实验报告

项目名称

一个典型物联网系统中传输机制的设计与实现

项目目标

设计并实现一个基于UDP协议的物联网系统，模拟虚拟路灯的状态传输和控制。系统包括感控层（虚拟路灯），网络层（UDP通信）和应用层（服务器和客户端界面）。

设计与实现

1. udp_protocol.py

该文件定义了UDP通信协议的相关内容，包括数据包类型、数据包的创建和解析函数，以及传感器数据的生成函数。

- **Packet类**：用于表示不同类型的数据包。
- **create_status_packet**：创建状态更新数据包，包含设备ID、温度、湿度、光照和灯的开关状态。
- **create_control_packet**：创建控制命令数据包，包含设备ID和控制命令（开/关）。
- **create_ack_packet**：创建确认数据包，包含设备ID。
- **parse_status_packet**：解析状态更新数据包，提取温度、湿度、光照和灯的开关状态。
- **parse_control_packet**：解析控制命令数据包，提取控制命令。
- **generate_sensor_data**：生成模拟的传感器数据（温度、湿度、光照）。

2. database.py

该文件实现了一个简单的数据库类，用于存储和查询虚拟路灯的历史状态数据。

- **Database类**：管理数据库连接和操作。
- **create_table**：创建存储状态数据的表。
- **insert_status**：插入状态数据，包括设备ID、温度、湿度、光照、灯的开关状态和时间戳。
- **query_history**：查询历史状态数据，根据设备ID和时间范围进行查询。
- **close**：关闭数据库连接。

3. server.py

该文件实现了服务器端应用程序，负责接收虚拟路灯的状态数据，并发送控制命令。

- **Server类**：管理服务器的启动、停止和数据接收。

- **setup_ui**：设置服务器的图形用户界面，包括连接的终端列表、上传数据包列表和下发数据包列表。
- **start_server**：启动服务器并开始监听UDP端口。
- **stop_server**：停止服务器。
- **receive_data**：接收来自虚拟路灯的状态数据。
- **handle_status_update**：处理状态更新数据包，更新客户端列表并存储状态数据。
- **handle_ack**：处理确认数据包，更新下发数据包列表。
- **update_client_list**：更新客户端列表显示，包括设备ID、地址、温度、湿度、光照、状态和最后更新时间。
- **turn_on_selected**：发送开灯命令。
- **turn_off_selected**：发送关灯命令。
- **send_control_command**：发送控制命令。
- **check_client_timeout**：检查客户端超时，移除断开连接的客户端。
- **remove_disconnected_client**：移除断开连接的客户端，并更新上传数据包列表。

4. streetlight_client.py

该文件实现了虚拟路灯客户端应用程序，模拟虚拟路灯的状态，并通过UDP协议发送给服务器。

- **StreetlightClient类**：管理客户端的启动、停止和数据发送。
- **setup_ui**：设置客户端的图形用户界面，包括显示温度、湿度、光照和灯的开关状态。
- **set_device_id**：设置虚拟路灯的设备ID。
- **start**：启动客户端并开始发送状态数据。
- **stop**：停止客户端。
- **update_status**：生成并发送状态数据，包括温度、湿度、光照和灯的开关状态。
- **receive_commands**：接收并处理来自服务器的控制命令，更新灯的开关状态并发送确认数据包。
- **update_status_display**：更新状态显示，包括灯的开关状态。

测试结果

通过运行服务器和多个虚拟路灯客户端，成功实现了客户端定期生成并发送状态数据，服务器接收并显示这些数据，并能够发送控制命令来控制虚拟路灯的开关状态。测试结果表明，系统能够稳定运行，并且能够正确处理多个客户端的连接和通信。

总结与展望

本项目实现了一个基于UDP协议的物联网系统，模拟了虚拟路灯的状态传输和控制。通过在同一台机器上使用不同的网络接口，可以模拟多个设备。未来可以进一步扩展系统功能，例如增加更多类型的传感器数据，优化通信协议的可靠性，以及实现更复杂的控制逻辑。此外，可以考虑引入更多的安全机制，确保数据传输的安全性和完整性。

