

Chapter 1

hosts \Rightarrow endsystems

transmission rate \Rightarrow bandwidth (bits per sec)

↳ communication link is defined

Internet \Rightarrow 수많은 network로 Networking 핵심

Protocol \Rightarrow message 들의 sending & receiving control

Protocol \Rightarrow network entity들 사이에서 보내고 받는다는 메시지들의 format, order 등 정의
message transmission의 수행되는 action 정의

Network

edge : hosts (clients and servers)
Core : interconnected routers and switches

Access Network

↳ DSL (Digital Subscriber line) \rightarrow dedicated

- data는 internet으로, voice는 telephone net으로 감.

↳ Cable network (HFC: hybrid fiber coax) \rightarrow shared

- 하나의 bandwidth가 여러 frequency로 나누어 각 frequency가 하나의 channel로对自己的 content 제공

↳ Ethernet

- network를 구성할 때 datalink layer, physical link layer의 mechanism을 정의하는 technology

↳ wireless

- wireless LAN (WiFi), wide-area wireless access (3G, 4G: LTE)

link transmission rate $R = \text{Link Capacity} = \text{Link bandwidth}$

• Network Core

↳ Packet Switching (sharing)

• 정의: host가 application message를 Packet으로 보내 Packet을 step by step으로 전달

• store-and-forward: 현재 Packet이 router에 다 도착한 이후에 다음 link로 전달됨.

• arrival rate of transmission rate를 초과하는 경우 queuing, loss 발생

↳ queuing: Packet이 transmit 되길 기다림

↳ loss: queuing buffer가 꽉 차면 Packet을 잃어버림.

• routing: source - destination으로 가는 Packet의 경로를 결정하는 것

• forwarding: router로 들어온 Packet을 적절한 output link로 내보내는 것

↳ Circuit switching (dedication)

• 정의: source에서 destination까지의 경로를 dedication하는 것

• 사용되지 않으면 놓고 있으면 됨

• FDM: 할당된 대역폭 나눔

• TDM: 대역폭을 시간단위로 나눔

↳ packet switching과 circuit switching은 비슷 많은 user를 수용.

↳ delay와 bandwidth는 circuit switching이 guarantee.

• Internet Service Provider (ISP)

↳ end system은 ISP 험들을 통해 Internet을 연결

↳ Internet Structure:

• Global ISP, IXP (Internet Exchange Point), regional network, content provider network

• Delay

- ↳ $\text{Delay} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$
- ↳ $d_{\text{proc}} \rightarrow$ check bit error, determine output link, fast
- ↳ $d_{\text{queue}} \rightarrow$ main issue, dynamic delay, output link의 transmission을 기다리는 delay
- ↳ $d_{\text{trans}} \rightarrow$ Packet을 transmission 하는 걸 표기는 시간, L/R (L : Packet length, R : link bandwidth)
- ↳ $d_{\text{prop}} \rightarrow$ Propagation delay, d/s (d : length of physical length, s : Propagation Speed)
- ↳ traffic intensity = $\frac{La}{R}$ (a =average Packet arrival rate)

• Packet loss

- ↳ queueing buffer가 finite capacity를 가지고 있어 loss 생김
- ↳ full queue인 도착한 Packet은 drop (lost)됨
- ↳ lost Packet은 retransmit하거나 인용수 있음.

• throughput

- 용량이 작은 link가 의해서 한계, throughput 결정
- ex) $R_s < R_c \rightarrow R_s$ 가 throughput) → Page 52, 53
- ex) $\min(R_c, R_s, R/10)$

↳ bottleneck link → end to end throughput을 강제하는 link.

