|  |  |
| --- | --- |
| **01** | Week 01. **Computer Network Overview** |

**[1. Motivations of Computer Network]**

|  |  |
| --- | --- |
| Motivations | Communication Lines, 자원 공유(데이터, 프린터, 프로그램), 정보 교환 |
| Use of Computer Networks | Business application, Home application, Mobile Users |

**[2. Usages of Computer Network]**

**<Business Application of Network>**

|  |  |
| --- | --- |
|  |  |

**<Home Network Applications>**

|  |
| --- |
| * Remote information에 대한 접근 * 사람 간 커뮤니케이션 * Interactive entertainment * Electronic commerce |

**<Mobile Network Users>**

|  |  |  |
| --- | --- | --- |
| Wireless | Mobile | 예시 |
| **No** | **No** | 회사의 데스크톱 컴퓨터 |
| **No** | **Yes** | 호텔의 방에서 사용하는 노트북 컴퓨터 |
| **Yes** | **No** | Unwired building에서 사용하는 네트워크 |
| **Yes** | **Yes** | Portable office, PDA for store inventory |

**[3. Network Hardware]**

|  |
| --- |
| (거리에 따른 네트워크의 분류) |

|  |  |
| --- | --- |
| **01** | Week 01. **Computer Network Overview** |

**“규모에 따라 사용하는 네트워크 기술이 다르다.”**

|  |  |  |  |
| --- | --- | --- | --- |
| Local Area Networks | Metropolitian Area Networks | | Wireless Networks |
| **(bus)**    **(ring)** | **Bus와 star 형태의 토폴로지가 혼합**된 형태 | | 일종의 **star 형태**의 구조 |
| Wide Area Networks | | | |
|  | |  | |
| **Bus와 Mesh (또는 star) 형태의 토폴로지가 혼합**된 형태 | | sender에서 receiver로의 packet stream | |

**[4. Network Software]**

**<1. Protocol Architecture: partition & hierarchy>**

|  |  |
| --- | --- |
| * **위쪽으로 갈수록 소프트웨어 관련**, 아래쪽으로 갈수록 하드웨어 관련 계층 * **상위계층이 하위계층의 서비스를** 받음 * **각 계층마다 프로토콜**이 있고, 이에 따라 정보 교환 | * 각 **계층을 통과할 때마다 header**가 붙는다. * Header는 **프로토콜을 구현하기 위한 정보 (가장 중요한 것은 자신과 상대방을 식별하는 정보)** 를 갖는다. |

|  |  |
| --- | --- |
| **01** | Week 01. **Computer Network Overview** |

**<2. Client-Server Protocol>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 실제 **프로세스들은 User Level**에서 작동하고, 각종 **프로토콜은 응용계층을 제외하고 OS Level**에서 작동한다. (**응용 계층만 User Level**, 나머지 모든 계층은 OS Level   * 시스템 콜(소켓)으로 OS의 프로토콜 스택을 통해 상대방 서버에 접속 * 프로토콜 계층이 **User와 OS로 분리됨**  |  |  | | --- | --- | | Connectionless | Connection-oriented | | **바로 데이터를 전송** | **Connection을 맺은 후** connection이 잘 개설되면 데이터 전송 | |

**<3. Design Issue for the Layers>**

|  |
| --- |
| Addressing, Error Control, Flow Control, Multiplexing, and Routing |

**<4. Connection-Oriented and Connectionless Services>**

|  |  |  |
| --- | --- | --- |
|  | Service | Example |
| Connection-oriented | Reliable message stream  Reliable byte stream  Unreliable connection | Sequence of pages  Remote login  Digitized voice |
| Connectionless | Unreliable datagram  Acknowledged datagram  Request-reply | Electronic junk mail  Registered mail  Database query |

**<5. Service Primitives>**

|  |  |
| --- | --- |
| Primitive | Meaning |
| **LISTEN** | Incoming connection을 기다리는 블럭 |
| **CONNECT** | Waiting peer와의 connection 개설 |
| **RECEIVE** | Incoming message를 기다리는 블럭 |
| **SEND** | peer에게 메시지 전송 |
| **DISCONNECT** | connection 종료 |

|  |  |
| --- | --- |
| **01** | Week 01. **Computer Network Overview** |

**<6. Service to Protocols Relationship>**

|  |
| --- |
| * 서비스는 **Layer k에서 Layer (k+1)로** 제공된다. * 프로토콜은 **서로 동일한 레이어 간의 상호 통신을 위한 규칙**이다. |

**[5. Reference Models]**

|  |  |  |  |
| --- | --- | --- | --- |
| OSI 7계층 | **1~3 계층은 subnet**으로 하며 이 부분에서는 subnet protocol을 이용한다. | | |
| TCP/IP | TCP/IP 모델에서는 OSI 모델에 존재하는 **5, 6 계층의 기능이 7계층(application)으로 통합**되었다.   * **구현의 용이성** 때문 * 7개 계층을 이용하면 체계적이지만 **복잡해서 성능이 떨어짐** | | |
| Protocols  Networks | Layer (OSI Model) | Protocols | Networks |
| Application | TELNET, FTP, SMTP, DNS |  |
| Transport | TCP, UDP |  |
| Network | IP |  |
| DataLink+Physical |  | ARPANET, SATNET, Packet radio, LAN |

**<Critiques of OSI and TCP/IP Model>**

|  |  |
| --- | --- |
| OSI | 타이밍, 기술, 구현, 정책의 문제점 |
| TCP/IP | * 서비스, 인터페이스, 프로토콜이 구분되지 않음 * 일반적인 모델이 아님 * Host-to-network ‘layer’는 진짜 layer가 아님 * 물리 계층과 데이터 링크 계층에 대한 설명이 없음 |

|  |  |
| --- | --- |
| **01** | Week 01. **Computer Network Overview** |

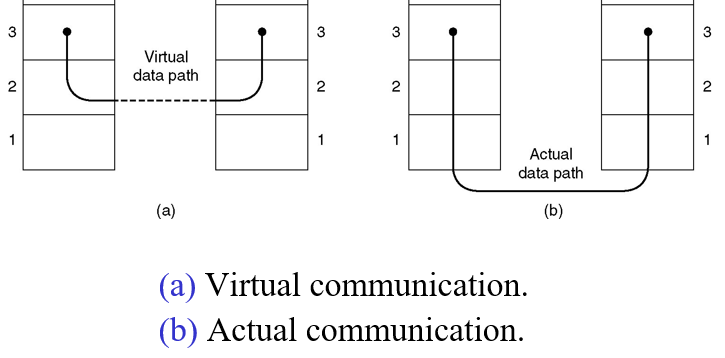
**[6. Example Networks]**

|  |  |
| --- | --- |
| ARPANET | NSFNET |
| (a)의 문제점은 **중앙 교환기가 고장 나면 통신이 불가능**하다는 것이다. -> 이러한 문제를 해결하기 위하여 (b) 형태의 ARPANET 등장 |  |
| Internet | ATM Virtual Circuits |
| * **인터넷 서비스 공급자(ISP)**가 제공 (KT, SK, LG 등)   + **ISP의 망에는 1~3계층**, User의 망에는 1~7계층 또는 1~5계층 존재 * **회사에서는 LAN**을 사용 | 물리적인 회선을 모방한 **가상 회선을 이용하여 Source와 Destination을 연결**하는 방식   * **Connection oriented** 방식     기본적인 **전송 단위는 cell**이다. |
| Ethernet | Wireless LANs |
| **Bus 형태**로 되어 있음 | 무선랜은 **Access Point를 중심으로 하여 단말기를 연결**함 (Ad-hoc에서는 Peer-to-Peer)    (Multi-cell 802.11 Network) – 간섭을 방지하기 위해 각각의 셀마다 주파수 채널이 다름 |

