**32776 Midterm Exam (Friday, May 01, 2020) / Reference not described means from MNI Lab**

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** **[03/06, 03/07, 03/09]** |

***01-01: INTRA- AND INTERDOMAIN ROUTING***

|  |  |
| --- | --- |
| **Intra**domain routing | 어떤 autonomous system **내부**에서의 라우팅 |
| **Inter**domain routing | Autonomous system **간**의 라우팅 |

|  |  |
| --- | --- |
| Autonomous system | Routing Protocols |
|  |  |

***01-02: DISTANCE VECTOR ROUTING***

**<Distance Vector Routing Table>**

|  |  |
| --- | --- |
|  | **회색:**  각 node로 이동하는 데 사용되는 **최적(최소)의 총 비용**  **분홍색:**  해당 node로 이동하는 비용을 **최소화시키기 위한 다음 node** |
|  | 각 node에 대한 distance table은 **해당 node 및 직접 연결된 node의 경우 비용**을 표시하고(해당 node는 0), **그렇지 않은 경우 무한대**로 표시한다. |

* 각 node는 **직접 연결된 다른 node와 라우팅 테이블을 공유**하여 **주기적으로/변동이 있을 때 업데이트**한다.

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<Distance Vector Routing에서의 업데이트>**

|  |  |
| --- | --- |
|  | **A’s Modified Table:**  **C의 Routing Table의 cost 값**에 A’s **Old Table의 A->C의 cost 값**을 더한다.  **모두 C**로 한다.  **A’s New Table:**  각 node로의 cost는 **min(기존 cost, A’s Modified Table의 해당 cost)**의 값으로 한다.  **업데이트되는 부분**을 **Modified Table의 해당 node**로 한다. |
|  | **<Two-node instability>**  **After failure:** A의 라우팅 테이블의 **node X에 대한 항목이 무한대로 업데이트**된다.  **After A receives update from B:** A의 라우팅 테이블의 X에 대한 항목은 **(old B->X cost)+(A->B cost) = 6+4 = 10**으로 업데이트된다.  **After B Receives update from A:** B의 라우팅 테이블의 X에 대한 항목은 **(old A->X cost)+(B->A cost) = 10+4 = 14**로 업데이트된다.  **Finally:** 이것이 반복되어 **결국 무한대**가 된다. |
|  | **<Three-node instability>**  **After A sends … :** A가 B에 route를 전송하였으므로 A, B의 라우팅 테이블의 node X에 대한 항목은 모두 무한대가 된다. (C는 전송 실패)  **After C sends the route to B:** B의 라우팅 테이블의 node X에 대한 항목은 **(old C->X cost)+(B->C cost) = 5+3 = 8로 업데이트**된다.  **After B sends the route to A:** A의 라우팅 테이블의 node X에 대한 항목은 **(old B->X cost)+(A->B cost) = 8+4 = 12로 업데이트**된다. |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

***01-03. RIP (Routing Information Protocol)***

**Routing Information Protocol (RIP):** autonomous system에서 사용되는 **intradomain 라우팅** 프로토콜로, **distance vector routing에 기반**한 매우 간단한 프로토콜이다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 각 Router의 Table에서…   |  |  | | --- | --- | | Dest. | 네트워크로 연결된 **각각의 다른 node의 IP 주소** | | Hop | Dest. Node로 최적의 경로로 이동하기 위해서 **거쳐야 하는 Hop의 수** | | Next | Dest. Node로 최적의 경로로 이동하기 위한 **다음 지점의 IP 주소** | |

**<RIP 메시지 형식>**

|  |  |
| --- | --- |
| **Command, Version, Reserved** 영역이 맨 위에 있고, 그 아래에 **Family, Network Address, Distance**가 반복된다. | |
| Request for some | Request for all |
|  | Request for some에서와 달리 **Network address 부분도 0으로 표시**된다. |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<RIP 예제 1>**

|  |
| --- |
| 왼쪽 그림은 **router R1에서 router R2로의 업데이트 메시지**이며, Source는 **130.10.0.2**이다.   * R1은 라우터 R2로부터 네트워크 **195.2.4.0, 195.2.5.0, 195.2.6.0에 대한 정보**를 얻었다. * R1이 R2에 update message를 전송할 때, **R2에서의 혼란을 방지하기 위해** 이 3개의 네트워크에 대한 **hop count의 실제 값은 16**으로 바뀐다. * R2는 R1(130.10.0.2)로부터의 **RIP message를 전송하는 IP datagram의 source 주소**를 사용한다. |

