XNode로 배우는

저전력 무선 네트워크 프로그래밍

7 Zigbee 통신

Zigbee 통신

- Zigbee 통신을 하기 위해서는 2개 이상의 XNode 필요
  - 각 XNode는 통신 전에 코디네이터 또는 네트워크 등록과 같은 설정 필요
    - PC와 XNode를 USB 케이블로 연결한 후 AT Command를 통해 설정
      - Coordinator operation
      - Router operation
      - Channel scanning
    - 설정이 완료되면 데이터 송수신 가능
  - pop 라이브러리의 xnode 모듈을 통해 AT Command 설정 및 데이터 송수신

xnode Module

- from pop import xnode
  - pop 라이브러리의 xnode 모듈 로드
  - xnode() : xnode모듈 생성자
  - atcmd(cmd, [value]) : AT Command로 XNode 설정 및 설정 상태 확인
    - cmd : 설정을 나타내는 문자열
    - value(optional) : 입력 시 cmd값에 입력, 입력하지 않을 시 cmd 설정값 반환
    - 전원 재부팅 이후에도 설정값이 유지되려면 "WR" 설정 전송필요
  - discover() : 동일 네트워크 내의 통신 가능 장치 검색 및 반환
    - sender_nwk : 16-bit network address
    - sender_eui64 : 8-byte bytes object with EUI-64 address
    - parent_nwk : coordinator, router에서 0xfffe로 설정된 값

xnode Module

- node_id : 디바이스의 NI 값 (최대 20자 문자열)
- node_type : 0=coordinator, 1=router
- device_type : 하드웨어 타입, 동일해야 통신 가능
- rssi : 신호 강도(dBm)

xnode Module

- Transmit(addr, message) : 메시지 전송
  - 성공적으로 전송되면 None 반환
    - ACK로 인해 전송이 실패한 경우, 수신자의 여유 버퍼 공간이 부족한 경우 전송된 패킷은 자동 삭제
  - addr : 목적지 장치 주소, 장치 주소를 직접 입력하면 unicast 가능
    - xnode.ADDR_BROADCAST를 사용하면 Broadcast로 전송
    - xnode.ADDR_COORDINATOR를 사용하면 coordinator로 전송
  - message : 전송할 메시지, str type

xnode Module

- receive() : 수신된 메시지 반환, 수신된 메시지가 없을 때는 None 반환
  - sender_eui64 : 64-bit address (bytes object) of the sending node
  - source_ep : source endpoint as an integer
  - dest_ep : destination endpoint as an integer
  - cluster : cluster id as an integer
  - profile : profile id as an integer
  - broadcast : True or False depending whether the frame was broadcast or unicast
  - payload : bytes object of the payload. This is a bytes object instead of a string

xnode Module

- Pop 라이브러리의 xnode모듈을 import
  - atcmd() 메소드의 인자에 "ID를 전달하면 "ID" 매개변수에 설정된 값 return

```
01: from pop import xnode
02:
03: xnode.atcmd("ID")
```

```
b'\x00\x00\x00\x00\x00\x00\t\x14'
```

xnode Module

- 매개변수 설정 또는 변경
  - atcmd()의 첫 번째 인자에 "ID", 두 번째 인자에 설정하고자 하는 값을 전달
  - 두 번째 인자를 생략하면 변경된 값 확인 가능

```
01: xnode.atcmd("ID",0x15)
02: xnode.atcmd("ID")
```

```
b'\x00\x00\x00\x00\x00\x00\x00\x15'
```

xnode Module

- Discover() 메소드 호출
  - 네트워크 내의 검색된 장치들이 반환됨.

```
01: for i in xnode.discover():
02:     print(i)
```

```
{'rssi': -14, 'node_id': '8D3C80', 'device_type': 1179648, 'parent_nwk': 65534,
'sender_nwk': 0, 'sender_eui64': b'\x00\x13\xa2\x00A\x8d<\x80', 'node_type': 0}
```

