jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2018, VOL. 64, NO. 3, PP. 329–334
Manuscript received May 8, 2018; revised July, 2018. DOI: 10.24425/123526

Video Streaming to Empowered Video Walls

Sven Ubik, Jiri Melnikov, and Zdeněk Trávníček

***Abstract*—Video walls are useful to display large size video content. Empowered video walls combine display functionality with computing power. Such video walls can display large scientific visualizations. If they can also display high-resolution video streamed over a network, they could enable distance collaboration over scientific data.**

We proposed several methods of network streaming of high-resolution video content to a major type of empowered video walls, which is the SAGE2 system. For all methods, we evaluated their performance and discussed their scalability and properties. The results should be applicable to other web-based empowered video walls as well.

***Keywords*—Video wall, Video streaming, SAGE2, WebRTC, High-definition video, Distance learning, Distance collaboration, Scientific visualizations**

I. Introduction

VIDEO walls built from LCD or LED panels are increasingly used to display large size video content. A high-resolution and a decreasing cost of thin-frame LCD panels makes them a popular choice for video walls. LED panels allow even completely seamless surface and may replace LCD panels in the future even for scientific use, if their resolution increases and the cost decreases.

We can see video walls in places such as shopping malls or airports, where they present video content played out in a loop. They are also used in control centers to display a state of some industrial system and a set of camera views. A resolution of such video walls is limited by a hardware controller used, regardless of the number of LCD or LED panels.

There is another type of video walls that we call *empowered video walls*, where each LCD panel is connected to an output of a powerful video graphical adapter with GPU circuits. Such video walls are usually installed in research laboratories or in university classrooms. An empowered video wall differs in several key aspects. First, it keeps a full resolution of each LCD panel, therefore a total resolution of the video wall can scale arbitrarily with the number of LCD panels used. Second, it can display content generated in real time, such as scientific visualizations or output of multiple running software applications. Finally, empowered video walls are usually connected to the Internet and can communicate with remote applications.

A suitable software control system is required to provide such functionality, particularly sharing a video wall for multiple concurrent applications or visualizations. Many video walls use the Scalable Amplified Group Environment (SAGE2) [1]. It is a web-based system allowing easy programming of applications with visual outputs that are scalable across video walls of arbitrary sizes and resolutions. SAGE2 is now installed in tens of laboratories around the world.

Video streaming to an empowered wall can be useful for distance collaboration scenarios. It can be used for video conferences or to stream remotely rendered visualizations in real time for discussions or distance processing among researchers, students or teachers. However, it is not straightforward to stream remote video content to a wall driven by a set of web browsers.

We implemented several techniques for streaming of high-resolution video to a SAGE2 video wall and evaluated their performance, advantages and limitations. We believe that the results are generally applicable to other empowered video walls with web-based software control systems.

II. Related work

We base our work on several previous works and existing technologies.

In SAGE2 [1], LCD panels are divided into groups, where each group (typically a vertical column) shows an output of one Chrome or Electron web browser, see Fig. 1. Applications run directly inside the web browsers using their Javascript engines. Multiple application instances exchange synchronization messages through a SAGE2 server. Based on the communication with the SAGE2 server, each application instance displays only its part of the whole application user interface. When compared to the original SAGE system [2], the new web-based architecture allows easier application programming and is well suited for applications like gigapixel image viewers.

Web Real-Time Communication (WebRTC) [3] is a collection of communications protocols and application programming interfaces (APIs) that enable real-time communication over peer-to-peer connections, typically for video applications. WebRTC is being standardized by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). Most modern web browsers implement WebRTC, including Google Chrome, Mozilla Firefox or Microsoft Edge. This makes WebRTC particularly useful for applications with web-based user interfaces and it is also an attractive option for the SAGE2 system, which itself uses web browsers for visual output.

Ultragrid [4] is open source software for video and audio network transmissions. It is available for Linux, Windows and Mac OS X platforms. Ultragrid can transmit video signals captured from various types of grabber cards. Several types of video compression are implemented, some of them accelerated using GPU. It has been used for transmissions of video signals in up to 8K resolution.

This work has been supported by the Ministry of Education, Youth and Sports of Czech Republic under the CESNET e-infrastructure Modernization project (CZ.02.1.01/0.0/0.0/16_013/0001797).

