실시간 영상감시를 위한 JPEG Streamer의 설계와 구현

A Design and Implementation of JPEG Streamer for Real Time Image Surveillance System

김 경 환* 유 해 영** 이 진 영***
Kyung-Hwan Kim Hae-Young Yoo Jin-Young Lee

요 약

네트워크 인프라의 성장과 디지털 영상압축 기술의 발달로 네트워크 카메라 서버를 이용한 실시간 영상감시 시스템의 수요가 증가하고 있다. 기존의 CCTV를 이용한 실시간 영상감시에 비해 네트워크 카메라 서버를 이용한 영상감시는 많은 이점이 있다. 본 논문에선 실시간 영상감시 시스템의 핵심 모듈로서 JPEG 영상의 수집과 전달을 담당하는 JPEG Streamer의 모델을 제시하고 이를 설계, 구현한다. JPEG Streamer의 안정성과 효율성을 위해서 쓰레드 풀과 공유 메모리를 사용했다. 실시간 영상의 품질을 높이기 위해서 더블 버퍼링의 개념을 도입했다. 이렇게 구현된 JPEG Streamer를 실시간 영상감시 시스템의 개발에 이용함으로써 생산성을 높일 수 있을 뿐 아니라, 신뢰성과 안정성도 확보 할 수 있다.

Abstract

Recently, network infra grows rapidly and the digital image compression technique have made remarkable progress, and therefore the demand of the real-time image surveillance system which uses a network camera server is increasing. Network Camera Server has emerged as an attractive alternative to the CCTV for the real-time image surveillance. From this reason, the demand regarding the development of the real-time image surveillance system which uses a network camera server is increasing as well.

In this paper, we will provide a model for JPEG Streamer which is used in real-time image surveillance system. And we will design and implement this model. For the stability and efficiency of the JPEG Streamer, we'll use the thread pool and shared memory. We will also introduce the concept of double buffering for increasing the quality of real-time image.

Using the JPEG Streamer, a raising of the productivity, reliability and stability will be able to secure to development of real-time image surveillance system.

☞ Keyword : digital image compression, JPEG Streamer, real-time image surveillance system

1. 서 론

최근 네트워크 인프라의 급성장과 디지털 영상압축 기술의 발달로 기존 실시간 영상감시 솔루션의 중심을 이루던 CCTV제품군들이 아날로그 영상을 JPEG 포맷으로 압축해서 전송해주는 Network Camera Server(NCS) 제품들로 대체되고 있다[6,9].

실시간 영상감시의 응용분야는 매우 다양하며 가장 핵심이 되는 요소는 네트워크를 통하여 실시간으로 JPEG 영상을 전송하는 영상전송 모듈이다. 그러나, 대다수의 NCS 벤더들이 제공하는 컴포넌트들은 여러 가지 제약 때문에 다수의 채널을 감시하기 어려워 일반적으로 최대 4개 채

* 정 회 원 : (주)동원S&S 연구소 소장
khkim@dongwonsns.com
** 정 회 원 : 단국대학교 정보컴퓨터학부 교수
yoohy@dankook.ac.kr
*** 정 회 원 : 강남대학교 교양학부 조교수
goodman3@kangnam.ac.kr
[2005/11/28 투고 - 2005/12/02 심사 - 2005/12/05 심사완료]

널까지의 영상만을 지원하고 있다. 그러나, 실시간 영상감시 시스템은 16개 채널 또는 그 이상을 감시의 대상으로 삼는 경우가 많다. 따라서 실시간 영상감시 시스템을 구성하기 위해서는 2개 이상의 NCS를 병렬로 구성하고 이들을 제어하기 위한 새로운 모듈을 개발해야 한다. 또 다른 제약은 Embedded OS를 탑재한 NCS들은 한정된 자원으로 인해 지원하는 동시사용자의 수가 5명을 넘지 못한다는 것이다. 이 또한 실시간 영상감시 시스템이 요구하는 수준의 동시사용자 수를 서비스하기 위해선 NCS의 능력(자원)을 어떤 식으로든 증폭시켜주어야 한다. 그러므로 실시간 영상감시 시스템에 대한 최근의 요구사항들을 만족시키기 위해서는 대안이 필요하다.

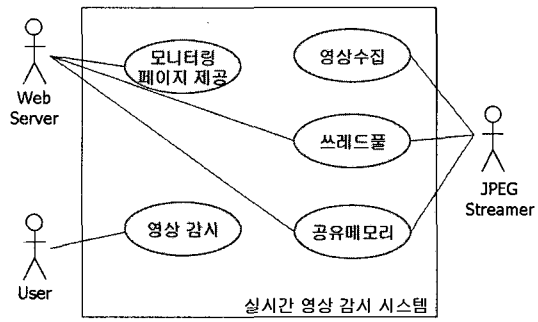
본 논문에서는 이에 대한 대안으로 실시간 영상감시 시스템을 구성하는 핵심 컴포넌트 중 하나인 JPEG Streamer의 모델을 제시하고, 이를 설계, 구현하고자 한다. 실시간 영상감시 시스템의 개발과정에 이 JPEG Streamer를 사용함으로서 개발자는 사용자 인터페이스에 집중할 수 있고, 그 결과 전체 시스템 개발을 단순화하고, 생산성을 향상시키며 실시간 영상감시 시스템의 신뢰성을 높이고 기존의 NCS가 갖는 몇몇 제약사항들을 극복할 수 있다[7].

또한 본 논문에선 실시간 영상감시 시스템을 설계하는 과정에서 시스템이 운영될 서버의 용량계획에 도움을 주고자 네트워크 사용량 추정 모델과 메모리 사용량 추정 모델을 제안한다. 이 자원사용량 추정모델과 JPEG Streamer 모델을 통해 다양한 환경의 실시간 영상감시 시스템을 모의실험해 볼 수 있을 것이다.

2. JPEG Streamer의 설계

2.1 요구분석

그림 2.1은 웹기반 실시간 영상감시 시스템을 구성한 Use Case 다이어그램으로 JPEG Streamer의 모델에 대한 요구사항을 설명한다. 실시간 영상감시 시스템의 actor들과 연관된 쓰임새들은 영상을 감시하는 사용자, 실시간 영상을 수집하고 분배하는 JPEG Streamer, 그리고 사용자와 JPEG Streamer 사이에서 인터페이스를 제공하는 웹서버로 정의된다.



