복합화력 발전소의 PMS 서버와 Client 간의 통신 패킷 추출 기법에 관한 연구

강필순* · 현석환* · 차동진* · 정재화**· 서석빈**· 안달홍**

한밭대학교 · 한국전력연구원

Extraction Technique of Communication Packet between PMS server and Clients in Combined Cycle Power Plant

Feel-soon Kang* · Surk_Hwan Hyun* · Dong-Jin Cha* · Jae-Hwa Chung** · Seok-Bin Seo** · Dal-Hong Ahn**

*Hanbat National University · **Korea Electric Power Research Institute

E-mail : feelsoon@hanbat.ac.kr

요 약

본 논문에서는 특정 서버와 클라이언트 간의 통신 패킷을 추출하는 새로운 방법에 대하여 제안한다. 추출된 통신 패킷은 일반적인 클라이언트가 특정 서버에 접속할 경우에 요구되는 드라이버를 제작하는 소스로 활용된다. 제안된 방식은 중계서버를 이용하여 동일한 네트워크 상에 연결된 많은 통신 패킷 중 필요한 패킷만을 추출하는데 이용된다. 제안된 방식은 복합화력 발전소의 PMS 서버와 ProDAS 클라이언트 사이의 패킷 추출을 통해 검증하였다. 제안된 방식은 일반적인 서버 시스템과 클라이언트 간의 통신 패킷 추출에도 응용할 수 있다.

ABSTRACT

A new packet extraction program is presented to extract communication packet between a server and clients. The extracted packet source is used to develope a special driver to access general clients to the server. The proposed scheme employs a relay server to take a specified packet among a large number of packets on the same network. The proposed method is tested in PMS (Plant Management System) server and ProDAS (Process Data Aquisition and Analysis System) in a combined cycle power plant. The developed scheme can be applied for extracting a specified communication packet between the general server and client.

키워드

Gas Turbine, Plant Management System (PMS), Communication Packet Extraction, Client

Ⅰ. 서 론

국내의 복합화력발전 설비기술은 국외기술의존도가 높아 기술료 징수, 정기/수시 점검, 부품교체 등의 비용으로 인한 경제적 부담이 상당한 실정이다. 이러한 문제점을 해결하기 위해서 한전 전력연구원 발전연구실의 연소열공학그룹을 중심으로 발전 장비 및 관련 기술의 국산화를 추진중에 있다. 국내 기술력 확보는 크게 가스터빈의 설계, 제작 등과 관련된 기계적인 부분과 운용 및 관리와 관련된 계측제어 분야로 구분할 수 있다.

복합화력발전 설비의 핵심인 가스터빈은 감시, 이상 진단, 제어 등을 위해 실시간으로 관리 되어야 하며 가스터빈에 설치된 센서들로부터 수집된 데이터는 중앙관리를 위한 서버로 전송되고 DB (Data Base)화 되어 관리되며, 클라이언트의 요구가 있을 경우에는 언제든지 DB의 데이터를 전송할 수 있어야 한다. 특히 가스터빈의 경우 정기적으로 전체 분해를 통한 정비를 수행하게 되며, 재조립 시에는 기존 정상 상태의 운전 조건을 만족

시키기 위한 튜닝이 필수적이다. 상용 가스터빈의 튜닝시 ProDAS (Process Data Aquisition and Analysis System) 장비가 활용되는데 이 장비는 PMS (Plant Management System) 서버로부터 실제 운전 중인 다른 호기의 데이터를 취득하기 위해 PMS 서버에 접속할 수 있는 기능을 갖추어야 한다[1]. 따라서 서버와의 접속과 원활한 통신을 위한 프로그램이 요구되며, 이를 위해서는 우선 상용 서버인 PMS와 ProDAS 간의 통신 패킷을 추출하여야 한다. 기존에 일반적으로 이용되던 통신 패킷 추출 프로그램은 네트워크에 접속하여 네트워크 상에 전송되는 모든 패킷을 추출하기 때문에 불필요한 패킷까지 모두 추출해 버려 실제 필요한 ProDAS와 PMS 서버 사이의 패킷만을 추출하기에는 많은 문제점이 발생한다.

