Internet of Things: Smart Home Security and Automation System with Cisco Packet Tracer.

Project Overview

This project involves designing and simulating a smart home system that integrates IoT devices to enhance security and automate daily tasks. Using Cisco Packet Tracer, you can create a network where sensors, actuators, and controllers communicate to monitor the home environment and respond to events in real time. The system will focus on security features like intrusion detection and automation features like lighting and temperature control, all manageable remotely via a simulated smartphone or server interface.

Project Details

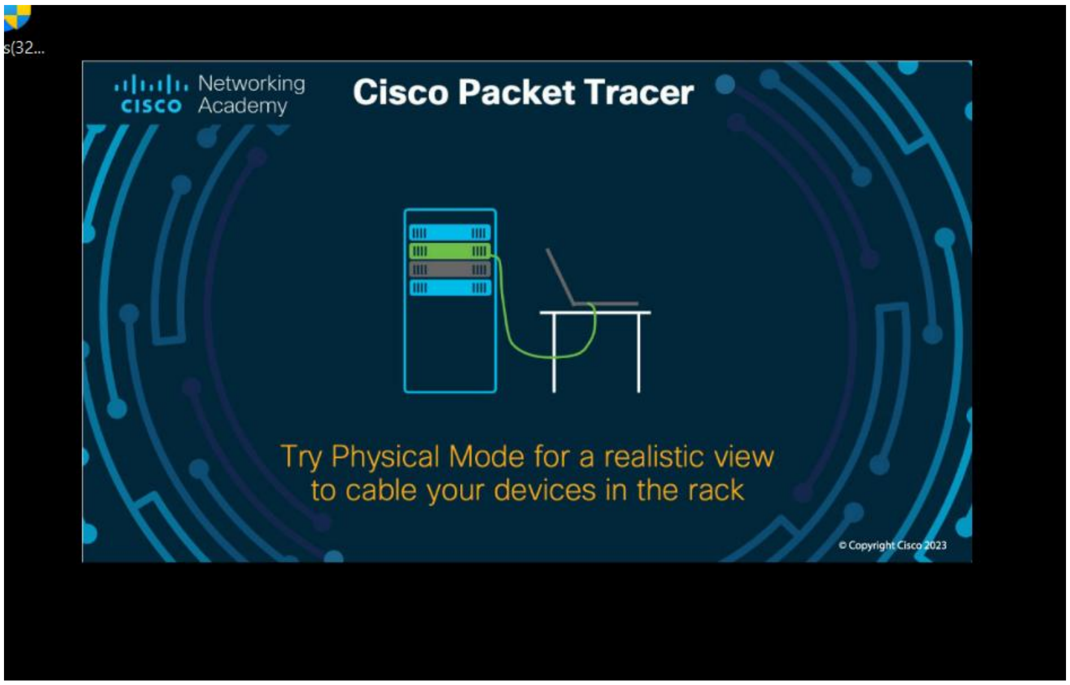
1. **Objective**: Build a simulated IoT-based smart home system that ensures security (e.g., detecting unauthorized entry) and automates home appliances (e.g., lights, doors, fan, thermostat) using Cisco Packet Tracer's IoT capabilities.
2. **Components in Cisco Packet Tracer**:
   a. **IoT Devices**: Motion sensors (for detecting movement), door/window sensors (for entry detection), smoke detectors (for fire safety), smart lights, and a thermostat.
   b. **Boards**: Microcontroller Unit (MCU-PT) or Single Board Computer (SBC-PT) to process sensor data and control actuators.
   c. **Network Devices**: Home Gateway for IoT device registration and connectivity, a wireless router for network access, and a server for remote monitoring.
   d. **End User Device**: A smartphone or PC to simulate remote control and alerts.
3. **Functionality**:
   a. **Security**: The motion sensor triggers an alarm and sends an alert to the user's device if movement is detected when the system is armed. Door/window sensors notify the user of unauthorized entry.
   b. **Automation**: Smart lights turn on/off based on motion detection or a schedule. The thermostat adjusts temperature based on predefined conditions (e.g., lowering at night).
   c. **Remote Access**: The Home Gateway connects all devices to a server, allowing the user to monitor and control the system via a web interface or simulated app.
4. **Implementation Steps**:

