https://doi.org/10.7840/kics.2025.50.12.1907

WebRTC 기반 실시간 포인트 클라우드 영상 스트리밍을 위한 데이터 직렬화 기법 비교

이 용 준◆, 정 한 희*, 유 동 호°

Comparison of Data Serialization Techniques for Point Cloud Video Streaming in Real Time over WebRTC

Yongjun Lee◆, Han Hee Jung*, Dongho You°

요 약

메타버스와 가상현실(VR) 콘텐츠 기술의 발전으로, 현실을 정밀하게 재현하는 포인트 클라우드 기술의 수요가 증가하고 있다. 하지만 데이터의 대용량 특성으로 인해 실시간 전송에는 높은 대역폭이 요구되어, 상호작용 기반 응용에서 제약이 발생한다. 본 연구에서는 WebRTC 기반의 실시간 전송 시스템을 구축하고, RDP, Protobuf, Draco 세 가지 방식의 인코딩 시간, 전송 지연, 디코딩 시간, 수신 FPS, 데이터 크기를 비교 분석하였다. 실험 결과, RDP는 빠른 인코딩·디코딩 속도를 보였고, Protobuf는 균형 잡힌 성능을 제공했으며, Draco는 높은 압축률과 낮은 전송 지연을 달성하였다. 각 방식은 서로 다른 특성을 보이며, 포인트 클라우드의 실시간 전송에는 전송 효율성과 실시간성 간의 균형 있는 선택이 필요함을 확인하였다.

키워드 : 포인트 클라우드, 저지연 스트리밍, WebRTC
Key Words : Point Cloud, Low-latency Streaming, WebRTC

ABSTRACT

With the advancement of metaverse and virtual reality (VR) content technologies, the demand for point cloud technology that precisely reproduces the real world is increasing. However, due to the large data size, real-time transmission requires high bandwidth, which poses challenges for interaction-based applications. This study builds a WebRTC-based real-time transmission system and compares the performance of three methods—RDP, Protobuf, and Draco—in terms of encoding time, transmission delay, decoding time, received FPS, and data size. Experimental results show that RDP achieved fast encoding and decoding speeds, Protobuf provided balanced performance, and Draco achieved high compression efficiency and low transmission delay. Each method exhibits different characteristics, confirming that a balanced choice between transmission efficiency and real-time performance is essential for point cloud transmission.

※ 본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. RS-2023-00278925).
◆ First Author : Hannam University, Department of Information and Communication Engineering, 20254055@m365.hnu.ac.kr, 정회원
° Corresponding Author : Kongju National University, Department of Computer Education, dongho.you@kongju.ac.kr, 정회원
* Hannam University, Department of Information and Communication Engineering, hanheejung18@hnu.kr, 정회원
논문번호 : 202506-143-C-RU, Received June 26, 2025; Revised July 15, 2025; Accepted July 28, 2025

Ⅰ. 서 론

최근 메타버스, 가상현실(VR) 등 초실감형 콘텐츠 기술이 급속도로 발전함에 따라, 현실 세계를 가상공간에 정확하고 생생하게 구현하려는 수요가 증가하고 있다.[1] 이러한 배경 속에서 3차원 공간 데이터를 효과적으로 표현할 수 있는 포인트 클라우드(Point Cloud)가 주요 기술로 주목받고 있다. 포인트 클라우드는 LiDAR, RGB-D 센서 등으로 획득한 수많은 점들의 3차원 좌표 정보로 구성되어 있으며, 실제 공간의 구조와 형태를 고정밀도로 묘사할 수 있다는 장점을 갖는다.[2]

그러나 포인트 클라우드는 수십만 개에서 수백만 개 이상의 점으로 이루어져 있어 데이터 용량이 매우 크고, 이를 실시간으로 전송s하기 위해서는 높은 대역폭이 요구된다. 이러한 문제를 해결하기 위해, MPEG에서 제안한 V-PCC(Video-based Point Cloud Compression), G-PCC(Geometry-based Point Cloud Compression) 등 다양한 압축 기술이 개발되어 왔다.[3] 이들 기술은 데이터 용량을 효과적으로 줄이지만, 인코딩 및 디코딩 과정에서 불가피하게 지연(latency)이 발생하며, 압축 과정에서 원본 정밀도가 손상될 수 있다. 이에 따라 상호작용성이 중요한 실시간 응용 환경에서는 사용에 제한이 따른다.

이러한 지연 문제를 해결하기 위한 방안으로, 선행 연구[4]에서는 WebRTC의 데이터 채널을 이용하여 압축하지 않은 형태의 포인트 클라우드 데이터를 직접 전송하는 RDP(RawDataPack) 방식이 제안된 바 있다. WebRTC(Web Real-Time Communication)는 웹 브라우저 환경에서 별도의 소프트웨어 설치 없이 음성, 영상, 데이터 등을 P2P 방식으로 실시간 전송할 수 있도록 구글에서 개발한 개방형 웹 표준 프로토콜이다.[5] WebRTC는 크게 GetUserMedia, RTCPeerConnection, RTCDataChannel[6]의 세 가지 API 계층으로 구성된다. GetUserMedia는 사용자의 카메라 및 마이크 장치로부터 영상 및 오디오 스트림을 수집하며, RTCPeer Connection은 해당 스트림을 네트워크를 통해 전송하는 역할을 수행한다. 이와 함께 포함되는 RTCData Channel은 텍스트 및 바이너리 데이터를 전송할 수 있는 채널을 제공하며, SCTP(Stream Control Transmission Protocol)를 기반으로 하여 신뢰성 및 순서 보장 여부를 설정할 수 있는 유연성을 갖춘다. 이러한 구조는 포인트 클라우드와 같이 고속 대용량 비정형 데이터를 전송할 때 효과적인 전송 방식을 설계할 수 있는 기반이 된다.

선행 연구에서는 이 RTCDataChannel의 특성을 활용하여, 포인트 클라우드 데이터를 압축 없이 실시간 전송하는 가능성을 입증하였다. 이 과정에서 JSON이나 XML과 같은 텍스트 기반 직렬화 포맷도 고려되었으나, 해당 방식들은 데이터 크기가 원본 대비 과도하게 증가하여 전송 효율이 매우 낮다는 한계가 있어 실시간 전송에는 적합하지 않은 것으로 판단되어 제외되었다. 또한 해당 연구는 전송 가능성과 저지연성에 대한 기본적인 확인 수준에 머물렀으며, 포인트 수의 증가에 따른 상세한 성능 분석이나, 포인트 클라우드 압축 방식과의 비교 분석은 수행되지 않았다.

