네트워크 프린터 환경에서 실시간 오류검지 서비스 설계 및 구현

김종필*, 여성구**, 최진영*
*고려대학교 디지털정보공학과
**안철수연구소 보안 컨설턴트
kjp007@korea.ac.kr*, barami@ahnlab.com**, choi@formal.korea.ac.kr*

The Realtime Error Detection Design and Implementation in Network Printing Environment

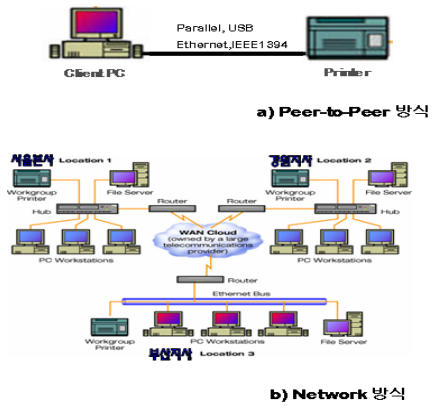
Jong-Pil Kim*, Sung-Koo Ryeo**, Jin-Young Choi*
*Dept. of Digital Information Engineering, Korea University
**Security Consultant Group, Ahnlab, Inc.

요 약

네트워크 프린터 방식은 가정 및 사무환경 내부의 각종 출력 장치를 하나의 통신망으로 통합하여 사용할 수 있는 프린터 솔루션을 제공한다. 네트워크 프린터 환경은 사용 인원 증가로 인한 각종 장애 요인 발생으로 실시간 오류 검지 서비스의 필요성이 요구 되었다. 본 논문에서는 네트워크 프린터 환경의 장애 발생 시에 대한 문제점을 빠르고 정확하게 판단할 수 있도록 오류검지 시스템을 설계 및 구현하였다. 이를 위해 Winsock을 통해 API 설정을 하였고, Packet 분석을 통해 장애를 발생 시켰던 Client PC의 IP를 참조하여 문제를 해결함으로서 네트워크 프린터 솔루션의 효율적인 관리를 할 수 있다.

1. 서 론

네트워크 환경이 발달하기 전에는 컴퓨터와 프린터 간의 접속 방식을 P2P 방식으로 연결하는 것이 대부분 이었지만, 현재의 환경 에서는 네트워크 자원의 효율적인 관리를 위해서 NetBIOS 및 TCP/IP Protocol을 이용한 네트워크 환경에 프린터를 연동시켜 프린터 서비스를 이용하고 있다.



(그림 1) Peer-to-Peer 연결 방식과 네트워크 연결방식

최근 주종을 이루고 있는 네트워크 프린터 방식은 유지비용의 절감 및 프린터 장비의 관리가 용이하며, 작업 공간의 절약과 다양한 컴퓨터 기종 및 Windows, Unix, Linux, Mac OS와 같은 운영체제를 동시에 연결하여 사용할 수 있는 장점이 있다. 하지만, Printer Driver 설치 오류 및 각 사용 기능에 대한 설정 불량과 바이러스 및 악성 코드에 의해 데이터의 손실이 일어날 경우 프린터 서비스의 장애 발생 및 네트워크 환경의 장애를 가져올 수 있다는 문제점을 안고 있다.

이렇게 프린터 환경의 장애를 가져올 수 있는 많은 환경적 요인들을 전산장비 관리자 및 네트워크 관리자가 실시간으로 검지하여 문제점을 분석하고 해결하기 위한 오류검지 서비스의 필요성이 날로 높아져 가고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 프린터 출력 데이터 Flow와 각 Protocol에 대한 출력 방법에 대하여 기술하며, 3장에서는 네트워크 프린터 환

경의 문제점을, 4장에서는 실시간 오류검지 서비스의 설계 및 구현 방법을, 5장에서는 서비스에 대한 성능을 평가 하였으며, 6장에서 결론을 맺는다.

