Journal of The Korea Society of Computer and Information
Vol. 18, No. 12, December 2013
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2013.18.12.035

웹캠과 스마트폰을 이용한 브로드 캐스팅 시스템 구현 및 성능 실험

김 정 명*, 박 근 덕*

Implementation and Performance Testing of a Broadcasting System using Webcams and Smartphones

Jeong-Myeong Kim*, Geun-Duk Park*

요 약

본 논문은 웹캠으로부터 추출한 영상을 Jpeg 이미지로 압축하여 다수 사용자의 스마트폰으로 브로드캐스팅하는 시스템 구현 방법을 제안한다. 제안하는 시스템은 가능한 많은 사용자에게 가능한 많은 프레임(frame)을 전달하기 위해, 사물 식별이 가능한 적절한 이미지 품질을 유지하면서 최소의 데이터 양을 사용하도록 구현하였다. 또한, 제안한 방법을 적용하여 영상정보를 제공하는 서버의 성능 테스트, 영상 정보를 받아 이를 스마트폰에 표시하는 클라이언트의 성능 테스트 및 지원 가능한 동시 접속 클라이언트의 수 등의 실험 결과를 제시하였다. 웹캠과 스마트폰을 이용하여 브로드캐스팅 시스템을 구현할 경우, 네트워크 성능에 따른 적합한 동시 접속자 수, 클라이언트 스마트폰의 성능, 초당 전송될 프레임 수 등 브로드캐스팅 시스템의 주요 요소를 추정하는데 본 실험 결과를 활용할 수 있다.

▸ Keywords : 웹캠, 스마트폰, 안드로이드, 소켓 통신, 브로드캐스팅

Abstract

This paper suggests a way to implement a system that broadcasts compressed JPEG images obtained from a webcam to multiple users on their smartphones. The system was implemented to maintain a suitable image quality so that things are identifiable while using the minimal amount of data, in order to deliver as many frames as possible to as many users as possible. Also, the suggested way was applied and various tests were done, including on: the performance of the server that provides image information; the performance of the client that receives the image information and displays it on the smartphone; and the max. number of simultaneous users supported by the system. When a broadcasting system is implemented using webcams and

•제1저자 : 김정명 •교신저자 : 박근덕
•투고일 : 2013. 10. 16, 심사일 : 2013. 10. 30, 게재확정일 : 2013. 11. 30.
* 호서대학교 컴퓨터공학과(Dept. of Computer Engineering, Hoseo University)
※ 본 논문은 한국산업기술진흥원 주관 지역산업기술개발사업의 지원을 받아 수행된 것임(2012-0513)

smartphones, the results of this paper can be used in estimating the suitable system parameters depending on network performance, including the max. number of simultaneous clients supported, the client smartphone performance required, and the number of frames that can be transmitted per second.

▸ Keywords : WebCam, Smartphone, Android, Socket Communication, Broadcasting

I. 서 론

웹캠(Webcam)은 USB, 이더넷, 와이파이 등을 통해 컴퓨터에 실시간으로 이미지를 공급할 수 있어 다양한 방면에서 사용되고 있다. 웹캠은 방송용 카메라만큼 고화질을 제공하지는 못하지만 저가형 캠코더와 비슷한 성능을 보이며 인터넷을 통하여 원거리 화상 전송이 가능하여, 화상 채팅 또는 개인 방송 등에 활용되고 있다. 또한, 웹캠은 보안 장치, 자녀 보호 등의 모니터링 용도로도 활용되고 있다. CCTV 카메라를 이용해 감시 시스템을 구축하려면 높은 비용이 필요한데, 웹캠은 상대적으로 저렴한 비용으로 감시 시스템을 구축할 수 있는 장점이 있어 중소규모의 업체나 개인 공간에 대한 모니터링 시스템 구축에 적합하다[1][2][3].

최근에는 영상정보 활용의 즉시성을 보장하기 위하여 웹캠으로 획득한 영상정보를 컴퓨터뿐만 아니라, 스마트폰으로 전송하는 시스템과 이를 응용하는 시스템에 대한 연구가 활발히 진행되고 있다[4][5].

이러한 웹캠 활용 분야에서는 적절한 영상 품질을 제공하는 것이 가장 중요한 요소이다. 웹캠에서 얻은 영상을 처리하여 전송하는 서버의 성능, 영상 데이터를 전송하는 네트워크의 성능, 전송받은 영상 정보를 클라이언트 화면에 재생하는 클라이언트의 성능 등에 의해 영상의 품질, 초당 전송되는 프레임의 수 (FPS, Frame per second) 및 동시 접속 가능한 클라이언트의 수 등이 결정된다.

본 논문은 웹캠으로부터 추출한 영상을 JPEG 이미지로 압축하여 TCP/IP 통신을 통해 스마트폰으로 전송하는 데 있어, 사물 식별을 할 수 있으면서 가능한 적은 데이터를 사용하는 이미지 품질을 찾고, 이를 여러 클라이언트에게 전송하는 방법을 제안한다. 또한 제안한 방법을 활용하여 영상정보를 제공하는 서버의 성능 테스트, 영상 정보를 받아 이를 스마트폰에 표시하는 클라이언트 성능 테스트 및 동시 접속 가능한 클라이언트 테스트 등의 실험 결과를 제시한다. 사용하는 네트워크의 성능에 따라 적합한 동시 접속자 수, 초당 전송될 프레임 수, 전송받은 영상을 재생하기 위한 클라이언트의 성능 등 브로드캐스팅 시스템 패러미터(parameter)들을 추정하는데 본 실험 결과를 활용할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 웹캠을 활용한 기존 연구 및 웹캠 영상을 처리하는 기술에 대하여 설명하고, 3장에서는 서버가 웹캠의 영상을 처리하여 브로드캐스팅하는 과정 및 클라이언트가 서버의 영상을 받아 사용자에게 보여주는 과정을 설명한다. 4장에서는 구현한 프로그램을 이용한 실험 결과를 제시하고, 5장에서는 결과 활용 방안 및 결론을 기술한다.

