WebRTC 기반 애플리케이션 사례 연구

이지호, 예도훈, 김도근, 이상준, 한정희

한국항공대학교

enslvizhsk179@kau.kr, ayedh0421@gmail.com, gunks20011@kau.kr, will115@kau.kr, junghee@kau.ac.kr

Case study on WebRTC-based Applications

Lee Jiho, Aye Dohoon, Kim Dogeun, Lee Sangjun, Han Junghee

요 약

본 논문은 WebRTC 기반 애플리케이션의 성능을 실제 구현 사례를 통해 분석하는 사례 연구이다. WebRTC (Web Real-Time Communication)는 브라우저 기반 실시간 음성·영상 통신을 가능하게 하는 기술로, 최근 다양한 applications에서 사용되고 있다. 본 논문에서는, 이러한 webRTC 기반 응용프로그램의 전송 성능을 분석함으로써 문제점을 파악하는 것을 목표로 한다. 실험 대상은 open source 인 millo GitHub의 Typescript-ReactJS-WebRTC-1-1-P2P 이며, Mesh 형태로 피어 간 직접 연결을 구성한 뒤 참여자 수 증가 및 해상도 변화 조건에서 화면 전송 성능을 측정하였다. 이를 통해 Mesh 구조는 고해상도, 다자 환경에서 네트워크 혼잡 및 개별 단말 처리 부담에 의해 품질 저하가 발생할 수 있음을 확인하였다.

Ⅰ. 서 론

WebRTC(Web Real-Time Communication)는 브라우저 기반 실시간 음성·영상 통신을 가능하게 하는 기술이다. 다자 통신에서 WebRTC는 일반적으로 Mesh, SFU, MCU 구조로 구현되며, 이 중 Mesh는 서버가 미디어를 중계하지 않고 피어들이 서로 직접 연결하는 방식이라 구현이 비교적 단순하다. 하지만, 참여자 수가 증가하면 각 피어가 동시에 송·수신해야 하는 스트림 수가 늘어나, 네트워크 트래픽과 개별 단말의 부담이 커질 수 있다. 특히 저사양 환경에서는 “서버는 여유가 있는데 개별 클라이언트가 먼저 한계에 도달”하는 상황이 발생할 수 있다. 이러한 상황이 어떤 조건에서 나타나는지 확인하는 것은 WebRTC 기반 애플리케이션 설계에 중요한 기초가 된다.

본 논문은 GitHub에 공개된 millo-L의 1:N WebRTC 예제 애플리케이션을 대상으로, Mesh 기반 1:N 구조에서 참여자 수와 해상도 변화가 성능에 미치는 영향을 사례 연구 형태로 분석한다. 실험은 해상도 240*240과 1920*1080 조건을 각각 적용하고, 참여자 수를 증가시키면서 FPS, 서버/클라이언트 CPU 사용량, 그리고 WebRTC의 재전송 관련 지표를 측정하는 방식으로 진행하였다.

Ⅱ. 본론

WebRTC는 브라우저에서 실시간 음성·영상 통신을 구현하기 위한 핵심 기술로, 화상회의·원격수업 등 다양한 서비스에서 활용된다. 예를 들어, Google Meet는 클라이언트가 WebRTC로 Meet 서버와 실시간 미디어를 송·수신하는 구조로 동작한다. Microsoft Teams는 브라우저의 WebRTC 진단 로그를 활용해 통화 품질 및 연결 상태를 분석할 수 있도록 하며, Zoom은 웹 애플리케이션에서 WebRTC를 이용해 커스텀 비디오 기능을 구현하도록 지원한다.

WebRTC 는 구현방식에 따라, Mesh, Ring, SFU, MCU 의 구조를 가진다. SFU (Selective Forwarding Unit) 와 MCU (Multipoint Contro Unit)의 경우, 서버에서 비디오 스트리밍을 전달하는 기능을 수행해야 하므로, 서버의 부하가 크고 서버 운용에 따른 비용 부담이 있는 반면, 각 클라이언트 피어들의 부하가 적은 장점이 있다. 이와 반대로 Mesh 구조의 경우, 서버 부담은 적어지나, 각 클라이언트가 직접 연결을 수행하고 데이터를 송수신 해야 한다. Ring 구조의 경우, Mesh 와 유사하게 서버가 아닌 각 클라이언트가 송수신을 책임지는 방식이나, 각 클라이언트의 데이터가 모두 공유되는 것이 아닌, 하나의 노드의 데이터를 전달하는 것을 목적으로 한다. 이 중, Mesh와 Ring 구조는 서버 유지의 비용이 적다는 장점으로, Adhoc 네트워크 기반 데이터 전송에 활용되고 있다.

