1.

일반적인 쿼리에는 별도 오버헤드가 없는 UDP를 사용하는 것이 적절하다. 하지만 다음 3가지 경우에는 TCP를 써야 한다

1. 보내야 하는 패킷의 크기가 512바이트를 넘을 경우. UDP Packet은 512바이트 이상이 될 수 없기에 TCP로 보내야한다.
2. 응답이 없을 경우. UDP는 retransmission를 지원하지 않기 때문에, 이런 경우 TCP로 다시 쿼리를 날려야 할 것이다.
3. 중요한 데이터를 동기화해야 할 경우. 특히 DNS 서버간 데이터 동기화 때 UDP는 데이터 누락이 있을 수 있으므로 TCP도 사용해야 한다.

2.

circuit switching aims at real-time services and hence does not introduce any overhead at all in data transmissions.

연결하는데 시간이 오래 걸릴 수 있다. 연결 오버헤드가 없다곤 할 수 없다. 언제까지 연결을 계속 해놓을 순 없기 때문이다.

the utilization of assigned bandwidth in circuit switching is more efficient than that in packet switching

연결 할 때 정해놓은 대역폭 크기를 고정해서 사용하는데, 그 과정에서 모든 교환기의 성능을 효과적으로 사용할 순 없다. 즉, A, B, C 를 걸쳐 데이터가 도착한다고 하면 최대 대역폭 크기는 min(A, B, C)가 되기 때문이다. 패킷 스위치드 네트워크에서는 모든 대역폭을 최대한 사용하려고 하기 때문에 오히려 이 부분은 패킷 스위치드 방식이 우세다.

3.

Percentage of the time that server is busy = p = h/u

P0 = probability that there are no customers in the queue = 1-p = 1-h/u

Then, pn = p^n \* p0

“probability of finding more than 3 customers waiting for ordering” -> there are 4 customer, one ordered, and three is waiting for ordering -> 1 – P3.

So, 1-P3 = 1 - (1-h/u) \* (h/u)^3

Average number of customer in waiting line = h \* average waiting time.

Average waiting time =

4.

Hourglass

다양한 사용자들의 니즈를 만족시키려면 빠르게 변화해야 합니다. 하지만 아키텍쳐의 모든 것들을 변화시키는 데에는 많은 시간이 걸립니다. 필요한 부분(Layer)만 바꾸고, 기존의 사용할 수 있는 부분과 인프라는 계속 사용할 수 있다는게 인터넷을 성공적으로 이끌었습니다. 그런 점이 저런 모래시계 모양을 만들지 않았을까 생각합니다.

5.

MUST, REQUIRED, SHALL : Absolute Requirement of the Spectification

MUST NOT, SAHLL NOT : Absolute Prohibition of the Specification.

SHOULD, RECOMMEND : 왠만하면 이것을 하시오, 단 다른 방법을 사용할 것이면 전체 구현된 모습을 고려해서 신중히 선택할 것.

SHOULD NOT, NOT RECOMMEND : 왠만하면 이것을 사용하지 마시오, 단 이 방법을 사용할 것이면 전체 구현된 모습을 고려해서 신중히 선택할 것.

MAY, OPTIONAL : 해도 되고 안해도 된다.

6.

TFTP : Trivial File Transfer Protocol

Implemented on UDP

지원 : Read/Write Files from/to a remote server.

미지원 : List Directories, Authentication, etc of a regular FTP

UDP므로 512바이트 패킷을 보내고, 보내는 쪽이 데이터 보내면 받는 쪽이 ACK를 보냄. 패킷이 Timeout안에 안오면 마지막 패킷을 다시 보냄. Go Back N은 없는 것 같음.

패킷 헤더에는 블럭 넘버가 적혀있는데, 1부터 시작하고, ACK를 보낼 때 제대로 받은 블럭 넘버를 적어서 보낸다.

UDP, TCP에서 port를 사용하는 것 처럼, 여기선 TID를 사용하는데, UDP의 port값을 그대로 사용하는 듯 하다. 한 컨넥션에 대해 센더와 리시버 둘다 Randomly created TID가 있어야 한다. 물론 중복된 TID 사용은 안된다.

나머지 패킷 디테일만 추가.

string들은 맨 마지막에 NULL로 끝난다고 표기.

7.

SOCKET : int domain, int type, int protocol

Domain : protocol family like AF\_INET(network protocol Ipv4) or AF\_INET6(ipv6) etc..

Type : SOCK\_STREAM, SOCK\_DGRAM(Datagram), etc

Protocol : IPPROTO\_TCP, IPPROTO\_UDP, etc

Return value : socket descriptor or error(-1)

BIND : int sockfd, const struct sockaddr \*my\_addr, socklen\_t addrlen

Sockfd : socket descriptor which will bind

My\_addr : sockaddr which contain ip and port

Addrlen : length of my\_addr struct

Return value : success(0) or error(-1)

Connect : int sockfd, const struct sockaddr \*serv\_addr, socklen\_t addrlen

Sockfd : socket descriptor which want to connect

Serv\_addr : 서버의 address 정보

Addrlen : serv\_addr의 길이

Return value : success 0 or error -1

Listen : int sockfd, int backlog

Sockfd : socket descriptor which will listen

Backlog : 희망하는 리스닝 큐 사이즈.

Return value : success(0) or error(-1)

Accpet : int sockfd, struct sockaddr \*cliaddr, socklen\_t \*addrlen

Sockfd : socket descriptor of the listening socket with queue

Cliaddr : 접속한 클라이언트의 address 정보가 저장될 struct의 주소

Addrlen : cliaddr의 길이

Return value : socket file descriptor or error -1

8.