源代码

源代码包括以下文件：

- udp_protocol.py

```python
# udp_protocol.py

import struct
import random

# Packet types
STATUS_UPDATE = 1
CONTROL_COMMAND = 2
ACK = 3

class Packet:
    def __init__(self, packet_type, device_id, data):
        self.packet_type = packet_type
        self.device_id = device_id
        self.data = data

    def to_bytes(self):
        header = struct.pack('!BI', self.packet_type, self.device_id)
        return header + self.data

    @classmethod
    def from_bytes(cls, packet_bytes):
        packet_type, device_id = struct.unpack('!BI', packet_bytes[:5])
        data = packet_bytes[5:]
        return cls(packet_type, device_id, data)

def create_status_packet(device_id, temperature, humidity, light, is_on):
    data = struct.pack('!fffB', temperature, humidity, light, is_on)
    return Packet(STATUS_UPDATE, device_id, data)

def create_control_packet(device_id, command):
    data = struct.pack('!B', command)
    return Packet(CONTROL_COMMAND, device_id, data)

def create_ack_packet(device_id):
    return Packet(ACK, device_id, b'')

def parse_status_packet(packet):
    temperature, humidity, light, is_on = struct.unpack('!fffB', packet.data)
    return temperature, humidity, light, bool(is_on)

def parse_control_packet(packet):
    return struct.unpack('!B', packet.data)[0]
```

```
def generate_sensor_data():
    return random.uniform(0, 40), random.uniform(0, 100), random.uniform(0, 1000)
```

- database.py

```python
# database.py

import sqlite3
from datetime import datetime
import queue
import threading

class Database:
    def __init__(self):
        self.conn = sqlite3.connect('streetlights.db', check_same_thread=False)
        self.create_table()
        self.queue = queue.Queue()
        self.worker_thread = threading.Thread(target=self._worker, daemon=True)
        self.worker_thread.start()

    def create_table(self):
        cursor = self.conn.cursor()
        cursor.execute('''
        CREATE TABLE IF NOT EXISTS status_history (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            device_id INTEGER,
            temperature REAL,
            humidity REAL,
            light REAL,
            is_on BOOLEAN,
            timestamp DATETIME
        )
        ''')
        self.conn.commit()

    def _worker(self):
        while True:
            function, args, kwargs, result_queue = self.queue.get()
            try:
                result = function(*args, **kwargs)
                if result_queue:
                    result_queue.put(result)
            except Exception as e:
                if result_queue:
                    result_queue.put(e)
            finally:
                self.queue.task_done()

    def _execute(self, function, *args, **kwargs):
        result_queue = queue.Queue()
        self.queue.put((function, args, kwargs, result_queue))
```

```
        result = result_queue.get()
        if isinstance(result, Exception):
            raise result
        return result

    def insert_status(self, device_id, temperature, humidity, light, is_on):
        def _insert():
            cursor = self.conn.cursor()
            cursor.execute('''
            INSERT INTO status_history (device_id, temperature, humidity, light, is_on, timesta
            VALUES (?, ?, ?, ?, ?, ?)
            ''', (device_id, temperature, humidity, light, is_on, datetime.now()))
            self.conn.commit()
        self._execute(_insert)

    def query_history(self, device_id, start_time, end_time):
        def _query():
            cursor = self.conn.cursor()
            cursor.execute('''
            SELECT * FROM status_history
            WHERE device_id = ? AND timestamp BETWEEN ? AND ?
            ''', (device_id, start_time, end_time))
            return cursor.fetchall()
        return self._execute(_query)

    def close(self):
        self.queue.join()
        self.conn.close()
```

