|  |
| --- |
| **ARP Packet Capture 모니터링**  **개발 설계서** |
| **박현준**  **<2020/06/18>** |

**목 차**

[1. ARP Packet Capture 모니터링 서비스 개요 3](#_Toc43381723)

[1.1 시스템 설계 3](#_Toc43381724)

[1.2 시스템 구현 환경 3](#_Toc43381725)

[1.3 설계목표 및 요구사항 4](#_Toc43381726)

[1.4 시스템 구현 도구 4](#_Toc43381727)

[1.4.1. Vim 4](#_Toc43381728)

[1.4.2. GCC / G++ 4](#_Toc43381729)

[1.5 ARP 패킷 구조 4](#_Toc43381730)

[1.6 라이브러리 4](#_Toc43381731)

[1.6.1 libpcap 4](#_Toc43381732)

[1.6.2 mysql 4](#_Toc43381733)

[2. 시스템 설계 및 모델 5](#_Toc43381734)

[2.1 Sequence Diagram 5](#_Toc43381735)

[2.2 Data-flow Model 5](#_Toc43381736)

[2.3 Block Diagram 6](#_Toc43381737)

[3. 데이터베이스 설계 7](#_Toc43381738)

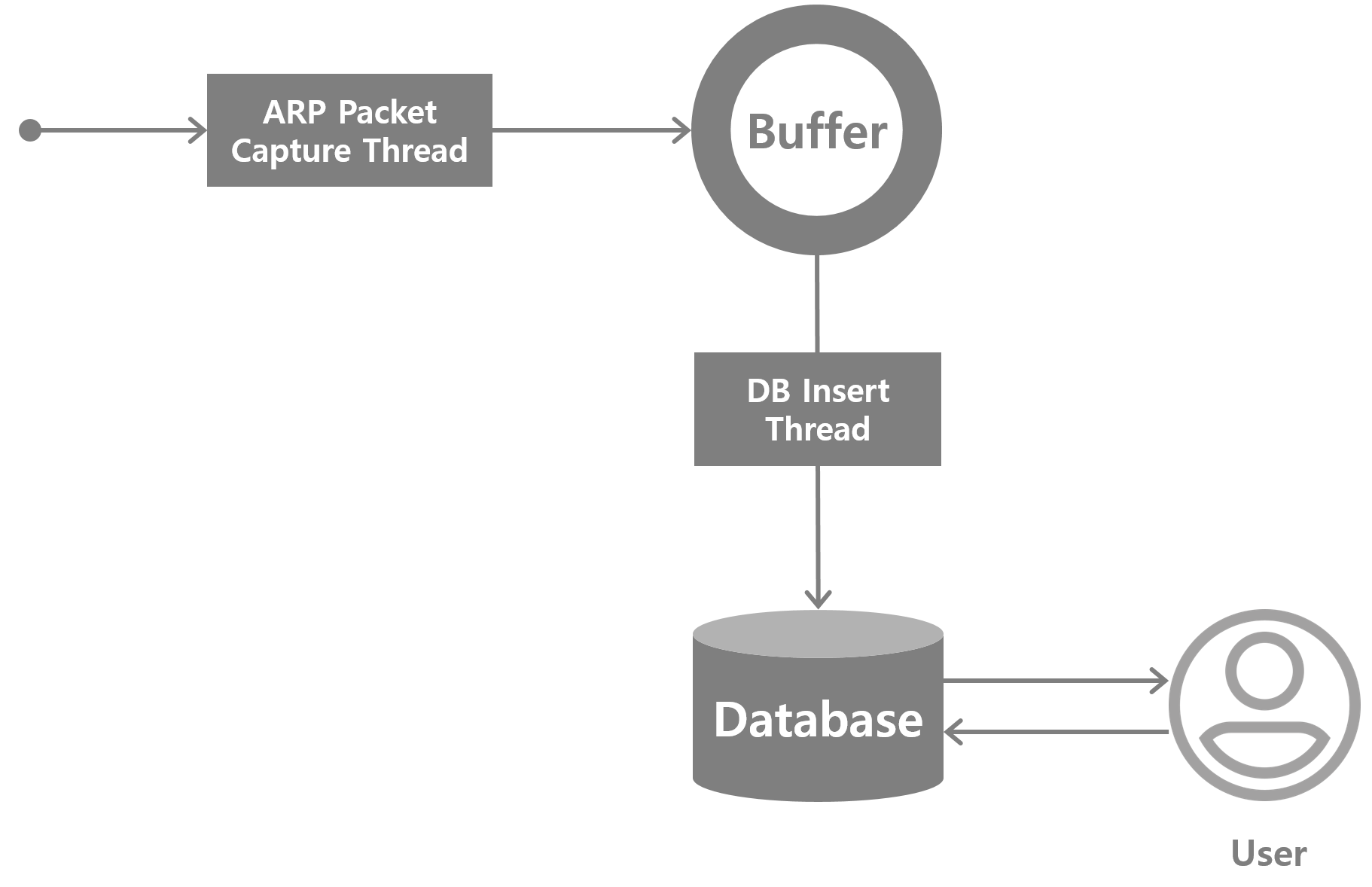
[3.1 데이터베이스 테이블 설계 7](#_Toc43381739)

