論文2001-38TC-3-3

VoIP를 위한 Session Description Protocol 스택에 관한 연구

(A Study on the Session Description Protocol Stack for VoIP)

鄭 成 玉*, 高 光 萬*

(Sung-Ok Jung and Kwang-Man Ko)

요 약

현재 국내·외적으로 VoIP 관련 시장의 지속적인 성장으로 인해 VoIP 표준 프로토콜에 대한 국제 표준화 작업과 병행하여 프로토콜 스택과 같은 소프트웨어 개발이 절실한 상황이다. 본 논문에서는 IETF RFC2327에서 제시한 SDP 문법을 기반으로 문법 지식적 변환 기법으로 SDP 요청 메시지를 구조체 형태로 변환하는 엔코더와 구조체 형태를 갖는 SDP 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현하였다. 이를 위해 IETF RFC2327에 제시한 Augmented-BNF 형식의 SDP 문법을 BNF 형식으로 기술하여 어휘 분석기 및 구문 분석기를 자동 생성한 후 구문 트리를 구성하였다. 또한 SDP 문법으로부터 생성 가능한 모든 규칙에 대한 정보를 저장할 수 있는 구조체 형태의 헤더 파일을 자동 생성하였다.

Abstract

Accordingly it is very important to not only develop the stack of protocol, but also try an international standardization regarding the standard protocol of VoIP. Has compared to the advanced countries having already some success in commercialization, Korea is relatively much less involved in relation to this technology and endeavors. In this regards, this paper is focused on developing a protocol stack made with encoder/decoder, the generator of the header file, syntax analyzer etc. based on the protocol grammars of Session Description Protocol supported by IETF RFC2327. For the sake of it, first describe the SDP BNF grammar based on IETF RFC2327 Augmented-BNF. And then we produce the Abstract Syntax Tree, header file generator for encoding/decoding as applying the method of syntax-directed to SDP protocol grammar.

I. 서 론

데이터 네트워크를 통해 음성 신호와 같은 멀티미디어 데이터를 전달하고자 하는 VoIP(Voice over IP)에 관련된 기술 개발 및 연구가 국내외적으로 활성화되고 있다. 초창기 IP 네트워크는 데이터 송수신을 목적으로 만들어진 네트워크로 아날로그 신호인 음성을 전달하도록 고안되지는 않았지만 IP 네트워크의 월등히 저렴한 비용 때문에 IP 네트워크를 통해 음성을 주고받을 수 있다면 값비싼 국제 전화료 또는 장거리 전화료를 파격적으로 줄일 수 있는 점에 착안해 꾸준히 IP 네트워크상에서 음성을 전달하려는 기술의 진보가 있어 왔다.

특히, 세계의 많은 기업들과 연구진의 참여로 IP 네트워크상에서 비디오, 오디오, 데이터 전송을 위한 표준 프로토콜인 H.323 등을 정의하였다. 이러한 VoIP 관련 시장 및 연구 동향은 국내외적으로 웹 기반의 기술을 통합하는 분야와 국제 표준 프로토콜 표준화 작업 및 프로트콜 스택 개발과 분야의 연구를 통해서 웹

* 正會員, 光州女子大學校 情報通信學部

(Kwang-Ju Women's University, Division of Information Communication)

接受日字 : 2000年 10月31日, 수정완료일 :2001年 2月27日

기반의 다양한 음성 서비스 제공하고자 하는 분야에 집중되어 있다. 인터넷 표준 프로토콜 및 VoIP 표준 프로토콜은 IETF(The Internet Engineering Task Force) 커뮤니티 등을 중심으로 H.323 중심의 ITU-T 표준안 업그레이드, 게이트웨이 제어 관련 프로토콜 국제 표준화 및 프로토콜 스택 개발, RSVP/DiffServ 등 품질(QoS) 보장 기술 적용 방안 반영, H.235 등 보안 기능 강화, Mobile H.323 등 차세대 통신망과의 연동 방안 표준화 작업과 같은 부분을 꾸준히 추진되어오고 있다.[1, 2]

