UDP Port 3131 Conflict Fix

Issue Summary

The application was experiencing `EADDRINUSE` errors when accessing the Audio Control Center page, caused by multiple components attempting to bind to UDP port 3131 simultaneously.

Root Cause Analysis

Problem Identified

TWO separate locations in the codebase were creating UDP sockets and binding to port 3131:

1. **`src/lib/atlasClient.ts` (lines 213-237)**
   - The `AtlasTCPClient` class creates a UDP socket in `initializeUdpSocket()` method
   - Called automatically when TCP connection is established (line 152)
   - Binds to UDP port 3131 (line 229) to receive meter updates from Atlas processors

2. **`src/app/api/audio-processor/input-levels/route.ts` (lines 89-127)**
   - The API route created a DUPLICATE UDP server in `startInputLevelMonitoring()` function
   - Also attempted to bind to UDP port 3131 (line 117)
   - Used a global map `activeUdpServers` to track servers

Why This Caused Conflicts

- When the Audio Control Center page loaded, it triggered the input level monitoring API
- The API tried to create a new UDP server on port 3131
- BUT port 3131 was already bound by the `AtlasTCPClient` UDP socket
- Result: **EADDRINUSE error** - “Address already in use”

Additional Issues Found

1. ❌ **No cleanup logic**: The route.ts UDP server was never properly cleaned up
2. ❌ **No singleton pattern**: Each processor could potentially create its own UDP server
3. ❌ **Race condition**: Multiple requests could try to create UDP servers simultaneously
4. ❌ **Duplicate functionality**: Both implementations received the same meter updates
5. ❌ **Poor resource management**: No reference counting or automatic cleanup

Solution Implemented

1. Created Centralized Atlas Client Manager ✅

File: `src/lib/atlas-client-manager.ts` (NEW)

Implemented a **Singleton Pattern** to manage all Atlas TCP/UDP connections:

- **Single UDP Socket per Processor**: Ensures only ONE UDP socket is created for each Atlas processor
- **Reference Counting**: Tracks how many components are using each client
- **Automatic Cleanup**: Idle clients (ref count = 0 for 10+ minutes) are automatically cleaned up

- **Connection Reuse**: Multiple requests to the same processor reuse the existing client
- **Callback System**: Extended `AtlasTCPClient` with callback support for meter updates

Key Features:

```
class AtlasClientManager {
  // Singleton instance
  private static instance: AtlasClientManager

  // Client management
  private clients: Map<string, ManagedClient>

  // Get or create client (with ref counting)
  public async getClient(processorId: string, config: AtlasConnectionConfig)

  // Release client (decrement ref count)
  public releaseClient(ipAddress: string, tcpPort?: number)

  // Auto cleanup of idle clients
  private cleanupIdleClients()
}
```

2. Extended AtlasTCPClient with Callbacks ✅

Created `ExtendedAtlasClient` class that:
- Extends `AtlasTCPClient` to add callback support
- Allows multiple consumers to register for meter updates
- Overrides `handleParameterUpdate()` to call registered callbacks
- No changes to existing Atlas protocol implementation

3. Refactored Input Levels API Route ✅

File: `src/app/api/audio-processor/input-levels/route.ts`

Removed:
- ❌ Duplicate UDP server creation (`dgram.createSocket`)
- ❌ Manual UDP socket binding to port 3131
- ❌ Custom TCP subscription management
- ❌ Global `activeUdpServers` map

Added:
- ✅ Use centralized `getAtlasClient()` function
- ✅ Register callback for meter updates
- ✅ Subscribe via centralized Atlas client
- ✅ Simplified subscription tracking

Before:

```
// OLD: Created duplicate UDP server
const udpServer = dgram.createSocket('udp4')
udpServer.bind(3131) // CONFLICT!
```

After:

```
// NEW: Use centralized client (only ONE UDP socket)
const atlasClient = await getAtlasClient(processor.id, {
  ipAddress: processor.ipAddress,
  tcpPort: processor.port || 5321,
  udpPort: processor.udpPort || 3131
})

// Register callback for updates
atlasClient.addUpdateCallback(async (processorId, param, value, fullParams) => {
  await handleMeterUpdate(processorId, { param, val: value, ...fullParams })
})

// Subscribe to parameters
await atlasClient.subscribe(inputMeter.parameterName, 'val')
```