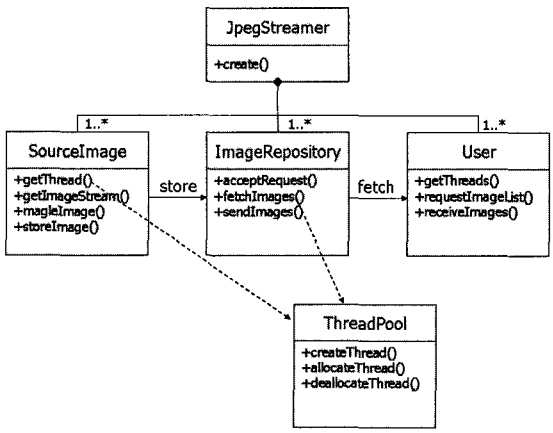
〈그림 2.1〉 실시간 영상감시 시스템의 Use Case Diagram

실시간 영상 감시 시스템의 핵심 엔진에 해당하는 JPEG Streamer는 실시간으로 영상을 수집해야하며 웹서버를 통하여 영상을 요청하는 사용자에게 이 영상을 제공해야 한다. JPEG Streamer는 이를 위해서 사용자와의 인터페이스를 마련해야 한다. 다수의 NCS로부터 영상들을 수집하고 다수의 동시사용자에게 영상들을 제공해야 하므로 많은 수의 쓰레드가 병렬처리 되어야 한다. 영상을 수집하는 쓰레드와 영상을 가져가는 쓰레드는 공유 메모리를 통하여 영상을 주고받는다. 안정적인 영상감시를 위해 JPEG Streamer는 이들 쓰레드들을 효율적으로 운영해야 하므로 쓰레드 풀을 운영한다.

2.2 JPEG Streamer의 기능

실시간 영상감시 시스템의 핵심 엔진에 해당하는 JPEG Streamer는 JPEG 영상을 수집하여 공유 메모리에 저장하고 요청이 있을 때 공유 메모리로부터 영상을 추출하여 네트워크로 전송

한다. 이때 안정적 서비스를 위해 쓰레드 풀을 관리한다. 그림 2.2의 JPEG Streamer 컴포넌트의 Class 다이어그램에서 JPEG Streamer를 구성하는 클래스들과 각 클래스들이 수행해야하는 메쏘드들을 보여준다.



〈그림 2.2〉 JPEG Streamer의 Class Diagram

(1) 영상수집

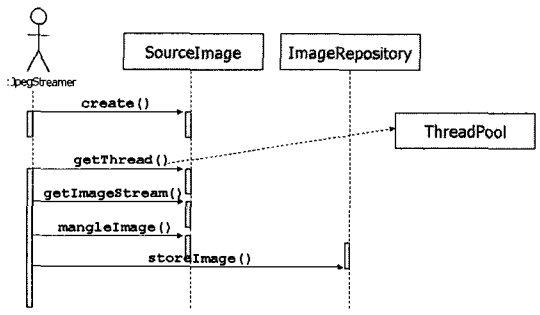
JPEG Streamer는 NCS들로부터 인터넷을 통하여 원시 영상을 JPEG 포맷으로 수집한다. 영상을 수집하기 위해 제일 먼저 해야 할 일은 네트워크 상 존재하는 NCS에 접속하는 것이다. 즉 Client Socket을 생성하고 NCS의 IP Address와 포트번호를 이용하여 접속한 후, 이 연결된 소켓을 이용하여 JPEG 영상을 Packet 형태로 수신한다는 것이다. 각각의 영상마다 하나의 소켓을 개설하고, 이들 영상들은 동시에 실시간으로 가져와야 하므로 각 영상의 처리를 위한 소켓들은 멀티쓰레딩 되어야 한다.

JPEG Streamer의 영상수집 과정은 구체적으로 다음과 같은 순서로 진행된다. 먼저 쓰레드 풀에서 메쏘드 getThread()를 이용하여 수집할 카메라 영상 수만큼의 가용한 쓰레드를 받아야 한다. 만약 가용한 쓰레드가 쓰레드 풀에 없다면 사용자의 요구는 거절된다.

각각의 쓰레드들은 모두 동일한 메쏘드에 의해서 동작하므로 한 개의 쓰레드가 어떻게 동작하는지만 정의한다. 멀티쓰레딩 API를 호출하면 쓰레드는 제일 먼저 소켓을 생성하고, 이 소켓에 자신이 요청한 영상을 제공할 NCS의 IP Address와 포트번호를 바인딩하여 NCS에 접속한다[2]. 접속이 성공하면 NCS에 영상을 요청하는 CGI를 보내고, NCS는 이 소켓에 HTTP 프로토콜을 이용하여 Server Push 방법으로 JPEG 영상의 스트림을 제공한다. SourceImage 클래스의 getImageStream() 메쏘드가 이 과정을 수행한다.

(2) 영상을 공유 메모리에 저장

Server Push 방식은 한 번의 클라이언트 요청만으로 서버가 지속적으로 동일한 형태의 복수 데이터를 제공하기 위해서 사용되는 방법이다. 실시간 영상감시에 사용되는 NCS들은 대부분 이 방식을 이용하는데 boundary string을 구분자로 해서 JPEG 이미지들을 계속해서 내보낸다. JPEG Streamer의 mangleImage()는 스트림을 boundary string을 이용하여 분리하고 분리된 스트림이 JPEG 영상인지 여부를 확인하기 위해 스트림의 처음 두 바이트와 마지막 두 바이트를 각각 비교하여 검증한다.