이에 본 연구는 이러한 선행 연구의 한계를 극복하고, WebRTC 환경에서의 포인트 클라우드 데이터 전송 성능을 보다 정밀하게 평가하고자 한다. 이를 위해, 기존의 RDP 방식뿐만 아니라 성능 평가 대상이었던 Protobuf[7] 기반 전송 방식, 그리고 기존 연구에서 다루지 않았던 Draco 압축 방식을 새롭게 도입하여 비교하였다. Draco는 구글에서 개발한 오픈소스 기반의 3차원 데이터 압축 기술로, 포인트 클라우드 데이터를 효과적으로 압축하면서도 타 압축기술인 G-PCC 또는 V-PCC에 비하여 빠른 인코딩 및 디코딩 속도를 제공하는 특징이 있다.[8] 본 연구에서는 RDP, Protobuf, Draco 방식 간의 성능을 포인트 수의 증가에 따라 비교·분석함으로써, WebRTC 기반 실시간 포인트 클라우드 전송 기술의 실용성과 한계를 종합적으로 평가하고자 한다.

Ⅱ. 본 론

2.1 시스템 구성

본 시스템은 획득부, 처리 및 전송부, 수신부의 세 구성 요소로 이루어져 있으며, 각 요소는 실시간 포인트 클라우드 생성 및 전송을 위한 연속적인 처리 파이프라인을 형성한다. 전체 시스템의 흐름은 그림 1의 시스템 구성도와 그림 2, 그림 6 알고리즘에 기반하여 다음과 같이 동작한다.

2.2 획득부

획득부에서는 포인트 클라우드 데이터는 RGB-D 센서인 Azure Kinect DK에서 Color Image와 Depth Image를 결합하여 생성된다. Color Image와 Depth Image를 통한 포인트 클라우드 생성 과정은 다음과 같다.

먼저, 카메라 초기 단계에서 Azure Kinect DK가 3D 변환을 위한 X, Y 테이블을 미리 생성하여, 이 값을 통해 실시간으로 3D 좌표 변환을 원활하게 수행할 수 있도록 한다. 그 후, 아래와 같은 수식을 사용하여 실시

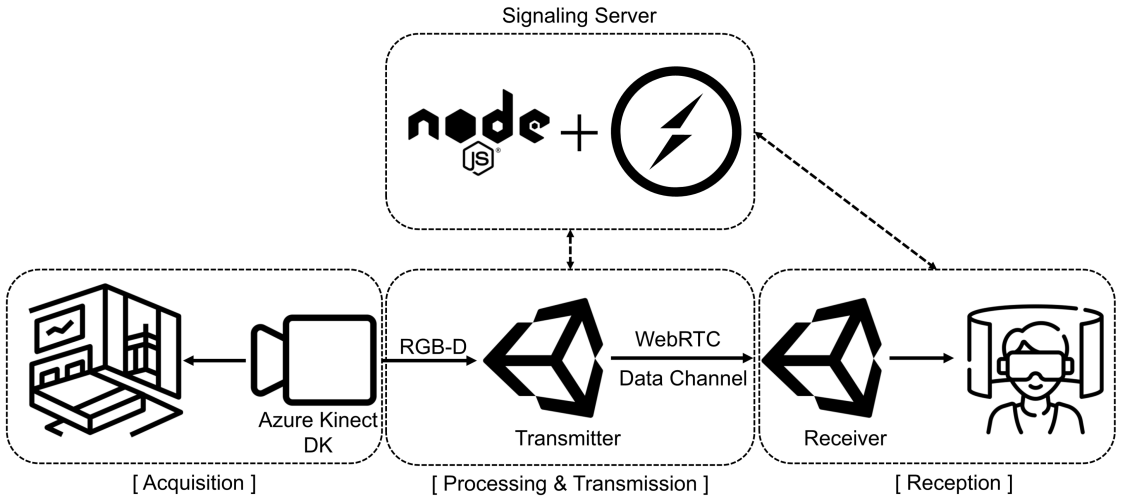

그림 1. 시스템 구성도
Fig. 1. System configuration diagram

간으로 캡처되는 Depth Image의 좌표를 X, Y 테이블 값과 곱하여 3D 좌표로 변환한다

$$\begin{aligned} x(u,v) &= X_{table}(u,v) \times I_d(u,v) \\ y(u,v) &= Y_{tavle}(u,v) \times I_d(u,v) \\ z(u,v) &= I_d(u,v) \end{aligned} \tag{1}$$

여기서 (u,v)는 Color 또는 Depth Image에서 캡처된 2D 좌표, X_{table}과 Y_{table}은 사전에 계산된 좌표 변환 값이며, I_d는 Depth Image를 의미한다.

이를 통해 2D 좌표와 Depth값을 이용해 $x(u,v), y(u,v), z(u,v)$ 라는 3D 좌표를 생성한다.

동시에, Color Image는 Depth Image 기준으로 생성된 Point Cloud에 맞는 색상을 추출하기 위해 Depth Image의 해상도로 변환한다. 변환된 Color Image에서 Depth Image의 좌표값을 이용하여 아래와 같은 수식으로 Depth Image에 맞는 Color값을 추출한다.

$$\begin{aligned} b(u,v) = I^B_{c,proj}(u,v),\ g(u,v) &= I^G_{c,proj}(u,v) \\ r(u,v) = I^R_{c,proj}(u,v),\ a(u,v) &= I^A_{c,proj}(u,v) \end{aligned} \tag{2}$$

여기서 $I_{c,proj}$는 캡처하여 Depth Image 해상도로 변환된 Color Image를 의미하며, 색상값은 BRGA 포맷으로 각 좌표에 맞는, B,R,G,A 값이 추출된다.