|  |  |
| --- | --- |
| **02** | Week 02. **The Data Link Layer (1)** |

**[1. The Data Link Layer]**

**<Service Provided to Network Layer>**



**<Service Type>** - **구분 기준: ACK의 유무, Connection Type**

|  |  |
| --- | --- |
| **Acked** CO (**connection oriented**) | ATM, FR, X.25 |
| **Acked** CL (**connectionless**) | WLAN |
| **Unacked** CL (**connectionless**) | Ethernet LAN |

* Local area network는 **wide area network보다 안정적**

**<서비스 구현>**

|  |
| --- |
| * **라우터는 1-2-3계층**으로 구성 * **3계층에서 라우팅 결정** 후 3-2-1계층으로 데이터 전송 * 2계층(데이터 링크 계층)은 **인접한 노드(router, computer) 간의 링크**를 담당 |

**<Network Packet vs. Link Frame>**

|  |
| --- |
| Sender가 **패킷을 나누어서 Receiver에 전송**한다.   * **데이터 링크 계층의 프레임:** **header(제어정보: 자신의 주소+상대방 주소 등), Payload field, Trailer**   + trailer에는 오류 검출용 **채널 코드(CRC)**가 들어감     - CRC는 **하드웨어**로 검출됨   + **Payload 다음에 trailer가 붙는 이유:** **header와 payload를 알아야지만 CRC를 계산**할 수 있으므로 |

|  |  |
| --- | --- |
| **02** | Week 02. **The Data Link Layer (1)** |

**<데이터 링크 계층의 기능>**

|  |
| --- |
| Framing, Error Control, Flow Control, Link Management, Addressing |

**<데이터 링크 계층의 프로토콜>**

|  |
| --- |
| **Unrestricted Simplex** Protocol, **Simplex Stop-and-Wait** Protocol, Simplex Protocol for a **Noisy Channel** |

**[2. Framing]**

|  |  |
| --- | --- |
| **프레임의 시작 부분에 character count를 삽입** | |
|  | **(a)** 각 프레임의 **시작 부분에 해당 프레임의 characher count** 삽입  **(b)** 오류 발생 시 **이후의 전체가 깨짐** (다음 프레임으로 계속 전파) |
| **앞뒤로 시작과 끝을 나타내는 flag 사용 (문자 또는 bit 단위)** | |
|  | **실제 데이터 부분에 flag**가 들어가면 오류가 발생한다.   * **[Sol: stuffing]** Flag 또는 escape 문자가 중간에 들어가면 **escape 문자를 앞에 붙여서 표시**한다. |
|  | **[flag: 01111110]**  비트 단위 stuffing의 경우 1이 6개 나오면 flag이므로 **1이 5개 나올 때마다 0을 넣어서 flag와 구분**한다. |

|  |  |
| --- | --- |
| **02** | Week 02. **The Data Link Layer (1)** |

**[3. Error Control]**

|  |  |
| --- | --- |
| Channel coding | EDC (Error **Detection** Code), ECC (Error **Correction** Code): FEC |
| ARQ (automatic repeat request) | **SW**(Stop-and-Wait), **GBN**(Go-back-N), **SR**(Selective-Repeat) |

* ARQ는 **자동으로 재전송**을 하는 방식이다.

**<Error Detection Code: CRC>**

**CRC: 다항식 코드(Polynomial code)**에 속하며, **G(x) : Generator Polynomial**을 이용하여 생성한다.

* 송신기와 수신기는 같은 G(x)를 이용하여 오류를 검출한다.

|  |
| --- |
| * CRC 알고리즘은 **module 2를 계산**한다. * Receiver에서는 G(x)로 나눈 결과를 계산하여 **나머지가 0이면 오류가 없고, 나머지가 있으면 오류가 있다**고 판단한다. * **CRC의 생성 개수**는 Generator의 차수와 같으며, 메시지에 **0이 CRC의 개수만큼 추가**된다. |
| [Frame=**1101011011**, Generator=**10011**] -> **Msg=11010110110000**  1101011011**0000**(2) / 10011(2) = 1100001010(2)(몫) … **1110**(2)(나머지) 이므로 Transmitted frame = **11010110111110**(2) |

**<Stop-and-Wait ARQ>**

**Stop-and-Wait ARQ:** 통신환경에 따라 **Simplest(단뱡향 통신)** 또는 **Half duplex transmission(반이중 통신)** 환경에서 사용

* **ACK 유형:** **Positive(정상)/Negative(오류)** ACK 또는 **Positive ACK Only**
* **1bit의 sequence number**를 사용 (SW ARQ에서는 **1개의 프레임을 전송한 후 ACK가 도착할 때까지 wait**하므로) – Data/ACK frame loss

**<Continuous ARQ>**

|  |  |
| --- | --- |
| SW-ARQ | **효율성** 문제 발생 |
| Full duplex transmission | |
| GBN-ARQ | 오류 발생 시 그 프레임과 이전 프레임, 즉 **버퍼의 모든 프레임 재전송**   * 구현이 쉬우므로 사용 |
| SR-ARQ | 해당 **오류가 발생한 프레임만 선택적으로 전송** (더 효율적임)   * 구현이 복잡함 (buffering, rendering) |

|  |  |
| --- | --- |
| **02** | Week 02. **The Data Link Layer (1)** |

**[4. Flow Control]**

**Flow Control:** **송수신기 사이에 성능 차이**가 날 때 **데이터 흐름을 제어**하기 위해 사용. 특히 송신기가 더 빠를 때 **송신기의 속도를 늦추기** 위하여 사용

**<Sliding Window Protocol: for Flow Control>**

|  |  |
| --- | --- |
| Sliding Window with SW-ARQ | Window size가 **1프레임**으로 제한 |
| Sliding Window with Continuous ARQ | Window size = **N > 1 frames**  수신 버퍼, 송수신 윈도우 |
| Window Size: ACK을 받지 않은 상태에서 프레임을 몇 개까지 전송할 수 있는가?   * 프레임 **전송** 시 **Sender의 right of window** 증가, **ACK** 시 **left of window** 증가 * 프레임 **수신** 시 **Receiver의 window**가 그 크기만큼 이동 * **Max window size = f(sequence number bits)** | |