II. 관련 연구

2.1. 원격 감시 시스템

웹캠은 다양한 방법으로 활용할 수 있다. 임베디드 시스템을 이용한 감시시스템을 구현하거나, 웹서비스를 통해서 일정 간격으로 영상 정보를 보여주기도 한다. 그 중에서는 IP카메라를 이용해서 감시 시스템을 구축하기도 하는데, IP카메라는 독립적인 IP 주소를 가지고 있어 클라이언트에서 직접 카메라 영상을 요청하고 수신할 수 있기 때문에 별도의 컴퓨터 서버를 구축하지 않아도 원격 감시 시스템을 구축할 수 있다. IP카메라는 실제 산업에서 사용하는 CCTV 카메라에 비해서 저렴하지만 웹캠에 비해서 상대적으로 고비용이기 때문에 저예산으로 구축하기는 어려운 단점이 있다[6].

인터넷을 통해 웹캠의 영상 정보를 중계를 하는 방법은 여러 가지가 있다. 기본적으로 웹캠을 연결한 컴퓨터가 필요하다는 점은 동일하지만 전송 방법에 따라 다양한 방식으로 웹

캠의 화면을 사용자에게 보여준다.

첫 번째로 웹브라우저를 이용한 방법으로 웹캠의 화면을 주기적으로 갱신하여 사용자가 웹페이지에 접속하면 미들웨어 서버가 웹페이지를 몇 초 간격으로 갱신한다. 이 방식에서는 자연스러운 모습을 보여주기 위해 갱신주기를 짧게 설정할 경우 화면이 지나치게 빨리 갱신되어 한 화면이 다 표시되기 전에 다른 페이지가 갱신되는 문제가 생긴다. 따라서 웹기반 방식은 특성상 페이지 갱신이 초단위로 이루어진다는 한계를 가지고 있다[7].

두 번째로 웹캠 이미지를 받아오는 서버 이외에 추가적으로 원격지 사용자를 위한 PC 클라이언트 프로그램을 이용해 사용자에게 화면을 중계하는 방법이다. 이 방식은 PC의 특성상 네트워크가 유선으로 연결되어 있고 스마트폰에 비해 높은 성능의 무선 네트워크 환경을 제공하기 때문에 고화질과 동시에 부드러운 화면을 보여준다. 또한 클라이언트 프로그램이 따로 존재하기 때문에 화면 중계뿐만 아니라 화면 녹화, 동작 감시 등의 다양한 기능을 제공할 수 있다. 하지만 높은 품질을 제공하는 만큼 네트워크 트래픽이 많고 사용자 또한 PC를 이용해야만 웹캠 감시가 가능하다[8].

마지막으로 최근 급속도로 보급되어 많은 사람들이 사용하는 스마트폰을 이용한 방법이다. 스마트폰은 휴대가 가능한 기기로 언제든지 사용할 수 있어 감시 중에 발생하는 상황을 바로 파악할 수 있는 장점이 있다. 하지만 스마트폰은 무선 통신 환경을 사용하기 때문에 대부분이 유선 환경인 PC 클라이언트에 비해 상대적으로 불안정한 통신 환경을 가지고 있다. 그렇기 때문에 고화질의 화면을 중계하는데 어려움이 있고, 초당 보여줄 수 있는 프레임의 수에도 한계가 있다.

2.2. 영상처리

OpenCV는 인텔에서 개발하여 공개해온 오픈소스 기반의 강력한 비전(Vision) 라이브러리이다. 최적의 속도를 위해 최적화된 라이브러리이기 때문에 OpenCV에서 지원하고 있는 알고리즘의 경우 직접 구현하여 사용하는 것 보다 내부에 있는 알고리즘을 사용하는 것이 시간적으로나 노력면에서나 더욱 효율적이다[9]. OpenCV는 다양한 방면에서 사용되고 있는데 대표적으로 물체 인식과 인간과 컴퓨터 상호 작용(HCI) 방면에서 주로 사용된다.

JPEG는 ISO와 ITU-T에서 제정하였으며 정지화상을 위한 손실 압축 방법의 대표적인 표준 중 하나이다. 이 압축 표준은 16,777,216색과 256색 그레이로 저장할 수 있다. JPEG는 기본적으로 사람의 시각적 한계를 이용한 것으로 사람의 시각은 휘도 신호(luminance)에는 민감하지만 색차 신호에는 둔감하다는 점과, 인접한 픽셀간의 픽셀 값이 급격히 변하지 않는다는 속성을 이용한다. 따라서 JPEG는 고주파 자료가 제거되어 나타난다. JPEG의 압축은 5단계의 과정을 거치는데 색공간 변환 - 크로마 서브샘플링 - 이산 코사인 변환 - 양자화 - 엔트로피 부호화의 단계를 거치며 위 과정을 반대로 진행하면 디코딩이 가능하다[10][11]. JPEG 압축은 손실 압축 형식이지만 높은 압축률을 제공하는 장점 때문에 웹 또는 사진 보관용 등으로 널리 사용되고 있다.

