International Journal of Advanced Smart Convergence Vol.14 No.4 1-14 (2025)
http://dx.doi.org/10.7236/IJASC.2025.14.4.1

IJASC 25-4-1

Design and Implementation of Multiplexer System for Display Sharing of Multiple Presenters on Meeting

Dawon Jeong, Yujin Seo, Eunbi Kim, Ahrin Jun, Danbi Yoon, Kitae Hwang*

Student, School of Computer Engineering, Hansung University, Korea
** Professor, School of Computer Engineering, Hansung University, Korea*

*dy030819@naver.com, sseo08267@naver.com, keunbi03@naver.com, wjsdkfls03@naver.com, yoondb1128@naver.com, calafk@hansung.ac.kr**

Abstract

Most everyday meetings today involve presenters connecting their computers to a shared TV or projector via HDMI during their turn. This requires the hassle of switching HDMI connections every time a presenter changes, and it also limits the ability to view multiple presenters' screens simultaneously. This paper proposes and implements a Multiplexer System that allows presenters to easily share displays. This system is a form of display sharing that connects the presenters' screens to a general-purpose multiplexer computer that is directly connected to a large display and is connected in real time. We used WebRTC to handle real-time transmission of screen images, and leveraged the hardware video decoder built into the multiplexer computer for fast decoding of encoded streams. To evaluate the performance of the Multiplexer System, we measured the screen latency required for the screen image of the presenter's computer to be displayed on the multiplexer's display. The result was evaluated to be around 221-362 ms when the number of simultaneous presenters is two, which is considered to be a level that does not cause any inconvenience to the presenters. We conclude that the Multiplexer System proposed in this paper is a system that conveniently supports everyday meetings without the use of special hardware or additional costs.

Keywords: *Display sharing, Meeting, WebRTC, GStreamer, MQTT*

1. INTRODUCTION

A typical meeting today involves presenters bringing their presentation materials to a laptop and connecting it to a large shared display, such as a large TV or projector, when it is their turn [1]. These simple meetings have the hassle of having to swap the HDMI cable from the laptop to the display every time a presenter changes, and the problem of not being able to see other presenters' materials together. Building a specialized meeting room with various functions would eliminate these problems, but the high cost of space and construction makes it a difficult problem for many organizations [2,3].

Manuscript Received: November. 3, 2025 / Revised: November. 11, 2025 / Accepted: November. 11, 2025
Corresponding Author: calafk@hansung.ac.kr
Tel: +82-2-760-4300, Fax: +82-2-760-5771
Professor, School of Computer Engineering, Hansung University, Korea

This paper presents and implements a method to facilitate everyday meetings where multiple presenters share a display with their own laptops containing their presentation materials. The system proposed in this paper is outlined in Figure 1. A computer, called a multiplexer, is dedicated to outputting images to the meeting display. It receives the computer screen images of presenters in real time and outputs them to the display. When the current presenter concludes his or her presentation, the multiplexer immediately outputs the next presenter's screen to the display without switching HDMI connections.

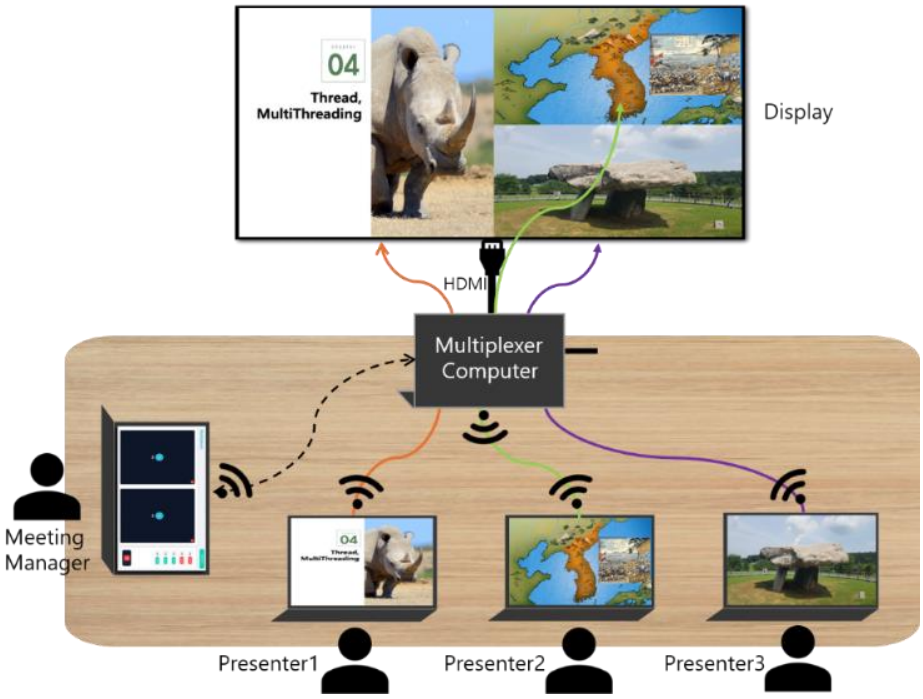

Figure 1. Overview of Multiplexer System

This research is not entirely new. To assess the feasibility of this idea in a previous study, our research team built a simple model that connects a presenter computer and a multiplexer computer in 1:1 and experimentally evaluated the specifications suitable for the multiplexer [4]. Previous research has concluded that computers commonly used today are suitable for use as multiplexers. This study, building on these findings, expands upon a system that simultaneously displays the screens of multiple presenters on a multiplexer display. A complete system with solutions for real-time transmission and rapid decoding of screen images has been designed and implemented.