[4. 시스템 종합 분석 방법 7](#_Toc43381740)

# **1. ARP Packet Capture 모니터링 서비스 개요**

- ARP는 네트워크 상에서 IP주소를 물리적 주소인 MAC으로 Bind를 시켜주는 프로토콜입니다. 이 프로그램은 네트워크 내에서의 패킷 정보를 수집합니다. 여러 패킷 중에서 ARP 패킷만 수집하여 필요한 정보만 데이터베이스에 저장 후 사용자가 데이터베이스를 통해 저장된 ARP 패킷 데이터를 볼 수 있는 기능을 제공합니다.

## 1.1 시스템 설계



- 사용자는 데이터베이스로부터 수집한 ARP 패킷에 대한 정보를 확인할 수 있습니다.  
- thread를 이용하여 캡쳐한 ARP는 데이터베이스에 바로 넣기보다 중간에 버퍼를 둡니다.  
- 데이터베이스는 수집한 ARP 패킷에 대한 정보를 저장합니다.  
- 수집한 ARP 정보를 데이터베이스를 통해 볼 수 있습니다.

## 1.2 시스템 구현 환경

시스템 구현은 Virtual Box에서 CentOS 7 운영체제 가상환경을 구성하였습니다. 구현 프로그래밍 언어로써는 C/C++를 사용하였으며 ARP 정보를 저장하기 위한 데이터베이스로 MySQL을 사용하였습니다.

## 1.3 설계목표 및 요구사항

이 문서는 프로그램을 설계하는데 있어서 다음의 목표 및 요구사항을 따릅니다.

- ARP Packet만을 수집하는 기능  
 - Capture된 ARP 패킷들은 세부적인 정보로 나누어 DB내에 저장  
 - 사용자가 프로그램을 실행한 후 DB내의 저장된 정보 조회 기능

## 1.4 시스템 구현 도구

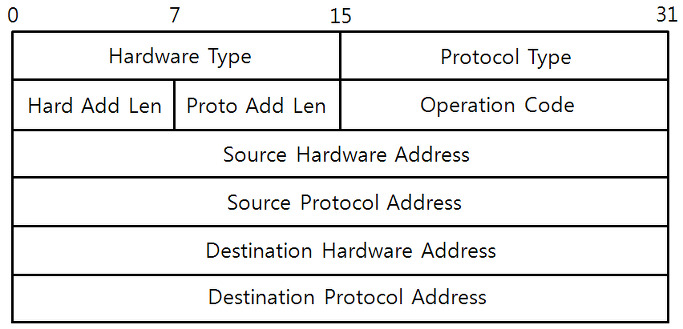
### 1.4.1. Vim

- vim은 유닉스 환경에서 사용되는 vi와 호환되면서도 독자적인 기능을 추가한 텍스트 편집기입니다. 문법 강조 기능, 되돌리기 기능, 유니코드 등등 여러 기능을 제공하며 추가적으로 많고 손쉽게 설치가 가능한 플러그인을 제공합니다. 강력한 기능이 제공되는 만큼 익숙해지는 데 상당히 시간이 걸린다는 단점이 있습니다.

### 1.4.2. GCC / G++

- gcc와 g++는 모두 리눅스 환경에서의 제공되는 컴파일러입니다. C와 C++를 컴파일을 하지만 차이점은 gcc의 경우 .c와 .cpp 파일을 모두 각각 C언어와 C++언어로 컴파일을 하지만 g++의 경우 .c와 .cpp 파일 모두 C++언어로 컴파일을 하게 됩니다. 추가적으로 gcc의 경우 C라이브러리와 링크되며, g++의 경우 C++ 라이브러리에 링크된다는 점이 다릅니다.

