주기성을 갖는 네트워크 관리 정보 수집을 위한 셔틀 프로토콜의 설계 및 구현

강 현 중† 이 상 일†† 정 진 욱†††

요 약

본 논문은 피관리 시스템으로부터 네트워크 관리 정보를 효율적으로 수집할 수 있는 새로운 폴링 방식인 셔틀 프로토콜을 제안, 설계, 구현하고 시뮬레이션을 수행한다. 셔틀 프로토콜의 주요 특성은, 첫째로 피관리 시스템들을 연결하는 링형태의 논리적인 연결이고, 두째는 수집해야 할 관리 정보가 논리적인 링형태의 연결로써 지정된 피관리 시스템들 사이에서 순회된다는 것이다. 피관리 시스템에 의해서 생성된 관리 정보는 이웃하는 피관리 시스템으로 연계되고 그 시스템은 자신의 데이타를 수신한 데이타에 추가하여 다음 피관리 시스템으로 전송한다. 결국, 관리자 스테이션은 모든 피관리 시스템에 의해 생성된 모든 관리 정보를 얻을 수 있다. 기존의 폴링 방식들을 사용하여 발생된 관리 트래픽의 발생 특성에 관하여 살펴본 후, 이들 기존의 폴링 메카니즘들의 단점을 개선하고 TCP/IP 네트워크 상에서 동작가능한 셔틀 프로토콜을 구현하였다. 또한, 시뮬레이션 패키지인 OPNET을 이용한 시뮬레이션을 통해 관리 시스템의 패킷 처리 시간과 분포, 게이트웨이에서의 패킷 처리 시간과 분포및 큐에서 대기중인 패킷수와 비트수를 기존의 폴링 방식과 제안된 셔틀 방식을 비교 분석하였다.

The Design and Implementation of the Shuttle Protocol for Gathering Management Information Periodically

Hyun-Joong Kang† Sang-ill Lee†† and Jin-Wook Chung†††

ABSTRACT

This paper proposes the shuttle protocol that can gather management information from managed systems in an efficient way. In this paper, we implement the protocol and evaluate the performance by simulation. The major feature of the shuttle protocol is a chained logical connection through managed systems, and management informations to be collected are circulated among specified managed systems in circular order on a logical ring connection. The data generated by an managed system are relayed to a neighbor managed system and the system sends its data which has additional management information to received data. Finally, a manager station can get all of data generated by every managed system. we will show the analysis of management traffic patterns using conventional polling schemes and the shuttle protocol implementation viable to TCP/IP network and improving existing polling mechanisms. Additionally, it is performed to evaluate the packet processing time and its distribution of a manager system and a gateway, and the queue length of packet and bit length of gateway against conventional polling schemes by simulation using OPNET, a simulation-dedicated package.

1. 서 론

최근, 네트워크는 점점 더 방대해지고, 이에 따라 네트워크의 구조 또한 복잡해지고 있다. 따라서 효율적이고 신뢰성있는 네트워크 환경을 제공하기 위하여 네트워크 관리가 필요하게 되었다. 그러나, 네트워크를 관리하기 위하여 생성된 이 부수적인 트래픽은, 종단점 응용 사용자들에

† 정 회 원 : 서일전문대학 전자계산과 조교수
†† 정 회 원 : 성균관대학교 정보공학과 석사과정
††† 종신회원 : 성균관대학교 정보공학과 교수
논문접수 : 1995년 8월 25일, 심사완료 : 1995년 11월 1일

게 신뢰성있고 효율적인 네트워크 환경을 제공하기 위하여 생성된다는 정당한 이유를 가지고 있음에도 불구하고, 사용자 응용의 입장에서 보면 오버헤드이다. 결과적으로, 이러한 트래픽의 과도한 발생은 네트워크 성능을 저하시키는 요인이 된다[1, 2]. 예를들어, 폴링 기반의 네트워크 관리 시스템에 의해 생성된 관리 트래픽은, 각 피관리 객체내의 에이전트들로 부터의 응답이 거의 동시에 관리 시스템으로 도달하기 때문에, 순간적으로 네트워크 관리자 스테이션으로 집중하게 된다. 따라서 이러한 방식을 이용하는 네트워크의 성능은 저하된다. 특히, 많은 피관리 시스템들을 갖고 있는 대규모 네트워크의 경우에는 순간적인 관리 트래픽의 폭발이 발생할 수 있으므로 더욱 그러하다. 이러한 오버헤드를 줄이기 위하여, 관리 트래픽을 생성하고 전송하기 위하여 최적화된 서비스를 지원할 수 있는 특별한 프로토콜을 필요로 하게 되었다. 즉, 효율적인 방식으로 네트워크 관리 트래픽을 처리하고, 신뢰성있는 서비스를 연속적으로 제공할 수 있도록 하는 메카니즘을 필요로 하게 되었다.

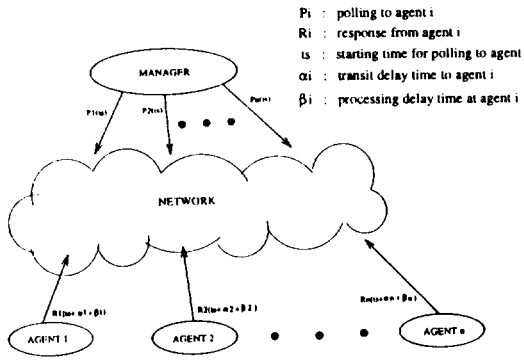
관리 트래픽을 생성하는 주요 방식은 폴링이다[3-6]. 예를 들어, 관리자 스테이션이 피관리 시스템들을 폴링하고, 이에 대해, 각 피관리 시스템은 자신의 응답을 즉각 송신하는 것이다. 효율적인 네트워크 관리를 위한 주요 연구 활동들은 주로 일반 사용자 응용 트래픽이 어느 정도냐에 따라 폴링에 의해 생성되는 트래픽의 양을 조절하는 것이었다[1 ,3]. 즉, 만약 일반 사용자 응용 트래픽이 적게 발생하면, 네트워크는 더 많은 관리 트래픽의 발생을 허용하고, 반대의 경우에는 관리 트래픽의 양을 줄이는 것이다[2]. 그러나, 이러한 접근은 네트워크상의 일반 사용자 트래픽이 많을 경우, 계속해서 네트워크 관리 정보의 발생을 억제할 수 있다는 취약점을 가지고 있다.

