**Advertise Mode (=Broadcast Mode)**

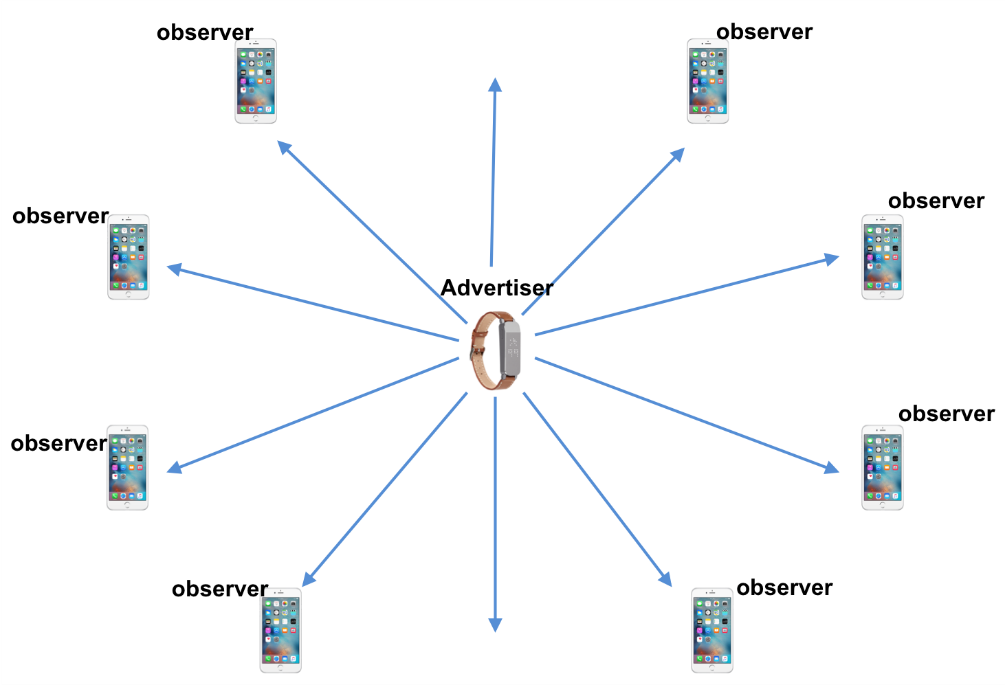
특정 디바이스를 지정하지 않고 주변의 모든 Device에게 Signal을 보낸다. 즉, 주변에 디바이스가 있건 없건, 다른 디바이스가 Signal을 듣는 상태이건 아니건, 자신의 Signal을 일방적으로 보냄.

* Advertiser(= Broadcaster) : Non-Connectable Advertising Packet을 주기적으로 보내는 디바이스

ex) Air pods, Apple Watch, Buds 등

* Observer: Advertiser가 Advertise를 Non-Connectable Advertising Packet을 듣기 위해 주기적으로 Scanning 하는 디바이스

ex) Air pods을 이용하기 위한 Cellphone, Apple pencil을 이용하기 위한 IPad

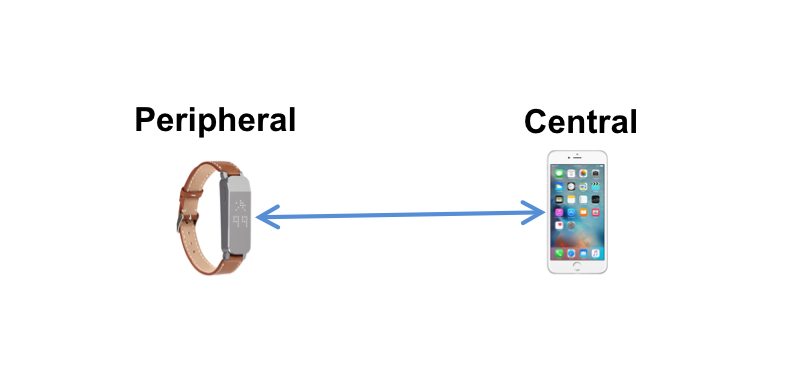


Advertise 방식은 한 번에 한 개 이상의 Device와 통신할 수 있는 유일한 방법이다. 주로 자신의 존재를 알리거나 적은 양의 User Data를 보낼 때도 사용한다.