## 1.5 ARP 패킷 구조



ARP 패킷은 다음과 같은 구조로 이루어져 있으며 각 역할은 다음과 같습니다.

- Hardware Type : MAC의 유형을 나타내며 Ethernet 통신은 1로 설정됩니다.  
 - Protocol Type : 매핑 대상 주소의 유형을 나타내며 IPv4의 경우 0x0800으로 설정됩니다.  
 - Hardware Address Length : MAC의 주소를 byte로 표시합니다. Ethernet에서는 6bytes 입니다.  
 - Protocol Address Length : Protocol 주소 길이를 byte로 표시합니다. IPv4의 경우 4bytes 입니다.  
 - Operation Code : ARP의 동작을 나타냅니다. OP Code에서 ARP의 유형을 파악할 수 있습니다.

|  |  |
| --- | --- |
| **OP Code** | **동작** |
| 1 | ARP Request |
| 2 | ARP Reply |
| 3 | RARP Request |
| 4 | RARP Reply |

- Source Hardware Address : 송신자 MAC 주소  
 - Source Protocol Address : 송신자 IP 주소  
 - Destination Hardware Address : 수신자 MAC 주소  
 - Destination Protocol Address : 수신자 IP 주소

추가적으로 ARP 유형에서 Gratuitous ARP 패킷도 존재합니다. 이는 부팅 시에 보내지는 broadcast 요청 패킷이며 IP충돌을 감지하거나 이전의 ARP 테이블을 갱신하기 위한 목적으로 사용됩니다. Operation Code에서는 1번으로 ARP Request 패킷과 유사하지만 출발지IP와 목적지 IP주소가 같다는 점이 GARP의 특징입니다.

## 1.6 라이브러리 함수

ARP Packet Capture Monitoring 시스템은 네트워크 패킷 캡쳐와 데이터베이스 저장이 주요 기능입니다. 따라서 Linux 환경에서 이를 구현하기 위한 두 가지 라이브러리를 사용합니다. 네트워크 패킷 캡쳐 라이브러리인 libpcap과 mysql 데이터베이스에 접근하고 데이터를 입력하기 위한 mysql 라이브러리가 있습니다.

### 1.6.1 libpcap

- Linux 계열 운영체제에서 간단하게 사용 가능한 네트워크 패킷을 캡쳐하기 위한   
함수들을 모아놓은 User-level 라이브러리입니다.

- 주요 함수는 다음과 같습니다.

|  |
| --- |
| **char pcap\_lookupdev(char \*errbuf);** |
| 패킷을 캡쳐할 NIC를 찾아 이를 지칭하는 String을 반환합니다. 리눅스에서는 eth0, eth1 혹은 enp0s3, enp0s8과 같은 형식으로 표현됩니다. |
| - char \*errbuf : 에러 발생 시 에러 메시지를 저장하는 버퍼  - return값 : 네트워크 디바이스의 alias |

|  |
| --- |
| **pcap\_t \*pcap\_open\_live(char \*device, int snaplen, int mode, int to\_ms, char \*ebuf);** |
| 네트워크 패킷을 보기 위해 필요한 Packet Capture Descriptor를 얻기 위해 사용됩니다. |
| - char \*device : 사용할 네트워크 장치를 지정합니다.  - int snaplen : 캡처할 최대 byte입니다. 크기를 지정하면 필요한 데이터만 받아옵니다.  - int mode : 1로 설정할 경우 promiscuous로 설정됩니다.  - int to\_ms : 패킷이 버퍼로 바로 전달되는 것이 아니라 설정한 시간이 지나면 전달합니다.  - char \*ebuf : pcap\_open\_live()가 실패한 경우에 대한 에러메시지가 저장됩니다. ( NULL ) |

|  |
| --- |
| **int pcap\_setfilter(pcap\_t \*p, struct bpf\_program \*fp);** |
| 네트워크 장치 descriptor에 filter를 지정합니다. |
| - pcap\_t \*p : 패킷 캡쳐 descriptor  - struct bpf\_program \*fp : pcap\_compile()에서의 결과를 표시합니다.  > 0일 경우 성공, -1은 실패를 의미합니다. |