이 두 과정을 통해 얻어진 3D 좌표와 색상값을 이용하여 색상이 포함된 포인트 클라우드 데이터를 생성한다. 또한, Azure Kinect에서 사용하는 좌표계와 Unity의 좌표계가 다르기 때문에, 일치시키기 위해 Y 좌표를 반전시키며, 원하는 크기를 위한 스케일링 팩터 s를 곱하여 크기를 조정할 수 있다.

$$\begin{aligned} x'(u,v) &= x(u,v) \times s \\ y'(u,v) &= -y(u,v) \times s \\ z'(u,v) &= z(u,v) \times s \end{aligned} \tag{3}$$

이러한 과정을 통해 얻어진 Unity 좌표계의 3D 좌표값과 색상값을 사용하여 아래 수식과 같이 포인트 클라우드 한 프레임을 표현할 수 있다. 포인트 수는 미리 실험을 위해 설정한 임계값 또는 Depth Image의 해상도 중 작은 값에 해당하는 포인트들만 수집된다. 이 과정을 반복하여 실시간 포인트 클라우드 생성을 구현하며, 포인트 클라우드 데이터가 생성되는 FPS(Frame Per Second)는 Azure Kinect가 촬영하는 FPS에 따라 설정되며, 수식으로 나타내면 다음과 같다.

$$P = (x'_n, y'_n, z'_n, b_n, g_n, r_n, a_n) \tag{4}$$

여기서 $n = 1,2,...,\min(N, W_d \times H_d)$이다.

2.3 처리 및 전송

본 시스템의 처리 및 전송부에서는 실시간으로 생성된 포인트 클라우드 데이터를 WebRTC의 데이터 채널을 통해 수신 측으로 전송한다. 포인트 클라우드는 Azure Kinect DK에서 획득한 RGB-D 데이터를 기반

으로 매 프레임마다 생성되며, 각 프레임에는 고유한 타임스탬프가 부여된다. 생성된 포인트 클라우드 데이터는 WebRTC 전송에 적합하도록 직렬화되며, 직렬화 방식으로는 RDP, Protobuf, Draco 세 가지를 지원한다. 전송에는 SCTP 기반의 WebRTC 데이터 채널이 사용되며, 이는 DTLS 및 UDP 위에서 동작하는 메시지 지향 전송 방식으로, 직렬화 방식에 따라 전송 성능과 실시간성에 영향을 미친다.

이때 데이터 채널의 설정을 위해 WebRTC 시그널링 과정이 선행되며, 본 논문에서는 Node.js 기반의 Socket.io를 이용하여 구현하였다. 시그널링 서버는 Peer 간 연결 설정 및 협상에 필요한 SDP(Session Description Protocol) 정보와 ICE(Interactive Connectivity Establishment) Candidate 정보를 교환하며, Unity 기반의 클라이언트와의 연동을 지원한다.

전송을 위해 직렬화된 데이터는 WebRTC 데이터 채널의 전송 가능 패킷 크기 제한을 고려하여 일정 크기의 청크 단위로 분할된다. 각 프레임은 복수의 청크로 나뉘며, 수신 측은 모든 청크를 수신한 뒤 이를 재조합하여 원본 프레임을 복원한다. 프레임이 완전히 수신되었음을 나타내기 위해, 마지막에는 별도의 "END" 신호가 전송된다. 전체 전송 흐름은 그림 2의 송신 알고리즘에서 확인할 수 있다.

또한, 본 시스템에서는 전송 성능을 향상시키기 위해 세 가지 데이터 표현 방식인 RDP, Protobuf, Draco를 각각 구현하였다.

RDP는 각 좌표축과 색상 정보를 개별 배열로 분리하여 순차적으로 나열하는 매우 단순한 구조를 가진다.

Algorithm 1 Real-time Point Cloud Generation and Transmission
Require: Azure Kinect DK, WebRTC Data Channel
Ensure: Real-time Point Cloud Transmission with Timestamp Synchronization
Initialize System
Establish WebRTC connection and configure data channel
Set up Kinect and prepare frame buffers
while WebRTC Data Channel is Open **do**
  Step 1: Capture Data
  Acquire RGB and Depth frames from Kinect
  Step 2: Point Cloud Generation
  Align RGB data to Depth space
  Convert Depth image to 3D Point Cloud
  for all valid points (u, v) in Depth image **do**
    Compute 3D coordinates (X, Y, Z)
    Assign corresponding color (R, G, B, A)
  end for
  Step 3: Data Serialization
  Encode point cloud data
  Step 4: Append Timestamp
  Get current timestamp t
  Attach t to the serialized data
  Step 5: Transmission
  Send serialized data with timestamp over WebRTC Data Channel
  Transmit "END" signal to mark frame completion
end while
Terminate Connection
Close WebRTC connection and release Kinect resources

그림 2. 송신 프로그램 알고리즘
Fig. 2. Transmission Program Algorithm

인코딩 속도가 빠르고 처리 부하가 낮다는 장점이 있으나, 압축이 적용되지 않아 데이터 크기가 크고 수신 측에서 포맷 구조를 미리 알고 있어야 한다는 제한이 있다.

Protobuf는 점 좌표와 색상 정보를 독립된 필드로 정의하여 저장하며, 각 필드는 데이터 종류를 식별하는 태그, 데이터 길이, 그리고 실제 값의 순으로 구성된다. 예를 들어 좌표 정보는 "Vertices"라는 필드로 묶이며, 색상 정보는 별도의 필드에 저장된다. 이 방식은 구조적으로 명확하여 수신 측에서 데이터 경계를 명확히 파악할 수 있지만, 구조 정보를 포함하므로 RDP보다 크기가 커지고 디코딩 과정이 더 복잡하다.

Draco는 고압축률을 목표로 설계된 포맷으로, 데이터를 헤더, 메타데이터, 연결 정보(Connectivity), 속성(Attribute) 블록 순서로 구성하여 저장한다. 좌표 및 색상 데이터는 중복 제거와 부호화를 통해 효율적으로 압축되며, 높은 압축률을 달성할 수 있다. 그러나 인코딩과 디코딩 시간이 비 압축방식에 비하여 상대적으로 길어 실시간 처리에는 제약이 있다. 세 방식의 비트스트림 구성은 그림 3에 나타나 있다.

이처럼 직렬화 방식의 구조와 압축률, 인코딩 복잡도는 서로 다르기 때문에, 동일한 조건에서도 청크 크기에 따라 전송 성능이 달라질 수 있다. 이를 정량적으로 분석하기 위해 각 방식에 대해 32KB, 64KB, 128KB, 256KB의 청크 크기를 설정하여 실험을 진행하였다.