All authors are with the CESNET, Prague, Czech Republic (e-mail: ubik@cesnet.cz, melnikov@cesnet.cz).

Zdeněk Trávníček is with Czech Technical University, Prague, Czech Republic (e-mail: travnicek@iim.cz).

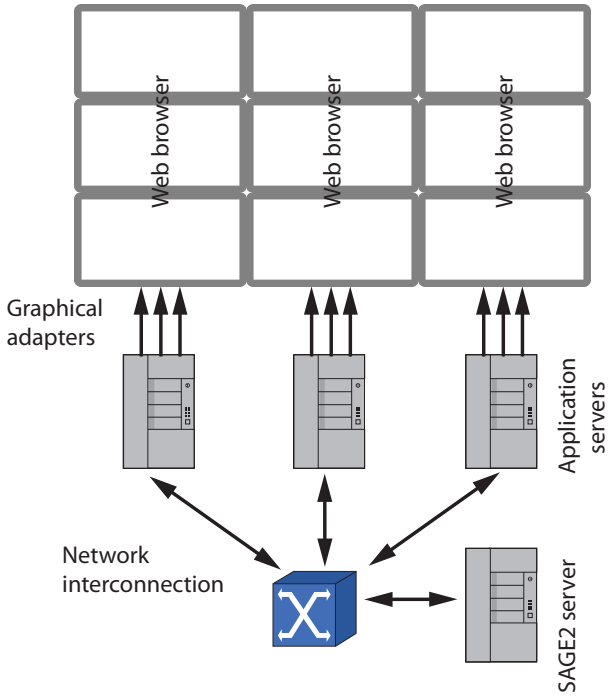

Fig. 1. SAGE2 empowered video wall

Libyuri [5] is a library of processing modules for video, audio and other data. Applications can be specified as directed graphs of modules in a data flow fashion. A particular application is created dynamically at startup by reading a module graph from an XML file or another source. The graph concept provides high scalability and ease of application creation with little programming. Libyuri can be linked with Ultragrid to create network applications.

A V4L2 Linux kernel driver [6] creates a virtual web camera in a Linux operating system. An application can insert real-time video content into a Linux device created by this driver. Other applications can then open this device as a regular web camera and read the video content from it. The webcam can be used with RGB or YUV uncompressed video.

A v4l2loopback [7] is another implementation of a virtual web camera for a Linux operating system. The advantage of this implementation is that it can be used also with compressed JPEG video.

GPUJPEG [8] is a compression and decompression library for JPEG images accelerated on GPU. This acceleration allows for compression and decompression of 4K images up to 125 fps. The library was developed as a part of the UltraGrid software project.

III. Design options

We implemented several methods of video streaming to a set of web browsers that form the display part of the SAGE2 system using WebRTC, JPEG images provided by a web server and the Ultragrid software. A window that displays the video stream can span multiple web browsers. Each method differs in a way the video content is delivered to individual web browsers. The window displaying the video, can be freely resized and moved across the video wall, except with the last method with video stripes.

In order to obtain reproducible results, we read prepared video content from files encoded in H.264. For live streaming from a camera using a grabber card, performance would be equal or higher, because we would save H.264 decoding on the sender side.

The number of frames displayed per second (fps) was counted in the Javascript code that ran inside the SAGE2 web browsers. An exception was the second configuration, with JPEG images retrieved from a web server, where the code did not allow to insert such instrumentation. In this configuration, we used a camera to capture the video wall and we counted the time code, which was inserted in the video content.

A. WebRTC

The configuration using WebRTC is shown in Fig. 2. An application based on libyuri reads content from a video file and inserts the content into a virtual web camera. A web browser started on the same PC opens a web page with `webrtc-send.js` Javascript application. This application uses WebRTC to connect to all SAGE2 web browsers and streams to them the content from the virtual web camera. Each remote web browser runs a `webrtc-receive.js` Javascript application, which displays a part of the original video frame selected based on the application window position and size on the video wall. Since the video content is being inserted into the virtual web camera in its original frame rate, if the WebRTC application cannot keep up, the frames are skipped and not displayed.