이러한 조건하에서, 관리자 스테이션과 다수의 피관리 시스템들 사이의 관리 정보 교환을 위한 효율적인 프로토콜을 개발하기 위하여 기존의 폴링 방식에 의한 관리 트래픽의 생성 및 전송 특성을 고찰한 후, 주기적인 관리 정보 수집에 적합한 셔틀 프로토콜을 설계하고 구현하였으며 시뮬레이션 전용 패키지인 OPNET을 이용한 시뮬레이션을 통해 그 성능을 입증하였다.

2. 폴링 방식에 따른 관리 트래픽 특성

2.1 동시 폴링 방식

이 폴링 방식에서는, 관리자 스테이션이 직접 피관리 시스템들을 폴링한다. 관리자 스테이션은 미리 정해진 폴링 간격에 따라 동시에 모든 피관리 시스템들을 폴링한다. 관리 정보 요구 명령어를 수신하는 모든 피관리 시스템들은 관리자 스테이션과의 일대일의 논리적인 연결상에서 즉시 응답을 한다. 따라서 관리자 스테이션은 피관리 시스템들로 부터 거의 동시에 모든 응답을 수신하게 된다. 많은 피관리 시스템을 가진 큰 네트워크인 경우, 네트워크 혼잡과 관리자 스테이션의 과부하 가능성이 매우 높아진다. 이 폴링 방식의 개념도가 (그림 1)에서 보여진다.



(그림 1) 동일한 주기의 폴링
(Fig. 1) Polling with a Same Time Interval

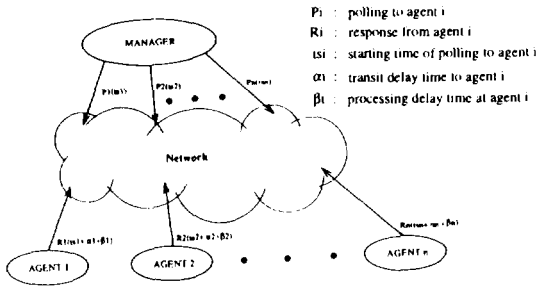
관리자가 한번의 폴링에 의해서 발생하는 관리 정보들이 각 피관리 시스템으로 부터 유입될 때까지의 전체 응답 시간 γm 은 다음과 같다.

$$\gamma m = P_i(t_s) + \alpha_i + \beta + R_i(t_s)$$

$$R_i(t_s) = \max\left(\sum_{k=1}^{n} (\gamma_1(k) + \gamma_{a1}(k)), \ldots, \sum_{k=1}^{n} (\gamma_i(k) + \gamma_{ai}(k)) \right)$$

2.2 개별 폴링 방식

이 폴링 방식은 폴링을 시작하는 시점이 서로 다르다는 점을 제외하고는 첫번째 폴링 방식과 유사하다. 즉, 관리자 스테이션이 모든 피관리 시스템들을 동시에 폴링하지 않고 각각의 폴링 시작 시점과 폴링 간격을 가지고 각 피관리 시스템을 폴링한다. 그래서 이 폴링 방식은 관리자 스테이션으로의 관리 트래픽의 유입이 각각의 폴링 시작 시점과 폴링 간격에 따라 분산되기 때문에 네트워크 혼잡과 중간 노드와 관리자 스테이션의 과부하를 어느정도 줄일 수 있다. 그러나 만약 피관리 시스템의 수가 점차 증가한다면, 이러한 좋은 효과도 줄어들 것이다. 관리 정보의 양은 피관리 시스템의 수에 따라 증가하기 때문에, 네트워크 혼잡이나 각 노드의 과부하 발생 확률은 여전히 존재한다. 그리고 만약 폴링 간격이 줄어든다면, 관리자 스테이션은 중첩된 응답을 수신하게 되고, 결과적으로 관리자 스테이션의 부하는 증가한다.



(그림 2) 개별적인 주기의 폴링
(Fig. 2) Polling with a Private Time Interval

개별 주기를 갖는 폴링 방식에서 관리자가 수신하는 관리 트래픽 전체 응답시간 $\gamma m'$은 다음과 같다.

$$\gamma m' = (P_1(t_s) - (P_1(t_s)) + (\sum_{i=1}^{j} \alpha_i - \sum_{i=1}^{j} (\eta i - 1)$$

$$\tau^{\eta_i}) + \beta_i + R'_1(t_s)$$

$$R'_1(t_s) = \max\{\sum_{k=1}^{n} (\gamma_1(k) + \gamma_{\alpha_1}(k)),$$

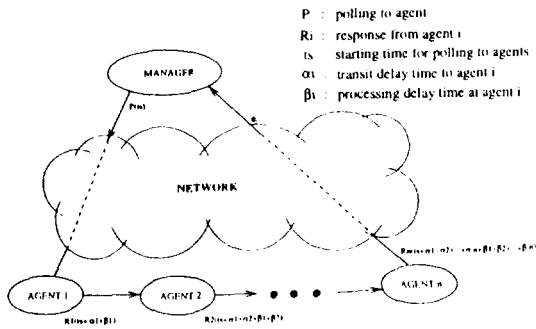
$$\max(\sum_{k=1}^{n} (\gamma_1(k) + \gamma\alpha_1(k)), \ldots, \sum_{k=1}^{n}$$

$$(\gamma_{i-1}(k) + \gamma\alpha_{i-1}(k)))\}$$

3. 제안하는 폴링 방식

본 논문에서 제안하는 폴링 방식에서는 관리자 스테이션이 피관리 시스템을 간접적으로 폴링한다. 제안하는 폴링 방식에서는 모든 피관리 시스템들을 링형태의 연결로 맺어 이 연결상에서 관리 정보 수집을 수행한다. 우리는 제안하는 폴링 방식을 셔틀 폴링 방식이라고 부르고 셔틀 폴링 방식이 사용하는 프로토콜을 셔틀 프로토콜이라고 한다. 셔틀 프로토콜에 의해 수행되는 첫번째 동작은 의도된 피관리 시스템들을 통한 셔틀 경로 설정이다. 이 절차가 끝난 후에 관리자 스테이션은 경로상의 첫번째 피관리 시스템에게 관리 정보 요구 패킷을 전송한다. 첫번째 피관리 시스템이 관리 정보 요구 패킷을 수신하였을 때, 그 시스템은 자신의 응답을 그 패킷의 끝에 추가한 후, 두번째 피관리 시스템으로 그 패킷을 전달한다. 두번째 피관리 시스템은 첫번째 피관리 시스템과 같은 작업을 수행한다. 마지막 피관리 시스템은 자신 이전의 모든 피관리 시스템으로 부터의 응답 정보와 관리 정보 요구를 수신하게 되고 자신의 응답을 그 패킷에 추가한 후, 하나의 데이타 단위로 관리자 스테이션으로 전송한다. 만약 관리자 스테이션이 이 절차를 사용한다면, 그 관리자 스테이션은 단지 하나의 관리 정보 요구 패킷을 사용함으로써 지정된 모든 피관리 시스템으로부터 응답 정보를 수집할 수 있다.