2. 관 련 연 구

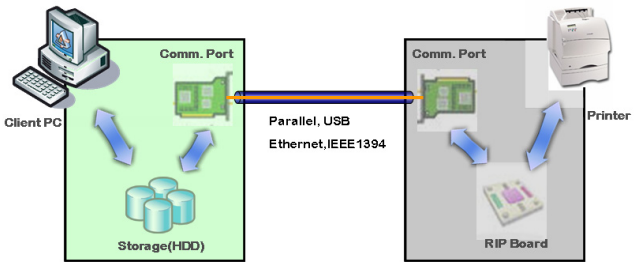
2. 1. 프린터 출력 데이터 Flow

각 컴퓨터 내부에 있는 Application Program에 의해 만들어진 출력 데이터의 값은 Emulation 데이터로 변환하고 이를 저장장치(Storage Unit)로 전송하여 출력 데이터를 생성한다. 이를 Spooling이라 하며, 이 데이터 들은 전송 선로를 이용하여 Printer Port로 전송된다.

Printer Port로 수신된 데이터는 프린터의 Rip Board 내부에 장착되어 있는 RAM에 저장된다.

한 장의 데이터가 수신되면 프린터 언어 해석기에서 PJL Code를 이용하여 Emulation 언어로 작성되어 있는 데이터를 번역하여 출력 가능한 문서로 만드는 작업을 하는데, 이를 Ripping이라 한다.[1]

이 데이터들은 프린터 내부의 Engine Unit를 제어하며 완성된 출력물을 만들게 된다.



(그림 2) 프린터 출력 데이터 Flow

2. 2. Windows Printer Protocol Stack

SMB(Server Message Block)는 전송 Protocol로 NetBIOS를 사용하며, <표 1>과 같이 Link Layer에서 직접 구동 하거나 IPX/SPX, TCP/IP 같은 Network Link Layer 에서 구동 할 수 있다.

NetBIOS는 작고 복잡한 환경에서 관리자가 항시 대기할 필요가 없게 설계 하였으며, 원래 TCP/IP가 널리 퍼지기 전에 LAN에 물린 소규모 작업 그룹을 연결 하려는 취지로 개발 했지만, TCP/IP 등장 이래로 네트워크 환경을 변화 시키는 작용을 하였다.

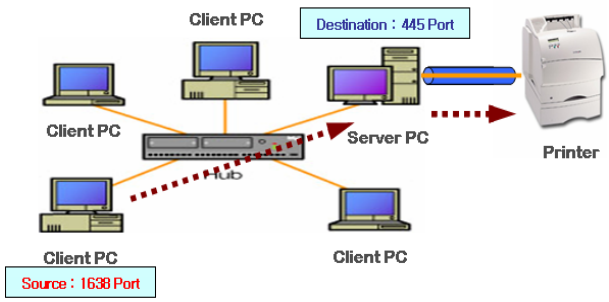
TCP/IP를 통한 NetBIOS는 NBT(NetBIOS over TCP/IP)로도 불리는데, 세 가지 서비스(NetBIOS Name Service, Datagram, Session Service)로 이루어진다.[2]

<표 1> SMB Protocol Stack

SMB(Server Message Block)		
TCP/IP	NetBIOS	IPX/SPX
Link Layer (802.3, PPP)		
Physical Layer (Ethernet, Token Ring)		

2. 3. NetBIOS Protocol 출력 방법

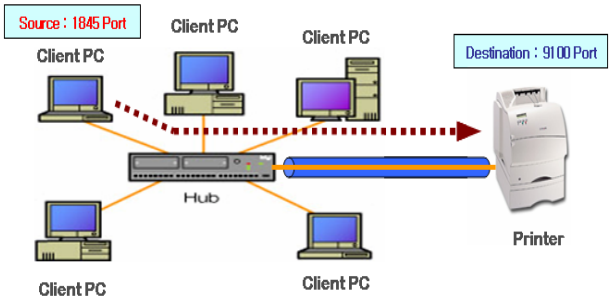
NetBIOS Protocol을 이용하여 출력하는 방법은 네트워크로 연결되어 있는 각각의 Client PC들이 출력 명령을 받아 1638 Port를 이용하여 데이터를 송신하고, Server PC의 445 Port를 이용하여 데이터를 수신하며, Server PC 내부에서 수신된 데이터의 Spooling 작업을 진행하여 프린터에 전송한다.