**<RIP 예제 2>**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Routing table에 20개의 엔트리가 있으며, 5개의 route에 대해서는 200초 동안 정보를 받지 않는다. 이때 얼마나 많은 타이머가 동작하는가?**   |  |  | | --- | --- | | Periodic Timer | **1** | | Expiration Timer | **15** = 20 – 5 | | Garbage Collection Timer | **5** | |

**<RIP Version 2 메시지 형식>**

|  |  |
| --- | --- |
| RIP Version 2 Format | Authentication |
|  |  |

* RIP은 **well-known port 520**에서 **UDP 서비스**를 이용한다.

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

***01-04. Link State Routing***

**Link State Routing:** 도메인에 있는 **각 node가 도메인의 전체 topology(연결 방식)를 알고 있다**면, node는 **Dijkstra’s Algorithm**을 사용하여 Routing Table을 생성할 수 있다.

**<Link State Routing의 개념도>**

|  |
| --- |
|  |

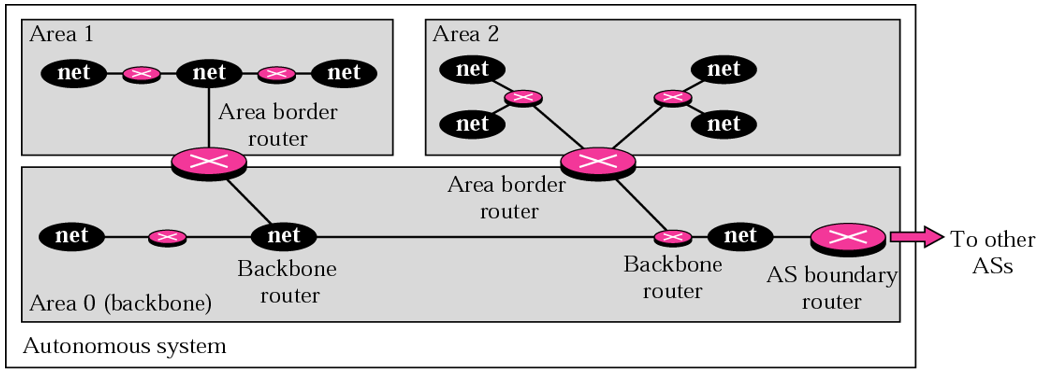
**<Dijkstra Algorithm>**

|  |
| --- |
| 1. **root node를 tentative list에 추가**한다. 2. **Tentative list가 비어 있게 될 때까지** 다음을 반복한다.    1. Tentative list의 각 node에 대하여, **shortest path에 해당하는 node를 permanent list**에 넣는다.    2. 마지막으로 이동된 node의 **아직 프로세싱되지 않은 이웃 node를 tentative list**에 넣는다.  * 이웃한 node의 **누적된 cost가 더 크다면, 새로운 cost로 바꾼다.** |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

***01-05. OSPF (Open Shortest Path First)***

**<Areas in an autonomous system>**



**<Types of links>**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Link Identification | Link Data |
| **Point-to-Point** link |  | **이웃 router**의 주소 | **Interface number** |
| **Transient** link |  | **Designated router**의 주소 | **라우터**의 주소 |
| **Stub** link |  | **네트워크** 주소 | **Network mask** |
| **Virtual** link |  | **이웃 router**의 주소 | **라우터**의 주소 |

**<Autonomous System and its graphical representation in OSPF>**

|  |  |
| --- | --- |
| Autonomous System | Graphical Representation **in OSPF** |
|  |  |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<OSPF 패킷의 종류>**

|  |  |
| --- | --- |
| Hello |  |
| Database description |  |
| Link state request |  |
| Link state update | **Router** link, **Network** link, Summary link to **network**, Summary link to **AS boundary router**, **External** link  **<Link state update packet>** |
| Link state acknowledgment |  |