모든 피관리 시스템은 셔틀 경로상에서 선형적인 순서로 자신의 응답을 처리해야만 하기 때문에, 이 폴링 방식은 네트워크 요소들의 기능 이



(그림 3) 셔틀 프로토콜을 이용한 폴링 방식
(Fig. 3) Polling Scheme using Shuttle Protocol

상을 실시간적으로 검출하지는 못하지만 TCP/IP 기반의 네트워크는 사건 보고 방식의 트랩(trap)[7-9]을 사용하여 실시간적으로 네트워크 요소들의 기능 이상을 검출할 수 있다.

이 폴링 방식은 계층적인 네트워크 관리에도 적합하다. 각 하부 네트워크내의 하부 도메인 네트워크 관리자는 동일 도메인 내에 있는 피관리 시스템으로부터 관리 정보를 수집한다. 중앙 제어 센터는 두번째 레벨 관리자를 연결하는 영구적인 혹은 주기적인 세션을 열고, 그 세션상의 두번째 레벨 관리자 스테이션에 저장된 관리 정보를 수집함으로써 하부 도메인 관리자들을 관리할 수 있다. 또한, 이 폴링 방식은 주기적으로 관리 정보를 수집하는 보다 쉬운 방식을 지원하고, 트래픽 안정화에 기여한다. 많은 피관리 시스템들에 의해 생성된 모든 네트워크 관리 정보는 경로상의 마지막 피관리 시스템에 모이고, 관리자 스테이션은 마지막 피관리 시스템으로의 단 하나의 연결을 사용하여 모든 관리 정보를 수신하기 때문에, 이 셔틀 프로토콜을 사용하는 폴링 방식은 네트워크 혼잡 발생 가능성을 줄인다. 경로 상의 링형태로 연결된 피관리 시스템의 수와 응답 시간은 반비례 관계에 있지만, 트래픽 안정화의 정도는 이에 비례한다.

셔틀 기법을 사용하여 관리 트래픽을 발생하는 경우 한번의 요구로 수신되는 전체 관리 정보의 수신 시간 $\gamma''m$ 은 다음과 같다.

$$\gamma''m = P_1(t_s) + \sum_{i=0}^{j} \alpha_i + \sum_{i=1}^{j} \beta_i + R''_1(t_s) + j\beta_i$$

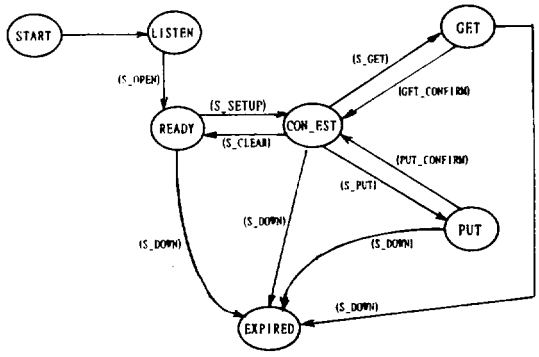
$$R''_1(t_s) = \alpha_1 + \sum_{i=1}^{j} \sum_{k=1}^{n} (\gamma_i(k) + \gamma_{ai}(k))$$

4. 셔틀 프로토콜의 설계 및 구현

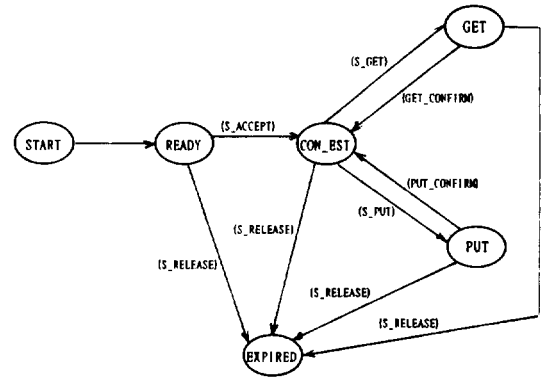
4.1 프로토콜 동작

프로토콜의 주요 특성은 네트워크 관리자 스테이션이 지정된 피관리 시스템들을 링형태의 연속적인 전송 연결을 설정하고, 연결이 설정된 후, 관리 정보 요구 패킷을 전송하고 이 연결을 통하여 관리 정보를 수신한다는 것이다. 셔틀 프로토콜은 연결 설정, 연결을 통한 데이타의 전송, 연결 해제, 그리고 에러 보고를 위해 몇가지 동작들을 사용한다.

클라이언트는 초기화 과정을 거쳐 응용 프로그램과의 연결을 위해 LISTEN 상태에 있게된다. 상위 계층으로 부터 연결 요청 사건이 발생하면 READY 상태에 머무른다. 이때, 상위 계층에서 RING SETUP 프리미티브를 보내면 연결을 위한 CON EST 상태로 천이한다. 연결 상태에서 SHUTTLE GET 요청이 들어오면 GET상태로 이동하고 SHUTTLE PUT 요청이 있으면 PUT 상태로 이동한다. 연결 종료를 위한 요청이 수신되면 EXPIRED 상태로 천이하고 프로세스를 종료시킨다. 이외의 경우는 에러 상황으로 가정된다. 이들의 관계를 (그림 4)에 나타내었으며 각 상태의 역할은 (표 1)에 설명하였다.



(그림 4) 셔틀 클라이언트의 동작
(Fig 4) Operation of Shuttle Protocol Client



(그림 5) 셔틀 서버의 동작
(Fig. 5) Operation of Shuttle Protocol Server

서버의 동작은 피관리 시스템의 동작에 해당된다. READY 상태에서 관리 시스템의 링 접속 요구를 대기하다가 접속 설정 요구가 유입되면 CON EST상태로 천이한다. 만일 SHUTTLE GET 패킷인 경우는 GET 상태로 이동하고 SHUTTLE PUT 패킷인 경우는 PUT 상태로 천이한다. 접속 해제에 대한 요청이 유입되면 연결을 해제하고 프로세스를 종료시킨다.