This study proposes the following specific ideas for designing and implementing a multiplexer system. First, a wireless hotspot is set up on the multiplexer computer, allowing presenter computers to connect wirelessly to it. This system has the advantage of eliminating the need for separate wired infrastructure and easing constraints on meeting location. Because all presenter computers wirelessly connected to the multiplexer form a single wireless network, communication between the multiplexer and presenters is fast. It is also convenient because the IP of the multiplexer computer is fixed. Second, the shared display's screen can be split to display the screen images of up to four presenters simultaneously. Third, to eliminate device limitations of the multiplexer, the software for the multiplexer is implemented in a platform-independent manner. Fourth, WebRTC real-time streaming technology is utilized to shorten the time interval between the screen image

output from the presenter's laptop and the display on the multiplexer. WebRTC achieves fast real-time communication because it communicates in a peer-to-peer manner [5,6,7]. Fifth, the hardware decoder built into the multiplexer computer is utilized to quickly decode the stream of screen images received from the multiplexer [8]. Most general-purpose computers in use today have built-in hardware decoders. If a hardware decoder is not available, a software codec is automatically used.

To evaluate the performance of the implemented multiplexer system, this paper built two test systems and conducted experiments. The performance evaluation results showed that for two presenters, the average time it took for their screens to be displayed on the multiplexer display was around 221-362 ms. While not a short time, it is considered acceptable for simple meetings without the need for additional hardware or construction costs.

2. RELATED WORKS

Many researches have been conducted on effective meetings. Most research has focused on building large-scale meeting systems, such as installing dedicated conference tables and cameras in separate meeting rooms and using electronic whiteboards [1,2,3]. Microsoft, for example, developed the Distributed Meeting (DM) conferencing system [1]. DM is a complete conferencing system equipped with a 360-degree camera, whiteboard, meeting room server, and kiosk, enabling meeting scheduling, recording, broadcasting, and remote participation. However, this system is not feasible for most companies, schools, or research institutes due to the lack of dedicated space and cost.

Ramesh et al. also developed a remote conferencing system called eChronicle [2]. This system focuses on utilizing the various recordings generated during meetings, including text, audio, video, and other materials. However, this system is also expensive and its purpose is somewhat different from facilitating simple, everyday meetings.

Dedicated video conferencing platforms such as Zoom, Microsoft Teams, and Google Meet have recently become increasingly popular [9]. Video conferencing platforms primarily aim to facilitate remote meetings using cameras, eliminating the challenges of face-to-face meetings. They offer useful features, such as the ability to utilize various audio/video devices like cameras and microphones and the ability to save meeting records, making meetings much more seamless, even when geographically separated. However, most meetings we experience in our daily lives are face-to-face, with participants bringing presentation materials to their laptops and gathering in a location with a relatively large display, such as a large TV or beam projector, portable or not. Furthermore, meetings are conducted without any special arrangements. Video conferencing platforms differ from the smoothness and purpose of everyday meetings, which this study focused on. Furthermore, they have limitations in multi-person screen sharing, slow screen sharing, and poor resolution.

Recently, as a preliminary study for the development of the Multiplexer System targeted in this study, we conducted an experiment to determine the performance specifications of a multiplexer computer capable of receiving screen images from a user's computer in real time and displaying them on a display [4]. In this experiment, the presenter computer and the multiplexer computer were connected 1:1. And experiments were conducted on three types of NVIDIA embedded computers and three laptops with different operating systems, as the multiplexer computer. The video codec used was VP8 [10]. Experimental results showed that a system with at least 8 MB of memory and processing performance comparable to an Intel Core i7 was required. While this research is crude and lacks detail, it demonstrates that a multiplexer system can be sufficiently implemented on today's general-purpose desktop computers. Based on the results and experience of this

preliminary research, we designed and implemented a practical screen sharing meeting system in this study.

3. MULTIPLEXER SYSTEM

This section details the design and implementation of the Multiplexer System proposed in this paper.

3.1 Overview of Multiplexer System

The overall architecture of the Multiplexer System, including software, is shown in Figure 2. A wireless hotspot is turned on on the multiplexer computer, and presenter computers connect wirelessly to this hotspot, forming a single wireless network between the multiplexer and the presenters' computers. Alternatively, a wireless hotspot from someone's smartphone can be used. This eliminates the need for the presenter to know the multiplexer's IP address and allows direct communication between the presenter's computer and the multiplexer, thereby increasing communication speed.