**[5. Algorithms]**

|  |  |
| --- | --- |
| Unrestricted Simplex Protocol | Simple Stop-and-Wait Protocol |
| **Sender1:**   1. Network layer에서 데이터 획득 2. 그 데이터를 s에 복사 3. Physical layer로 데이터 전송   **Receiver1:**   1. Physical layer에서 프레임 획득 2. Network layer로 데이터 전송 | **Sender2:**   1. Network layer에서 데이터 획득 2. 그 데이터를 s에 복사 3. Physical layer로 데이터 전송 4. **다음 이벤트 발생 시까지 대기**   **Receiver2:**   1. **다음 이벤트 발생 시까지 대기** 2. Physical layer에서 프레임 획득 3. Network layer로 데이터 전송 4. **Physical layer로 dummy frame 전송하여 sender를 깨움** |
| A Simplex Protocol for a Noisy Channel | |
| **Sender3:**  Next\_frame\_to\_send를 0으로 초기화  1번째 packet을 fetch함   1. Physical layer로 데이터 전송 2. 타이머 시작 3. Frame\_arrival, cksum\_err 또는 timeout 대기 4. **프레임이 도착하면**    1. Physical layer로부터 ACK 획득    2. **ACK==next\_frame\_to\_send이면**       1. 타이머 중지       2. Network Layer에서 다음에 전송할 데이터 획득       3. Next\_frame\_to\_send를 증가 | **Receiver3:**  Frame\_expected를 0으로 초기화   1. Frame\_arrival 또는 cksum\_err 대기 2. **프레임이 도착하면**    1. Physical layer로부터 프레임 획득    2. **R.seq==frame\_arrival이면**       1. Network Layer로 데이터 전송       2. Frame\_expected를 증가    3. **S.ack = 1-frame\_expected** (어떤 프레임이 ACK되었는가?)    4. ACK을 physical layer로 전송 |

|  |  |
| --- | --- |
| **03** | Week 03. **The Data Link Layer (2)** |

**[1. Sliding Window Protocol]**

|  |  |  |
| --- | --- | --- |
| 3-13 | A | 초기 상태 |
| B | 첫 번째 프레임이 **전송**된 직후 |
| C | 첫 번째 프레임이 **수신**된 직후 |
| D | 첫 번째 프레임의 **ACK가 Sender에 도착**한 직후 |

**<Algorithms>**

|  |
| --- |
| A One-Bit Sliding Window Protocol |
| 1. Frame\_arrival, cksum\_err 또는 timeout 대기 2. **프레임이 도착하면**    1. Physical layer로부터 프레임 수신    2. **R.seq==frame\_expected이면, 즉 sequence number가 프레임 번호와 같으면**       1. Network Layer로 데이터 전송       2. Frame\_expected의 값을 증가    3. **R.ack==next\_frame\_to\_send이면, 즉 ACK이 다음에 송신할 프레임 번호와 같으면**       1. Timer를 정지한다.       2. Network layer로부터 packet을 받는다.       3. Next\_frame\_to\_send의 값을 증가시킨다. 3. Sequence number를 증가시킨다. 4. **S.ack = 1 – frame\_expected**, 즉 마지막으로 수신한 프레임의 sequence number로 한다. 5. Physical layer로 프레임을 전송한다. 6. Timer를 시작한다. |

**<One-bit sliding window protocol 예시>**

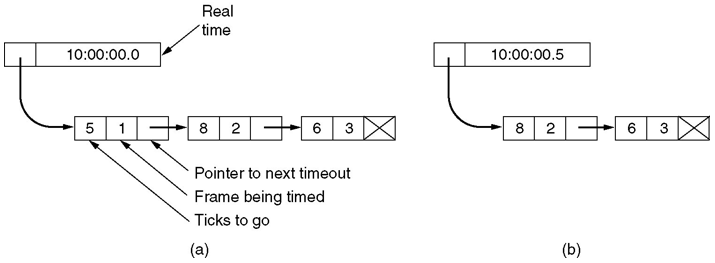
|  |  |
| --- | --- |
|  | 1. **정상적인 경우: 양방향 통신** 2. **비정상적인 경우**  * **B sends (0,0,B0):** **(0,1,B0)에 대한 ACK가 없으므로** 재전송 * **A sends (0,0,A0):** **(0,1,A0)에 대한 ACK가 없으므로** 재전송 * 마찬가지로 (1,0,A1), (1,0,B1)도 각각 (1,1,A1), (1,1,B1)로 재전송 * **(X,Y,Z)에서 Y가 1로 시작하는 이유:** **초기값을 0으로 설정하면 혼동**되므로 1로 설정함 |

**\*: 패킷 수신**

|  |  |
| --- | --- |
| **03** | Week 03. **The Data Link Layer (2)** |

**[2. Continuous ARQ]**

|  |  |  |  |
| --- | --- | --- | --- |
| ARQ | 성능 | 구현 | Window Size |
| Stop-and-Wait  (SW) | **떨어짐**  (window size=1이므로 **한번에 하나씩만 전송**) |  | **1** |
| Go-back-N  (GBN) | **떨어짐**  (오류가 발생한 프레임 다음의 프레임을 잘 받았는데도 **N개만큼 무시하므로 불필요한 재전송 필요**) | **쉬움** |  |
| Selective Repeat  (SR) | **좋음**  (**Error가 발생한 프레임만 재전송**하므로) | **복잡함**  (**NAK, buffer, 순서 맞추는 기능**) |  |
| * **예상 외의 프레임이 들어오면 NAK**을 보낸다. * **순서가 다른 프레임을 discard하지 않고 buffer에 저장**하고, 나중에 원하는 프레임이 들어오면 순서를 맞춰야 한다. * 최근에는 구현이 복잡해도 성능이 좋은 SR-ARQ 사용 | | |

**타이머 구현:** 프레임을 보낼 때마다 타이머를 프레임 단위로 작동시키면 **타이머가 너무 많아지므로** 독립적으로 구현하는 것은 **오버헤드가 크다**. 따라서 **시간의 차이를 이용, 포인터로 링크**하는 형태로 구현한다.

