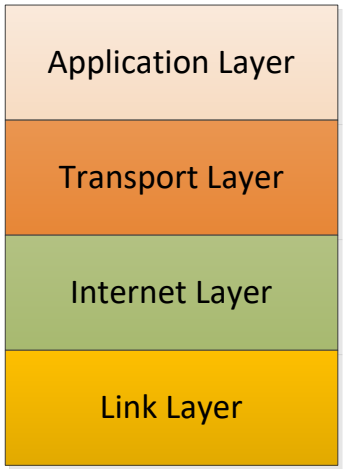
네트워크 프로그래밍

TCP/IP

- 컴퓨터끼리 네트워크에서 데이터를 주고받기 위해서는 그 네트워크에서 통용되는 "프로토콜(Protocol)"을 따라야 함.
- 프로토콜은 규약, 규칙이라는 뜻의 낱말로써, 여기에서는 "컴퓨터들이 네트워크를 통해 데이터를 주고받기 위한" 통신 규약을 말함.
- 프로토콜에는 굉장히 다양한 종류가 있으나, 인터넷의 실질적인 표준 프로토콜은 TCP/IP 임.
- TCP/IP는 여러 가지 프로토콜의 모음(Suite)로써, 4개의 계층으로 구성되어 있으며, 한 계층 위에 다른 계층이 포개어져 있는 형태로 이루어져 있음.
  그래서 이것을 TCP/IP 스택(Stack)이라고 부르기도 함.



TCP/IP

– Link Layer

- 네트워크의 물리적인 연결 매체를 통해 패킷을 주고 받는 작업을 담당
- LAN 케이블로 연결되어 있든 ADSL로 연결되어 있든 또는 Wi-Fi로 연결되어 있든 링크 계층에서 이를 담당하여 처리함.
- 가령 어떤 패킷이 네트워크를 통해 컴퓨터에 들어오면 제일 먼저 바로 이 링크 계층이 맞이함.
- 링크 계층은 이 패킷에서 물리적 데이터 전송에 사용되던 부분을 제거하고 인터넷 계층에 넘김.
- 이렇게 함으로써 인터넷 계층에서는 패킷이 전파를 타고 넘어왔든 광케이블을 타고 넘어왔든 간에 아무 신경도 쓰지 않고 자신의 일을 처리할 수 있음.

TCP/IP

– Internet Layer

- 인터넷 계층은 패킷을 수신해야 할 상대의 주소를 지정하고, 나가는 패킷에 대해서는 적절한 크기로 분할 하며 들어오는 패킷에 대해서는 재조립을 수행
- 이 계층에서 사용되는 규약이 바로 인터넷 프로토콜(Internet Protocol) 즉, IP임.
- TCP/IP에서 IP가 바로 이 계층의 프로토콜을 나타냄.
- IP는 내보낸 패킷이 상대방이 잘 수신했는지, 여러 개의 패킷을 송신한 경우에는 순서대로 수신했는지에 대한 제어는 전혀 수행하지 않음.