• layering

- ↳ 복잡한 System을 다루는데 편리
- ↳ modularization은 maintenance, updating을 쉽게함 (layer간의 transparency를 통해)

message	application	→ FTP, SMTP, HTTP
segment	transport	→ TCP, UDP
datagram	network	→ IP, routing Protocol
frame	link	→ ethernet
	Physical	

• Chapter2 Application Layer

• Network APP

- ↳ 서로 다른 end System에서 작동하는 Program
- ↳ network를 통해 communication 하는 Program
- ↳ network app을 만들 때 network core를 신경쓸 필요 없음 → 개발 속도 빨라짐

• Application Architecture

- ↳ Client-Server
 - ↳ always on permanent IP, data center for scaling
 - ↳ servers communicate.一般都是 connect with dynamic IP, no direct 통신 인터넷
- ↳ Peer-to-Peer(P2P)
 - ↳ always on server 예상
 - ↳ Server, client end system끼리 양방향 통신
 - ↳ self scalability 가능性强 (서버는 service capacity를 증가시킬 때 동시에 service IP)
 - ↳ 간접적으로 연결됨

• Process Communication

- ↳ Process : host 위에서 돌아가는 Process
- ↳ 같은 host 위에서 inter Process communication을 통해 communication
- ↳ 서로 다른 host 위에서는 message 교환을 통해 communication
- ↳ Client Process : communication을 initiate 하는 Process
- ↳ Server Process : contact 요청을 가진다는 Process.
- ↳ P2P application은 client, server Process가 모두 될 수 있음.

• Socket

- ↳ Process는 Socket을 통해 message 주고 받음. 02 70S - 8144
- ↳ sending Process는 receiving Process의 socket에 message를 보내거나,反之, receiving Process의 socket으로 message를 보내는 경우에 Socket의 막대 페더에 있는 transport infrastructure에 의존.

• Addressing Process

- ↳ Identifier를 사용하는 데 identifier는 IP address + Port number 等.

• App layer Protocol

↳ define type of message exchanged, message syntax, message semantic, rules

↳ Open Protocol: defined in RFC (HTTP, SMTP), free

↳ Proprietary Protocol: STP, SNA

• APP이 필요로 하는 transport service는 data integrity, timing, throughput

TCP, UDP의 특성

• Web

↳ web page는 여럿개의 object들로 이루어져 있음.

• HTTP

↳ Hypertext transfer Protocol (application layer Protocol)

↳ client/server 모델 [client: request하고 response를 받고 이를 화면에 표시 (browser)]

[server: 받은 request에 대한 response를 보냄 (web server)]

↳ TCP를 이용 (① client initiate connection, ② server accept connection, ③ message exchange, ④ connection closed)

↳ stateless (각각 요청에 대한 정보를 server가 유지하지 않음)

↳ Non-Persistent HTTP

단점의 RTT: small packet of client to server 같다 대기

• TCP connection을 통해 최대 하나의 object만 보낼 수 있음

client가 읽어야는데 걸친 시간

• 이후 connection이 closed됨.

• multiple object를 download하는 것은 multiple connection을 필요로 함.

• HTTP response time = 1RTT + 1RTT + file transmission time = 2RTT + file transmission time

TCP Connection [HTTP Request and first few bytes of response]

• Object 하나당 2RTT 요구

↳ Persistent HTTP

• 하나의 TCP connection으로 여럿개의 object들을 보낼 수 있음

• Response를 보낸 이후 Server의 Connection은 close하지 않음.

• Open Connection을 통해 계속 이미지 message 전달됨

• Object 하나당 1RTT 필요.

- ↳ ① client: initiate TCP connection to server
- ② server: connection accept
- ③ client: request 보내기
- ④ server: request에 대한 response 보내기
- ⑤ client: connection close
- ⑥ client: response message 받기

결과: object 하나당 1RTT의 과정 빠짐

• Cookies

- ↳ 4가지 구성요소
 - HTTP response message에 있는 cookie header line
 - 다음 HTTP request message에 있는 cookie header line
 - User's host의 유저와는 Cookie file (user의 browser) 관리
 - web site의 back end database

↳ 시나리오 (chap.2 27 Page)

- ↳ authorization, recommendation 등에 사용, Privacy 문제 있음

• Web Caches (Proxy server)

- ↳ Origin server의 관여 없이 Client의 request 만족시킴
- ↳ browser는 모든 HTTP request를 Web Cache로 보냄
- ↳ Web cache에 object가 있는 경우 object return
- ↳ 없는 경우 Web cache는 object를 origin server로 요청해 받아온다 이를 Client에게 return함.
- ↳ Client request의 response time 줄여줌, access link의 traffic 감소시킴