**<OSPF common header and LSA (Link State Advertisement) general header>**

|  |  |
| --- | --- |
| OSPF common header | LSA general header |
|  |  |

**<Router link and Router link LSA>**

|  |  |
| --- | --- |
| Router link | Router link LSA |
|  |  |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<Example 3>**

|  |
| --- |
| **Router 10.24.7.9에 의해 전송된 router link LSA는?**  이 라우터는 **type 1 (point-to-point) 링크를 2개, type 3 (stub network)를 1개** 가지고 있다. 정답은 오른쪽과 같다. |

**<Network link and its advertisement format>**

|  |  |
| --- | --- |
| Network link | Network link advertisement format |
|  |  |

**<Example 4>**

|  |
| --- |
| **오른쪽 그림에 대한 network link LSA는?**  정답은 아래 그림과 같다. |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<Example 5 and Example 6>**

|  |
| --- |
| **<Example 5>**  **오른쪽 그림에서 어떤 라우터가 router link LSA를 보내는가?**  정답:  모든 라우터가 router link LSA를 보낸다.   * R1은 **N1, N2**의 2개의 link * R2는 **N1**의 1개의 link * R3은 **N2, N3**의 2개의 link   **<Example 6>**  **위 그림에서 어떤 라우터가 network link LSA를 보내는가?**  정답:  모든 네트워크가 network link를 광고해야 한다.   * N1에 대한 광고는 **R1**에 의해 이루어지며, **연결된 라우터가 R1뿐**이므로 R1은 자동으로 designated router가 된다. * N2에 대한 광고는 **R1, R2 또는 R3**에 의해 이루어진다. **(이들 중 designated router로 선택된 라우터)** * N3에 대한 광고는 **R3**에 의해 이루어지며, **연결된 라우터가 R3**뿐이므로 R3은 자동으로 designated router가 된다. |

**<OSPF Packet Museum>**

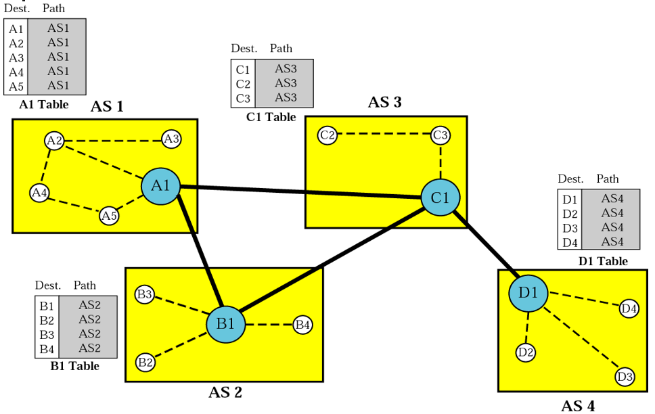
|  |  |
| --- | --- |
| Summary link to network | Summary link to network LSA |
|  |  |
| Summary link to AS boundary router | Summary link to AS boundary router LSA |
|  |  |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<OSPF Packet Museum>**

|  |  |
| --- | --- |
| External link | External link LSA |
|  |  |
| Hello packet | Database description packet |
|  |  |
| Link state request packet | Link state acknowledgment packet |
|  |  |

* OSPF 패킷은 **IP datagram으로 캡슐화**된다.

***01-06. Path Vector Routing***

**Path Vector Routing:** distance vector routing과 비슷하지만, **각 AS에 speaker node**가 있으며, **speaker node는 routing table을 생성**하고 그것을 **이웃한 AS의 speaker node에 전송**한다.

|  |
| --- |
| **<각 autonomous system의 stabilized table>** |

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

***01-07. Border Gateway Protocol (BGP)***

**Border Gateway Protocol (BGP):** **path vector routing**을 이용한 **interdomain routing** protocol

|  |  |
| --- | --- |
| Internal and external BGP sessions | Types of BGP messages |
|  |  |

**<BGP packet>**

|  |  |
| --- | --- |
| BGP packet header |  |
| Open message |  |
| Update message |  |

* BGP는 **classless addressing과 CIDR (classless inter-domain routing)**을 지원한다.