한 번에 보내야 하는 Data 크기가 작다면, 굳이 오버헤드가 큰 Connection 과정을 거쳐서 Data를 보내기 보다는, Advertise를 이용하는 것이 더 효율적이기 때문이다. 게다가 전송할 수 있는 Data 크기 제한을 보완하기 위해 Scan Request, Scan Response을 이용해서 추가적인 데이터를 주고 받을 수 있다. Advertise 방식은 말 그대로 Signal을 일방적으로 뿌리는 것이기 때문에, 보안에 취약하다.

**Connection Mode**

양방향으로 데이터를 주고받거나, Advertising Packet으로만 전달하기에는 많은 양의 데이터를 주고받아야 하는 경우에는, Connection Mode로 통신을 한다. Advertise 처럼 1:N 방식이 아닌 1:1 방식으로 Device 간에 데이터 교환이 일어난다. Device간에 Channel hopping 규칙을 정해놓고 통신하기 때문에 Advertise보다 안전하다.

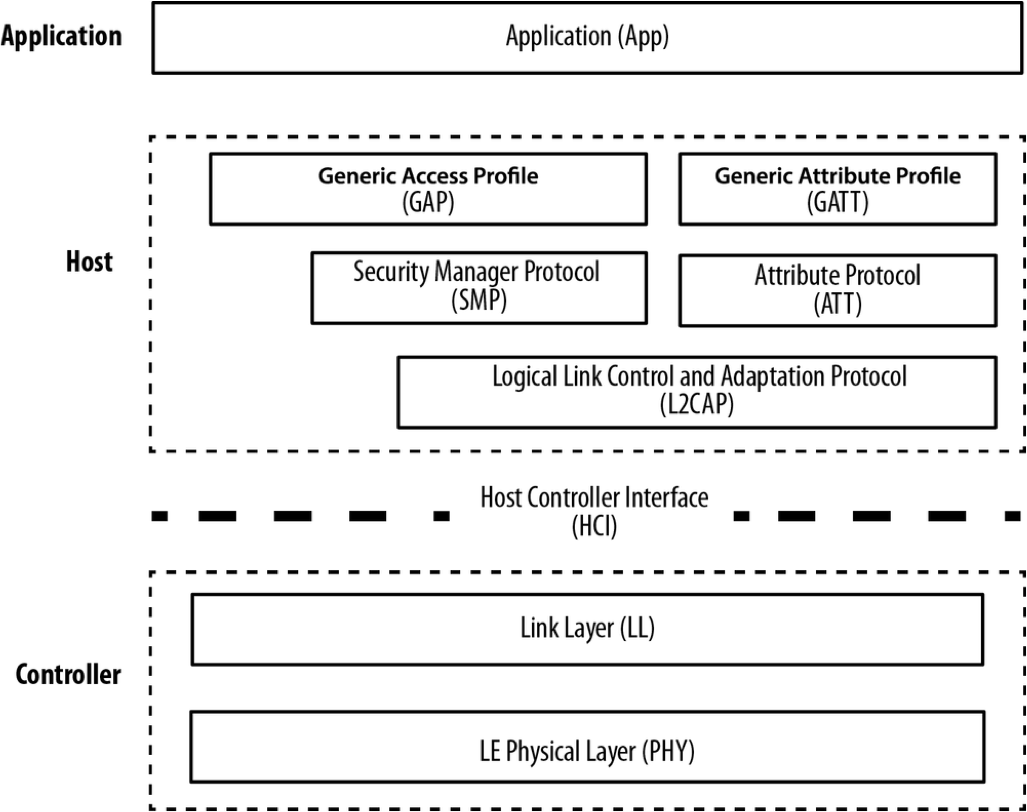


**Central (Master)** : Central Device는 다른 Device와 Connection을 맺기 위해, Connectable Advertising Signal을 주기적으로 스캔하다가, 적절한 디바이스에 연결을 요청한다. 연결 후에, Central Device는 Timing을 설정하고 주기적인 데이터 교환을 주도한다. 여기서 Timing이란, 두 Device가 매번 같은 Channel에서 Data를 주고 받기 위해 정하는 hopping 규칙이라고 생각하면 된다.