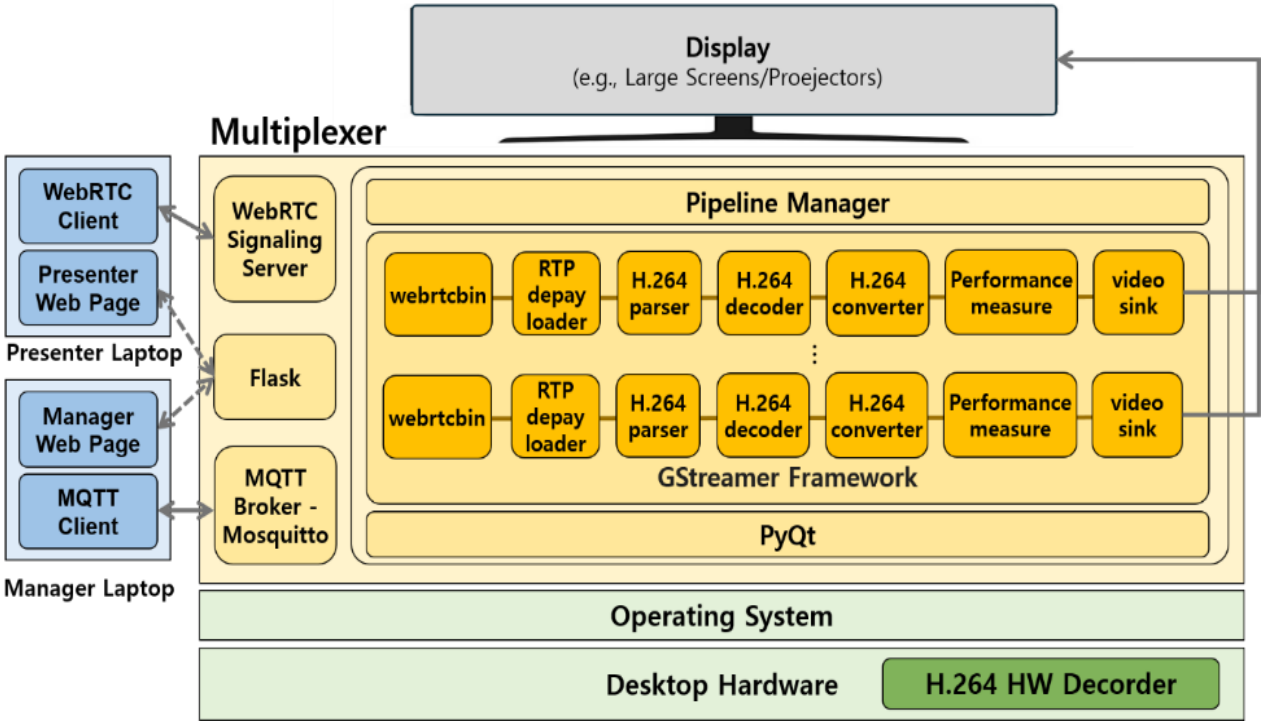

Figure 2. Multiplexer System Architecture

We built a web service based on the Flask framework [11] on the multiplexer to allow presenters to easily access the multiplexer via their web browser. Presenters access the multiplexer via the URL for them in a web browser, enter their name on the web page, and click the "Start" button. A WebRTC client written in JavaScript on the web page connects to the webrtcbin plugin and transmits screen images. The WebRTC protocol was used to transmit screen images between the presenter and the multiplexer.

To implement a pipeline that outputs screen images received from presenters to the display, we used the GStreamer framework [12]. An independent GStreamer pipeline is created for each presenter. This approach facilitates stream control for each presenter. For example, it's easy to switch presenters, simply remove the pipeline when the presentation is over, pause the pipeline to stop the presentation, and activate it to resume it.

We utilized the PyQT library to output screen images to the display.

We categorize meeting participants into non-presenters, presenters, and a manager. A manager, who host the meeting, manage presentations, including screen layout and the start and end of presentations for single or multiple presenters. The status of the presenter is divided into connection status and presentation status. Connection status means that the presenter is connected to the multiplexer and the corresponding pipeline is created, but no display output is going. When the manager places the presenter in the presentation status, the presenter's screen image begins to be displayed in real time on the multiplexer's display.

One of the presenters can become a manager. When the manager accesses the multiplexer using a URL for the manager in a web browser, he or she receives a manager web page. The web page receives a message from the multiplexer each time a presenter connects to the multiplexer, displaying a list of connected presenters to the manager.

The manager sends Pipeline Manager the layout information indicating where on the display some of the connected presenters that appear on his web page should be displayed, and Pipeline Manager activates the corresponding pipeline to display them on the display. We used the MQTT protocol to exchange these control messages [13,14]. We used Mosquitto [15] as the MQTT broker, and embedded code that acts as an MQTT client into the manager web page and the Pipeline Manager, respectively.

The Multiplexer System implemented in this paper limits the number of simultaneous presenters among the meeting attendees to a maximum of four. This means that the screens of up to four presenters can be displayed simultaneously on the multiplexer display. Figure 1 shows a case where three presenter screens are displayed simultaneously in their corresponding locations on the display when the manager arranges them on a multiplexer from the web page of the manager computer. The four layouts that the manager arrange where each presenter’s screen image is displayed will be shown in Figure 6 below. All software implemented on the multiplexer was written in Python, while the WebRTC client code running on the presenter’s web page and MQTT client code running on the manager’s web page were both written in JavaScript.