포인트 클라우드 데이터 P는 직렬화 과정을 거쳐 직렬화 데이터 D로 변환되며, 전체 청크 개수 M은 다음과 같이 계산된다.

$$M = \left\lceil \frac{D}{C_{\max}} \right\rceil \tag{5}$$

여기서 $C_{\max}$는 설정된 최대 청크 크기로, 다양한 값으로 설정되어 성능비교에 활용된다. 최대 청크 크기가 정해졌다면, 포인트 클라우드 데이터 직렬화 결과인 D를 각 청크 크기별로 자른다. 이때 자른 각 청크 C_k는 아래와 같이 직렬화된 데이터의 슬라이스로 정의된다.

$$C_k = D[(k-1)C_{\max} : kC_{\max}],\ 1 \le k \le M \tag{6}$$

모든 청크가 성공적으로 전송된 경우에만 해당 포인트 클라우드 직렬화 데이터 D가 완전히 전송되었다고 판단하며, 이는 다음과 같이 표현된다.

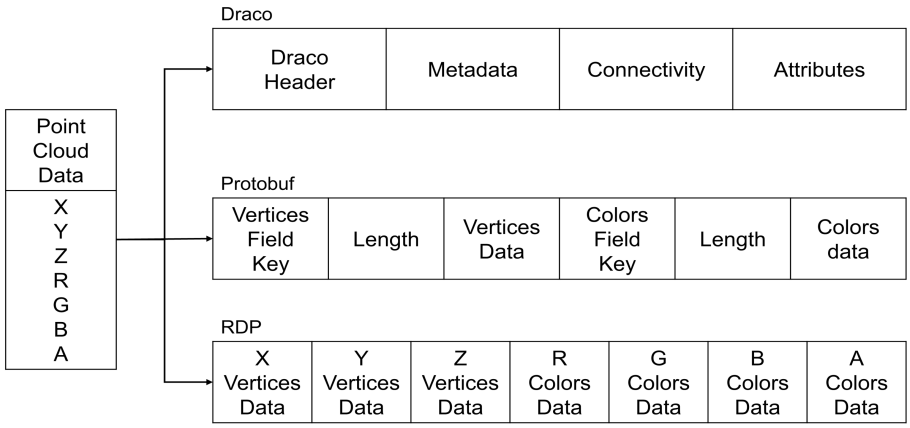

그림 3. 직렬화 방식에 따른 비트스트림 구성
Fig. 3. Bitstream configuration according to the serialization method

$$D_{sent} \Leftrightarrow \forall\, k \in 1,2,\ldots,M,\ C_k \ \ is \ \ sent \qquad (7)$$

실험은 Azure Kinect DK를 통해 획득한 약 10만 개의 포인트로 구성된 포인트 클라우드 프레임을 10만 프레임 연속 전송하는 방식으로 수행되었으며, 전송 방식은 RDP, Protobuf, Draco를 각각 동일한 조건에서 적용하였다. 원본 청크와 비교하였을 때, 모든 청크가 종료 신호 이전에 수신되지 않아 디코딩에 실패한 프레임의 비율로 정의되는 에러 확률을 기준으로 이루어졌다.

그림 4는 청크 크기에 따른 수신 FPS의 변화를 나타낸다. 전반적으로 청크 크기가 증가할수록 수신 FPS가 점진적으로 향상되는 경향을 보였으며, 이는 동일한 데이터 양을 더 큰 단위로 나눌 경우 청크 분할 횟수가 감소하고, 이에 따라 전송 중 발생하는 패킷 수와 수신 측의 병합 및 처리 오버헤드가 줄어들기 때문이다. 이러한 효율성은 결과적으로 전송 지연을 완화하고 수신 FPS를 높이는 데 긍정적인 영향을 미친다.

반면, 그림 5의 결과에 따르면, 청크 크기가 증가할수록 에러 발생 확률이 높아지는 경향이 나타났다. 이는 압축 방식 자체의 정밀도 손실과는 무관하게, 하나의 청크라도 프레임 종료 신호보다 늦게 도착할 경우 전체 프레임 복원이 실패하는 구조적 특성에 기인한다. RDP는 직렬화 구조가 단순하고 압축이 적용되지 않아 인코딩 속도가 빠르다는 장점이 있으나, 데이터 크기와 청크 수가 많아져 전송 지연이 발생하고 손실에 취약한 특성이 있다. Protobuf 또한 압축을 사용하지 않으며 구조화된 데이터 형식을 기반으로 하여 데이터 경계 식별에는 유리하지만, 디코딩 과정이 복잡하고 데이터 크기가 커 전송 효율이 떨어진다. 반면, Draco는 고압축률 기반의

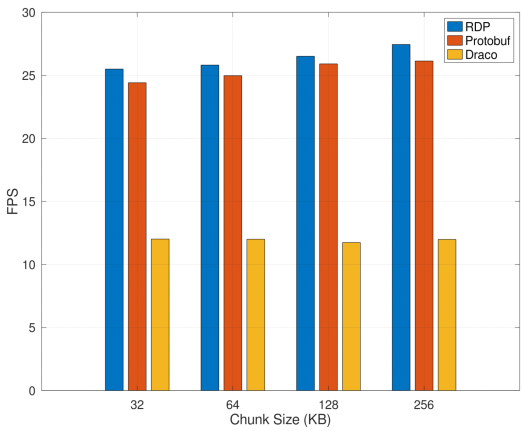

그림 4. 청크 크기별 수신 FPS
Fig. 4. Received FPS by chunk size

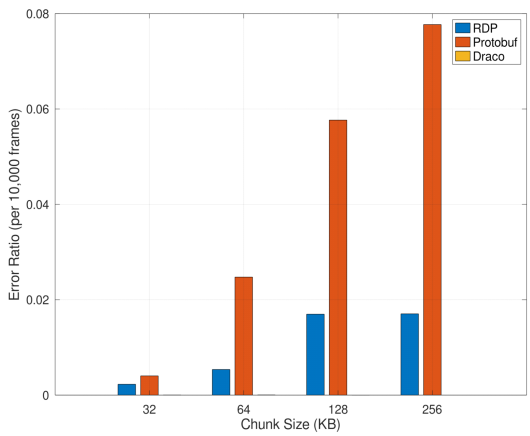

그림 5. 청크 크기별 에러 비율
Fig. 5. Error ratio by chunk size

직렬화 및 압축 기법을 사용하여 청크 수를 효과적으로 줄일 수 있으며, 그 결과 에러율이 낮고 전송 효율성이 개선된다. 하지만 인코딩 및 디코딩 시간이 길어 실시간 처리 성능에서는 불리하다.