|  |  |
| --- | --- |
| **01** | Week 09. **Routing** |

**<BGP packet>**

|  |  |
| --- | --- |
| Keepalive message | Notification message |
|  |  |

**<BGP의 오류 코드>**

|  |  |  |
| --- | --- | --- |
| 코드 | Error Code 설명 | Error Subcode 설명 |
| **1** | **Message header** error | 3개의 서로 다른 subcode   * Synchronization problem (1) * Bad message length (2) * Bad message type (3) |
| **2** | **Open message** error | 6개의 서로 다른 subcode   * Unsupported version number (1) * Bad peer AS (2) * Bad BGP identifier (3) * Unsupported optional parameter (4) * Authentication failure (5) * Unacceptable hold time (6) |
| **3** | **Update message** error | 11개의 서로 다른 subcode   * Malformed attribute list (1) * **[**Unrecognized (2) / Missing (3)**]** well-known attribute * Attribute **[**flag (4) / length (5)**]** error * Invalid **[**origin (6) / next hop (8)**]** attribute * AS routing loop (7) * Optional attribute error (9) * Invalid network field (10) * Malformed AS\_PATH (11) |
| **4** | Hold timer expired | Subcode 없음 |
| **5** | Finite state machine error | Procedural error를 정의하므로 Subcode 없음 |
| **6** | Cease | Subcode 없음 |

* BGP는 **port 179에서 TCP 서비스를 이용**한다.

|  |  |
| --- | --- |
| **02** | Week 10. **Transport Layer Overview** **[03/10]** |

***02-01. Network Layer vs. Transport Layer***

|  |
| --- |
| Network Layer는 Internet까지 다루지만, **Transport Layer는 프로세스까지** 다룬다. |

***02-02. Port Number***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * Daytime Client의 Port number는 52000, Daytime Server의 Port number는 13이다. * Data의 **header**에는 **Source Port number와 Destination Port number**가 있다. * Daytime Client는 Port number 52000인 Daytime Server에 데이터를 전송한다.   **<Port number vs. IP address>**   |  |  | | --- | --- | | Destination **Port number** | **프로세스를 선택**한다. | | Destination **IP address** | **서버를 선택**한다. | |

**<ICANN ranges>**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ICANN range:** **Port number의 범위**를 well-known, Registered, Dynamic or private로 나눈 것   |  |  | | --- | --- | | 0~1023 | Well-known | | 1024~49151 | Registered | | 49152~65535 | Dynamic or private | |

|  |  |
| --- | --- |
| **02** | Week 10. **Transport Layer Overview** |

**<Example 1>**

|  |
| --- |
| UNIX에서는 well-known port가 /etc/services 파일에 저장되어 있다.   * 이 파일의 각 line은 server의 이름과 well-known port 번호를 저장하고 있다. * 다음 그림은 TFTP에 대한 port를 보여 준다.   + **TFTP는 UDP 또는 TCP에서 port 69를 사용**할 수 있다. * **SNMP는 161, 162의 서로 다른 목적의 2개의 port 번호**를 가지고 있다. |

**<Socket address>**

|  |
| --- |
| Socket address는 **IP address와 Port number가 합쳐져서** 만들어진다. |

***02-03. Encapsulation and Decapsulation***

**<Encapsulation and Decapsulation>**

|  |
| --- |
| * Application Layer에서 Transport Layer로 메시지를 전송할 때 **메시지에 header가 추가되어 Header + Packet(=Message, Payroll)로 캡슐화**된다. * **Logical channel**을 통해 다른 node의 Transport Layer로 전송된다. * Transport Layer에서 Application Layer로 **Header + Packet(=Message, Payroll)을 전송할 때 decapsulation되어서 원래 메시지가 복구**된다. |

***02-04. Flow and Error Control***

**<Pushing and Pulling>**

|  |  |
| --- | --- |
| Pushing | Pulling |
|  |  |

|  |  |
| --- | --- |
| **02** | Week 10. **Transport Layer Overview** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **<FLOW control at the TRANSPORT layer>**     |  |  | | --- | --- | | Pushing  **(Flow control)** | 송신 측의 Application Layer와 Transport Layer  송신 측과 수신 측의 Transport Layer | | Pulling **(request)** | 수신 측의 Transport Layer와 Application Layer |   **Buffer가 가득 차거나 비어 있을 때** Consumer과 producer와 통신해야 한다.   * Consumer와 producer가 **1개의 슬롯이 있는 버퍼**를 사용하면 통신이 쉬워진다. * **전송 측의 transport layer**에 있는 이 슬롯이 비어 있으면 그 transport layer는 **application layer**에 **다음 chunk에 대한 note**를 보낸다. * **수신 측의 transport layer**에 있는 이 슬롯이 비어 있으면 그 transport layer는 **전송 측의 transport layer**에 **다음 패킷에 대한 acknowledgment**를 보낸다. |
| **<ERROR control at the TRANSPORT layer>**     * Sequence number는 **으로 나눈 나머지**이다. (m은 sequence number 필드의 비트 수) |