3.2 Real-time Streaming of screen images using WebRTC

WebRTC is a web standard communication method that utilizes the websocket and is a communication technology that can transmit high-quality media through P2P communication [5,6,7]. In this study, a WebRTC Signaling Server that relays the initial connection between two WebRTC clients is installed on a multiplexer computer, as shown in Figure 2.

When the WebRTC client embedded in the presenter's web page connects to the WebRTC Signaling Server, the WebRTC Signaling Server notifies the Pipeline Manager, which then creates an instance of the webrtcbin plugin. Webrtcbin establishes a peer-to-peer connection with the WebRTC client through the mediation of the WebRTC Signaling Server. Pipeline Manager then builds a GStreamer pipeline that handles the entire process of displaying the screen images received in real time, starting with the webrtcbin plugin.

3.3 Multi-streaming using GStreamer

The GStreamer is a framework for writing media streaming applications [12]. The GStreamer framework builds and manages a pipeline in which media data is processed by connecting modular plug-ins that perform various functions such as stream input, encoding/decoding, and media output to a display. We build pipelines that handle the entire process of receiving screen images and displaying them on the GStreamer framework, as shown in Figure 3. To facilitate stream management for each presenter, a separate pipeline was built for

each presenter. Each pipeline consists of six plugins, as shown in Table 1, and uses the H.264 standard for the video codec.

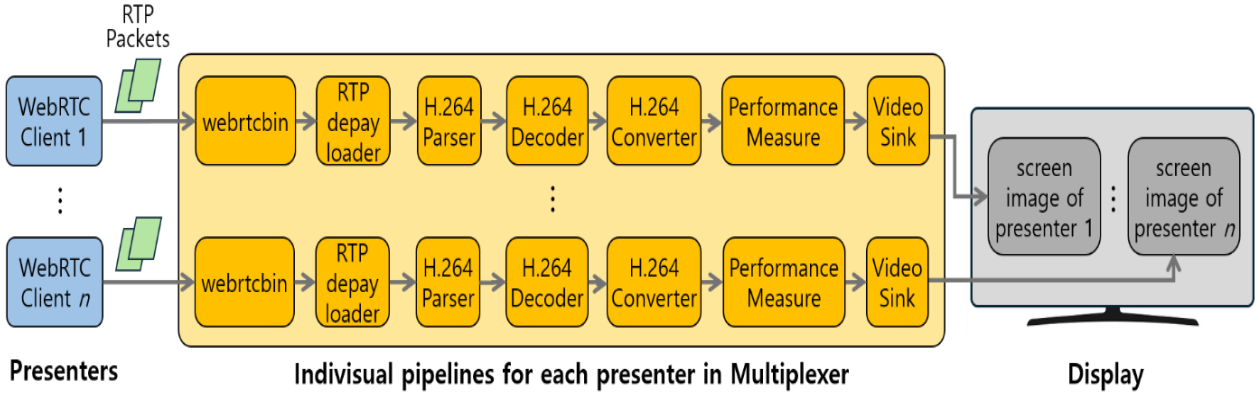

Figure 3. Multi-pipelines for multiple presenters

Table 1. Plugins in the pipeline

Plugins	Function
webrtcbin	WebRTC client. Receive screen images in real time from the presenter's WebRTC client.
RTP depay loader	Separate payload(the screen image) from the RTP packet
H,264 parser	Distinguish video frames and analyze them according to H.264 format
H.264 decoder	Decode the screen image
H.264 converter	Format into pixel form to be output on the display
Performance measure	Measure performance factors such as screen latency
Video sink	Render the decoded screen image to a window created with PyQT

3.4 H.264 Hardware codec

Most CPUs, including recent Intel CPUs, AMD CPUs, and Mac M3 chips, include built-in hardware decoders that support high-compression codecs such as H.264, H.265, and AV1. Because hardware decoders are faster than software decoders, this study uses a hardware decoder in the multiplexer pipeline to decode the compressed screen images received from the presenter's computer as quickly as possible.

Since AV1 is a cutting-edge compression technology, this study uses the H.264 codec, which is likely to be the most widely deployed codec today, to enhance the system's versatility. And we decided to compress and transmit screen images using the H.264 software codec on the presenter's computer. When building a pipeline on the GStreamer framework, if the current CPU has a hardware decoder, an H.264 Decoder plugin that utilizes it is installed. If not, a software H.264 decoder is installed as a plugin. In an era where H.265 or AV1 become mainstream, you can simply replace plugins in the pipeline at any time. This is why we use the GStreamer framework and pipeline.