Coordinator operation

- Form a network
  - Coordinator
    - 네트워크에 대한 채널, PAN ID, 보안 정책, 스택 프로필 선택
    - 네트워크를 시작할 수 있는 유일한 장치 유형
      - 각 Zigbee 네트워크에는 Coordinator가 반드시 있어야 함
    - 네트워크를 시작한 후 새 장치가 네트워크에 연결 가능
    - 데이터 패킷을 라우팅하고 네트워크의 다른 장치와 통신 가능
    - 양호한 채널과 사용되지 않은 PAN ID에서 시작
      - Coordinator는 일련의 스캔을 수행하여 서로 다른 채널에서 RF 활동 (에너지 스캔)을 발견
      - 주변 작동 PAN (PAN 스캔)을 발견

Coordinator operation

- Channel selection
  - 네트워크를 시작할 때 Coordinator는 네트워크가 작동할 "양호한" 채널 선택
    - 여러 채널에서 에너지 스캔을 수행하여 에너지 수준을 감지
    - 시작할 잠재적 채널 목록에서 과도한 에너지 레벨이 있는 채널을 제거
    - 무작위로 채널을 선택하여 네트워크를 구성
    - "SD" (Scan Duration)
      - 에너지 스캔 중에 장치가 각 채널에 머무르는 시간 조정

Coordinator operation

- PAN ID selection
  - 에너지 스캔을 완료한 후 Coordinator는 잠재적인 채널 목록 스캔
    - 인접 PAN 목록 얻음
    - Coordinator는 각 잠재적 채널에서 비콘 요청 (브로드 캐스트) 전송
    - 네트워크에 가입 한 모든 Coordinator 및 Router는 Coordinator에게 비콘 요청 응답
      - 비콘에는 PAN 식별자 (16비트 및 64비트)를 포함하여 장치가 켜져 있는 PAN에 대한 정보 포함
    - 이 스캔 (잠재 채널에서 비콘 수집)을 일반적으로 활성 스캔 또는 PAN 스캔이라고 함
    - 채널 및 PAN 스캔을 완료한 후 시작할 임의 채널과 사용되지 않은 16비트 PAN ID 선택

Coordinator operation

- Persistent data
  - Coordinator가 네트워크를 시작하면 전원주기 동안 다음 정보 유지
    - PAN ID
    - Operating channel
    - Security policy and frame counter value
    - Child table (end device children that are joined to the coordinator)
    - Binding table
    - Group table

Coordinator operation

- Coordinator는 네트워크를 떠날 때까지 이 정보를 무기한 유지
- Coordinator가 네트워크를 떠나 새 네트워크를 시작하는 경우
  - 이전 PAN ID, 운영 채널, 링크 키 테이블 및 하위 테이블 데이터 손실

Coordinator operation

- Coordinator startup
  - Coordinator가 네트워크를 형성하는 데 사용하는 네트워크 형성 명령

Command	Description
CE	Coordinator 역할을 하고 네트워크를 형성하도록 지정하려면 1로 설정
ID	64비트 PAN ID를 결정하는 데 사용. 0으로 설정하면 임의의 64비트 PAN ID가 선택.
SC	Coordinator가 네트워크를 구성할 때 사용하는 스캔 채널 비트 마스크 결정. Coordinator는 활성화된 모든 SC 채널에서 에너지 스캔을 한 후 PAN ID 스캔 수행.
SD	검색 기간 또는 Router가 각 채널에서 비콘을 수신하는 시간을 설정

Coordinator operation

- 구성 변경은 마지막 변경 후 5초 동안 네트워크 형성 시작 지연
- Coordinator가 네트워크를 시작하는 경우
  - 네트워크 구성 설정 및 하위 테이블 데이터는 전원주기 동안 유지
  - 'WR' 파라미터 사용 시 전원 재인가해도 설정 정보 유지
- Coordinator가 네트워크를 성공적으로 시작한 경우
  - 다른 장치가 한동안 네트워크에 연결되도록 허용하고 "AI"를 0으로 설정

Command	Description	
AI (Association Indication)	0x00	Successfully formed or joined a Zigbee network.
	0x21	Scan found no PANs
	0x22	Scan found no valid PANs based on SC and ID settings.
	0x23	Valid PAN found, but joining is currently disabled
	0x24	No joinable beacons were found
	0xFF	Initialization time; no association status has been determined yet

Router operation

- Router 특징
  - Zigbee 네트워크에 참여하기 전에 유효한 Zigbee 네트워크 발견, 가입 필요
  - Router가 네트워크에 연결되면 새 장치가 네트워크에 연결되도록 할 수 있음
  - 데이터 패킷을 라우팅하고 네트워크의 다른 장치와 통신할 수 있음

