$

Walking Speed Uncertainty (Delta_v). Published studies on indoor pedestrian walking speed report a mean of 1.4 m/s with a standard deviation of approximately 0.2 m/s. The fastest practical indoor walking speed (brisk, no running) is approximately 2.0 m/s; the slowest deliberate walk through a doorway is approximately 0.8 m/s. These are used as worst-case bounds:

$$v_{min} = 0.8 \text{ m/s}, \quad v_{nom} = 1.4 \text{ m/s}, \quad v_{max} = 2.0 \text{ m/s} \quad ...(3)$$

FSM Timeout Window (T). The timeout is a firmware constant set in software. There is no hardware tolerance; it is either set correctly or not. The nominal value is T = 500 ms. This value is chosen to be large enough to capture all valid crossings (which complete in $\Delta t < d_{max} / v_{min} = 200$ ms) and small enough that a person standing in the beam zone does not produce a false classification.

Combining these, the worst-case inter-beam delay across all combinations of beam spacing and walking speed is computed by substituting the extreme values into Equation 1:

$$\Delta t_{min} = d_{min} / v_{max} = 0.140 \text{ m} / 2.0 \text{ m/s} = 70 \text{ ms} \quad ...(4)$$

$$\Delta t_{nom} = d_{nom} / v_{nom} = 0.150 \text{ m} / 1.4 \text{ m/s} = 107 \text{ ms} \quad ...(5)$$

$$\Delta t_{max} = d_{max} / v_{min} = 0.160 \text{ m} / 0.8 \text{ m/s} = 200 \text{ ms} \quad ...(6)$$

Table 2 summarizes the inter-beam delay across the extremal and nominal combinations of beam spacing and walking speed, and evaluates whether each combination satisfies the two design

constraints: (a) Delta_t > t_db = 20 ms (debounce threshold must not filter valid crossings), and (b) Delta_t < T = 500 ms (crossing must complete before timeout).

| Condition | d (mm) | v (m/s) | Delta_t (ms) | > 20 ms? | < 500 ms? |
|---|---|---|---|---|---|
| Worst case (fast walk, min spacing) | 140 | 2.0 | 70 | Yes (70 > 20) | Yes (70 < 500) |
| Nominal (typical walk, nom spacing) | 150 | 1.4 | 107 | Yes (107 > 20) | Yes (107 < 500) |
| Best case (slow walk, max spacing) | 160 | 0.8 | 200 | Yes (200 > 20) | Yes (200 < 500) |

*Table 2: Inter-beam delay Delta_t across worst-case, nominal, and best-case combinations of beam spacing d and walking speed v. Both design constraints are satisfied across the full parameter range.*

It is apparent from Table 2 that the design satisfies both constraints across all combinations of beam spacing tolerance and walking speed range. In the worst case (fastest walker, minimum beam spacing), Delta_t = 70 ms, which is still 3.5x above the 20 ms debounce threshold. No valid crossing will be filtered by debouncing under any expected condition.

The second failure mode — FSM state overlap from close-following pedestrians (tailgating) — is analyzed next. The FSM returns to IDLE after a crossing is classified, which occurs at the later of the two beam triggers. The minimum gap time t_gap between the completion of one person's crossing and the start of the next person's crossing, as a function of inter-person following distance s, is:

$$t_{gap} = s / v \quad ...(7)$$

For the FSM to successfully return to IDLE before the second person arrives, t_gap must exceed Delta_t (the time for the first crossing to complete). Substituting the worst-case Delta_t = 70 ms from Equation 4, the minimum following distance required for reliable classification is:

$$s_{min} = \Delta t_{worst} * v_{max} = 0.070\ s * 2.0\ m/s = 0.14\ m \quad ...(8)$$

A minimum inter-person following distance of 14 cm is therefore required to avoid FSM state overlap. In practice, the physical width of a human torso (approximately 40-50 cm) prevents two people from entering a standard doorway simultaneously at a following distance of less than approximately 30-40 cm. Since 30 cm >> 14 cm, the 90% accuracy requirement is satisfied for all physically realistic sequential pedestrian traffic through a standard single doorway. Table 3 summarizes this result.