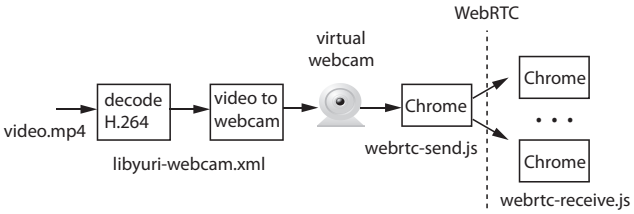

Fig. 2. Streaming by WebRTC

B. JPEG images

The system configuration using JPEG images is shown in Fig. 3. An application based on libyuri reads content from a video file, decodes H.264, splits the uncompressed video into individual frames, encodes these frames using the GPUJPEG library and provides the resulting JPEG images to a web server. Web browsers that drive the video wall open a `jpeg.js` Javascript application. This application runs in a loop, which repeatedly requests a JPEG image from a remote web server and displays the image. Since frames are taken from the input source on request by web browsers, if the Javascript application cannot keep up, the video playout is either slowed down for a file-based video source or scattered for a live (camera) video source.

C. Ultragrid with H.264 streaming

The system configuration using Ultragrid with H.264 streaming is shown in Fig. 4. An application based on libyuri

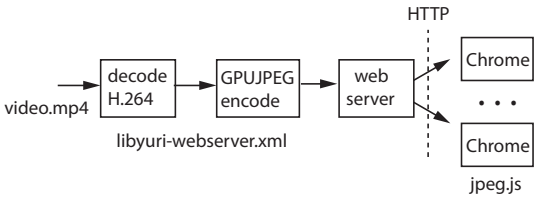

Fig. 3. Streaming by JPEG images

reads content from a video file, decodes H.264, encodes the video again with H.264 and uses Ultragrid software to stream the H.264 encoded video over a network to multiple SAGE2 PCs. The reason why the input video file is first decoded by the H.264 module is to keep H.264 encoding in a processing chain, because it would be needed for a real-time video source, such as a camera. Each SAGE2 PC runs again the Ultragrid software, which receives the video content and decodes H.264. A virtual web camera is created on each PC and an application based on libyuri linked with Ultragrid is used to put content into the virtual web camera. Finally, the web browser displays the content from the virtual web camera.

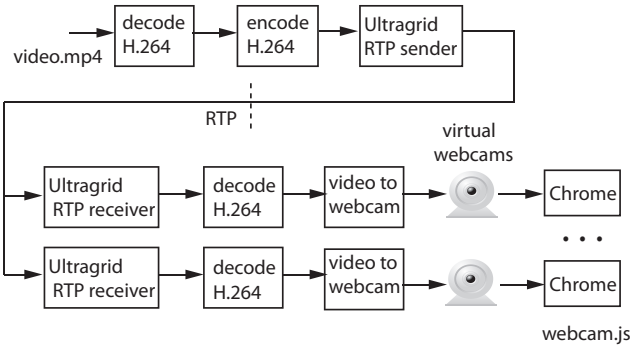

Fig. 4. Streaming by Ultragrid with H.264

D. Ultragrid with JPEG streaming and decoding in Chrome web browsers

The system configuration using Ultragrid with JPEG streaming and decoding in Chrome web browsers is shown in Fig. 5. An application based on libyuri reads content from a video file, decodes H.264 and uses Ultragrid software to encode video in JPEG using the GPUJPEG library and to stream the encoded video over a network to multiple SAGE2 PCs. Each SAGE2 PC runs again Ultragrid software, which receives the video content. A virtual web camera is created on each PC and an application based on libyuri linked with Ultragrid is used to put JPEG encoded video into the virtual web camera. Finally, the web browser reads video from the virtual web camera, decodes it using its own JPEG implementation and displays the decoded video in its window frame.