• Conditional GET

- ↳ CacheHit: up to date Version이 기재되고 있으면 object 보내지 않음 (no object transmission delay, lower link utilization)
- ↳ Cache: HTTP request의 cached copy의 date를 보냄
- ↳ Server: date를 보고 확인하면 response에 object를 포함하지 않고 확인 버전이 아니면 object 보냄

• Electronic mail

↳ 3가지 구성요소 (User agent, mail server, SMTP)

↳ User Agent

- mail reader, mail message 구성 편집, 표기
- outgoing, incoming mail message를 server로 전송

↳ Mail Server

- mail box, message queue
- SMTP: mail server 간에 email message를 보내기 위한 Protocol
 - ↳ Client: mail message를 받는 mail server
 - Server: mail message를 받는 mail server

↳ SMTP

- TCP를 사용(reliable transfer를 위한), Port 25
- direct transfer
- Persistent Connection 사용
- message는 7 bit ASCII 문자 형



↳ HTTP와 SMTP의 비교

- HTTP: Pull, 각 object가 its own response message를 encapsulate
- SMTP: Push, multiple object는 multipart message로 함께 보내짐
- 단점: ASCII Command/response Interaction, Status Code

• DNS: Domain Name System

↳ IP address와 name은 mapping

↳ distributed database, hierarchical struc 구조

↳ application layer Protocol

↳ DNS Service

- host name을 IP address로 translation

- host aliasing, mail server aliasing.

• load distribution (같은 서버에 여러 IP가 할당되어 있으면 가장 적합한 것 찾아주) Congestion 막음)

↳ Root DNS Server: Contacted by local name server that cannot resolve name.

↳ TLD DNS Server: responsible for .com, .org, .net ~

↳ Authoritative DNS Server: organization's own DNS Server

↳ Local DNS Name Server

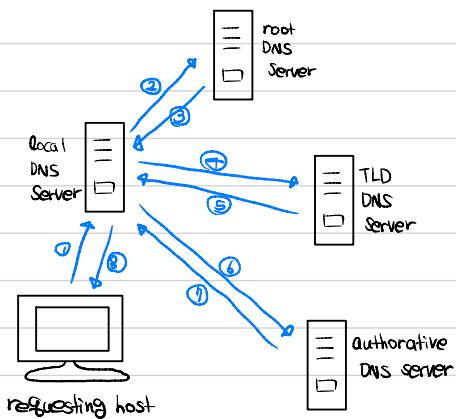
- hierarchical이 엄격하게 독하지 않음

- 대부분이 예전 소유

- 각 ISP가 가지고 있음

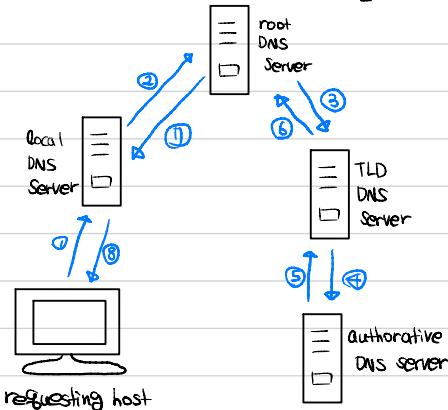
↳ iterative query

- Contacted Server가 Contact 한 후에 Server 정보를 알려줌



↳ recursive query

- 부모는 contacted name server에게 넘김



↳ Caching

- name server가 한번 mapping을 정보를 알게되면 mapping 정보를 일정시간(TTL)동안 caching을 해놓음
- Cache entry가 out-of-date되면 TTL이 만료되기 전까지는 코้ม
↳ host의 IP address가 바뀜.

Pure P2P Architecture

- ↳ No Always on server
- ↳ 임의의 end system끼리 directly communication
- ↳ Peer는 네트워크 연결되고 IP address change
- ↳ file distribution, streaming, VOIP