**Peripheral (Slave)** : Peripheral Device는 다른 Device와 Connection을 맺기 위해, Connectable Advertising Signal을 주기적으로 보낸다. 이를 수신한 Central Device가 Connection Request를 보내면, 이를 수학하여 Connection을 맺는다. Connection 후에 Central Device가 지정한 Timing에 맞추어 Channel을 같이 hopping을 하면서 주기적으로 데이터를 교환한다.

**Protocol Stack**

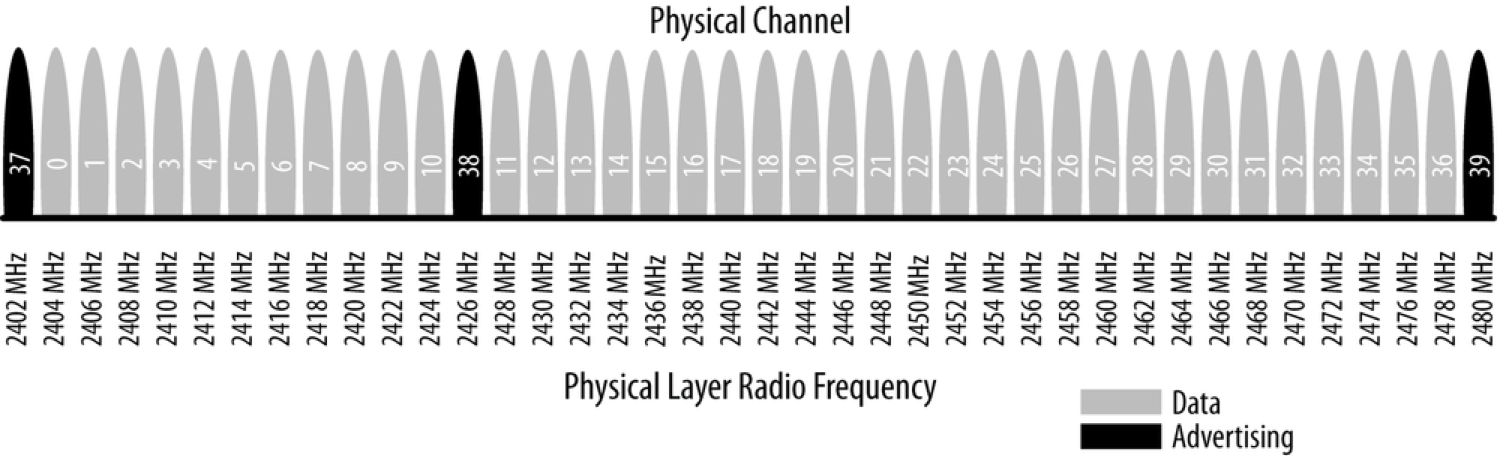
Device들은 Bluetooth로 통신을 하기 위한 Protocol Stack을 가지고 있다. 일반적으로 네트워크 통신을 하기 위해서는, 통신을 위한 규약인 Protocol을 정의해야 되는데, 이렇게 정의된 Protocol들을 층층이 쌓아놓은 그룹이 Protocol Stack이다. Bluetooth Signal Packet을 수신하거나 송신할 때, Protocol Stack을 거치면서 Packet들이 분석되거나 생성된다.



Protocol Stack은 가장 아랫단부터 크게 **Controller(Physical Layer, Link Layer), Host(GAP, GATT), Application** 로 나뉜다.

**Physical Layer**

Physical Layer에는 실제 Bluetooth Analog Signal과 통신할 수 있는 회로가 구성되어 있어서, Analog Signal -> Digital Signal or Digital Signal -> Analog Signal 로 바꾼다. 또한 Bluetooth에서는 2.4 GHz 밴드를 총 40개의 Channel로 나누어 통신을 한다. 40개 Channel 중 3개 Channel은 Advertising Channel 로써 각종 Advertising Packet을 비롯하여 Connection을 맺기 위해 주고 받는 Packet들의 교환에 이용된다. 나머지 37개의 Channel은 Data Channel로써 Connection 이후의 Data Packet 교환에 이용된다.