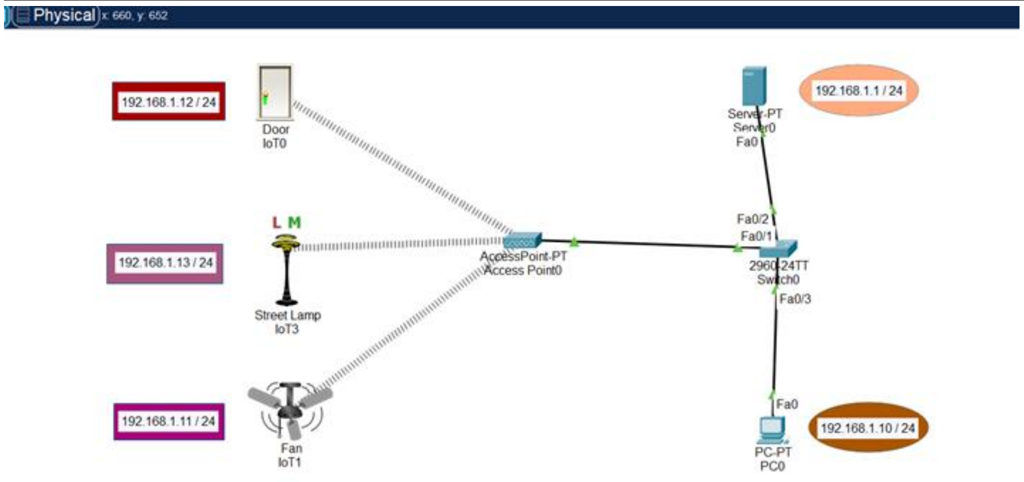
   a. Set up the network topology in Cisco Packet Tracer with a Home Gateway, wireless router, and IoT devices.

   b. Configure the MCU-PT or SBC-PT to process inputs from sensors (e.g., motion, door state) and control outputs (e.g., lights, alarm).
   c. Use Packet Tracer's programming environment (e.g., Python or JavaScript) to define logic, such as "if motion detected and system armed, trigger alarm."
   d. Connect the system to a server or Home Gateway for IoT registration and remote access.
   e. Simulate real-world scenarios (e.g., opening a door, detecting smoke) and test the system's response.
5. **Cybersecurity Angle**: Incorporate basic security measures like encrypted communication between devices and the server to prevent unauthorized access, showcasing how IoT security can be simulated in Packet Tracer.

Why Cisco Packet Tracer?

Cisco Packet Tracer supports IoT simulation with a variety of smart devices, sensors, and programming capabilities, making it ideal for this project. It allows you to visualize data flow, test configurations, and simulate real-time interactions without physical hardware

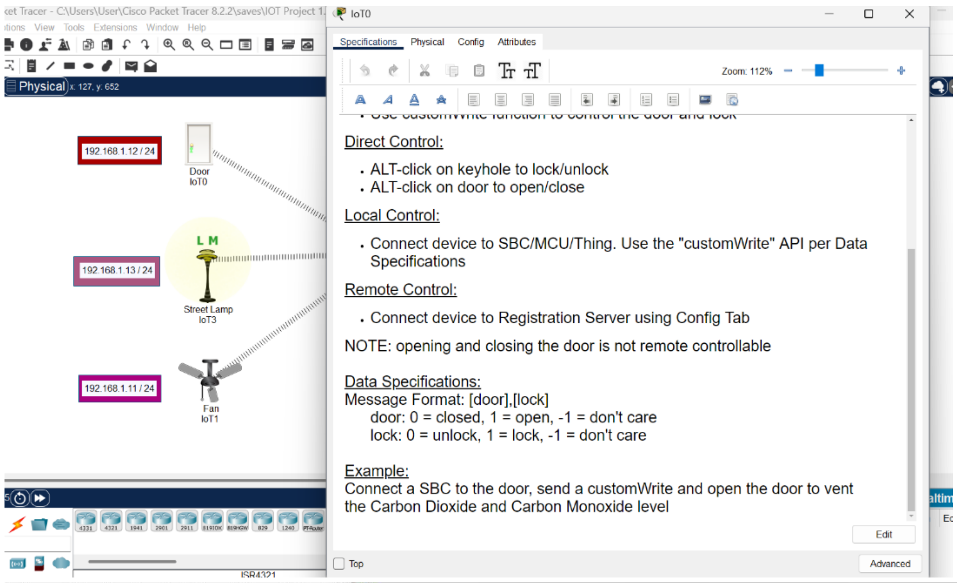

s(32...
Networking Academy
Cisco Packet Tracer
Try Physical Mode for a realistic view to cable your devices in the rack
© Copyright Cisco 2023