E. Ultragrid with JPEG streaming and decoding in GPUJPEG

The system configuration using Ultragrid with JPEG streaming and decoding in GPUJPEG is shown in Fig. 6. An application based on libyuri reads content from a video file,

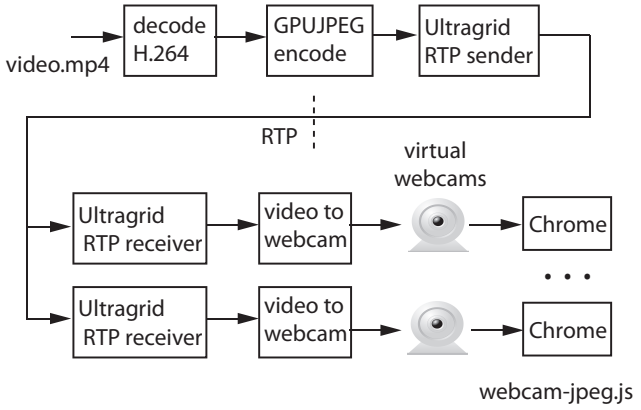

Fig. 5. Streaming by Ultragrid with JPEG and decoding in Chrome web browsers

decodes H.264 and uses Ultragrid software to encode video in JPEG using the GPUJPEG library and to stream the encoded video over a network to multiple SAGE2 PCs. Each SAGE2 PC runs again the Ultragrid software, which receives the video content and decodes it using the GPUJPEG library. A virtual web camera is created on each PC and an application based on libyuri linked with Ultragrid is used to put the uncompressed video into the virtual web camera. Finally, the web browser reads video from the virtual web camera and displays the video in its window frame.

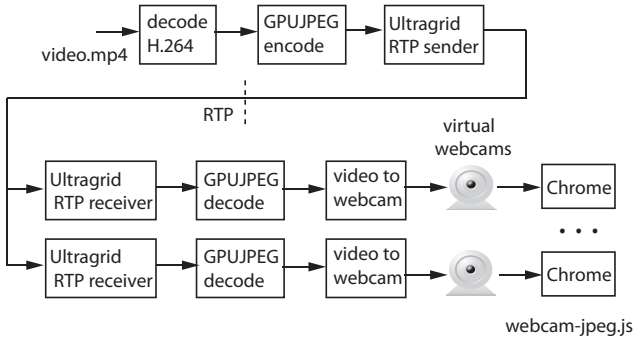

Fig. 6. Streaming by Ultragrid with JPEG and decoding in GPUJPEG

F. Ultragrid with H.264 streaming in stripes

The system configuration is similar to the case of Ultragrid with H.264 streaming, as shown in Fig. 7. However, instead of sending the whole video frame to each PC, we send only a cropped subframe corresponding to each PC's part of the application window. The advantage of this configuration is a lower load of SAGE2 PCs as well as a lower network bitrate. The disadvantage is the need for splitting the video content into subframes, which are typically vertical stripes of LCD panels. When the application window is moved or resized on the wall, the coordinates of subframes will change. Therefore, some synchronization mechanism is required between the transmitting applications and the SAGE2 server, which knows the coordinates of individual web browsers. We have not implemented such mechanism yet, therefore for this solution the video window has a fixed size and position on the video wall.

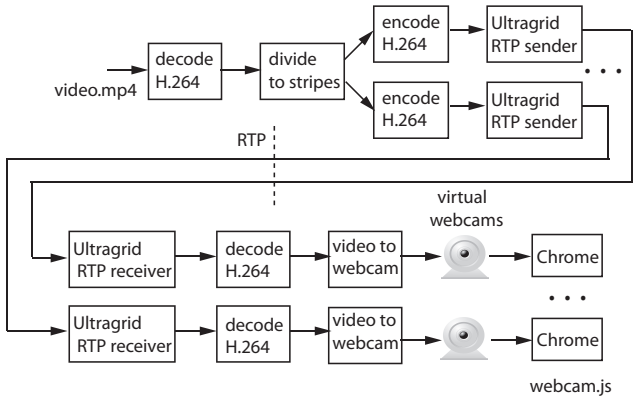

Fig. 7. Streaming by Ultragrid with stripes

IV. Performance evaluation and discussion

We assume that a user has a video wall driven by the SAGE2 system. The video wall may have any physical configuration, varying in the number of panel rows, columns and the resolution of individual LCD panels. The video content can be stored in a file or available from a live source (camera). The video content should be displayed in a window that can be freely moved and resized around the video wall.