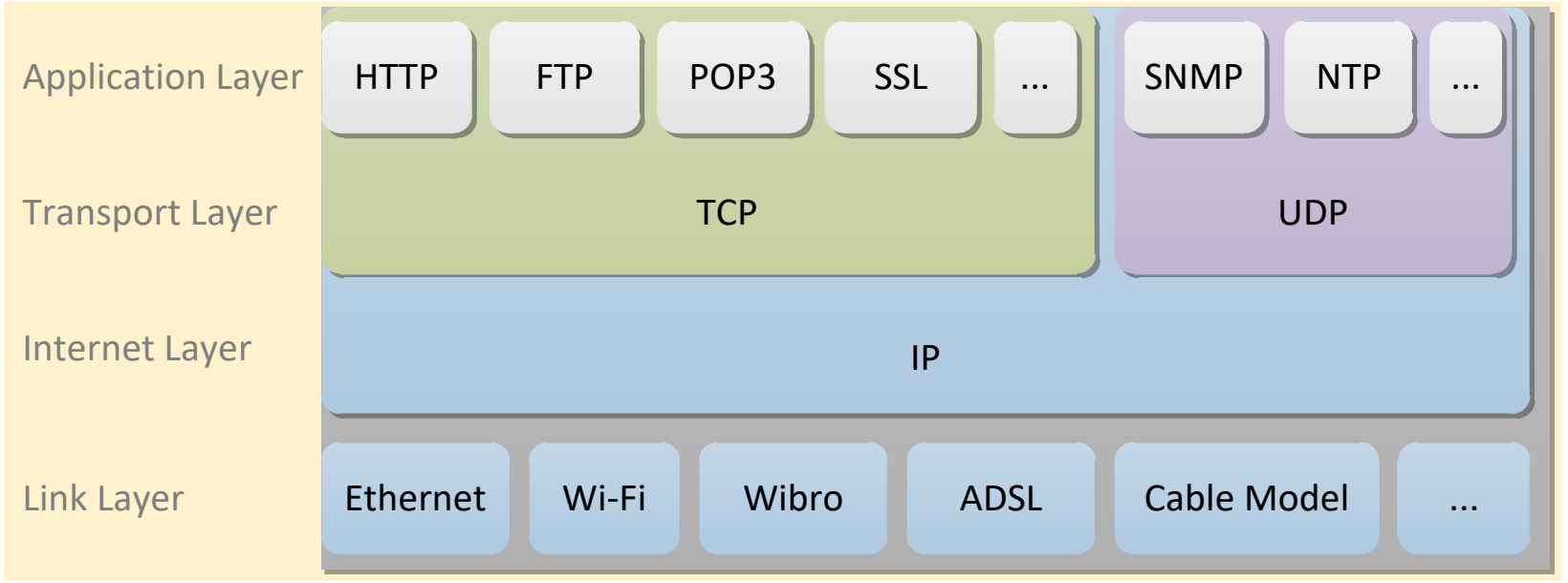
– Transport Layer

- 전송 계층(Transport Layer)에는 이름 그대로 패킷의 "운송"을 담당하는 프로토콜들이 정의됨.
- 전송 제어 프로토콜(Transport Control Layer)가 바로 이 계층에서 정의됨.
- TCP는 IP에서 수행하지 않는 패킷 송/수신 제어를 수행하여 신뢰성을 보완함.
- UDP(User Datagram Protocol)도 이 계층에서 선언되는데, TCP와 같이 신뢰성을 보장하지 않음. 신뢰성을 위한 작업을 하지 않음으로 인한 성능 향상 효과를 제공.

TCP/IP

– Application Layer

- 이 계층은 각 응용 프로그램 나름의 프로토콜들이 정의됨.
- HTTP, FTP, SNMP 등이 바로 이 계층에서 정의되는 프로토콜들임.
- 우리가 작성하는 네트워크 프로그래밍 코드 대부분이 바로 이 계층에 관한 것임.