**<Sliding window in CIRCULAR and LINEAR format>**

|  |  |
| --- | --- |
| **CIRCULAR** format | **LINEAR** format |
|  |  |

|  |  |
| --- | --- |
| **02** | Week 10. **Transport Layer Overview** |

***02-05. Multiplexing and Demultiplexing***

|  |
| --- |
| **<Multiplexing and Demultiplexing>**  **Multiplexing:** application layer에서 여러 개의 **페이지를 메시지로 만들어서 합쳐서 패킷으로 변환**시키는 것  **Demultiplexing:** Transport layer에서 **패킷을 합쳐서 메시지로 만들고, 그 메시지를 페이지로 변환**시키는 것   * 클라이언트에서 여러 개의 패킷을 서버로 보낼 때 **각각 서로 다른 demultiplexer로 전송**된다. |

***02-06. Connectionless and Connection-oriented Services***

|  |  |
| --- | --- |
| Connectionless service | Client에서 Message 0, 1, 2를 전달했는데 **Packet 1이 Packet 2보다 먼저 도착**하여 Message 2는 순서에 어긋나게 도착하는 것과 같은 유형의 오류 발생 |
| Connection-oriented service | Client와 Server가 **connection을 형성한 후 메시지를 전송**한다.  따라서 Packet 2가 도착한 후 전송되는 **Packet 1의 메시지인 Message1은 window에 hold**된다. |

|  |  |
| --- | --- |
| **02** | Week 10. **Transport Layer Overview** |

**<Connectionless and connection-oriented services as FSMs>**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** **[03/12]** |

***03-01. TCP Services***

**<Well-known ports used by TCP>**

|  |  |  |
| --- | --- | --- |
| Port | Protocol | 설명 |
| 7 | Echo | 수신된 **datagram을 sender에게 재전송**한다. |
| 9 | Discard | 수신된 **어떤 datagram이라도 기각**한다. |
| 11 | Users | Active users |
| 13 | Daytime | **날짜와 시간** 반환 |
| 17 | Quote | **Day의 quote**를 반환 |
| 19 | Chargen | **Character의 string**을 반환 |
| 20 | FTP, Data | 파일 전송 프로토콜 (**data** connection) |
| 21 | FTP, Control | 파일 전송 프로토콜 (**control** connection) |
| 23 | TELNET | Terminal Network |
| 25 | SMTP | 간단한 메일 전송 프로토콜 |
| 53 | DNS | Domain Name Server |
| 67 | BOOTP | Bootstrap Protocol |
| 79 | Finger | Finger |
| 80 | HTTP | HyperText Transfer Protocol |
| 111 | RPC | Remote Procedure Call |

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

**<Example 1>**

|  |
| --- |
| Grep 유틸리티를 통해 목표한 application에 대응되는 line을 추출할 수 있다. 아래는 FTP에 대한 포트 번호이다. |

**<Stream delivery>**

|  |
| --- |
| Sending process는 **TCP 프로토콜을 통해 Receiving process에 byte stream을 전송**한다. |

***03-02. TCP Features***

|  |
| --- |
| 각 connection에서 전송되는 데이터의 **각 바이트에는 TCP에 의해 숫자가 매겨진다**.   * 넘버링은 **랜덤하게 생성된 특정 값**에서 시작된다. |

**<Example 2>**

|  |
| --- |
| **TCP connection이 5000바이트의 파일을 전송하고 첫 번째 바이트가 10001로 넘버링되었다고 할 때, 파일이 5개의 segment로 쪼개지고 각 segment가 1000바이트이면 sequence number는 어떻게 되는가?**  **[정답]** |

|  |  |
| --- | --- |
| 필드 | 나타내는 것 |
| segment에서 **sequence number** | 해당 segment에 포함된 **첫 번째 데이터의 number** |
| segment에서 **acknowledgment** | **수신할 것으로 예상되는 다음 byte**의 number   * Acknowledgment number는 **cumulative하다.** |