We used three video content types for testing - a series of static images (slide show), a dynamic movie content and white noise. Nature photographs were used as static images and a section of the Tears of Steel movie was used for the dynamic scene. Each content type was prepared in three pixel resolutions of 720p (1280x720), 1080p (1920x1080) and 2160p (3840x2160). All content samples were stored in MP4 files with H.264 compression, 4:2:0 color subsampling. We aimed at playout at 30 fps and measured the achieved fps and CPU load, depending on the transmitted bitrate per second. We also subjectively observed the image quality, limiting our observation to finding the limits when the video still looked without visible frame losses and distortions. The SAGE2 web browsers were running on Dell Precision T620 PCs, one per a vertical stripe of monitors. Each PC had two 6-core Xeon E5-2640 CPUs running at 2.6 GHz. The same type of PC was used for the SAGE2 server that streamed the video content. The performance of streaming by different methods is shown Tab. I through Tab. VI.

WebRTC streaming (Tab. I) was unstable, resolution and frame rate changed during streaming. Generally, WebRTC tried to keep resolution at the expense of a lower frame rate. It also tried to keep low network bandwidth at the expense of image quality. With 2160p, frame rate dropped to 1. Overall, the image quality was subjectively poor, including a lot of compression artefacts.

Streaming by JPEG images (Tab. II) provided the full resolution of the original video content, but fps was reduced. CPU load depended strongly on the type of video content and its resolution. The noise content was difficult to compress, resulting in very high network bitrate up to 6 Gbps. At 2160p, the CPU load approached the limit of the 12-core PC. Interestingly, the network bitrate was higher for static images than for a dynamic scene. The reason was in that the JPEG compression of individual images could not utilize the inter-frame similarity of the dynamic scene and the static images included more details.

TABLE I
Performance of streaming by WebRTC

Content type	Static images	Dynamic scene	Noise
Format	resolution fps CPU load bitrate	resolution fps CPU load bitrate	resolution fps CPU load bitrate
720p	1280x720 px 13-16 3 10-12.5 Mbps	780x320 px 17 3.7-4 10-12.5 Mbps	1280x720 px 16 1 2.5 Mbps
1080p	1920x1080 px 13-16 4.5-6 11-16 Mbps	1920x1080 px 16 10-13 12.5 Mbps	0280x270 px 16 2.5 12.5 Mbps
2160p	2880x1620 px 15-16 3-6 10-12.5 Mbps	2880x1620 px 15-16 1 12.5-20 Mbps	1280x720 px 13-16 3 10-12.5 Mbps

Streaming by Ultragrid with H.264 encoding for network transmission (Tab. III) was different in that the network bitrate was specified by the user when starting the transmitting Ultragrid application. The network bitrate was low enough thanks to the H.264 compression. Therefore, we tried to set the maximum possible network bitrate to maximize the image quality, before the frame rate started to drop from the original 30 fps or before the image imperfections appeared. We started with 10 Mbps for each column of LCD panels and increased bitrate in 5 Mbps steps. With 2160p, however, the frame rate dropped to as low as 1 even with low bitrate.

Performance of streaming by Ultragrid with JPEG encoding for network transmission is summarized in Tab. IV for JPEG decoding in Chrome web browsers and in Tab. V for JPEG decoding in the GPUJPEG library. The use of the GPUJPEG library allowed to achieve a slightly higher fps at a slightly lower CPU load.

Streaming by Ultragrid with stripes (Tab. VI) allowed to achieve the full original frame rate of 30 fps with the exception of the noise content at 2160p when the image was of poor quality under 50 Mbps, but already started to lose frames at 50 Mbps. However, in all other cases the full resolution and frame rate with image quality without visible imperfections was achieved.

A comparison of achieved fps for the 2160p format based on the streaming method used and the video content streamed is shown in Fig. 8. A comparison of corresponding bitrates in shown in Fig. 9. We can see that the highest fps was achieved with Ultragrid with H.264 encoding and dividing the video content into stripes. At the same time, the network bitrate was significantly lower than with JPEG encoding. However, the solution with Ultragrid with JPEG streaming and decoding in GPUJPEG provided nearly the same fps and allowed free moving and resize of the video window on the wall.

