1. 네트워크 프로그래밍 개요

- 통신 프로토콜
- 클라이언트-서버 모델
- 서버 구축 기법
- Unix 프로그래밍 환경

1.1 컴퓨터 통신 프로토콜

컴퓨터 통신 프로토콜

- **컴퓨터 통신 프로토콜**
  - **컴퓨터들이 데이터를 원활이 주고받을 수 있도록 정한 약속**
- **네트워크 프로그램의 목적**
  - **통신 프로토콜을 이용하여 원하는 네트워크 서비스를 제공**
- OSI(Open System Interconnection) 7 **계층**
- TCP/IP(Transmission Control Protocol/Internet Protocol)

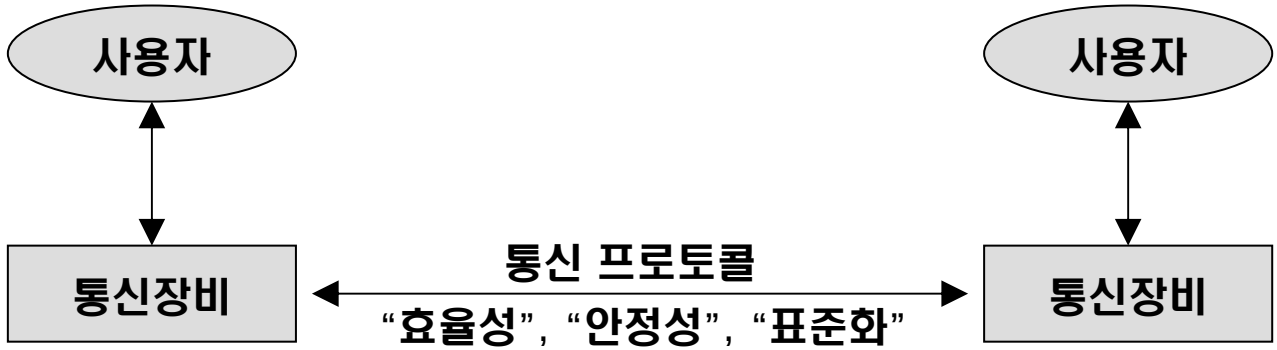
1.1.1 컴퓨터 통신 프로토콜의 정의

- **컴퓨터 네트워크 프로토콜**
  - **통신 장비는 서로간의 통신 방법이 미리 정의 되어 있어야 함**
  - **같은 통신 프로토콜을 지원하는 장비 간에만 통신이 가능**

- **컴퓨터 통신은 네트워크 형태로 운영**
  - **컴퓨터 네트워크 프로토콜이라 부름**

통신 프로토콜의 특징

- 통신 프로토콜
  - “효율적”, “안정적”으로 통신할 수 있도록 “미리 정한” 약속



- 효율적
  - 주어진 통신 채널(매체, 전송속도 등)을 최대로 이용할 수 있어야 한다
  - 프로토콜의 처리순서, 패킷의 최적 크기, 흐름제어를 사용
- 안정적
  - 비정상적인 장애(고의로 잘못된 데이터 송신, 통신망 장애 발생, 전송 중 오류 발생) 발생시에도 안정되게 동작해야 한다
- 표준화
  - 널리 사용되기 위해서는 미리 표준화되어야 한다

통신 프로토콜의 종류

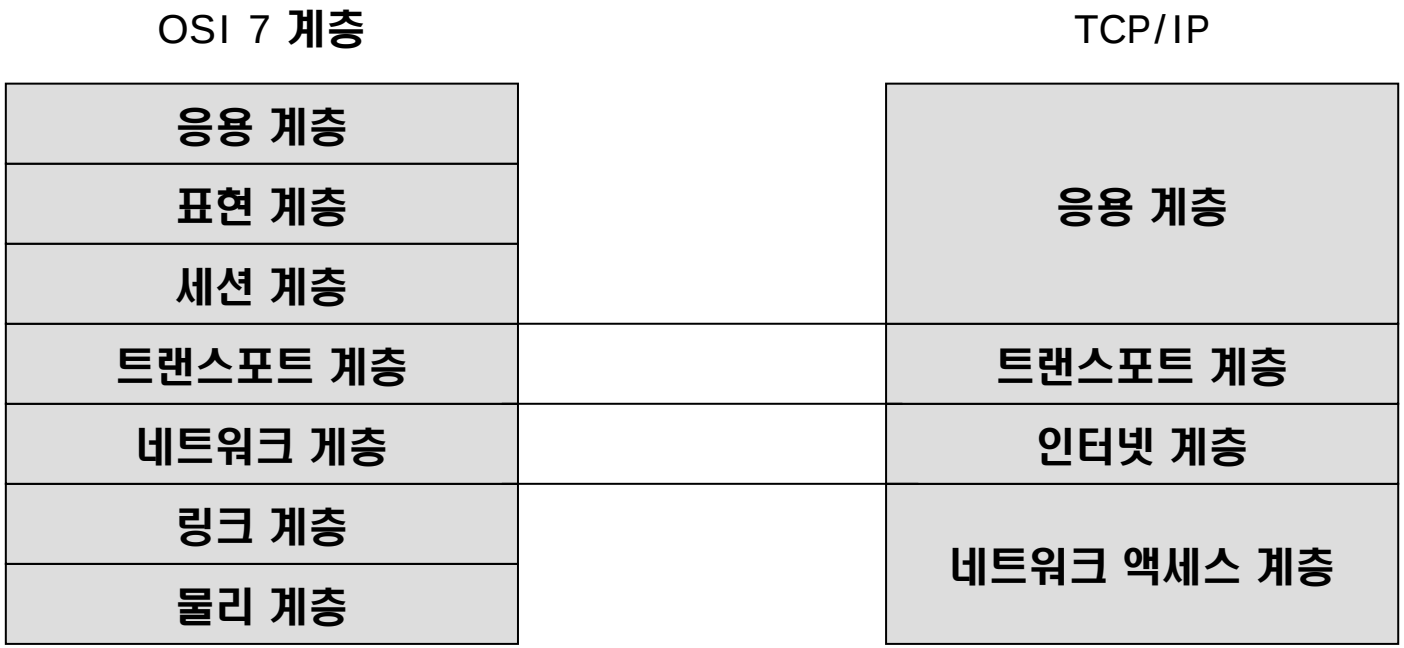
- 통신 프로토콜은 누가 정했는지에 따라 두 가지로 분류
  - 표준 통신 프로토콜
  - 사용자 정의 통신 프로토콜

- 표준 통신 프로토콜
  - 국제적으로 통용되는 표준
  - TCP/IP, http, telnet, ftp, ...