또한 국내외적으로 VoIP 관련 시장의 지속적인 성장으로 인해 VoIP 표준 프로토콜에 대한 표준화 작업과 병행하여 프로토콜 스택의 개발이 절실한 상황이다. 국외에서 게이트웨이를 제어하기 위한 VoIP 표준 프로토콜에 대한 프로토콜 스택의 개발이 진행중이며 일부 제품은 상용화되어 판매되고 있지만 아직은 초기 단계이다. 실제 상용화되어 판매되고 있는 제품은 엄청난 비용을 지불해야 하는 어려움을 가지고 있다. 현재까지 국내에서 MGCP, MEGACO, SIP, SDP 등 게이트웨이 제어 관련 프로토콜 스택의 개발을 위한 연구는 아직까지 열악한 환경과 많은 시간 및 기술을 요하는 관계로 인해 이를 진행하기 위한 시도만 일부 이루어지고 있다.[2]

본 논문에서는 IETF RFC2327[1]에서 제시한 SDP 문법을 기반으로 문법 지식적 변환 기법으로 SDP 요청 메시지를 구조체 형태로 변환하는 엔코더와 구조체 형태를 갖는 SDP 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현하였다. 이를 위해 IETF RFC2327에 제시한 Augmented-BNF 형식의 SDP 문법을 BNF 형식으로 기술하여 어휘 분석기 및 구문 분석기를 자동 생성한 후 구문 트리를 구성하였다. 또한 SDP 문법으로부터 생성 가능한 모든 규칙에 대한 정보를 저장할 수 있는 구조체 형태의 헤더 파일을 자동 생성하였다.

Ⅱ. SDP 특징

1. Session Description Protocol

SDP는 인터넷 커뮤니티상에서 멀티미디어 세션를 기술하기 위해 IETF2327에서 제안되었다. 세션 기술은 "멀티미디어 세션에 참가하거나 멀티미디어 세션을 발견하기 위해 충분한 정보를 전송하기 위해 잘 정의된(well-defined) 형식"이라고 정의할 수 있다. 따라서 SDP는 멀티미디어 세션을 위한 세션 기술 프로토콜이다. SDP가 사용되는 일반적인 형태는 클라이언트가 이미 알려진 멀티캐스트 주소나 포트에 SAP(Session Announcement Protol)을 사용하여 일정한 주기로 컨퍼런스 세션을 알려주며 그림 1과 같이 SDP는 SAP 혼용되어 사용되며 SAP 패킷이라 부른다.[2]

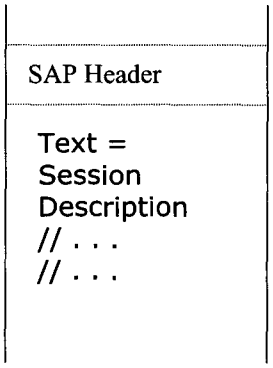

그림 1. SAP 패킷
Fig. 1. SAP Packet.

SDP의 목적은 멀티미디어 세션상에서 정보를 전송하기 위해 사용되므로 세션의 이름, 세션이 활성화되어 있는 시간 등에 대한 정보를 포함하고 있어야 하며 세션에 참여할 수 있도록 충분한 정보를 전송하는 것은 물론 현재, 세션에 참가하지 않고 있는 참가자가 필요로하는 정보를 제공해 줄 수 있어야 한다. 따라서 SDP는 미디어 타입(예, 비디오, 오디오, ...), 전송 프로토콜(RTP/UDP/IP, ...), 미디어 포맷 등과 같은 미디어 정보를 가지고 있어야 한다. 또한 IP 멀티캐스트 세션을 위해서는 미디어에 대한 멀티캐스트 주소 및 미디어 대한 전송 포트 등이 전송되어져야 한다. IP 유니캐스트를 위해서는 미디어의 원격 주소 및 접속 주소를 위한 전송 포트 등이 전송되어져야 한다.

세션은 지정된 시간내에서 결합되거나 결합되지 않을 수 있지만 특정한 시간동안에는 결합되어 있어야 한다. 따라서 SDP는 시작 시점과 정지 시점에서 결합되어 있는 세션에 대한 정보를 전송할 수 있으며 각각 결합된 세션에 대해 특정 시간에 반복되어 전송되도록 지정할 수 있다. 이러한 시간에 대한 정보는 국지적인 시간 정보에 의존하는 것이 아니라 전세계의 시간을 고려하여 지정한다.