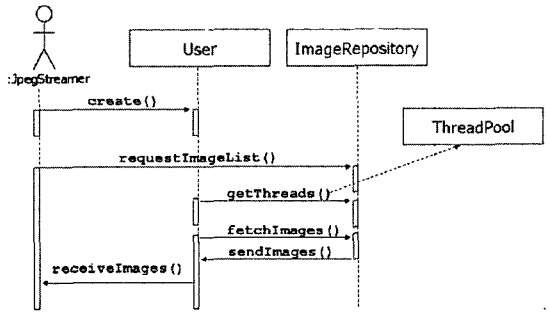
〈그림 2.3〉 JPEG Streamer의 영상수집 / 저장 Sequence Diagram

storeImage() 메쏘드는 분리된 JPEG 영상을 공유 메모리의 각 영상채널마다 지정된 주소에

저장하게 된다. 이 과정이 계속 반복되면서 공유 메모리의 지정된 저장소는 해당 카메라 영상의 가장 최근 스냅샷으로 계속 업데이트된다. 그림 2.3의 JPEG Streamer의 영상수집 / 저장 Sequence Diagram은 영상의 수집에서부터 분리하고 저장하는 과정을 설명하는 순차 다이어그램이다.

(3) JPEG 영상의 전송

사용자는 requestImageList() 메쏘드를 통해 자신이 원하는 영상채널의 목록을 JPEG Streamer에게 요청한다. 한편 JPEG Streamer는 사용자들의 요청을 처리하기 위해 서버소켓을 개설하고 사용자들의 요청을 기다린다[4]. 서버 소켓이 사용자의 요청을 받게 되면 별도의 사용자용 소켓을 만들어 사용자가 처음 영상채널의 목록을 요청할 때 할당 받은 쓰레드에 넘겨준다. 사용자가 요청한 영상채널들은 각기 쓰레드 하나씩을 차지하여 멀티쓰레딩으로 동작하게 된다. requestImageList() 메쏘드는 제일 먼저 JPEG Streamer에 소켓접속을 시도한다. JPEG Streamer는 이를 위해 서버소켓을 개설하고 기다리고 있다. JPEG Streamer는 연결을 수락하고 사용자에게 클라이언트 소켓을 하나 만들어 준다. 그리고는 getThread() 메쏘드를 이용해 쓰레드 풀에서 가용한 쓰레드 한 개를 얻어 와서 이 쓰레드에 방금 만든 클라이언트 소켓과 사용자가 requestImageList()를 통해서 보낸 정보, 즉 원하는 영상채널의 번호를 넘겨준다. 메쏘드 fetchImage()는 넘겨받은 영상채널번호를 이용해 JPEG Streamer의 공유 메모리로부터 해당 영상채널의 최근 JPEG 영상을 버퍼메모리에 복사하고, 이를 sendImage() 메쏘드를 이용해 소켓에 실어 내보낸다. 사용자 쪽 웹브라우저는 이를 받아서 화면에 그려주고, 다시 최근 영상을 요청하게 된다. 물론 이 때 TCP 소켓연결은 끊어지지 않은 상태이다. 즉 Client Pull 방식으로 JPEG 영상을 지속적으로 가져오는 것이다[13]. 이 과정을 순차 다이어그램으로 나타낸 것이 바로 그림 2.4의 JPEG Streamer의 영상요청 / 수신 Sequence Diagram이다.



〈그림 2.4〉 JPEG Streamer의 영상요청 / 수신 Sequence Diagram

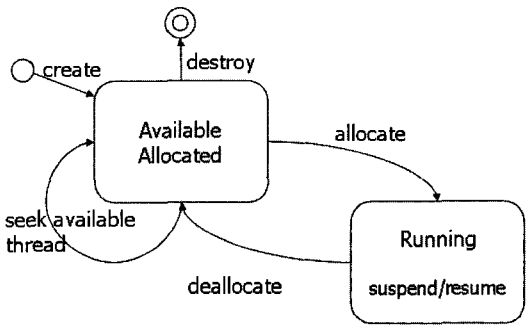
2.3 쓰레드 풀

실시간 영상감시에서는 모든 영상들을 실시간으로 처리하기 위해 소켓들의 입출력이 병렬로 처리되어야 하며, 특정 소켓의 입출력이 다른 소켓의 입출력에 영향을 미치지 않도록 안정적인 병렬처리를 위해 멀티쓰레딩 모델이 필요하다.

멀티쓰레딩을 이용하여 병렬처리를 구현하기 위해서는 시스템의 한정된 자원 때문에 멀티쓰레딩의 정도를 결정해야 한다[3,10]. JPEG Streamer는 안정적인 프로그램의 수행을 위해 최초 시스템으로부터 자신에게 필요한 수만큼의 쓰레드를 위한 일정량의 자원을 미리 할당받아서 각각의 쓰레드들에게 미리 이 자원을 배정하고 상황에 따라 이 쓰레드들을 할당하고 회수하는 방식으로 동작하도록 설계하였다. 그러므로 JPEG Streamer는 영상을 수집하거나 사용자의 요청에 새로운 소켓을 배정할 때마다 쓰레드 풀에서 가용한 쓰레드가 있는지를 먼저 확인하고, 가용한 쓰레드가 있는 경우에만 쓰레드를 배정받아서 이 쓰레드 안에서 소켓 통신을 하게된다.

작업이 끝난 소켓은 자신이 사용한 쓰레드를

쓰레드 풀에 반납한다. 이와 같이 쓰레드 풀을 운용하기 위해서는 쓰레드 풀에 있는 각각의 쓰레드들의 상태를 알 필요가 있다. 그림 2.5는 쓰레드의 상태가 이벤트에 따라 변화는 것을 설명하는 상태 다이어그램이다.



〈그림 2.5〉 쓰레드 풀의 Statechart Diagram

쓰레드 풀에 있는 쓰레드들은 할당 가능한 상태, 할당된 상태를 갖는다. 그리고 할당된 상태를 갖는 쓰레드는 다시 실행중인 상태와 할당된 후 정지된 상태를 갖게 된다. JPEG Streamer는 기동될 때 환경정보에서 최대 쓰레드 개수를 읽어 와서 그 개수만큼 쓰레드 풀을 조성한다. 초기 상태의 쓰레드들은 모두 할당 가능한 상태이다. 이후 getThread()에 의해서 할당된 쓰레드는 할당된 상태로 바뀐다. getThread()가 호출될 때마다 쓰레드 풀에서 가용한 쓰레드는 감소되고 최종적으로 모든 쓰레드가 할당되면 쓰레드 풀은 더 이상의 요청을 받아들이지 않게 된다. 잠시 후 쓰레드들이 작업을 마치면 할당 가능한 상태로 전환된다. 이와 같은 과정에 의해 JPEG Streamer는 시스템의 자원을 안정적으로 사용하고 시스템에 무리를 주지 않으면서 동시사용자 수를 보장할 수 있게 된다.