3.5 Presentation control using MQTT

In this system, the manager and multiplexer exchange control messages in various situations during a

presentation. For example, if the presenter connects to the multiplexer or the presenter disconnects for any reason, the multiplexer's Pipeline Manager notifies the manager by sending the presenter's name. Additionally, when the manager selects a presenter among the connected presenters or determines the presenter's screen image location on the display, this information is passed to the multiplexer. This changes the pipeline of the multiplexer computer. To exchange messages between the manager's web page and Pipeline Manager, we established MQTT message communication, as shown in Figure 4. The MQTT broker was installed on the multiplexer using the open-source software Mosquitto, an MQTT client code was created in the manager's web page, and also a MQTT client code was added to Pipeline Manager. Table 2 shows the MQTT messages and Table 3 shows the JSON-formatted payload contained in the messages.

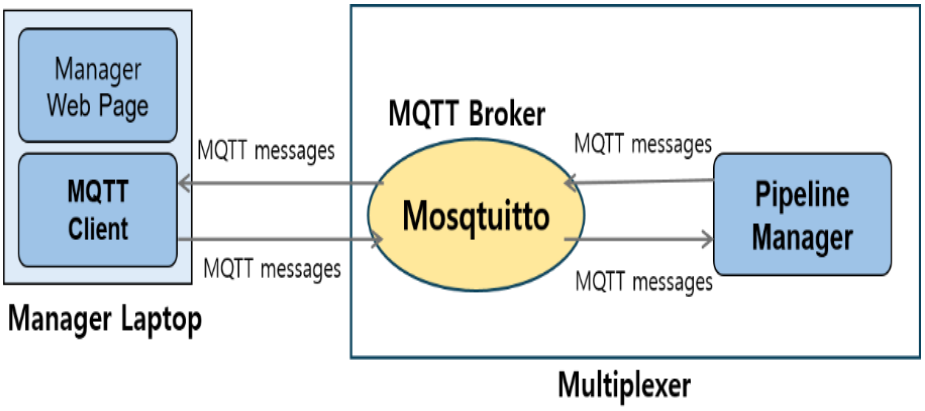

Figure 4. MQTT message communication

Table 2. MQTT Messages for Presentation Control

No	Message	Topic	Payload	Publisher	Subscriber
1	Req-list	“list/req”	empty	Manager	Multiplexer
2	Res-list	“list/res”	list of presenters’ names	Multiplexer	Manager
3	Update-list	“list/update”	list of presenters’ names	Manager	Multiplexer
4	Req-layout	“layout/req”	empty	Manager	Multiplexer
5	Res-layout	“layout/res”	layout information	Multiplexer	Manager
6	Update-layout	“layout/update”	layout information	Manager	Multiplexer

Table 3. Payloads of MQTT Messages

Payload in Message No.2 and No. 3	Payload in Message No.5 and No. 6

[
{"name": "Eunbi"},
{ "name": "Arin" }
]
	{
	"layout":2,
	"participants": [
	{"name": "Eunbi"},
	{ "name": "Arin" }]
	}

3.5.1 Presenter Connection

When a presenter enters a “name” on a web page and clicks the “connect button”, the WebRTC client code embedded in the web page connects to the WebRTC Signaling Server. The Pipeline Manager learns of the connection status from the WebRTC Signaling Server, configures a pipeline for the connected presenter, and then sends message No. 2 to the MQTT client embedded in the manager’s web page via Mosquitto. This MQTT client displays the presenter name included in the received message on the web page. If the manager is not connected, this message is not delivered and disappears.

3.5.2 Presenter List and Layout status

When the manager first connects to the multiplexer or reconnects, it sends message No. 1 to obtain a list of currently connected presenters. The Pipeline Manager responds with message No. 2 upon receiving this message. If the manager wishes to obtain information about the current display layout under certain circumstances, it sends message No. 4, and when the Pipeline Manager receives this message, it responds with message No. 5.

3.5.2 Screen Layout and Presentation End

In this study, only four types of layouts are supported as shown in the Figure 5. Figure 5(a) shows the layout for one presenter, Figure 5(b) for two, Figure 5(c) for three, and Figure 5(d) for four simultaneous presentations, respectively. The manager arranges presentations on the web page into one of the four types according to the presentation plan, and then selects a presenter for each area.

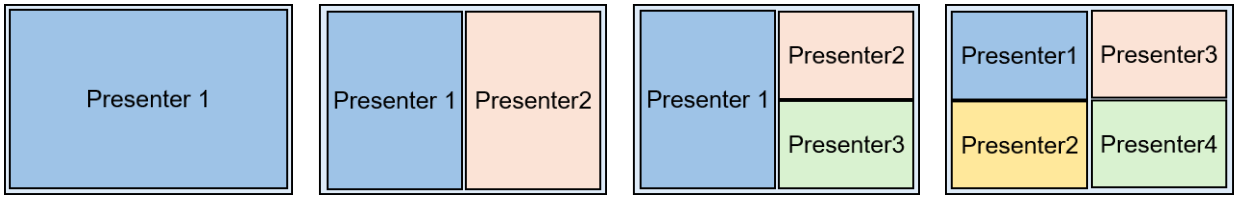

Figure 5. Four Types of Display Layouts

Then, the layout number and presenters’ names are created in JSON format as shown on the right side of Table 3 and sent to the Pipeline Manager in message No. 6. The same process is performed when the layout is changed or a presentation ends. If the layout remains the same but the presenter is changed, message No. 3 is sent. When the Pipeline Manager receives message No. 3 or No. 6, it controls the pipeline.