**[Example 1]**

|  |
| --- |
| 1. **3-16GBN-ARQ 프로토콜**  * Sender 측에서는 **0, 1번을 보냈고 이에 대한 ACK**를 받는다. 이후 2~8번을 보내고 2번에서 error가 발생하여, **3~8번은 2가 먼저 들어와야 하므로 discard**된다. * Timeout이 되면 **N개만큼 back하여 2번**으로 되돌아간다.  1. **SR-ARQ 프로토콜**  * 이번에는 **오류가 발생한 2번만 Error가 발생**하고 **나머지는 버퍼에 저장**된다. * 3번 프레임이 전송되었을 때 **예상했던 2번 프레임이 아니므로 NAK**을 보낸다. * 나중에 **2번 프레임이 들어오면 순서를 맞춘다**. |

|  |  |
| --- | --- |
| **03** | Week 03. **The Data Link Layer (2)** |

**[Example 2]**

|  |  |
| --- | --- |
| **(seqnum : 3 bits)** | |
| **Window Size = 7 (a) (b)** | **Window Size = 4 (c) (d)** |
| 0~6의 **7개 프레임이 send and receive되었지만 ACK되지 않은** 경우   * Sender 측은 ACK를 받지 못했으므로 **0~6을 재전송** * Receiver 측은 ACK가 손실된 것을 모르므로 **7~0~5를 대기** * 이때 **재전송한 데이터 0~5와 새 데이터 0~5를 혼동**할 수 있음 | 0~3의 **4개 프레임이 send and receive되었지만 ACK되지 않은** 경우   * Sender 측은 ACK를 받지 못했으므로 **0~3을 재전송** * Receiver 측은 **4~7을 대기** * **Sequence number가 중복되지 않음** |

**<Algorithms>**

|  |  |
| --- | --- |
| Sliding Window Protocol Using Selective Repeat | |
| **Send\_frame:**  프레임을 physical layer로 보낸다.  Data frame이면 타이머를 시작한다.  ACK timer를 중지한다.  **Protocol6:**   1. Event를 대기한다. 2. **Network\_layer\_ready 이벤트가 발생하면**    1. Window를 확장한다.    2. Network layer로부터 패킷을 받는다.    3. Send\_frame(data);    4. Upper window edge를 늘린다. 3. **Frame\_arrival 이벤트가 발생하면**    1. Physical layer로부터 프레임을 받는다.    2. **Data frame이면**       1. (undamaged) Frame\_expected가 아니고 no\_nak이면 send\_frame (NAK) | * + 1. **프레임이 어떤 순서로든 도착할 수 있으면**        1. Buffer의 상태를 full로 설정        2. Buffer에 데이터 추가        3. 프레임을 pass하고 receiver의 lower, upper edge를 증가시킨다.        4. ACK timer를 시작한다.   1. **NAK이면** 일련변호 확인 후 재전송   2. 프레임을 버퍼에서 뺀다.   3. 타이머를 중지한다.  1. **Cksum\_err 이벤트가 발생하면**    1. NAK이 아니면 damaged frame이므로 Send\_frame(NAK); 2. **timeout이면**    1. Send\_frame(data); 3. **ack\_timeout이면**    1. Send\_frame(ACK); |

* 버퍼에 프레임을 계속 가지고 있는 이유는 **재전송하기 위해서**이다.

|  |  |
| --- | --- |
| **03** | Week 03. **The Data Link Layer (2)** |

**[3. Protocol Verification]**

**Protocol Verification:** 프로토콜에 논리적 오류가 있는지 판단한다.

|  |  |
| --- | --- |
| Finite State Machine | **상태 값: (일련번호, 수신번호, 채널상태)**   * 상태 및 상태 간의 transition을 정의 * **채널상태:** n(n번 프레임), A(ACK), -(아무것도 없음)   3-21 |
| HDLC | 3-24   |  |  |  |  | | --- | --- | --- | --- | | Flag | 01111110 (8 bits) | | | | Address | 목적지 주소 (**multipoint일 때 목적지를 구분**하기 위하여 사용) | | | | Control | Information | 0+3(seq)+1(P/F)+3(Next) | 용도 | | Supervisory | 10+2(type)+1(P/F)+3(Next) | **Flow/Error control용 ACK, NAK** | | Unnumbered | 11+2(type)+1(P/F)  +3(Modifier) | **링크 관리**(개설, 끊기, 테스트 등) | | **P/F:** 보통은 0, **비상 상황에서는 1**  **Type:** ACK/NAK 구분, **Next:** ACK 번호 | | | | Data | 0비트 이상의 데이터 | | | | Checksum | 오류 검사에 사용 | | | |
| PPP (Point to Point Protocol) | * **Payload는 상위 계층의 패킷** * Home Personal Computer에서 주로 사용 * HDLC(bit level)과 달리 **byte level, 즉 문자 단위의 flag**임  |  |  | | --- | --- | | Establish | Network | | Link level에 연결 **(LCP: link control protocol 사용)** | Network level 연결 | |

|  |  |
| --- | --- |
| **04** | Week 04. **The Medium Access Control Sublayer** |

**[1. MAC]**

**MAC(Medium Access Control Sublayer):** **전송 매체를 공유하기 위하여** 필요한 프로토콜

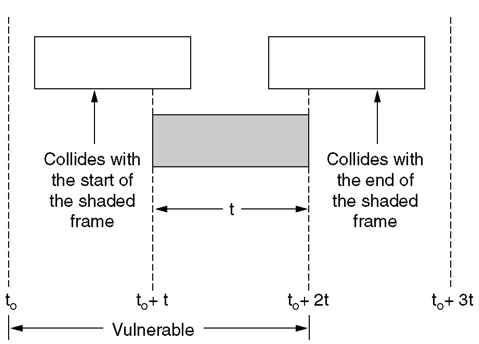
* 1:1 독점 형태에서는 불필요, **LAN이라는 근거리망(Ethernet)**이 들어오면서 필요하게 됨

**<채널 할당 방식>**

|  |  |
| --- | --- |
| Static Channel Allocation | **다중 채널을 각 사용자마다 구현**   * **FDMA/TDMA/CDMA** 방식 * 데이터가 있을 때뿐만 아니라 **전체 주기 동안 채널이 정적으로 할당**됨 * **문제점:** **bursty**(간헐적 트래픽 패턴), **효율이 떨어짐**(사용하지 않아도 채널이 할당) |
| Dynamic Channel Allocation | 필요할 때(데이터 생성, 전송)만 **채널을 동적**으로 할당   * **Random Access:** **랜덤하게 경합**을 해서 채널을 할당 |

**[2. ALOHA 계열 MAC 프로토콜]**

|  |
| --- |
| Pure ALOHA |
| 데이터가 있으면 **일단 보내고 보는 방식**   * 성공적으로 전송되면 종료, **충돌이 생기면 random delay** 후 재전송 * 임의의 시간에 slot 전송 가능 * **성능 평가 기준:** **Vulnerable Period (충돌이 일어날 수 있는 기간)** |
| Slotted ALOHA |
| **Slot 단위의 전송만 가능**한 ALOHA 방식   * Pure ALOHA의 **throughput이 떨어지는 문제 해결** 가능 * 프레임 전송은 **항상 slot의 경계에서만 가능** |

**<성능 평가: Poisson Distribution>**

|  |  |  |
| --- | --- | --- |
|  | Pure ALOHA | Slotted ALOHA |
| x (사건 횟수) |  | |
| (성공 확률) |  | |
| (초당 프레임수) |  | |
| S (throughput) |  | |
| (vulnerable period) |  |  |

**는 Offered Load, 는 프레임 전송 시간**

|  |  |
| --- | --- |
| **04** | Week 04. **The Medium Access Control Sublayer** |

따라서 Pure ALOHA와 Slotted ALOHA의 성능은 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| 4-03 |  | Pure  ALOHA | Slotted  ALOHA |
| S |  |  |
| Max S |  |  |
| Value of G  (Max S) |  |  |