|  |
| --- |
| **int pcap\_loop(pcap\_t \*p, int cnt, pcap\_handler callback, u\_char \*user);** |
| 실제 패킷을 잡고 실행할 함수를 지정합니다. 지정한 cnt만큼 패킷을 잡고 이를 처리하기 위한 callback 함수를 수행하게 됩니다. |
| - pcap\_t \*p : 패킷 캡쳐 descriptor  - int cnt : 읽을 패킷의 개수  - pcap\_handler callback : 패킷을 처리할 함수  - u\_char \*user : 사용자 데이터   - return값 : 성공 시 읽은 패킷의 개수, 저장 파일의 EOF는 0, 에러발생은 -1입니다. |

|  |
| --- |
| **void pcap\_close(pcap\_t \*p)** |
| pcap\_t와 관계되는 파일을 닫고 사용한 자원들을 닫습니다. |
| - pcap\_t \*p : 패킷 캡쳐 descriptor |

다음은 네트워크 인터페이스의 주소와 이름을 출력하는 예제입니다.

#include <stdio.h>

#include <pcap.h>

int main()

{

// pcap\_if\_t : 네트워크 인터페이스 정보

pcap\_if\_t \*alldevs;

pcap\_if\_t \*d;

int i = 0;

char errbuf[PCAP\_ERRBUF\_SIZE];

// 모든 네트워크 인터페이스 정보를 확인

if(pcap\_findalldevs(&alldevs, errbuf) < 0) {

printf("[+] pcap\_findalldevs error\n");

return 1;

}

// 네트워크 인터페이스가 없다면

if(!alldevs) {

printf("%s\n", errbuf);

}

// 네트워크 인터페이스가 있다면 처음부터 계속해서 인터페이스 이름과 정보 출력

for(d=alldevs; d; d=d->next) {

printf("%p : %d. %s", d, ++i, d->name);

if(d->description)

printf(" (%s)", d->description);

printf("\n");

}

// findalldevs는 동적할당되기 때문에 freealldevs로 동적 메모리 할당 해제가 필요

pcap\_freealldevs(alldevs);

return 0;

}

### 1.6.2 mysql

- mysql.h은 C 프로그램에서 MySQL 데이터베이스에 연동하여 관리하기 위한 라이브러리입니다. mysql 데이터베이스에 접속하거나 쿼리를 보내어 테이블 내 데이터의 CRUD가 가능합니다.

- 주요 함수는 다음과 같습니다.

|  |
| --- |
| **MYSQL\* mysql\_init(MYSQL\* mysql)** |
| mysql 객체를 초기화합니다. 인자가 NULL이면 새로운 MYSQL 객체 생성 후 리턴합니다. |
| - MYSQL\* mysql : 초기화 할 mysql 객체 |

|  |
| --- |
| **MYSQL\* mysql\_real\_connect(MYSQL \* mysql, const char \*host, const char \*user, const char \*password, const char \*db, uint port, const char \*unix\_socket, uint client\_flag)** |
| host에 있는 데이터베이스에 연결을 시도합니다. |
| - MYSQL \*mysql : mysql 객체(구조체)의 주소  - const char \*host : 연결 할 host 주소  - const char \*user : 연결 할 user db계정  - const char \*password : 연결 할 user 계정의 패스워드  - const char \*db : 연결 할 데이터베이스  - uint port : 접속할 port  - const char \*unix\_socket : 소켓 ( 보통은 default 설정 )  - uint client\_flag : flag (0) |

|  |
| --- |
| **int mysql\_query(MYSQL \*mysql, const char \* query)** |
| query를 보내어 수행합니다. query의 끝은 NULL입니다. |
| - MYSQL \*mysql : mysql 객체(구조체)의 주소  - const char \* query : query문을 입력합니다.  - return 값 : |

|  |
| --- |
| **void mysql\_close(MYSQL\* mysql)** |
| 데이터베이스를 닫습니다. |
| - MYSQL\* mysql : 연결 종료 할 mysql 객체 |

|  |
| --- |
| **void mysql\_close(MYSQL\* mysql)** |
| 데이터베이스를 닫습니다. |
| - MYSQL\* mysql : 연결 종료 할 mysql 객체 |

다음은 mysql DB에 연결해서 DB내의 테이블 이름을 출력하는 예제입니다.