〈표 1〉 클라이언트 상태 기술
〈Table 1〉 Decription of Client States

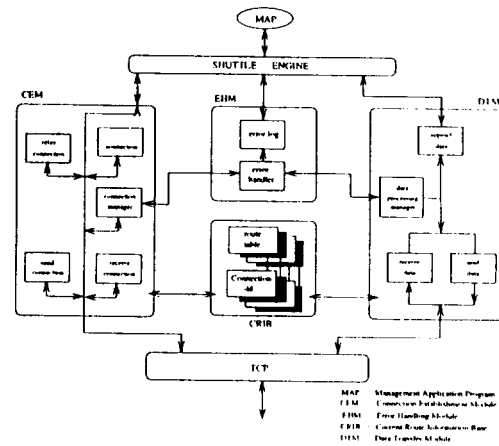
상 태	기 능
START	클라이언트 프로세스가 개시될 때 초기화 작업을 수행하는 상태
LISTEN	프로세스가 개시된 후 응용 프로그램으로부터 파이프 개방 요구를 기다리는 상태. 개방 요구가 들어오면 응용 프로그램과 클라이언트 프로세스간에 IPC를 위한 파이프가 생긴다.
READY	응용 프로그램으로부터 파이프를 통해 RING-SETUP 요구 프리미티브를 기다린다. 요구 프리미티브가 들어오면 접속 설정을 개시하고 RING-SETUP 패킷을 만들어 전송한 후, 링이 완전히 구성되어 요구 패킷이 되돌아오기를 기다리는 상태
CON EST	접속이 설정된 후 다음의 명령을 요구하는 프리미티브를 응용 프로그램으로부터 수신하기를 기다리는 상태
GET	SHUTTLE-GET 패킷을 만들어 링 접속을 통해 보낸 후 응답이 순환하여 되돌아 오기를 기다림
PUT	SHUTTLE-PUT 패킷을 만들어 링 접속을 통해 보낸 후 응답이 순환하여 되돌아 오기를 기다림
EXPIRED	더 이상 클라이언트 프로세스가 존재할 필요가 없는 상태로서 프로세스를 종료시킴

〈표 2〉 서버 상태 기술
〈Table 2〉 Description of Server States

상 태	기 능
START	서버 프로세스가 개시될 때 초기화 작업을 수행하는 상태
READY	서버 프로그램이 수신 포트를 열어 두고 클라이언트로부터 링 접속 요구가 들어오기를 기다리는 상태
CON EST	접속이 설정된 후 클라이언트로부터 어떤 요구 패킷이 도착하기를 기다리는 상태
GET	SHUTTLE GET 패킷이 클라이언트로부터 도착하여 관리 서버 프로그램으로 Indication을 보내고 결과가 반환되기를 기다리는 상태
PUT	SHUTTLE PUT 패킷이 클라이언트로부터 도착하여 관리 서버 프로그램으로 Indication을 보내고 응답이 반환되기를 기다리는 상태
EXPIRED	클라이언트로부터 링 접속 해제 명령이 들어와서 더 이상 서버 프로그램이 필요하지 않은 상태로서 프로세스를 종료 시킴

4.2 구현 모델

셔틀 프로토콜에 대한 구현을 위해서는 (그림 6)과 같은 구현 모델이 설정되어야 한다. 이 모델에서는 구현에 관련된 셔틀의 전반적인 구조를 담고 있다. 셔틀은 그 기능상 네가지의 모듈로 구성되어야 한다. 하나는 CEM(Connection Establishment Module)로서 셔틀 연결 설정을 관리하고 제어하는 모듈이다. 이 모듈은 연결 설정, 연결 해제, 연결 중계에 대한 책임을 지는 부분으로 연결 관리자가 전체를 통제하게 된다. EHM(Error Handling Module)은 연결에 관련된 문제의 발생이나 관리 정보의 전송중에 발생할 수 있는 에러에 대한 전반적인 처리를 담당하는 부분이다. 에러가 발생하게 되면 로그를 생성하고 해제에 대한 처리를 수행하도록 되어있다. DTM(Data Transfer Module)은 셔틀 경로 상에서 관리 정보의 송수신에 관련된 처리를 수행하는 부분으로 데이타 송수신과 데이타 추가 기능을 갖는다. 데이타 추가 기능은 중계 노드에 국한되어 처리된다. 이는 셔틀 정보의 구성에 대한 기본 기능이기도 하다. 마지막으로 CRIB (Current Route Information Base)는 연결 설

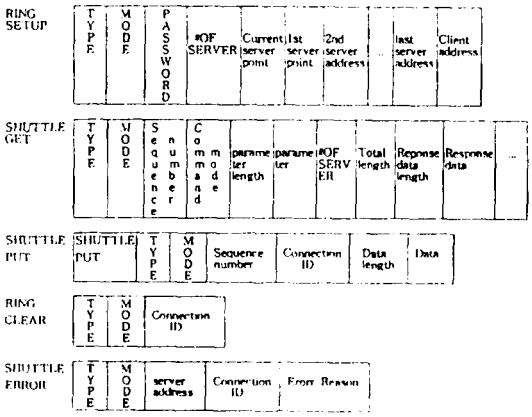


(그림 6) 셔틀 구현 모델
(Fig. 6) Shuttle Protocol Implementation Model

정에 관한 정보를 유지하고 셔틀 경로의 식별자를 관리하는 부분이다. 이 정보들을 토대로 하여 정확한 셔틀 경로를 통해 관리 정보가 흐르도록 한다.

4.3 설계 및 구현

셔틀 프로토콜은 다섯 가지의 패킷을 사용하며, 사용되는 패킷의 형태는 (그림 7)과 같다. 다섯 가지 패킷 모두 공통적으로 type 필드와 mode 필드를 갖는다. type 필드는 패킷을 구별해주는 역할을 하며 mode 필드는 각 패킷의 세부적인 선택 사항을 나타낸다. 이 필드를 이용하여 패킷의 용도를 세부적으로 명시한다.