**[3. CSMA 계열 MAC 프로토콜]**

**CSMA:** **CS(Carrier Sense)를 이용**하는 MAC 프로토콜

* **Slot ALOHA도 효율이 좋지 않으므로** 도입
* CS(Carrier Sense)가 있으면 CSMA, 그렇지 않으면 ALOHA라고 할 수 있음

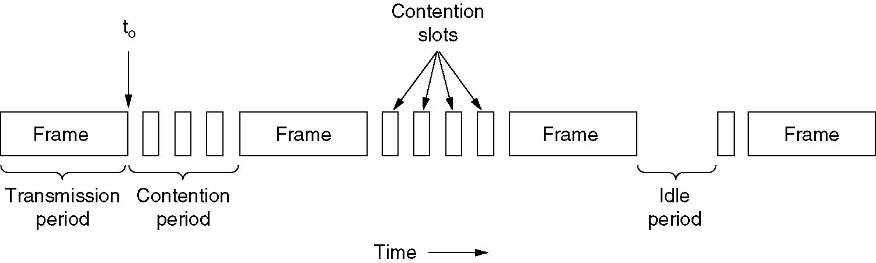
|  |  |
| --- | --- |
| Random Access | Carrier Sense: **carrier(반송파)가 있는가?** |
| 사용자마다 **delay time을 다르게 하여 데이터 전송 시 충돌 확률을 감소**시킨다. | * 채널이 **busy 상태이면 전송하지 않는다.** * idle이면 를 계산하여 전송한다. **(r: 0~1의 랜덤값)**   + **r>p이면 sleep 상태**가 됨 * busy 상태이면 다음과 같이 한다.  |  |  | | --- | --- | | Persistent | 루프를 돌면서 idle이 될 때까지 대기 | | Non-persistent | Sleep 상태로 전환하여 종료 | |

**<Persistent vs. Non-Persistent CSMA>**

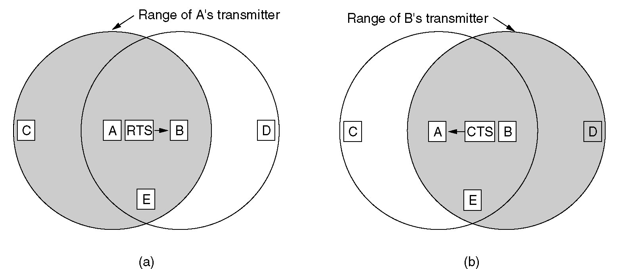
|  |  |  |
| --- | --- | --- |
|  | Persistent CSMA | Non-persistent CSMA |
| 예시 | **P=1이면 Ethernet** (CSMA-CD, 충돌검출가능)  **P=0이면 WLAN** (충돌 최소화) | **WSN** |
| 사용 | **루프를 돌면서 지속적으로 채널 상태를 확인**, **전력소모 많음** | **Sleep상태로 전환하여 종료**하므로 **전력소모 적음**, 임베디드 디바이스에서 사용 |

* p>0이면 **p의 값이 작아질수록 throughput이 증가**한다.
* 1-persistent는 Ethernet에서, Non-persistent는 무선(충돌 최소화)에서 사용한다.

|  |  |
| --- | --- |
| **04** | Week 04. **The Medium Access Control Sublayer** |

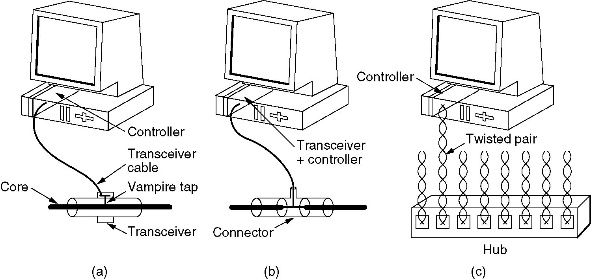
**CSMA-CD (CSMA with Collision Detection):** **1-persistent CDMA의 성능을 보완**하기 위해 사용

* 충돌이 생기면 전송을 중단하고 나중에 재전송
* 충돌 검출을 위해 아날로그 회로 사용, **신호의 세기가 크면 충돌이 발생**했다고 판단

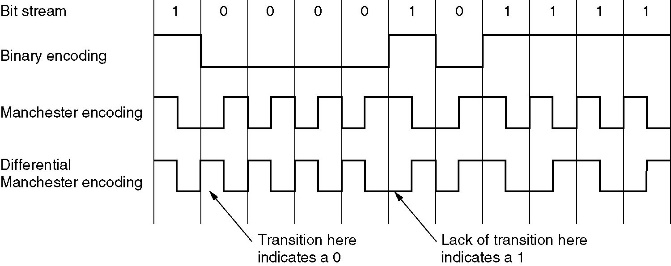
**Wireless LAN Protocol:** **(전송 범위의 문제)** A의 전송 범위 안에 B, C, E가 들어가므로 A는 B, C, E에 전송할 수 있다.

**[4. IEEE 802 프로토콜과 Ethernet]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IEEE 802.2 | IEEE 802.3 | IEEE 802.11 | IEEE 802.15 | IEEE 802.16 |
| LLC (데이터 링크 프로토콜) | CSMA/CD  (Ethernet) | Wireless LAN | Wireless PAN | Wireless MAN |

**<이더넷 케이블링>**

**(a), (b)는 bus type, (c)는 star type**이다.

**Encoding:** **디지털 데이터를 전압 펄스 형태**로 보낸다.

|  |  |
| --- | --- |
| Binary encoding | 원래 bit를 그대로 사용 |
| Manchester encoding | 각 bit에서 translation 발생  (**장점:** 동기를 위한 별도의 clock line 없이 **송수신기의 인코딩 방식으로 translation을 이용**하여 동기화) |
| Differential Manchester encoding | 각 bit의 시작점에서 해당 bit가 **1이면 연속, 0이면 불연속**이다. |

|  |  |
| --- | --- |
| **04** | Week 04. **The Medium Access Control Sublayer** |

**<Ethernet MAC Sublayer Protocol>**

|  |  |
| --- | --- |
| DIX Ethernet | **Preamble:** 처음 시작할 때의 프레임 동기를 설정하기 위하여 지정 |
| IEEE 802.3 | **SOF:** 프레임의 시작을 나타냄 |

**Pad:** null data로, 데이터의 크기가 **46바이트 미만일 때 46바이트 이상이 되게 만들기 위하여** 사용

* **Data가 46바이트 이상이여야 하는 이유:** **CSMA/CD Mac Protocol의 최소 프레임 크기 제한**(Preamble을 제외한 64바이트)을 지키기 위하여
* 다음 실험에서 **64바이트 이상이어야 CSMA/CD 프로토콜에 의한 collision detection이 가능**하기 때문이다.