#include <stdio.h>

#include <stdlib.h>

#include <mysql.h>

int main(int argc, char \*argv[])

{

// MYSQL 객체

MYSQL \*conn;

MYSQL\_RES \*result;

MYSQL\_ROW row;

char query\_buffer[2048];

conn = mysql\_init(NULL);

// 연결시도 후 “test”라는 Database 선택

if(!mysql\_real\_connect(conn, "localhost", "root", "", NULL, 0, NULL, 0)) {

printf("[+] Can't Connect");

exit(1);

} else {

if(mysql\_select\_db(conn, "test")){

printf("[+] Can't use DB");

exit(1);

}

}

// 테이블명을 알아내는 쿼리문 전송

sprintf(query\_buffer, "%s", "show tables");

if(mysql\_query(conn, query\_buffer)) {

printf("query faild : %s\n", query\_buffer);

exit(1);

}

// 쿼리에 대한 결과 출력

result = mysql\_use\_result(conn);

while ((row = mysql\_fetch\_row(result)) != NULL)

printf("%s \n", row[0]);

mysql\_free\_result(result);

mysql\_close(conn);

// MySQL 버전 출력

printf("[+] MySQL Client Version : %s\n", mysql\_get\_client\_info());

return 0;

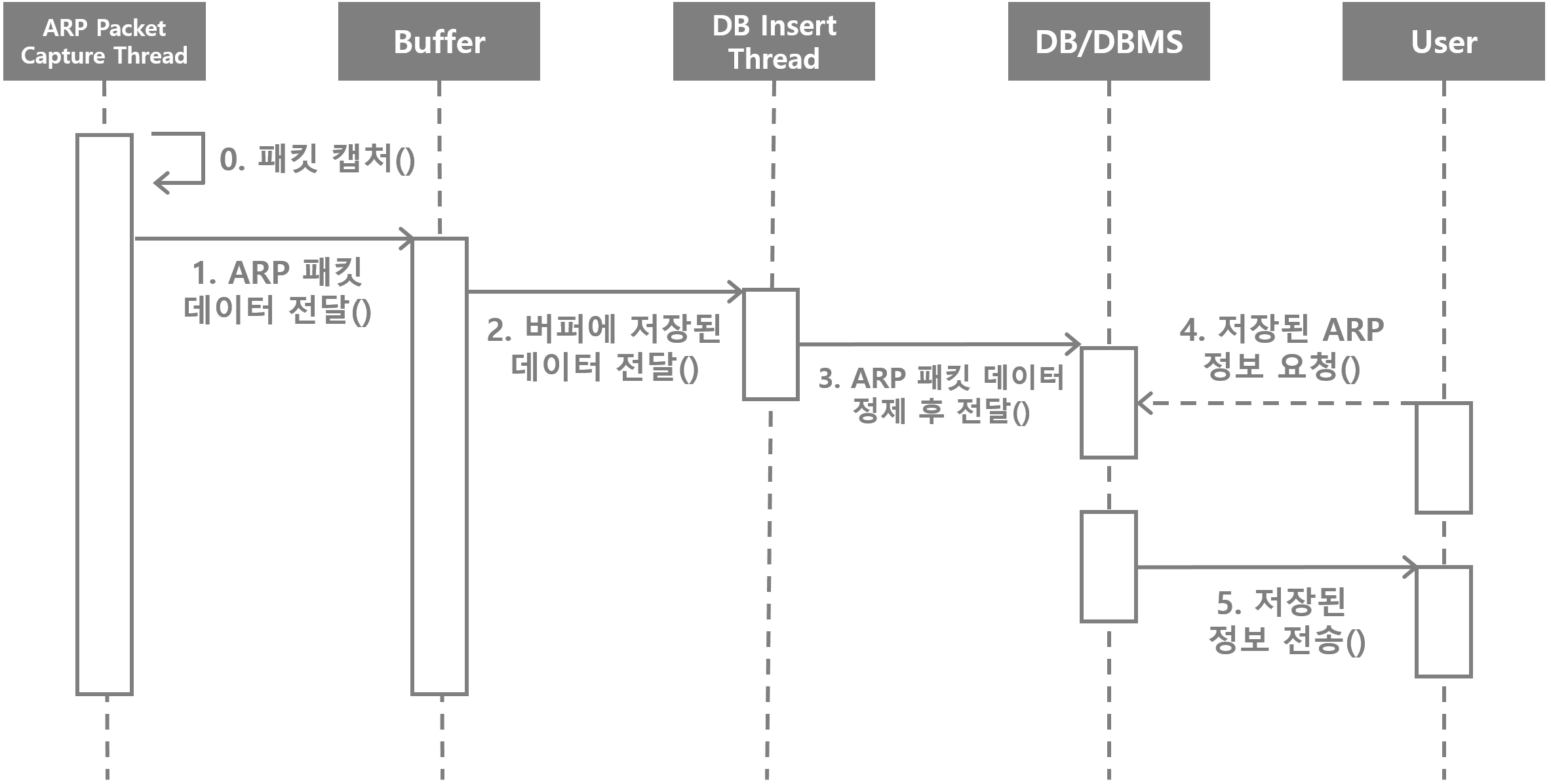
}

# **2. 시스템 설계 및 모델**

해당 section은 시스템 요구사항을 만족할 수 있도록 하는 아키텍처를 설계하기 위함입니다.