III. 브로드 캐스팅 시스템의 구현

3.1. C++ / 자바 소켓 통신

소켓 통신은 동일한 기종뿐만 아니라 서로 다른 기종의 시스템을 가지고 있어도 시스템에 상관없이 서버/클라이언트 환경을 구축하여 데이터를 전송하기 위한 통신 프로토콜이다. 소켓 통신은 신뢰성 프로토콜인 TCP(Transfer Control Protocol) 방식과 비신뢰성 프로토콜인 UDP(User Datagram Protocol)로 나뉜다. 영상과 같은 스트리밍 서비스는 보통 UDP를 사용하여 속도를 향상하지만 본 시스템에서는 JPEG 압축을 이용하여 이미지 단위로 상태를 갱신하기 적합한 TCP 방식을 사용한다[12].

본 논문은 C++ 서버와 자바(JAVA) 클라이언트들 간의 소켓 통신을 구현하는데 동작방식이 다른 언어간 통신 구현이기 때문에 몇 가지 제한이 생긴다.

첫째로 자바에서 제공하는 객체 스트림을 사용할 수 없다. 대표적으로 ObjectInputStream / ObjectOutputStream을 들 수 있는데 이들은 자바 프로그램 사이에서 빠른 개발을 위해 사용하는 고유한 형식을 사용하기 때문에 객체에 대한 정보가 함께 전송되어 C/C++ 프로그램 측에서 적용시키기엔 어렵기 때문이다. 따라서 자바와 C++ 사이의 통신을 위해서 기본 유형으로 데이터를 변화시키는 작업이 필요하다. 예를 들어, [그림 1]과 같이 대응되는 형식으로 구성된 C++ 구조체와 자바 클래스를 사용하여 데이터를 주고받는다고 하면 여러 가지 문제가 발생한다. 먼저 C++과 자바의 기본 유형의 데이터 길이가 다른 문제점이 있다. C++의 char 형식은 1바이트를 가지고 있는데 반해 자바의 char 형식은 기본적으로 유니코드를 취하기 때문에 2바이트를 가지고 있다. 본 시스템에서는 자바의 char형을 byte형으로 변경하여 이 문제를 해결하였다. 다음으로 ch 변수의 취급 방법이 C++과 자바가 서로 다르다 C++은 중간에 char가 들어가

```
typdef struct {
  char dat[100];

  char ch;
  bool bl;
  char* string;
}
```

```
class cData{
  char[] dat =
    new char[100];
  char ch;
  boolean bl;
  String str;
}
```

그림 1. C++ 데이터(좌), 자바 데이터(우)
Fig. 1. C++ Data(left), Java Data(right)

게 되면 연산 단위인 4바이트를 맞추기 위해 나머지 3바이트를 덧붙이게 되기 때문에 서로 다른 언어 사이의 소켓 통신에 문제가 발생한다. 그리고 char* 와 string 형식의 문자열을 저장하는 변수는 위에 언급한 바와 같이 자바는 유니코드를 기본으로 사용하고 C++은 ASCII 코드를 기본으로 사용하기 때문에 이를 통일하는 작업이 필요하다.

두 번째로 기본형 변수의 바이트 길이가 문제가 될 수 있다. 자바의 경우는 자바 가상 머신을 통해 작동하기 때문에 모든 자바 응용 프로그램의 기본형 크기가 같지만, C++의 경우 CPU와 컴파일러에 의존적이기 때문에 CPU에 따라 int 형이 4바이트가 될 수 도 있고 8바이트가 될 수도 있다. 따라서 이러한 특성을 고려하여야 한다.

세 번째로 기본형 데이터의 바이트 순서가 다를 수 있다. 자바는 Big Endian으로 고정되어서 사용되고 있지만 C++은 언급한 바와 같이 컴파일러, CPU에 의존적이기 때문에 이를 보정하여야 한다.

3.2 브로드캐스팅 서버 구현

본 절에서는 웹캠의 정보를 다수의 안드로이드 폰으로 전송하기 위한 브로드캐스팅 서버에 대해 기술한다. 서버 측은 C++로 작성되었으며, 두 개의 스레드가 동시에 작동된다.

하나의 스레드는 클라이언트와 소켓통신을 확립하기 위한 것으로 서버 구동 시에 항상 포트를 개방하고 클라이언트의 접속을 기다리며, 클라이언트로부터 메시지가 오면 이미지를 클라이언트에게 전송한다. 클라이언트에게 전송하는 이미지의 용량은 JPEG 포맷을 사용하기 때문에 상황에 따라 용량이 달라진다. 따라서 이미지를 전송하기 전에 이미지의 크기를 전송하여하며, 그 이후에 이미지를 전송한다.

서버는 브로드 캐스팅을 위해 100대 까지의 클라이언트의 접속을 허용한다. 즉 동시 접속 인원의 수를 의미한다. 이벤트 핸들은 각 클라이언트가 접속할 때마다 한 개씩 할당되며 접속이 끊어질 경우 할당된 핸들을 해제한다.

다른 하나의 스레드는 웹캠을 제어하기 위한 것이다. 웹캠이 작동해야지만 서버가 작동하는 것이 의미가 있기 때문에 웹캠의 동작을 중지하면 동시에 서버도 중지된다. 그 후 전송이 활성화된 핸들에 한하여 웹캠 이미지를 일정 주기로 계속해서 전송하는 스레드를 생성시킨다. 이도 메인 서버와 마찬가지로 웹캠이 종료되면 같이 종료된다.

