main.py - FastAPI Server & API Endpoints

Overview

File: `server/app/main.py`
Purpose: HTTP server that receives packets and provides web dashboards
Role: Central data receiver and API provider
Runs as: Web server (via uvicorn)

What It Does

`main.py` is the FastAPI server that handles all HTTP communication:

1. **Receives** raw packet data from sender.py via POST `/ingest_packets`
2. **Stores** packets in PostgreSQL database (`raw_packets` table)
3. **Provides** web dashboards to view traffic flows
4. **Serves** JSON APIs for programmatic access
5. **Handles** ML predictions for aggregated flows (legacy endpoint)

Architecture

```
sender.py → POST /ingest_packets → main.py → PostgreSQL
                                              ↓
                                        raw_packets table
                                              ↓
                                   (aggregator.py processes)
                                              ↓
                                         traffic_data table
                                              ↓
                              GET /api/last10 ← main.py serves
```

Key Components

1. Database Setup (Lines 20-82)

Tables Created:

raw_packets - Complete packet logs

```
raw_packets_table = Table(
    "raw_packets", metadata,
    Column("id", Integer, primary_key=True, autoincrement=True),
    Column("timestamp", Float),
```

```
    Column("interface", String),
    Column("src_ip", String),
    Column("dst_ip", String),
    Column("protocol", String),
    Column("length", Integer),
    Column("src_port", Integer, nullable=True),
    Column("dst_port", Integer, nullable=True),
    Column("tcp_flags", String, nullable=True),
    # ... 20+ more columns for DNS, HTTP, ICMP, etc.
    Column("inserted_at", DateTime, default=datetime.utcnow)
)
```

Purpose: Audit trail and forensics - stores every packet captured

---

traffic_data - Aggregated flows with ML predictions

```
traffic_table = Table(
    "traffic_data", metadata,
    Column("id_num", Integer, primary_key=True, autoincrement=True),
    Column("dest_ip", String),
    Column("source_mac", String),
    Column("dest_mac", String),
    Column("packet_count", Integer),
    Column("packet_per_sec", Float),
    Column("byte_count", Integer),
    Column("byte_per_sec", Float),
    Column("tcp_flags", String),
    Column("connection_attempts", Integer),
    Column("unique_ports", Integer),
    Column("protocol", String),
    Column("predicted_label", String),  # ML prediction
    Column("created_at", DateTime, default=datetime.utcnow)
)
```

Purpose: Stores aggregated network flows with ML intrusion detection labels

---

2. Pydantic Models (Lines 92-141)

Models define API request/response formats:

RawPacket - For `/ingest_packets` endpoint

```
class RawPacket(BaseModel):
    timestamp: float
    interface: str
```

```
    src_ip: Optional[str] = None
    dst_ip: Optional[str] = None
    protocol: Optional[str] = None
    length: int
    # ... 25+ more optional fields
```

Validates incoming packet data from sender.py

Traffic - For /predict endpoint (legacy)

```
class Traffic(BaseModel):
    dest_ip: str
    source_mac: str
    dest_mac: str
    packet_count: int
    packet_per_sec: float
    # ... aggregated flow features
```

For manual ML prediction requests

---

3. ML Model Loading (Lines 85-90)

```
try:
    model = joblib.load("AI_model.pkl")
    logger.info("ML model loaded successfully")
except Exception as e:
    logger.error(f"Failed to load ML model: {e}")
    model = None
```

Model file: `server/models/AI_model.pkl`
Used by: `/predict` endpoint for intrusion detection
Graceful handling: Server still runs if model missing

---

4. API Endpoints

POST /ingest_packets (Lines 172-191) **Purpose:** Receive batch of raw packets from sender.py

Request:

```
[
  {
    "timestamp": 1701234567.123,
    "interface": "Wi-Fi",
```

```
    "src_ip": "192.168.1.100",
    "dst_ip": "8.8.8.8",
    "protocol": "TCP",
    "length": 60,
    "src_port": 54321,
    "dst_port": 443,
    "tcp_flags": "SYN",
    "tcp_syn": true,
    ...
  },
  ...
]
```

Response:

```
{
  "status": "success",
  "packets_received": 150,
  "message": "Packets stored in raw_packets table"
}
```

Code:

```
@app.post("/ingest_packets")
async def ingest_packets(packets: List[RawPacket]):
    with engine.begin() as conn:
        for packet in packets:
            ins = raw_packets_table.insert().values(**packet.dict())
            conn.execute(ins)

    logger.info(f"Inserted {len(packets)} raw packets")
    return {"status": "success", "packets_received": len(packets)}
```

POST /predict (Lines 193-214) **Purpose:** Predict traffic label for aggregated flow (legacy)

Request:

```
{
  "dest_ip": "8.8.8.8",
  "source_mac": "unknown",
  "dest_mac": "unknown",
  "packet_count": 150,
  "packet_per_sec": 5.0,
  "byte_count": 9000,
  "byte_per_sec": 300.0,
  "tcp_flags": "SYN,ACK",
```

```
  "connection_attempts": 3,
  "unique_ports": 2,
  "protocol": "TCP"
}
```

Response:

```
{
  "predicted_label": "Normal",
  "data": { ... }
}
```

Note: Mostly replaced by aggregator.py which automates this

GET / (Lines 219-267) **Purpose:** Web dashboard showing last 10 traffic flows

Returns: HTML page with auto-refreshing table

Features: - Shows last 10 flows from `traffic_data` table - Auto-refreshes every 5 seconds via JavaScript - Displays: ID, IPs, MACs, packet stats, predicted label