(그림 3) NetBIOS Protocol Printing 기법

2. 4. TCP/IP Protocol 출력 방법

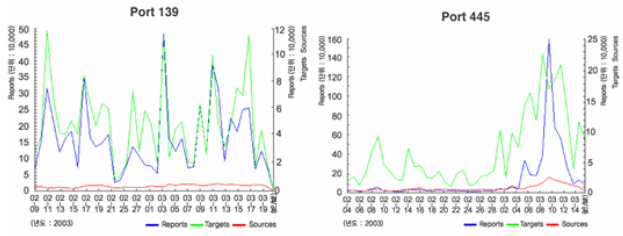
TCP/IP Protocol을 이용하여 출력하는 방법은 Client PC들이 1845 Port를 이용하여 Spooling 데이터를 전송하고, 네트워크에 연결되어 있는 프린터의 NIC(Network Interface Card) Option Port를 이용하여 9100 Port를 통해 데이터를 수신하며, 프린터 내부에서 Ripping 작업이 실행된다.



(그림 4) TCP/IP Protocol Printing 기법

3. 네트워크 프린터 환경의 문제점

네트워크 프린터 환경의 변화에 따라 각 사용자의 프린터 사용 범위가 확대 되고, 전산장비 및 네트워크 관리자의 관리 범위가 확대 되면서 각 Client PC의 드라이버 설치 오류와 사용자의 잘못된 설정에 의한 문제점이 증가 하였지만, 드라이버 설치방법 및 사용자 설정 방법의 공지로 인하여 네트워크 프린터 환경의 장애를 줄일 수가 있다. 네트워크 프린터 사용 환경과는 거리가 있는 컴퓨터 작업 및 Outlook의 사용과 같은 E-mail 확인에 의해서도 Win32/Bugbear.Worm.50688 등과 같은 프린터 환경에 영향을 미치는 바이러스에 System이 감염되어, 네트워크 및 Local에 연결되어 있는 프린터에 쓰레기 데이터를 전송하여 잘못된 출력물을 내보낸다.[3]



(그림 5) 139 Port, 445 Port의 Network Traffic 발생량

또한 2003년에 발생된 1.25 대란과 같이 NetBIOS와 SMB에 관련된 139 Port 및 445 Port에 대하여 (그림 5)와 같이 대량의 Network Traffic을 발생하게 되어 Internet 장애와 네트워크 장비들의 기능 마비를 일으키게 되었다.[4]

네트워크 프린터의 원활한 사용을 위한 문제 해결 방법으로 네트워크 프린터 사용 시 데이터의 경로로 사용되는 Port에 대해 문제점이 발견 되었을 경우 문제 해결을 위하여 수신 Port에서 이동하는 데이터의 실시간 Monitoring 및 Log File의 자료가 필요하다.