※ Session description
v= (protocol version)
o= (owner/creator and session identifier).
s= (session name)
i=* (session information)
u=* (URI of description)
e=* (email address)
p=* (phone number)
c=* (connection information - not required if included in all media)
b=* (bandwidth information)
One or more time descriptions (see below)
z=* (time zone adjustments)
k=* (encryption key)
a=* (zero or more session attribute lines)
Zero or more media descriptions (see below)

※ Time description
t= (time the session is active)
r=* (zero or more repeat times)

※ Media description
m= (media name and transport address)
i=* (media title)
c=* (connection information - optional if included at session-level)
b=* (bandwidth information)
k=* (encryption key)
a=* (zero or more media attribute lines)

그림 2. SDP 필드 이름 및 속성 이름 규칙
Fig. 2. SDP field name and attribute name rule.

v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
a=orient:portrait

그림 3. SDP 메시지 예
Fig. 3. Example of SDP message.

2. SDP 명세(IETF RFC2327)

SDP 세션 기술은 필드 이름과 속성 이름 등으로 그림 2와 같이 3개의 부분으로 기술된다.[1, 2]

위와 같은 필드 이름 및 속성 이름 등을 이용하여 그림 3과 SDP 메시지 예를 볼 수 있다.

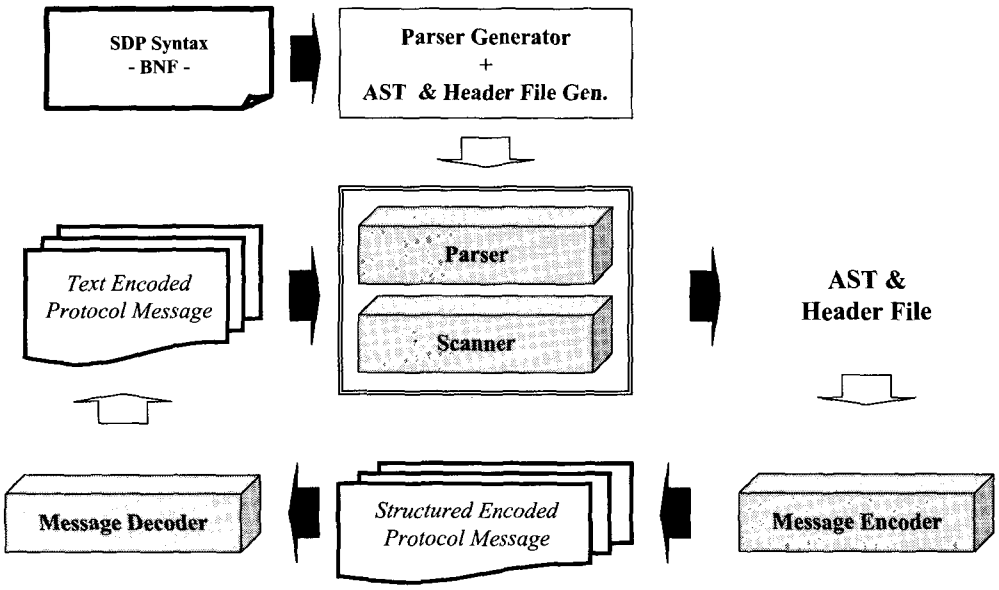

그림 4. MGCP 프로토콜 스택 구현 모델
Fig. 4. Implementation Model of MGCP Protocol Stack.

Ⅲ. SDP 프로토콜 스택 구현

1. 프로토콜 스택 구현 개요

본 논문에서는 IETF RFC2327에서 제시한 SDP 문법을 기반으로 문법 지식적 변환 기법으로 SDP 요청 메시지를 구조화된 형태로 변환하는 엔코더와 구조화된 형태의 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현한다. 이를 위해 BNF 형식의 SDP 문법을 기술하여 구문 분석 동작후 구문 트리 및 헤더 파일을 정형화된 방법으로 생성할 수 있는 프로토콜 스택을 그림 4와 같이 시스템 구성 모델을 설정하고 관련 부분을 구현하였다.