2.4 공유 메모리

공유 메모리의 사용과 관리에서 고려해야할 중요한 사항은 dirty read 문제와 문제 해결시 성능 저하이다. 따라서, JPEG Streamer에서 사용된 공유 메모리에서도 동일한 문제를 해결하여야 한다.

JPEG Streamer를 구성하는 객체중 하나인 공유 메모리는 SoureceImage 객체의 storeImage() 메쏘드에 의해서 그 내용이 갱신되고, User 객체의 fetchImage() 메쏘드에 의해서 그 내용이 읽혀진다. 공유 메모리를 이용하는 두 객체는 각각 읽기 작업만 하는 객체와 쓰기 작업만을 수행하는 객체이며 storeImage()와 공유 메모리 그리고 fetchImage()는 1:1:N의 관계를 갖는다. 이때 N은 최대 동시 사용자수를 의미하기도 한다.

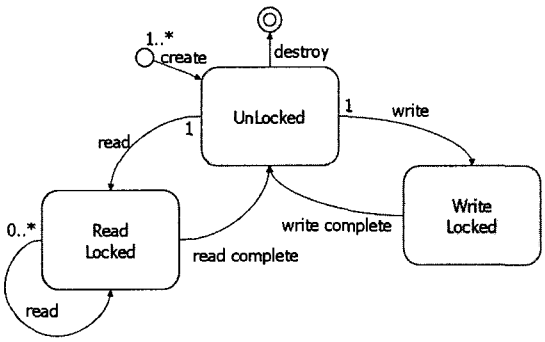
공유 메모리는 JPEG 영상들을 저장하는 저장소로 JPEG 영상 한 장의 최대크기를 단위 메모리의 크기로 하며 단위 메모리는 각각의 영상 채널에 매핑된다.

SourceImage 객체는 매핑 번호를 이용해서 매핑되는 영상채널의 가장 최근 스냅샷을 지정된 번호의 공유 메모리에 저장하게 된다. 마찬가지로 User 객체가 요청한 카메라 영상들은 각각의 영상채널의 고유번호가 된다. JPEG Streamer는 이 영상채널의 번호에 해당되는 공유 메모리를 찾아서 그 안에 저장되어 있는 JPEG 영상을 사용자 쓰레드의 버퍼에 복사해준다. 쓰레드는 이 버퍼의 내용을 자신에게 배정된 소켓을 통하여 User 객체에게 전송하게 된다.

공유 메모리에 JPEG 영상을 저장하는 쓰레드와 이 공유 메모리에 있는 JPEG 영상을 읽어가는 쓰레드는 병렬로 수행된다. 따라서 최악의 경우 SourceImage 객체가 특정 번호의 공유 메모리에 쓰기 작업을 시작하는 순간에 다수의 User 객체들이 같은 번호의 공유 메모리를 읽으려고 시도할 수도 있다. 이 경우 각각의 User 객체들은 완성되지 않은 JPEG 영상을 가져가게 되거나 메모리 보호 오류가 발생하게 된다.

공유 메모리객체 ImageRepository는 이러한 문제, 즉 쓰레드 간의 메모리에 대한 읽기, 쓰기

작업 중에 발생하는 dirty read문제의 해결을 위해 상호배제를 구현해야 한다[11].



〈그림 2.6〉 ImageRepository의 Statechart Diagram

각각의 공유 메모리는 하나의 쓰기작업을 하는 쓰레드에 의해서 Locked 상태가 되고, 이 동안에는 어떠한 읽기 작업도 허용되지 않는다. 쓰기작업이 완료되면 쓰레드는 공유 메모리의 상태를 UnLocked으로 변경한다. 만약 이 공유 메모리에 대한 Lock이 해제되기를 기다리고 있는 읽기작업 쓰레드가 있었다면 바로 작업을 진행한다. 쓰레드는 읽기작업을 시작하면서 역시 이 공유 메모리를 Locked상태로 변경한다. 그러나, 쓰기작업을 위한 Lock과는 달리 이 공유 메모리에 대해서 같은 읽기작업을 하는 또 다른 쓰레드들은 자유롭게 공유 메모리의 JPEG 영상을 가져갈 수 있다. 그림 2.6은 상호배제에 대한 상태 다이어그램이다.

2.5 운영환경 매개변수에 따른 용량계획

JPEG Streamer는 다수의 JPEG 영상을 실시간으로 네트워크를 통해 전송해야하기 때문에 충분한 네트워크 대역폭이 제공되어야 한다. 따라서, 실시간 영상감시 시스템을 위한 JPEG Streamer는 네트워크에 소통될 패킷량이 매우 중요한 요소 중 하나이다.

본 절에서는 JPEG Streamer 모델을 통하여 실시간 영상감시 시스템을 위한 자원의 용량계획의 지표가 되는 두 가지 자원 사용량 추정 모델을 제시한다.

(1) 네트워크 대역폭 추정 모델

자원들의 용량계획을 세우기에 앞서 실시간 영상감시 시스템의 운영환경을 설정할 필요가 있다. 이들 용량계획에 영향을 주는 실시간 영상감시 시스템의 구성요소와 매개변수는 <표 2.1>과 같다.