청크 크기의 증가는 수신 FPS 향상과 전송 효율 개선 측면에서는 긍정적인 영향을 미치지만, 손실 발생 시 전체 프레임의 복원이 불가능해지는 위험 또한 증가하게 된다. 이러한 특성은 실시간성과 전송 안정성 간의 전형적인 trade-off로 볼 수 있으며, 응용 환경에 따라 적절한 균형 설정이 필수적이다. 본 연구에서는 실시간 스트리밍 환경에서 사용자 체감 품질 확보를 목표로 하여, 이후 모든 성능 실험에서 청크 크기를 256KB로 고정하여 일관된 조건 하에 성능을 비교하였다.

2.4 수신 및 렌더링

WebRTC 데이터 채널을 통해 송신 측으로부터 전송된 포인트 클라우드 데이터는 수신 측에서 실시간으로 수신되며, 이후 복원 및 시각화 과정을 거쳐 사용자에게 제공된다. 수신된 데이터는 프레임 단위로 버퍼에 저장되며, 마지막 청크를 식별하는 "END" 신호가 수신되었을 때 해당 프레임의 복원 처리가 시작된다.

각 프레임은 전송 전 직렬화되어 바이트 시퀀스 형태로 변환되고, 이후 설정된 최대 청크 크기에 따라 여러 청크로 분할되어 전송된다. 수신 측에서는 전체 청크가 정상적으로 도달했는지를 확인한 후 복원 처리를 수행하며, 이 과정을 수식으로 표현하면 다음과 같다.

$$D_{recv} \Leftrightarrow \forall k \in 1,2,...,M, C_k \ is \ received \tag{8}$$

Algorithm 2 Real-time Point Cloud Reception and Rendering

Require: WebRTC Data Channel
Ensure: Continuous Reception, Processing, and Rendering of Point Cloud Data

Initialize WebRTC Connection
Establish WebSocket connection with the signaling server
Create WebRTC peer connection and configure data channel
Set up buffers for incoming point cloud data
while WebRTC Data Channel is Open **do**
    Step 1: Receive Data Packet
    if Received Data == "END" **then**
        Step 2: Reconstruct Full Point Cloud
        Deserialize received data
        Extract and process point cloud information
        Clear received data buffer
    else
        Store received packet in buffer
    end if
end while
Step 3: Parse and Extract Point Cloud Data
for all Points i in received data **do**
    Extract 3D coordinates (X_i, Y_i, Z_i)
    Extract color values (R_i, G_i, B_i, A_i)
end for
Step 4: Render Point Cloud
Update mesh vertices and colors with extracted data
Recalculate mesh bounds for optimal rendering

그림 6. 수신 프로그램 알고리즘
Fig. 6. Reception Program Algorithm

위 수식은 수신 측이 M개의 모든 청크에 해당하는 $C_1 \sim C_{\max}$ 를 수신한 경우에만 해당 데이터가 수신 완료되었다고 판단함을 의미한다. 즉, 일부 청크가 누락되거나 손상되었을 경우, 해당 프레임은 복원 대상에서 제외된다.

모든 청크가 수신되면, 수신된 데이터 D_{recv} 는 각 청크의 결합을 통해 다음과 같이 정의된다.

$$D_{recv} = \bigcup_{k=1}^{M} C_k \tag{9}$$

이와 같이 결합된 바이트 시퀀스는 직렬화 이전의 포인트 클라우드 데이터 구조로 복원되어야 하며, 이를 위해 역직렬화 함수 S^{-1} 이 사용된다. 복원된 포인트 클라우드 P_{recv} 는 다음과 같이 표현된다.

$$P_{recv} = S^{-1}(D_{recv}) \tag{10}$$

복원된 데이터 P_{recv} 는 각 포인트의 좌표와 색상 정보가 포함된 3차원 점들의 집합으로 구성되며, Unity 엔진 내 렌더링 모듈을 통해 실시간으로 시각화된다. 전체 수신 및 복원 알고리즘은 그림 6의 Algorithm 2에 상세히 기술되어 있다.

결과적으로, 포인트 클라우드 데이터가 렌더링되어 시각적으로 표현되기 위해서는 모든 청크가 손실 없이 정확히 수신되어야 하며, 역직렬화 과정을 통해 원래의 3차원 좌표 데이터를 성공적으로 복원해야 한다. 이 과정은 실시간 처리 상황에서의 데이터 손실 복원 여부 및 디코딩 정확도와 직결되며, 에러율 지표를 정의하는 핵심적인 기준이 된다.

이와 같이, 본 시스템은 송신 측에서의 포인트 클라우드 생성 및 압축부터 수신 측에서의 복원 및 실시간 렌더링까지의 전 과정을 프레임 단위로 반복 처리하며, WebRTC 기반 저지연 통신 구조를 통해 실시간 저지연 포인트 클라우드 스트리밍을 가능하게 한다.

Ⅲ. 성능 분석

3.1 테스트 환경 및 설정

본 연구에서는 포인트 클라우드 전송 시스템의 성능을 정량적으로 평가하기 위해 송신기와 수신기를 동일한 로컬 컴퓨터 환경에서 구동하였다.

이는 외부 변수의 영향을 최소화하고 비교 실험의 일관성을 확보하기 위한 설정으로, 실제 네트워크 환경

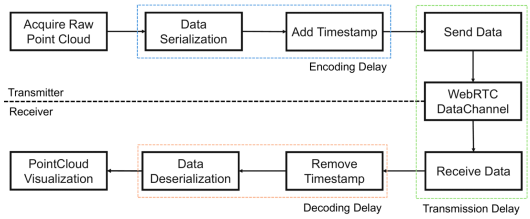

그림 7. Delay 구조도
Fig. 7. Delay structure diagram

에서 발생할 수 있는 지연, 패킷 손실, 연결 품질의 변동성과 같은 요소는 반영되지 않았다는 한계가 있다. 실험에 사용된 하드웨어는 Intel Core i9-13900KF CPU, NVIDIA RTX 4080 GPU, 64GB RAM이며, 운영체제는 Windows 11이다. 포인트 클라우드 생성에는 Azure Kinect DK를 사용하였고, 640×576 해상도에서 초당 30FPS으로 Depth 이미지를 획득하였다. 송신기 및 수신기 애플리케이션은 Unity 기반으로 구현되었으며, WebRTC 데이터 채널을 통해 실시간 포인트 클라우드 전송을 수행하였다.