Although WebRTC seems to be the nearest technology to the web-based architecture of the SAGE2 system, it achieved the lowest performance. With 2160p video content, it was

TABLE II
PERFORMANCE OF STREAMING BY JPEG IMAGES

Content type	Static images	Dynamic scene	Noise
Format	fps CPU load bitrate	fps CPU load bitrate	fps CPU load bitrate
720p	23 0.35 260 Mbps	20 0.37 75-78 Mbps	23 1.4 720 Mbps
1080p	21 0.7 550 Mbps	20 0.7 160 Mbps	21 3.3 1.6 Gbps
2160p	7 2.5 1.7-2 Gbps	11 2.6 480 Mbps	7 12 6 Gbps

TABLE III
PERFORMANCE OF STREAMING BY ULTRAGRID WITH H.264

Content type	Static images	Dynamic scene	Noise
Format	fps CPU load bitrate	fps CPU load bitrate	fps CPU load bitrate
720p	30 1 50 Mbps	30 1 50 Mbps	30 1.4 100 Mbps
1080p	30 1.6 75 Mbps	30 1.8 100 Mbps	30 4.5 40 Mbps
2160p	8-12 3.3 40	1 3.5 40	1 8.5-9 40

TABLE IV
PERFORMANCE OF STREAMING BY ULTRAGRID WITH JPEG AND DECODING IN CHROME WEB BROWSERS

Content type	Static images	Dynamic scene	Noise
Format	fps CPU load bitrate	fps CPU load bitrate	fps CPU load bitrate
720p	25 0.7 230 Mbps	30 0.9 152 Mbps	25 1.2 594 Mbps
1080p	25 0.9 504 Mbps	30 0.9 291 Mbps	21 1.3 1300 Mbps
2160p	18 1.7 1910 Mbps	26 1.8 851 Mbps	8 1.6 2850 Mbps

TABLE V
PERFORMANCE OF STREAMING BY ULTRAGRID WITH JPEG AND DECODING IN GPUJPEG

Content type	Static images	Dynamic scene	Noise
Format	fps CPU load bitrate	fps CPU load bitrate	fps CPU load bitrate
720p	25 0.6 231 Mbps	30 0.8 152 Mbps	25 1.1 593 Mbps
1080p	25 0.5 502 Mbps	30 0.6 291 Mbps	25 0.8 1290 Mbps
2160p	25 1.5 1920 Mbps	30 1.7 852 Mbps	12 1.7 2830 Mbps

TABLE VI
PERFORMANCE OF STREAMING BY ULTRAGRID WITH STRIPES

Content type	Static images	Dynamic scene	Noise
Format	fps CPU load bitrate	fps CPU load bitrate	fps CPU load bitrate
720p	30 0.9 100 Mbps	30 0.7 125 Mbps	30 0.9 100 Mbps
1080p	30 1.2 25 Mbps	30 1 100 Mbps	30 1.2 125 Mbps
2160p	30 2.5 50 Mbps	30 1.4 175 Mbps	30* 1.4 50

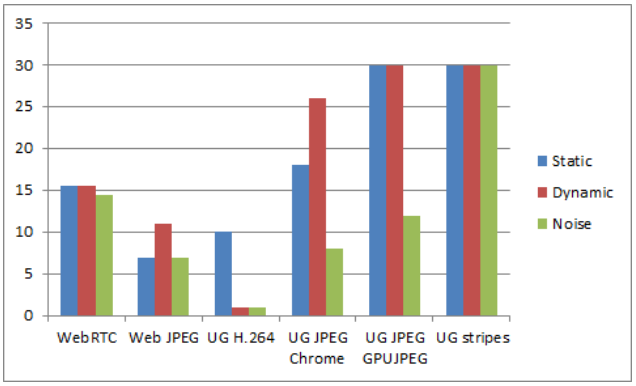

Fig. 8. Comparison of fps for 2160p

as low as approx. 1 fps. H.264 or VC9 compression used inside WebRTC could definitely run faster on the PC hardware used. The problem appears to be in an inefficient WebRTC implementation inside a Chrome web browser. Additionally, WebRTC often reduced resolution from the original all by itself, without any possibility to configure this behaviour inside the Chrome web browser. The network bitrate was also configured automatically by the Chrome web browser, keeping it under approx. 3 Mbps for 1080p video, resulting in poor image quality. We assume that WebRTC can still be used for a video conferencing application for the SAGE2 system, where lower resolution and possibly lower frame rate is acceptable.