〈표 2.1〉 실시간 영상 감시 시스템의 운영에 필요한 매개변수들

항 목	매개변수	비 고
카메라 영상의 수	MaxCamera	감시 대상 영상의 수
JPEG Image의 크기	ImgSize	각 카메라 영상의 한 프레임이 갖는 평균 Image의 크기
매 초당 전송되는 JPEG 프레임 수	FPS	Frame Per Second
동시 사용자 수	MaxUser	
한 화면 최대 분할 수	MaxDiv	보통 16분할

실시간 영상감시 시스템을 위한 JPEG Streamer의 네트워크 대역폭(NBW : Network Band Width)은 영상채널들의 원시영상을 수집하기 위해서 필요한 대역폭과 사용자들의 요청에 따라 영상을 내보내기 위해 필요한 대역폭의 합으로 다음과 같다.

$$NBW = (MaxCamera \times ImgSize \times FPS) + (MaxDivision \times FPS \times ImgSize \times MaxUser)$$

위 수식은 순수 JPEG 영상의 대역폭만을 계산한 것이므로 TCP/IP 통신규약에 따라 추가 발생되는 헤더에 필요한 대역폭을 추가해야 한다. 이 추가되는 정보는 네트워크의 MTU에 따라서 달라진다[2]. TCP/IP는 메시지를 통째로

보내지 않고 MTU 단위로 쪼개서 보낸다. 통상 MTU는 1,500바이트이고, 이 중에 40바이트가 TCP와 IP 헤더로 사용된다. 그러므로 원본 JPEG 영상은 1,460바이트 단위로 쪼개지고 각각의 조각에 TCP/IP 헤더가 추가되어 하나의 MTU, 즉 패킷으로 만들어진다. MTU는 다시 14바이트의 MAC 헤더가 추가되고, 여기에 이더넷 컨트롤러는 프리앰블과 SOF, FSC 등 12바이트의 부가적인 정보를 MTU의 앞뒤에 덧붙여서 네트워크에 내보내게 된다. 이 패킷을 전달받은 상대편 TCP/IP는 이 패킷에 대한 응답패킷으로 66byte를 되돌려 준다[2]. 그러므로 JPEG 영상의 매 1,460바이트 마다 132바이트의 부가적인 트래픽이 발생한다고 볼 수 있다. 이를 계산하면 다음 수식에서처럼 원래 순수메시지의 약 9%에 해당하는 부가정보가 발생하게 된다.

$$\begin{aligned} \text{부가정보} &= \frac{\text{실제메시지의 전체 바이트수}}{1{,}460} \times 132 \\ &= \frac{132}{1{,}460} \times \text{실제메시지의 전체 바이트수} \\ &= 0.09 \times \text{실제메시지의 전체 바이트수} \end{aligned}$$

원래 메시지 크기의 9%가 부가정보로 추가되므로 부가정보계수를 1.09라고 볼 수 있다. 이 부가정보계수를 NBW에 적용하면 다음의 수식과 같다.

$$\begin{aligned} NBW = \; & 1.09 \times (MaxCamera \times \\ & ImgSize \times FPS + MaxDivision \\ & \times FPS \times ImgSize \times MaxUser) \end{aligned}$$

수식을 보다 간단히 하기 위해 표 2.2와 같이 상수로 정의한다. NCS 장비의 특성상 각 장비가 4개 영상채널을 모두 서비스 할 때의 각 채널당 최대 FPS는 5이다. 그리고 사용자가 모니터를 통해서 한 화면에 감시하기에 좋은 최대 분할 수는 16이 적당하다. 그리고 352×240 크기의 리솔루션에서 중간정도의 압축률을 갖는 JPEG 영상 한 장의 평균 크기는 13KB 정도이다.

〈표 2.2〉 실시간 영상 감시의 상수값

항목	상수 이름	상 수 값
JPEG Image의 크기	ImgSize	13KB
매 초당 JPEG 프레임 수	FPS	5
한 화면 최대 분할 수	MaxDiv	16

그러므로 이들 상수들을 NBW에 적용하면 다음과 같다.

$$\begin{aligned} NBW = \; & 1.09 \times (MaxCamera \times 13 \times \\ & 5 + 16 \times 5 \times 13 \times MaxUser)KB \\ = \; & 1.09 \times (65 \times MaxCamera + \\ & 1040 \times MaxUser)KB \\ = \; & (70.85 \times MaxCamera + \\ & 1133.6 \times MaxUser)KB \\ = \; & (566.8 \times MaxCamera + \\ & 9068.8 \times MaxUser)Kbps \end{aligned}$$

만약, 16대의 카메라 영상을 10명의 유저가 동시에 감시하기 위한 실시간 영상감시 시스템에 소요되는 네트워크 대역폭은 다음과 같다.

$$\begin{aligned} NBW &= (566.8 \times 16 + 9068.8 \times 10)Kbps \\ &= 97.41Mbps \end{aligned}$$

즉, 최대 97.41Mbps의 대역폭이 필요하다. 물론 상수의 값을 실시간 영상 감시의 적용분야에 따라서 적절하게 조정하게 되면 위의 결과는 조금 다르게 나올 수 있다. 이 추정 모델에선 패킷충돌에 따른 재전송 등의 부가적인 트래픽은 고려하지 않았다. JPEG Streamer의 설계과정에서 도출된 네트워크 대역폭 추정 모델은 다양한 운영환경의 실시간 영상감시를 위한 시스템의 설계과정에서 필요한 네트워크 대역폭을 계산하거나 반대로 현재 구성된 네트워크 환경에서 운영될 수 있는 최대 영상채널의 수와 동시사용자 수를 계산할 수 있다.

(2) 메모리 사용량 추정 모델

메모리 사용량 추정을 위해 사용할 JPEG Streamer의 매개변수로는 쓰레드 풀을 구성하는 쓰레드의 개수와 공유 메모리를 구성하는 메모리의 개수, 그리고 JPEG 영상의 평균 크기 등이 있다. 표 2.3에 기술한 매개변수들은 앞서 기술한 표 2.1의 내용과 더불어서 메모리사용량 추정을 위해서 확장된 매개변수의 집합이다.