(그림 7) 셔틀 패킷의 형태
(Fig. 7) Formats of Shuttle Packets

4.3.1 RING SETUP 패킷

접속을 설정하기 위해서 사용되는 패킷이다. password 필드는 허가 받지 않은 클라이언 트가 접속을 시도할 수 없도록 하기 위해서 요구자 신원 확인 용도로 사용된다. 패킷은 접속 설정 동작을 쉽게 수행할 수 있도록 요구된 피관리 시스템의 총 수를 나타내는 # of server 필드와 현재 접속을 처리한 시스템의 수를 나타내는 current server point 필드를 갖는다. current server point는 0으로 초기화되어 전송되고 각 피관리 시스템의 서버 프로세스는 RING SETUP 패킷을 수신하면 이 값을 1 증가시킨다. 증가된 값과 # of server의 값이 같으면 자신이 경로 상의 마지막 피관리 시스템인 것을 알게 되고 서버의 수신 포트 번호를 이용하여 TCP 접속을 요구한다. 또한, RING SETUP 패킷에 는 실제 접속될 피관리 시스템들의 주소가 각 발신지에서 채워진 후 전송되며, 각 서버 프로세스는 current server point를 이용하여 다음 피관리 시스템의 주소를 찾아서 사용한다.

4.3.2 SHUTTLE GET 패킷

관리 시스템이 피관리 시스템들에게 관리 정보를 반환하라는 명령을 보내고 응답을 회수 하기 위해서 사용되는 패킷의 형태이다. 각 명령을 식별하기 위해서 sequence number 필드가 사용되고 명령의 세부적인 용도를 나타내는 command mode 필드가 사용된다. connection Id는 TCP 접속을 기반으로 현재 연결된 링의 식별자가 된다. 또한, 명령과 함께 부가적인 정보를 보내기 위해서 추가되는 파라미터의 값과 파라미터의 길이를 나타 내는 parameter length와 parameter 필드가 사용된다. 현재 응답을 채운 피관리 시스템 의 수를 나타내는 # of server 필드가 있으며 전체 응답의 길이를 나타내는 total length 필드가 사용된다. 응답 필드는 다시 세부적으로 각 서버의 데이타 길이와 실제 응답 데이타들로 나뉘어 진다.

4.3.3 SHUTTLE PUT 패킷

관리 시스템에서 피관리 시스템으로 데이타를 전송할 때 사용된다. 데이타가 정확히 전 송되었는지의 여부는 SHUTTLE PUT 패킷이 링을 한 바퀴 돌아서 관리 시스템으로 돌 아오면 모든 피관리 시스템이 데이타를 복사한 것으로 가정할 수 있다. 데이타의 처리를 위해서 data length 필드가 사용되고 실제 데이타가 상주하는 data 필드가 존재한다.

4.3.4 RING CLEAR 패킷

설정된 접속을 해제하기 위해서 사용되며 접속 식별을 위한 connection-ID 필드가 포함 된다. 접속 해제는 링상의 첫번째 피관리 시스템부터 시작되며 최종적으로 관리 시스템 이 수신 포트를 통해서 접속 해제를 실행한다.

4.3.5 SHUTTLE ERROR 패킷

오류를 일으킨 피관리 시스템이 그러한 사실을 관리 시스템으로 알릴때 사용하는 패킷이다. agent address 필드는 오류가 발생한 피관리 시스템의 주소를 나타내며, connection ID 필드는 접속을 식별하기 위한 필드이다. error reason 필드는 오류의 원인을 명시하는 필드이다.

이와 같은 셔틀 구성 패킷들은 상위 계층과의 인터페이스를 위해서 프리미티브를 지원하게 된다. 프리미티브는 패킷의 형태와 맞추어서 연결 설정 요구, 데이타 교환, 연결 해제로 나누어 진다. 이때, 오류에 대한 것은 셔틀 자체의 구성 패킷이므로 프리미티브로 존재할 필요가 없다. 각 프리미티브와 관련 파라미터를 나타내면 (표 3)과 같다.

셔틀 클라이언트와 셔틀 서버는 네트워크 상에서 가상적인 링의 구조를 유지하기 위해서 동작한다. 클라이언트와 서버간의 통신은 TLI를 이용하여 구현되며 상위 응용 프로그램과는 파이프를 이용한다[10, 11]. 이러한 클라이언트와 서버는 서로 동작 과정이 상이하게 되는데 이들중 클라이언트의 동작을 나타내는 가상 코드는 (그림 8)과 같다. 클라이언트는 관리 시스템에 상주하는 프로세스로서 관리에 대한 명령을 발하고 링을 유지하는 역할을 하게된다.

이에 반해 서버는 피관리 시스템의 동작을 나타내는 것으로 관리 시스템에서 요청된 셔틀 링을 구성하고 이를 중계하는 역할을 한다. 아울러, 관리 정보의 요청에 대해서 자신의 정보를 추가하여 이웃하는 피관리 시스템으로 전송한다. 이러한 과정을 수행하는 서버의 가상 코드는 (그림 9)와 같다.

〈표 3〉 프리미티브 형식
〈Table 3〉 Primitives and its parameters

Primitives	Parameters
RING_SETUP.request	Connection Type, Called Addresses
RING_SETUP.indication	Connection ID
RING_SETUP response	Connection ID
RING_SETUP.confirm	Connection ID
SHUTTLE_GET.request	Connection ID, Command Mode, Command Parameter
SHUTTLE_GET.indication	Connection ID, Command Mode, Command Parameter
SHUTTLE_GET. response	Connection ID, Response Data
SHUTTLE_GET.confirm	Connection ID, Response Data
SHUTTLE_PUT.request	Connection ID, Put Data
SHUTTLE_PUT.indication	Connection ID, Put Data
SHUTTLE_PUT. response	Connection ID
SHUTTLE_PUT.confirm	Connection ID
RING_CLEAR.request	Connection ID
RING_CLEAR.indication	Connection ID
RING_CLEAR.response	Connection ID
RING_CLEAR.confirm	Connection ID

5. 시뮬레이션

본 논문에서 제안한 셔틀 프로토콜의 우수성을

```
procedure ShuttleClientMain
  begin
    CreatePipetoApp;
    while true do
      begin
        WaitaOpenRequestfromApplication;
        ForkChildProcess;
        while true do
          begin
            WaitaNextCommandfromApplication;
            if ReceiveConnectionSetupPrimitive
              then begin
                SendRingSetupRequesttoServer;
                WaitaConnectionRequestConfirmfromTcp;
              then end
            while true do
              begin
            WaitaNextCommandfromApp;
            if ReceiveDataTransferCommandfromApp
              begin
                then WaitaConfirm;
                else ExitWhileLoop;
              end
            if ConnectionReleaseCommandReceivedfromApp
              begin
                ConnectionReleaseProcedure;
                ExitWhileLoop;
              end
          end of while
      end of while
    end of while
end of procedure
```

