Web-based Streamed Waveform Display using MDSplus events and Node.js

G. Manduchi, G. Moro, A.Luchetta, C. Taliercio, A. Rigoni
Consorzio RFX, Padova, Italy

Corresponding author: Gabriele Manduchi gabriele.manduchi@igi.cnr.it

12th IAEA Technical Meeting on Control, Data Acquisition and Remote Participation for Fusion Research
Daejeon, Republic of Korea (South Korea)

Abstract *- Streamed data visualization is a new requirement for long lasting discharges and more in general for every long lasting related experiment, such as the ITER Neutral Beam test facility. Implementing streamed data visualization, such as strip charts, would overload the data system, especially if a large number of charts are being displayed. A different approach for streamed data visualization is proposed here, using* ***MDSplus events****, rather than directly accessing stored data. Events are implemented as UDP multicast packets and can bring data. Data carried by events are made available to Web applications by means of a* ***Node.js server****, listening for the UDP packets and updating the connected Web clients using HTML5 Server-Sent Events. In turn, Web client will update the displayed waveforms using* ***plotly js****.*

1 - Introduction

- Streamed Data Visualization is a new requirement for ***long lasting*** discharges
  - It complements traditional waveform visualization and is mainly used in control room
- Providing streamed visualization requires accessing the data and may overload the data system for a large number of visualizations.
- On the other hand, streamed visualization often requires ***a subset of the acquired data***, due to screen resolution.
- A new approach is proposed here, based on ***MDSplus events*** implemented as UDP datagrams.
- Data are not read online from pulse files, but it are ***pushed*** by the data acquisition programs that will also perform subsampling when needed.
- As data producers are not aware of the listener clients, a ***publish-subscribe*** approach is adopted: MDSplus events tagged with appropriate name will be generated by the data acquisition program that generates data candidate for streamed visualization. Listeners (i.e. Web clients carrying out streaming visualization) will register for the signals of interest and will be notified whenever new data are available.
- As visualization is carried out by a Web interface, direct management of UDP communication from Web browsers is not desirable. For this reason, a new actor converting UDP messages into **HTTP Server Sent** messages is introduced.
- This actor is implemented as a **Node.js** application.

2 – MDSplus Events for Streaming

- MDSplus events are normally used to signal asynchronous events related to data acquisition, such as the availability of new stored signals.
- The event architecture is based on a ***publish-subscribe design pattern***. A name is associated with events, and the listener will register for events with that name.
- Events are implemented as UDP datagrams, using ***native UDP multicasting*** in order to provide fast and lightweight implementation.
- MDSplus events can also ***bring data***, whose total size is limited by the UDP datagram maximum dimension (64KBytes). No assumption on the underlying event-related data format is made by MDSplus.
- Data for streamed visualization is encoded in textual format:
  - ***Signal Name***: unique name for that signal
  - ***Number of Samples***
  - ***Times***: associated time to sample(s). It can be either a relative or absolute time
  - ***Samples***: actual sample values

3 – The Node.js Server

- Streamed data visualization will be carried out by Web clients.
- MDSplus events will not be directly received by Web browser
  - UDP datagrams reception is not feasible
- A new actor will listen MDSplus events carrying out streamed data.
  - It will register for MDSPlus events named STREAMING
  - It will implement a new publish-subscribe interface based on the Signal Name
- ***Node js***, an asynchronous event driven JavaScript runtime, is best suited for the actor implementation.
- It is an open-source, cross-platform run-time environment that executes JavaScript code outside of a browser. Scripts are executed server-side to produce dynamic web page content before the page is sent to the user's web browser.
- A wide user community has produced a large number of packages for Node js, greatly simplifying development of new applications.
- Here, node js will implement a UDP server for listening at MDSplus events and a HTTP server using the ***express*** package.