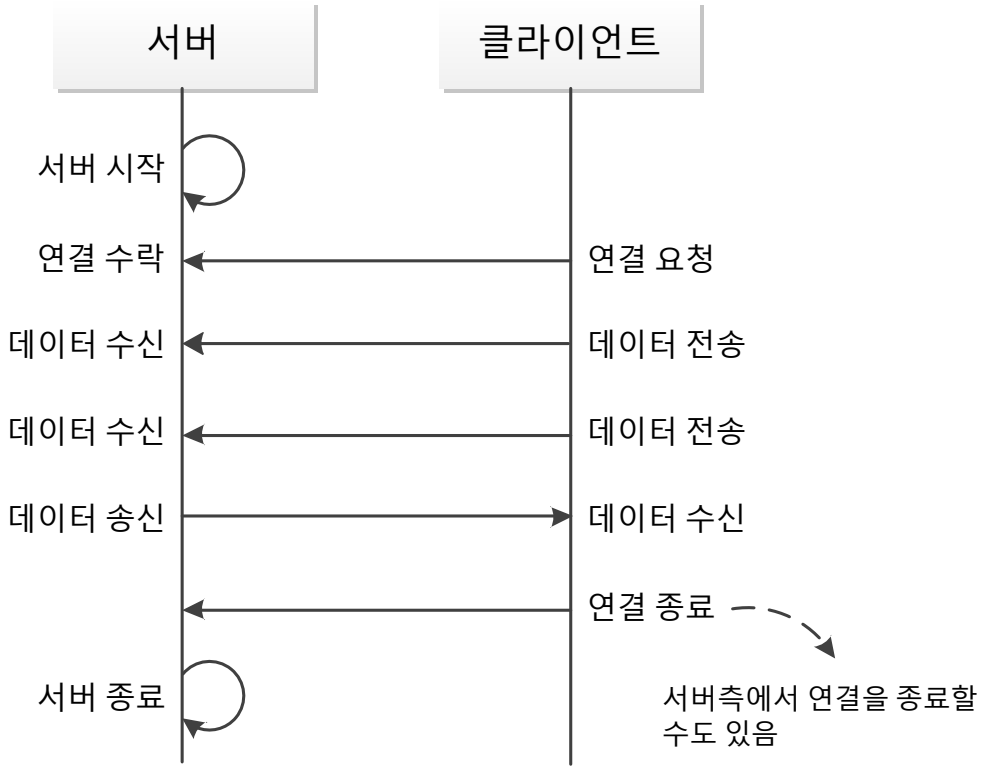


주소체계

- 인터넷에서 사용하는 주소를 일컬어 "IP 주소(Address)"라고 함.
  - IPV4 : 8비트 정수 4개로 구성(즉, 32비트)되는 주소.
    - 예) 211.56.101.37
  - IPV6 : 128비트의 주소 체계
    - IPV4 주소가 빠르게 고갈됨에 따라 새롭게 제정된 주소 체계.
    - 현재 빠르게 IPV6로의 주소체계 전환이 진행중임.
    - 예) 3FFE:FFFF:7654:FEDA:1245:BA98:3210:4562
- 포트(Port)
  - IP주소가 건물 주소라면, 포트는 출입구에 해당함.
  - 정수값을 가지며, 잘 알려진 포트 번호(Well-known Port)는 다음과 같음.
    - HTTP : 80, HTTPS : 443, FTP : 21, Telnet : 23, SMTP : 25

TCP/IP의 동작과정

TCP/IP는 서버/클라이언트 방식으로 동작함.
즉, 통신을 수행하는 양단 중 한 쪽에서는 한쪽에게 서비스를 제공해야 함.



TcpListener / TcpClient

- TcpListener와 TcpClient는 .NET 프레임워크가 TCP/IP 통신을 위해 제공하는 클래스

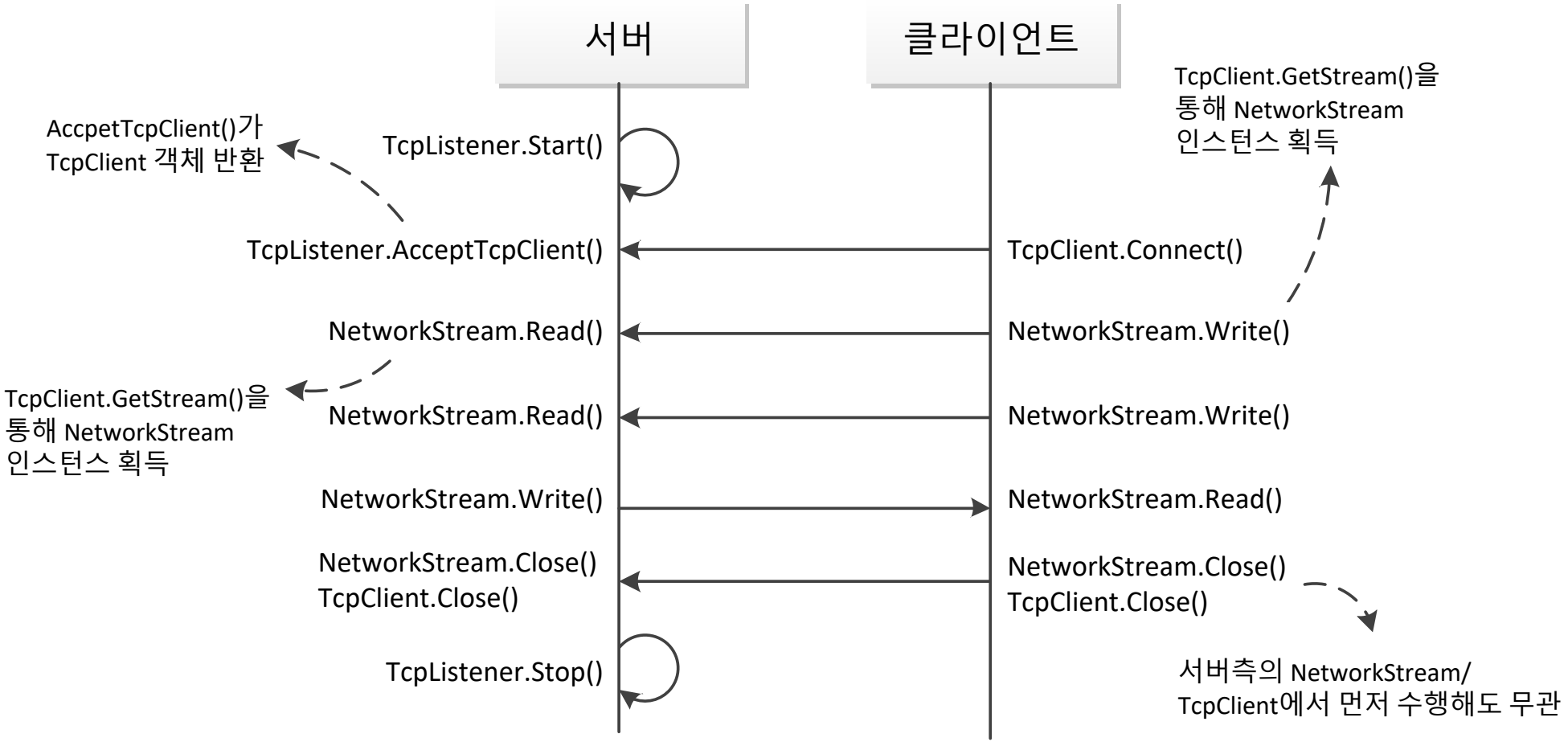
- TcpListener 클래스
  - 서버 애플리케이션에서 사용되며, 클라이언트의 연결 요청을 기다리는 역할을 수행