**Link Layer**

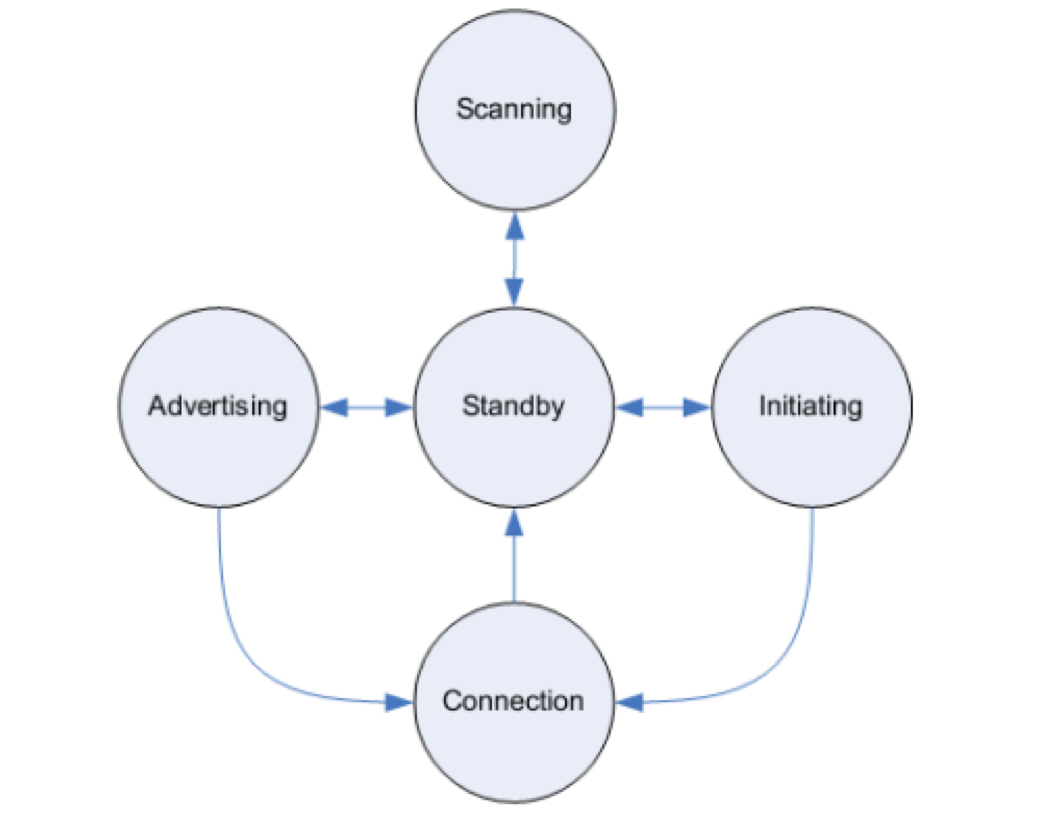
Physical Layer의 바로 위 단에는 Link Layer이 있다. Link Layer은 하드웨어와 소프트웨어의 조합으로 구성되어 있다. 하드웨어 단에서는 높은 컴퓨팅 능력이 요구되는 작업들이 처리되고, 소프트웨어 단에서는 디바이스의 연결 상태를 관리한다. 또한 통신하는데 있어서 디바이스의 Role을 정의하고 이에 따라 변경되는 State를 가지고 있다.

**Role**

* **Master :** 연결을 시도하고, 연결 후에 전체 connection을 관리
* **Salve :** Master의 연결 요청을 받고, Master의 Timing 규약을 따르는 역할
* **Advertiser :** Advertising Packet을 보내는 역할
* **Scanner :** Advertising Packet을 Scanning하는 역할.