Router operation

- Discover Zigbee networks
  - Router는 PAN (또는 활성) 스캔하여 근처의 Zigbee 네트워크를 검색
    - PAN 스캔 중 Router는 스캔 채널 목록의 첫 번째 채널에서 비콘 요청 (브로드 캐스트) 전송
    - Zigbee 해당 채널에서 작동하는 모든 인근 Coordinator 및 Router는 비콘 요청 응답
    - 비콘에는 PAN ID, 참여 허용 여부, 주변 장치가 있는 PAN에 대한 정보 포함
    - Router는 채널에서 수신된 각 비콘을 평가하여 유효한 PAN인지 확인
    - Router가 유효한 PAN을 찾지 못하면 스캔 채널 목록의 다음 채널에서 PAN 스캔을 수행
    - 유효한 네트워크를 찾거나 모든 채널이 스캔 될 때까지 스캔을 계속
    - Rounter가 모든 채널을 스캔하고 유효한 PAN을 찾지 못하면 모든 채널을 다시 스캔
    - 유효한 PAN이 연결 Router 범위 내에 있으면 일반적으로 몇 초 내에 PAN 검색

Router operation

- Join a network
  - Router가 유효한 네트워크를 발견한 경우
    - Zigbee 네트워크에서 가입을 요청하는 유효한 비콘을 보낸 장치에 연결 요청 송신
    - 결합을 허용하는 장치는 결합을 허용하거나 거부하는 연관 응답 프레임 송신
    - Router가 네트워크에 가입하면 가입을 허용한 장치로부터 16비트 주소 수신
    - 결합을 허용한 장치는 16비트 주소를 임의로 선택

Router operation

- Persistent data
  - Router가 네트워크에 연결되면 전원주기 동안 다음 정보 유지
    - PAN ID
    - Operating channel
    - Security policy and frame counter value
    - Child table (end device children that are joined to the coordinator)
    - Binding table
    - Group table

Router operation

- Router는 네트워크를 떠날 때까지 이 정보를 무기한 유지
- Router가 네트워크를 떠나는 경우
  - 이전 PAN ID, 운영 채널, 하위 테이블 데이터 손실

Router operation

- Router joining
  - Router의 전원을 켰을 경우
    - 유효한 Zigbee 네트워크에 아직 연결되어 있지 않은 경우
      - 유효한 Zigbee 네트워크를 찾아 연결 시도
  - Router 연결 프로세스 제어 명령

Command	Description
CE	Router 역할로 지정하려면 0으로 설정
ID	64비트 PAN ID를 결정하는 데 사용. 0으로 설정하면 임의의 64비트 PAN ID가 선택.
SC	Router가 유효한 네트워크를 찾기 위해 스캔하는 채널을 결정하는 스캔 채널 비트 마스크 설정. Coordinator의 SC와 일치하도록 Router의 SC를 설정.
SD	검색 기간 또는 Router가 각 채널에서 비콘을 수신하는 시간을 설정
JV	전원 인가 시 네트워크 연결확인 여부 설정. 1로 설정되어 있으면 네트워크에 처음 가입할 때 Coordinator의 64비트 주소 검색을 시도

Router operation

- 구성 변경은 마지막 변경 후 5초 동안 네트워크 형성 시작 지연
- Router가 네트워크에 연결되는 경우
  - 네트워크 구성 설정 및 하위 테이블 데이터는 전원주기 동안 유지
  - 연결이 실패하면 AI 명령 레지스터에서 마지막 연결 시도의 상태를 읽음

Command	Description	
AI (Association Indication)	0x00	Successfully formed or joined a Zigbee network.
	0x21	Scan found no PANs
	0x22	Scan found no valid PANs based on SC and ID settings.
	0x23	Valid PAN found, but joining is currently disabled
	0x24	No joinable beacons were found
	0xFF	Initialization time; no association status has been determined yet

Router operation

- Router네트워크 형성 명령 값이 변경된 경우
  - Router는 현재 네트워크를 떠나 새로운 유효한 네트워크를 발견하고 연결 시도
- Router가 네트워크에 성공적으로 연결된 경우
  - 장치가 한동안 네트워크에 연결되도록 허용하고 "AI"를 0으로 설정
- 'WR' 파라미터
  - 현재 설정된 파라미터들이 전원을 재시작하더라도 유지