(그림 8) 클라이언트 구동 가상 코드
(Fig. 8) Pseudo Code for Shuttle Client Operation

```
procedure ShuttleServerMain
  begin
    OpenReceivePort;
    while true do
      begin
        ListenaConnectionRequestfromTcp;
        OpenRespondingPort;
        AcceptConnectionRequestwithRespondingPort;
        ForkChildProcess;
        ReceiveConnectionRequestandRelay;
        while true do
          begin
            WaitaNextCommandfromTcp;
            if ReceiveDataPacketfromApp
              begin
                IssueaCorrespondingPrimitive;
                if ReceiveaResponse
                  then RelayCorrespondingPacket;
                  else ErrorReporttoClient;
              end
            if ConnectionRreleaseCommandReceivedfromApp
              begin
                doing connection release procedure;
                exit this while loop;
              end
          end of while
      end of while
  end of procedure
```

(그림 9) 서버 구동 가상 코드
(Fig. 9) Pseudo Code for Shuttle Server Operation

살펴보기 위해서 네트워크 전용 시뮬레이션 도구인 OPNET[12]을 이용하여 시뮬레이션을 수행하였다. 시뮬레이션의 결과 값들은 OPNET에서 제공하는 IP 계층 노드 모델 큐의 값들이다. 셔틀 프로토콜을 사용하는 경우 라우터와 관리 호스트로 유입되는 관리 정보의 감소로 인하여 라우터및 관리 시스템에서의 패킷 처리 지연 시간을 비롯하여, 라우터및 관리 시스템에서의 패킷 처리 시간분포, 라우터에서의 대기중인 패킷 수와 비트수 등에서 개선된 결과를 얻을 수 있었다.

시뮬레이션을 위해서 가정한 사항은 다음과 같다.

[가정 1] 네트워크 연동 통신 시스템의 송신 처리율 cs와 수신 처리율 cr은 동일하며 cs=cr=1,000 pkt/s로 한다.

[가정 2] 각 피관리 시스템과 관리 시스템의 송신 처리율 ms와 수신 처리율 mr은 동일하며 ms=mr=200pkt/s로 한다.

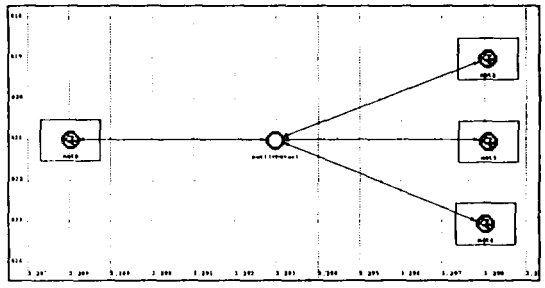
[가정 3] 관리 시스템에서의 폴링 주기 tp=1.0sec로 한다.

[가정 4] 네트워크 모델은 4개의 Ethernet [13]으로 구성되어 있고 각 Ethernet에는 5개의 피 관리 시스템이 존재한다. 네트워크 모델과 서브네트워크 모델이 그림 10과 그림 11에서 보여진다.

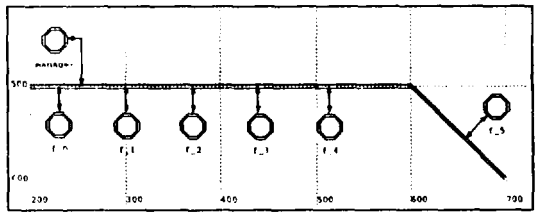
[가정 5] 일반 트래픽의 발생은 평균 2pkt/s로 하며 지수 분포를 따른다.

[가정 6] 관리 응답 메세지의 길이 mres는 256bytes로 한다.

[가정 7] 관리 요구 메시지의 길이 mreq는 32bytes로 한다.



(그림 10) 네트워크 모델
(Fig. 10) Network Model



(그림 11) 서브네트워크 모델
(Fig. 11) Subnetwork Model

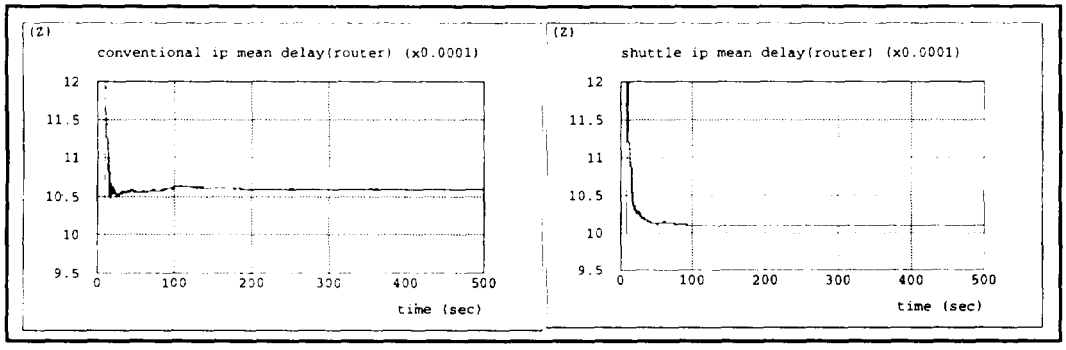
(그림 12)는 라우터에서 발생하는 IP 데이터그램 처리 지연 시간을 나타내고 있다. 기존의 폴링 기법을 사용하는 경우에는 약 1.06㎲가 걸리는 것으로 나타나고, 셔틀 기법을 사용하는 경우에는 약 1.01㎲가 걸리는 것으로 나타나고 있다. 이 그림은 이전에 측정된 모든 값에 대한 평균값을 나타내기 때문에 값이 큰 차이를 보이지 않으나, 모든 값을 출력한 (그림 13)에서는 관리 트래픽이 몰려드는 순간의 최대 값이 비교적

큰 차이를 보여주고 있으며, 트래픽의 밀도 역시 큰 차이를 나타냄을 알 수 있다.