The configuration of a web server and JPEG images is the simplest solution. We can note that the network bitrate was different than for Ultragrid with JPEG compression. The first reason was a slightly different compression ratio. The second reason was that the number of JPEG images transferred was limited by the client in the web server configuration and by the server in the Ultragrid configuration.

The Ultragrid configuration can significantly decrease the network bitrate using H.264 or JPEG compression. The original frame rate of 30 fps was kept until 1080p. However, the frame rate dropped significantly with 2160p content. The reason was a high CPU load on the transmitting PC, because full 2160p frames were transmitted to all SAGE2 PCs.

The final configuration with Ultragrid and video frames split into stripes provided 30fps up to 2160p, although the network bitrate had to be kept low at approx. 10 Mbps per stripe, before

visual distortions appeared in the image. We assume that this could be improved by implementation optimization.

An example of an image from the dynamic scene in a 2160p format streamed by Ultragrid with JPEG on a 5x4 video wall is shown in Fig. 10.

Fig. 10. Streaming of the dynamic scene at 2160p

V. Conclusion

We presented several methods of high-definition video streaming to a video wall driven by the web-based SAGE2 system and did performance comparison of these methods. Until the WebRTC implementation inside web browsers is optimized, significantly better results can be obtained using an external video streaming application such as Ultragrid and optionally splitting the video frames into subframes. For that case, we plan implementing synchronization with the SAGE2 server, allowing to keep the video frame intact when the application window is moved or resized.

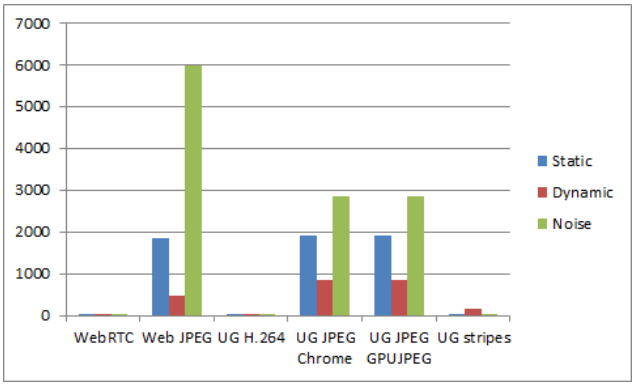

Fig. 9. Comparison of bitrate for 2160p

References

[1] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot, L. Long, A. Johnson, and J. Leigh,"SAGE2: A New Approach for Data Intensive Collaboration Using Scalable Resolution Shared Displays", *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2014.

[2] Byungil Jeong, Ratko Jagodic, Luc Renambot, Rajvikram Singh, Andrew Johnson, Jason Leigh, "Scalable Graphics Architecture for High-Resolution Displays", *The 4th Workshop on Advanced Collaborative Environments (WACE)*, Nice, France, 2004.

[3] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, Anant Narayanan, Bernard Aboba (eds.), "WebRTC 1.0: Real-time Communication Between Browser", W3C Working Draft, 24 November 2016.

[4] P. Holub, L. Matyska, M. Lika, L. Hejtmánek, J. Denemark, T. Rebok, A. Hutanu, R. Paruchuri, J. Radil, E. Hladk. "High-definition multimedia for multiparty low-latency interactive communication", *Future Generation Computer Systems*, Amsterdam, The Netherlands, Elsevier Science, Netherlands, ISSN 0167-739X, 2006, vol. 22, no. 8, pp. 856-861.

[5] Z. Trávníček, "Libyuri framework" (in Czech), report, CESNET and Czech Technical University, 2015, http://projects.iim.cz/yuri.

[6] J. Sebechlebský, "A Linux Kernel Module for Virtual Video Source", Bachelor Thesis, Faculty of Information Technologies, Czech Technical University, 2014, https://github.com/jsebechlebsky/vcdev.

[7] v4l2loopback - a kernel module to create V4L2 loopback devices, https://github.com/umlaeute/v4l2loopback.

[8] P. Holub, M. Łrom, M. Pulec, J. Matela, M. Jirman. "GPU-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2K, and 4K video". *Future Generation Computer Systems*, Amsterdam, The Netherlands, Vol. 29, No. 8, October 2013, pp. 1991-2006, https://doi.org/10.1016/j.future.2013.06.006.