Channel scanning

- Router는 가입할 유효한 네트워크를 찾기 위해 채널 스캔
  - 조인 시도 시작
    - 장치는 SC (scan channel) 비트 마스크에 지정된 최하위 채널에서 비콘 요청 전송
  - 장치가 채널에서 유효한 PAN을 찾으면
    - 해당 채널에서 PAN에 참여하려고 시도

Channel scanning

- 장치가 채널에서 유효한 PAN을 찾지 못한 경우
  - SC 비트 마스크에서 다음 상위 채널에서 스캔 시도
  - 장치는 유효한 PAN을 찾거나 모든 채널이 스캔 될 때까지 SC 비트 마스크의 각 채널 계속 스캔
  - 장치가 모든 채널을 스캔하면 다음 연결 시도는 SC 비트 마스크에 지정된 최하위 채널에서부터 스캔을 시작

Channel scanning

- Manage multiple Zigbee networks
  - 일부 애플리케이션에서는 여러 Zigbee 네트워크가 서로 인접 가능
    - ID(PAN ID) 매개변수를 0이 아닌 값으로 설정
    - 장치는 동일한 PAN ID를 가진 네트워크에만 연결

AT Command

- CE (Coordinator Enable -> Device Role)
  - 장치가 Coordinator/Router 역할을 할지 결정
  - 네트워크 연결 후 CE를 변경하면 장치가 네트워크를 벗어남
  - Parameter range : 0x00-0x01

Parameter	Description
0x01	'Coordinator'
0x00	'Router'

AT Command

- CE

```
01:  from pop import xnode
02:
03:  xnode.atcmd('CE', 1)
04:  print('Device Role: ', xnode.atcmd('CE'))
05:  xnode.atcmd('CE', 0)
06:  print('Device Role: ', xnode.atcmd('CE'))
```

01: pop 라이브러리에서 xnode class import
03: Device Role을 Coordinator로 변경
05: Device Role을 Router로 변경

```
Device Role:  1
Device Role:  0
```

AT Command

- ID (->Extended PAN ID(EPID))
  - 네트워크를 형성하거나 연결할 때 사용되는 사전 구성된 확장 8Byte 확장 PAN ID
  - ID는 OP (Operating Pan) 값이 일치하는 네트워크로만 연결 제한
  - Coodinator
    - ID를 0으로 설정하면 임의의 확장 PAN ID가 생성
  - Router
    - ID가 0으로 설정되면 장치는 열려 있는 네트워크에 연결 시도
  - 네트워크 연결 후 ID를 변경하면 장치가 네트워크를 벗어남
  - Parameter range : 0x00-0x0FFFFFFFFFFFFFFF

AT Command

```
01:  from pop import xnode
02:
03:  xnode.atcmd('ID', 0x15)
04:  print('my Extended PAN ID:', xnode.atcmd('ID'))
05:  xnode.atcmd('ID', 0xFF)
06:  print('my Extended PAN ID:', xnode.atcmd('ID'))
```

```
my Extended PAN ID: b'\x00\x00\x00\x00\x00\x00\x00\x15'
my Extended PAN ID: b'\x00\x00\x00\x00\x00\x00\x00\xff'
```

AT Command

- JV (Coordinator Join Verification)
  - Router가 주변에 Coordinator가 있는지 확인 후 해당 네트워크에 참여
  - Parameter range : 0x00-0x01
  - JV = 1인 경우
    - Router는 네트워크에 연결하거나 전원을 껐다 켤 때 Coordinator가 작동 채널에 있는지 확인
    - Coordinator가 감지되지 않으면 Router는 현재 채널을 떠나 새 PAN에 참여하려고 시도
  - JV = 0인 경우
    - Coordinator가 감지되지 않더라도 Router는 현재 채널에서 계속 작동

Parameter	Description
0x00	No coordinator verification
0x01	Coordinator verification enabled

AT Command

```
01:  from pop import xnode
02:
03:  xnode.atcmd('JV', 0x01)
04:  print('Coordinator Join Verification:', xnode.atcmd('JV'))
05:  xnode.atcmd('JV', 0x00)
06:  print('Coordinator Join Verification:', xnode.atcmd('JV'))
```

```
Coordinator Join Verification: 1
Coordinator Join Verification: 0
```