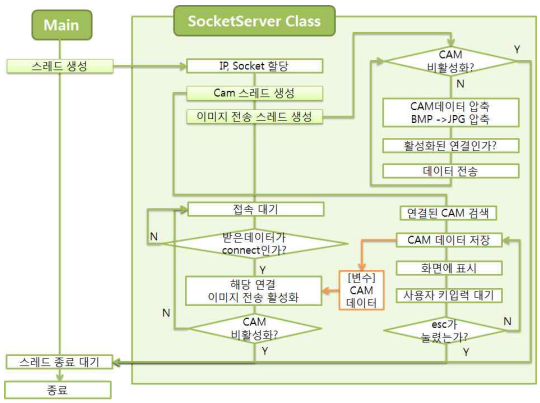

그림 2. 서버 흐름도
Fig. 2. Flowchart of a Server

클라이언트로부터 "connect" 요청이 오면 서버는 해당 연결에 대해 웹캠의 이미지를 전송하도록 전송 여부를 활성화한다. 메인 서버 스레드의 역할은 "connect" 요청이 오는 클라이언트들에 대한 전송 활성화 작업을 하며, 연결이 정상적으로 종료되거나 강제로 끊긴 클라이언트들에 대해서 소켓 종료 작업을 수행한다. 즉, 클라이언트 전체의 접근 제어를 수행한다. 또한 전송 스레드는 활성화된 클라이언트들에 한해 캠 이미지를 일정 간격으로 계속 전송한다. 웹캠 이미지는 한 프레임을 전송할 때 마다 한번만 수행되며 그 후 현재 활성화된 클라이언트에게 이미지를 전송한다. 전송되는 이미지의 크기는 임의로 변경할 수 있지만 전송할 때의 이미지 용량과 JPEG 압축 후 사물의 구분 여부를 위해 640 X 480 해상도로 이미지를 가져온다. 이 상태는 비압축 상태로 그대로 저장 시 BMP이미지와 비슷한 크기를 갖는다. 비압축 이미지의 데이터 크기(I)는 해상도가 M(너비)*N(높이)인 경우 식(1)과 같이 구할 수 있다[13].

$$I = M \times N \times 8 \times 3 \ (bit) \qquad (1)$$

(1)식을 통해 640 X 480 해상도를 가진 비압축 이미지의 크기는 7,372,800bit, 약 0.9MB의 용량을 가지게 된다. 해당 이미지를 JPEG 포맷을 통해 용량을 줄이고, JPEG의 압축률을 설정하여 용량을 더 미세하게 조절할 수 있다.

웹캠 스레드는 웹캠 화면을 계속해서 서버PC의 화면에 갱신해서 보여주는 작업을 수행한다. 화면 갱신 작업 중에 서버 스레드로부터 이미지 요청을 받으면 웹캠 스레드는 갱신 중이던 이미지를 JPEG 포맷으로 압축하고, 압축된 이미지의 길이와 압축된 이미지를 전달한다. 서버 스레드는 첫 4byte에 이미지의 길이를 넣고 그 뒤로는 이미지 데이터를 추가하여 클라이언트로 전달한다. 이 이미지 전달 작업은 "connect" 메시지를 보낸 온 모든 클라이언트에게 순차적으로 진행된다.

3.3 스마트폰 클라이언트 구현

클라이언트 프로그램은 안드로이드 애플리케이션으로 제작되었다. 사용자가 서버에 접속을 시작하면 서버로부터 받아온 이미지는 연속적으로 UI 스레드를 통해 갱신된다.

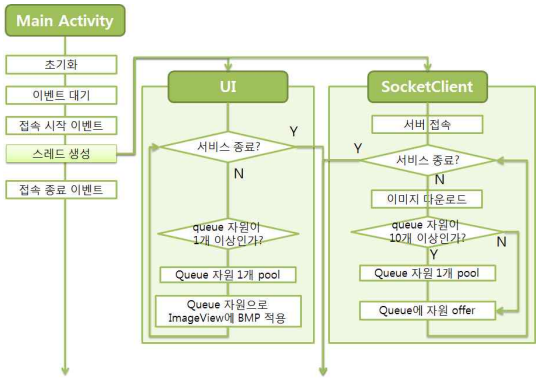

그림 3. 클라이언트 흐름도
Fig. 3. Flowchart of Clients

클라이언트 측 소켓 통신은 서버와 연결 후 "connect" 메시지를 전달하고, 서버 측의 응답을 기다린다. 서버로부터 받은 데이터는 BIG_ENDIAN으로 설정하여 데이터를 수신하도록 설정하고, 먼저 데이터 길이를 얻기 위해 4byte int 데이터를 얻어 총 이미지 길이를 설정한 후 이미지 길이만큼 이미지를 ByteArrayOutputStream에 저장한다. 전체가 저장된 이미지는 바이트 배열에 저장하여 이미지로 변환하고, UI 스레드를 통해 이미지 갱신을 요청한다. 이 때 이미지를 받아오는 다운로드 작업과 이미지 갱신 작업은 분리되어서 두 개의 스레드로 동작한다. 다운로드 스레드는 먼저 "connect" 요청 후서버에서 일정간격으로 보내주는 이미지를 큐(Queue)에 계속해서 저장한다. 하지만 큐에 이미지를 쌓는 속도보다 이미지를 갱신하는 속도가 느릴 경우 큐에 계속해서 이미지가 쌓일 수 있다. 그럴 경우 사용자가 보는 이미지는 점점 지연이 길게 생기기 때문에 큐는 지연율을 낮추기 위해 10개의 한도를 설정하여, 10개 이상이 큐에 쌓이는 경우 가장 오래된 이미지를 큐에서 삭제한다. 다운로드 스레드와 동시에 이미지 갱신 스레드는 큐에 이미지가 1개 이상 존재하면 큐에서 이미지를 가져와서 사용자의 ImageView에 계속해서 갱신해준다.