따라서 본 논문에서는 네트워크 상에서 특정한 두 기기만의 통신 패킷만을 추출할 수 있는 중계서버를 이용한 패킷 추출 방법을 제안한다. 제안된 방식은 접속을 위한 드라이버 파일을 제공 받을 수 없는 경우, 일반적인 서버와 클라이언트 간의 접속을 위한 유사 프로그램을 작성하는데 있어서도 응용이 가능하다.

II. 중계서버를 이용한 통신 패킷 추출

일반적인 복합화력발전소에 설치된 내부 네트워크에는 여러 서버와 연동되어 통신을 주고받는 다양한 통신 패킷으로 인하여 일반적인 패킷 취득 방법으로는 분석이 불가능하다.

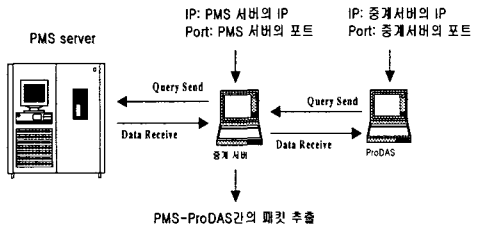

그림 4. 중계서버를 이용한 패킷 추출

그림 1은 복합화력발전소에 설치된 PMS 서버와 ProDAS 간의 패킷을 분석하기 위한 방법을 그림으로 보여준다. PMS 서버와 ProDAS 사이에 PC를 한대 더 장착 시키고 중계 서버 프로그램을 구동한다. ProDAS는 중계서버에 접속하고 중계서버는 ProDAS로부터 전해지는 Query 문장을 PMS 서버로 전송하게 된다. 이때 중계 서버를 통해 ProDAS와 PMS 서버만의 패킷을 전송 받을 수 있게 된다. 따라서 중계서버에는 PMS 서버의 IP address와 허용 포트를 할당하여야 한다. Query 문장이 중계서버를 통해 PMS 서버로 요청되면 PMS는 Binary로 구성된 이들 Query 문장에 해당하는 Action을 취하게 되어 해당 데이터의 전송을 시작하게 된다. 이 경우 역시 중계서버를 통해 ProDAS에 전송되게 되므로 ProDAS와 PMS 서버 둘 사이의 모든 통신 패킷은 중계서버에 기록되게 된다.

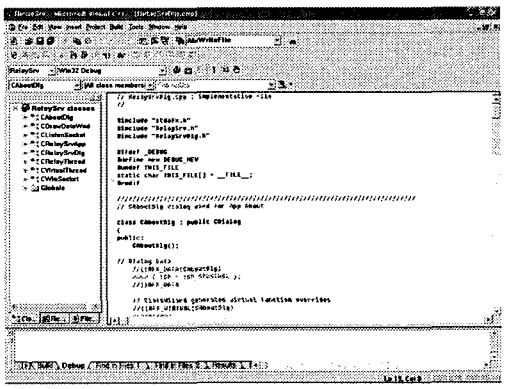

그림 5. Relay sever 프로그램 환경

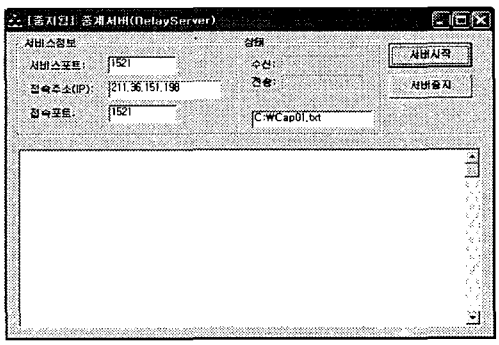

그림 6. 중계서버 실행창

그림 2는 ProDAS와 PMS 서버간의 패킷을 추출하기 위한 프로그램 화면을 보여준다. 본 프로그램은 그림 1의 패킷 추출 원리를 Visual C++ (ver 6.0)에 의해 구현한 것이다. 주요 프로그램 내용은 부록에 정리하였으며, 주요부는 주석문으로 표시하여 이해를 돕도록 하였다.

그림 3은 중계서버의 패킷 추출 프로그램을 실행 시킬 경우 형성되는 팝업 창이다. 서비스 정보란에 서비스 포트(중계서버 PC의 포트번호 부여), 접속주소 (PMS 서버의 주소), 접속포트 (PMS 서버의 포트) 번호를 할당하면 된다. 상태 부분에는 수신과 전송 상태를 표시하도록 구성하여 패킷 송수신의 유무를 쉽게 판단할 수 있도록 구성하였으며, 추출된 패킷을 저장시키는 파일 저장 경로와 이름을 지정할 수 있는 Edit 창을 구성하도록 하였다. 구동시 주의할 점은 ProDAS와 PMS 서버간의 통신 시작 전에 반드시 서버시작 버튼을 눌러 대기 상태에 있어야 한다는 점이다.