AT Command

- NI (Node Identifier)
  - 장치의 사용자 정의 이름
  - 네트워크에서 장치를 쉽게 식별하기 위해 문자열을 네트워크 검색 명령과 함께 사용 가능
  - 문자열과 함께 ND(Network Discovery) 명령을 인수로 사용하여 네트워크 검색 결과 필터링.
  - 문자열과 DN (Discover Node) 명령을 인수로 사용하여 일치하는 NI 문자열이 있는 노드의 64 비트 주소를 확인.
    - Parameter range : 0-20바이트 길이의 대소 문자를 구분하는 ASCII 인쇄 가능 문자열. 캐리지 리턴 또는 쉼표는 자동으로 명령 종료.

AT Command

```
01:  from pop import xnode
01:
02:  xnode.atcmd('CE', 0)
03:  xnode.atcmd('NI', 'Coordinator')
04:  print('I am', xnode.atcmd('NI'))
05:  xnode.atcmd('CE', 1)
06:  xnode.atcmd('NI', 'Router')
07:  print('I am', xnode.atcmd('NI'))
```

01: pop 라이브러리에서 xnode 모듈 로드
03: Device Role을 Coordinator로 변경
04: Node Identifier 값을 'Coordinator'로 변경
06: Device Role을 Router로 변경
07: Node Identifier 값을 'Router'로 변경

```
I am Coordinator
I am Router
```

AT Command

- AI (Association Indication)
  - 마지막 노드 연결 요청에 대한 정보를 읽음
  - 연결 시도 중에 AI를 쿼리하여 현재 상태 식별
  - Parameter range : 0x00-0xFF

Command	Description	
AI (Association Indication)	0x00	Successfully formed or joined a Zigbee network.
	0x21	Scan found no PANs
	0x22	Scan found no valid PANs based on SC and ID settings.
	0x23	Valid PAN found, but joining is currently disabled
	0x24	No joinable beacons were found
	0xFF	Initialization time; no association status has been determined yet

AT Command

```
01:  from pop import xnode
02:  ai = xnode.atcmd('AI')
03:  if ai == 0x00:
04:      print("Successfully formed or joined a Zigbee network.")
05:  if ai == 0x21:
06:      print("Scan found no PANs")
07:  if ai == 0x22:
08:      print("Scan found no valid PANs based on SC and ID settings.")
09:  if ai == 0x23:
10:      print("Valid PAN found, but joining is currently disabled")
11:  if ai == 0x24:
12:      print("No joinable beacons were found")
13:  if ai == 0xFF:
14:      print("no association status has been determined yet")
```

01: pop 라이브러리에서 xnode class import
02: ai 변수에 Assocication Indication 값 할당
03: 상단의 표에 있는 16진수 코드와 비교 출력

```
Successfully formed or joined a Zigbee network.
```

AT Command

- OP (Operating Extended PAN ID)
  - 연결된 네트워크의 64비트 확장 PAN ID 반영
  - Parameter range : 0x00-0xFFFFFFFFFFFFFFFF

```
01:  from pop import xnode
02:  print(xnode.atcmd('OP'))
```

01: pop 라이브러리에서 xnode class import
02: Operating Extended PAN ID출력

```
b'\x00\x00\x00\x00\x00\x00\x00\x15'
```

AT Command

- OI (Operating 16-bit PAN ID)
  - 연결된 네트워크의 16비트 PAN ID 반영
  - Parameter range : 0x00-0xFFFF

```
01:  from pop import xnode
02:  print(hex(xnode.atcmd('OI')))
```

01: pop 라이브러리에서 xnode class import
02: Operating 16-bit PAN ID출력

```
0xdd07
```

AT Command

- CH (Operating Channel)
  - 연결된 네트워크의 채널 번호 반영
  - 채널은 IEEE 802.15.4 채널 번호로 표시
  - 값 0은 장치가 PAN에 연결되지 않았으며 어떤 채널에서도 작동하지 않음을 의미
  - Parameter range : 0, 0x0B - 0x1A (Channels 11 through 26) [read-only]

```
01:  from pop import xnode
02:  print(xnode.atcmd('CH'))
```