- TcpClient 클래스
  - 서버 애플리케이션과 클라이언트 애플리케이션 양쪽에서 사용됨.
  - 클라이언트에서는 TcpClient가 서버에 연결 요청을 하는 역할을 수행하며, 서버에서는 클라이언트의 요청을 수락하면 클라이언트와의 통신에 사용할 수 있는 TcpClient의 인스턴스가 반환됨.

- 서버와 클라이언트 각각이 갖고 있는 TcpClient는 GetStream()이라는 메소드를 갖고 있어서, 양쪽의 응용 프로그램은 이 메소드가 반환하는 NetworkStream 객체를 통해 데이터를 주고 받음.

- NetworkStream은 FileStream과 사용 방법이 동일

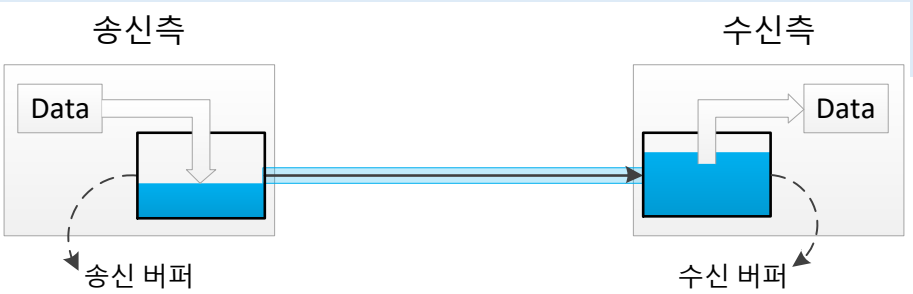
TcpListener / TcpClient



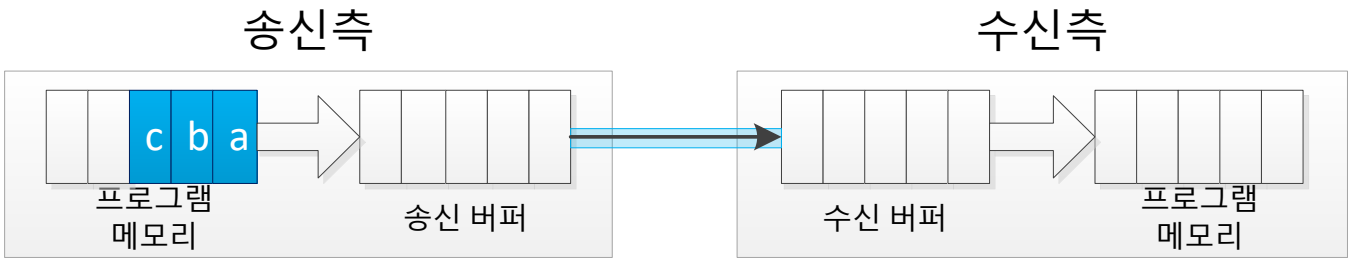
TcpListener / TcpClient

클래스	메쏘드	설명
TcpListener	Start()	연결 요청 수신 대기를 시작합니다.
	AcceptTcpClient()	클라이언트의 연결 요청을 수락합니다. 이 메소드는 TcpClient 객체를 반환합니다.
	Stop()	연결 요청 수신 대기를 종료합니다.
TcpClient	Connect()	서버에 연결을 요청합니다.
	GetStream()	데이터를 주고 받는데 사용하는 매개체인 NetworkStream을 가져옵니다.
	Close()	연결을 닫습니다.