**State**

* **Standby State :** Signal Packet을 보내지도, 받지도 않는 상태
* **Advertising State :** Advertising Packet을 보내고, 해당 Advertising Packet에 대한 상대 Device의 Response를 받을 수 있고 이에 응답할 수 있는 상태
* **Scanning State :** Advertising Channel에서 Scanning하고 있는 상태
* **Initiating State :** Advertiser의 Connectable Advertising Packet을 받고 난 후 Connection Request를 보내는 상태
* **Connection State :** connection 이후의 상태



**Generic Access Profile (GAP)**

Generic Access Profile(GAP)는 서로 다른 제조사가 만든 BLE Device들끼리 서로 호환되어 통신 할 수 있도록 해주는 주춧돌 역할을 한다. 즉, 어떻게 Device간에 서로를 인지하고, Data를 Advertising하고, Connection을 맺을 지에 대한 Framework를 제공한다.

**Role**

* **Broadcaster** : Link Layer에서 Advertiser 역할에 상응한다. 주기적으로 Advertising Packet을 보낸다.
* **Observer** : Link Layer에서 Scanner 역할에 상응한다. Broadcaster가 뿌리는 Advertising Packet에서 data를 얻는다. 온도센서로부터 온도데이터를 받아서 디스플레이에 나타내는 Tablet 컴퓨터의 역할이다.
* **Central** : Link Layer에서 Master 역할과 상응한다. Central 역할은 다른 디바이스의 Advertising Packet을 듣고 Connection을 시작할 때 시작된다. 좋은 성능의 CPU를 가지고 있는 스마트폰이나, Tablet 컴퓨터들의 역할이다.
* **Peripheral** : Link Layer에서 Slave 역할과 상응한다. Advertising Packet을 보내서 Central 역할의 디바이스가 Connection을 시작할 수 있도록 하게끔 유도한다. 센서 기능이 달린 디바이스들의 역할이다.

**Generic Attribute Profile (GATT)**

BLE Data 교환을 관리하는 GATT는 Device들이 Data를 발견하고 읽고, 쓰는 것을 가능하게 하는 기초적인 Data Model과 Procedure를 정의한다. 그래서 GATT는 최상위 Data Layer라고도 불린다. Gatt는 오직 Data의 Format 및 전달에 대해서만 처리한다. Connection Mode일 때, GATT Service와 Characteristic을 이용하여 양방향 통신을 하게 된다.

**Role**

* **Client** : Server에 Data를 요청한다. 하지만 처음에는 Server에 대해서 아는 것이 없기 때문에, Service Discovery라는 것을 수행한다. 이 후, Server에서 전송된 Response, Indication, Notification을 수신할 수 있다.
* **Server** : Client에게 Request를 받으면 Response를 보낸다. 또한 Client가 사용할 수 있는 User Data를 생성하고 저장 해놓는 역할을 한다.