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

***03-03. Segment***

**Segment:** **TCP에서의 패킷**

|  |  |  |
| --- | --- | --- |
| TCP Segment format | Control field | |
|  | URG | **Urgent pointer** is valid |
| ACK | **Acknowledgment** is valid |
| PSH | Request for **push** |
| RST | **Reset** the connection |
| SYN | Synchronize **sequence numbers** |
| FIN | **Terminate** the connection |

**<Description of flags in the control field>**

|  |  |
| --- | --- |
| URG | **Urgent pointer field**의 값이 유효하다. |
| ACK | **Acknowledgment field**의 값이 유효하다. |
| PSH | Push the data |
| RST | **Connection이 reset**되어야 한다. |
| SYN | 연결 중일 때 **sequence number를 synchronize**한다. |
| FIN | Connection을 **종료**한다. |

**<Pseudoheader added to the TCP diagram>**

|  |
| --- |
| TCP에서 **checksum은 반드시 필요하다.** |

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

***03-04. TCP connection***

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **<TCP connection and Data transfer>**   |  |  | | --- | --- | | **TCP connection: Three-way handshaking**을 이용하여 이루어진다. | **Data transfer** | |  |  |   **<TCP에서의 segment>**   |  |  | | --- | --- | | **SYN** segment | Data를 이동시킬 수 없지만, **1개의 sequence number를 사용**한다. | | **SYN+ACK** segment | | **ACK** segment | Data를 이동시키지 않으면 sequence number를 **사용하지 않는다.** | | **FIN** segment | Data를 이동시키지 않으면 sequence number를 **사용한다.** | | **FIN+ACK** segment |   **<Half-close>**   1. Client가 **FIN 패킷을 Server에 전송**하고, 이에 대해 Server는 **ACK 패킷을 Client에게 전송**한다. 2. **Server는 Client에 data segment를 전송**하고, Client는 이에 대한 ACK을 Server에 전송한다. 3. **Server는 Client에 FIN 패킷을 전송**하고, Client는 이에 대한 ACK 패킷을 Server에 전송하여 통신을 종료한다. |

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

***03-05. State Transition Diagram***

**<State for TCP>**

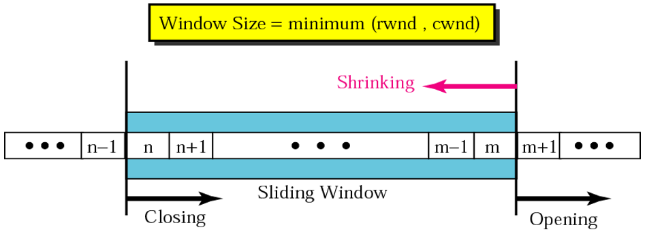
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **State** | Sent | Received | Waiting for | Other |
| **CLOSED** |  |  |  | 연결 없음 |
| **LISTEN** |  | Passive open | SYN |  |
| **SYN-SENT** | SYN |  | ACK |  |
| **STN-RCVD** | SYN+ACK |  | ACK |  |
| **ESTABLISHED** |  |  |  | 연결 생성됨, 데이터 전송 중 |
| **FIN-WAIT-1** | 1st FIN |  | ACK |  |
| **FIN-WAIT-2** |  | ACK to 1st FIN | 2nd FIN |  |
| **CLOSE-WAIT** | ACK | 1st FIN | 앱 종료 |  |
| **TIME-WAIT** | ACK | 2nd FIN | 2MSL time-out |  |
| **LAST-ACK** | 2nd FIN |  | ACK |  |
| **CLOSING** |  |  |  | 양측이 동시 종료하기로 결정 |

|  |  |
| --- | --- |
| State translation diagram | Common scenario |
|  |  |

* MSL은 보통 30초에서 1분 사이의 값이다.

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

|  |  |
| --- | --- |
| 3-way handshake | |
|  | |
| Simultaneous open | Simultaneous close |
|  |  |
| Denying a connection | Aborting a connection |
|  |  |

***03-06. Flow Control***

**Sliding window:** **전송을 효과적으로 하면서 데이터의 흐름을 제어**할 수 있으므로, destination은 데이터에 지배되지 않는다.

* **TCP**의 Sliding window는 **byte-oriented**이다.