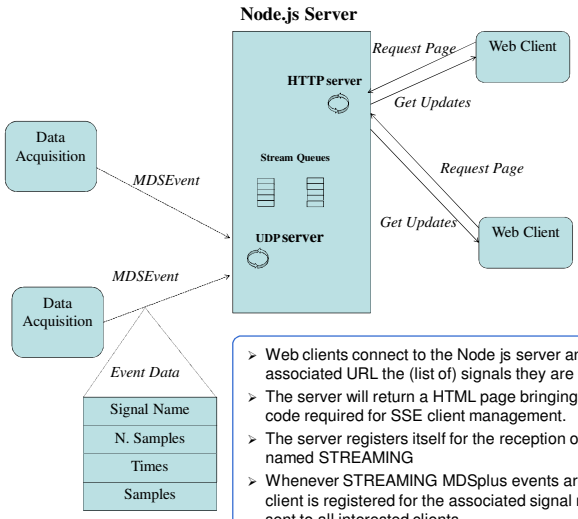
4 – Server-Sent Events

- UDP datagrams cannot be directly received by Web browsers
- TCP/IP communication is feasible via ***WebSockets***, but firewalls may block TCP/IP protocol
- HTTP based asynchronous data notification is preferable and it is provided by ***HTML5 Server-Sent Events***
- ***Server-Sent Events (SSE)*** allow a web page to get updates from a server. Unlike Ajax, where a request is first issued by the client, data are pushed directly from the server.
- Client side SSE management is provided by most browsers, including Chrome, Firefox and Safari.
- Server side SSE implementation is available for Node js via the ***sse package***.

Disclaimer

The work leading to this publication has been funded partially by Fusion for Energy under the contract F4E-OFC-280. This publication reflects the views only of the authors, and Fusion for Energy cannot be held responsible for any use which may be made of the information contained therein. The views and opinions expressed herein do not necessarily reflect those of the ITER Organization.

5 - Architecture



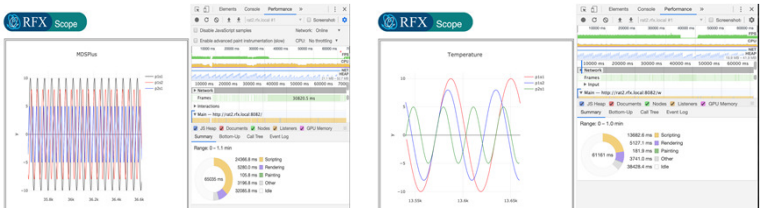
- Web clients connect to the Node js server and specify in the associated URL the (list of) signals they are interested in.
- The server will return a HTML page bringing the JavaScript code required for SSE client management.
- The server registers itself for the reception of MDSplus events named STREAMING
- Whenever STREAMING MDSplus events are received, if any client is registered for the associated signal name, data are sent to all interested clients
- Data history is internally maintained so that no data are lost upon client re-connection

6 – VUE JS and PLOTLY JS

- Implementation of the visualization page relies on **Vue.js** and **plotly.js** for building interfaces and displaying graphs.
- ***Vue.js*** is an open-source JavaScript framework for building user interfaces and single-page application.
  - It uses an HTML-based template syntax that allows binding the rendered DOM to the underlying Vue instance's data.
- ***plotly.js*** is a high-level, declarative JavaScript charting library, built on top of d3.js and stack.gl
  - It includes over 40 chart types, including scientific charts, 3D graphs, statistical charts, SVG maps and financial charts.

6 – System Load

- The whole implementation is lightweight for the following reasons:
  1) Data are pushed by Data Acquisition programs that will reduce the amount of sent data, as a refresh rate of 10-20 Hz is normally enough for visualization;
  2) UDP implementation of MDSplus events is extremely efficient since it relies on native UDP multicast;
  3) The system load to the the Node.js server is almost negligible even for a large number of streamed signals, being the system entirely event driven;
- Visualizing streamed waveforms requires rendering the streamed data. Even in this case, plotly proved extremely efficient as shown in the figure below, where the system was loaded by streaming three signals with an update rate of 1 kHz



System load at the client side for three signals with 1 kHz update rate, showing an history of 1000 samples (left) and 120 samples(right). In practice an update frequency up to 20 Hz is enough for a fluid visualization.

7 - Conclusion

Compared with other approaches for streamed data visualization, the presented one offers several advantages. Firstly, MDSplus events represent a much lighter solution in respect to repeatedly accessing stored data in pulse files. Then, Node.js proved a very effective environment for originating the Web pages and to implement the bridge between MDSplus events and HTML5 Server Sent events resulting in an amazingly low number of lines of JavaScript code. Finally, plotly js proved an effective tool for waveform display in Web pages, with a stunning performance in animation.

The tool is being installed in the overhead display of the control room of the ITER Neutral Beam Test Facility (NBTF)