〈표 2.3〉 공유 메모리 사용량 추정을 위한 매개변수들

항 목	매개변수 이름	비고
쓰레드풀의 쓰레드 개수	MaxThreadNum	
공유 메모리 개수	MaxMemNum	
JPEG Image Size	ImgSize	13KB, Max 26KB

쓰레드 풀에서 사용하는 쓰레드의 총 개수는 $MaxTreadNum = MaxCamera + MaxUser \times MaxDiv$ 이다. JPEG Streamer는 NCS로부터 카메라 영상을 수집하기 위해서 최대 카메라 영상채널 수만큼의 Client Socket을 생성하게 된다. 여기에 사용자들을 위한 소켓들을 더해주면 JPEG Streamer가 운영해야 할 전체 소켓의 개수를 구할 수 있고, 이 소켓 수가 쓰레드 풀을 구성하는 쓰레드의 개수가 된다. 이 때 사용자들을 위한 소켓의 최대값은 [최대사용자수 × 한 화면 최대분할 수]로 구할 수 있다.

메모리 사용량은 이들 쓰레드들이 소켓에 입출력을 하기 위해서 사용하는 버퍼 메모리와 공유 메모리 크기의 합을 구하면 된다. 우선 공유 메모리는 ImageRepository의 크기가 되는데, 이는 평균 13KB, 최대 26KB의 JPEG 영상을 각각 저장할 수 있는 단위 메모리를 최대 카메라 영상 수만큼 준비해 두면 되므로 $SM = MaxCamera \times 26KB$ 이다. 쓰레드 풀이 사용할 버퍼 메모리는 처음 설계에서 나타나지 않은 부분으로 JPEG Streamer를 실제로 개발하는 과정에서 발견된 영상의 지연 현상이나 끊김 현상을 해결하기 위해 더블 버퍼링을 적용함으로써 필요하게된 것이다. 2.4절의 공유 메모리에서 그림 2.6을 보면 공유 메모리에 쓰기, 읽기 작업을 하는 동안 상호배제를 위해 Lock을 거는 부분이 있다. 더블버퍼링을 미적용시 Lock이 걸리는 시간은 소켓 입출력에 소요되는 시간 만큼이었다. 예를 들어 공유 메모리에 쓰기 작업을 하는 경우 일단 해당 영상채널 번호의 공유 메모리에 Lock을 걸고 소켓으로부터 최대 26KB만큼의 스트림을 읽어 메모리에 기록한 다음 Lock을 해제한다. 이 과정에서 소켓 입출력이 블록될 수 있고, 이에 따라 장시간 메모리가 Lock됨으로 해서 사용자 쓰레드들이 이 공유 메모리를 읽어갈 수 없게되어 영상 지연이 생기게 된다.

이 문제의 해결방법은 공유 메모리에 Lock이 걸리는 시간을 최소화 하는 것이다. 소켓의 입출력 시간은 네트워크의 트래픽에 종속적이다. 이는 CPU의 클럭과 버스의 클럭에 종속적인 메모리 액세스 시간에 비하면 꽤 긴 시간이다. 그러므로 공유 메모리에 Lock을 거는 시간을 소켓입출력 시간에서 메모리 액세스 시간으로 줄여주는 것이 영상의 지연을 해소하는 방법이 될 수 있다. 이를 위해서 더블 버퍼링의 개념이 필요하다. NCS로부터 소켓을 통하여 들어오는 JPEG 영상을 버퍼 메모리에 저장한 다음, 공유 메모리에 Lock을 걸고 버퍼 메모리의 내용을 그대로 공유 메모리에 복사한 후 바로 Lock을 해제한다. 이와 같은 방법으로 공유 메모리로부터 JPEG 영상을 읽어가는 사용자들의 쓰레드들도 더블버퍼링을 구현함으로 공유 메모리가 Lock에 걸려 있는 시간을 줄일 수 있다.

이와 같이 영상의 지연을 막기 위해서 버퍼 메모리의 운영은 불가피하다. 그러므로 전체 메모리 사용량은 공유 메모리와 쓰레드 풀이 사용하는 버퍼 메모리의 합이 된다. 공유 메모리의 개수는 최대 카메라 영상채널수 만큼이고 버퍼 메모리의 개수는 최대쓰레드 개수 만큼이므로 $MaxMem = (MaxTreadNum + MaxCamera)$

× *26KB*이다. 물론 기타 프로그램의 운영에 필요한 변수들을 위한 메모리 공간은 생각하지 않기로 한다. 다음은 표 2.1과 표 2.2 그리고 표 2.3의 매개변수들과 앞서 네트워크 대역폭 추정 모델을 구할 때 사용한 상수값을 이용하여 구한 메모리 사용량 추정 모델의 수식은 다음과 같다.

$$\begin{aligned} MaxMem &= (MaxCamera + MaxDiv \times \\ &\quad MaxUser + MaxCamera) \times 26KB \\ &= (2MaxCamera + 16MaxUser) \times 26KB \end{aligned}$$

이제 여기에 실제 네트워크 대역폭을 구할 때 사용했던 것처럼 최대 영상채널 수를 16으로 보고 최대 동시사용자 수를 10명으로 하여 위의 수식에 적용하면 다음과 같이 메모리 사용량을 추정할 수 있다.

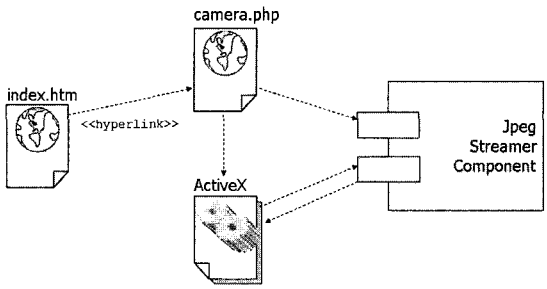
$$\begin{aligned} MaxMem &= (2 \times 16 + 16 \times 10) \times 26KB \\ &= 192 \times 26KB \\ &= 4.875MB \end{aligned}$$

3. 구 현

2장의 설계를 바탕으로 구현된 JPEG Streamer는 독립적으로 동작하는 완성된 애플리케이션이 아니라 실시간 영상감시 시스템을 구성하는데 필요한 모듈이고 일종의 컴포넌트이다. 그러므로 이 컴포넌트가 제대로 동작하는지를 알아보려면 JPEG Streamer를 이용하여 실제 실시간 영상감시 시스템을 구축하여야 한다. 이 컴포넌트의 구현을 검증하기 위해서 웹기반 실시간 영상감시 시스템의 경우를 모델로 하였다.