실험은 포인트 수를 50K에서 350K까지 50K 단위로 변화시키며 진행하였고, RDP, Protobuf, Draco 세 가지 전송 방식을 각각 적용하여 성능을 비교하였다. 특히 Draco 방식의 인코딩 및 디코딩은 Unity에서 기본으로 제공하는 설정값(default configuration)을 그대로 사용하였으며, 별도의 파라미터 최적화는 수행하지 않았다. 이는 실제 Unity 기반 응용에서의 기본적인 성능을 평가하기 위함이다. 주요 측정 항목은 그림 7과 같이 인코딩 지연, 전송 지연, 디코딩 지연, 수신 FPS, 데이터 크기로 설정하였으며, 각 방식에 대해 총 100만 프레임을 연속 수신하면서 해당 항목의 누적 평균값을 산출하였다.

3.2 전송 방식에 따른 성능 비교

3.2.1 인코딩 지연 비교

그림 8은 포인트 수 증가에 따른 각 전송 방식별 인코딩 지연을 나타낸다. RDP 방식은 50K 포인트 기준 7.3ms에서 350K 포인트 기준 56.9ms까지 선형적으로 증가하였다. Protobuf 방식은 같은 범위에서 3.8ms에서 27.6ms로 증가하여, RDP 대비 약 50% 이하의 인코딩 지연을 기록하였다. 반면, Draco 방식은 50K 포인트에서도 36.8ms의 높은 인코딩 지연이 소요되었으며, 350K 포인트에서 297ms에 이르러 다른 방식 대비 현저히 긴 인코딩 지연을 보였다. 이는 Draco가 높은 압축률을 얻기 위해 복잡한 압축 연산을 수행함을 보여주는 결과이다.

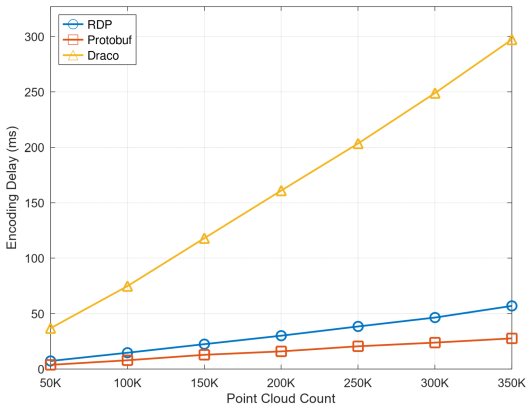

그림 8. 인코딩 지연 비교
Fig. 8. Comparison of encoding delays

3.2.2 전송 지연 비교

그림 9은 포인트 수에 따른 전송 지연을 나타낸다. RDP 방식은 50K 포인트 기준 23.4ms, 350K 포인트에서는 87.2ms로 선형 증가하였다. Protobuf 방식은 22.6ms에서 110.2ms로 증가하여 RDP보다 항상 높은 전송 지연을 기록하였다. 이는 Protobuf의 데이터 크기가 상대적으로 크기 때문으로 해석된다.Draco 방식은 데이터 크기 절감 덕분에 50K 포인트 기준 2.7ms, 350K 포인트에서도 22.1ms에 불과하여, 다른 두 방식 대비 최대 4배 이상의 전송 지연 개선을 보였다. 전송 지연 최소화 측면에서는 Draco 방식이 압도적인 우위를 보였다.

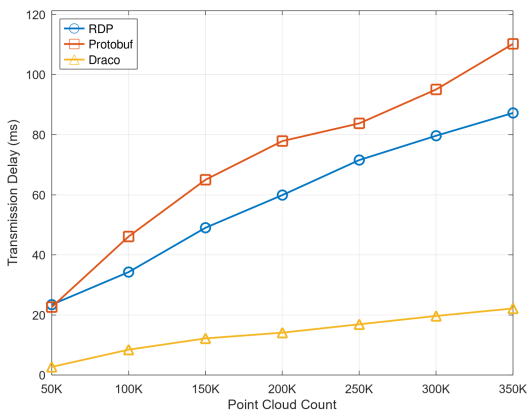

그림 9. 전송 지연 비교
Fig. 9. Comparison of transmission delays

3.2.3 디코딩 지연 비교

그림 10은 전송된 데이터를 수신한 후 디코딩하는 데 소요된 시간을 나타낸다. RDP 방식은 50K 포인트

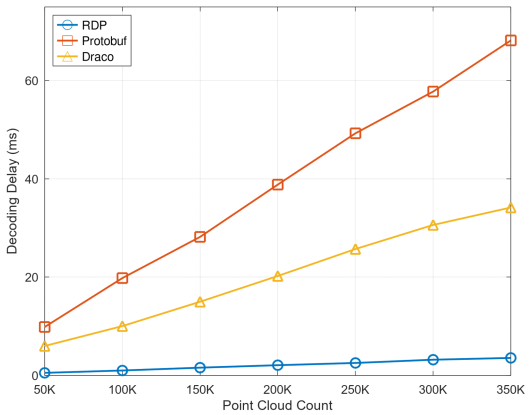

그림 10. 디코딩 지연 비교
Fig. 10. Comparison of decoding delays

기준 0.53ms, 350K 포인트 기준 3.57ms로 매우 낮은 디코딩 지연을 보였다.

이는 압축 과정이 없기 때문에 수신 후 바로 메모리에 로드할 수 있기 때문이다. Protobuf 방식은 50K 포인트 기준 9.86ms에서 350K 포인트 기준 68.1ms까지 꾸준히 증가하였으며, 압축 해제 및 데이터 구조 복원에 따른 부하가 원인으로 분석된다. Draco 방식은 5.99ms에서 34.16ms로 증가하여, Protobuf보다는 디코딩 속도가 빠르지만인코딩 지연에 비하면 상대적으로 부담이 적은 편이었다.

3.2.4 데이터 크기 비교

그림 11는 전송 데이터의 크기 변화를 보여준다. RDP 방식은 50K 포인트 기준 0.8MB에서 350K포인트 기준 5.6MB까지 증가하였다. Protobuf는 0.95MB에서 6.65MB로 RDP보다 약 15~20% 큰 데이터 크기를 유지하였다. Draco는 압도적인 데이터 절감을 보여주었으며, 50K 포인트 기준 0.1MB, 350K 포인트 기준에서도 0.8MB 수준에 머물렀다. 이는 Draco가 RDP 대비 약 85% 이상의 데이터 압축률을 제공함을 나타낸다.