SDP 문법은 bison 파서 생성기에 입력되는 형태로 정형화하여 BNF 형식으로 재구성되었으며 문법 기술시에 구문 트리 생성 및 헤더 파일 생성을 위한 의미 수행 코드를 기술한다. 따라서 파서 생성기는 정형화 방법으로 기술된 SDP에 대한 BNF 문법을 입력으로 받아 어휘 분석기, 구문 분석기, 헤더 파일 생성기 등을 자동으로 생성한다. 또한 입력되는 SDP 텍스트 메시지에 대해 파서는 구문 분석을 수행한 후 구문 트리를 자동으로 생성하며 SDP 텍스트 메시지에 대한 정보를 저장하기 위해 헤더 파일을 생성한다. 헤더 파일 생성기에 의해 자동으로 생성되는 헤더 파일은 문법에서 허용되는 모든 가능한 규칙을 포함하고 있으므로 모든 SDP 메시지 정보를 저장할 수 있다. 따라서 헤더 파일은 입력으로 허용되는 요청 및 응답 메시지에 대한 모든 정보를 저장하고 있는 저장소 역할을 한다. 메시지 엔코더(message encoder)는 입력으로 허용되는 올바른 텍스트 SDP 메시지에 대해 헤더 파일이 저장된 정보를 참조하여 구조체(struct) 형식의 구조화된 메시지를 출력한다. 또한 메시지 디코더(decoder)는 헤더 파일구조에 저장된 구조체 형식의 메시지 정보를 다시 문법 규칙을 적용하여 텍스트 SDP 메시지 형태로 반환하는 역할을 한다.

3. 어휘 분석 및 구문 분석

어휘 분석기 및 구문 분석기는 IETF RFC2327에서 제안한 SDP 문법을 기반으로 기술한 형태를 입력으로 받아 SDP 메시지에 대한 어휘 분석 및 구문 분석을 수행한다. 이를 위해 본 연구에서는 컴파일러 전단부 구성의 자동화 도구인 flex[3]를 이용하여 어휘 분석기를 자동 생성하였으며 bison[4]을 이용하여 구문 분석기를 자동 생성하였다.

어휘 분석기와 구문 분석기는 SDP 요청(request) 메시지에 대해 어휘 분석 및 구문 분석 동작을 수행한 후 올바른 텍스트 메시지에 대해 본 연구에서 설계한 구문 트리(abstract syntax tree) 및 헤더 파일을 생성한다. 구문 트리는 파서 생성기의 출력인 구문 분석기에 입력되는 SDP 메시지에 대한 중간 표현 형태로서 올바른 구조를 가진 메시지에 대한 정보를 저장하고

```
SDP_Info_list          : announcement
                         { $$ = maketree( NULL,$1,NIL) ; }
                       |
                       ;
announcement           : protoVersion originField session_nameField informationField
                         uriField phoneFields
                         { $$ = maketree( NULL,$1,$2,$3,$4,$5,$6,NIL) ; }
                       ;
protoVersion           : T_v TOKEN11 NtDigit TEOL
                         { $$ = maketree( NULL,$1,$2,$3,$4,NIL) ; }
                       ;
originField            : T_o TOKEN11  SuitableCharacter NtDigit WSP_list NtDigit
                         WSP_list netType WSP_list addrType WSP_list addr TEOL
                         { $$=maketree(NULL,$1,$2,$3,$4,$5,$6,$7,
                                        $8,$9,$10,$11,$12,$13,NIL) ;
                         }
                       ;
//중략
phone                  : TOKEN10 NtPDigit space_DIGIT_list
                         { $$ = maketree( NULL,$1,$2,$3,NIL) ; }
                       ;
space_DIGIT_list               : space_DIGIT_list space_DIGIT
                         { $$ = maketree( NULL,$1,$2,NIL) ; }
                       | space_DIGIT
                       { $$ = maketree( NULL,$1,NIL) ; }
                       ;
// 생략
```

그림 5. SDP 메시지 BISON 입력 문법 예
Fig. 5. Example of SDP message BISON input.

있으며 이러한 정보를 이용하여 헤더 파일이 정의된 구조화된 형태의 메시지로 변환된다. 헤더 파일은 구문 트리에 저장된 SDP 요청 메시지 정보를 구조체 형식으로 변환하여 저장하며 발생되는 응답 메시지에 대한 정보를 저장한 후 다시 텍스트 메시지로 변환하기 위한 정보를 저장하는 곳으로 사용된다.