- server.py

```python
# server.py

import socket
import tkinter as tk
from tkinter import ttk
import threading
from udp_protocol import *
from database import Database
import time

class Server:
    def __init__(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(('0.0.0.0', 12345))
        self.clients = {}
        self.is_running = False
        self.db = Database()

        self.root = tk.Tk()
        self.root.title("路灯服务器")
        self.setup_ui()

    def setup_ui(self):
        self.root.geometry("1000x600")

        style = ttk.Style()
        style.theme_use('clam')

        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
        self.root.columnconfigure(0, weight=1)
        self.root.rowconfigure(0, weight=1)

        # 客户端列表
        client_frame = ttk.LabelFrame(main_frame, text="连接的路灯", padding="5")
        client_frame.grid(row=0, column=0, columnspan=2, sticky=(tk.W, tk.E, tk.N, tk.S), pady=
        main_frame.columnconfigure(0, weight=1)
        main_frame.rowconfigure(0, weight=1)

        self.client_tree = ttk.Treeview(client_frame, columns=('ID', 'Address', 'Temperature',
        self.client_tree.heading('ID', text='设备ID')
        self.client_tree.heading('Address', text='设备地址')
        self.client_tree.heading('Temperature', text='温度')
        self.client_tree.heading('Humidity', text='湿度')
        self.client_tree.heading('Light', text='光照')
        self.client_tree.heading('Status', text='状态')
```

```
        self.client_tree.heading('LastUpdate', text='最后更新')
        self.client_tree.column('#0', width=0, stretch=tk.NO)
        self.client_tree.column('ID', width=60, anchor=tk.CENTER)
        self.client_tree.column('Address', width=150, anchor=tk.CENTER)
        self.client_tree.column('Temperature', width=80, anchor=tk.CENTER)
        self.client_tree.column('Humidity', width=80, anchor=tk.CENTER)
        self.client_tree.column('Light', width=80, anchor=tk.CENTER)
        self.client_tree.column('Status', width=80, anchor=tk.CENTER)
        self.client_tree.column('LastUpdate', width=150, anchor=tk.CENTER)
        self.client_tree.pack(fill=tk.BOTH, expand=True)

        # 控制按钮
        control_frame = ttk.Frame(main_frame)
        control_frame.grid(row=1, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(0, 10))

        self.on_button = ttk.Button(control_frame, text="开灯", command=self.turn_on_selected)
        self.on_button.pack(side=tk.LEFT, padx=(0, 5))

        self.off_button = ttk.Button(control_frame, text="关灯", command=self.turn_off_selected
        self.off_button.pack(side=tk.LEFT)

        # 数据包列表
        packet_frame = ttk.Frame(main_frame)
        packet_frame.grid(row=2, column=0, columnspan=2, sticky=(tk.W, tk.E, tk.N, tk.S), pady=
        main_frame.rowconfigure(2, weight=1)

        upload_frame = ttk.LabelFrame(packet_frame, text="上传数据包", padding="5")
        upload_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0, 5))

        self.upload_list = tk.Listbox(upload_frame)
        self.upload_list.pack(fill=tk.BOTH, expand=True)

        download_frame = ttk.LabelFrame(packet_frame, text="下发数据包", padding="5")
        download_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        self.download_list = tk.Listbox(download_frame)
        self.download_list.pack(fill=tk.BOTH, expand=True)

        # 服务器控制
        server_frame = ttk.Frame(main_frame)
        server_frame.grid(row=3, column=0, columnspan=2, sticky=(tk.W, tk.E))

        self.start_button = ttk.Button(server_frame, text="启动服务器", command=self.start_serv
        self.start_button.pack(side=tk.LEFT, padx=(0, 5))

        self.stop_button = ttk.Button(server_frame, text="停止服务器", command=self.stop_server
```

```
        self.stop_button.pack(side=tk.LEFT)

    def start_server(self):