- 사용자 정의 통신 프로토콜
  - 특정한 서비스를 제공하기 위해 개발자가 임의로 정의한 통신 프로토콜
  - 예 : 온라인 수강신청 서비스를 위한 프로토콜

1.1.2 OSI 7 계층 구조

- **인터넷은** TCP/IP **통신 프로토콜을 기반으로 동작한다**
- OSI 7 **계층 구조**
  - 1970**년대 초**
  - **현재는 거의 사용되지 않음**



OSI 7 계층 구조 (1)

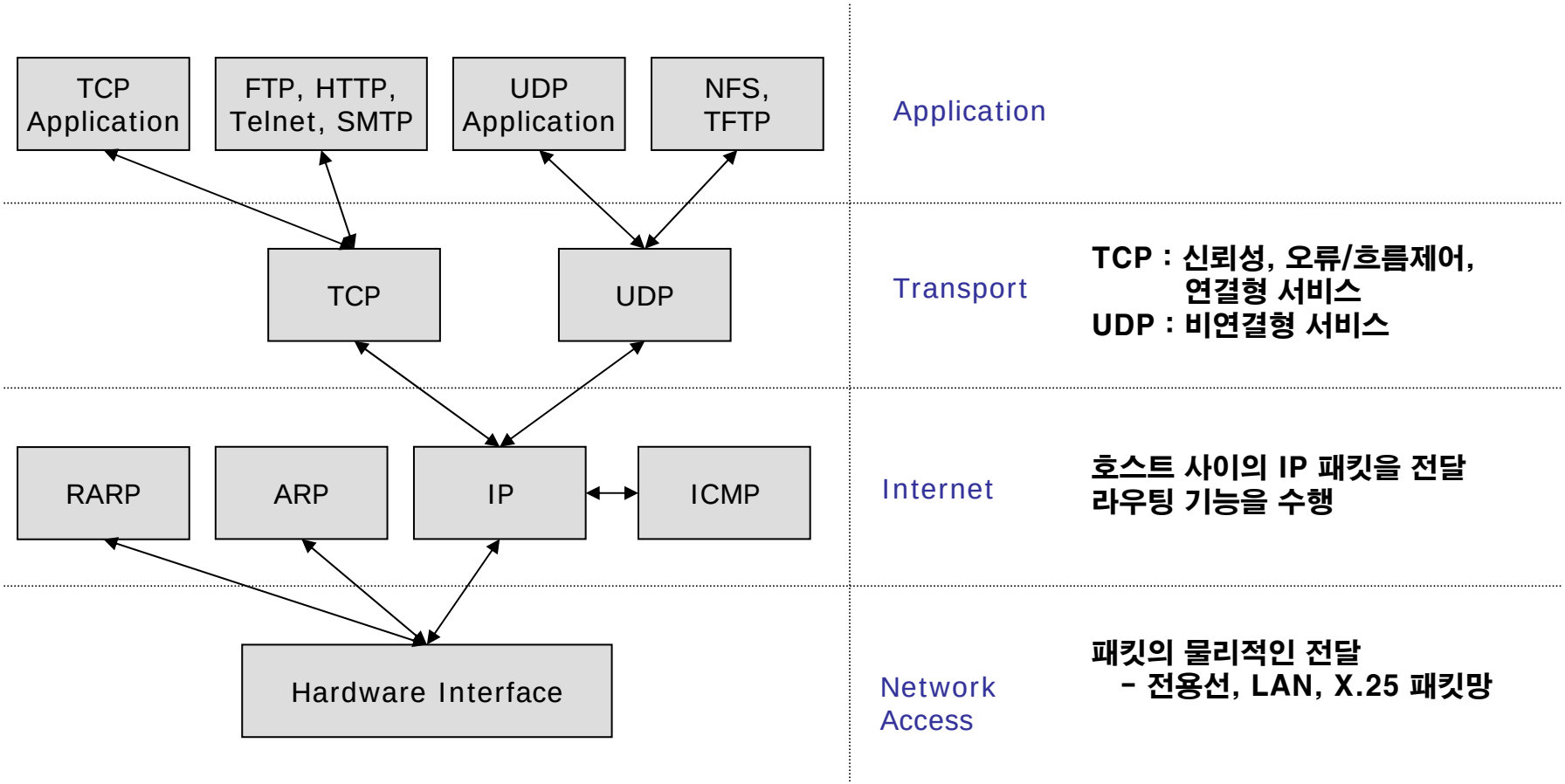
- 물리 계층
  - 비트를 전송매체를 통하여 효율적으로 전송
- 링크 계층
  - 프레임(PDU)을 노드 사이에 신뢰성 있게 전송
  - 링크 개설/해제, 프레임 경계식별, 에러제어, 흐름제어
- 네트워크 계층
  - 패킷을 목적 호스트까지 전달하는 교환 기능
  - 데이터의 직접적인 운반은 다루지 않음
  - 주소 분석, 논리적 연결설정/해제, 패킷 단위 흐름제어, 경로배정
- 트랜스포트 계층
  - 계층 3을 이용하여 종점 호스트 사이의 데이터 송수신
  - 종점간 연결관리, 에러제어, 흐름제어 수행
  - 종점간 프로토콜(네트워크 내부 장비에서는 처리하지 않는 계층)

OSI 7 계층 구조 (2)

- **세션 계층**
  - **통신 서비스의 개설, 유지, 종료**
  - **계층 4이하의 기능을 이용**
  - **종점 호스트 프로세스에서 이뤄지는 통신 관리 프로토콜**
- **표현 계층**
  - **데이터 표현 방식이 서로 다른 호스트 사이의 통신 지원**
  - **데이터의 표준화된 표현 방식의 사용**
  - **코드 변환, 데이터 압축**
  - **데이터의 암호화, 복호화**
- **응용 계층**
  - **네트워크를 이용한 최종 응용 서비스 제공**
  - ftp, telnet, e-mail, **웹 서비스, 망관리, 분산처리 등**

1.1.3 TCP/IP 개요

- TCP/IP **프로토콜 구조**



TCP/IP 프로토콜 스택 (1)

- TCP/IP 프로토콜
  - TCP, IP, UDP, IGMP, ARP, RARP를 모두 포함
- 네트워크 액세스 계층
  - IP 데이터 그램을 전달
  - 서브네트워크를 이용하는 프로토콜
    - 이더넷, DSL(Digital Subscriber Line)
- 인터넷 계층(IP 계층)
  - IP 데이터그램을 최종 목적지까지 전달
  - 핵심기능 : IP 데이터그램을 어떤 경로로 전달할지를 정하는 라우팅
  - ICMP
    - IP 데이터그램의 전송을 돕는 제어 프로토콜
    - 호스트 또는 라우터 사이에 에러 정보나 제어 정보 전달
  - IGMP
    - 멀티캐스팅을 지원하는 프로토콜

TCP/IP 프로토콜 스택 (2)

- **트랜스포트 계층**
  - TCP**와** UDP
  - TCP
    - **신뢰성 있는 연결형 서비스 제공**
      - **확인 응답**(ACK), **체섬**(checksum), **재전송**, **종점간 흐름제어 등을 사용**
    - ftp, telnet, e-mail, http **등**
    - **종점간 바이트 스트림 제공**
  - UDP
    - **비연결형 트랜스포트 서비스 제공**
    - **연결설정**, **연결종료 과정이 없이** IP **데이터그램을 전송**
    - **스트림이 아닌 데이터그램 단위로 송수신**
    - **데이터 분실 확인**, **전달 순서를 보장하지 않음**
    - TCP**와 달리 네트워크로부터 혼잡제어를 받지 않음**
  - **혼잡제어** : **인터넷의 전송 용량의 한계로 혼잡이 발생하면 송신 측이 전송 속도를 늦추어 혼잡을 피하게 함**