3.6 System Execution

Figure 6 shows the real appearance of the two test systems implemented in this study. The specifications of the two test systems can be found in Table 4. Figure 6(a) shows an HP laptop with Windows 11 used as a multiplexer and a 50-inch UHD display connected to it. Three presenter screens are displayed on the display. Figure 6(b) shows a MacBook as a multiplexer and a portable beam projector connected to it. Three presenters are displayed simultaneously. All computers used as presenters are the same HP laptop model used as the multiplexer in Figure 6(a).

(a) Implemented System 1 **(b) Implemented System 2**

Figure 6. Operational Samples of Implemented Systems

4. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the Multiplexer System implemented in this paper.

4.1 Test Systems

We implemented two systems as test systems for performance evaluation, as shown in Table 4, and their operation cases are already shown in Figure 6. These are all laptops with Wi-Fi capabilities. While system performance could be improved depending on the performance of the multiplexer computer, the goal of this study was to propose a system that can be easily used in everyday meetings. Therefore, a system utilizing a general-purpose laptop as the multiplexer was chosen for performance evaluation. The graphic resolution of the presenter computers and multiplexers used in the experiment is Full HD (1920x1280). All computers used as presenters are the same laptop model used as the multiplexer of Test System 1 in the Table 4.

Table 4. Specifications of Test Systems

	Test System 1	Test System 2
Multiplexer	HP 250 G10	MacBook
CPU	Intel Core I7	M3
Memory	32GB	16GB
H/W Codec	Quick Sync Video	Media Engine
OS	Windows 11	Mac OS

4.2 Performance Evaluation and Results

4.2.1 Performance Metrics and Workloads

In this paper, we set the screen latency as the most important performance indicator of the proposed and implemented system. Screen latency is the total time it takes for the screen image of the presenter's computer to be displayed on the multiplexer's display. This time is the time it takes for the screen image of the presenter's computer to be captured and encoded, transmitted to the multiplexer over the network, received by webrtcbin,

passed through the pipeline of the GStreamer framework, and then output by the Video sink plugin.

The workload used for performance evaluation is shown in Table 5. Since a human presentation would have been unreasonable and would have extended the experimental time, a method of playing recorded video on the presenter's computer was used. Since this study targeted typical meetings, the experiment was conducted using two types of video: static screen video with little change between frames, and dynamic screen video with significant change between frames.

Table 5. Workloads

Item	Type	Value
Video Type 1	Static Screen Video	Recorded Lecture Video
Video Type 2	Dynamic Screen Video	Recorded Lecture Video
Experiment Time	Presentation Time	5 minutes
FPS	Frames per second	Max: 30fps, Ideal: 25fps
Screen Size	Full HD Level	1920x1080

For static screen videos, high compression rates are expected, resulting in shorter communication times. For dynamic screen videos, low compression rates are expected, resulting in longer communication times. Ten static screen videos were sampled from our university's "Operating System Online Lecture". The slide template remains the same, with only changes occurring during the lecture, such as underlining or using the mouse to write text for explanations. For the dynamic screen videos, ten videos from the "Korean History Online Lecture" on YouTube were used. In these videos, the slides are image-centric, and each slide has very different features.

4.2.2 Screen Latencies and Frame Drop Rate

The screen latency is measured as the difference between the time the presenter's computer captures the screen and the time the Video sink plugin on the multiplexer outputs it to the display. However, there were two problems with measuring this time. First, the two computers had different clocks. To address this issue, before the performance evaluation experiment, we ran a program on both computers that output the current time in milliseconds and recorded the screens of both computers side by side with a single camera. Then, we looked at the time displayed on the recorded video and calculated the difference in milliseconds by subtracting the presenter's clock from the multiplexer clock to obtain the difference T_{diff} between the two clocks.

The second issue is that the GStreamer framework restricts plugins from being inserted after the sink plugin, making it impossible to insert the Performance measure plugin, which calculates the screen latency, after the Video sink plugin. Therefore, we inserted the Performance measure plugin directly before the Video sink plugin.

We measure the time T_{start} just before capturing the screen on the presenter's computer, and record T_{start} in the Time Stamp field of the frame header when the WebRTC client transmits the screen image. The Performance measure plugin measures the current time T_{end} for the arrived frame and compares the current time T_{end} with the Time Stamp value (T_{start}) recorded in the frame header to calculate the screen latency $T_{screenlatency}$ as follows:

$$T_{screenlatency} = T_{end} - T_{start} - T_{diff} + \alpha \quad (1)$$

In Equation (1), α is the time it takes for a decoded frame to be displayed, minus the various overhead times for timing measurement. In fact, precisely measuring α is not easy, and since the purpose of this experiment was not to measure precise screen latency, we arbitrarily set α to 10 ms. Therefore, there may be very small error in our experimental results.