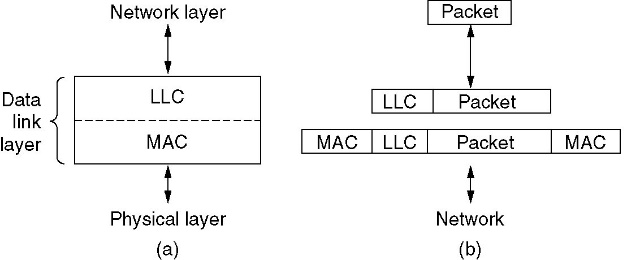
|  |
| --- |
| 4-09B의 채널에 들어오기 직전에 B의 채널은 idle 상태이므로 B에서 전송하면 충돌이 발생한다.  **“Collision detection time은 최대 2까지 가능”** |

|  |
| --- |
| 왼쪽 다이어그램을 보고 **Ethernet MAC이 CSMA/CD 방식**임을 이해해야 한다.   * **P-persistent**(DEF\_WAIT의 순환루프), **Collision Detection**을 확인 |

**<Switched/Fast/Gigabit Ethernet>**

|  |  |
| --- | --- |
| Switched Ethernet | 최근 Ethernet의 추세 (단말기들이 **star 형태**로 연결) |
| Fast Ethernet | 원래 **10Mbps**였는데 802.3u에서 **100Mbps**로 증가, 동축케이블 대신 **twisted pair 또는 광케이블** 사용 |
| Gigabit Ethernet | Two-station 또는 multi-station, **1Gbps**로 속도 증가, **광케이블** 중심 |

|  |  |
| --- | --- |
| **04** | Week 04. **The Medium Access Control Sublayer** |

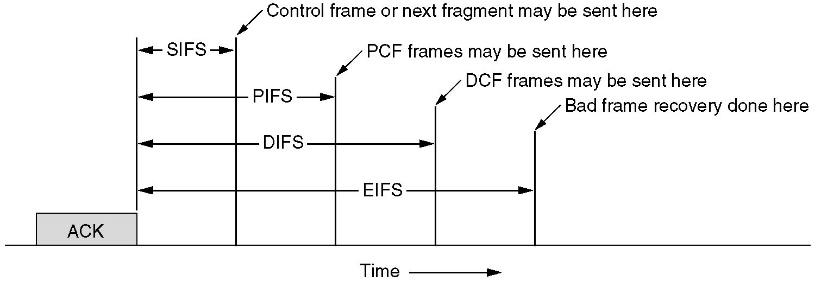
**IEEE 802.2(LLC):** 데이터 링크 계층의 5대 기능 수행

* MAC이 추가되어 **데이터 링크 계층이 LLC, MAC으로** 나누어짐
* 패킷이 들어오면 **LLC, MAC에 대해 각각의 헤더**가 있으며, 물리 계층으로 들어가기 전에 **CRC를 이용한 오류 검출을 위한 trailer**가 있음

|  |  |
| --- | --- |
| 802.11 Protocol Stack | Data Link Layer에서 공통 LLC를 사용하며, **MAC은 CSMA/CA 공통**이다. (같은 CSMA이지만 /CD, /CA간 호환은 안됨)   * **물리계층은 서로 다른 기술** 이용 |
| 802.11 MAC Sublayer Protocol | **Hidden/exposed station** 문제 (전송 가능 범위, 무선 환경에서의 성능저하) |

**[Hidden/Exposed station problem]**

|  |  |
| --- | --- |
| Hidden | C의 전송 범위 밖에 A가 있으므로, **A가 Carrier Sensing을 해도 idle로 판단되므로 C가 B에 데이터를 보내면 충돌** 가능 |
| Exposed | A에서 B로 데이터를 보내고 있는데 **B는 이것 때문에 자신의 채널이 busy하다고 판단하므로 C로 데이터를 보내지 않음** (그러나 보내도 충돌하지 않음) |

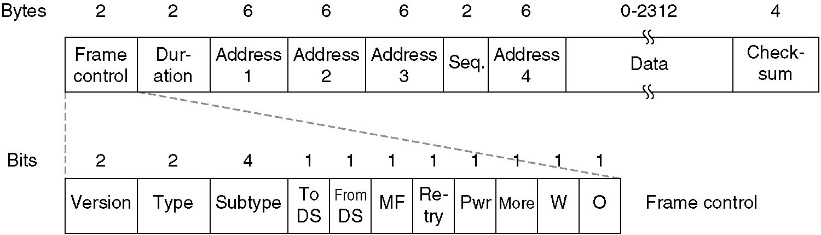
* 이 문제를 해결하기 위하여 **RTS, DTS 프레임을 교환**하여 데이터 전송 전에 주변 node에 알려준다. **(NAV: 전송주기)가 지난 후 C가 프레임을 전송**한다.
* 무선 네트워크는 Collision Detection이 불가능하므로 **Collision Avoid를 하며, 이는 신호의 세기에 따라 판단**한다.

**[Interframe Space]** 프레임 간의 간격은 **SIFS(Control frame에서 사용) < PIFS(PCF 방식에서 사용) < DIFS(DCF 방식에서 사용) < EIFS(특수한 경우 사용)**

* 간격이 짧을수록 우선권이 높아서 먼저 전송된다.

|  |  |
| --- | --- |
| **04** | Week 04. **The Medium Access Control Sublayer** |

**[802.11 Frame Structure]**



|  |  |  |  |
| --- | --- | --- | --- |
| Frame Control | Duration | Address | Seq. |
| 프로토콜 제어 정보 | 전송 주기 | 4개인 이유는 **Access Point (AP)를 통해** 보내기 때문 (자신과 상대방이 속한 AP의 주소) | 일련번호 |

**[5. Internetworking]**

|  |  |
| --- | --- |
| 물리 계층 | Repeater, Hub |
| 데이터 링크 계층 | 4-42Bridge, Switch   * Bridge는 **서로 다른 종류의 LAN 간의 연결**에 사용된다. * **LAN type마다 format이 다르므로** bridge를 통해 format을 맞춰야 한다. * **이중 브릿지**를 사용할 수 있는데, 이때 **사이클이 발생하여 자원 낭비**가 발생할 수 있다.   + 이를 해결하기 위해 **Spanning Tree 알고리즘** 적용 |
| 네트워크 계층 | Router |
| 전송 계층부터 | Gateway |

|  |  |
| --- | --- |
| **05** | Week 05. **The Network Layer (1)** |

**[1. Network Service Model]**

**Network Service Model의 요구사항:** bandwidth, inter-packet timing 방지, loss-free 전송, in-order 전송, congestion feedback to sender