| Condition | Delta_t (ms) | s_min (cm) | Conclusion |
|---|---|---|---|
| Worst case (v=2.0 m/s, d=140mm) | 70 ms | 14 cm | Safe: physical min. following distance ~30 cm >> 14 cm |
| Nominal (v=1.4 m/s, d=150mm) | 107 ms | 21 cm | Safe: physical min. following distance ~30 cm >> 21 cm |
| Best case (v=0.8 m/s, d=160mm) | 200 ms | 16 cm | Safe: physical min. following distance ~30 cm >> 16 cm |

*Table 3: Minimum following distance s_min required for reliable FSM classification across worst-case, nominal, and best-case operating conditions. In all cases, s_min is well below the physical minimum inter-person separation through a standard doorway (~30 cm).*

Insights from the Tolerance Analysis. The tolerance analysis revealed that the original design considered a beam spacing of 10 cm, which at the worst-case walking speed of 2.0 m/s would

yield Delta_t_min = 50 ms. While this still exceeds the 20 ms debounce threshold, it provides only a 2.5x margin rather than the 3.5x margin achieved with d = 15 cm. The 15 cm spacing was adopted to provide additional robustness at the expense of a slightly larger mounting footprint. If beam spacing were reduced below approximately 4 cm (Delta_t_min = 20 ms at v = 2.0 m/s), the worst-case walking speed would cause the debounce filter to reject valid beam triggers, directly causing miscounts. The 15 cm nominal spacing with +/- 5 mm tolerance ensures a comfortable 3.5x margin even at maximum walking speed and minimum spacing, confirming the design is robust to both physical mounting variation and the full range of pedestrian behavior. The performance-limiting factor in the current design is not beam spacing or walking speed but rather simultaneous bi-directional traffic (two people crossing in opposite directions at the same time), which the FSM cannot disambiguate and which is outside the scope of the stated HLR1 condition of 'moderate sequential traffic.'

## 2.8 Cost Analysis

The following table itemizes all non-standard parts required to complete two sensor nodes and the gateway system. Labor cost is estimated at $40/hour × 2.5 × estimated hours to complete for each team member, consistent with a typical entry-level ECE graduate salary of $40/hour.

| Description | Manufacturer / Source | Qty | Unit Cost | Extended |
|---|---|---|---|---|
| ESP32-WROOM-32 DevKit (sensor nodes) | Espressif / Amazon | 2 | $8.00 | $16.00 |
| Adafruit 2168 IR Break-Beam Sensor (10cm) | Adafruit | 4 | $4.95 | $19.80 |
| AMS1117-3.3 LDO Regulator SOT-223 | Digikey | 5 | $0.50 | $2.50 |
| 5V/2A Wall Adapter w/ Barrel Jack | Amazon | 2 | $7.00 | $14.00 |
| 500 mA Polyfuse (Resettable Fuse) | Digikey | 5 | $0.30 | $1.50 |
| 1N4007 Diode | Digikey | 10 | $0.05 | $0.50 |
| 10kΩ, 20kΩ Resistors (1/4W, 1%) | Digikey | 20 ea. | $0.02 | $0.80 |
| 10 µF Ceramic Capacitors (0805) | Digikey | 10 | $0.10 | $1.00 |

| | | | | |
|---|---|---|---|---|
| JST-PH 3-pin Headers (IR sensor connectors) | Digikey | 8 | $0.45 | $3.60 |
| 5.5mm Barrel Jack (PCB mount) | Digikey | 2 | $0.60 | $1.20 |
| 3-pin UART Debug Header (0.1" pitch) | Digikey | 4 | $0.20 | $0.80 |
| LED (red/green, 0805) | Digikey | 8 | $0.05 | $0.40 |
| Custom PCB Fabrication (2-layer, JLCPCB) | JLCPCB | 5 | $2.00 | $10.00 |
| Raspberry Pi 4 Model B (2 GB) | Adafruit / Lab | 1 | $45.00 | $45.00 |
| 32 GB MicroSD Card | Amazon | 1 | $8.00 | $8.00 |
| Mounting Brackets / 3D Print Filament | Lab / Amazon | 1 | $5.00 | $5.00 |
| Miscellaneous (solder, wire, headers) | Lab | 1 | $5.00 | $5.00 |

Parts subtotal: $135.10. Shipping (estimated 5%): $6.76. Tax (10%): $13.51. Parts total: $155.37. Labor: Each team member is estimated to work approximately 100 hours over the semester on this project. Using the formula ($40/hr) × 2.5 × 100 hrs = $10,000 per member. Total labor for three members: $30,000.  Grand Total (parts + labor): $30,155.37.