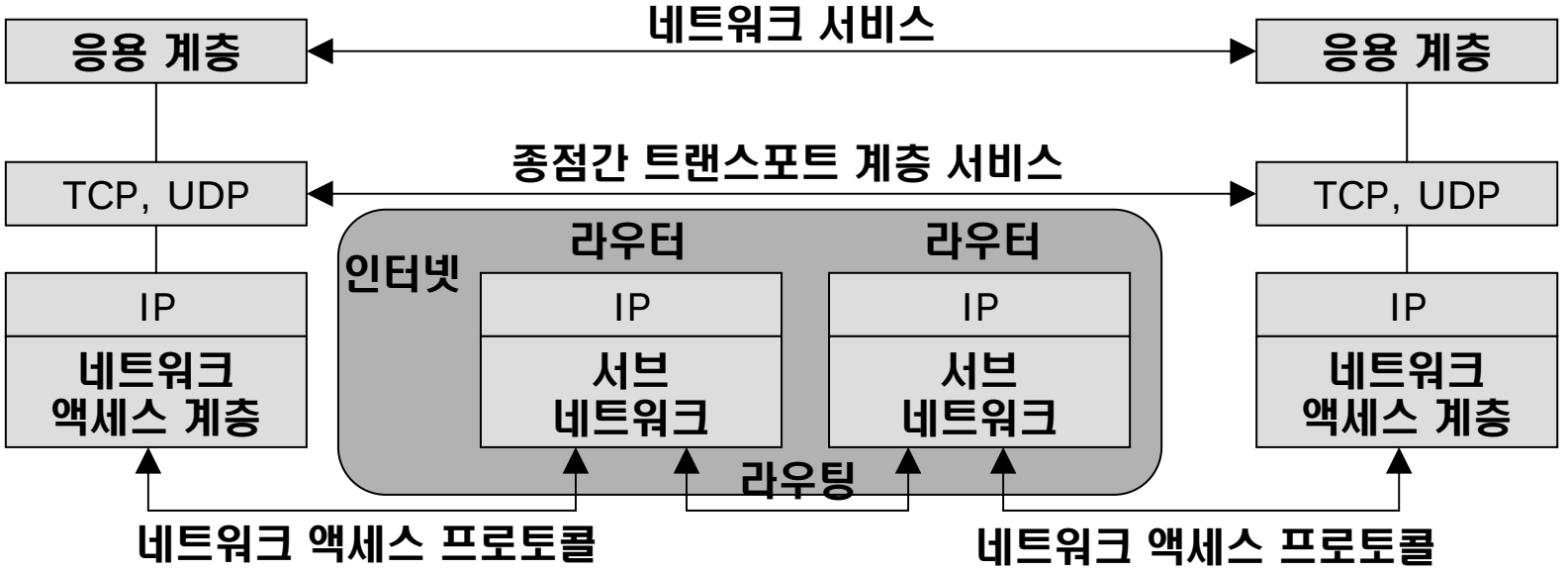
TCP/IP 프로토콜 스택 (3)

- **TCP와 UDP가 지원하는 응용 계층 서비스**

트랜스포트 프로토콜	응용 계층 서비스
TCP	• File Transfer Protocol(FTP) • Telnet • Simple Mail Transfer Protocol(SMTP) • HyperText Transfer Protocol(HTTP) • Border Gateway Protocol(BGP)
TCP와 UDP **모 두 지 원**	• Network File System(NFS) • Domain Name System(DNS) • Remote Procedure Call(RPC)
UDP	• Trivial FTP(TFTP) • Bootstrap Protocol(BOOTP) • Dynamic Host Configuration Protocol(DHCP) • Routing Information Protocol(RIP)

인터넷과 서브네트워크

- 인터넷을 이용한 두 호스트 사이의 통신
  - 인터넷을 이용하기 위해 TCP/IP 프로토콜이 설치되어 있어야 함
  - 호스트는 라우터들을 경유하여 서로 연결
  - 호스트
    - 인터넷에 물리적으로 접속하기 위해 네트워크 액세스 프로토콜이 필요
    - 네트워크 액세스 프로토콜 : IP 계층이 서브네트워크 이용을 위한 프로토콜
  - 서브네트워크
    - 이더넷, 패킷 교환망, DSL, ATM등 데이터를 실제로 전달해 주는 네트워크

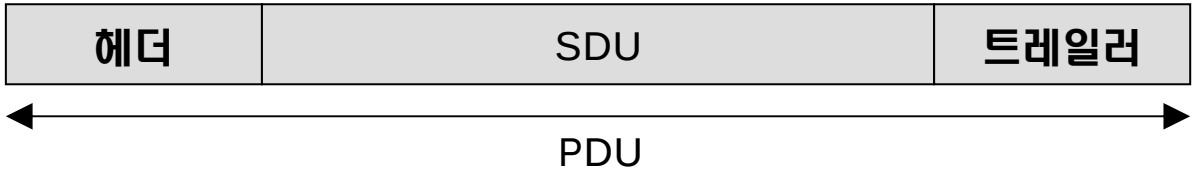


1.1.4 통신 프로토콜 표현 방식

- **통신 프로토콜의 내용을 표현하기 위한 방식**
  - **필드정의**
  - **파라미터 정의**
  - **상태 정의**
  - **동작 내용 정의**

필드 정의

- 프로토콜 데이터 단위(PDU)의 헤더나 트레일러의 필드 이용
- 서비스 데이터 단위(SDU)
  - 통신 프로토콜 계층에서 상위 계층으로 전달하는 데이터 부분
  - 사용자 데이터 또는 유료부하(payload)라고 함
  - SDU에 헤더와 트레일러가 추가된 것이 PDU



- 필드 정의에 의한 프로토콜 표현 방식
  - 헤더나 트레일러의 지정된 필드에 프로토콜 처리 내용을 기록하는 방식
    - 주소 필드, 순서번호, 에러처리 코드 필드 ...
  - 헤더나 트레일러를 크게 정의할 경우
    - 다양한 프로토콜을 쉽게 표현
    - 오버헤드가 증가하여 전송 효율 저하

파라미터 정의

- 파라미터 사용 예
  - 연결요청을 보내고 응답이 올 때까지 기다림
    - 기다리는 시간(파라미터)를 미리 정하고 그 시간이 지나면 오류를 발생
  - 데이터 전송에서 에러가 발생하면 계속 재전송을 하도록 한 경우
    - 미리 정한 최대 재전송 횟수(파라미터)를 통해서 프로토콜을 표현

- 통신 처리 성능
  - 파라미터를 너무 크거나 작게 하면 통신 처리 성능이 저하될 수 있음
  - 파라미터 값을 적절히 정해야 함
  - 경우에 따라서는 네트워크 상태에 따라 자동으로 변경되기도 함