안드로이드 시스템에서 액티비티의 메인 스레드를 사용하지 않고 별도의 스레드를 사용하여 이미지를 갱신할 경우, 해당 스레드에는 View 제어 권한이 없기 때문에 View의 데이터를 변경해도 View의 상태는 변하지 않는다. 따라서 UI 스레드의 핸들러를 이용하여 View의 갱신을 요청하는 단계가 필요하다. 사용자가 서버와 연결을 중단하거나 연결이 끊어지면 소켓을 닫고 소켓통신 및 이미지 갱신 스레드를 종료한다.

IV. 브로드캐스팅 시스템 실험

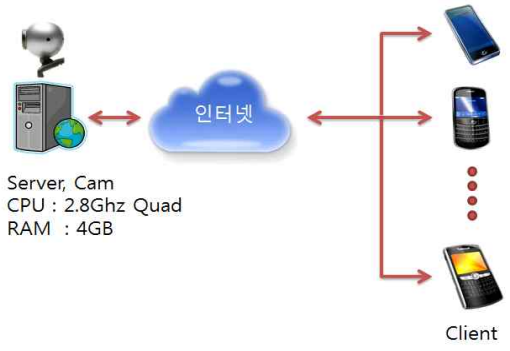

그림 4. 실험 환경 구성
Fig. 4. Configuration of the Experimental Environment

본 절에서는 구현한 브로드캐스팅 시스템을 활용하여 영상 정보를 제공하는 서버의 성능 테스트, 영상 정보를 받아 이를 스마트폰에 표시하는 클라이언트 성능 테스트 및 동시 접속 가능한 클라이언트 테스트를 수행하고 그 결과를 제시한다.

실험 환경은 서버로 2.8GHz QuadCore CPU와 4GB RAM을 가진 PC, 클라이언트로 낮은 사양과 높은 사양의 스마트 폰, 무선 통신 환경은 WiFi로 실험하였다.

JPEG는 0%~100% 범위의 이미지 품질을 결정할 수 있는데, 이는 640 X 480 해상도를 기준으로 5KB~200KB의 용량을 가지게 된다. 너무 품질이 낮을 경우 용량이 작아 전송량을 줄일 수 있지만 사물과 사물간의 경계가 모호해서 사물 식별이 어려울 수 있으며, 품질을 높게 하면 사물이 정확하게 보이지만 선명해지는 정도에 비해서 용량이 상당량 증가

한다. 적정 수준의 이미지 품질을 찾기 위해 5 ~ 50%의 품질을 가진 이미지를 추출하여 각 이미지의 용량과 사물 구분 여부를 보관함과 모니터 등이 놓여있는 [그림 5]와 같이 비교하였다. 5% 품질로 이미지를 추출할 경우 사물의 색상이 단순화되어서 전체적인 이미지가 뭉개지는 문제가 발생했다. 그리고 10% 품질의 경우는 비슷한 색상인 경우 사물 경계를 구분하기 어려웠다. 20%~ 50%의 품질 이미지는 사물을 구분하기에 적절한 화질로 판단되며, 20% 품질은 16KB, 50% 품질은 27KB로 데이터 크기에서 효율적인 20% 품질 이미지를 사용하는 것이 효율적인 것으로 파악되었다.

실험에 사용할 서버는 PC 성능 때문에 전송에 지연이 발생하지 않도록 적절한 사양의 PC를 사용하였는데, CPU는 Intel i5-2300(2.8GHz), RAM은 4GB이다. 서버에서 사용하는 이미지는 20% 품질의 이미지를 전송하도록 하고, 서버에 동시에 접속하는 클라이언트 기기의 수에 따라 전송 품질이 어떻게 변화하는지 알아보기 위해 동시 접속 클라이언트의 개수를 조절하면서 서버의 전송 횟수 및 전송 트래픽 발생 정도를 측정하였다.

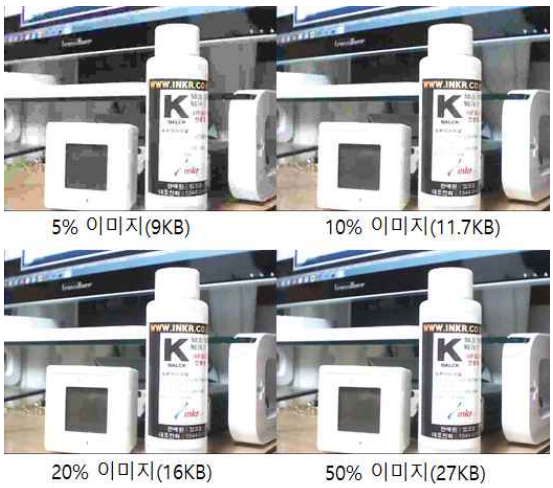

그림 5. 이미지 압축률 비교
Fig. 5. Comparison of Image Compression Rate

[표 1]은 동시접속기기의 숫자에 따른 서버의 전송 결과 측정 표이다. [표 1]에서 AVG는 1초에 전송한 이미지 수의 평균값이며, MAX는 최대값, MIN은 최소값을 의미한다.

