aggregator.py - Flow Aggregation & ML Prediction Service

Overview

File: `server/aggregator.py`
Purpose: Convert raw packets into aggregated flows and run ML predictions
Role: Background data processor
Runs as: Standalone async service

What It Does

`aggregator.py` transforms raw packet data into meaningful network flows:

1. **Queries** new packets from `raw_packets` table every 30 seconds
2. **Groups** packets by flow key: `(src_ip, dst_ip, protocol)`
3. **Calculates** flow statistics (packet count, byte rates, unique ports, etc.)
4. **Predicts** attack labels using ML model
5. **Stores** results in `traffic_data` table
6. **Cleans up** old raw packets (>7 days) every hour

Architecture

```
PostgreSQL: raw_packets table
       ↓
(Every 30 seconds)
       ↓
aggregator.py queries new packets
       ↓
Group by (src_ip, dst_ip, protocol)
       ↓
Calculate flow statistics
       ↓
Run ML model prediction
       ↓
Insert into traffic_data table
       ↓
Web dashboard displays flows
```

Key Components

1. Configuration (Lines 26-40)

```
DATABASE_URL = "postgresql://postgres:987456@localhost:5432/Traffic_Analyzer"
model_path = os.path.join(os.path.dirname(__file__), 'models', 'AI_model.pkl')
```

Aggregation Timing (Line 138):

```
await asyncio.sleep(30)  # Aggregate every 30 seconds
```

Cleanup Timing (Line 150):

```
await asyncio.sleep(3600)  # Run cleanup every hour
timedelta(days=7)  # Keep packets for 7 days
```

---

2. Flow Aggregation Logic (Lines 70-135)

Function: `aggregate_packets_to_flows(packets)`

Purpose: Convert list of raw packets into aggregated flow statistics

Flow Key:

```
flow_key = (src_ip, dst_ip, protocol)
```

Example:

```
Packets:
1. 192.168.1.1 → 8.8.8.8  TCP  60 bytes
2. 192.168.1.1 → 8.8.8.8  TCP  80 bytes
3. 192.168.1.1 → 8.8.8.8  TCP  100 bytes
4. 192.168.1.1 → 1.1.1.1  TCP  50 bytes

Flows:
1. (192.168.1.1, 8.8.8.8, TCP) → 3 packets, 240 bytes
2. (192.168.1.1, 1.1.1.1, TCP) → 1 packet, 50 bytes
```

Calculated Statistics:

```
for pkt in packets:
    flow_key = (pkt['src_ip'], pkt['dst_ip'], pkt['protocol'])
    flow = flows[flow_key]

    # Count packets
    flow['packet_count'] += 1

    # Sum bytes
    flow['byte_count'] += pkt['length']

    # Track unique ports
```

```
    if pkt['src_port']:
        flow['unique_ports'].add(pkt['src_port'])
    if pkt['dst_port']:
        flow['unique_ports'].add(pkt['dst_port'])

    # Collect TCP flags
    if pkt['tcp_flags']:
        for flag in pkt['tcp_flags'].split(','):
            flow['tcp_flags'].add(flag)

    # Count connection attempts (SYN packets)
    if pkt.get('tcp_syn'):
        flow['connection_attempts'] += 1

    # Track time range
    flow['start_time'] = min(flow['start_time'], pkt['timestamp'])
    flow['end_time'] = max(flow['end_time'], pkt['timestamp'])
```

Rate Calculations:

```
duration = (end_time - start_time).total_seconds()
if duration == 0:
    duration = 1  # Avoid division by zero

flow['packet_per_sec'] = packet_count / duration
flow['byte_per_sec'] = byte_count / duration
```

Example Output:

```
{
    'dest_ip': '8.8.8.8',
    'protocol': 'TCP',
    'packet_count': 150,
    'byte_count': 9000,
    'packet_per_sec': 5.0,
    'byte_per_sec': 300.0,
    'tcp_flags': 'ACK,PSH,SYN',
    'connection_attempts': 3,
    'unique_ports': 2,
    'source_mac': 'unknown',  # Not captured
    'dest_mac': 'unknown'
}
```

---

3. ML Model Integration (Lines 51-68)

Function: `prepare_features(data)`

Purpose: Convert flow dictionary into ML model input

Steps:

1. **Create DataFrame:**

```
df = pd.DataFrame([data])
```

2. **Ensure correct types:**

```
df['protocol'] = df['protocol'].astype(str)
df['tcp_flags'] = df['tcp_flags'].astype(str)
```

3. **One-hot encoding:**

```
df = pd.get_dummies(df, columns=['protocol', 'tcp_flags'])
# Before: protocol='TCP', tcp_flags='SYN,ACK'
# After:  protocol_TCP=1, tcp_flags_SYN;ACK=1, protocol_UDP=0, ...
```

4. **Add missing columns:**

```
for col in model.feature_names_in_:
    if col not in df.columns:
        df[col] = 0
```

5. **Reorder columns:**

```
df = df[model.feature_names_in_]
```

6. **Convert to float:**

```
df = df.astype(float)
```

Result: DataFrame ready for `model.predict()`

---

4. Main Aggregation Loop (Lines 138-185)

Function: `aggregate_and_predict()`

Purpose: Continuously process new packets

Flow:

```
last_processed_id = 0  # Track position in table

while True:  # Forever loop
    # 1. Query new packets since last run
    query = select(raw_packets_table).where(
        raw_packets_table.c.id > last_processed_id
    ).order_by(raw_packets_table.c.id)

    packets = session.execute(query).all()
```

```
    if packets:
        # 2. Aggregate into flows
        flows = aggregate_packets_to_flows(packets)

        # 3. For each flow:
        for flow in flows:
            # Run ML prediction
            if model:
                features = prepare_features(flow)
                flow['predicted_label'] = model.predict(features)[0]
            else:
                flow['predicted_label'] = 'unknown'

            # Insert into traffic_data table
            session.execute(traffic_table.insert().values(**flow))

        session.commit()

        # 4. Update tracker
        last_processed_id = max(pkt['id'] for pkt in packets)

    # 5. Wait 30 seconds before next cycle
    await asyncio.sleep(30)
```

Timeline:

```
00:00 - Process packets 1-500    → Create 50 flows
00:30 - Process packets 501-1200 → Create 75 flows
01:00 - Process packets 1201-1800 → Create 60 flows
...
```

---

5. Cleanup Service (Lines 187-209)

Function: `cleanup_old_raw_packets()`

Purpose: Prevent database bloat by deleting old packets

```
async def cleanup_old_raw_packets():
    while True:
        await asyncio.sleep(3600)  # Run every hour

        # Delete packets older than 7 days
        cutoff_date = datetime.utcnow() - timedelta(days=7)

        delete_query = raw_packets_table.delete().where(
            raw_packets_table.c.inserted_at < cutoff_date
```

```
        )

        result = session.execute(delete_query)
        logger.info(f"Cleaned up {result.rowcount} old packets")
```

Why needed? - `raw_packets` table grows continuously - On high-traffic network: 10,000+ packets/sec = 864 million/day - After aggregation, raw packets less useful

Retention policy: 7 days (configurable)

Data Flow Example

Scenario: Aggregating 1000 packets into flows

```
1. Raw packets in database:
   ID: 1001, src_ip: 192.168.1.1, dst_ip: 8.8.8.8, protocol: TCP
   ID: 1002, src_ip: 192.168.1.1, dst_ip: 8.8.8.8, protocol: TCP
   ID: 1003, src_ip: 192.168.1.1, dst_ip: 8.8.8.8, protocol: TCP
   ... (1000 total)

2. Aggregator queries (every 30s):
   SELECT * FROM raw_packets WHERE id > 1000 ORDER BY id

3. Group by flow key:
   Flow 1: (192.168.1.1, 8.8.8.8, TCP) → 750 packets
   Flow 2: (192.168.1.1, 1.1.1.1, TCP) → 150 packets
   Flow 3: (192.168.1.2, 8.8.8.8, UDP) → 100 packets

4. Calculate statistics:
   Flow 1: {
     packet_count: 750,
     byte_count: 45000,
     packet_per_sec: 25.0,
     byte_per_sec: 1500.0,
     unique_ports: 2,
     connection_attempts: 1,
     tcp_flags: 'SYN,ACK,PSH,FIN'
   }

5. ML Prediction:
   features = prepare_features(Flow 1)
   prediction = model.predict(features)
   → "Normal"

6. Insert into traffic_data:
```

```
   INSERT INTO traffic_data VALUES (
     dest_ip='8.8.8.8',
     predicted_label='Normal',
     ...
   )
```