The experiment was conducted with a total of four presenters connected to the multiplexer, creating four pipelines. We averaged the results across 10 trials for each evaluation. Figure 7 shows the average screen latency for each case, with the number of simultaneous presenters ranging from 1 to 4, on two test systems. As expected, the screen latency for static screen videos was significantly shorter than that for dynamic screen videos. As the number of simultaneous presenters increases, the latency tends to increase. However, in most presentations, it's rare for three or more presenters to share their screens on a multiplexer display. In practice, the most common scenarios are single presenters or those viewing content from previous presenters. When two presenters were presenting simultaneously, the screen latency was measured to be approximately 221ms to 362ms. Based on our experience using the implemented system, we believe that this latency is not a significant amount of time that presenters experience discomfort during their presentations.

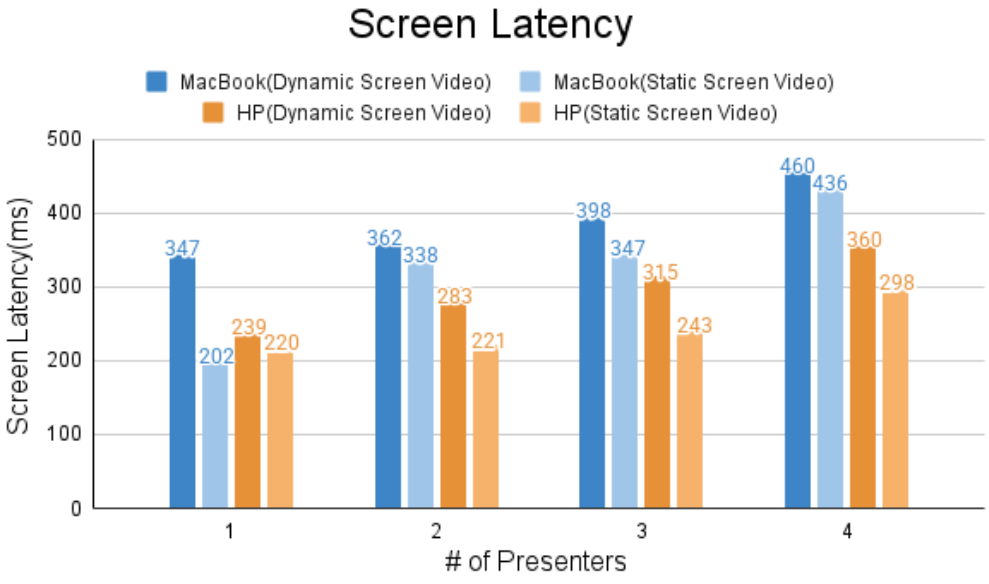

Figure 7. Results of Screen Delays

To understand other aspects related to screen latency, we additionally measured the average packet drop rate of WebRTC. WebRTC automatically drops packets to reduce load when packets become excessive and congested to the point where playback becomes problematic. Table 6 shows the packet drop rate observed during execution on the multiplexer.

Table 6. Frame Drop Rate(%)

# of Presenters	1	2	3	4
Static Screen Video	1.08	1.11	1.43	1.50
Dynamic Screen Video	5.16	5.22	5.47	5.96

Rather than interpreting the absolute meaning of the packet drop rate, we focus on understanding the phenomenon. As the number of simultaneous presenters increases, the packet drop rate increases because more packets reach the multiplexer. Furthermore, the packet drop rate for dynamic screen video was found to be approximately four times higher than that for static screen video. This is interpreted as a result of the lower compression ratio for dynamic screen video, placing a greater load on the multiplexer. Furthermore, even if 5 or 6 frames out of 100 are lost, there is little impact on viewing.

4.2.3 System Loads on Multiplexer

In addition to screen latency, we measured the load on the multiplexer during a meeting to assess the system's stability. Table 7 shows the results of measuring five factors: CPU Utilization, GPU Utilization, Memory Utilization, Overall Network Traffic, and Incoming Network Traffic on the multiplexer.

Table 7. System Load on Multiplexer

# of Presenters	CPU Util(%)	GPU Util(%)	Memory Util(%)	Overall Network Traffic(Mbps)	Incoming Network Traffic(Mbps)
1	4.47	7.30	20.05	0.009	0.903
2	6.51	10.04	20.05	0.015	1.661
3	8.12	14.42	23.12	0.023	2.125
4	9.80	18.81	25.97	0.035	2.510

As a result of this experiment, it was determined that the system is stable and unlikely to suddenly slow down or stop, as the utilization rate of overall resources such as CPU and memory is not high even in a high-load situation where four people are presenting simultaneously.

4.3 Cost Assessment

If you build a conference system that can easily switch HDMI cables or share the screens of multiple presenters on a single display. it is predicted that it will cost a lot of money to build a conference room, etc., although it is impossible to estimate the cost. However, the multiplexer system proposed in this paper utilizes a Wi-Fi-enabled desktop computer, such as a laptop, that most people own, as a multiplexer, so there are no costs involved.