3.1 웹기반 실시간 영상감시의 구현

본 논문에서 구현한 JPEG Streamer를 이용해서 웹기반의 실시간 영상감시 시스템을 구성해 보았다. 그림 3.7은 웹기반 실시간 영상감시 시스템을 구성하는 JPEG Streamer와 기타 구성요소들의 관계를 보여주는 컴포넌트 다이어그램이다.



〈그림 3.7〉 웹기반 실시간 영상감시에 적용된 JPEG Streamer

(1) 웹 사용자 인터페이스

HTML과 서버사이드 스크립트 등을 통해서 사용자 인터페이스를 구성한다. 이때 필요한 사용자 인터페이스에는 화면분할을 선택할 수 있는 분할선택 버튼과, 다음 또는 이전 채널로 이동할 수 있는 Next, Prev 버튼, 기타 카메라를 제어하는 데 필요한 Pan, Tilt, Zoom버튼 등이 있다. 경우에 따라선 허가 받은 사용자들만 영상을 볼 수 있도록 로그인 인터페이스도 필요할 것이다.

서버사이드 스크립트는 웹서버와 함께 탑재된 데이터베이스로부터 영상감시를 위한 NCS 및 영상채널에 대한 구성정보를 읽어 와서 복잡한 사용자 인터페이스를 효율적으로 구성해 준다. 클라이언트사이드 스크립트는 정적인 HTML의 인터페이스에 동적인 이벤트 처리를 더해준다.

(2) ActiveX와 자바스크립트

JPEG Streamer는 웹서버와는 별개의 애플리케이션으로 동작한다. HTTP 프로토콜 대신 별도의 규약을 가지고 독립적으로 동작한다. 그러므로 브라우저는 별도의 통신 규약을 가지고 소켓통신을 하며 JPEG 영상을 실시간으로 화면에 표출해 줄 수 있는 ActiveX의 도움을 받아야

한다. 그리고 이 ActiveX는 서버사이드 스크립트나 자바스크립트의 함수호출로 사용자 인터페이스의 이벤트에 반응한다. 반대로 ActiveX에서 발생하는 이벤트에 대해서는 자바스크립트의 함수를 호출하여 사용자 인터페이스와 반응한다. 이렇게 함으로써 ActiveX와 사용자는 서로 메시지를 주고받을 수 있으며, 궁극적으로 JPEG Streamer와 교신할 수 있게 된다.

3.2 JPEG Streamer의 테스팅

본 구현과정에서 JPEG Streamer의 기능, 안정성, 성능 및 효용성을 평가하기 위해서 실시간 영삼감시 시스템이 본질적으로 요구하는 항목들을 표 3.1의 질문으로 요약했다. 다음은 이 평가 항목들에 대한 검증 방법들이다.

〈표 3.1〉 JPEG Streamer의 테스팅을 위한 평가 항목

테스트 범주	평 가 항 목
실시간 영상	영상은 지연 또는 끊김 현상 없이 잘 보이는가?
네트워크 안정성	네트워크가 불안정한 경우에도 프로그램이 큰 문제없이 유연하게 대처하는가?
다중 사용자	다수의 사용자가 접속했을 때에도 각각의 사용자들이 영상을 실시간으로 보는 데 문제가 없는가?
사용의 용이성	JPEG Streamer 컴포넌트를 웹페이지에서 사용하는데 어려움은 없는가?
안정성	JPEG Streamer 컴포넌트를 탑재한 웹서버는 안정적으로 동작하는가?

더블버퍼링을 적용하여 영상의 지연이나 깜박임 문제는 해결하였다. 영상의 지연유무를 확인하기 위해서 아날로그 시계를 이용했다. 카메라가 시계를 비추도록 하고, JPEG Streamer를 거쳐 온 영상에서의 지연유무는 현재시각과 초침의 움직임 등을 관찰함으로써 검증이 가능했다.

불안정한 네트워크 상황에 대해서 JPEG Streamer가 유연하게 대응하는지를 검증하기 위하여 네트워크에 극단적인 상황, 즉 이더넷 케이블을 단절시키거나, NCS를 재부팅하는 등의 상황을 만들어 보았다. 소켓이 어떠한 이유로든 갑자기 사라지게 되면 JPEG Streamer는 비정상적으로 종료하게 된다. 그러므로 이런 갑작스런 소켓의 단절에 JPEG Streamer가 유연하게 대응할 수 있도록 소켓의 비정상적인 단절에 대비해 소켓상태를 지속적으로 감시하고 소켓이 끊어졌을 때, 바로 일정간격으로 재접속을 시도하게 함으로써 네트워크 단절 이후에도 네트워크가 재개되었을 때에는 영상의 감시를 계속 수행 할 수 있도록 알고리즘을 수정하였다.

지원할 동시사용자 수만큼의 쓰레드 풀을 미리 조성해 놓았기 때문에 지정된 범위 내의 동시사용자들에 대해서도 원활한 실시간 영상감시 서비스를 제공할 수 있었다. 물론 준비 된 쓰레드 풀을 초과하는 요청은 거절된다. 이렇게 함으로 해서 시스템의 자원을 안정적으로 사용할 수 있게 된다.

웹 개발자는 JPEG Streamer의 내부를 몰라도 된다. 객체지향 프로그래밍의 캡슐화 개념과 같이 JPEG Streamer의 내부동작은 웹 개발자에게는 감춰진다. 웹 개발자는 단지 JPEG Streamer와의 인터페이스 규정만 준수하면 된다.

또한 웹서버가 운영되는 시스템의 CPU, 메모리 사용량을 주기적으로 체크해 보았을 때, JPEG Streamer가 시스템에 큰 무리를 주지 않고 안정적으로 동작하는 것을 확인하였다.

4. 결 론

최근 인터넷 인프라의 성장은 실시간 영상 감시의 개념을 많이 바꾸어 놓았다. 기존 CCTV중심의 실시간 영상감시 솔루션이 NCS 중심으로 점차 바뀌어가고 있다. 그뿐 아니라 실시간 영상감시 애플리케이션에 대한 요구사항도 급증하고 있다. 이러한 요구에 발 빠르게 대처하기 위해선 기존 애플리케이션의 개발에 적용한 컴포넌트기

반 개발방법을 실시간 영상감시 시스템에도 적용해야 한다. 본 논문에서는 이러한 실시간 영상감시 시스템 개발에 꼭 필요한 JPEG Streamer의 모델을 제시하였고, 이를 설계, 구현하였다.