01: pop 라이브러리에서 xnode class import
02: Operating Channel 출력

AT Command

- SH (Serial Number High)
  - XNode에 할당된 고유한 IEEE 64비트 확장 주소의 상위 32비트 표시
  - 이 값은 읽기 전용이며 변경되지 않음.
  - Parameter range : 0x0013A200 - 0x0013A2FF [read-only]

```
01:  from pop import xnode
02:  print(xnode.atcmd('SH'))
```

01: pop 라이브러리에서 xnode class import
02: 64비트 확장 주소의 상위 32비트 출력

```
b'\x00\x13\xa2\x00'
```

AT Command

- SL (Serial Number Low)
  - XNode에 할당 된 고유한 IEEE 64비트 확장 주소의 하위 32비트 표시
  - 이 값은 읽기 전용이며 변경되지 않음.
  - Parameter range : 0x00 - 0xFFFFFFFF [read-only]

```
01:  from pop import xnode
02:  print(xnode.atcmd('SL'))
```

01: pop 라이브러리에서 xnode class import
02: 64비트 확장 주소의 하위 32비트 출력

```
b'A\xaf\x03\x7f'
```

AT Command

- MY (16-bit Network Address)
  - 연결 시 네트워크 관리자가 임의로 할당한 장치의 16비트 네트워크 주소를 읽음
  - 0xFFFE 값은 장치가 Zigbee 네트워크에 연결되지 않았음을 의미
  - Parameter range : 0x00 - 0xFFFF [read-only]

```
01:  from pop import xnode
02:  print("My Network Address", hex(xnode.atcmd('MY')))
```

01: pop 라이브러리에서 xnode class import
02: 16비트 네트워크 주소를 16진수로 출력

```
My Network Address 0x2e91
```

AT Command

- MP (16-bit Parent Network Address)
  - 상위장치의 16비트 네트워크 주소를 읽음
  - 0xFFFE 값은 장치에 상위 장치가 없거나 최종 장치로 구성되지 않았음을 의미
  - Parameter range : 0x00 - 0xFFFE [read-only]

```
01:  from pop import xnode
02:  print("Parent Network Address", hex(xnode.atcmd('MP')))
```

01: pop 라이브러리에서 xnode class import
02: 상위 장치의 16비트 네트워크 주소를 16진수로 출력

```
Parent Network Address 0xfffe
```

AT Command

- DH (Destination Address High)
  - 데이터 전송 대상의 64비트 주소의 상위 32비트를 설정하거나 읽음
  - DH를 DL과 결합하면 장치가 데이터 전송에 사용하는 64비트 대상 주소 정의
  - 이 대상 주소는 대상 장치의 일련번호 (SH + SL)에 해당
  - 0x000000000000FFFF is a broadcast address (DH = 0, DL = 0xFFFF).
  - 0x0000000000000000 addresses the network coordinator.
  - Parameter range : 0x00 - 0xFFFFFFFF

AT Command

```
01:  from pop import xnode
02:  print("Destination Address High", xnode.atcmd('DH'))
03:  xnode.atcmd('DH', 0xdeadbeef)
04:  print("Destination Address High", xnode.atcmd('DH'))
```

01: pop 라이브러리에서 xnode class import
02: 데이터 전송 대상의 64비트 주소의 상위 32비트를 16진수로 출력
03: DH command 변경
04: 변경된 데이터 전송 대상의 64비트 주소의 상위 32비트를 16진수로 출력

```
Destination Address High b'\x00\x00\x00\x00'
Destination Address High b'\xde\xad\xbe\xef'
```

AT Command

- DL (Destination Address Low)
  - 데이터 전송 대상의 64 비트 주소의 하위 32비트를 설정하거나 읽음
  - DH를 DL과 결합하면 장치가 데이터 전송에 사용하는 64비트 대상 주소 정의
  - 이 대상 주소는 대상 장치의 일련번호 (SH + SL)에 해당
  - 0x000000000000FFFF is a broadcast address (DH = 0, DL = 0xFFFF).
  - 0x0000000000000000 addresses the network coordinator.
  - Parameter range : 0x00 - 0xFFFFFFFF

AT Command

```
01:  from pop import xnode
02:  print("Destination Address Low", xnode.atcmd('DL'))
03:  xnode.atcmd('DL', 0xaabbccdd)
04:  print("Destination Address Low", xnode.atcmd('DL'))
```