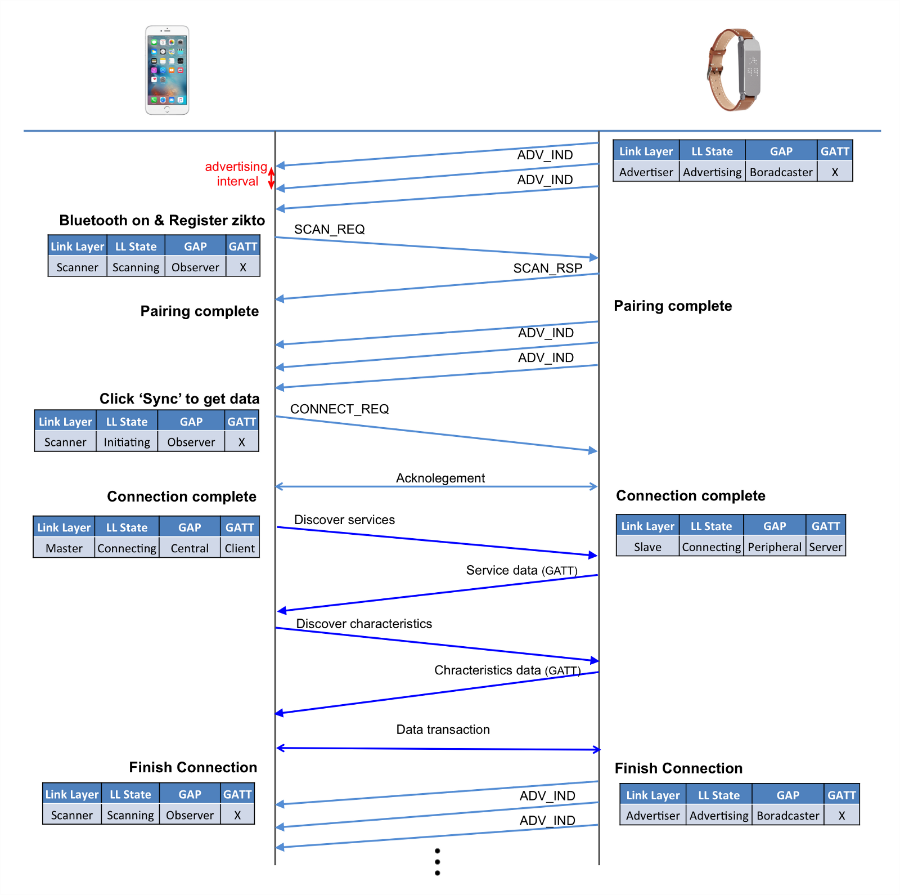
**BLE의 통신 방법 – 핵심은 Timing**

**Before Connection**

Connection 전, Device는 3개의 Advertising Channel을 이용해서 데이터를 주고 받는다고 했다. 이들은 3개의 Channel을 자신만의 Time interval로 hopping한다. 서로의 hopping 규칙이 일치하지 않기 때문에 Channel이 서로 엇갈리는 경우가 많을 것이다. 예를 들어, Advertiser는 1번 Channel에 Advertising Packet을 보냈는데, 같은 시간에 Scanner는 3번 Channel에 대해서 Scanning을 하게 되면 데이터 전달이 되지 않는 것이다. 하지만 이러한 hopping이 빠르게 자주 일어나기 때문에, 두 Device가 같은 Channel에 대해 Advertising와 Scanning이 발생하는 경우도 많이 생긴다. 이 경우에 서로 데이터를 주고 받을 수 있다.

**After Connection**

Connection 후, Advertising은 종료되고 기기들은 Central, Peripheral 중 하나의 역할을 하게 된다. Connection을 개시한 기기가 Central이며, Advertiser가 Peripheral이 된다. 그리고 두 Device는 엇갈렸던 hopping 규칙을 통일시킨다. 그렇게 함으로써, 매번 같은 채널을 동시에 hopping하면서 Signal을 주고 받을 수 있게 된다. 이는 둘 간의 Connection이 끊어질 때까지 지속된다.



**1) Zikto Walk가 Advertising Channel을 hopping하면서 Advertising Packet을 보낸다.(Zikto Walk의 Advertising Packet 유형은 ADV\_IND이다)**

**2) iPhone Bluetooth를 켠 후, Zikto 앱에 Zikto Walk를 등록한다. iPhone은 Advertising Channel을 hopping하면서 Scan을 하다가 연결하려는 Zikto의 디바이스 이름 등의 추가적인 정보를 얻기위해 SCAN\_REQ를 보낸다.**

**3) SCANREQ를 받은 Zikto Walk는 SCANRSP를 보낸다.**

**4) Pairing이 완료되고, Zikto Walk는 다시 Advertising Packet을 다시 일정 주기마다 보낸다.**

**5) iPhone에서 Zikto Walk로부터 걸음 수 등의 Data를 받기 위해 Sync 버튼을 누른다. 이 버튼을 누르면 iPhone은 CONNECT\_REQ를 보낸다.**

**6) Zikto와 iPhone은 서로 Acknowledging을 시작하고, timing 정보 등을 동기화 한다.**

**7) Connection이 완료된다.**

**8) Connection이 완료된 후, Service Data, Characteristic Data 등에 대한 Data 교환이 일어난다.**

**9) iPhone과 Zikto Walk간에 Data Sync가 완료되면, Connection이 해제되고, 다시 Advertising Packet을 보낸다. 이를 그림으로 표현하면 아래와 같다.**