상태 정의

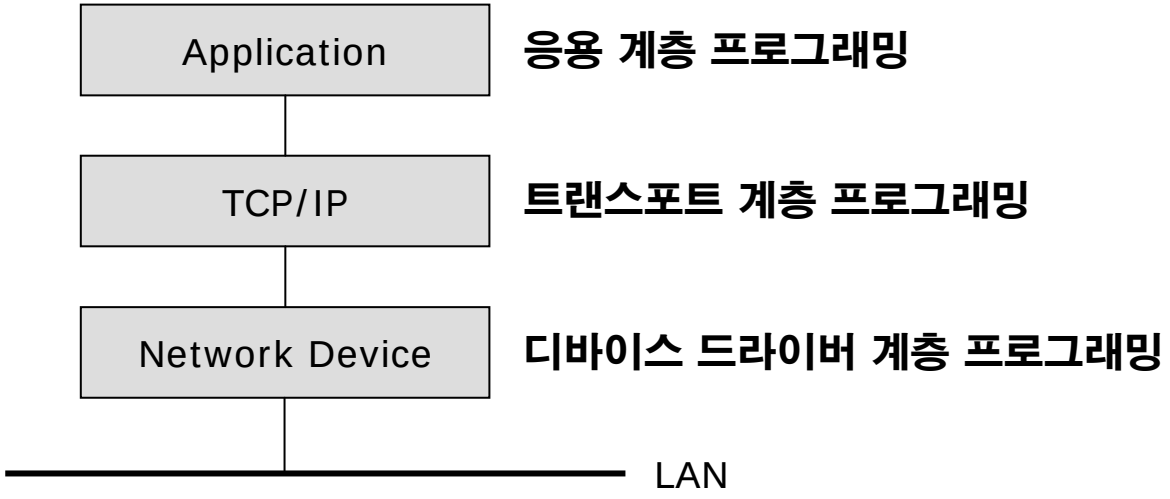
- **통신 상태**
  - **과거의 동작 결과를 바탕으로 얻어진 현재의 동작 환경**
  - TCP **계층의 여러 상태**
    - **통신 시작 전의 준비 상태**
    - **연결 요청 상태**
    - **데이터 송수신 상태**
    - **에러가 발생한 상태**
    - **정상적인 통신 종료 상태**
  - **통신 서비스는 자신이 존재하는 상태에 따라 처리할 내용을 결정**
    - **예 : 종료된 상태에서는 데이터 수신이 불가**

동작 내용 정의

- 상태 정의와 밀접한 관련
- 동작 내용 구현
  - 모든 종류의 상태에 대하여 발생 가능한 모든 입력을 처리해야 한다
    - 예 : 파일 전송 준비 상태에서 정확한 파일명이 입력되면 파일을 전송
- 프로토콜의 어떤 계층에서 받을 수 있는 세 가지 입력의 종류
  - 상위 계층으로부터의 명령
  - 하위 계층으로부터의 서비스
  - 자체 이벤트
- 상위 계층으로부터의 명령
  - 해당 계층이 처리하도록 주어지는 작업
  - TCP 계층 : 연결 요청 명령, 데이터 송수신 명령, 연결종료 요청 등
- 하위 계층으로부터의 서비스
  - 하위 계층이 보내주는 SDU의 내용
- 자체 이벤트
  - 타이머 종료, 카운터 횟수의 만료 등

1.1.5 네트워크 프로그래밍 계층별 분류

- **프로토콜 계층에 따른 네트워크 프로그래밍 분류**
  - **응용 계층 프로그래밍**
  - **트랜스포트 계층 프로그래밍**
  - **디바이스 드라이버 계층 프로그래밍**



응용 계층 프로그래밍

- 네트워크 응용 패키지, TCP/IP 응용 프로토콜을 이용
- 데이터의 송수신을 구체적으로 다루지 않고 응용작업을 네트워크 환경에서 실행
  - 유닉스의 rsh, rcp
  - RPC
  - http
  - CORBA를 이용한 분산 프로그래밍
- 장점
  - 복잡한 네트워크 서비스를 편리하게 구현할 수 있음
- 단점
  - 하위 계층의 구체적인 동작(연결설정, 데이터그램 송수신, 흐름제어)을 직접 제어할 수 없음
  - 아래 계층 프로그래밍에 비해 통신 효율이 떨어질 수 있음
    - 트랜스포트 계층
    - 디바이스 드라이버 계층 프로그래밍

트랜스포트 계층 프로그래밍

- TCP나 UDP와 같은 트랜스포트 계층의 기능을 직접 이용
- TCP를 이용할 경우 연결설정, 흐름제어, 에러제가가 가능
- 대표적인 API
  - socket

디바이스 드라이버 계층 프로그래밍

- OSI 계층 2 이하의 인터페이스를 직접 다루는 프로그래밍
  - 예) MS의 NDIS(Network Driver Interface Specification) API
    - Windows에서 이더넷 프레임의 송수신을 직접 처리
- 프레임의 송수신을 구체적으로 제어
  - LAN 카드 개발 시 테스트 프로그램 작성에 사용
- 프레임을 송수신하는 기능만 이용
  - 흐름제어, 에러제어, 인터넷 주소 관리 등의 기능은 별도로 구현해야 함

네트워크 프로그래밍의 계층별 분류와 특징

계 층	사용 프로토콜	특 징
응용 계층	http, CORBA, rsh, rcp, RPC	• 이미 작성된 네트워크 패키지 활용 • 복잡한 서비스를 간단히 제공할 수 있음 • 전송 효율이 떨어질 수 있음 • 프로그램 작성, 변경 및 운영이 용이
트랜스포트 계 층	TCP/IP, 유닉스 socket, Winsock	• 데이터그램 단위위 데이터 송수신 처리 • 종점간 연결관리, 흐름/에러제어 이용 가능 • 인터넷 응용 프로그래밍의 기초
디바이스 드라이버 계층	NDIS, Raw 소켓, 패킷 캡쳐	• LAN에서 프레임 단위의 송수신 처리 • 흐름제어, 에러제어는 사용자가 직접 처리