**Network Service Model의 유형:** virtual circuit 또는 datagram

|  |  |
| --- | --- |
| **05** | Week 05. **The Network Layer (1)** |

**<Connection Oriented 서비스: Virtual Circuit 형태>**

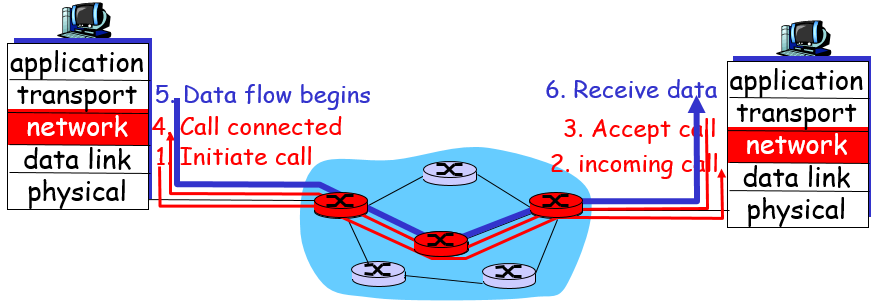
* 데이터를 보내기 전에 **connection**을 맺는다.
* **Source-to-Dest path**는 telephone circuit처럼 **경로를 정하여** 사용한다.
* 성능은 **좋은 편**이며(**여러 자원을 할당**할 수 있으므로), **Source부터 Destination까지 connection**이 필요하다.

|  |
| --- |
| 1. **call setup**을 하여 data가 흐르게 한다.  2. 가상회선 구분을 위해 **각 packet은 VC identifier를 carry**한다. (destination host ID가 아님)  3. **Source-dest path**는 각각의 연결에 대하여 **상태를 관리**해야 한다.   * 전송 계층 연결은 2개의 end system만을 포함한다.   4. 실제 circuit과 같은 성능을 위해 **link, router 자원(bandwidth, buffer)이 VC에 의해 할당**되어야 한다. |

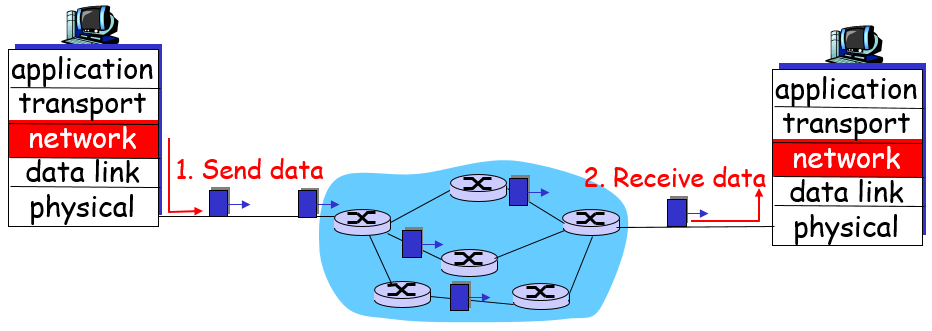
**<Virtual circuits: Signaling protocols>**

**Signaling protocol:** **setup, teardown VC 유지, ATM, frame-relay, X.25**에서 사용하며 현재 인터넷에서는 사용하지 않는다.

|  |
| --- |
| 1. call setup을 위하여 **call 요청**을 한다.  2. call 요청이 **accept되면 양쪽으로 signaling을 통해 connection**이 생성된다.  3. 한번 setup되며 종료될 때까지 유지되며, **데이터는 정해진 virtual circuit을 통해 이동**한다. |



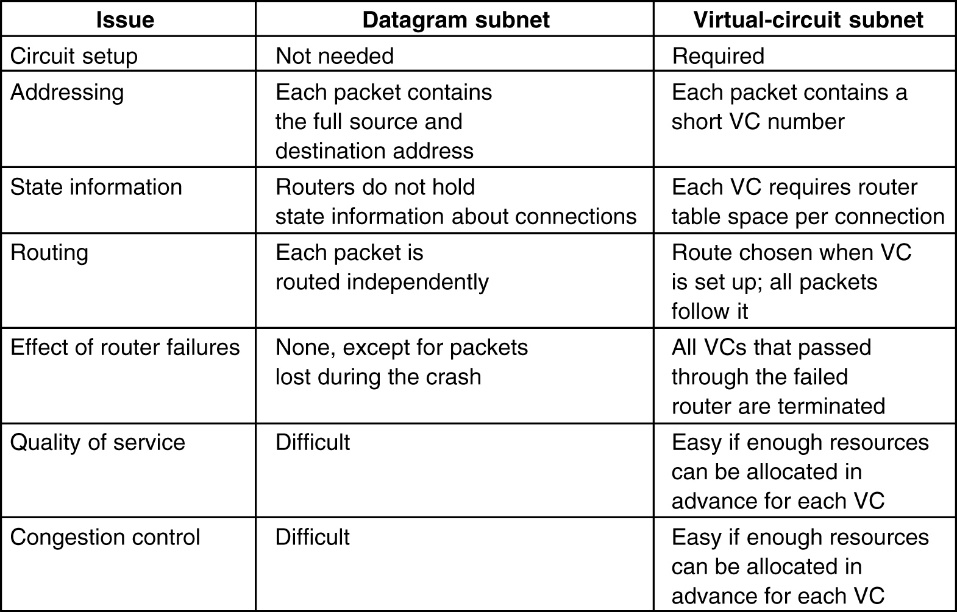
**<CL Service: Datagram (인터넷)>**

* **Call setup이 없다.**
* 라우터는 end-to-end 연결에 대한 상태가 없다. 즉 **network level의 connection** 개념이 없다.
* 패킷은 **목적지 주소를 이용하여 포워딩**된다. 따라서 **서로 같은 Source, Destination을 갖는 패킷의 이동 경로가 다를** 수 있다.

|  |  |
| --- | --- |
| **05** | Week 05. **The Network Layer (1)** |

**<Datagram or VC network>**

|  |  |
| --- | --- |
| Internet | ATM |
| **Connectionless방식으로 elastic(flexable)**하다.   * 우회 경로를 찾는다. * 타이밍 요구 사항을 만족하는 것은 어려움   **일반적인 컴퓨터(smart end system)**를 사용하므로 성능이 좋다. | **가상 회선**을 사용하고, 기존의 전화망과 상당히 유사   * **타이밍 요구사항** 만족 * **Bandwidth** 보장 가능 * **단말기는 기능이 약함(dumb)** |



**네트워크의 3대 주요 기능:** **라우팅(routing)**, 혼잡 제어(congestion control), 인터네트워킹

**[2. Routing]**

**Routing(라우팅):** **Source to Dest로 가는 ‘좋은’ 경로**를 찾는 것

* 일반적으로 **그래프 이론**을 사용하며, **node는 router, edge는 물리적 link**이다.
* **Good Path:** 일반적으로 **최소의 cost**가 발생하는 path로 정의한다.