4. 실시간 오류검지 서비스 설계 및 구현

TCP/IP 환경의 네트워크에서 상호 통신이 가능하도록 API(Application Programming Interface) 설정을 구축 하여야 한다.[5] Windows 상에서 서비스 구현이 가능한 것은 (그림 6)과 같은 windows socket의 Code를 사용하여 통신이 가능한 Winsock을 이용, 프린터 서비스와 네트워크 접속이 가능 하도록 하였다.

```
UINT _stdcall CMySocket::accept(void* owner)
{
CMySocket* pOwner;
pOwner = (CMySocket*)owner;
SOCKADDR_IN ClientAddress;
SOCKET ClientSocket;
int AddressSize = sizeof(ClientAddress);
WSANETWORKEVENTS event;
WSAEVENT hRecvEvent = WSACreateEvent();

WSAEventSelect(pOwner->m_Socket,hRecvEvent,FD_ACCEPT|FD_READ|FD_CLOSE|FD_WRITE);

bool EventLoopStopFlag = false;
while(!EventLoopStopFlag)
{
  Sleep(10);
  WSAEnumNetworkEvents(pOwner->m_Socket,hRecvEvent,&event);

  if((event.lNetworkEvents & FD_ACCEPT) == FD_ACCEPT)
  {
        if((ClientSocket = ::accept(pOwner->m_Socket,(struct sockaddr*)&ClientAddress,&AddressSize)) == INVALID_SOCKET)
        {
::SendMessage((struct HWND__*)pOwner->m_hWnd,WM_INVALID_SOCKET,0,0);
        }
        else
        {
              UINT tid;
              ARGLIST* cli_thread_arg;
              cli_thread_arg = new ARGLIST;
              cli_thread_arg->cli_info = new SOCKNODE;
              cli_thread_arg->pOwner = pOwner;
              cli_thread_arg->cli_info->cli_soc = ClientSocket;
              cli_thread_arg->cli_info->t_id = _beginthreadex(NULL, 0,CMySocket::ClientMessageThread,cli_thread_arg,0,&tid);
        cli_thread_arg->cli_info->cli_ip = ClientAddress.sin_addr.s_addr;
        ::SendMessage((struct HWND__*)pOwner->m_hWnd,WM_ACCEPTED_SOCKET,0,0);
        }
  }
  else if((event.lNetworkEvents & FD_CLOSE) == FD_CLOSE)
  {
        closesocket(pOwner->m_Socket);
        EventLoopStopFlag = true;
  }
  else if((event.lNetworkEvents & FD_READ) == FD_READ)
  {
  }
}
return 0;
}
```

(그림 6) Winsock을 이용한 네트워크 통신

네트워크 연결이 되어 통신이 가능하게 되면 Packet 내부의 데이터 정보를 해석하여 실시간 오류검지 데이터 값을 구한다.

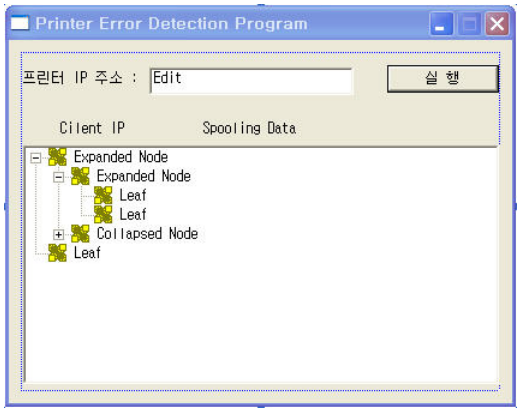
(그림 7)과 같이 SrcIP와 DestPort값을 이용하여 Client PC의 IP값과 프린터 Port로 이동하는 Packet의 값을 찾아낼 수 있으며, Type 값과 Data값 내부에서 출력 데이터의 정보를 분류하여 서비스를 구현한다.

```
int CEthernetPacket::Parse(LPBYTE pStr, int nSize)
{
        memset(&m_strDestMAC,     0x00, 128);
        memset(&m_strSrcMAC,      0x00, 128);
        memset(&szDestIP,         0x00, 128);
        memset(&szSrcIP,          0x00, 128);
        memset(&szDestPORT,       0x00, 128);
        memset(&szSrcPORT,        0x00, 128);
        memset(&szType,           0x00, 128);
        memset(&szInfo,           0x00, 128);
        memset(&szData,           0x00, 2048);

        m_nProtocol = 0;
        SetBuff(pStr, nSize);
        GoNext(m_nProtocol, pStr);

        return 0;
}
```

(그림 7) Packet 데이터의 분류

출력 데이터의 분석 값을 참조하여 (그림 8)과

같은 팝업창을 이용, 실시간으로 네트워크 프린터 환경에서 오류 발생에 대한 정보를 파악할 수 있도록 하였다.