텍스트 형태의 SDP 요청 메시지에 대한 구문 분석을 수행한 후 구문 트리 및 헤더 파일을 생성하기 위해 IETF RFC2327의 Augmented-BNF 문법을 기반으로 하여 파서 생성기인 bison을 입력을 그림 5와 같이 BNF 형식으로 기술하였다.

3. 구문 트리 및 헤더 파일 생성

올바른 SDP 텍스트 메시지 입력에 대해 어휘 분석 및 구문 분석을 수행한 후 구조체 형식을 가진 엔코딩된 메시지를 생성하기 위해 입력 메시지에 대한 그림 6과 같은 노드 구조를 갖는 구문 트리를 생성한다.

```
typedef enum kind { terminal, nonterminal }
n_kind ;
typedef struct nodetype {
        n_kind node_kind ;  //노드 종류
        char  *node_name ; //노드 이름
        struct nodetype *brother ;
        struct nodetype *son ;
} NODE ;
```

그림 6. 구문 트리의 노드 구조
Fig. 6. Node Structure of the Abstract Syntax Tree.

구문 트리 생성에 필요한 노드 구조을 이용하여 본 연구에서 기술한 문법 생성 규칙에 따라 노드를 구성하고 궁극적으로 구문 트리를 생성하기 위해 bison 입력으로 BNF 형식의 SDP 문법과 해당 문법이 축약(reduce) 또는 이동(shift) 되었을 때 구문 트리를 구성하는 동작 코드를 작성하였다. 먼저 노드를 구성하기 위해서는 문법 규칙에서 이동이 발생하는 경우 그림 7

```
NODE *buildnode(struct tokentype token) {
        NODE *ptr;
        ptr = (NODE *) malloc (sizeof(NODE));
        if (!ptr){ printf("buildnode malloc error\n ");
            exit(1);
        }
        ptr->token = token;
        ptr->noderep = terminal;
        ptr->son = NULL;
        ptr->brother = NULL;
        return(ptr);
}
```

그림 7. 노드 구성 함수 : buildNode()
Fig. 7. Node Building Function : buildNode().

에 기술된 buileNode() 함수를 호출하여 노드를 구성한다.

하나의 생성 규칙이 축약되면 그림 8과 같은 buildTree() 함수를 호출하여 구문 트리를 완성한다.

실질적으로 SDP 요청 메시지에 대한 구문 트리 생성 결과는 그림 9와 같은 구조를 갖는 구문 트리를 생성한다.

```
NODE *buildtree(int nodenum, int rhslen)
{
        int i, j, start;
        NODE *n, *first, *temp;
        i =  yyTop - rhslen + 1;
        while (i <= yyTop && valueStack[i] == NULL) i++;
        if (!nodenum && I > yyTop) retrun NULL;
        start = i;
        while (i <= yyTop-1) {
                j = i + 1 ;
                while ((j <= yyTop && valueStack[j] == NULL)) j++ ;
                if (j <= yyTop) {
                   temp = valueStack[i] ;
                   while (temp->brother) temp = temp->brother ;
                   temp->brother = valueStack[j] ;
                }
                i = j;
        }
        first = valueStack[start];
        if (nodenum) {
                temp = (NODE *) malloc(sizeof(NODE));
                if (!temp) { printf(" buildtree malloc error\n");
                   exit(1);
                }
        temp->token.tokennumber = nodenum;
        temp->token.tokenvalue  = NULL;
        temp->noderep = nonterm;
        temp->son = first; n->brother = NULL;
        return(temp);
        }
        else {
                if (i > 0) return NULL;
    else
        return first ;
} /* end buildtree */
```

그림 8. 구문 트리 구성 함수 : buildTree()
Fig. 8. Abstract Syntax Tree Building Function : buildTree().