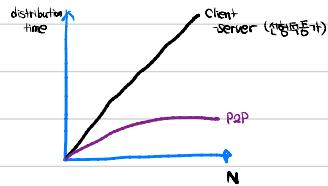
(Ex) file size F, One Server, N Peers

- file distribution time

↳ Client Server: $D_{cs} \geq \max\{NF/us, F/dmin\}$

↳ P2P: $D_{p2p} \geq \max\{F/us, F/dmin, NF/(us + \sum u_i)\}$

각각은 Server가 아닌 Peer라
접근하는 경로 때문



CDN (Content distributed network) → Protocol 아님

- ↳ User 대로 다른 capability를 가지며 있어 이에대한 문제 해결
- ↳ Video의 multiple copies를 multiple geographically sites로 저장
 - enter deep → CDN server를 깊숙히 많은 지역에 넣는 것 (빠르지만 가격 비쌈)
 - bring home → Cluster를 통해 Pop-up에 둠 (느리지만 가격 합)

DASH (Dynamic Adaptive Streaming over HTTP)

Server

- video file을 서버는 rate의 여러 chunk로 포함.
- manifest file: different chunk의 대한 URL 제공

Client

- Periodically Server to client bandwidth 측정
- 이를 통해 적당한 chunk 요청
- Client가 인지, 어떤 encoding rate chunk, 어디로 요청할지 결정 (intelligence)

↳ OTT (Over the top): end device를 넘어서 원하는 많은 multimedia data들을 제공해주는 service.

Transport layer

↳ 서로 다른 host에서 돌아가는 Process를 사이버네트의 logical communication 제공

↳ send Side → APP message를 Segment 단위로 표기 network layer로 보냄. ↳ network layer는 서로 다른 host 사이버네트의 logical communication

↳ receive Side → Segment들을 합쳐 message로 만든이 application layer로 전달.

TCP

↳ reliable, In-order delivery

↳ Congestion control, flow control

↳ Connection set up

↳ No guarantee of delay and bandwidth.

↳ No Message boundary

UDP (streaming multimedia app, DNS, SNMP)

↳ unreliable, un-ordered delivery

↳ No Congestion control and flow control

↳ No Connection set up

↳ No guarantee delay and bandwidth

↳ has message boundary

↳ only focus on transmission

↳ reliability를 위해서는 application 단위로 reliability를 지원함

Multiplexing (Sender)

↳ multiple socket으로 data를 다루고 transport header를 붙임.

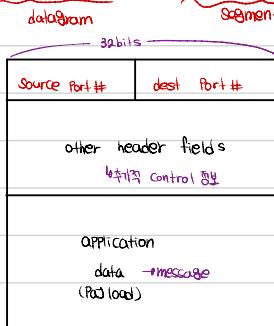
↳ demultiplexing이나 사용

도록 해야함.

Demultiplexing (Receiver)

↳ header info를 이용하여 받은 Segment를 올바른 Socket으로 전달

→ host는 IP address와 Port number를 이용하여 Segment를 올바른 Socket으로 전달



→ 여러 IP address 블로면 datagram.

TCP / UDP Segment format

Connection less demultiplexing (UDP)

- ↳ UDP Socket을 통해서 datagram을 블 때에는 반드시 datagram의 destination IP address와 destination Port number^(Sender) 명시해야 함. (하나의 application마다 socket 하나만 있으므로, source에 관한 정보는 필요 없음)
- ↳ (receiver) host가 UDP segment을 블 때에는 Port #을 보고 그에 해당하는 socket으로 UDP segment 전달
- ↳ 같은 dest. Port #을 가지고 있는 datagram들은 Source IP address / Source Port #이 다르더라도 같은 socket으로 전달

Connection Oriented Demux (TCP)

↳ 하나의 server application이 서버는 host가 access 할 때마다 새로운 socket을 열어줌.

↳ 따라서 이 경우 receiver가 demultiplexing을 할 때 필요한 TCP socket은 Source IP, Port #, destination IP, Port #의 정보를 가지고 있어야 함.

UDP

32 bits	
Source Port #	dest. Port #
length	checksum → error check or no.
↳ header를 포함한 UDP segment의 길이 (byte 수)	
Application data → message (Payload)	

UDP Segment format

사용하는 Delay 풀어짐 (No connection)

간단함, header size가 작음, congestion control 없어 빠름.