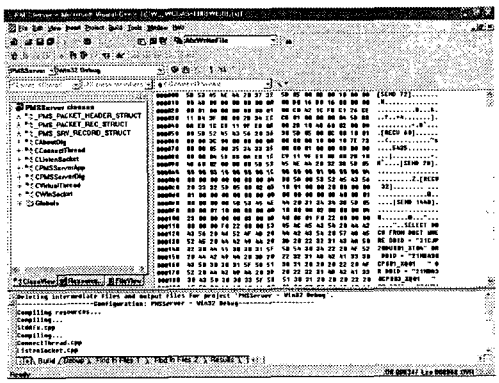

그림 7. ProDAS와 PMS간 통신 패킷 내용

그림 4는 패킷 추출 프로그램을 이용하여 획득한 ProDAS와 PMS 서버간의 통신 패킷을 보여준다. Binary 형태로 주어지기 때문에 보이는 값만으로는 어떠한 정보를 주고받는지에 대하여 알 수가 없다. 하지만 그림 4의 오른쪽 부분을 보면 SEND와 RECV 된 Binary 값을 알 수 있고 이들 규칙을 동일하게 처리할 수 있는 프로그램을 작성하게 되면 기존의 ODBC 드라이버와 동일한 동작을 수행하는 프로그램을 작성할 수 있다는 결론을 얻는다. 특히 ProDAS에서 PMS로 보내는 값들만 동일하게 전송한다면 사용자가 요구하는 데이터를 정확히 전송 받을 수 있다. 예를 들어 PMS와의 접속을 위해서는 SEND 72를 통해 패킷에서 보듯이 72개의 Binary 값을 보내면, RECV 60 즉, 60개의 Binary 값을 PMS 서버로부터 전송받게 된다. Binary 값이 정확히 어떠한 규칙에 의해 만들어 져야 하는지는 중요하지 않다. 왜냐하면 결과적으로 동일한 방법으로 Query 문장을 PMS 서버에게 전달하게 되면 PMS 서버는 약속된 규칙대로 데이터를 전송하기 때문이다. 일반적인 서버와 클라이언트의 경우도 동일하다. 그림 4에서 보면 원하는 데이터 값을 전송 받기 위해서 Query 문장은 각각 원하는 데이터의 Tag list 값을 가지고 있음을 확인할 수 있다. 분석 결과 ProDAS는 274개의 데이터를 Query 문장을 통해 요청하게 되며, PMS 서버는 276개의 데이터를 순차적으로 전송하게 된다.

III. 결 론

본 논문에서는 네트워크 상에서 특정한 두 기기간의 통신 패킷만을 추출할 수 있는 중계서버를 이용한 패킷 추출 방법을 제안하였다. 복합화력발전소의 PMS 서버와 가스터빈 상용 테스트 장비인 ProDAS 간의 통신패킷을 취득하여 패킷 취득 방법과 프로그램의 실용성을 검증하였다. 제안된 방식과 코딩된 프로그램은 접속을 위한 드라이버 파일이 제공되지 않는 경우, 일반적인 서버와 클라이언트 간의 접속을 위한 통신 패킷 분석에 있어서도 충분한 응용이 가능하다.