        self.is_running = True
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)
        threading.Thread(target=self.receive_data, daemon=True).start()
        threading.Thread(target=self.check_client_timeout, daemon=True).start()

    def stop_server(self):
        self.is_running = False
        self.start_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)

    def receive_data(self):
        while self.is_running:
            try:
                self.sock.settimeout(1.0)
                data, addr = self.sock.recvfrom(1024)
                packet = Packet.from_bytes(data)

                if packet.packet_type == STATUS_UPDATE:
                    self.handle_status_update(addr, packet)
                elif packet.packet_type == ACK:
                    self.handle_ack(packet)

            except socket.timeout:
                pass
            except Exception as e:
                print(f"Error receiving data: {e}")

    def handle_status_update(self, addr, packet):
        temperature, humidity, light, is_on = parse_status_packet(packet)
        self.clients[packet.device_id] = {'addr': addr, 'last_update': time.time()}
        self.db.insert_status(packet.device_id, temperature, humidity, light, is_on)

        self.root.after(0, self.update_client_list, packet.device_id, addr, temperature, humid
        self.root.after(0, self.upload_list.insert, tk.END, f"状态更新 (设备 {packet.device_id}

    def handle_ack(self, packet):
        self.root.after(0, self.download_list.insert, tk.END, f"确认 (设备 {packet.device_id})"

    def update_client_list(self, device_id, addr, temperature, humidity, light, is_on):
        current_time = time.strftime("%Y-%m-%d %H:%M:%S")
        item_found = False
        for item in self.client_tree.get_children():
```

```
            if self.client_tree.item(item)['values'][0] == device_id:
                item_found = True
                self.client_tree.item(item, values=(device_id, f"{addr[0]}:{addr[1]}", f"{tempe
                break

        if not item_found:
            self.client_tree.insert('', tk.END, values=(device_id, f"{addr[0]}:{addr[1]}", f"{t

    def turn_on_selected(self):
        self.send_control_command(True)

    def turn_off_selected(self):
        self.send_control_command(False)

    def send_control_command(self, is_on):
        selected = self.client_tree.selection()
        if not selected:
            return

        device_id = int(self.client_tree.item(selected[0])['values'][0])
        if device_id not in self.clients:
            return

        packet = create_control_packet(device_id, int(is_on))
        self.sock.sendto(packet.to_bytes(), self.clients[device_id]['addr'])
        self.download_list.insert(tk.END, f"控制命令 (设备 {device_id}): {'开启' if is_on else

    def check_client_timeout(self):
        while self.is_running:
            current_time = time.time()
            disconnected_clients = []
            for device_id, client_info in self.clients.items():
                if current_time - client_info['last_update'] > 5:  # 5秒超时
                    disconnected_clients.append(device_id)

            for device_id in disconnected_clients:
                del self.clients[device_id]
                self.root.after(0, self.remove_disconnected_client, device_id)

            time.sleep(1)

    def remove_disconnected_client(self, device_id):
        for item in self.client_tree.get_children():
            if self.client_tree.item(item)['values'][0] == device_id:
                self.client_tree.delete(item)
                break
```

```
            self.upload_list.insert(tk.END, f"设备 {device_id} 已断开连接")

    def run(self):
        self.root.mainloop()
        self.db.close()

if __name__ == "__main__":
    server = Server()
    server.run()
```