IoT for Doors

- **Smart Access and Security**

- **Remote Locking/Unlocking:**
  - Control door locks from anywhere via a smartphone app.
  - Generate temporary access codes for guests or service providers.
  - Integration with voice assistants (Google Assistant, Alexa).
- **Biometric Authentication:**
  - Fingerprint, facial recognition, or iris scanning for keyless entry.
  - Integration with user databases for managing authorized personnel.
- **Activity Monitoring and Alerts:**
  - Sensors (magnetic contact, PIR) to detect door open/close status and motion near the door.
  - Receive real-time notifications on your phone for forced entry attempts, door left ajar, or suspicious activity.
  - Integration with cameras to capture images or video of visitors/intruders.
- **Automatic Locking:**
  - Geofencing: Door locks automatically when you leave a defined area around your home.
  - Timer-based locking: Door locks after a set period if left unlocked.
  - Proximity-based locking: Locks when authorized devices are out of range.
- **Emergency Access:**
  - Integration with smart smoke detectors or fire alarms to automatically unlock doors in case of an emergency for quick evacuation.Remote unlocking for emergency services.



• Use customWrite function to control the door and lock

Direct Control:

- ALT-click on keyhole to lock/unlock
- ALT-click on door to open/close

Local Control:

- Connect device to SBC/MCU/Thing. Use the "customWrite" API per Data Specifications

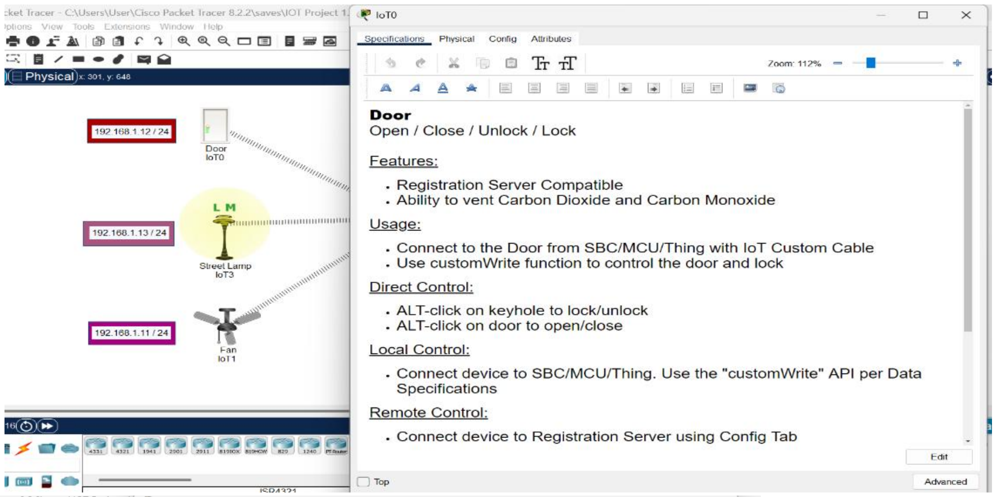
Remote Control:

- Connect device to Registration Server using Config Tab

NOTE: opening and closing the door is not remote controllable

Data Specifications:
Message Format: [door],[lock]
    door: 0 = closed, 1 = open, -1 = don't care
    lock: 0 = unlock, 1 = lock, -1 = don't care

Example:
Connect a SBC to the door, send a customWrite and open the door to vent the Carbon Dioxide and Carbon Monoxide level



Door
Open / Close / Unlock / Lock

Features:

- Registration Server Compatible
- Ability to vent Carbon Dioxide and Carbon Monoxide

Usage:

- Connect to the Door from SBC/MCU/Thing with IoT Custom Cable
- Use customWrite function to control the door and lock

Direct Control:

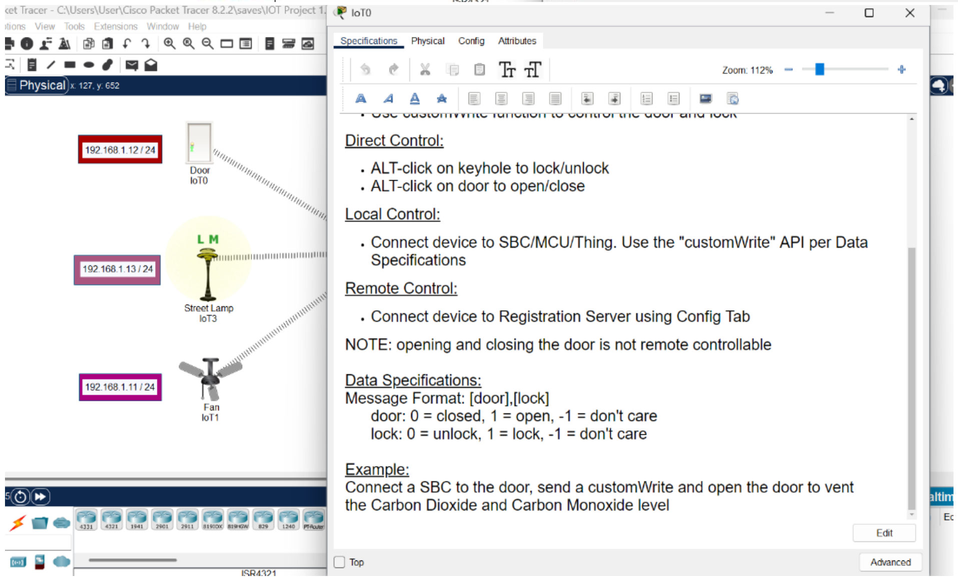
- ALT-click on keyhole to lock/unlock
- ALT-click on door to open/close

Local Control:

- Connect device to SBC/MCU/Thing. Use the "customWrite" API per Data Specifications

Remote Control:

- Connect device to Registration Server using Config Tab



• Use customWrite function to control the door and lock

Direct Control:

- ALT-click on keyhole to lock/unlock
- ALT-click on door to open/close

Local Control:

- Connect device to SBC/MCU/Thing. Use the "customWrite" API per Data Specifications

Remote Control:

- Connect device to Registration Server using Config Tab

NOTE: opening and closing the door is not remote controllable

Data Specifications:
Message Format: [door],[lock]
    door: 0 = closed, 1 = open, -1 = don't care
    lock: 0 = unlock, 1 = lock, -1 = don't care

Example:
Connect a SBC to the door, send a customWrite and open the door to vent the Carbon Dioxide and Carbon Monoxide level