## 2.9 Schedule

The following schedule assigns tasks by week from the current date through the final demo week of April 27. Owners are listed by first name: John (hardware/PCB), Ananya (firmware/embedded), Tanvika (gateway/dashboard).

| Week | Task | Owner |
|---|---|---|
| Feb 24 – Mar 1 | Finalize schematic in KiCad; order all parts from Digikey/Adafruit; set up ESP32 Arduino dev environment and blink test | John, Ananya |
| Feb 24 – Mar 1 | Set up Raspberry Pi with Mosquitto broker and Python MQTT subscriber; confirm basic publish/subscribe | Tanvika |
| Mar 2 – Mar 8 | Complete PCB layout in KiCad; pass DRC; submit PCB order to JLCPCB | John |
| Mar 2 – Mar 8 | Implement Beam A / Beam B GPIO interrupt firmware; implement FSM skeleton (no MQTT yet); test with bench IR sensors | Ananya |
| Mar 2 – Mar 8 | Build gateway Python aggregation logic; implement CSV logging; confirm occupancy counter math | Tanvika |
| Mar 9 – Mar 15 | PCB arrives; solder and bring up board; verify 3.3V LDO output and power LED | John |

| | | |
|---|---|---|
| Mar 9 – Mar 15 | Integrate MQTT publish into ESP32 firmware; implement sequence numbers and heartbeat; test with Pi broker | Ananya |
| Mar 9 – Mar 15 | Build React dashboard skeleton: occupancy display, node health indicator, WebSocket integration | Tanvika |
| Mar 16 – Mar 22 | Mount IR sensors on test doorframe; verify 15 cm beam spacing; test voltage divider on PCB | John |
| Mar 16 – Mar 22 | Implement local RAM event buffer; test WiFi outage buffering and reconnect flush | Ananya |
| Mar 16 – Mar 22 | Add flow rate computation (rolling 5-min window) and congestion threshold indicator to dashboard | Tanvika |
| Mar 23 – Mar 29 | Full node integration test: PCB + sensors + ESP32 firmware → MQTT → gateway → dashboard end-to-end | Everyone |
| Mar 30 – Apr 5 | Run 60-minute continuous operation test; measure end-to-end latency; run 90% accuracy counting test | Everyone |
| Apr 6 – Apr 12 | Fabricate second PCB node (or bring up spare); mount both nodes on two doorways of test room | John |
| Apr 6 – Apr 12 | Bug fixes from integration testing; firmware tuning (debounce threshold, FSM timeout) | Ananya |

| | | |
|---|---|---|
| Apr 6 – Apr 12 | Dashboard polish: multi-zone view, historical trend display, alert notifications | Tanvika |
| Apr 13 – Apr 19 | Full system test with both nodes and two-doorway room; verify all HLR requirements met | Everyone |
| Apr 20 – Apr 26 | Final debugging and polish; prepare demo script and documentation; mock demo run-through | Everyone |
| Apr 27 | Final Demo | Everyone |

# 3 Discussion of Societal Impact, Engineering Standards, Ethics, and Safety Considerations

## 3.1 Societal Impact

Crowdsurf addresses a concrete and widespread problem: the inefficient and sometimes unsafe use of shared indoor spaces in academic and public settings. By providing real-time, room-level occupancy data, the system enables students to make informed decisions about where to study or exercise, reducing unnecessary travel to already-overcrowded spaces. In safety-critical situations such as building evacuations or public health capacity limits, real-time headcounts can support faster and more accurate decision-making by facility staff. From an economic perspective, the system's low hardware cost (approximately $155 in parts) makes it accessible to resource-constrained institutions such as community colleges or public libraries that cannot afford camera-based commercial alternatives. Environmentally, the system uses low-power embedded electronics that draw under 2W per node, minimizing energy consumption during

continuous operation. Socially and culturally, the privacy-preserving design (no cameras, no biometric data, no individual tracking) is particularly significant: it allows deployment in spaces where surveillance would be ethically unacceptable, such as gender-neutral restrooms, mental health counseling centers, or places of worship, expanding the range of environments that can benefit from occupancy monitoring. The approach is globally applicable and does not depend on any specific cultural context or personal identifier, making it scalable across diverse international settings