표 1. 서버 전송 실험
Table 1. Performance Testing of a Server

동시 접속		1	10	20
Send Count	AVG	37.18	36.28	36.18
	MAX	38	37	37
	MIN	36	32	35
Send Hit	AVG	36.98	35.72	35.9
	MAX	38	36.7	36.85
	MIN	35	31.6	34.35
Send Fail	AVG	0.2	0.56	0.31
	MAX	2	1.3	0.65
	MIN	0	0.3	0.05
Total Send Byte	AVG	304291.8	2938550	5871857
	MAX	312976	3020237	6030158
	MIN	287709	2599131	5620804
동시 접속		40	50	100
Send Count	AVG	35.88	35.62	35.43
	MAX	37	37	36
	MIN	34	32	34
Send Hit	AVG	34.52	29.53	14.08
	MAX	36.03	30.44	14.45
	MIN	32.7	27.5	13.68
Send Fail	AVG	1.36	6.08	21.4
	MAX	2.65	6.82	21.9
	MIN	0.85	4.5	20.3
Total Send Byte	AVG	11006251	11766956	11831271
	MAX	11489554	12125309	12140429
	MIN	10432725	10952586	11492228

Send Count는 접속된 클라이언트로 초당 이미지를 보내는 횟수, Send Hit는 Send Count중에서 전송에 성공한 횟수, Send Fail은 전송 실패 횟수, Send Byte는 클라이언트에 보낸 데이터 크기의 총합을 의미한다. 여기서 Total Send Byte 항목 외에는 클라이언트 1대당 전송되는 평균 데이터를 의미한다. 서버 전송 속도 테스트에서 1대가 접속했을 때 전송 성공 횟수는 초당 평균 36.98회로 측정되었다. 동시 접속이 40대인 경우 서버의 대역폭인 100Mbps에 거의 근접한 Send Fail 수치가 약간 증가하였지만 초당 전송 성공 횟수는 34.42로 적절한 초당 프레임 수를 보였다. 하지만 동시 접속이 50대가 되면서 서버의 대역폭을 넘는 데이터를 전송해야하는 상황이 발생하여, 40대 일 때 1대당 평균 1.36개가 발생하던 Send Fail이 6.08로 급격하게 증가하였고 100대의 경우엔 21.4까지 증가하였다. 즉 각 클라이언트에게 초당 14회의 전송 속도를 보였다.

클라이언트 전송 실험에는 여러 스마트 폰을 이용하여 WiFi 무선통신 환경에서 전송되는 프레임의 수와, 각 스마트 폰의 갱신 속도 비교하였다. 실험에 사용한 스마트 폰의 성능은 고성능(DualCore 이상), 저성능(SingleCore)로 나누어서 실험했다.

표 2. WiFi 실험 환경 속도 측정
Table 2. Speed Test of WiFi Environment

No	Latency(ms)			Up (bps)	Down (bps)
	MIN	MAX	AVG		
1	6	207	39.2	11.2	16.6
2	7	425	62.1	1.2	20.1
3	7	606	85.6	8.2	13.3
4	7	725	82	13.6	17.4
5	7	408	65.2	4.8	12.0
6	7	606	79.7	20.0	16.5
7	7	505	104	18.5	22.7
8	6	821	104	10.3	18.1
9	7	293	41.3	10.8	16.2
10	7	412	54.1	14.7	20.1
Total	6.8	500.8	71.7	11.3	17.3

먼저 정확한 실험 결과를 알기 위해 BenchBee 애플리케이션을 이용해 고성능의 스마트 폰에서 WiFi 속도를 체크하였다. 일정 횟수를 반복으로 실험하고 평균 전손송도를 측정하였다. 실험은 10회 진행하였고 평균 지연 속도는 71.7ms로 측정되었고 업로드 속도는 11.3Mbps (1.42MB/s)이고, 다운로드 속도는 17.3Mbps(2.16MB/s)로 측정되었다.

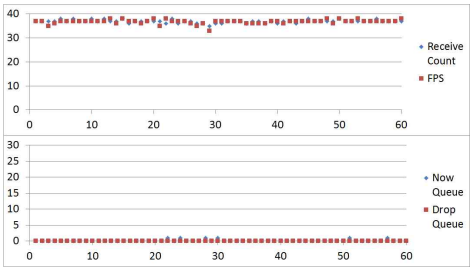

그림 6. 고성능 클라이언트 실험 그래프
Fig. 6. Experimental Results of Clients with High Spec.

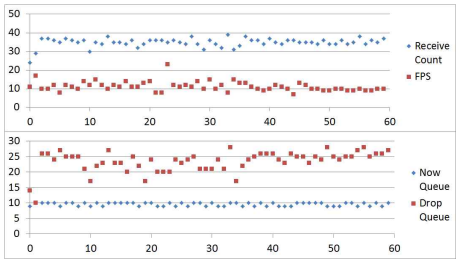

그림 7. 저성능 클라이언트 실험 그래프
Fig. 7. Experimental Results of Clients with Low Spec.

클라이언트의 성능에 따라 수신 받는 이미지의 수와 화면 갱신 횟수를 측정하였으며, 초당 전송되는 이미지를 1회로 하여 실험 마다 60회씩 측정하였다. 실험결과는 〔그림 6〕〔그림 7〕에 제시되어 있으며, 각 그림은 두 개의 그래프로 구성되어 있는데 클라이언트에서 이미지를 받는 횟수와 화면 갱신 횟수는 첫 번째 그래프에, 이미지 데이터 큐에 들어있는 이미지 개수와 큐가 가득 차서 버려지는 이미지의 개수 두 번째 그래프에 제시되어 있다.〔그림 6〕〔그림 7〕은 각각 고성능, 저성능의 스마트 폰을 사용한 실험 결과를 나타낸다. 네트워크의 일시적인 오류로 인하여 간헐적으로 발생하는 현저하게 낮은 수치는 실험 결과에서 제외하였다.