7. Update tracker:
   last_processed_id = 2000

Configuration

Change Aggregation Interval

```
# Line 138
await asyncio.sleep(60)  # Aggregate every 60 seconds
```

Effect: Less frequent aggregation, lower CPU usage

Change Retention Period

```
# Line 150
cutoff_date = datetime.utcnow() - timedelta(days=30)  # Keep 30 days
```

Effect: More disk space used, longer history

Disable Cleanup (Keep All Raw Packets)

```
# Comment out in main() function:
# await cleanup_old_raw_packets()
```

Running the Aggregator

Start Service

```
cd server
python aggregator.py
```

Run as Background Service (Linux)

```
nohup python aggregator.py > aggregator.log 2>&1 &
```

Run with systemd (Linux)

Create `/etc/systemd/system/aggregator.service`:

```
[Unit]
Description=Network Analyzer Aggregator
After=network.target postgresql.service

[Service]
Type=simple
User=www-data
WorkingDirectory=/path/to/server
ExecStart=/usr/bin/python3 aggregator.py
Restart=always

[Install]
WantedBy=multi-user.target
```

```
sudo systemctl enable aggregator
sudo systemctl start aggregator
```

Monitoring

Check if Running

```
ps aux | grep aggregator.py
```

View Logs

```
# If running in foreground:
python aggregator.py

# If running in background:
tail -f aggregator.log
```

Check Database Activity

```
-- Count raw packets
SELECT COUNT(*) FROM raw_packets;

-- Count flows
SELECT COUNT(*) FROM traffic_data;

-- Recent flows
SELECT * FROM traffic_data ORDER BY created_at DESC LIMIT 10;

-- Flows by label
SELECT predicted_label, COUNT(*)
FROM traffic_data
GROUP BY predicted_label;
```

Troubleshooting

Aggregator Not Processing Packets

Check: 1. Are new packets in `raw_packets`? `sql SELECT COUNT(*) FROM raw_packets;`

2. Is aggregator running?

   ```
   ps aux | grep aggregator
   ```

3. Check logs for errors

"ML model not loaded"

Cause: `AI_model.pkl` missing

Solution: 1. Verify file exists: `bash ls server/models/AI_model.pkl`

2. Check path in aggregator.py:

   ```
   model_path = os.path.join(os.path.dirname(__file__), 'models', 'AI_model.pkl')
   ```

3. Aggregator still works without model (sets `predicted_label='unknown'`)

Slow Aggregation

Cause: Too many packets to process

Solutions: 1. Increase aggregation interval: `python await asyncio.sleep(60) # 60s instead of 30s`

2. Add database index:

   ```
   CREATE INDEX idx_id ON raw_packets(id);
   ```

3. Process in smaller batches:

   ```
   query = select(raw_packets_table).where(...).limit(1000)
   ```

Database Growing Too Large

Cause: Cleanup not running or retention too long

Solutions: 1. Verify cleanup is active: `python # Check main() function has both tasks await asyncio.gather( aggregate_and_predict(), cleanup_old_raw_packets() # This line needed! )`

2. Reduce retention:

   ```
   timedelta(days=3)  # 3 days instead of 7
   ```

3. Manual cleanup:

   ```
   DELETE FROM raw_packets
   WHERE inserted_at < NOW() - INTERVAL '7 days';
   ```

Performance Tips

Batch Processing

```
# Process max 5000 packets per cycle
query = select(raw_packets_table).where(...).limit(5000)
```

Optimize Flow Grouping

Use defaultdict for faster key access (already implemented):

```
from collections import defaultdict
flows = defaultdict(lambda: {...})
```

Database Optimization

```
-- Add indexes
CREATE INDEX idx_timestamp ON raw_packets(timestamp);
CREATE INDEX idx_id ON raw_packets(id);
CREATE INDEX idx_inserted_at ON raw_packets(inserted_at);
```

Summary

aggregator.py provides: Automatic packet → flow conversion
Network traffic statistics
ML-based intrusion detection
Continuous background processing
Automatic database cleanup

It does NOT: Capture packets (that's sniffer.py)
Receive HTTP requests (that's main.py)
Upload files (that's sender.py)

Dependencies: SQLAlchemy, pandas, joblib, asyncio
Runs: Continuously as background service
CPU Usage: Low (only during 30s processing cycles)