UDP check sum

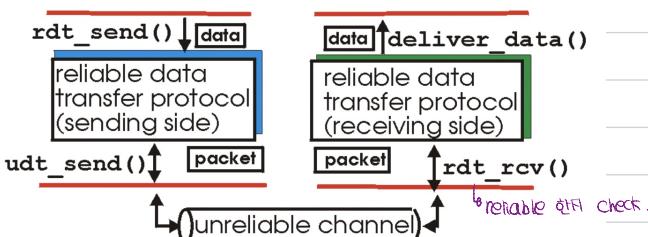
Sender → Segment contents (header field 포함)을 16 bits integer의 sequence로 정렬해

각 디렉트리의 블록을 합하여 check sum field에 저장

Receiver → 각 segment의 checksum을 계산한 check sum field의 값과 비교.

Reliable data transfer

↳ Application layer은 below part는 reliable channel이라고 생각하지만 transport layer에서 보면 below part는 unreliable임.



no bit errors, no packet loss \rightarrow
rdt 1.0, rdt 2.0, 2.1, 2.2, 3.0
↳ bit error ↳ bit error and loss.

2. \rightsquigarrow Stop and Wait \rightarrow Sender는 Packet을 보내보고 receiver의 response를 기다림.

Acks \rightarrow receiver가 Sender에게 Packet을 잘 받았다고 알려주는 신호

NACKs \rightarrow receiver가 Sender에게 Packet이 잘 못 되었라고 알려주는 신호.

• ACK, NAK 신호가 잘못된 경우

↳ Sender는 다시 보내는 데 duplicate이 발생할 수 있음.

↳ 이를 처리하기 위해 Sender는 각 Packet의 sequence number를 블임.

↳ receiver는 duplicate Packet을 버림.

• ACK만 이용해서 NAK 신호가능

↳ NAK 신호 대신 receiver는 이전에 잘 받은 Packet의 ACK 신호를 다시 보냄 (ACK 신호에 sequence 번호가 들어가 있음)

↳ 풍복된 ACK는 Sender가 들여온 것은 NAK를 의미.

rdt 3.0

↳ Sender는 합리적인 시간만큼 ACK 신호를 기다림.

↳ 이 시간 안에 ACK가 오지 않으면 다시 보냄

↳ 만약 절 전달 됨지만 delay에 의해 timeout이 발생한 경우 retransmitter는 duplicate이 발생하지만 seq#에 의해 처리됨 (discard)

P Retiming \rightarrow Stop-and-wait 하지 않고 ack이 돌아오기 전에 \checkmark Packet을 계속 보냄
↳ performance가 좋아짐

↳ Sender와 receiver의 buffering.

↳ Go-Back-N, Selective Repeat이 있음.

Go-Back-N \rightarrow 초기 기도 discard 한 경우나 성과 duplication이 많이 발생

↳ N개의 unacked Packet을 Pipeline에 가지고 있을 수 있음

↳ receiver는 Cumulative ACK을 보냄

↳ Sender는 oldest unacked Packet의 대량 timer를 관리함 (timer 하나만 필요)

↳ timer가 expire되면 모든 unacked Packet을 다시 보냄.

↳ receiver는 in-order seq#만 봄, out order 된 seq#은 discard 된다. highest in-order seq#은 ACK로 보냄.

ex) 1 2 3 4 6를 받았을 때 discard되는 ack #을 보냄.
5가 없음

Selective Repeat \rightarrow timeout 있음 (SSPage)

- ↳ N개의 unacked Packet을 Pipeline에 가지고 있을 수 있음
- ↳ receiver는 각 Packet에 대한 ack을 보냄
(중복전송을 막기위해, 하나하나의 Packet마다 timer 필요)
- ↳ timer가 expire되면 해당 Packet만 retransmit 하며 restart timer.

TCP

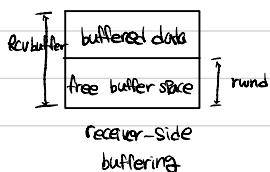
- ↳ Congestion, flow control는 window size set与时 Pipelineing.
- ↳ sequence number: segment의 payload에 포함되는 data의 첫번째 byte의 번호
- ↳ acknowledgement: 상대쪽이 sequence 번호로 보내주길 기대하는 번호
 - : cumulative ack.
- ↳ RTT를 잘 관리해야 함.
 - ↳ Sample RTT: ACK이 올때까지 걸린 시간 (transmission time)
 - ↳ estimated RTT: $(1-\alpha) \cdot \text{Estimated RTT} + \alpha \cdot \text{Sample RTT}$
 - ↳ Sample RTT는 최근 253 weighted moving average 사용
- ↳ retransmission timeout과 duplicate acks의 유통 발생.