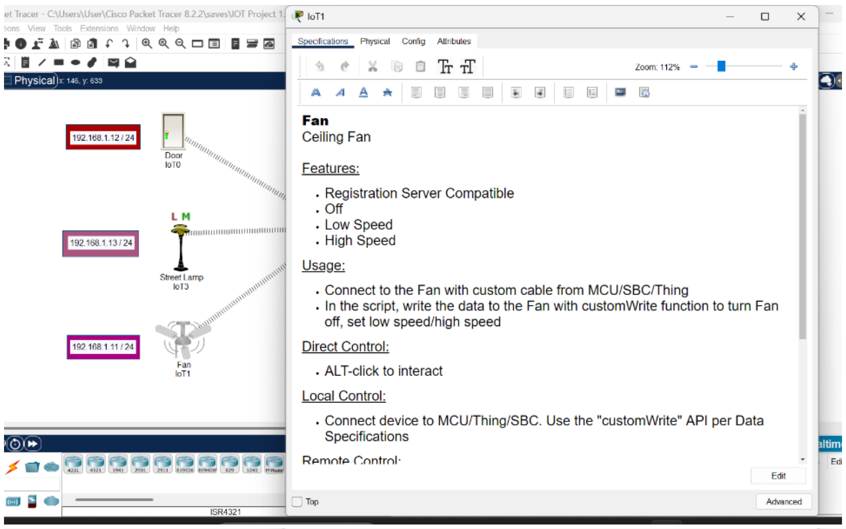
IoT for Fans

1. Smart Climate Control and Air Quality

- **Temperature and Humidity-Controlled Operation:**
    - Fans automatically adjust speed based on room temperature and humidity detected by sensors.
    - Integration with smart thermostats for a holistic climate control system.
- **Occupancy-Based Operation:**
    - PIR or ultrasonic sensors detect human presence. Fans turn on when a room is occupied and off when empty to save energy.
    - Adjust fan speed based on the number of occupants.
- **Air Quality Monitoring:**
    - Integration with air quality sensors (PM2.5, VOCs, CO2). Fans can automatically turn on or increase speed to improve air circulation when pollutants are detected.
    - Trigger alerts for poor air quality.
- **Predictive Cooling/Ventilation:**
    - Learn user preferences and typical routines.
    - Integrate with weather forecasts to pre-cool or ventilate a room before it gets too hot.

2. Energy Efficiency and Optimization

- **Dynamic Speed Control:**
    - Fine-grained control over fan speed based on real-time environmental data and user preferences, rather than just fixed settings.
- **Scheduling and Timers:**
    - Program fans to turn on/off or adjust speed at specific times of day.
    - "Sleep mode" that gradually reduces fan speed overnight.



Fan
Ceiling Fan

Features:

- Registration Server Compatible
- Off
- Low Speed
- High Speed

Usage:

- Connect to the Fan with custom cable from MCU/SBC/Thing
- In the script, write the data to the Fan with customWrite function to turn Fan off, set low speed/high speed

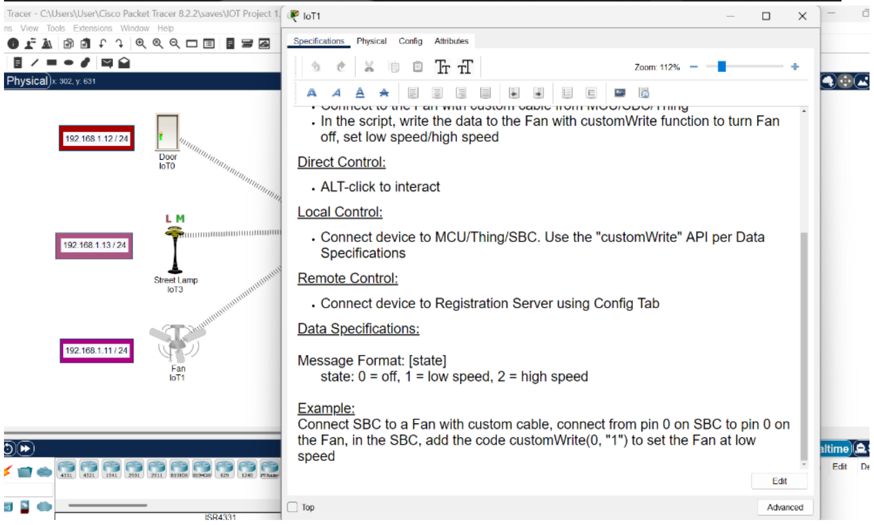
Direct Control:

- ALT-click to interact

Local Control:

- Connect device to MCU/Thing/SBC. Use the "customWrite" API per Data Specifications

Remote Control:



- In the script, write the data to the Fan with customWrite function to turn Fan off, set low speed/high speed

Direct Control:

- ALT-click to interact

Local Control:

- Connect device to MCU/Thing/SBC. Use the "customWrite" API per Data Specifications

Remote Control:

- Connect device to Registration Server using Config Tab

Data Specifications:

Message Format: [state]
state: 0 = off, 1 = low speed, 2 = high speed

Example:
Connect SBC to a Fan with custom cable, connect from pin 0 on SBC to pin 0 on the Fan, in the SBC, add the code customWrite(0, "1") to set the Fan at low speed