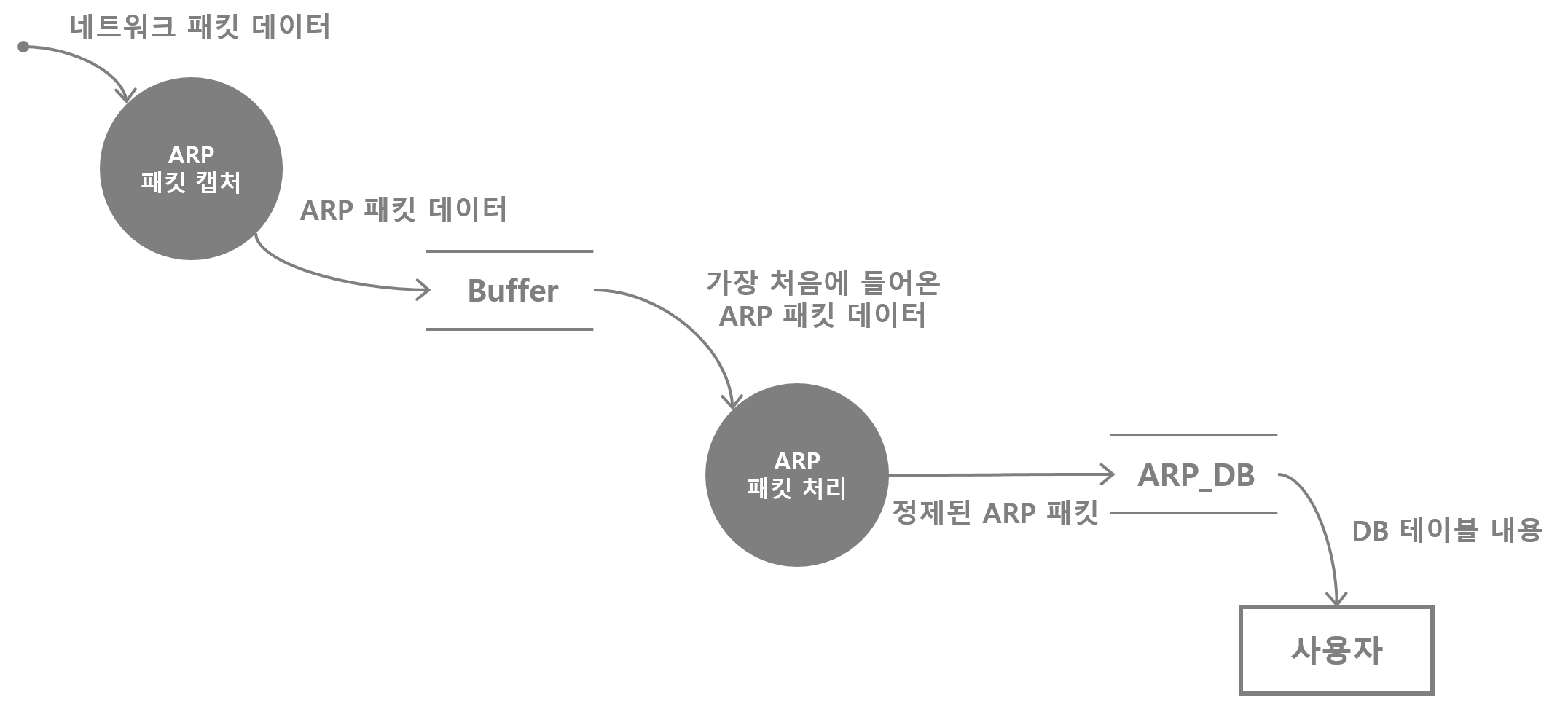
## 2.1 Sequence Diagram

Sequence Diagram은 행동순서를 시간에 따라 보여줍니다. 각각의 역할부분에서 메소드의 사용이 시간 순서대로 나타나게 됩니다. 시스템의 전반적인 작동과정을 한 눈에 알아보기 쉽습니다. Sequence Diagram의 가로축은 객체 혹은 동작의 수명을 나타내며 세로축은 객체의 생성과 사용 순서를 보여줍니다.



## 2.2 Data-flow Model

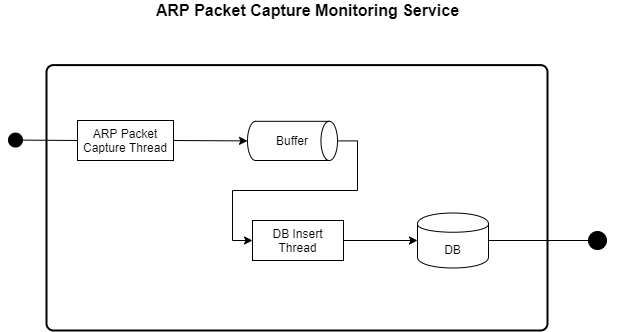
일반 네트워크 패킷 데이터가 어떠한 프로세스를 거쳐 최종적인 ARP 정보가 사용자에게 전해지기까지의 데이터 흐름을 나타낸 모델입니다.



처음 네트워크 내의 무분별한 패킷으로부터 시작됩니다. 여러 패킷 중에서 ARP 패킷만을 필터링 합니다. 이후 ARP 패킷에서 필요한 부분만을 추출하여 데이터베이스에 저장하는 구조로 이루어져있으며 최종적으로 사용자에게는 필요한 ARP 정보만을 보여주게 됩니다.

## 2.3 Block Diagram

해당 프로그램이 진행되는 프로세스를 블록 다이어그램으로 도식화 하였습니다.



전체적으로 ARP 패킷을 캡쳐하는 “ARP Packet Capture thread”, ARP 패킷을 가공하여 DB에 저장하는 “DB insert thread”, 중간에 완충지대 역할을 해 주는 buffer와 저장된 ARP에 대한 정보가 있는 DB로 구성되어 있습니다.

