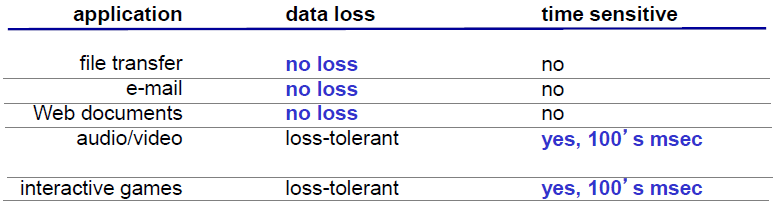
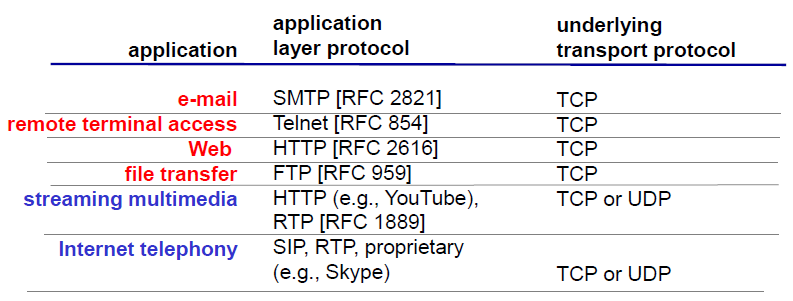
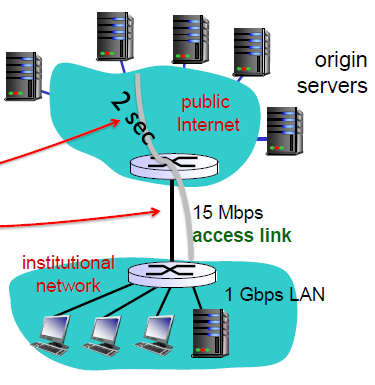
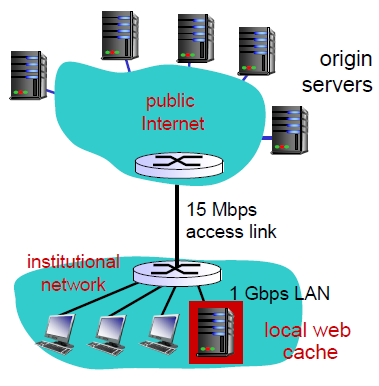
**Computer Network  
Chapter 2: Application Layer– part 1**

**Principles of network applications  
 Creating a network app:** 프로그램은 only end system(host)에서면 동작한다. 라우터 같은 다른 network core에서 동작할 필요가 없다.  
 **Application architectures** Client-server architecture:  
 server - host는 항상 켜져 있고, IP는 고정되어 있음. Data centers의 규모에 따름.  
 client - server와 communicate하며 연결이 되어있을 수도 있고 아닐 수도 있다. IP도 달라진다. 만약 client가 증가하면 data center를 증설하지 않는 이상 서비스 질이 낮아진다.  
 P2P architecture:  
서버 없이 구동되며, 다른 client랑 연결하여 (end system끼리 연결) communicate한다. 🡪 peer끼리 하기 때문에 🡪 **self scalability** (자가 확장성)이 있으며, 인원(client)가 증가하면 server 서비스도 증가한다.  
 **Processes communicating** Process: 호스트내에서 실행되는 프로그램  
**inter-process communicatio**n – 같은 host내에서 communicate하는 것 (OS에 의해 정의됨), message와 shared memory를 주고받는다. 다른 host에 있는 원격 process와 communicate할 때에는 message방식으로만 한다.  
client process가 서버에 요청 후 server가 client로 응답 🡪 그래서 server는 항상 켜져 있어야 하며, 요청 받을 준비를 하야한다.   
 **Sockets** 소켓은 application layer와 transport layer 사이에 위치하며, application이 networking하도록 도와준다. 🡪 소켓은 application layer와 하위 layer에서 interface역할을 해준다. Socket API를 이용하며, 네트워킹에서는 소켓이 최우선이다. (항상 먼저 만들어 줘야함.)  
 **Addressing processes** (IP address) host는 유일한 32bit IP address를 가짐 🡪 host의 identifier. 하지만 아이피 주소만으로 서버 패킷이 누구 것인지 모름. 🡪 Port Number를 통해서 서버의 응답이 누구 것인지 알 수 있다. IP address & Port Number  
  
  
  
 **Application-Layer Protocols**: applications(end systems)사이에 메시지를 주고받기 위해서는 그 메시지의 형태를 정의해야한다. – 소켓을 통해 애플리케이션들이 소통하는데 그 사이에 메시지 (문법 syntax, 형태 semantics를 결정해야한다. (메시지 타입: request, response , syntax: 문법, semantics: 의미, 규칙 등을 정의함)  
애플리케이션은 여러 종류이므로 다양한 어플리케이션들이 각각의 문법과 의미를 가지고 있다. (FTP, HTTP, SMTP,….) – 애플리케이션 종류에 따라 맞춤형.  
HTTP, SMTP는 open protocols이다. Skype는 proprietary protocols.  
  
Application Layer는 하위레이어의 서비스를 받는다. 각 애플리케이션들이 필요한 서비스는 다르다.  
🡪 data integrity(transport layer -TCP) – 데이터에 에러가 있는지 확인(무결성 유지) // 오디오나 비디오 (media)등은 중간중간에 에러(loss)가 발생해도 큰 문제가 없다.  
🡪 timing 전화 같은 경우는 바로바로 되야한다.  
  