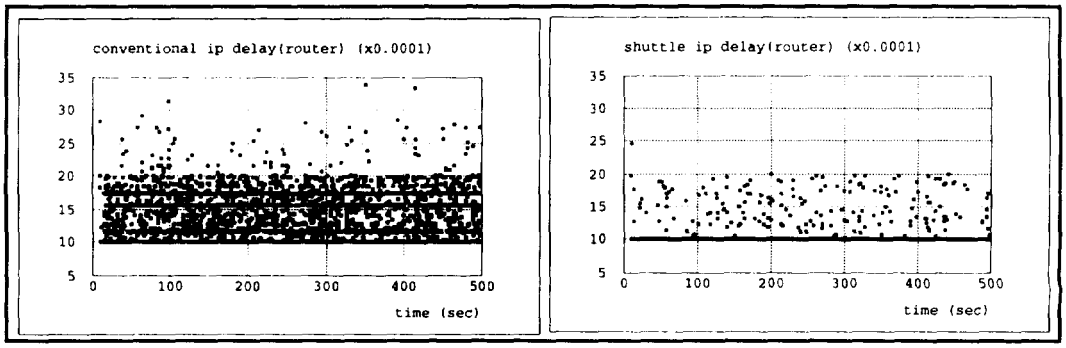
(그림 14)와 (그림 15)는 관리 시스템의 IP 데이터그램 처리 지연 시간을 나타내고 있다. 일반적인 폴링 기법을 사용하는 경우에는 1.8ms의 지연시간이 발생하고, 셔틀 프로토콜을 사용하는 경우에는 0.8ms의 지연 시간이 발생하고 있다. 이 경우에는 라우터보다 큰 차이를 볼 수 있는데, 이것은 관리 정보를 요구하는 패킷이 셔틀 프로토콜의 경우, 피관리 시스템 그룹별로 한개만 생성되어 전송되므로 훨씬 적은 부하가 걸리기 때문이다. 반면에 일반적인 폴링 기법의 경우에는 5개가 생성되어 한꺼번에 전송된다. 즉, 관리 정보 요구를 구성하는 IP 데이터그램의 생성수도 호스트의 통신 처리 장비에 미치는 영향이 큼을 알 수 있다.

(그림 16)은 라우터에 새로운 IP 데이터그램이 도착하는 순간에 큐에 서비스를 받기 위해서 대기하고 있는 패킷의 평균 갯수와 총 비트수의 평균값을 나타내고 있다. 그림에서와 같이, 일반적인 폴링 기법을 사용하는 경우에는 대기하는 패킷의 갯수는 많지만 처리될 비트수는 오히려 셔틀 기법을 사용하는 경우보다 적음을 볼 수 있다. 이는 셔틀 프로토콜을 사용하는 경우에는 여러 피관리 시스템의 응답이 하나의 단위 패킷으로 구성되기 때문에 하부 네트워크에 나타나는 프레임의 수가 현저히 줄어드는 것을 입증하는 것이다.

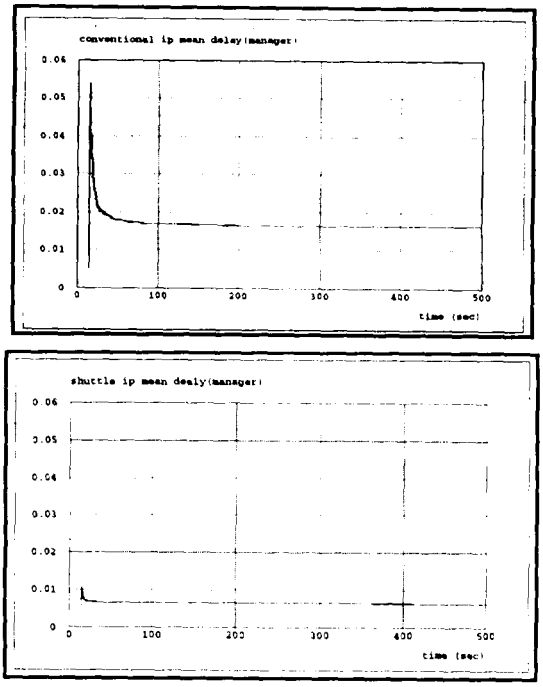
관리 데이타는 특성상 단위가 그다지 크지 않기 때문에 프레임으로 구성될 때, 하부 네트워크의 MTU(Maximum Transfer Unit)를 채우지 못하는 경우가 대부분이다. 따라서 매 관리 응답 메세지마다 하나의 독립된 단위로 처리되므로, 피관리 시스템의 수가 급격히 증가하면 라우터와 같은 중계 노드의 부담이 매우 커지게 된다. 반면에 셔틀 프로토콜을 이용하는 경우에는 여러개



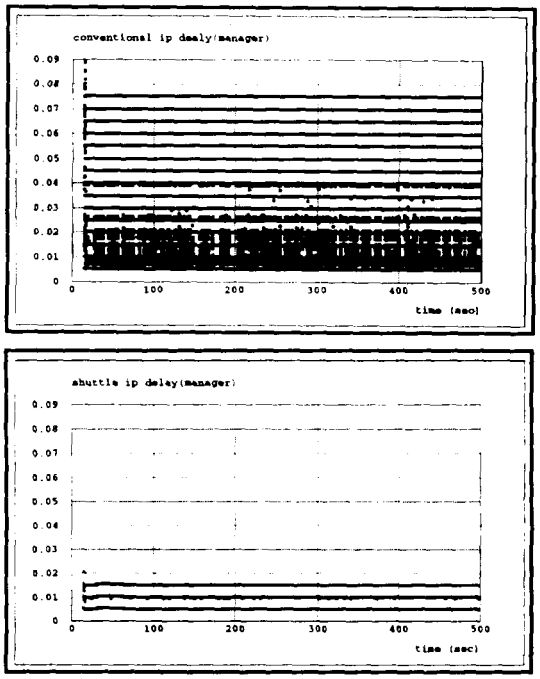
(그림 12) 라우터에서의 패킷 평균 처리 시간
(Fig. 12) Packet Mean Processing Time in a Router



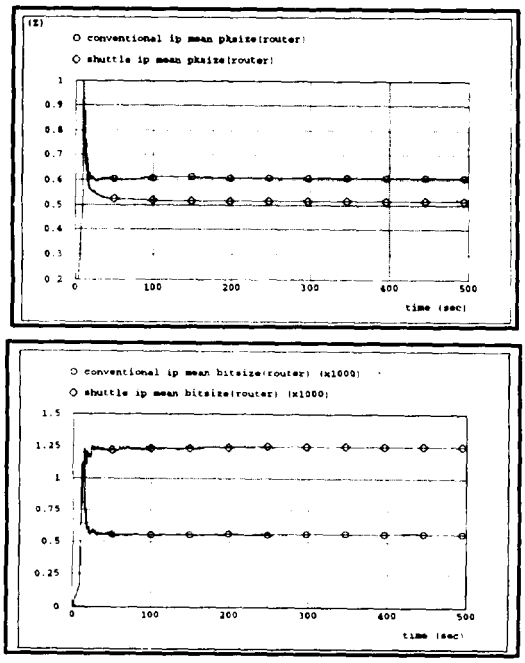
(그림 13) 라우터에서의 패킷 처리 시간
(Fig 13) Packet Processing Time in a Router