1.2 네트워크 프로그램 구현 모델

- **클라이언트-서버 모델**
- P2P
- **분산객체 모델**

1.2.1 클라이언트-서버 모델

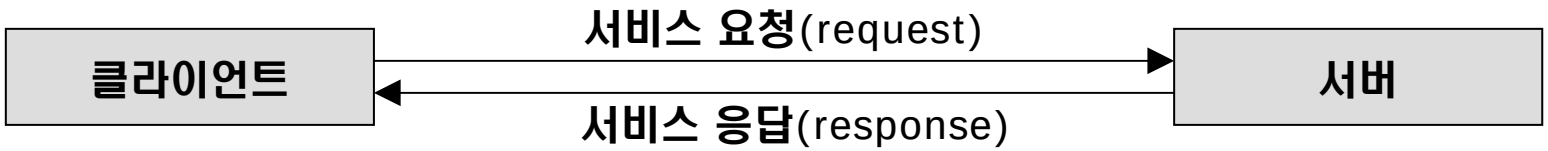
- 대부분의 네트워크 프로그램의 구현 모델

- 서버와 클라이언트
  - 서버 : 서비스를 제공하는 장비
  - 클라이언트 : 서비스를 이용하는 장비

- 서버
  - 일반적으로 클라이언트보다 구현이 복잡
    - 클라이언트의 인증
    - 정보 보호
    - 동시 서비스
    - 안정성

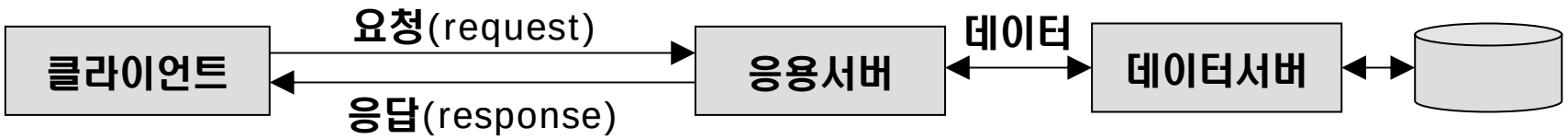
- 서버를 먼저 설계하고 클라이언트를 설계하는 것이 편리

2-tier 클라이언트-서버 모델



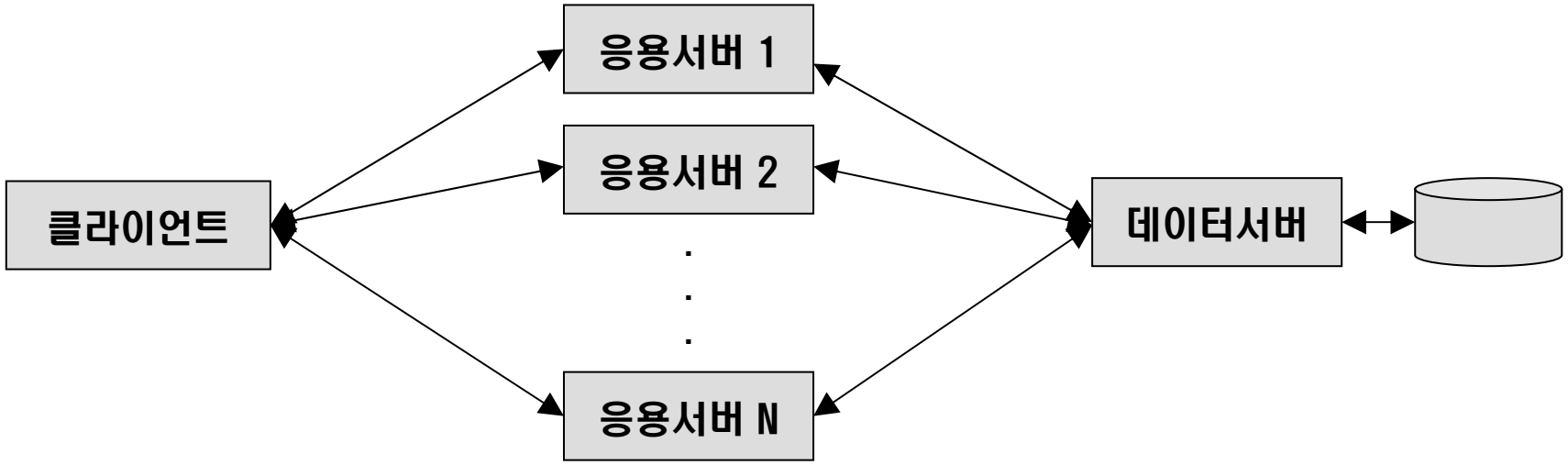
- **클라이언트가 서버로 서비스를 요청하고 서버는 이에 응답**
- **대부분의 통신 프로그램**
  - **웹**(http), ftp, telnet, mail
- **단점**
  - **서버에서의 병목 현상**
  - **클라이언트의 증가는 서버에 트래픽 집중과 처리 용량 부족 현상 발생**
- fat **클라이언트**
  - **서버의 기능 일부를 클라이언트에 구현 한 것**
  - **프로그램의 업그레이드, 버전관리 문제 발생**

3-tier 클라이언트 서버 모델



- 2-tier 클라이언트-서버 모델의 문제점을 개선한 구조
- "응용서버"와 "데이터서버"로 구분
  - 클라이언트는 응용 서버에 서비스를 요청
  - 응용서버는 데이터 서버로부터 데이터를 얻어 클라이언트에 응답
- 장점
  - 클라이언트는 데이터서버의 정보가 필요 없음
  - 클라이언트가 같은 요청을 동시에 하면 응용 서버는 이를 한번만 처리
    - 데이터 서버의 통신 부담 저하

n-tier 클라이언트-서버 모델



- 3-tier 모델을 확장
  - 여러 버전의 응용서버가 존재
  - 기본 동작은 3-tier 모델과 같지만 응용 서버가 여러 형태로 구현
- 장점
  - 클라이언트가 필요에 따라 다른 응용 서버를 선택할 수 있음
  - 서비스 제공 도중 새로운 응용서버 추가가 가능
  - 서비스의 다양성, 확장성, 버전관리(업그레이드)

P2P 모델 (1)

- 서버, 클라이언트의 역할이 미리 정해지지 않음
  - 경우에 따라 서버 또는 클라이언트가 될 수 있음
  - 동등한 자격으로 정보를 주고받는 통신 모델
- 내부적으로는 클라이언트-서버 모델로 동작
  - 참가자가 필요한 시기에 서버와 클라이언트의 역할을 수행
- P2P 모델의 종류
  - 순수 P2P 모델
  - 하이브리드형 P2P 모델 : 인덱스 서버를 두는 형태

P2P 모델 (2)

- **순수 P2P 모델**
  - **참가자들이 동등한 자격으로 정보를 이용하는 모델**
    - **원하는 자료를 주변의 참가자에게 요청**
    - **원하는 정보를 찾을 때까지 인접한 참가자에게 계속 전파**
    - **원하는 정보를 갖고 있는 참가자가 응답**
  - **특징**
    - **동작이 단순**
    - **비효율적**

- **하이브리드형 P2P 모델**
  - **순수 P2P 모델을 개선한 방식**
  - **인덱스 서버가 존재**
    - **원하는 자료를 인덱스 서버에게 문의(요청)**
    - **인덱스 서버는 요청한 정보를 가지고 있는 참가자의 주소를 반환**

1.2.2 분산객체 모델

- **미들웨어**
  - TCP/IP**와 같은 통신 서비스를 이용한 지능화된 네트워크 서비스 제공**
  - **제공하는 기능**
    - **서비스 등록, 검색, 이용**
  - **분산객체가 통신 기능을 제공**
  - CORBA, Java RMI, .NET, SOAP**등이 미들웨어 환경을 제공**
- **클라이언트, 미들웨어, 서버의 관계**
  - **클라이언트 : 미들웨어를 통해서 서비스 요청을 서버로 전송**
  - **서버 : 미들웨어를 통해서 클라이언트로 서비스 결과를 전송**
- **클라이언트와 서버**
  - **서버와 클라이언트의 역할이 미리 지정되지 않고 모두 객체로 구현**
  - **임의의 컴퓨터에 이 객체들이 존재**
- **인터페이스 정의 부분과 서비스 내용 구현**
  - **클라이언트가 인터페이스만 알면 분산객체 서비스를 받을 수 있음**
  - **클라이언트와 서버 프로그램이 독립적으로 구현 및 업그레이드 가능**

