Advanced Visualization System for Monitoring the ATLAS TDAQ Network in Real-Time

S. Batraneanu, D. Campora, B. Martin, D. Savu, S. Stancu, L. Leahu
Outline

Introduction
- ATLAS trigger and data acquisition (TDAQ) system
- Monitoring the ATLAS TDAQ networks

Advanced visualization system for the TDAQ networks
- Design
  - Motivation, requirements and challenges
  - Hierarchical 3D model and its layout rules
- Implementation
  - Client architecture
  - Performance optimizations
  - Real-time update
  - Interaction mechanisms

Conclusions and future work
ATLAS Trigger and Data Acquisition System (TDAQ) (Level-1)

- Rejection factor: $10^6$
- Outstanding accuracy and efficiency

- 3 networks
  - DataCollection (Level-2)
  - BackEnd (Level-3)
  - Control
  - 6 chassis routers, 200 edge switches
  - 7000 interfaces, 2000 nodes

- Demanding performance requirements
  - Packet loss close to zero in the DC network $\rightarrow$ link load $<60\%$
  - Minimal end-to-end latency
Monitoring software framework

- **Net-IS**: Timeseries plots, sFlow stats, Plot aggregation
- **Net-RT**: Tabular data, Traffic-aware map
- **3DNetViz**: 
- **OneClick**: 
- **CentralizedDB**: Topology, Device description, Real-time statistics, Historical statistics (RRD)
- **SYNC & CROSS-CHECKS**: 
- **RW Nagios DCS**: External Data sources
- **SNMP Poller (Apoll)**
- **Topology Discovery (NetDiscovery)**
- **sFlow Engine (NetsFlow)**

THE ATLAS NETWORK
Advanced visualization system - requirements and challenges -

An effective visualization system should:
- Be intuitive
- Follow the system’s architecture and data flow
- Display the different types of monitoring data in real-time
- Offer the right level of detail
- Provide clear indications regarding the problem

Main implementation challenges
- Large scale system and overlapping networks
- Large variable space
- Real time update (30 seconds)
- Operation on multiple OSes: Windows, Linux
## 2D vs. 3D Visualization

<table>
<thead>
<tr>
<th>2D Visualization</th>
<th>3D Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Relatively <strong>inexpensive</strong> in terms of resources and setup</td>
<td>- <strong>Demanding</strong> in terms of processing power, configuration</td>
</tr>
<tr>
<td>- <strong>Visual clutter</strong> for overlapping networks</td>
<td>- Offers <strong>additional dimension</strong> -&gt; better candidate for large scale complex models</td>
</tr>
<tr>
<td>- <strong>Two</strong> degrees of freedom and restricted navigation paradigms</td>
<td>- <strong>Six</strong> degrees of freedom and natural navigation paradigms (walk, fly)</td>
</tr>
<tr>
<td></td>
<td>- <strong>Camera-object distance</strong> can be evaluated and used for levels-of-detail and object update</td>
</tr>
</tbody>
</table>
Hierarchical 3D Model

Overall status -> Color

Network affiliation -> Color

Device type -> Shape

Traffic quantity -> Size

Traffic status -> Color

Data network type -> Saturation

Processing farm

Top level view

Network core

Aggregated traffic panels
Optimized object layout

Top level view

- Follows data flow
- Control network as a backplane

Panoramic rack view
OSG implementation - overview

OpenSceneGraph (OSG)

- Open source, portable, high-performance framework based on C++ and OpenGL
- Rigorous structure based on STL and design patterns (visitor, callback)

Why did we choose OSG?

- Thin wrapper on top of OpenGL -> access to low-level rendering options
- Rendering statistics display and API -> essential for performance tuning
- Bit masks for selection and specialized event handlers for interaction

Used the scene creation to perform several optimizations

- Frame rate >30fps
  - Minimize overall traversal time: UPDATE+CULL+DRAW
  - Adjust LOD ranges

Introduced a targeted update mechanism to minimize the impact of real-time updates on rendering
Client architecture
Performance optimizations

Rendering optimizations

- Improve DRAW traversal time
- Geometry rendering -> best solution was to use vertex arrays + triangle primitives + color binding per vertex -> 14% decrease
- Custom geometry nodes optimized for fast rendering -> 15% decrease
- Text rendering -> ~75% decrease
- Low resolution object versions to use in Level Of Detail

Scene graph restructuring

- Improves CULL traversal time
- Eliminated Transform nodes at the panel level-> ~66% decrease
- LOD node rearrangement for flexibility
Real-time update

Based on visibility and proximity

New targeted update mechanism based on temporal coherence

Tested different granularities
  - Individual node: <30fps
  - Device node: 26% increase in fps
  - Rack node: 38% increase in fps

Decreased maximum completion time by spreading the updates over multiple frames

Used update and cull callbacks to update in the current frame the objects which were visible in the previous frame
Interaction mechanisms

Intuitive interaction mechanisms allowing access to the most detailed views in only a few clicks

Mixing free and guided navigation
- Guided between levels
- Free on the same level

Context aware navigation
- Based on layout parameters
- Different navigation paradigm
- Radial navigation
- Field of view and speed control

Details-On-Demand in a Head-Up Display
Conclusions and future work

- Identified specific visualization requirements
- Chose 3D visualization and information visualization guidelines
- Used open-source low-level framework OSG
- Intuitive interaction and navigation
- Frame rate > 30fps

Future work

- Integration of data taking parameters
- Rule-based expert system to improve error propagation rules
- Multiple views