Testing & Verification

Expected Behavior After Fix

1. ✅ No `EADDRINUSE` errors when loading Audio Control Center
2. ✅ Only ONE UDP socket bound to port 3131 per Atlas processor
3. ✅ Meter updates still received and processed correctly
4. ✅ Multiple API calls reuse the same Atlas client
5. ✅ Automatic cleanup of unused connections

How to Test

1. Start the application
2. Navigate to Audio Control Center page
3. Check console/logs for:
   - No `EADDRINUSE` errors
   - Log: "Creating new Atlas client" (only once per processor)
   - Log: "Reusing existing Atlas client" (for subsequent requests)
4. Verify meter updates are received and displayed
5. Leave page idle for 10+ minutes, verify client cleanup

Debugging Commands

```
# Check if port 3131 is in use
lsof -i :3131

# Check active Atlas clients
# (Add debug endpoint to expose atlasClientManager.getActiveClients())
```

Benefits of This Fix

1. Eliminates Port Conflicts ✅

- Only ONE UDP socket per processor
- No more `EADDRINUSE` errors
- Proper resource management

2. Better Performance 🚀

- Connection reuse reduces overhead
- Single UDP socket handles all meter updates
- No redundant TCP connections

3. Improved Reliability 💪

- Automatic cleanup prevents resource leaks
- Reference counting ensures clients aren't prematurely closed
- Centralized error handling

4. Easier Maintenance 🛠️

- All Atlas connection logic in one place
- Clear separation of concerns
- Easier to debug and monitor

5. Scalability 📈

- Can handle multiple components using same processor
- Automatic cleanup prevents resource exhaustion
- Ready for WebSocket real-time updates

Migration Notes

Breaking Changes

None - This is a transparent fix. All existing API interfaces remain the same.

New Exports

```
// From src/lib/atlas-client-manager.ts
export { getAtlasClient, releaseAtlasClient, disconnectAtlasClient }
export { atlasClientManager }
export type { MeterUpdateCallback }
```

Deprecated (Not Removed)

- `src/lib/atlasControlService.ts` - Old implementation (not currently used)

Recommendations for Future

1. Add WebSocket Support for Real-Time Updates

Instead of polling the database, push meter updates to frontend via WebSockets:

```
// Register WebSocket callback
atlasClient.addUpdateCallback(async (processorId, param, value) => {
  // Broadcast to all connected WebSocket clients
  wsServer.broadcast({ processorId, param, value })
})
```

2. Add Health Check Endpoint

Create API endpoint to monitor active Atlas clients:

```
GET /api/audio-processor/health
Response: {
  activeClients: [
    { key: "192.168.1.101:5321", processorId: "xyz", refCount: 2 }
  ]
}
```

3. Add Metrics/Monitoring

Track:
- Number of active Atlas connections
- UDP packets received per second
- Connection errors and retries
- Meter update latency

4. Consider Connection Pooling

For very high-traffic scenarios, implement connection pooling with max pool size.

5. Add Graceful Shutdown

Ensure all Atlas clients are properly disconnected on application shutdown:

```
process.on('SIGTERM', () => {
  atlasClientManager.shutdown()
})
```

Files Changed

New Files

- ✅ `src/lib/atlas-client-manager.ts` - Centralized Atlas client management

Modified Files

- ✅ `src/app/api/audio-processor/input-levels/route.ts` - Removed duplicate UDP server

Unchanged (Still Valid)

- `src/lib/atlasClient.ts` - Core Atlas TCP/UDP client
- `src/config/atlasConfig.ts` - Configuration constants
- `src/lib/atlas-logger.ts` - Logging utilities

Summary

This fix **eliminates the UDP port 3131 conflict** by implementing a **centralized singleton pattern** for Atlas client management. All components now share a single UDP socket per processor, preventing `EADDRINUSE` errors while improving performance, reliability, and maintainability.

Key Takeaway

One UDP socket, many consumers - The client manager acts as a multiplexer, allowing multiple API routes and components to receive meter updates through a single UDP connection.

Fix Implemented By: DeepAgent (Abacus.AI)
Date: October 21, 2025
Status: ✅ Complete and Ready for Testing