(그림 14) 관리 시스템에서의 패킷 평균 처리 시간
(Fig 14) Packet Mean Processing Time in a Manager System



(그림 15) 관리 시스템에서의 패킷 처리 시간 분포
(Fig 15) Distribution of Processing Time in a Manager System



(그림 16) 라우터에 대기중인 패킷의 갯수와 비트수
(Fig 16) Queued Packet Counts and Length of Bits in a Router

의 응답이 하나의 프레임으로 구성되는 경우가 많이 발생하므로 독립적으로 처리되어야 할 총 갯수가 일반적인 폴링 기법을 사용하는 경우보다 현저히 줄어들고 결국 중간 노드의 부담이 줄어들게 된다.

6. 결 론

본 논문에서는 TCP/IP 기반의 네트워크에서 주기성을 갖는 네트워크 관리 정보의 효율성을 제공하는 셔틀 프로토콜을 구현하고 시뮬레이션을 통해 그 성능을 입증하였다. 이러한 프로토콜을 개발하는데 있어서 가장 중요하게 고려한 것은 네트워크 관리를 위해서 발생하는 트래픽의 특성이다. 특히, 성능관리와 같은 네트워크 관리는 하나의 관리자가 여러 개의 피관리 시스템들에게 동일한 명령을 반복적으로 폴링 방식으로 보내고 그에 대한 응답을 회수한다. 그러한 응답의 전송은 폴링 방식의 특성에 따라 한꺼번에 생성되고 또한 각기 개별적인 논리적 접속이나 전송 계층 접속을 통해서 관리 시스템으로 보내진

다. 따라서 피관리 시스템의 수가 증가하면, 네트워크의 중간 노드로 동시에 모든 응답이 유입되므로 네트워크 폭주 가능성이 높아지게 된다. 이러한 특성을 고려하여 여러 개의 피관리 시스템에서 생성된 응답을 하나의 전송 계층 접속으로 묶어서 전송할 수 있는 셔틀 프로토콜을 사용함으로써 네트워크 관리 트래픽의 안정화를 얻을 수 있다. 이러한 방식의 데이터 수집을 사용하면, 실시간 처리 측면에서는 기존의 폴링 방식과 비교하여 그 성능이 저하된다. 그러나, 긴급성을 요구하는 데이터의 전송은 비동기적 사건 보고 메세지를 사용하여 피관리 시스템에 의해서 자의적으로 수행되기 때문에 셔틀을 이용한 순방향 정보 수집과 사건 보고에 의한 비동기적 사건 처리를 결합하면 완전한 네트워크 관리 시스템이 구축되리라고 본다. 그리고, 시뮬레이션을 통해 관리 시스템의 패킷 처리 시간과 분포, 게이트웨이에서의 패킷 처리 시간과 분포및 큐에서 대기중인 패킷 수와 비트 수를 기존의 폴링 방식과 제안된 셔틀 방식을 비교 분석하여 그 우수성을 보였다.

참 고 문 헌

[1] 변옥환, "OSI 망관리 트래픽의 효율적 제어를 위한 PCC와 UFOA 기법 연구", 경희 대학교 공학 박사 논문, 1993.

[2] 김종우, 강현중, 정진욱, "효율적인 네트워크 관리 시스템을 위한 프로토콜의 설계및 구현", 한국 정보 처리 응용 학회 추계 학술 발표 논문집, pp.476-479, 1994.

[3] Richard E. Caruso, "Network Management : A Tutorial Overview", IEEE Comm. Magazine, pp.20-25, Mar. 1990.

[4] 변옥환, 안성진, "연구전산망의 망관리 현황및 계획", OSIA, 제16권 제3호(통권15호), pp.40-56,1992. 9.

[5] Heinz-Gerd Hegering, "Integrated Network Management,III, Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management", Elsevier Science Publishers, April, 1993

[6] Tony Jefferee, "A Review of OSI Management Standards", Computer Networks and ISDN Systems, pp.167-174, 1988.

[7] Douglas E.Comer, "Internetworking with TCP/IP(vol.1)", Prentice Hall, pp.171-203, 1991.

[8] McCloghrie,K.,and Rose,M.T. , "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II," RFC1213, Mar. 1991

[9] Case, J.D., Fedor, M.S. Schoffstall, M.L., and Davin, C. "Simple Network Management(SNMP),"RFC 1157, pp. 36 May 1990.

[10] Michael Padovano, "Networking Applications on UNIX system V Release 4", Prentice Hall, PP.315-412, 1993.

[11] Stephen A.rago, "UNIX System V Networking Programming", Addison-Wesley, pp.149-215, 1993.

[12] Al"ain C. Jerome, Steven P. Baraniuk," Communication Network and Protocol Design Automation, Military Comm. Conference, pp.334-339, Oct. 1987

[13] Advanced Micro Devices Inc., "Ethernet /IEEE 802.3 Family (Data Book)", 1992.

강 현 중

1980년 성균관대학교 수학교육과 졸업(학사)
1986년 연세대학교 전자계산과 졸업(석사)
1994년 성균관대 정보공학과 박사과정 수료
1979년~82년 한국과학기술연구소 근무
1982년~89년 삼희투자금융(주)전산실장 근무
1989년~현재 서일전문대학 전자계산과 조교수
관심분야 : 데이터 통신, 프로그래밍언어

이 상 일

1994년 성균관대학교 정보공학과 졸업(학사)
1994년 현재 성균관대학교 정보공학과 석사과정
관심 분야 : 네트워크, 네트워크 관리

정 진 욱

1974년 성균관대학교 전기공학과 졸업(학사)
1979년 성균관대학교 전자공학과 졸업(석사)
1991년 서울대학교 계산통계학과 졸업(박사)
1973년~85년 학국과학기술연구소 데이터 통신 실장 근무
1981년~82년 RACAL-MILGO Co 객원 연구
1992년~93년 미국 Maryland 대학 객원 교수
1985년~현재 성균관대학교 정보공학과 교수
관심분야 : 네트워크 보안, 네트워크 관리, 고속및 무선 통신 프로토콜