**<Routing Factors>**

|  |  |
| --- | --- |
| Where | Source, 각 node(인터넷에서 사용), 중심 node(중앙 집중 방식) |
| How | **Static** vs. **Dynamic** |
| When | **Virtual Circuit** vs. **Datagram** |
| Criteria | Hop Count, delay, cost, load, bandwidth |

|  |  |
| --- | --- |
| **05** | Week 05. **The Network Layer (1)** |

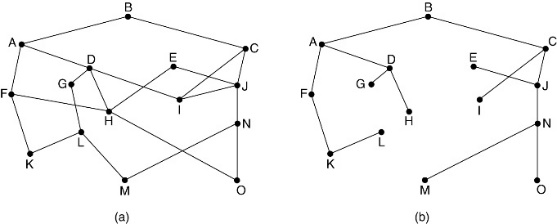
**<Global vs. Decentralized>**

|  |  |
| --- | --- |
| Global | 모든 node가 link cost 정보를 포함하여 완전한 topology를 갖는다.  “link state” 알고리즘 |
| Decentralized | 각 node가 물리적으로 연결된 neighbor와 그들 간의 cost를 알 때   * 계산을 반복하고 이웃과 정보를 교환한다. * 대표적으로 “Distance Vector” 알고리즘이 있다. |

**<Static vs. Dynamic>**

|  |  |
| --- | --- |
| Static | **라우팅 경로가 시간이 지남에 따라 조금씩** 변한다. |
| Dynamic | 라우팅 경로가 **빠르게** 변한다.   * **Link cost change**에 따라 주기적으로 업데이트 |

**<Sink Tree>**

**Optimality Principle(최적 원리):** **Bellman의 최적 원리**에 기반한 Q-learning 기법을 사용한다.

* Dijkstra 알고리즘 등의 **최단거리 알고리즘**을 통해 **최단거리를 나타낸 sink tree**를 생성
* Sink Tree에서 전체 경로의 **부분 경로들도 최적**임 (즉 최단거리가 부분집합에도 적용)

**<Distance Vector Routing> - Bellman-Ford 알고리즘 사용**

|  |  |
| --- | --- |
| Iterative | 어떤 node도 더 이상 정보를 교환하지 않을 때까지 반복   * **Stop signal이 없음** |
| Asynchronous | **Lock step**에서는 node가 정보를 교환하지 않아도 됨 |
| Distributed | 각 node는 **직접 연결된 node들과**만 통신 |
| Distance Table 자료구조 | * 각 node는 **자신의 distance table**을 갖는다. * 각 행은 **가능한 destination**을, 각 열은 직접 연결된 이웃 node를 나타낸다. * **Loop** 발생 가능 |
| 핵심 수식 | * 목적지 **Y**에 도착할 때까지 recursive하게 최단경로 탐색 * **주기적으로 이웃 node와 정보 교환** 시 업데이트 |

* **Network diameter에 따라 hop 수**가 정해진다.

|  |  |
| --- | --- |
| **05** | Week 05. **The Network Layer (1)** |

**<Distance Vector Routing에서 Routing Table 생성>**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 를 이용한 계산  (**Distance Table 생성**) | Distance Table을 이용한 **라우팅 테이블 생성**  **(각 열에서 최솟값을 선택)** | Distance Table을 **모든 이웃 node에 전송**한다. |

**<Distance Vector Algorithm>**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **06** | Week 06. **The Network Layer (2)** |

**[1. Routing]**

**<Link State Changes>**

|  |
| --- |
| 1. node는 local link cost가 바뀐 것을 탐지한다.  2. distance table을 업데이트한다.  3. cost가 **least cost path**에 대하여 바뀌면 이웃 node에 알린다. |
| **[Good News Travels Fast]**  dv_goodAfter t1:  After t2: |
| **[Bad News Travels Slow] – “Count to Infinity Problem”**  After t1:  After t2:  After t3: |

**<Link State Routing>**

|  |  |
| --- | --- |
| 단계 | Distance Vector와 공통 |
| 1. 이웃 node를 찾아서 그들의 **network address**를 찾는다. |  |
| 2. 각 neighbor의 **delay 또는 cost를 측정**한다. |  |
| 3. **이웃 node의 링크 상태(delay, cost 정보)**를 의미하는 패킷을 만든다. |  |
| 4. 이 패킷을 **모든 router로 전송**한다. |  |
| 5. **Dijkstra 알고리즘**을 적용하여 **다른 모든 라우터와의 shortest path**를 계산한다. |  |

* **“전체 network의 topology를 알고 적용하는 방법”**

|  |  |
| --- | --- |
| **06** | Week 06. **The Network Layer (2)** |

**<Dijkstra’s Algorithm>**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 자신 주변의 **이웃 node**를 찾는다. (hello packet 전송)  2. **cost를 측정**한다. (패킷을 전송하여 반사시킨 후 시간을 반으로 나눔)  3. **Link state packet**을 만든다. (각 node마다 패킷을 만들어서 이웃 node와의 링크 상태 표현)   * **(Source node – 일련번호 – 수명 – 각 node로 갈 때의 cost)** * 각 node는 **LS packet**을 만들어서 방송한다. * **Acknowledge를 하는 이유:** **못 받으면 재전송**하기 위해   + **Reliable broadcast**, 즉 방송을 하되 받지 못하면 재전송하는 방법을 이용한다.   + **하나라도 받지 못하면 topology가 깨지기** 때문 | | | |
| **[Dijkstra’s Algorithm]**  1. network topology와 link cost는 **모든 node가 알고 있다**.   * **Link state broadcast**를 이용하여 구현 가능 * 모든 node가 서로 같은 정보를 갖는다.   2. 어떤 노드에서 **모든 다른 노드와의 least cost path를 계산**한다.   * 해당 node에 대한 routing table을 생성할 수 있게 한다. | | **[Notations]** | |
| c(i,j) | Node i에서 j로의 **link cost** (직접 연결되지 않으면 무한대) |
| D(v) | Source에서 destination v로의 **현재 cost** 값 |
| p(v) | Source->v path에서의 **predecessor node** (next v) |
| N | **Least cost path가 알려진 node**의 집합 (즉 **영구 node**의 집합) |
| **[Program]** | **[Example]** | | |

**<Algorithm Complexity: n nodes>**

* 각 iteration에서 N에 있는 것을 제외한 모든 node를 탐색한다.
* **n(n+1)/2회의 비교**가 필요하므로 Complexity는 이다. (더 효율적인 경우 )

|  |  |
| --- | --- |
| **06** | Week 06. **The Network Layer (2)** |

**<Comparison of LS(Link State) and DV(Distance Vector) algorithms>**

|  |  |  |
| --- | --- | --- |
|  | Link State Algorithm | Distance Vector Algorithm |
| Message Complexity | n개의 node와 E개의 link가 있는 경우 개의 메시지 전송 | **이웃 간에만** 메시지를 교환 |
| Convergence Speed | 알고리즘을 사용하는 경우 **개의 메시지**가 전송된 후 수렴 | **Convergence time의 편차가 심함**   * routing loop 존재 * count-to-infinity 문제점 |
| Robustness  (라우터 오작동 시) | * **부정확한 link cost를 광고**할 수 있다. * 각 node는 **해당 node에 대한 table만**을 계산한다. | * DV node가 **부정확한 path cost를 광고**할 수 있다. * 각 node의 **table은 다른 node가 사용**할 수 있다. |

* Distance Vector에서는 **hop 수**를 이용하는데, **cost 관점에서는 열악하고 부정확**하다.
* **Link State 방식을 주로 사용하는 이유:** **Distance Vector는 수렴 속도**의 문제가 크고, **Link State는 link의 cost를 다양하게 표현**할 수 있다.