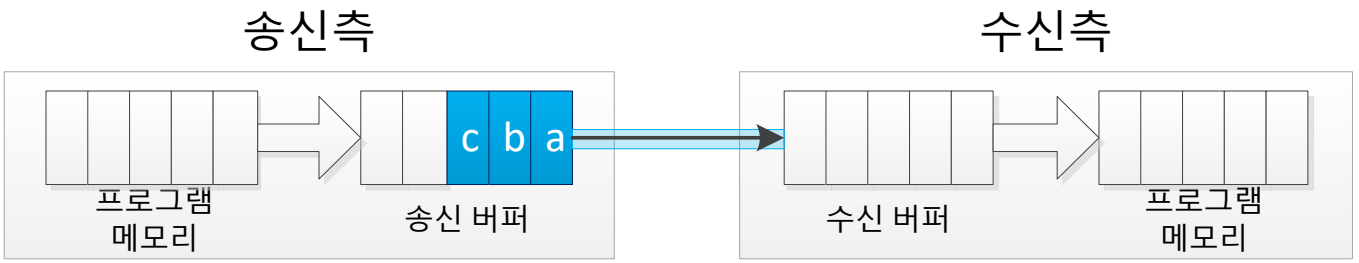
TCP 통신



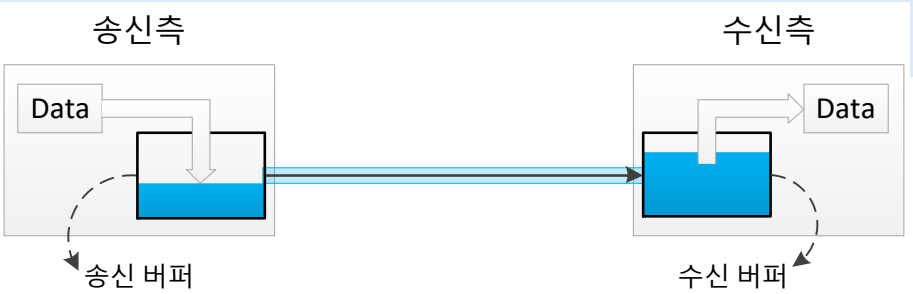
1) 두 어플리케이션이 TCP 연결을 맺고 있고, 송신 어플리케이션이 메모리에 들고 있는 데이터 'a', 'b', 'c'를 수신 어플리케이션에 보내려 한다고 해보자.
   - 그리고 'a', 'b', 'c'는 wBuffer라는 이름의 바이트 배열에 담겨 있다고 가정.



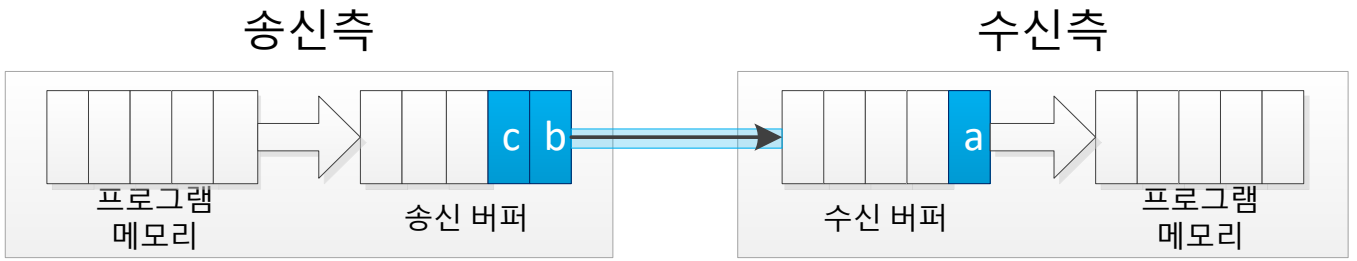
2) 송신측 어플리케이션에서 writer.Write(wBuffer, 0, 3)를 호출하면 데이터는 다음과 같이 어플리케이션의 메모리에서부터 송신 버퍼로 이동