본 논문에서는, 이 중 Mesh 구조에서, 참여자 수 증가 시 단말 처리 부담과 네트워크 혼잡이 문제가 될 수 있는 제약점을 분석하고자 하다. 즉, 본 연구는 Mesh 기반 구조에서 해상도 및 참여자 수 변화에 따른 성능 저하 양상을 실측하여, 다자 환경에서의 한계를 정량적으로 사례 분석함으로써 문제점을 확인하고자 한다.

실험 대상 애플리케이션은 millo-L GitHub의 1:N WebRTC이며, 각 피어는 다수의 상대에게 RTCPeerConnection을 생성하는 방식으로 Mesh 연결을 구성한다. 본 연구에서는 참여자 수를 증가시키며 동일 애플리케이션 설정에서 해상도별 성능 변화를 관측하였다.

본 논문에서 수행한 실험 환경은 아래 표 1에 제시되어 있다.

해상도 조건	240×240, 1920×1080
참여자 수	2, 3, 5, 7 (및 1920*1080에서 9 추가)
서버:	Intel® Core™ Ultra 5 125H
클라이언트	Intel® Celeron® N5100 @ 1.10GHz

표 1. 실험환경

아래 표 2 에서는, 참여 클라이언트 노드 수 및 전송 화면 데이터의 해상도 변화에 따른 전송 효율을 계측한 결과 이다. 결과에서 보듯이, 240*240 같은 저해상도에서는, FP는 큰 차이가 없었으나, 전송노드의 CPU 부하가 점차적으로 높아졌다. 그러나, 해상도를 HD급 이상인 1920×1080 로 상향했을 경우, 참여자 수가 증가할수록 FPS 저하 및 데이터 송신으로 인한 CPU 부하가 크게 증가함을 알 수 있다. 특히, 10대 이상 Mesh 로 연결할 경우, 전송이 단절되고 전송이 중단됨을 볼 수 있었다.

참여자 수	240×240 FPS	240×240 클라이언트 CPU(%)	1920×1080 FPS	1920×1080 클라이언트 CPU(%)
2	30	35	30	60
3	30	45	30	70
5	30	65	27	80
7	30	70	20(out) / 10(in)	100
9	N/A	N/A	N/A(불안정)	100

표 2. 참여자 수 및 해상도 변화에 따른 성능(FPS, CPU) 측정 결과

위의 결과를 좀 더 자세히 분석하기 위해, 본 연구에서는, 전송 중 inbound와 outboud 의 FPS의 변화와, 재전송 관련 데이터를 자세히 조사하였다. 그림 1과 그림 2의 그래프는 1920×1080 환경에서 참여 client 수 7명 및 9명 조건의 Inbound/Outbound FramesPerSecond (FPS)를 나타내며, 프레임 드롭과 급격한 변동이 반복되어 통신 품질이 안정적으로 유지되지 않음을 확인할 수 있다.

저해상도에서는 7명의 참여자까지 FPS 30을 유지하였다. 서버 CPU는 약 5%로 안정적이었으나, 참여자 수 증가에 따라 클라이언트 CPU가 증가하였다. 고해상도에서는 참여자 수가 증가하면서 FPS가 30→27로 감소하였고, 7명 이후로는 불안정한 모습을 보였다. 클라이언트 CPU는 7명 이후 100%로 포화되었다. 서버 CPU는 저해상도와 동일하게 5% 수준으로, 본 실험의 결과는 서버보다는 클라이언트 개별 단말의 처리 성능 및 네트워크 상태에 더 의존하는 것으로 나타났다.