(그림 8) 실시간 오류검지 서비스 실행화면

5. 성 능 평 가

오류검지 서비스의 성능을 평가하기 위해 전체적인 거래처의 특성을 고려하여 목동 지역에 있는 대형 거래처에 프로그램을 적용 시켰다.

거래처의 환경적 사항은 <표 2>와 같다.

<표 2> 거래처 현황 분석

	Printer 대수	PC 사용수	근무인원
P2P 사용	46대	316대	426명
Network 사용	15대		

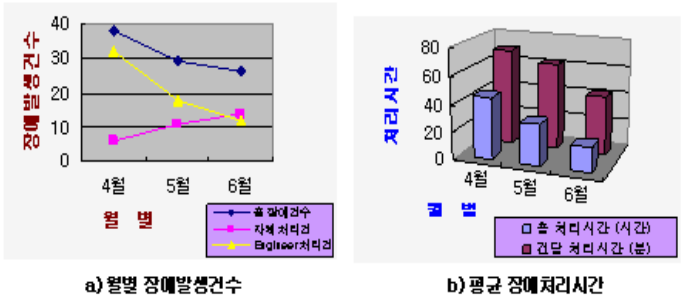
'05년 2/4분기 동안 네트워크 프린터 장애 발생 건수 및 처리 시간이 <표 3>에서와 같이 발생하였으며, 6월은 네트워크 프린터 오류검지 서비스 프로그램을 실행하여 발생된 수치이다.

<표 3> 2/4분기 월별 장애발생 건수 및 처리시간

	4월	5월	6월	증감율
총 장애건수	38	29	26	77.6%
자체 처리건	6	11	14	164.7%
Engineer 처리건	32	18	12	48.0%
총 처리시간 (시간)	45	30	18	49.1%
건당 처리시간 (분)	70.6	62.5	42.5	63.8%

실시간 오류검지 프로그램 적용결과 장애 요인의 정확한 분석으로 인해 (그림 9)의 a)와 같이 거래처의 네트워크, 프린터 관리자가 문제를 처리하는 건수가 증가 하였고, 총 장애건수 및 Engineer 방문 처리건수는 22.4% 및 52.0% 감소하였다.

장애 발생 시 문제를 처리하는 시간에 있어서도 정확한 문제점을 분석할 수 있어 4, 5월의 프로그램을 적용하기 전보다 6월이 (그림 9)의 b)에서와 같이 장애 발생 당 2/3 수준인 42.5분의 처리 시간을 보여 평균 24분의 문제 처리 시간이 단축, 네트워크 프린터 장애로 인한 업무 효율의 마비 시간을 줄일 수 있다.



(그림 9) 월별 장애 발생건수 및 처리시간 변화

6. 결론 및 향후과제

본 논문에서는 네트워크 프린터 환경에 대하여 효율적인 관리를 할 수 있도록 실시간 오류검지 서비스 모델을 제안 하였다. 이 서비스 모델로 인하여 장애 발생의 원인이 되는 Client PC를 신속히 발견, 처리하여 작업 환경의 업무 효율을 높일 수 있으며 오류 데이터 생성에 의한 네트워크 부하를 줄일 수 있다.

향후 연구로는 실시간 검색에 대한 Log File 분석 툴과 프로그램 자체의 데이터 제어에 관한 연구가 필요하다.

참 고 문 헌

[1] 정보기기사업부 기술팀, "Printer Emulation", 프린터의 이해, pp 53~64, 1994

[2] Matthew Gast & Todd Radermacher, "Introduction to the Server Message Block Protocol (SMB)", Networking Printing, pp 131~133, 2000

[3] 안철수 연구소, "Win32/Bugbear.worm.50688", http://home.ahnlab.com/smart2u/virus_detail_1034.html

[4] Hauri, "윈도우즈 환경에서의 공유폴더 취약점을 이용한 공격 유형 및 취약점 대응 방법", http://www.hauri.co.kr/virus/virusnsecurity/virusnsecurity_read.html

[5] 윤성진, "Socket Programming with MFC in Win32 Environment", http://50001.com/language/cppside/data/