TCP/IP Model Implementation in Station Arcturus

Overview

Station Arcturus is a real-time beacon tracking system demonstrating the TCP/IP networking model. It uses a Python FastAPI backend communicating with Unity 3D and HTML5 clients via WebSocket for real-time beacon position updates.

TCP/IP Model Layers

1. Network Access Layer (Link Layer)

What it does: Handles physical data transmission over network hardware (Ethernet, Wi-Fi).

Implementation: Managed entirely by the OS and network interface cards. Our application doesn't interact with this layer directly - Windows uses Winsock2, macOS uses BSD Sockets.

2. Internet Layer

What it does: Handles IP addressing and packet routing.

Implementation:

```
uvicorn.run("app.main:socket_app", host="0.0.0.0", port=8000, reload=True)
```

- **Server:** Binds to `0.0.0.0:8000` (all network interfaces)
- **Clients:** Connect to `127.0.0.1:8000` (IPv4 loopback for local development)
- All HTTP and WebSocket traffic uses IPv4 packets managed by the OS

3. Transport Layer

What it does: Provides reliable, connection-oriented communication using TCP. Handles connection setup/teardown, flow control, and error checking.

Implementation: We use **TCP exclusively** for all communication.

HTTP over TCP (REST API):

```
@app.get("/beacons")
async def get_beacons():
    beacon_list = [b.to_unity_dict() for b in beacons.beacons_data.values()]
    return {"beacons": beacon_list, "count": len(beacon_list)}
```

WebSocket over TCP (Real-Time Updates):

```
sio = socketio.AsyncServer(async_mode='asgi', cors_allowed_origins='*')
await sio.emit("beacon_update", {"beacons": beacon_list, "time": time.time()})
```

Key Libraries:

- **Backend:** `uvicorn`, `python-socketio`, `python-engineio`, `wsproto`, `h11`
- **Unity:** `SocketIOClient` (C#)
- **HTML:** Socket.IO JavaScript client (v4.5.4)

Port Multiplexing: Our single port (8000) handles multiple traffic types:

- HTTP REST API (`/beacons`, `/status`, `/ui`)
- WebSocket connections (Socket.IO real-time updates)
- Static files (`/static/*`)

Each client gets a unique socket pair (e.g., `127.0.0.1:54321 → 127.0.0.1:8000`), allowing multiple simultaneous connections on one port.

4. Application Layer

What it does: Implements high-level protocols for data formatting and application-specific logic.

Protocols Used:

1. HTTP/1.1 (REST API)

```
@app.get("/beacons")
async def get_beacons():
    return {"beacons": beacon_list, "count": len(beacon_list)}
```

- Libraries: `fastapi` (0.121.1), `starlette` (0.49.3), `pydantic` (2.12.4)

2. WebSocket (RFC 6455)

- Upgrades from HTTP via `101 Switching Protocols`
- Uses text frames for JSON data, ping/pong for health checks
- Maintains persistent bidirectional connection

3. Socket.IO Protocol

```
@sio.event
async def connect(sid, environ):
    await sio.emit('beacon_update', {"beacons": beacon_list}, to=sid)

async def beacon_updater(sio):
    while True:
        await sio.emit("beacon_update", {"beacons": beacon_list})
        await asyncio.sleep(2)  # Broadcast every 2 seconds
```

- Event-based messaging: `beacon_update`, `connect`, `disconnect`, `ping`, `pong`
- Automatic reconnection and JSON serialization

4. JSON Data Format

```
class Beacon(BaseModel):
    id: str
    x: float
    altitude: float
    z: float
    status: BeaconStatus  # ACTIVE, DAMAGED, OFFLINE
```

5. CORS (Development Mode)

```
app.add_middleware(CORSMiddleware, allow_origins=["*"])
```

Data Flow Example: Beacon Update Through TCP/IP Stack

```
APPLICATION:  beacon_updater() → JSON → sio.emit("beacon_update") → Socket.IO
                ↓
TRANSPORT:    python-socketio → WebSocket frame → TCP socket (port 8000)
                ↓
INTERNET:     IP packet (127.0.0.1:8000 → 127.0.0.1:54321)
                ↓
NETWORK:      Loopback interface (OS kernel routing)
                ↓
CLIENT:       Unity receives → JSON.parse → Updates GameObjects
```

Connection Flow:

1. **TCP Handshake:** SYN → SYN-ACK → ACK
2. **HTTP Upgrade:** `GET /socket.io/` → `101 Switching Protocols`
3. **WebSocket Established:** Persistent bidirectional connection
4. **Real-Time Updates:** Server broadcasts beacon data every 2 seconds

---

Key Libraries by Layer

Component	Libraries	Layer
Backend	`uvicorn`, `fastapi`, `python-socketio`, `python-engineio`, `wsproto`, `h11`, `pydantic`	Application/Transport
Unity Client	`SocketIOClient`, `Newtonsoft.Json`	Application
HTML Client	Socket.IO client (v4.5.4), Browser WebSocket API	Application

Quick Troubleshooting

Issue	Layer	Solution
"Connection refused"	Transport	Start backend server (port 8000)
"CORS error"	Application	Check CORS middleware configuration
"WebSocket handshake failed"	Application	Verify WebSocket upgrade support
Beacon data not updating	Application	Check Socket.IO event listeners

Summary

Station Arcturus demonstrates the TCP/IP model with:

- **Network Access Layer:** OS/hardware (transparent to application)
- **Internet Layer:** IPv4 addressing (`127.0.0.1:8000`)
- **Transport Layer:** TCP for reliable delivery (HTTP and WebSocket)
- **Application Layer:** HTTP/1.1, WebSocket, Socket.IO, JSON

The architecture uses FastAPI + Socket.IO for real-time bidirectional communication between Python backend and Unity/HTML clients, with all four TCP/IP layers working together to deliver beacon updates every 2 seconds.