3.2.5 수신 FPS 비교

그림 12은 수신 측에서 측정된 FPS 결과를 나타낸다. 50K 포인트 기준으로 RDP와 Protobuf는 약 30FPS를 유지했으나 Draco는 약 23FPS로 시작하였다. 포인트 수가 증가함에 따라 모든 방식에서 수신 FPS가 감소하는 경향을 보였으며, 특히 Draco는 350K 포인트에서 약 3.2FPS까지 급격히 하락하였다. 반면 RDP와 Protobuf는 8~9FPS 수준을 유지하였다. 이는 Draco 방식의 높은 인코딩 및 디코딩 부하가 실시간 처리 성능에 영향을 주어 송신부에서 실시간으로 포인트 클라우드 직렬화를 타 직렬화 방식보다 빠르게 처리하지 못하여 FPS가 낮은 것을 확인하였다.

다만 본 실험은 동일 시스템 내에서의 이상적인 네트워크 환경을 전제로 하였기 때문에, 실제 스트리밍 환경에서의 조건과는 차이가 존재한다. 특히 대역폭이 제한되거나 패킷 손실이 빈번한 환경에서는, 고압축률로 인해 전송 데이터 크기와 청크 수가 적은 Draco 방식이 오히려 전송 안정성과 복원성공률 측면에서 더 유리할 수 있다. 반면, RDP와 Protobuf는 수신 FPS는 높게 유지하더라도 데이터 크기와 청크 수가 많아 손실에 더 취약한 구조를 가지므로, 제한된 네트워크 조건에서는 실시간성이 저하될 가능성이 있다. 이러한 점을 고려하면, 네트워크 환경의 특성에 따라 직렬화 방식의 선택 전략이 달라져야 하며, 후속 연구에서는 다양한 네트워크 조건을 반영한 실험을 통해 이 같은 예측을 정 량적으로 검증할 예정이다.

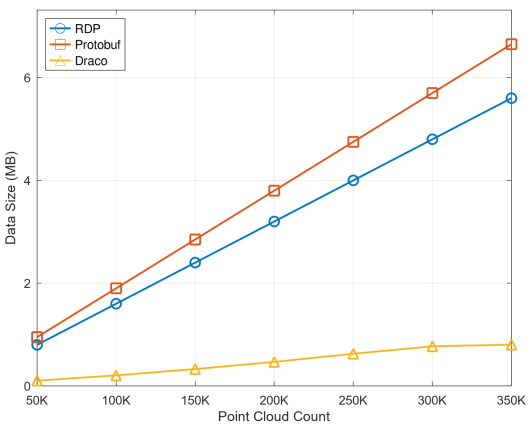

그림 11. 데이터 크기 비교
Fig. 11. Comparison of data sizes

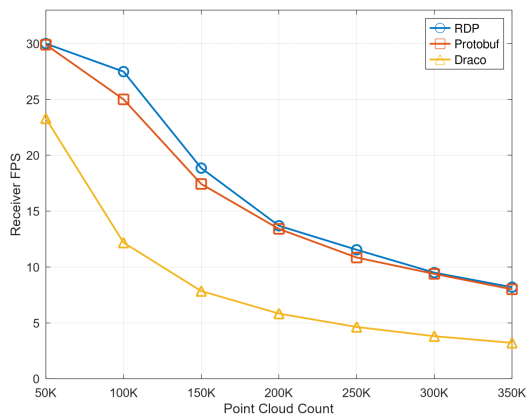

그림 12. 수신 FPS 비교
Fig. 12. Comparision of incoming FPS

3.2.6 종합 분석

본 실험 결과를 종합하면, 각 전송 방식은 고유의 장단점을 지닌다. RDP 방식은 인코딩 및 디코딩이 빠르고 수신 FPS가 높지만, 데이터 크기가 커서 전송 지연이 상대적으로 크다. Protobuf는 구조화된 데이터 직렬화로 인해 인코딩은 빠르지만, 디코딩 지연이 길고 데이터 크기가 더 커 실시간성에는 다소 불리하다. Draco는 높은 압축률 덕분에 전송 지연이 가장 적지만, 인코딩과 디코딩 부하가 크고 수신 FPS가 급격히 저하되어 실시간 3D 스트리밍에는 한계가 존재한다.

또한, 포인트 수가 증가할수록 모든 전송 방식에서 성능 저하가 관찰되었으며, 특히 수신 FPS와 전송 지연 측면에서 민감한 반응을 보였다. 실시간성과 데이터 무결성 간의 균형을 고려할 때, 특정 응용 환경에 따라 전송 방식을 적절히 선택하는 전략이 필요함을 확인할 수 있다.

표 1과 같은 성능 비교를 바탕으로, 대역폭 조건에 따라 적합한 전송 방식을 선택하는 전략이 필요함을 알 수 있다.

대역폭이 제한된 환경에서는 프레임당 데이터 크기와 전송 지연이 시스템 성능에 가장 큰 영향을 미치므로, 압축 효율이 우수한 Draco 방식이 가장 적합하다. Draco는 인코딩 지연이 크고 수신 FPS는 낮지만, 상대적으로 훨씬 작은 데이터 크기로 인해 동일 시간 내 더 많은 프레임 전송이 가능해 전송 효율성 측면에서 유리하다. 반면, 충분한 대역폭이 확보된 환경에서는 인코딩 및 디코딩 속도가 빠르고 수신 FPS가 높은 RDP 방식이 적합하며, 실시간성과 응답성이 중요한 응용에서 효과적으로 활용될 수 있다.