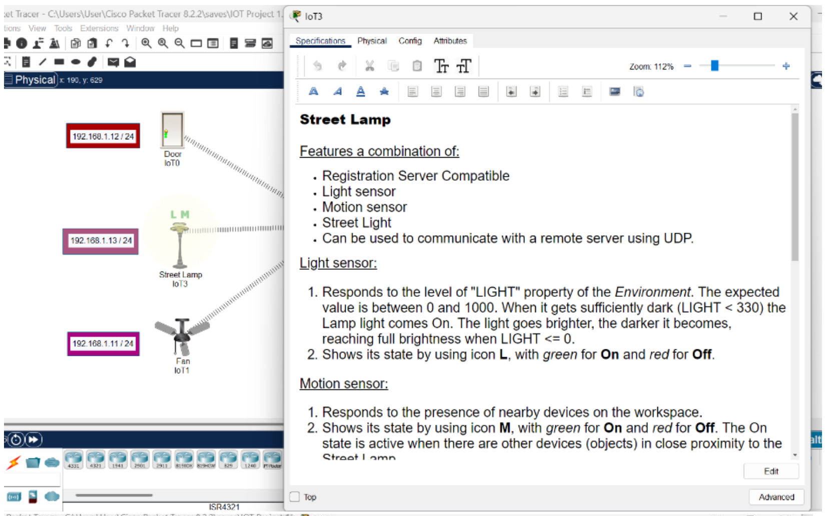
IoT for Light Bulbs (Smart Lighting)

1. Advanced Control and Automation

- Presence-Aware Lighting (Beyond Simple Motion):

- **Sensors:** PIR for basic motion, **mmWave radar sensors** for more precise human presence detection (can detect stillness, unlike PIR).
- **Logic:** Lights not only turn on/off with motion but also adjust brightness or color temperature based on continuous presence. If someone is sitting still reading, the light remains on. If the room is empty for 10 minutes, it turns off or dims significantly.

    - **Networking:** Constant, but small, data packets for presence updates.
    - **Innovation:** Solves the common problem of lights turning off when you're still in the room.
- **Contextual Scene Automation:**
    - **Sensors/Inputs:** Integrate with other smart home data – alarm clock, calendar, weather forecast, smart speaker activity.
    - **Logic:**
        - **Wake-up Light:** Gradually brightens and shifts color temperature to mimic sunrise before your alarm.
        - **Movie Night:** Dims lights, sets a warm color, and turns on accent lighting when your smart TV starts streaming.
        - **Weather-Adaptive:** If it's a cloudy day, lights might be brighter and cooler white; if sunny, warmer and dimmer.
        - **Dinner Time:** Sets a specific scene based on a calendar event or verbal command to a smart speaker.
    - **Networking:** API calls to external services (weather), integration with smart home hubs, local triggers from other devices.
    - **Innovation:** Truly intelligent lighting that adapts to daily routines and external factors.
- **Dynamic Light Art & Ambiance:**
    - **Hardware:** Addressable RGBW LEDs (like WS2812B strips or individual smart bulbs with multiple LED types).
    - **Logic:** Create complex, dynamic light patterns or "light shows" that can be triggered by music, external events (e.g., favorite sports team scores, stock market changes), or pre-programmed sequences.
    - **Networking:** High-bandwidth communication for rapid updates to individual LEDs (e.g., UDP packets), potentially local processing on the bulb controller.
    - **Innovation:** Transforms lighting from functional to an artistic or expressive medium.



Street Lamp

Features a combination of:

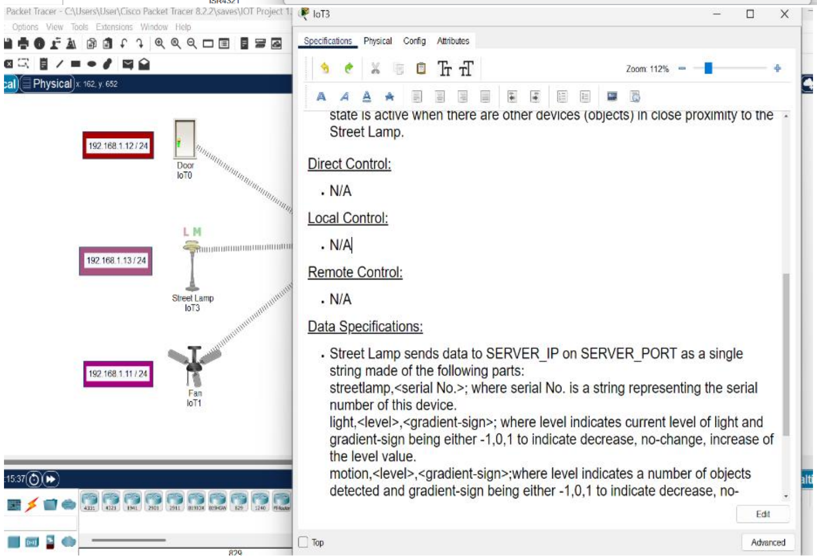
- Registration Server Compatible
- Light sensor
- Motion sensor
- Street Light
- Can be used to communicate with a remote server using UDP.