5. CONCLUSION

This paper proposed and implemented a Multiplexer System that allows multiple presenters in a typical meeting to easily share a single display. The Multiplexer System consists of a single multiplexer computer directly connected to a large display, with presenters' screens connected in real time to share the display. The multiplexer computer, the core of the system proposed in this paper, can be any general-purpose desktop with Wi-Fi functionality. Meeting participants' laptops can also be used as multiplexers, making them cost-effective. Furthermore, WebRTC with a peer-to-peer protocol, is used for real-time transmission of screen images, and a hardware decoder built into the multiplexer computer is utilized for fast decoding. We implemented test systems that allow four presenters to simultaneously share a single multiplexer and output their presentations

to a display, and evaluated their performance. The performance evaluation results showed that the screen latency required for the screen images from the presenters' computers to be displayed on the multiplexer's display was approximately 221 ms to 362ms depending on presentation materials when two presenters were simultaneously presenting. Since this figure is judged to be at a level where presenters do not feel uncomfortable, we conclude that the Multiplexer System presented in this paper is a system that conveniently supports everyday meetings without the use of special hardware or additional costs.

Acknowledgement

This research was financially supported by the Hansung University

REFERENCES

[1] Ross Cutler, et. al, “Distributed meetings: a meeting capture and broadcasting system”, MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia, pp. 503-512, 2002. https://doi.org/10.1145/641007.641112

[2] Ramesh Jain, Pilho Kim, Zhao Li, “Experiential meeting system”, Proceedings of the 2003 ACM SIGMM Workshop on Experiential Telepresence. pp. 1-12, 2003. https://doi.org/10.1145/982484.982486

[3] Zhiwen Yu, Yuichi Nakamura, “Smart meeting systems: A survey of state-of-the-art and open issues”, ACM Computing Surveys (CSUR), vol. 42, issue 2, pp. 1-20, 2010. https://doi.org/10.1145/1667062.1667065

[4] Ahrin Jun, Donggeon Lee, Eunbi Kim, Danbi Yoon, Jaehyun Choi, Kitae Hwang, “Experimental Performance Evaluation for Real time Transmission of Computer Screens,“ The Journal of The Institute of Internet, Broadcasting and Communication (IIBC), vol. 25, no. 3, pp. 143-150, 2025. https://doi.org/10.7236/JIIBC.2025.25.3.143

[5] Bart Jansen, et. al., "Performance Evaluation of WebRTC-based Video Conferencing," ACM SIGMETRICS Performance Evaluation Review, vol. 45, issue 3, pp. 56 - 68, 2018. https://doi.org/10.1145/3199524.3199534

[6] B. Garcia, et. al., "WebRTC Testing: Challenges and Practical Solutions," in IEEE Communications Standards Magazine, vol. 1, no. 2, pp. 36-42, 2017. https://doi.org/10.1109/MCOMSTD.2017.1700005

[7] Jiyoon Ok, Juyeon Soung, Chaewon Park, Kitae Hwang, “Implementation Details of EPUB Reader using GraphRAG”, International Journal of Internet, Broadcasting and Communication, vol. 17, no. 2, pp. 223-231, 2025. https://doi.org/10.7236/IJIBC.2025.17.2.223

[8] R. Safin, et. al., "Hardware and software video encoding comparison," 2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), pp. 924-929, 2020. https://doi.org/10.23919/SICE48898.2020.9240439.

[9] A.M. Suduc, M. Bizoi, F.G. Filip, "Status, Challenges and Trends in Videoconferencing Platforms," International Journal of Computers Communications & Control, vol. 18, issue 3, 2023. https://doi.org/10.15837/ijccc.2023.3.5465

[10] Jim Bankoski, Paul Wilkins, Yaowu Xu, "Technical overview of VP8, an open source video codec for the web," 2011 IEEE International Conference on Multimedia and Expo, 2011, pp. 1-6. https://doi.org/10.1109/ICME.2011.6012227

[11] M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah and A. rokhim, "Design an MVC Model using Python for Flask Framework Development," International Electronics Symposium (IES), Surabaya, Indonesia, 2019, pp. 214-219. https://doi.org/10.1109/ELECSYM.2019.8901656.

[12] L. Wang, L. Zhang and Y. Ma, "Gstreamer accomplish video capture and coding with PyGI in Python language," First International Conference on Electronics Instrumentation & Information Systems (EIIS), Harbin, China, 2017, pp. 1-4, https://doi.org/10.1109/EIIS.2017.8298579

[13] Sukjun Hong, Jinkyu Kang and Soonchul Kwon, "Performance Comparison of HTTP, HTTPS, and MQTT for IoT Applications," International Journal of Advanced Smart Convergence, vol. 12, no. 1, pp. 9-17, 2023. https://doi.org/10.7236/IJASC.2023.12.1.9

[14] Tae-Hyung Kim and Young-Gon Kim, "Improvement of IoT sensor data loss rate of wireless network-based smart factory management system," International Journal of Advanced Smart Convergence, vol. 12, no. 2, pp. 173-181, 2023. https://doi.org/10.7236/IJASC.2023.12.2.173

[15] Kitae Hwang, Jae Moom Lee, In Hwan Jung and Dong Hee Lee, "Modification of Mosquitto Broker for Delivery of Urgent MQTT Message," IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan, pp. 166-167, 2019. https:/doi.org/10.1109/ECICE47484.2019.8942800