표 1. 전송 방식별 전반적인 성능 수준 비교 (50K~350K 기준)
Table 1. Overall Performance Comparison of Transmission Methods (Based on 50K – 350K Points)

Method	Encoding Delay	Transmission Delay	Decoding Delay	Data Size	Received FPS
RDP	Medium	High	Low	Large	Medium
Protobuf	Low	High	High	Large	Medium
Draco	High	Low	Medium	Small	Low

Ⅳ. 결 론

본 연구에서는 WebRTC 기반의 포인트 클라우드 실시간 전송 시스템을 구축하고, RDP, Protobuf, Draco 세 가지 전송 방식을 적용하여 성능을 정량적으로 비교·분석하였다. 기존 연구가 단순 전송 가능성과 지연 측정에 머물렀던 것에 비해, 본 연구는 포인트 수 증가에 따른 인코딩 지연, 전송 지연, 디코딩 지연, 데이터 크기, 수신 FPS 등 다양한 지표를 종합적으로 측정하고 분석함으로써, 실시간 3D 데이터 전송 기술의 한계와 가능성을 보다 구체적으로 규명하였다.

성능 분석 결과, RDP 방식은 인코딩과 디코딩 속도 면에서는 가장 우수하였지만, 데이터 크기가 크고 전송 지연이 커지는 단점이 있었다. Protobuf 방식은 구조화된 직렬화 포맷 덕분에 인코딩은 빠르지만 디코딩 부하가 크고, 데이터 크기도 RDP보다 더 커서 전송 지연이 가장 크게 나타났다. 반면, Draco 방식은 높은 압축률을 통해 데이터 크기와 전송 지연을 획기적으로 줄이는 데 성공하였으나, 인코딩과 디코딩 과정에서 상당한 처리 시간이 소요되어 수신 FPS가 급격히 저하되는 문제를 보였다.

특히 포인트 수가 증가할수록 모든 방식에서 수신 FPS 저하와 지연 증가가 선형적으로 나타났으며, 실시간성이 요구되는 환경에서는 압축률뿐만 아니라 인코딩 및 디코딩 지연까지 함께 고려해야 함을 확인할 수 있었다. 또한, 데이터 전송의 신뢰성 측면에서는 청크 단위 분할 전송 구조가 성능에 중요한 영향을 미쳤으며, 전송 청크 손실 시 복원 실패로 이어지는 구조적 한계도 존재함을 확인하였다.

향후에는 실제 네트워크 환경에서 발생할 수 있는 다양한 변수들, 예를 들어 전송 지연, 패킷 손실 등을 고려한 실험을 수행할 계획이다. 이를 통해 현재 시스템이 가진 한계를 보다 현실적인 조건에서 점검하고, 데이터 손실이나 지연에 대응할 수 있는 전송 방식 개선 방안을 도출하고자 한다.

본 연구는 WebRTC를 활용한 포인트 클라우드 실시간 전송 기술의 실용성과 제약을 실험적으로 규명하였으며, 향후 메타버스, 원격 협업, 실시간 3D 스트리밍 서비스 등 다양한 초실감형 응용 분야에서 핵심 기술로 활용될 수 있는 기반을 마련하였다는 점에서 의미가 있다.

References

[1] S. Kim, Y. W. Kim, K. J. Sun, and S. G. Yoo, “Trends in digital twin-based real-virtual fusion interaction technology for metaverse services,” *J. KICS*, vol. 40, no. 11, pp. 24-31, 2023.

[2] Y. Cheong, W. Jun, and S. Lee, “Study on point cloud based 3D object detection for

autonomous driving," *J. KICS,* vol. 49, no. 1, pp. 31-40, 2024. (https://doi.org/10.7840/kics.2024.49.1.31)

[3] S. J. Hwang, S. C. Park, Y. J. Jung, E. K. Kim, I. H. Yeo, and Y. H. Lim, "Patent trends of MPEG-I video coding standard," *J. Broadcast Eng.,* vol. 29, no. 2, pp. 187-197, 2024. (https://doi.org/10.5909/JBE.2024.29.2.187)

[4] Y. Lee, J. Sim, D. H. Kim, and D. You, "A comparison of serialization formats for point cloud live video streaming over WebRTC," in *Proc. IEEE ICCE,* pp. 123-126, Las Vegas, USA, Jan. 2024. (https://doi.org/10.1109/ICCE59016.2024.10444424)

[5] WebRTC Working Group, *WebRTC 1.0: Real-time communication between browsers*(2025), Retrieved May., 22, 2025, from https://www.w3.org/TR/webrtc

[6] F. Weinrank, M. Becke, J. Flohr, E. Rathgeb, I. Rungeler, and M. Tuxen, "WebRTC data channels," *IEEE Commun. Standards Mag.*, vol. 1, no. 2, pp. 28-35, Jun. 2017. (https://doi.org/10.1109/MCOMSTD.2017.1700007)

[7] Google Developers, *Protocol Buffers: Developer Guide*(2025), Retrieved May., 22, 2025, from https://developers.google.com/protocol-buffers

[8] J. H. Byun and D. K. Shim, "Introduction to DRACO 3D mesh compression technology," *Broadcast and Media*, vol. 28, no. 3, pp. 33-44, 2023.

[9] Google Inc., *Draco: 3D Data Compression Specification*(2017), Retrieved May., 26, 2025, from https://google.github.io/draco/spec/

이 용 준 (Yongjun Lee)

2019년 3월~현재 : 한남대학교 정보통신공학과 학석사통합과정

<관심분야> 실감미디어 통신, 저지연 스트리 밍

[ORCID:0009-0005-7924-6445]

정 한 희 (Han Hee Jung)

2013년 12월 : 일리노이 시카고 대학 전기공학 공학사

2019년 2월 : 대구경북과학기술원 로봇공학전공 공학석사

2023년 8월 : 대구경북과학기술원 로봇및기계전자공학 공학박사

2023년 11월~2024년 2월 : 대구경북과학기술원 박사후연구원

2024년 3월~현재 : 한남대학교 정보통신공학과 조교수

<관심분야> 생체 통합 전자장치, 웨어러블 기기 무선통신

[ORCID:0009-0006-4141-1254]

유 동 호 (Dongho You)

2012년 2월 : 서울과학기술대학교 매체공학 공학사

2014년 2월 : 서울과학기술대학교 미디어IT공학 공학석사

2018년 8월 : 서울과학기술대학교 방송정보통신융합공학 공학박사

2018년 9월~2021년 2월 : 독일 드레스덴공과대학교 도이치텔레콤연구그룹 선임연구원

2021년 3월~2025년 8월 : 한남대학교 정보통신공학과 부교수

2025년 9월~현재 : 국립공주대학교 컴퓨터교육과 부교수

<관심분야> 실감미디어 통신, 다중감각 통신, 저지연 통신 네트워크

[ORCID:0000-0003-3724-3244]