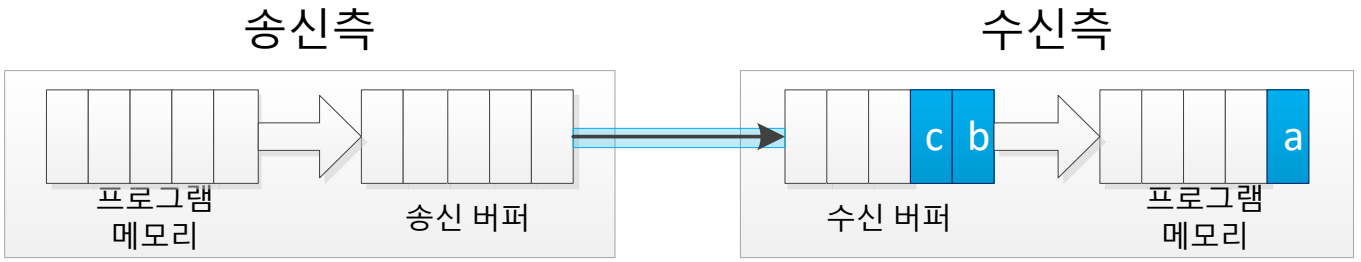
TCP 통신



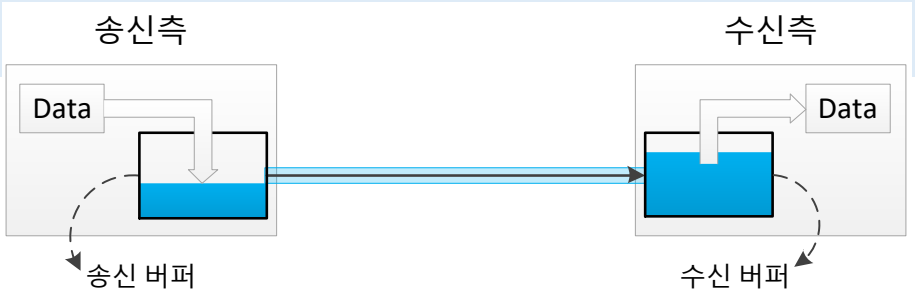
3) 운영체제는 송신버퍼에 있는 내용을 연결을 맺고 있는 수신측으로 보내기 시작.



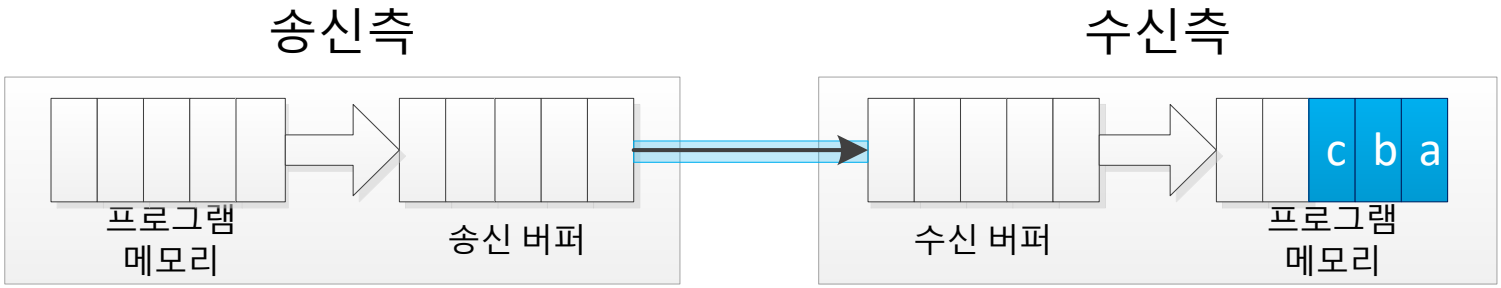
4) 한편, 수신측의 어플리케이션에서는 데이터를 담기 위한 rBuffer를 선언하고, reader.Read(rBuffer, 0, 16)을 호출.
이 코드는 16바이트를 읽어오려고 시도하지만 실제 수신 버퍼에는 'a' 하나밖에 없으므로 rBuffer에는 'a'가 담기고 Read() 메소드는 실제로 읽은 바이트 수 1을 반환.
한편, 그러는 동안 수신 버퍼에는 송신측에서 보낸 'b', 'c'가 도착함.



TCP 통신



5) 이번에도 수신측은 reader.Read(rBuffer, 0, 16)을 호출했는데 이번엔 'b', 'c'가 rBuffer에 담기고 Read() 메소드는 읽은 바이트 수 2를 반환함.
이렇게 해서 송신측의 프로그램 메모리에 있던 'a', 'b', 'c'가 모두 수신측의 프로그램 메모리로 전달됨.




Visual C#