- streetlight_client.py

```python
# streetlight_client.py

import socket
import time
import tkinter as tk
from tkinter import ttk, simpledialog
import threading
from udp_protocol import *

class StreetlightClient:
    def __init__(self, server_address, network_interface=''):
        self.server_address = server_address
        self.device_id = None
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((network_interface, 0))  # Bind to the specified network interface
        self.ip, self.port = self.sock.getsockname()
        self.is_on = False
        self.is_running = False

        self.root = tk.Tk()
        self.root.title("路灯客户端")
        self.setup_ui()

    def setup_ui(self):
        self.root.geometry("300x350")
        self.root.resizable(False, False)

        style = ttk.Style()
        style.theme_use('clam')

        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.temp_var = tk.StringVar(value="温度: N/A")
        self.humidity_var = tk.StringVar(value="湿度: N/A")
        self.light_var = tk.StringVar(value="光照: N/A")
        self.status_var = tk.StringVar(value="状态: 关闭")
        self.id_var = tk.StringVar(value="设备ID: 未设置")
        self.ip_var = tk.StringVar(value=f"IP地址: {self.ip}")
        self.port_var = tk.StringVar(value=f"端口: {self.port}")

        ttk.Label(main_frame, textvariable=self.temp_var).grid(column=0, row=0, sticky=tk.W, pa
        ttk.Label(main_frame, textvariable=self.humidity_var).grid(column=0, row=1, sticky=tk.
        ttk.Label(main_frame, textvariable=self.light_var).grid(column=0, row=2, sticky=tk.W, p

        self.status_label = ttk.Label(main_frame, textvariable=self.status_var, background="re
```

```
        self.status_label.grid(column=0, row=3, sticky=(tk.W, tk.E), pady=10)

        ttk.Label(main_frame, textvariable=self.id_var).grid(column=0, row=4, sticky=tk.W, pady
        ttk.Label(main_frame, textvariable=self.ip_var).grid(column=0, row=5, sticky=tk.W, pady
        ttk.Label(main_frame, textvariable=self.port_var).grid(column=0, row=6, sticky=tk.W, pa

        button_frame = ttk.Frame(main_frame)
        button_frame.grid(column=0, row=7, sticky=(tk.W, tk.E), pady=10)

        self.start_button = ttk.Button(button_frame, text="启动", command=self.start)
        self.start_button.pack(side=tk.LEFT, padx=(0, 10))

        self.stop_button = ttk.Button(button_frame, text="停止", command=self.stop, state=tk.DI
        self.stop_button.pack(side=tk.LEFT, padx=(0, 10))

        self.set_id_button = ttk.Button(button_frame, text="设置ID", command=self.set_device_ic
        self.set_id_button.pack(side=tk.LEFT)

    def set_device_id(self):
        new_id = simpledialog.askinteger("设置设备ID", "请输入新的设备ID (1-1000):", minvalue=1,
        if new_id:
            self.device_id = new_id
            self.id_var.set(f"设备ID: {self.device_id}")
            self.root.title(f"路灯客户端 {self.device_id}")

    def start(self):
        if not self.device_id:
            tk.messagebox.showerror("错误", "请先设置设备ID")
            return
        if not self.is_running:
            self.is_running = True
            self.start_button.config(state=tk.DISABLED)
            self.stop_button.config(state=tk.NORMAL)
            self.set_id_button.config(state=tk.DISABLED)
            threading.Thread(target=self.update_status, daemon=True).start()
            threading.Thread(target=self.receive_commands, daemon=True).start()

    def stop(self):
        self.is_running = False
        self.start_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)
        self.set_id_button.config(state=tk.NORMAL)

    def update_status(self):
        while self.is_running:
            temperature, humidity, light = generate_sensor_data()
```

```
            self.temp_var.set(f"温度: {temperature:.2f}°C")
            self.humidity_var.set(f"湿度: {humidity:.2f}%")
            self.light_var.set(f"光照: {light:.2f} lux")

            packet = create_status_packet(self.device_id, temperature, humidity, light, self.i
            self.sock.sendto(packet.to_bytes(), self.server_address)

            time.sleep(1)

    def receive_commands(self):
        while self.is_running:
            try:
                self.sock.settimeout(1.0)
                data, _ = self.sock.recvfrom(1024)
                packet = Packet.from_bytes(data)
                if packet.packet_type == CONTROL_COMMAND:
                    command = parse_control_packet(packet)
                    self.is_on = bool(command)
                    self.root.after(0, self.update_status_display)

                    # Send ACK
                    ack_packet = create_ack_packet(self.device_id)
                    self.sock.sendto(ack_packet.to_bytes(), self.server_address)
            except socket.timeout:
                pass
            except Exception as e:
                print(f"Error receiving command: {e}")

    def update_status_display(self):
        if self.is_on:
            self.status_var.set("状态: 开启")
            self.status_label.config(background="green")
        else:
            self.status_var.set("状态: 关闭")
            self.status_label.config(background="red")

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    server_address = ('192.168.56.1', 12345)  # 替换为实际的服务器IP地址

    # 为不同的虚拟网络创建客户端实例.
    clients = [
        StreetlightClient(server_address, '192.168.56.1'),  # VirtualBox Host-Only Network
        # StreetlightClient(server_address, '192.168.11.1'),  # VMware Network Adapter VMnet1
```

```
        # StreetlightClient(server_address, '192.168.226.1'), # VMware Network Adapter VMnet8
        # StreetlightClient(server_address, '10.15.0.242')    # Wireless LAN adapter WLAN
    ]

    # 运行所有客户端
    for client in clients:
        client.run()
```