“ARP Packet Capture thread”는 네트워크 내의 패킷을 계속해서 스니핑을 합니다. 이 중에서 ARP 패킷이 검출되면 바로 Buffer로 전달됩니다. Buffer에 전달된 ARP 패킷은 “DB Insert Thread” 에 의해서 DB에 저장하기 위한 구조로 가공된 후에 DB에 저장됩니다. 최종적으로 사용자는 ARP 정보를 데이터베이스에서 조회할 수 있게 됩니다.

# **3. 데이터베이스 설계**

데이터베이스는 mysql을 사용하였습니다. C 프로그램과 데이터베이스 간 데이터 저장을 위해서 mysql 라이브러리를 이용할 예정입니다.

## **3.1 데이터베이스 테이블 설계**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **테이블명** | **arp\_db** | | | |
| **속성** | **속 성** | **필드명** | **데이터 타입** | **비고** |
| Index | idx | int unsigned | PK |
| ARP 타입 | arp\_type | char(10) |  |
| 송신측 MAC 주소 | sender\_mac | char(20) |  |
| 수신측 MAC 주소 | receiver\_mac | char(20) |  |
| 송신측 IP 주소 | sender\_ip | char(20) |  |
| 수신측 IP 주소 | receiver\_ip | char(20) |  |
| 시간 | timestamp | char(20) |  |

[표 1-1] ARP Packet Table

**4. 시스템 종합 분석 방법**

프로그램 구현 후 평가 및 분석 사항입니다. 아래의 항목을 기준으로 분석할 예정입니다.

- 당초 설계목표를 달성하였는가?  
 - 설계 및 개발에서 개선되어야 할 사항은?  
 - 상용화 및 양산 단계에서 개선될 사항은?