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

**<Example 3>**

|  |
| --- |
| **Receiver (host B)의 버퍼 크기가 5000byte이고 received and unprocessed data의 크기가 1000byte일 때, host A의 receiver window (rwnd) 의 값은?**  **[정답]**  Host B는 5000-1000 = 4000byte의 데이터를 버퍼에 더 받을 수 있으므로, **rwnd = 4000**이다. 이때 host B는 이 값을 host A로의 다음 segment에 광고한다. |

**<Example 4>**

|  |
| --- |
| **Rwnd의 값이 3000이고 cwnd의 값이 3500일 때, host A의 window size는?**  **[정답]**  (size of window) = min(rwnd, cwnd) = min(3000, 3500) = **3000 bytes** |

**<Example 5>**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| * 그림의 sliding window에서 **sender는 202번째 byte까지 전송**(cwnd의 값은 20)하였다. * Receiver는 **rwnd의 값이 9인 number 200의 acknowledgment**를 전송하였다. * **(size of sender) = min(rwnd, cwnd) = 9**  |  |  |  | | --- | --- | --- | | Bytes 200~202 | Bytes 203~208 | Bytes 209~ | | 전송되었으나 **acknowledge되지 않음** | **ACK을 신경쓰지 않고 전송**될 수 있음 | 전송될 수 **없음** | |

**<Example 6>**

|  |
| --- |
| **위 그림에서 Server는 acknowledgment value 202, rwnd 9의 패킷을 수신한다. Host가 byte 203~205를 이미 전송했을 때 cwnd의 값은 여전히 20이다. 이때 새로운 window는?**  **[정답]**  Acknowledgement value 202는 200, 201이 수신되었고 sender는 그것을 더 이상 신경쓰지 않아도 된다는 것이다. 따라서 window는 그것을 넘어갈 수 있다. Size of window는 바뀌지 않는다.   * 이 경우는 **window가 left를 close하고 그만큼 right을 open**하는 예시이다. |

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

**<Example 7>**

|  |
| --- |
| **위 그림에서 Sender는 acknowledgment value 206, rwnd 12의 패킷을 받는다. 이때 host는 새로운 바이트를 보내지 않았고, cwnd의 값은 여전히 20이다. 이때 새로운 window는?**  **[정답]**     * Acknowledgment value=206이므로 205까지는 sender가 더 이상 신경쓰지 않아도 됨 * Window size = min(rwnd, cwnd) = 12 * 마지막으로 전송된 바이트는 205 |

**<Example 8>**

|  |
| --- |
| **위 그림에서 host는 acknowledgment value 210의 패킷(rwnd=5)을 받았다. Host는 206~209의 바이트를 전송하였으며, cwnd의 값은 여전히 20이다. 이때의 window는?**  **[정답]**     * Acknowledgment value=210이므로 209까지는 sender가 신경쓰지 않아도 됨 * Window size = min(rwnd, cwnd) = 5 * 마지막으로 전송된 바이트는 209 |

**<Example 9>**

|  |
| --- |
| **Receiver가 Example 8에서 window shrinking이 발생하는 것을 어떻게 방지할 수 있는가?**  **[정답]**  **Receiver는 last acknowledgment number와 last rwnd를 추적**해야 한다.   * Acknowledgment number를 rwnd에 더하면 right wall에 대한 byte number를 얻을 수 있다. * Right wall의 shrinking (moving to the left) 을 방지하려면 다음과 같은 관계식을 따라야 한다. (두 식은 동치) |

* Sender window가 **shrinking하는 것을 방지**하기 위해서, receiver는 **버퍼에 더 많은 공간이 생길 때까지 기다려야** 한다.

|  |  |
| --- | --- |
| **03** | Week 11~12. **TCP Protocol** |

**[TCP Sliding window의 핵심 포인트]**

|  |
| --- |
| 1. **Window size는 min(rwnd, cwnd)** 2. Source는 full window’s worth만큼의 데이터를 **보낼 필요가 없다.** 3. Window는 **receiver에 의해 open/close**될 수 있지만, **shrink되면 안 된다.** 4. **Destination**은 shrinking window를 유발하지 않는 한 **언제든지 acknowledgment를 보낼 수 있다**. 5. Receiver는 **임시적으로 window를 닫을** 수 있다.    1. 그러나 이때도 sender는 **1바이트의 segment를 항상 전송**할 수 있다. |