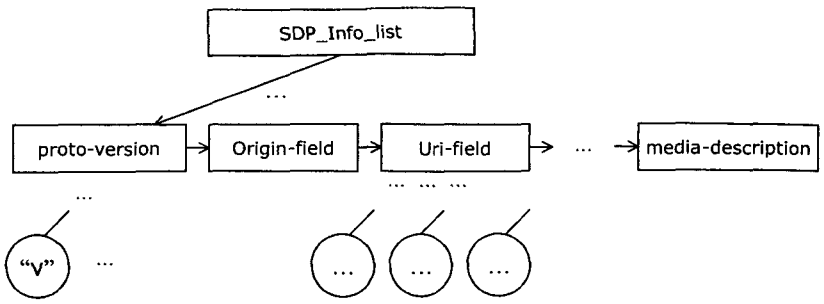

그림 9. 구문 트리 생성 예
Fig. 9. Example of Abstract Syntax Tree.

4. 엔코더 및 디코더

헤더 파일은 SDP 문법으로부터 생성 가능한 모든 규칙에 대한 정보를 저장하기 위해 사용되는 그림 10과 같은 구조체로서 구문 분석기에 의해 자동 생성된다. 이를 위해 본 연구에서는 문법 기술시에 각 문법 규칙으로부터 생성될 수 있는 모든 가능한 규칙에 대한 정보를 저장할 수 있도록 구문 트리 생성 정보를 참조하여 의미 수행 코드를 작성하였다. 따라서 헤더 파일은 SDP 요청 메시지를 구조체 형식으로 변환하여 저장하기 위한 저장소로서 사용되며 또한 구조체에 저장된 응답 메시지를 텍스트 메시지로 변환하여 사용자에게 전달하기 위한 정보를 저장하는 곳으로 사용된다.

입력되는 모든 요청 메시지는 구문 트리를 참조하여 인코더에 의해 구조체 형태로 변환되며 전달되는 응답 메시지도 헤더 파일에 저장된 후 디코더에 의해 사용자에게 텍스트 메시지로 전달된다. 따라서 엔코더는 SDP 문법으로부터 생성되는 헤더 파일에 텍스트 메시지를 구조체 형식에 맞게 변환하여 저장하는 기능을 하며 디코더는 구조체에 저장되는 응답 메시지를 다시 텍스트 메시지로 변환하는 역할을 한다.

실질적으로 적용된 입력에 대한 검증을 위해 그림 3에서 적용하였던 SDP 기술 예제에 대한 구문 트리의 구성은 중간의 불필요한 과정을 생략하고 그림 11과 같은 터널로 구성된 구문 트리를 생성한다.

입력에 대한 SDP 응답 메시지의 최종 출력 결과는 그림 10의 헤더 파일에 입력되는 내용을 추적하여 다시 텍스트 메시지로 변환하는 과정을 통해 확인하였으며 입력되는 요청 메시지와 똑같은 결과를 생성하는지를 통해서 본 시스템의 성능을 검사하였다.

```
typedef struct SessionInformationSDP {
    unsigned short  length;
    unsigned char   value[16];
} SessionInformationSDP;

typedef enum AddressType {
    ip4 = 0,
    ip6 = 1,
    otherAddressType = 2
} AddressType;

typedef struct OriginSDP {
    struct {
        unsigned short  length;
        unsigned char   value[16];
    } userName;
    int             seesionID;
    int             version;
    NetworkType     networkType;
    AddressType     addressType;
    struct {
        unsigned short  length;
        unsigned char   value[16];
    } address;
} OriginSDP;

typedef struct UriISDP {
    unsigned short  length;
    unsigned char   value[32];
} UriISDP;

// 중략

typedef struct ConnectionDataSDP {
    NetworkType     networkType;
    AddressType     addressType;
    struct {
        unsigned short  length;
        unsigned char   value[16];
    } cnxAddress;
} ConnectionDataSDP;
```

그림 10. 헤더 파일의 구조
Fig. 10. Structure of the Header File.

[T] v=0
[T] o=mhandley
[T] 2890844526
[T] 2890842807
[T] IN
[T] IP4
[T] 126.16.64.4
[T] s=SDP
. . .

그림 11. 구문 트리 생성 예
Fig. 11. Example the Abstract Syntax Tree Generation.

또한 요구되는 성능에 관한 검사를 SUN Ultra 60, Solaris 7 환경에서 엔코딩 시간과 디코딩 시간이 평균 1msc이내에서 처리되도록 하였고 95%이상의 성공률의 결과를 얻었으며 이러한 메시지 변환 작업을 10만 이상 수행하여 만족할 만한 결과를 볼 수 있다.