Light sensor:

1. Responds to the level of "LIGHT" property of the *Environment*. The expected value is between 0 and 1000. When it gets sufficiently dark (LIGHT < 330) the Lamp light comes On. The light goes brighter, the darker it becomes, reaching full brightness when LIGHT <= 0.
2. Shows its state by using icon **L**, with *green* for **On** and *red* for **Off**.

Motion sensor:

1. Responds to the presence of nearby devices on the workspace.
2. Shows its state by using icon **M**, with *green* for **On** and *red* for **Off**. The On state is active when there are other devices (objects) in close proximity to the Street Lamp



state is active when there are other devices (objects) in close proximity to the Street Lamp.

Direct Control:

- N/A

Local Control:

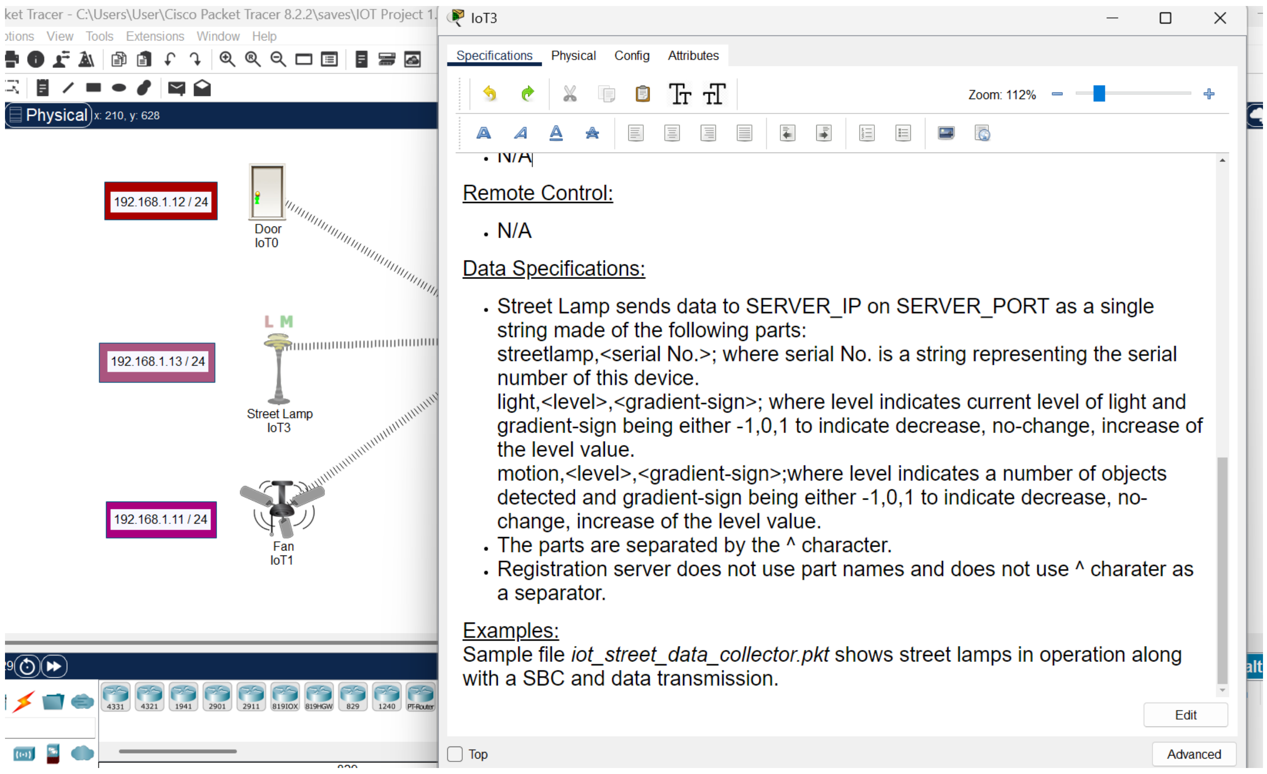
- N/A

Remote Control:

- N/A

Data Specifications:

- Street Lamp sends data to SERVER_IP on SERVER_PORT as a single string made of the following parts:
  streetlamp,<serial No.>; where serial No. is a string representing the serial number of this device.
  light,<level>,<gradient-sign>; where level indicates current level of light and gradient-sign being either -1,0,1 to indicate decrease, no-change, increase of the level value.
  motion,<level>,<gradient-sign>;where level indicates a number of objects detected and gradient-sign being either -1,0,1 to indicate decrease, no-



• N/A

Remote Control:

• N/A

Data Specifications:

- Street Lamp sends data to SERVER_IP on SERVER_PORT as a single string made of the following parts:
  streetlamp,<serial No.>; where serial No. is a string representing the serial number of this device.
  light,<level>,<gradient-sign>; where level indicates current level of light and gradient-sign being either -1,0,1 to indicate decrease, no-change, increase of the level value.
  motion,<level>,<gradient-sign>;where level indicates a number of objects detected and gradient-sign being either -1,0,1 to indicate decrease, no-change, increase of the level value.
- The parts are separated by the ^ character.
- Registration server does not use part names and does not use ^ charater as a separator.

Examples:
Sample file *iot_street_data_collector.pkt* shows street lamps in operation along with a SBC and data transmission.

My networking project successfully demonstrated how **Dynamic Host Configuration Protocol (DHCP)** served as the practical and efficient solution for managing the IP addresses of my numerous IoT endpoints, including smart doors, fans, and light bulbs. This design choice was fundamental to the project's scalability and ease of operation.

Why DHCP Was Essential for My IoT Devices

- For these widespread IoT devices, **DHCP proved invaluable**, offering significant advantages over individually assigned IP addresses:
- **Effortless Scalability and Management:** Imagine the sheer impracticality of manually assigning and tracking IP addresses for hundreds or even thousands of smart devices across a large deployment. DHCP automated this entire process, drastically cutting down administrative work and allowing the network to effortlessly accommodate new devices.
- **No IP Address Conflicts:** In a network with many devices constantly connecting and disconnecting, manually set IPs would have been a constant source of frustrating conflicts. DHCP's centralized management inherently prevented these clashes, ensuring smooth and reliable communication for all IoT devices.

- **Seamless Mobility and Replacement:** The ability to easily move a smart fan to a different room or replace a light bulb without any manual network reconfiguration was a huge benefit. DHCP ensured devices seamlessly acquired new IP addresses, eliminating the need for constant updates to network settings.
- **Resource Efficiency:** For IoT endpoints primarily tasked with reporting sensor data or receiving simple commands, DHCP's dynamic allocation proved highly resource-efficient. It avoided tying up fixed, potentially underutilized, addresses for devices that don't require constant, predictable inbound connections.

The Role of Consistent IP Assignments in My Network Architecture

- While my IoT endpoints utilized DHCP, **critical network infrastructure** like my routers, servers, and firewalls required consistent, predictable access. This was achieved through their assigned IP addresses, which remained stable for reliable network management, service delivery, and stringent security configurations. To ensure these critical devices always maintained their designated addresses and were not accidentally assigned an IP by DHCP, I specifically configured an exclusion range on my DHCP server. For instance, a key part of my configuration included:

```
ip dhcp excluded-address 192.168.1.10 192.168.1.20
```

- This command ensures the DHCP server will **never** hand out any IP addresses within the 192.168.1.10 to 192.168.1.20 range. This reserved block was then used for those essential, stable network components.

Consistent IoT Communication Despite Dynamic Ips

- **Crucially, even though my IoT devices used dynamic IP addresses, their interaction within the larger network remained entirely consistent. This was achieved through the clever use of other identifiers like unique device IDs, MAC addresses, or application-layer identifiers used by the specific IoT platforms (such as Blynk for my smart fan project). These consistent identifiers allowed the system to reliably recognize and communicate with each "door," "fan," or "light bulb," regardless of its current dynamic IP address.**