1.2.3 서버 구현 기술

- **서버의 구현 방식이 네트워크 서비스 성능에 중요한 영향을 줌**
- **서버 구현 기술**
  - **연결형과 비연결형 서버**
  - stateless**와** stateful **서버**
  - iterative**와** concurrent **서버**

연결형과 비연결형 서버 (1)

- **연결형 서비스**
  - **종점간 연결 설정**, **데이터 송수신**, **연결 해제 등 세 단계의 절차를 거침**
  - **주로** TCP**를 이용하여 작성**
  - **데이터의 안정적인 전달을 보장**
  - **회선교환 서비스**, TCP **프로토콜**, telnet, ftp **등**

- **비연결형 서비스**
  - **종점간 연결 설정/해제 작업 없이 바로 데이터를 주고받는 방식**
  - **주로** UDP**를 이용하여 작성**
  - **안정적인 데이터 전달을 보장하지 않으므로 응용 프로그램에서 처리**
  - **이더넷 프로토콜**, IP **계층 프로토콜**, UDP **프로토콜**, **웹 서비스 등**

연결형과 비연결형 서버 (2)

- 연결형 서버와 비연결형 서버의 특징 비교

서버의 종류	특 징
연결형 서버	• TCP와 같은 연결형 프로토콜을 주로 사용 • 데이터의 안정적인 전달을 보장 • 각 클라이언트마다 연결을 개설 • 클라이언트 수가 증가하면 서버에 부담이 클 수 있음
비연결형 서버	• UDP와 같은 비연결형 프로토콜을 주로 사용 • 데이터의 안정적인 전달을 보장하지 않음 • 클라이언트마다 연결을 설정할 필요가 없음 - 서버의 부담이 적음(메모리 사용 등) • 방송형, 멀티캐스팅형 서비스에 적합

Stateful과 Stateless 서버

- Stateful 서버
  - 클라이언트와의 통신 상태를 계속 추적하여 서비스를 제공
    - 상태 : 연결설정, 데이터 송수신 기록, 에러 등의 모든 서비스 결과
  - 서버의 상태에 따라서 요청마다 처리할 응답이 달라질 수 있음
    - 상태 정보의 사용으로 신속한 응답과 송수신 데이터의 양도 줄일 수 있음

- Stateless 서버
  - 상태 정보를 이용하지 않고 독립적인 요청에 의해 서비스를 제공
    - 클라이언트로부터 새로 도착한 요청 정보만을 이용하여 서비스를 제공
    - 틀린 상태 정보를 사용할 가능성이 없으므로 안정적인 동작이 가능
    - 서버가 처리하는 모든 정보가 요청 메시지마다 있어야 함

- Stateful과 Stateless의 선택
  - 네트워크가 안정적일 경우 stateful 이 유리
  - 인터넷 환경은 데이터그램의 분실, 에러, 지연, 순서 바뀜의가능성이 크므로 stateless 서버를 사용하는 것이 일반적으로 안전

Iterative와 Concurrent 서버 (1)

- Iterative 서버
  - 클라이언트의 요청을 순서대로 처리
  - 클라이언트의 요청이 짧은 시간에 처리할 수 있는 경우에 적합
  - Concurrent 서버보다 구현이 간단함

- Concurrent 서버
  - 클라이언트의 요청을 동시에 처리
  - 다중처리 기능이 필요
    - 멀티프로세스에 의한 다중처리
      - 새로운 클라이언트 접속시 이 클라이언트를 담당하는 프로세스 생성
      - 클라이언트의 증가에 따라 프로세스 수도 증가
    - 다중화 이용
      - 하나의 프로세스가 여러 작업을 동시에 처리하는 기능

Iterative와 Concurrent 서버 (2)

- Iterative 서버와 Concurrent 서버의 특징 비교

서버의 종류	특 징
Iterative 서 버	• 요청 순서대로 서비스를 처리 • 각 서비스의 처리 시간이 짧을 때 사용 • 서버 프로그램의 구현이 비교적 단순함
Concurrent 서 버	• 여러 요청에 대해 동시에 서비스를 제공 • 서버 프로그램의 구현이 다소 복잡함 • 다중처리 기능이 필요 • 각 서비스 처리 시간이 불규칙적이거나 길 때 사용

1.3 네트워크 서비스의 성능

- 네트워크 프로그램의 성능을 나타내는 주요 요소

1.3.1 성능 척도

- **네트워크 프로그램은 서비스 종류에 따라 요구 조건이 다르다**
  - **비디오 스트리밍 서비스는 실시간 데이터 전달 보장 이 중요**
  - **은행 거래를 처리하는 프로그램은 에러가 발생하지 않아야 함**

- **네트워크 서비스 성능 척도의 주요 요소**
  - **처리 용량**(capacity)
  - **안정성**(stability)
  - **처리율**(throughput)
  - **지연**(delay)
  - **효율**(efficiency)

성능 척도의 주요 요소 (1)

- **처리 용량**
  - **최대 가입자수**
  - **최대 동시 이용자 수**
  - **초당 서비스 처리량**

- **안정성**
  - **서비스가 다운되지 않는 정도**

- **처리율**
  - **순수 데이터 전달 속도를 채널의 용량(대역폭)으로 나눈 값**
  - **예)**
    - 5Mpbs **통신 채널을 통해 평균** 2Mbps**의 속도로 데이터를 전송할 경우의 처리율** : 0.4

성능 척도의 주요 요소 (2)

- **지연**
  - **어떤 서비스를 요청해서 응답을 받는 데까지 걸리는 소요 시간**
  - **지연의 종류**
    - **전송지연** : **프레임을 통신 채널로 보내는데 걸리는 지연**
    - **전파지연** : **데이터를 목적지까지 전달하는데 걸리는 지연**
    - **처리지연** : **통신장비에서 서비스를 처리하는데 걸리는 지연**
    - **대기지연** : **네트워크에 트래픽이 일시적으로 몰려서 발생하는 지연**

- **효율**
  - **전송된 총 데이터 중에 순수 데이터가 차지하는 비율**
  - **예)**
    - **패킷의 평균 길이**: 1000 **바이트**
      그 중 200**바이트가 통신 프로토콜 처리를 위한 오버헤드일 경우의**
      효율 : 80%

최적의 패킷 크기

- 유료 부하가 큰 경우
  - 패킷 내에 빈 공간이 생길 확률이 높아짐
  - 전송 에러가 발생했을 경우 재전송 데이터 량이 많아짐
- 유료 부하가 작은 경우
  - 유료 부하에 비해 오버헤드가 큼
  - 전송 횟수가 증가
- 표준 통신 프로토콜의 PDU
  - 적절한 크기로 정해졌음
- 사용자 정의 통신 프로토콜 설계
  - 최적의 패킷 크기 선택이 필요