JPEG Streamer의 설계 개념과 목적은 다양한 실시간 영상감시 시스템의 구축에 꼭 필요한 핵심 모듈을 재사용 가능한 컴포넌트로 만들어서, 시스템 개발의 생산성을 높이고 시스템의 안정성 및 신뢰성을 확보하는 것이다.

이 JPEG Streamer의 설계 핵심은 쓰레드 풀과 공유 메모리의 사용이다. 시스템 자원사용의 안정성과 다중 사용자들에게 안정적인 서비스를 위해서 쓰레드 풀을 사용했고, 다수채널의 영상을 여러 대의 NCS로부터 수집하고 이를 다중사용자에게 제공하기 위해서 공유 메모리를 사용했다. 쓰레드 풀을 구현하기 위해서 POSIX thread API를 사용했으며, 이때 병렬로 수행되는 각각의 쓰레드들이 공유하는 공유 메모리의 안정적인 사용을 위해서는 상호배제 기법을 사용하였다. 또한 실시간 영상감시의 기본적인 요구사항인 지연 및 끊김 현상이 없는 영상을 위해서는 더블버퍼링 기법을 이용하였다.

본 논문의 설계 과정에서 제시한 네트워크 대역폭 추정모델과 메모리 사용량 추정모델은 실제로 실시간 영상감시 시스템을 구현하기 전에 네트워크를 설계하고, 필요한 서버의 용량계획을 수립하는 데 지표로 삼을 수 있어서 전체 시스템의 운영환경을 명세하고 이를 설계하며, 일정을 수립하여 이에 따른 비용을 산정하는데 도움을 줄 수 있다.

웹기반 실시간 영상감시는 사용자 중심의 실용적인 애플리케이션이다. 언제, 어디서나 부가적인 프로그램의 설치 없이 사용 가능하다. 다양한 형태의 실시간 영상감시의 활용은 지속적으로 발전하고 있는 네트워크 인프라와 디지털 영상압축 기술의 발달, 그리고 컴퓨터의 컴퓨팅 파워의 증강으로 가속화 될 것이다. 이에 따라 실시간 영상감시 시스템의 개발요구는 그 형태가 다양해 질것이고, JPEG Streamer는 실시간 영상감시 시스템을 위한 표준 컴포넌트로 자리 매김하게 될 것이다.

참 고 문 헌

[1] 김경주, 조남규,"UML Components: 컴포넌트 기반 소프트웨어 명세를 위한 실용적인 프로세스", 인터비젼, 2001.

[2] 김성훈 역,"성공과 실패를 결정하는 1%의 네트워크 원리", 성안당, pp. 82-127, 2004.

[3] 김세화, 박정근, 홍성수, "실시간 응용 프로그래밍을 위한 Pthread API 확장",한국정보과학회 컴퓨터시스템 연구회 추계 학술발표회 논문집, pp. 105-112, 1999.

[4] 김화종, "컴퓨터 네트워크크 프로그래밍", 홍릉과학출판사, pp. 103-144, 2002.

[5] 유해영 역, Stephen R. Schach 저,"UML과 C++ 중심의 구조적 · 객체-지향소프트웨어 공학", 이한출판사, pp. 454-516, 2001.

[6] 이진영, 윤영민, "무선망을 이용한 가정 및 사무실 보안 시스템에 관한 연구", 강남대학교 산학기술연구소논문집, pp.117-134, 제17호, 2004

[7] 정기원, 최윤석, 김영희, "컴포넌트 기반 개발을 위한 통합 CASE 프레임워크", 숭실대학교 대학원 논문집 제 20권, pp. 329-343, 2002.

[8] 조경산 역,"UNIX for Programmers and Users", 이한출판사, pp. 544-561, 2003.

[9] 한국인터넷정보센터,"2004년 4월 기준 인터넷통계 월보", 한국인터넷정보센터(KRNIC) 인터넷통계월보, 2004.

[10] Harvey M. Deitel, Paul J. Deitel, David R. Choffnes,"Operating Systems, 3rd Edition", Prentice-Hall, 2004.

[11] James H. Anderson, Yong-Jik Kim,

"Shared-memory Mutual Exclusion: Major Research Trends Since 1986", 2001.

[12] Kale, T. Socolofsky, "A TCP/IP Tutorial", Network Working Group Request for Comments: 1180, RFC-1180, 1991.

[13] Manfred Hauswirth, Mehdi Jazayeri, "A Component and Communication Model for Push Systems1", 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-7), 1999.

[14] Peter Seebach, "An Instruction to POSIX thread", IBM, 2004.

◐ 저 자 소 개 ◑

김 경 환(Kyung Hwan Kim)
1993년 조선대학교 전자계산학과 졸업
2004년 단국대학교 대학원 컴퓨터과학및통계학과 졸업(석사)
2000년~2005년 신흥대학 컴퓨터정보계열 겸임교수
2005년~현재 (주)동원S&S 연구소 소장
관심분야 : u-Class, VoIP, RFID 응용
E-mail : khkim@dongwonsns.com

유 해 영(Hae-Young Yoo)
1979년 단국대학교 수학과 졸업(학사)
1982년 단국대학교 대학원 졸업(석사)
1994년 아주대학교 대학원 컴퓨터공학과 졸업(박사)
1983년~현재 단국대학교 정보컴퓨터학부 교수
관심분야 : 소프트웨어 공학, 시스템 프로그램 등
E-mail : yoohy@dankook.ac.kr

이 진 영(Jin-Young Lee)
1995년 단국대학교 대학원 전산통계학과 졸업(석사)
1998년 단국대학교 대학원 전산통계학과 수료(박사)
1999년~현재 강남대학교 교양학부 조교수
관심분야 : 이미지프로세스, 생체인식, 침입탐지 시스템
E-mail : goodman3@kangnam.ac.kr