TCP service:   
- reliable transport – 에러발생시 다시 보냄. 패킷의 순서를 맞춰주는 것.. // 앞 뒤 패킷은 연관이 있어서 TCP로 데이터를 보내기 전에  
🡪 connection-oriented: connection setup을 해야한다.  
UDP service:  
- unreliable data transfer: connection setup을 안 해도 된다. 🡪 가볍기 때문에 속도 빠름  
🡺 이런 이유로 대부분의 application들은 **TCP를 기본으로** 한다. 하지만 멀티미디어나 **전화같은 timing이 중요한 것은 UDP로** 하는 경우가 있다.  


**Web and HTTP**웹페이지는 object(text, picture등등)을 가져오기 위해 HTTP를 사용  
HTTP (Hyper Text Transfer Protocol) – web의 application layer protocol  
Web client: 웹 브라우저 – 웹 서버에 request를 하여 object를 받는다.   
Web server: 웹 클라이언트의 요청을 받아 response해준다.  
Uses TCP: 80포트를 이용하여 transfer한다.  
 **HTTP is stateless**: 서버가 stateless하다 = client가 서버에 연결해도 server에서는 client의 기록을 유지하지않는다. (server memory에 이득, 서버부화가 작아짐.) – HTTP는 stateless하지만, cookies를 이용하여 http state를 유지하는 경우도 있다.  
 **HTTP connection**TCP connection (client) 🡪 accept (server) 🡪 HTTP request message (client) 🡪 response message (server) 🡪 close TCP connection (server) 🡪 receives response message (client)  
이 과정에 걸린 시간을 **response time**: 2RTT + file transmission time(= transmission delay\_ packet size에 따라 달라짐)이라고 한다.  
RTT (round trip time): 작은 패킷이 client에서 server로 갔다 돌아오는 시간  
 **HTTP connections issues**: server는 요청에 대한 대답을 하면 TCP연결을 끊기 때문에 object transfer을 한번 밖에 못한다. 🡪 **Non-Persistent HTTP**  
🡺 그래서 여러 object를 2RTT로 받는 것보다 한번의 TCP연결로 받는 경우가 있다. 🡪 **Persistent HTTP**: 한번의 TCP connection으로 여러 object를 받는다. Server는 response를 보내고 connection을 끊지 않고 뒤에 올 request를 기다린다. 하지만 web object가 하나일 때도 server는 client의 request를 계속 기다리기 때문에 서버관리자가 time out을 걸어준다. 대부분 persistent HTTP를 default로 사용.  
 **HTTP request message**: 대부분 GET으로 요청, 암호 같은 것은 POST로 요청 – last modified라는 시간을 보고 캐시에 저장된 것이 오래되면 새로 요청해서 받아옴. 시간이 오래 안됐으면 그냥 띄움. – keep alive (time out에 대한 정보) – persistent를 위함.  
 **Cookies:** keeping state at server – stateless한 HTTP에서 server의 state를 유지하기 위해서! 유저의 identity에 따름.. cookies는 http랑 같이 사용.. cookies header line이 HTTP header line에 들어간다. 유저 정보는 웹사이트의 back-end DB에 저장되며 HTTP안에 쿠키 정보가 있다.  
server가 client에 response를 주는데, 그 때 임의의 아이디를 만들어서 자신의 DB에 저장하고 그 아이디를 response message에 같이 넣어서 보내준다. Client는 쿠키 id를 넣어서 request를 해주면 server에서 누군지 바로 안다. 🡪 그 유저가 뭘 보고 뭘 했는지 쿠키에 남는다. 하지만 쿠키 때문에 privacy 문제가 있다.  
 **Web caches (proxy server)**: 일반적으로 access ISP내에 존재 🡪 client가 server로 HTTP request를 보내면 proxy server를 거쳐서 간다. Proxy server는 마치 client의 HTTP request를 자기가 보낸 것처럼 server 보내고 response를 받아 다시 client에 response를 보낸다. Proxy server를 사용하는 이유는 **proxy server가 client쪽에 가까이 있기 때문에** client가 원하는 **object request가 proxy server에 저장되어 있으면, client의 요청이 굳이 server로까지 갈 필요없이 빠르게 proxy server만** 갔다 와도 된다. (response time이 줄어듦) 🡪 대부분 proxy server까지 갔다 오는데 (< 1ms), server까지 갔다 오는 시간은 (>100ms) (client에서 직접 sever로 갔다 오는 시간이나, client가 proxy server를 들렸다 server로 가는 시간이나 큰 차이가 없다.)  
**proxy server는 client 입장에서는 server, server입장에서는 client**  
ISP에서 밖으로 나가는 링크를 access link라고 한다.  
ISP안에는 proxy server가 있기 대문에 ISP입장에서는 access link를 이용해 밖으로 나가는 **횟수가 적어져서 부화가 줄어듦**  
ISP에서 외부서버까지 access link(15Mbps)를 통해서 갔다 오는데 RTT가 2초, 평균 bit size는 1Mbits, 평균 request rate는 15/sec  
🡺평균 데이터 rate는 15Mbps… access link 사용률이 거의 100퍼센트  
2초 + minutes (<< 100%) + msec 🡺 minutes이상의 시간이 걸림  
  
만약 access link를 100Mbps로 늘려주면 사용률은 약 15퍼로 줄고.. 걸리던 시간이 minute에서 🡪 2초 + msec +msec ==> 3secs 🡺 근데 비쌈  
  
Local Web cache를 두고, 만약 hit rate가 0.4면 access link로 외부서버로 나가는 것이 0.6이므로 사용률이 60퍼로 줄어든다.  
0.6 (2초) + 0.4 (msec) = 1.2sec  
  
proxy server를 두는 것이 값싸고 훨씬 빠르다.