---

GET /alltraffic_page (Lines 269-317) **Purpose:** Web dashboard showing all traffic flows

Same as / but shows all flows (could be thousands)

GET /api/last10 (Lines 320-326) **Purpose:** JSON API for last 10 flows

Response:

```
[
  {
    "id_num": 123,
    "dest_ip": "8.8.8.8",
    "packet_count": 150,
    "predicted_label": "Normal",
    ...
  },
  ...
]
```

Used by: Web dashboards, external tools

GET /api/alltraffic (Lines 328-334) **Purpose:** JSON API for all flows

Warning: Can return huge amounts of data

GET /api/raw_packets/last/{count} (Lines 336-342) **Purpose:** Get last N raw packets

Example: `/api/raw_packets/last/100`

Response:

```
[
  {
    "id": 5234,
    "timestamp": 1701234567.123,
    "src_ip": "192.168.1.100",
    "dst_ip": "8.8.8.8",
    ...
  },
  ...
]
```

GET /health (Lines 344-351) **Purpose:** Health check for monitoring

Response:

```
{
  "status": "healthy",
  "model_loaded": true,
  "timestamp": "2025-11-29T21:50:00"
}
```

Used by: sender.py to verify server is up

Data Flow Example

Scenario: sender.py uploads 150 packets

```
1. sender.py sends HTTP POST:
   POST /ingest_packets
   Content-Type: application/json

   [150 packet objects]

2. FastAPI receives request:
```

```
   - Validates each packet against RawPacket model
   - Converts JSON → Python dictionaries

3. Database insertion:
   with engine.begin() as conn:
       for packet in packets:
           INSERT INTO raw_packets VALUES (...)

4. Response sent:
   {
     "status": "success",
     "packets_received": 150,
     "message": "Packets stored in raw_packets table"
   }

5. aggregator.py (separate process):
   - Queries new packets from raw_packets
   - Aggregates into flows
   - Runs ML predictions
   - Inserts into traffic_data

6. Web dashboard refreshes:
   - Calls /api/last10
   - Gets aggregated flows with predictions
   - Updates HTML table
```

Configuration

Database Connection (Line 21):

```
DATABASE_URL = "postgresql://postgres:987456@localhost:5432/Traffic_Analyzer"
```

Change to:

```
DATABASE_URL = "postgresql://USER:PASSWORD@HOST:PORT/DATABASE"
```

Model Path (Line 86):

```
model = joblib.load("AI_model.pkl")
```

Make sure `AI_model.pkl` is in `server/models/` directory

Running the Server

Start Server

```
cd server
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

Options: - `--host 0.0.0.0` - Listen on all network interfaces - `--port 8000` - Port number - `--reload` - Auto-reload on code changes (development) - `--workers 4` - Multiple worker processes (production)

Production Deployment

```
uvicorn app.main:app \
  --host 0.0.0.0 \
  --port 8000 \
  --workers 4 \
  --log-level info \
  --access-log
```

Monitoring

Check Server Status

```
curl http://localhost:8000/health
```

Test Packet Ingestion

```
curl -X POST http://localhost:8000/ingest_packets \
  -H "Content-Type: application/json" \
  -d '[
    {
      "timestamp": 1701234567.123,
      "interface": "test",
      "src_ip": "192.168.1.1",
      "dst_ip": "8.8.8.8",
      "protocol": "TCP",
      "length": 60
    }
  ]'
```

View Web Dashboard

```
http://SERVER_IP:8000/
```

Query API

```
curl http://localhost:8000/api/last10
```

Troubleshooting

"Connection refused"

Cause: Server not running

Solution:

```
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

"relation 'raw_packets' does not exist"

Cause: Database tables not created

Solution:

```
# Tables auto-created on startup:
metadata.create_all(engine)  # Line 81
```

Check database:

```
psql -U postgres -d Traffic_Analyzer
\dt  # List tables
```

"ML model not loaded"

Cause: AI_model.pkl missing

Solution: 1. Check file exists: `ls server/models/AI_model.pkl` 2. If missing, train model or copy from backup 3. Server still works without model (for ingestion only)

Slow Response Times

Causes: - Large database queries - Too many concurrent requests

Solutions: 1. Add database indexes: `sql CREATE INDEX idx_timestamp ON raw_packets(timestamp); CREATE INDEX idx_created_at ON traffic_data(created_at DESC);`

2. Use pagination:

```
    # Instead of all:
    limit(100).offset(0)
```

3. Scale with more workers:

```
uvicorn app.main:app --workers 8
```

Performance Tips

Database Connection Pooling

```
engine = create_engine(
    DATABASE_URL,
    pool_pre_ping=True,     # Already set
    pool_size=20,           # Add this
    max_overflow=40         # Add this
)
```

Batch Inserts (More Efficient)

```
# Current: Loop with individual inserts
for packet in packets:
    conn.execute(ins)

# Better: Bulk insert
conn.execute(
    raw_packets_table.insert(),
    [packet.dict() for packet in packets]
)
```

Add Response Compression

```
pip install fastapi[all]
```

```
from fastapi.middleware.gzip import GZipMiddleware
app.add_middleware(GZipMiddleware, minimum_size=1000)
```

Summary

main.py provides: HTTP API for receiving packet data
Database storage for raw packets
Web dashboards for viewing flows
JSON APIs for programmatic access
Health check endpoint
ML prediction capability (with model)

It does NOT: Capture packets (that's sniffer.py)
Upload files (that's sender.py)
Aggregate flows (that's aggregator.py)

Dependencies: FastAPI, SQLAlchemy, PostgreSQL, ML model

Default Port: 8000
Web UI: `http://SERVER_IP:8000/`