표 3. 클라이언트 성능 실험
Table 3. Performance Testing of Clients

성능		고성능	저성능
Recv Count	AVG	36.933	33.5
	MAX	38	39
	MIN	35	24
FPS	AVG	36.78	10.82
	MAX	38	23
	MIN	33	4
Queue (amount/Drop)	AVG	0.1/0	9.2667/22.3
	MAX	1/0	10/28
	MIN	0/0	9/10

고성능의 스마트 폰은 안정적인 이미지 수신 및 이미지 갱신 결과를 보였으며 초당 평균 갱신되는 이미지 수는 36.78로 측정되었다. 큐의 데이터는 거의 즉시 처리되어 평균 0.1개의 낮은 큐 크기를 보였고, 큐가 가득차서 버려지는 데이터는 전혀 없었다. 저성능의 스마트 폰은 고성능 스마튼 폰과는 달리 일정하게 낮은 FPS를 보여서 평균 10.82프레임이 측정되었고, 전송 받은 데이터에 비해 처리량이 적어 최대 10개의 크기를 가진 대기 큐는 평균 9.2의 크기를 보였으며, 초당 평균 22.3개의 이미지가 무시되었다. 따라서 single core를 사용하는 저성능 스마트폰은 전송받은 프레임의 상당량을 활용하지 못하는 것으로 파악되었다.

표 4. 클라이언트 동시 접속 테스트
Table 4. Testing of Simultaneous Clients

동시접속	2	3	4	5	6
AVG	37.2583	37.0333	37.0556	36.6667	36.3444
MAX	38	40	40	39	40
MIN	35	28	28	27	28
STDEV	0.66453	1.36178	1.37728	1.77639	1.91143

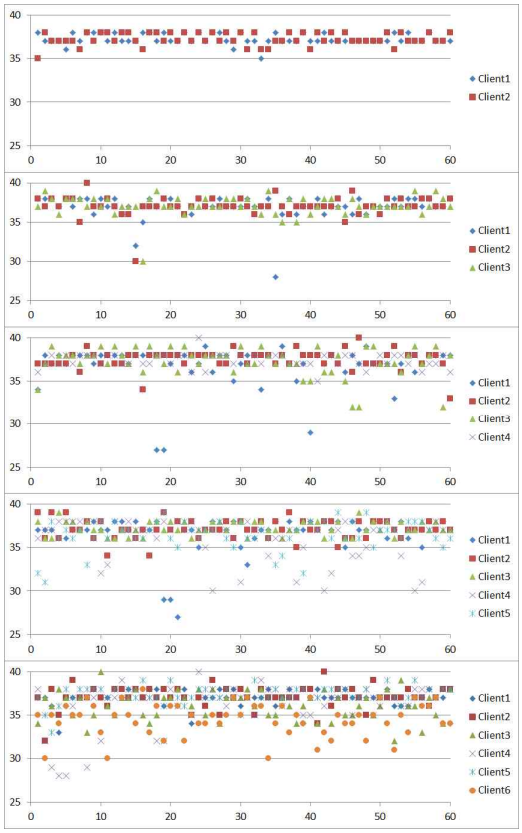

그림 8. 클라이언트 동시 접속 테스트
Fig. 8. Testing of Simultaneous Clients

또한, 서버에 여러 대의 스마트 폰이 접속하는 동시 접속 실험을 진행하였는데, 2~6대의 클라이언트가 서버에 접속하여 이미지를 받아오는 횟수를 측정하였다. 클라이언트 스마트폰은 고사양 4대, 저사양 2대를 실험에 사용하였다.

2~6 대의 클라이언트가 동시 접속할 경우 수신하는 초당 이미지 수는 〔표 4〕에 제시되어 있다. 〔표 4〕에서 STDEV는 표준 편차를 나타내고, 각 데이터는 한 대당 수신하는 이미지 프레임의 수를 나타낸다.

〔그림 8〕에서 X축은 진행 시간을 나타내고 Y축은 이미지 수신 횟수를 나타낸다. 〔그림 8〕은 위에서부터 순서대로 2~6대의 동시 접속 실험 결과를 나타낸다. 〔그림 8〕〔표 4〕는 2~6대 동시 접속의 경우 모두 30회 이상의 이미지 수신 횟수를 보이며 평균 37회 전송하는 서버의 전송을 정상적으로 수신하며 동시접속이 많아질수록 표준편차가 커지는 것을 확인할 수 있다. 하지만 6대 동시 접속 시에도 이미지 수신이 36회 이상 안정적인 모습을 보이는 결과를 얻었다. 이는 서버에서 보내주는 이미지 속도가 WiFi 무선 통신 환경의 다운로드 속도 이하라면, 전송되는 이미지의 수는 동시 접속 대수에 관계없이 일정하다는 것을 의미한다.

이상의 실험을 통해 결과적으로 본 논문에서 구현한 브로드캐스팅 시스템을 이용할 경우, 50대의 클라이언트에게 초당 30프레임 수준의 이미지를 전송하고 100대의 경우 14 프레임 수준의 전송 속도를 보일 수 있음을 확인하였다. 또한 DualCore 이상 고성능 스마트폰의 경우 수신한 대부분의 프레임을 화면에 표시할 수 있으며, SingleCore를 사용하는 저성능 스마트폰의 경우 약 9프레임의 이미지를 화면에 표시할 수 있음을 확인하였다. 따라서 서버의 초당 전송되는 프레임 수를 1로 설정할 경우, 1400대 이상의 클라이언트를 지원할 수 있을 것으로 예상된다.