V. 부 록

```
m_pRelayThread = NULL;
m_nListenPort = 1521; //접속포트 번호를 할당
m_strServerIp = "211.36.151.198";
        //접속할 PMS 서버의 IP Address를 설정
m_nServerPort = 1521;
        //중계서버의 서비스 포트 할당
CString strValue;
strValue.Format( _T("%d"), m_nListenPort );
m_EditCtrl01.SetWindowText( strValue );
m_EditCtrl02.SetWindowText( m_strServerIp );
strValue.Format( _T("%d"), m_nServerPort );
m_EditCtrl03.SetWindowText( strValue );
        //받은 패킷을 저장시킬 경로와 이름 지정
m_strSaveFileName = "C:\\Cap01.txt";
m_SaveFileCtrl.SetWindowText (m_strSaveFileName);
        //프레임 영역 설정
CRect rtRect;
GetClientRect( &rtRect );
rtRect.left    = 13;
rtRect.top     = 145;
rtRect.right  -= 13;
rtRect.bottom -=  13;
m_ListWndCtrl.CreateEx(WS_EX_CLIENTEDGE, _T
("STATIC"), "DataListWnd", WS_CHILD| WS_
            VISIBLE, rtRect, this, IDC_LIST_WND );
}

void CRelaySrvDlg::OnStartButton()
{
 if( m_bListenStart == TRUE ) return;
 CString strValue;
 m_EditCtrl01.GetWindowText( strValue );
 m_nListenPort = atoi( (LPCTSTR)strValue );
 m_EditCtrl02.GetWindowText( m_strServerIp );
 m_EditCtrl03.GetWindowText( strValue );
 m_nServerPort = atoi( (LPCTSTR)strValue );
 m_SaveFileCtrl.GetWindowText(    m_strSaveFileName
);          // 서버의 서비스 포트를 설정
 if(  m_ListenSocket.Create(  m_nListenPort,  SOCK_
STREAM ) )
  {
        // 클라이언트의 Connect를 wait
 if(        m_ListenSocket.AsyncSelect(        m_hWnd,
FD_ACCEPT | FD_CLOSE ) )
  {
    if( !m_ListenSocket.Listen(1) )
      {
       AfxMessageBox( "*** 리슨 시작 오류 ***",
                MB_OK | MB_ICONERROR );
       m_ListenSocket.Close();
       return;
      }
  }
    else
      {
```

```
        m_bListenStart = TRUE;
          //프레임 캡션 지정
        SetWindowText(        "[시작됨]        중계서버
(RelayServer)" );
      }

void CRelaySrvDlg::OnStopButton()
{
  if( m_bListenStart != TRUE ) return;
  m_ListenSocket.Close();
  if( m_pRelayThread )
   {
     m_pRelayThread->m_bLoop = FALSE;
     Sleep( 200 );              // Time Delay
     delete m_pRelayThread;
     m_pRelayThread = NULL;
   }
  m_bListenStart = FALSE;
  SetWindowText( "[중지됨] 중계서버(RelayServer)" );
}

void CRelaySrvDlg::OnDestroy()
{
  if( m_pRelayThread )
   {
    delete m_pRelayThread;
    m_pRelayThread = NULL;
   }
  CDialog::OnDestroy();
}

LRESULT CRelaySrvDlg::OnEventMessage(WPARAM
          wParam, LPARAM lParam)
{
  if (WSAGETSELECTERROR(lParam) != 0)
    {
      return 0;
    }
  switch (WSAGETSELECTEVENT(lParam))
    {
      case FD_ACCEPT:
        {
          if( m_pRelayThread ) return 0;
          m_pRelayThread = new CRelayThread();
          CWinSocket          *pWinSocket          =
m_pRelayThread

->m_pClientSocket;
          if( pWinSocket->Accept( m_ListenSocket.m_
                                  hSocket,      NULL,
NULL ) )
          {
          pWinSocket = m_pRelayThread->

m_pServerSocket;
          // 소켓을 생성 한다.
          pWinSocket->Socket( SOCK_STREAM,
                                        IPPROTO_TCP,
PF_INET );
          if( pWinSocket->Connect( (LPCSTR)m_
                                        strServerIp,
m_nServerPort ) )
          {
          // Thread를 생성 한다.
          if(       m_pRelayThread->CreateBeginThread(
NULL ) )
          {
          // Thread를 시작(RUN) 한다.
          m_pRelayThread->ResumeThread();
          break;
          }
          }
    }
  m_pRelayThread->Destroy();
  delete m_pRelayThread;
  m_pRelayThread = NULL;
  break;
  }
  case FD_CLOSE:
        {
          // close the client socket
          m_ListenSocket.Close();
          break;
        }
          default : break;
}
   return 0;
}
```

참고문헌

[1] H. Boller, R. Jeckel, and H. Meister, "ProDAS Technical Instructions", ABB Power Generation Ltd., pp. 1-7, Jan. 1998.

[2] P. Biedermann, P. Gygax, and R. Thonen, " ProDAS Configurations for the ProDAS-IMS Link 1AHX611 881," ALSTOM POWER Ltd., pp. 1-15, July 2001.