## 3.2 Engineering Standards

The following engineering standards are relevant to the Crowdsurf project. IEEE 802.11 (Wi-Fi) governs the wireless communication protocol used between the ESP32 nodes and the Raspberry Pi gateway. MQTT protocol version 3.1.1 (standardized under OASIS) defines the publish-subscribe messaging format used for telemetry transmission. IPC-2221 (Generic Standard on Printed Board Design) governs PCB layout and design rules applied in KiCad during PCB fabrication. UL 60950-1 (Safety of Information Technology Equipment) is relevant to the low-voltage wall adapter power supplies used to power the sensor nodes. IEEE 1584 and NFPA 70 (National Electrical Code) provide general guidance on safe handling of low-voltage electronics and wiring in building environments.

## 3.3 IEEE and ACM Code of Ethics

This project aligns with several principles of the IEEE Code of Ethics. Under Section I.1, the system prioritizes public safety and welfare by providing occupancy data that can improve safety during crowding or emergencies, while the privacy-preserving design protects user welfare by

eliminating surveillance risk. Under Section I.5, the team commits to honest and realistic claims: the 90% accuracy threshold is stated with specific test conditions (moderate sequential traffic) and the system explicitly acknowledges edge cases (tailgating, simultaneous entry) where accuracy may degrade, rather than claiming universal reliability. Under Section I.2, by open-sourcing the hardware design files and firmware, the team aims to improve community understanding of privacy-preserving sensing approaches. The ACM Code of Ethics Principle 1.6 (Respect Privacy) is directly embodied in the system design: only aggregate, anonymous counts are collected, with no mechanism to reconstruct individual identities or movement patterns from any logged data.

## 3.4 Electrical and Mechanical Safety

The Crowdsurf system operates entirely at low voltages (5V and 3.3V DC) and presents minimal electrical hazard to users or developers. The 5V wall adapters are commercially certified power supplies with UL or CE markings, and the PCB includes polyfuse overcurrent protection and reverse-polarity protection to prevent damage during assembly or accidental miswiring. Developers handling PCBs should follow standard ESD precautions (anti-static wrist straps, ESD-safe workbenches) when handling the ESP32 module and PCB assemblies. No soldering should be performed on powered boards. The IR emitters operate at standard 940 nm wavelength and low power levels consistent with the Adafruit 2168 specifications; they do not present any eye hazard under normal operation.

Mechanically, the sensor mounting brackets must be secured firmly to doorframes to prevent them from falling and creating a trip hazard or blocking emergency egress. Mounting hardware must not damage the doorframe structure. All wiring between the sensor brackets and PCB

enclosure must be routed and secured to avoid creating a tripping hazard in the doorway. The system must not restrict or delay emergency egress in any way; it is a passive monitoring system only and has no ability to lock or block doorways. These requirements were present in the original proposal and remain unchanged.

## 3.5 Safety Mitigation Procedures

The following procedures will be followed to mitigate safety concerns during development and deployment. During PCB assembly and bring-up, all soldering will be performed at the designated ECE 445 lab bench with a fume extractor running. The board will be powered for the first time through a bench power supply with current limiting set to 200 mA to prevent damage from assembly errors before connecting the wall adapter. During sensor mounting at a test doorway, the team will obtain permission from the relevant facility manager before installing any hardware and will ensure all cables are taped down or routed through cable channels to eliminate trip hazards. The system will not be deployed in any space without advance approval from facility management. All wall adapters used will be commercially certified products (UL or CE listed) and will not be modified. No lithium batteries are used in this project, so no battery safety documentation is required.