1.3.2 통신 자원과 비용

- **네트워크 프로그램 구현 시 고려해야 할 사항**
  - **시스템 자원**(resource)
  - **자원을 사용하는 비용**(cost)

시스템 자원

- 시스템을 구성하기 위해 세 가지 자원을 고려

1. 통신 채널
   - 자원을 사용하려면 비용이 발생
   - 개발자는 전체 비용이 최소화 되도록 시스템을 구성해야 함
2. CPU 용량
   - 핸드폰과 같은 소형 무선 기기
     - 통신 비용과 CPU 사용량(전력 사용량)을 최소화
3. 메모리
   - 충분히 사용할 수 있는 환경
     - 통신 비용을 줄이기 위해 필요한 모든 데이터를 메모리에 올리는 것이 가능

- 비중과 비용을 고려하여 총 비용을 최소화
- 통신 장비의 종류, 통신 채널, 서비스 요구 조건, 총 투자 비용을 고려

1.3.3 플랫폼의 선택

- **플랫폼**
  - **프로그램이 실행되는 환경**
  - **하드웨어, 운영체제, 컴파일러의 조합**

- **널리 사용되는 플랫폼**
  - **유닉스(리눅스)**
  - **마이크로소프트** Windows
  - **자바**
  - **웹**
  - **모바일 플랫폼**

유닉스 플랫폼

- 버전
  - 시스템 V, BSD 유닉스, FreeBSD, Solaris, HPUX, AIX, Linux 등
- 유닉스
  - 높은 안정성으로 대형 네트워크 시스템에서 사용
  - 1980년대 초부터 보급
  - 다양한 하드웨어에 설치
    - 대형 컴퓨터, 워크 스테이션, PC, 임베디드 시스템
- 리눅스
  - 최근 PC 뿐 아니라 여러 기종의 대형 컴퓨터에도 사용되는 추세
- 유닉스 환경에서의 네트워크 프로그램 개발
  - 유닉스 시스템 프로그래밍에 대한 수준 높은 이해가 필요
    - 프로세스
    - 프로세스간 통신
    - 시스널 처리
    - 스레드 프로그래밍
    - 파일 관리

Windows 플랫폼

- Windows 계열의 운영체제가 설치된 환경
- 주요 프로그래밍 언어
  - Visual C++
  - Visual Basic
  - C#
- 장점
  - 가장 널리 보급된 플랫폼
  - 프로그램 개발 도구가 다양하게 제공
  - 개발할 프로그램에 GUI가 많은 경우에 유리
- 단점
  - 유닉스에 비해 안정성이 떨어짐
  - 보안 대책은 Microsoft가 제공하는 패치 방법 외에는 없음

자바 플랫폼

- 장점
  - 플랫폼에 독립적으로 프로그램 개발이 가능
    - 유닉스, Windows 등의 운영체제에 관계없이 동일하게 실행
  - 순수 객체지향적인 개발 방식 제공
  - 코드 가독성의 우수함
  - 서버측 프로그램의 안정성
  - 모바일 프로그래밍의 용이성
- 단점
  - 느린 실행 속도
    - 시스템 고유의 운영체제 위에 JVM 설치
  - 운영체제의 고유 기능을 모두 수용할 수 없음
    - 운영체제에 의존적인 기능을 자바만으로 구현할 수 없고 C로 작성
    - 예) JNI(java Native Interface) : 시스템에 의존적인 기능을 C로 작성하고 자바에서 호출하여 사용

웹 플랫폼

- http 프로토콜을 이용하는 통신 프로그램 환경
- 장점
  - 특정 운영체제에 종속되지 않는 프로그램 개발이 용이
  - 방화벽으로 보호된 서버 사용이 용이
  - 통일된 표현 방식 (HTML, XML)
    - 운영체제에 종속되지 않는 문서 관리가 가능
- 단점
  - 서비스 요청 및 응답의 느린 처리 속도
    - 모든 데이터가 http 프로토콜을 통과
- 프로그램 도구
  - PHP
  - Perl
  - Java Script
  - JSP
  - Java Servlet
  - ASP

모바일 플랫폼

- 무선 정보 기기의 프로그램 실행 환경
  - 핸드폰, 소형 PC, PDA 등
- 모바일 기기의 특징
  - 전지 사용으로 CPU 사용량을 최소화
  - 무선 통신시 많은 전력 사용
    - 통신 시간과 데이터 송수신 량을 최소화
  - 작은 화면
  - 작은 메모리 용량
- 무선 플랫폼 프로그래밍에서는 제한된 시스템 자원을 효율적으로 사용할 수 있도록 프로토콜을 설계하고 구현 방식을 최적화해야한다.

1.3.4 네트워크 프로그램 영역

- **네트워크 프로그램의 구현 영역에 따른 분류**
  - **컴퓨터 사이의 데이터 통신**(data communication)
  - **컴퓨터 내의 데이터 처리**(data processing)
  - **컴퓨터 내의 프로세스간 통신**(inter-process communication)

- **컴퓨터 사이의 데이터 통신**
  - TCP/IP**등의 통신 프로토콜을 이용한 패킷 송수신**
  - **대역폭에 의존**
- **컴퓨터 내의 데이터 처리**
  - **파일 입출력, 데이터베이스 액세스, 캐싱, 멀티미디어 신호 처리 등**
  - CPU **성능에 의존**
- **컴퓨터 내의 프로세스간 통신**
  - **멀티프로세스, 멀티스레드로 구현하는 경우에 필요**
  - **프로세스 작업 분담 기술, 프로세스간 데이터를 주고받는 기술, 자원을 프로세스들이 충돌없이 사용하는 기술에 의존**

1.4 유닉스 프로그래밍 환경

1.4.1 네트워크 환경

- **현재 네트워크의 정보를 알아내는데 사용되는 유닉스 명령어들**
  - ping
  - arp
  - host
  - ifconfig(winipcfg, ipconfig)
  - netstat
  - traceroute

네트워크 서비스 시작 (1)

- **리눅스에서 인터넷 환경을 확인하고 설정하는 방법**
  - **자신의 호스트가 인터넷에 연결되어 동작 중인지 확인**
    - **네트워크 디바이스의 상태 확인**

```
$ ifconfig -a
###  리눅스에서의 출력 예
eth0   Link encap:Ethernet  HWaddr 00:C0:26:29:DE:C7
         inet addr:213.152.51.63  Bcast:213.152.51.255
            Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:5819072 errors:0 dropped:0 overruns:0 frame:0
         TX packets:2225607 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:100
         RX bytes:1983603057 (1891.7 Mb)  TX bytes:2881327720 (2747.8 Mb)
       Interrupt:11 Base address:0x9000

###  솔라리스에서의 출력 예
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
       inet 213.152.51.63 netmask ffffff00 broadcast 213.152.51.255
```

네트워크 서비스 시작 (2)