TCP fast retransmit

- ↳ time out은 상대쪽으로 너무 가깝게 lost segment를 duplicate ack으로 판별
- ↳ 만약 sender가 같은 data를 다시 보냈을 때의 duplicate ack를 발견하면 unacked segment를 다시 보냄 (기본 것은 seq# 외)

Flow Control

- ↳ sender가 receiver buffer의 양을 넘겨 transmit하지 않도록 조절하는 것
- ↳ receiver의 rwnd는 TCP header에 넣어 sender에게 보냄. (sender는 이를 알고 있음)
 - ↳ free buffer space.
- ↳ receiver buffer를 넘기지 않는 것을 guarantee.



Connection establish

- 2-way hand shake: variable delays, retransmitted message due to message loss, message reordering
Can't see other side issue → 3-way handshake 사용.

Connection close

- Client & Server는 (other side) Connection을 각각 close함.
 - ↳ TCP Segment 2 FIN bit = 1로 먼저 보내남.
 - 받은 FIN에 대한 ACK 신호를 보냄.

Congestion → Network가 다수의 힘들정도로 많은 소스가 많은 데이터를 빠르게 보낼 때 발생

- ↳ Packet losses → router의 buffer가 꽉 차울 때
- ↳ Long delays → router buffer에서의 queuing에 의해.
 - known loss (good Put은 $\frac{R}{2}$ 임, Packet loss로 대한 재전송 일어나므로, 보내는 양만 줄보다 품 많음, T_u 사용 때문에)
 - , duplicate (time out Prematurely) (goodPut이 $\frac{R}{2}$ 보다 적음, 동일한 Packet이 두번 들어온 경우 goodPut으로 볼 수 없어도)

TCP Congestion Control

- Additive Increase Multiplicative Decrease (AIMD)
 - ↳ TCP sender Congestion Window Size
- Additive Increase (Cwnd를 1 MSS씩 증가, every RTT마다, loss가 발생할 때까지)
- Multiplicative Decrease (loss가 발생하면 Cwnd를 반으로 감소시킴)
- sender's transmission \leq Last Byte Sent - Last Byte Acked \leq Cwnd를 제한함.

- ↳ TCP Slow Start: Initially Cwnd=1MSS, loss가 일어날 때까지 0.5 RTT마다 Cwnd 증가

고정: threshold 기준에는 window가 1부터 시작해 exponentially grow되다가 이후에는 linearly 증가

- loss 발견, 다음에 따라 TCP Reno, TCP Tahoe로 나뉨.
- ↳ timeout으로 loss 발견 (threshold Cwnd를 증가하고 Cwnd를 $\frac{1}{2}C + \text{exponentially threshold} \geq 1$)
 - ↳ TCP Reno: 3duplicate Ack로 loss 발견 (Cwnd를 반으로 증가하고 그 이후부터 Cwnd를 linearly grow 시킴)

- ↳ TCP Tahoe: timeout이나 3duplicate Ack로 loss 발견, loss 발견 시 threshold를 Cwnd의 $\frac{1}{2}$ 로 증가하고 Cwnd를 1부터 exponential하게 증가시킨다가 threshold에 도달하면 linear하게 증가

TCP Fairness → 만약 k 개의 TCP session이 같은 link를 공유한다면 각 세션은 $\frac{1}{k}$ 의 average bandwidth를 가져야 함 (bandwidth R을 가진)

Explicit Congestion Notification (ECN)

- ↳ Congenital network assisted congestion control을 하는 것
- ↳ IP header의 2bit가 network router에 의해 marking이 됨 (congestion을 알기 위한 bit)
↳ receiving host는 congestion을 알게됨
- ↳ receiver는 ACK 신호에 ECE bit를 Set해 sender에게 congestion 알림.
- ↳ ECE bit가 1로 오면 sender는 congestion을 알게되고 window size 줄임.

Chapter 4. Network Layer

Network layer는 모든 host와 router의 역할.