Ⅳ. 결론 및 향후 연구과제

현재 국내외적으로 VoIP 관련 시장의 지속적인 성장으로 인해 VoIP 표준 프로토콜에 대한 표준화 작업과 병행하여 프로토콜 스택의 개발이 절실한 상황이다. 국외에서 게이트웨이를 제어하기 위한 VoIP 표준 프로토콜에 대한 프로토콜 스택의 개발이 진행중이며 일부 제품은 상용화되어 판매되고 있지만 아직은 초기 단계이다. 실제 상용화되어 판매되고 있는 제품은 엄청난 비용을 지불해야 하는 어려움을 가지고 있다. 현재까지 국내에서 SDP, MEGACO, SIP, SDP 등 게이트웨이 제어 관련 프로토콜 스택의 개발을 위한 연구는 아직은 열악한 환경과 많은 시간 및 기술을 요하는 관계로 인해 이를 진행하기 위한 시도만 일부 이루어지고 있다.

본 논문에서는 IETF RFC2327에서 제시한 SDP 문법을 기반으로 문법 지식적 변환 기법으로 SDP 요청 메시지를 구조체 형태로 변환하는 엔코더와 구조체 형태의 응답 메시지를 텍스트 형태로 변환하는 디코더를 구현하였다. 이를 위해 BNF 형식의 SDP 문법을 기술하여 구문 분석 동작 후 구문 트리 및 SDP 문법으로부터 생성 가능한 모든 정보를 저장할 수 있는 구조체 형태의 헤더 파일을 자동 생성하였다. 또한 요구되는 성능에 관한 검사를 SUN Ultra 60, Solaris 7 환경에서 엔코딩 시간과 디코딩 시간이 평균 1msc이내에서 처리되도록 하였고 95%이상의 성공률의 결과를 얻었으며 이러한 메시지 변환 작업을 10만 이상 수행하여 만족할 만한 결과를 볼 수 있다.

본 논문에서 시도한 문법 지시적 변환 방법을 이용하여 VoIP 표준 프로토콜 문법에 대한 프로토콜 스택에 관한 연구는 현재까지 SDP에 국한되어 시도되었지만 앞으로 MGCP, MEGACO, SIP 등과 같은 프로토콜과 연관지어 본 연구에서 제안한 모델을 기반으로 구현할 예정이다. 본 연구에서 시도한 개발 방법과 성능은 아직 체계화되지 않은 SIP, MGCP 등의 프로토콜 스택 개발에 적극 활용될 수 있다.

참 고 문 헌

[1] IETF, RFC2327, SDP : Session Description Protocol.
[2] ITU-T, APC-1855, Proposal For an Advanced Audio Server Package for H.248.
[3] FLEX : http://www.combo.org/lex_yacc_page/.
[4] BISON : http://www.combo.org/lex_yacc_page/.
[5] http://www.netmanias.com.
[6] http://www.hsswrold.com.
[7] http://www.prptocols.com.
[8] http://www.catapult.com.
[9] http://www.ietf.com.
[10] http://www.iab.org/iab/.
[11] http://www.iana.org/.
[12] http://www.irtf.org/.

저 자 소 개

鄭 成 玉(正會員)

1987년 2월 조선대학교 전자계산학과 졸업(이학사). 1989년 2월 조선대학교 전자계산학과 졸업(이학석사). 2001년 2월 조선대학교 전자계산학과 졸업(이학박사). 1992년~1997년 2월 광주여자전문대학 전산정보처리과 조교수. 1997년 3월~현재 광주여자대학교 정보통신학부 조교수 재직. 주관심분야는 소프트웨어 공학, 객체지향 소프트웨어, 시스템 소프트웨어, 프로그래밍 언어론, 시스템분석 및 설계

高 光 萬(正會員)

1991년 2월 원광대학교 전자계산공학과 졸업(공학사). 1993년 2월 동국대학교 컴퓨터공학과 졸업(공학석사). 1998년 2월 동국대학교 컴퓨터공학과 졸업(공학박사). 1998년 3월~현재 광주여자대학교 정보통신학부 전임강사 재직. 주관심분야는 객체지향 프로그래밍 언어, 병행 프로그래밍 언어, 원도우즈 프로그래밍 언어, 시각 프로그래밍 언어, 컴파일러 구성론