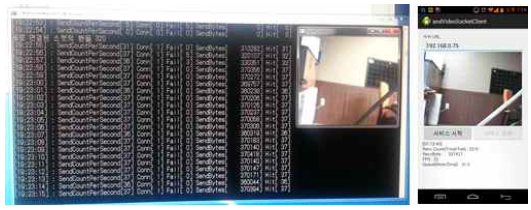

그림 9. 실행 화면
Fig. 9. Run Screen

〔그림 9〕은 본 논문에서 구현한 브로드캐스팅 시스템의 실행화면이며, 서버측에서 현재 접속 상황과 전송 상황을 1초마다 갱신 함과 동시에 현재 캠의 이미지를 실시간으로 보여주고, 클라이언트측은 앞에서 기술한 바와 같이 20% 이미지 품질로 이미지를 전송한 결과를 보여준다.

V. 결론

본 논문에서는 웹캠을 이용하여 여러 안드로이드 스마트폰에 영상정보를 전달하기 위한 브로드캐스팅 시스템을 구현하였다. 본 시스템은 사물 식별이 가능한 범위에서 최소의 데이터를 사용하여, 초당 이미지 전송 횟수와 동시 접속 기기수를 가능한 최대로 지원하도록 하였다. 무선 통신 환경과 저사양의 스마트 폰을 고려하여 5~10프레임 정도로 전송을 제어한다면 1분에 3~6MB 정도의 적은 용량으로 실시간 영상을 제공할 수 있다.

기존의 연구는 원격지의 대상을 웹이나 스마트폰으로 확인하는 것이 목적이어서 초당 0.5~2 프레임 수준의 갱신율을 제공하지만, 본 논문은 이를 크게 개선하여 초당 35~40 프레임의 갱신율을 제공한다. 또한 기존 연구에서 와는 달리 동

시 접속 사용자를 지원을 고려하여 설계되었기 때문에, 초당 1 프레임을 전송할 경우 약 1400 사용자를 동시에 지원할 수 있음을 실험 결과로 제시하였다.

본 논문에서 구현한 시스템을 이용하여 웹캠을 설치해 보안 시스템을 구축하여 원하는 장소를 모니터링할 수 있고, 동·식물 등의 상황을 확인하는 용도로 활용할 수 있다. 또한 다수 접속자를 대상하는 실시간 방송을 저렴한 비용으로 구축할 수 있다. 또한 본 논문에서 제시한 실험결과를 바탕으로 네트워크 환경과 스마트폰의 성능에 따라, 동시 접속 가능한 클라이언트 수와 초당 전송 가능한 프레임 수를 추정하여 브로드캐스팅 시스템을 설계하는데 활용할 수 있다.

참고문헌

〔1〕 Nam-Ho Kang, "An Implementation of Embedded Monitoring System with USB Webcam", the Korean Institute of Information and Commucation Sciences, Proceedings of KIMICS Sprint Conference, pp. 632-635, May. 2010.

〔2〕 Dae-Won Park, "Implementation of Intrusion Detection System Using Internet Camera", Proceedings of The 30th KISS Fall Conferences, Vol. 30, No 2, pp. 244-246, Oct. 2003.

〔3〕 H. C. Kim, W. S. Kim, D. H. Im, G. D. Park, "Wireless Security System Incorporating Tilt Sensors and Web Carmers", in Proc. ICISA 2013, pp. 69-70, 2013.

〔4〕 Watchdoing , https://www.watchdoing.com

〔5〕 Jenaus Cam, http://www.jenauscam.com/

〔6〕 Y. Gu, M. J. Kim, Y, Gui, H. K. Lee, O. K. Choi, M. W. Pyeon, J. I. Kim, "Design and Implementation of UPnP-based Surveillance Camera System for Home Security", in Proc. ICISA 2013, pp. 104-107, 2013.

〔7〕 Chul Nam, "*Development of a Real-Time Video Image Broadcasting System Using an USB Camera on Embedded Linux*", JOURNAL OF THE INDUSTRIAL TECHNOLOGY INSTITUTE, Vol. 12, pp. 137-140, Dec. 2004.

〔8〕 Jae-Heung Shin, "Development on the System for Real-Time Image Information Transfer and Store", The Transactions of The Korean Institute of Electrical Engineers, Vol. 10, pp. 194-198, Oct. 2006.

〔9〕 OpenCV, http://opencv.org/

〔10〕 JPEG, http://www.jpeg.org/jpeg/

〔11〕 JPEG Wiki, http://ko.wikipedia.org/wiki/JPEG

〔12〕 Youngdoo Kim, "Realtime multimedia contents display system in Iphone", the Korean Institute of Information and Commucation Sciences, Proceedings of KIMICS Sprint Conference, Vol. 14, No. 1, pp. 585-588, May. 2010.

〔13〕 Gwang-Bum Park, "Design of Webcam Monitoring System Using Embedded linux", Korea Information Processing Society, Journal of KIPS. Vol. 9, No. 2, pp. 2281-2284, Nov. 2002.

저 자 소 개

김 정 명

2012: 호서대학교 컴퓨터공학과 공학사.
현 재: 호서대학교 컴퓨터공학과 석사과정
관심분야: 모바일 컴퓨팅, 네트워크
Email : kimjm86@gmail.com

박 근 덕

2005: 서울대학교 전기컴퓨터공학부 공학박사
현 재: 호서대학교 컴퓨터공학과 부교수
관심분야: 웹공학, 모바일 컴퓨팅, ML 응용
Email : gdpark@hoseo.edu