Forwarding → router의 input으로 들어온 packets을 어떤 특정한 output link로 보내줄지 결정하는 것

Routing → source와 destination 사이의 packets의 경로를 결정하는 것

Data Plane

- forwarding에 관련
- local, per-router function

Control Plane

- routing에 관련
- network wide logic
- traditional routing algorithm, Software Defined Networking (SDN)
 - ↳ Control Plane의 router側
 - ↳ Control Plane의 remote controller

Per-router Control Plane

- ↳ traditional routing algorithm
- ↳ 각 router가 data Plane과 Control Plane 모두 가지고 있음
- ↳ 각 router가 심호작용하여 경로를 결정
- ↳ 하드웨어 문제로 생긴 경우 다른 경로를 고는 데 시간 오래 걸림.

Logically Centralized Control Plane

- ↳ SDN (Software Defined Networking)
- ↳ 특정 영역을 지배하는 Controller에서 routing table router는 forwarding 만 함
- ↳ router에는 data Plane만 있고 Control Plane은 router로 부터 분리

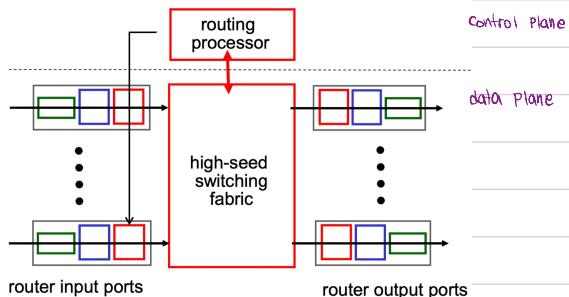
Router Architecture

↳ routing, management Control Plane

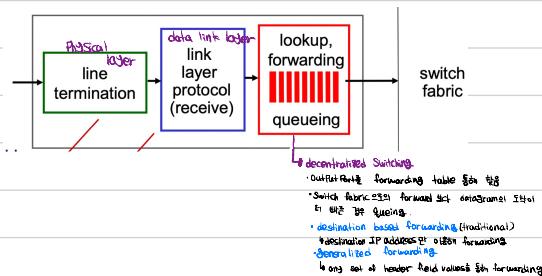
- Software, 비교적 느림

↳ forwarding data Plane

- Hardware, 비교적 빠름



Input Port functions



• Destination based forwarding

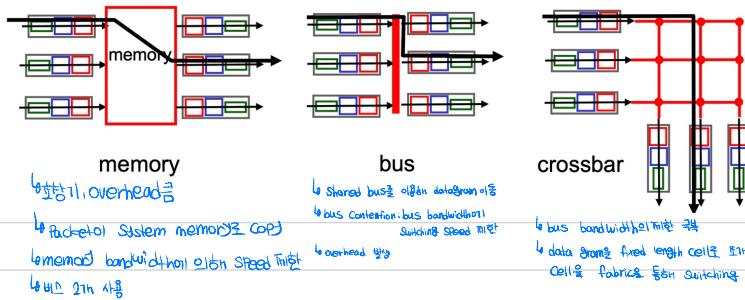
- ↳ Longest Prefix matching: Prefix을 가장 길게 matching되는 link로 넘김.

↳ 과거에는 bit by bit로 매칭(속도느림)

↳ 속도 위해 TCAM (Ternary content addressable memory) 사용 → address table size 1GB 관계 없이 1 clock cycle 걸림.

• Switching fabrics

- Packet을 input buffer에서 적절한 output buffer로 전달



• Input Port Queueing

- Switching 속도가 line으로 들어오는 속도 따라잡지 못하면 queueing (buffer)

→ queuing delay와 loss 발생

- Head of Line (HOL) blocking → queue 맨 앞의 data gram이 queue 안의 다른 data gram을 moving forward하는 것을 막음 (Page 22)

• Output Port

- fabric으로부터 data gram이 들어오는 속도가 transmission rate보다 빠르면 buffering 필요 + queuing delay, loss 발생 가능
- queued data gram 중 어떤 걸 transmit 할 것인지 선택 (scheduling discipline)

• Scheduling mechanisms

↳ link를 통해 보낼 다음 Packet을 선택

↳ FIFO, Priority, Round Robin, Weighted Fair Queuing (Generalized round robin)

↳ Discard Policy → full queue인 경우 어떤 것을 drop 할지

- tail drop (노ック아웃 Packet), Priority (우선순위에 따라), random

• IP datagram

- Overhead = 20 bytes of TCP + 20 bytes of IP + app layer overhead

• IP fragmentation, reassembly

↳ Network layer는 MTU (Max. transfer size)를 가지고 있음

↳ 큰 IP datagram은 여러개의 datagram으로 fragment되고 final destination에서 reassemble 됨.