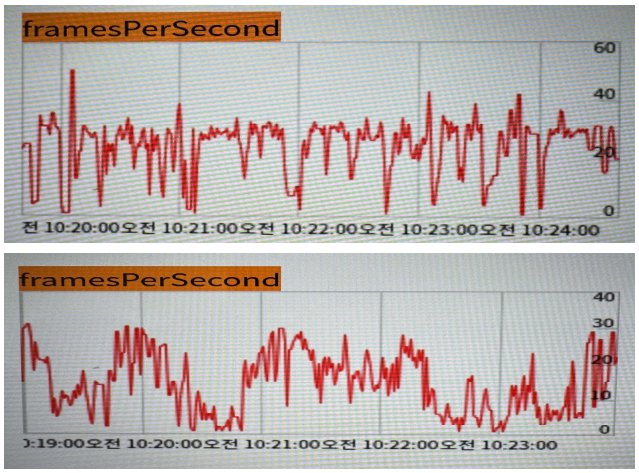

그림 1 클라이언트 7대의 경우 FPS (상) inbound (하)outbound

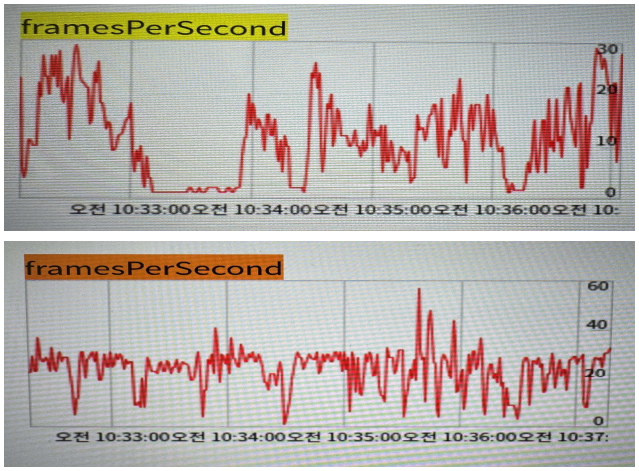

그림 2 클라이언트 9대의 경우 FPS (상) inbound (하)outbound

또한, retransmittedPacketsSent/Received는 통신이 진행되는 동안 누적되는 지표이므로 시간이 지날수록 값이 증가하는데. 본 연구에서는 참여자 수가 늘어날수록 retransmittedPackets 지표의 누적 증가가 두드러졌다.

Ⅲ. 결론

본 논문은 millo-L GitHub의 1:N WebRTC 예제를 대상으로, Mesh 기반 1:N 구조에서 참여자 수 및 해상도 변화가 성능에 미치는 영향을 사례 연구 형태로 분석하였다. 실험은 240×240과 1920×1080 해상도 조건에서 참여자 수를 2명부터 최대 9명까지 증가시키며 FPS(1대 기준)와 서버/클라이언트 CPU 사용률을 측정하는 방식으로 수행하였다.

측정 결과, 240×240 저해상도 조건에서는 7명의 참여자까지 FPS 30을 유지하여 비교적 안정적인 동작이 확인되었다. 반면 1920×1080 고해상도 조건에서는 참여자 수 증가에 따라 FPS가 30에서 27로 감소하였고, 7명 조건에서 프레임율(FPS)이 크게 저하되며 통신이 불안정해지는 양상이 나타났다. 또한 클라이언트 CPU 사용률은 7명 이후 100%로 포화되어, 본 실험 환경에서는 서버 자원보다 클라이언트 단말의 처리 성능 및 네트워크 상태가 성능 저하의 주요 요인으로 작용함을 시사한다. 추가로 retransmittedPacketsSent/Received는 실험을 진행하는 동안 누적되는 지표로 관측되었으며, 참여자 수 증가에 따라 동일 관측 구간에서 재전송 관련 지표의 누적 증가가 두드러지는 경향을 확인하였다.

본 사례 연구는 Mesh 기반 1:N WebRTC가 저해상도·소규모 참여자 환경에서는 구현 및 운영이 용이한 반면, 고해상도·다자 환경에서는 단말 처리 한계와 네트워크 혼잡으로 인해 품질 저하가 조기에 발생할 수 있음을 실측 결과로 제시하였다.

이러한 문제를 해결하기 위해서는, Scalable Tree 구조와 같이, 클라이언트 노드 수의 증가를 지원할 수 있는 새로운 webRTC구조가 필요할 것으로 보이며, 본 연구의 후속연구로 해당 구조의 개발과 구현을 진행할 계획이다.

ACKNOWLEDGMENT

본 연구논문은 한국연구재단 기초 연구 사업의 지원을 받아 수행된 연구임 (NRF-2022R1A2C1009302)

참 고 문 헌

[1] Google, Upload client metrics to the Meet Media API https://developers.google.com/workspace/meet/media-api/guides/metrics

[2] Microsoft, Browser logs and tracing for Teams https://learn.microsoft.com/en-us/microsoftteams/browser-logs-and-tracing-for-teams

[3] Zoom, Zoom Video SDK for web https://developers.zoom.us/docs/video-sdk/web/

[4] millo-L, Typescript-ReactJS-WebRTC-1-N-P2P, GitHub repository. https://github.com/millo-L/Typescript-ReactJS-WebRTC-1-N-P2P

[5] "V. Stanciu, M. I. Andreica, V. Olaru, and N. Ţăpuş, “Design and Development of a UDP-Based Connection-Oriented Multi-Stream One-to-Many Communication Protocol”, JTIT, vol. 47, no. 1, pp. 3 - 15, Mar. 2012, doi: 10.26636/jtit.2012.1.1247.

[6] WebRTC, WebRTC 공식 사이트 https://webrtc.org/