- **네트워크 설정 파일**

```
# 파일명 : /etc/sysconfig/network
NETWORKING=yes                    # 호스트가 네트워크를 지원함
HOSTNAME=yicho # 자신 호스트의 도메인 이름 등록
GATEWAY=211.221.225.1             # 게이트웨이 IP 주소

# 파일명 : /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0                       # 이더넷 네트워크 카드 이름
ONBOOT=yes                        # 부팅 시 자동 활성화
BOOTPROTO=static
BROADCAST=211.221.225.255
IPADDR=211.221.225.175            # 자신의 IP 주소
NETMASK=255.255.255.0
GATEWAY=211.221.225.0
# 파일명 : /etc/resolve.conf
nameserver 168.126.63.1           # 도메인 네임(DNS) 서버 등록
```

- **아래의 명령으로 네트워크 서비스 시작**

  /etc/init.d/network restart

ping

- **다른 호스트와의 연결 상태를 확인**

```
$ ping yahoo.co.kr
PING yahoo.co.kr (211.32.119.151) from 210.115.36.231 : 56(84) bytes
    of data.
64 bytes from yahoo.co.kr (211.32.119.151): icmp_seq=1 ttl=243
    time=15.2 ms
64 bytes from yahoo.co.kr (211.32.119.151): icmp_seq=2 ttl=243
    time=10.6 ms
64 bytes from yahoo.co.kr (211.32.119.151): icmp_seq=3 ttl=243
    time=14.2 ms
64 bytes from yahoo.co.kr (211.32.119.151): icmp_seq=4 ttl=243
    time=9.91 ms
.....
```

arp

- **자신의 호스트가 가지고 있는 arp 캐쉬의 내용 확인**

```
$ arp -a
dmrl3.kangwon.ac.kr (210.115.36.120) at 00:90:08:03:C8:85 [ether] on
   eth0
? (210.115.36.1) at 00:00:00:00:00:0E [ether] on eth0 #IP호스트의 도메인 이름이 캐시에
```

```
$ arp -s ccnl.kangwon.ac.kr 11:22:33:44:55:66          # 정적인 MAC 주소 추가
$ arp –a                                               # arp 캐쉬 모두 보기
   ........
   ccnl.kangwon.ac.kr (210.115.36.232) at 11:22:33:44:55:66 [ether]
     PERM on eth0


$ arp -d ccnl.kangwon.ac.kr                            # 정보 삭제하기
$ arp –a                                               # arp 캐쉬 모두 보기
   ........
   ccnl.kangwon.ac.kr (210.115.36.232) at <incomplete> on eth0
```

host

- DNS **서버를 이용하여 임의의 호스트에 대한 정보를 확인**

```
$ host yahoo.co.kr
yahoo.co.kr has address 211.32.119.151


$ host 211.32.119.151
151.119.32.211.in-addr.arpa domain name pointer yahoo.co.kr.
```

ifconfig

- LAN **카드 등의 네트워크 인터페이스 카드의 설정 내용 확인**
- IP **주소, 서브넷 마스크 등의 설정 내용 변경**

```
$ ifconfig -a
eth0  Link encap:Ethernet  HWaddr 00:C0:26:29:DE:C7
      inet addr:210.115.36.231  Bcast:210.115.36.255
        Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:12320991 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4114404 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:3093272805 (2949.9 Mb)  TX bytes:2137817963 (2038.7 Mb)
      Interrupt:11 Base address:0x9000


lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:312549 errors:0 dropped:0 overruns:0 frame:0
      TX packets:312549 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:16657849 (15.8 Mb)  TX bytes:16657849 (15.8 Mb)
```

netstat (1)

- 네트워크의 여러 가지 상해는 확인하는 명령어
  - 네트워크 인터페이스 테이블 보기
  - 각 프로토콜이나 라우팅 테이블 정보 보기
  - 프로토콜별 통계 정보 출력

- 네트워크 인터페이스 상태

```
# netstat -i
Kernel Interface table
Iface  MTU  Met  RX-OK  RX-ERR  RX-DRP  RX-OVR  TX-OK  TX-ERR  TX-DRP TX-OVR Flg
eth0  1500    0   82646     0        0        0      35103      0        0        0     BMRU
lo     16436  0   68        0        0        0      68         0        0        0      LRU
```

- 라우팅 테이블 정보

```
# netstat -r
Kernel IP routing table
Destination     Gateway      Genmask        Flags    MSS  Window   irtt    Iface
210.115.36.0      *          255.255.255.0    U       40      0        0      eth0
127.0.0.0         *          255.0.0.0        U       40      0        0      lo
default        210.115.36.1   0.0.0.0         UG      40      0        0      eth0
```

netstat (2)

- **프로토콜별 통계 정보 출력**

```
# netstat -C
Active Internet connections (w/o servers)
Proto  Recv-Q  Send-Q Local Address          Foreign Address            State
tcp      0         0       myhost:http       61.100.10.212:41541      TIME_WAIT
tcp      0         0       myhost:http       61.100.10.208:33298      TIME_WAIT
tcp      0         0       myhost:ssh        myhost:1597              ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto  RefCnt   Flags      Type      State              I-Node       Path
unix    16       [ ]        DGRAM                       838          /dev/log
unix     3       [ ]        STREAM    CONNECTED         45442
unix     3       [ ]        STREAM    CONNECTED         45441
```

traceroute

- **임의의 호스트 어떤 경로(라우터들)를 통하여 연결되었는지 확인**

```
$ traceroute aaa.com
traceroute to aaa.com (192.201.144.33), 30 hops max, 40 byte packets
 1  210.115.36.1 (210.115.36.1)  0.359 ms  0.227 ms  0.219 ms
 2  210.115.63.145 (210.115.63.145)  0.432 ms  0.265 ms  0.257 ms
 3  aaa.com (192.201.144.33)  0.588 ms  *  0.499 ms
```

- **-f 옵션을 이용**

```
$ traceroute -f 9 allspice.lcs.mit.edu.
9   129.140.71.6 (129.140.71.6)  139 ms  139 ms  159 ms
10  129.140.81.7 (129.140.81.7)  199 ms  180 ms  300 ms
11  129.140.72.17 (129.140.72.17)  300 ms  239 ms  239 ms
12  * * *
13  128.121.54.72 (128.121.54.72)  259 ms  499 ms  279 ms
14  * * *                                          // 5초 이내에 ICMP 응답이 없을 경우
15  * * *
16  ALLSPICE.LCS.MIT.EDU (18.26.0.115)  339 ms  279 ms  279 ms
```

1.4.2 컴파일 환경

- gcc
  - 리눅스에 기본적으로 포함된 C 컴파일러
  - cc 컴파일러도 사용 가능
  - gcc 사용을 위해서는 gcc 컴파일러 경로를 환경변수에 등록해야 함
  - 리눅스는 Bash Shell을 기본으로 이용
- 환경 변수 지정 및 확인

```
$ PATH="/usr/bin:/usr/local/bin“              # 배시쉘
$ set path=( /usr/bin /usr/local/bin )        # C쉘
$ set PATH=( /usr/bin /usr/local/bin )        # C쉘

$ echo  $PATH
  /usr/bin:/usr/local/bin                     # 배시쉘의 경우(콜론으로 구분)
  /usr/bin /usr/local/bin                     # C쉘의 경우(공백으로 구분)
```