Ex) 4000 byte data gram, 1500 bytes MTU

→ 3개의 datagram으로 나누기.

① length = 1500 bytes = 1480 + 20 (overhead in datagram), offset = 0

② length = 1500 bytes = 1480 + 20 (overhead in datagram), offset = 1480 / 8 = 185

③ length = 1040 bytes = 1040, offset = 2960 / 8 = 370

↑ 1st datagram overhead가 예상되는 경우
⇒ 3개의 datagram으로 나뉨 (4000 = 3880 + 20)

↳ offset이 185로

• IP addressing

↳ IP address: 32 bit identifier for host, router interface

↳ Interface: host/router 간 physical link 사이의 연결

• router는 일반적으로 여러개의 interface 가지고 있음

• host는 일반적으로 한개 또는 2개의 interface (wired, wireless)

↳ IP address는 각 interface와 관계 있음.

• Subnets

↳ router 없이 물리적으로 통신할 수 있는 local area network (router를 뜯어놓을 때 생기는 isolated network)

↳ IP address [Subnet Part] → high order bit
[host Part] → low order bit

↳ CIDR (Classless InterDomain Routing)

• address format: a.b.c.d/x

↳ Subnet Part가 몇비트인지를 나타냄.

• address의 subnet portion이 20bit를 dynamically 할당 가능.

• DHCP

↳ Dynamic Host Configuration Protocol

↳ Server로부터 address를 dynamically 받음.

↳ DHCP는 IP address 뿐만 아니라 first hop router의 address, DNS Server의 IP address와 name, network mask를 반환(통신을 빨리 할 수 있도록)

↳ $\xrightarrow{\text{(client)}} \text{DHCP discover} \rightarrow \xrightarrow{\text{(server)}} \text{DHCP offer} \rightarrow \xrightarrow{\text{(client)}} \text{DHCP request} \rightarrow \xrightarrow{\text{(server)}} \text{DHCP Ack}$ (44 Page)

• ICANN

↳ Internet Corporation for Assigned Names and Numbers (Protocol 이름, 호스트 이름 등)

↳ address 할당, DNS 관리, domain name 할당.

• NAT (Network Address translation)

↳ local network 안에서만 각각의 host들이 unique한 IP address를 가지고 있음

↳ 하지만 local network를 떠나면 서버들은 Port의 하나의 NAT IP address로 나누어 전달.

↳ 하나의 IP address로 여러 host 관리가 가능.

↳ 구현

• Outgoing datagram: data frame의 (Source IP address, Port#)은 (NAT IP address, New Port#)으로 replace

• NAT translation table의 (Source IP address, Port#) (NAT IP address, New Port#)의 translation pair 저장

• incoming datagram: datagram dest field의 (NAT IP address, Port#)은 이미 대응되는 (Source IP address, Port #)으로 replace

↳ 16bit port number field로 1개의 IP를 이용해 65536개의 host 관리 가능.

↳ 하지만 NAT는 (Layer 3 transparency)에 위배.

• IPv6

↳ 64bytes fixed length header, No fragmentation allowed

↳ tunneling: IPv6 data frame이 IPv4 datagram의 payload에 담겨 IPv4 router 통과

• Generalized Forwarding and SDN

↳ 각 router는 logically centralized routing controller의 의해 계산되고 분배된 flow table을 담고 있음.

↳ open flow: 네트워크를 유연성 있게 Control Plane과 data Plane에서 뜯어내 구현하는 것

↳ router의 있는 flow table의 router의 match+action rule을 통의

header field를 통해 \Rightarrow $\left\{ \begin{array}{l} \text{to drop, forward, modify, send to controller} \\ \text{value matching} \end{array} \right.$