01: pop 라이브러리에서 xnode class import
02: 데이터 전송 대상의 64비트 주소의 상위 32비트를 16진수로 출력
03: DL command 변경
03: 변경된 데이터 전송 대상의 64비트 주소의 상위 32비트를 16진수로 출력

```
Destination Address Low b'\x00\x00\x00\x00'
Destination Address Low b'\xaa\xbb\xcc\xdd'
```

AT Command

- PL (TX Power Level)
  - 장치의 데이터 송신 파워 설정
  - CH = 0x1A에서 작동하는 경우 출력 전력은 제한되며 8dBm을 초과할 수 없음
  - Parameter range : 0x00 - 0x04

Parameter	dBm
0x00	-5dBm
0x01	+3dBm
0x02	+8dBm
0x03	+15dBm
0x04	+19dBm

AT Command

```
01:  from pop import xnode
02:
03:  for i in range(0x05):
04:      xnode.atcmd('PL', i)
05:      print("TX Power Level 0x%02x" % xnode.atcmd('PL'))
```

01: pop 라이브러리에서 xnode class import
03: 0x00에서 0x04까지 반복
04: PL command 변경
05: 변경된 송신 파워를 16진수로 출력

```
TX Power Level 0x00
TX Power Level 0x01
TX Power Level 0x02
TX Power Level 0x03
TX Power Level 0x04
```

AT Command

- PP(Output Power in dBm)
  - 작동 출력 전력을 표시
  - 반환된 값은 dBm이며 음수 값은 2의 보수로 표시
    - ex. -5dBm = 0xFB.
  - Parameter range : 0x00 - 0xFF [read-only]

AT Command

```
01:  from pop import xnode
02:
03:  for i in range(0x05):
04:      xnode.atcmd('PL', i)
05:      pp = xnode.atcmd('PP')
06:      if pp >= 127:
07:          pp = -(0x100 - pp)
08:      print("Output Power in dBm: %d dBm" % pp)
```

01: pop 라이브러리에서 xnode class import
03: 0x00에서 0x04까지 반복
04: PL command 변경
05: 작동 출력 전력 반환
06: 작동 출력 전력이 음수인지 검사
07: 2의 보수를 음수로 변환
08: 변환된 dBm을 정수로 출력

```
Output Power in dBm: -5 dBm
Output Power in dBm: 3 dBm
Output Power in dBm: 8 dBm
Output Power in dBm: 15 dBm
Output Power in dBm: 19 dBm
```

AT Command

- DB(Last Packet RSSI)
  - 이 명령은 마지막으로 수신된 RF 데이터 패킷 수신 신호 강도 반환
  - DB 명령은 마지막 홉의 신호 강도만 나타냄
  - 멀티 홉 링크에 대한 정확한 품질 측정을 제공하지 않음
  - DB 명령 값은-dBm 단위로 측정
    - DB가 0x50을 반환하면 마지막으로 수신된 패킷의 RSSI는 -80dBm
  - Parameter range : 0x00 - 0xFF [read-only]

AT Command

```
01:  from pop import xnode
02:  db = -(xnode.atcmd('DB'))
03:  print('Last Packet RSSI %d dBm' % db)
```

01: pop 라이브러리에서 xnode class import
02: DB command 반환 후 음수로 변환
04: Last Packet RSSI 출력

```
Last Packet RSSI -6 dBm
```

AT Command

- SC (Scan Channels)
  - 로컬 장치에서 활성 스캔을 수행할 때 사용되는 채널
  - 활성 스캔은 네트워크가 형성될 때마다 또는 결합 시도 이전에 수행
  - 네트워크 연결 후 SC를 변경하면 작동 채널 (CH) 이 SC 마스크에서 제외된 경우 장치가 네트워크를 벗어날 수 있음
    - 26번 채널은 출력 전력이 +8dBm으로 제한되므로 되도록 사용을 피하도록 함
  - Parameter range : 0x00 - 0xFFFF [bit field]

AT Command

- SC (Scan Channels)

Bit	IEEE 802.15.4 Channel	Frequency (GHz)
0	11 (0x0B)	2.405
1	12 (0x0C)	2.410
2	13 (0x0D)	2.415
3	14 (0x0E)	2.420
4	15 (0x0F)	2.425
5	16 (0x10)	2.430
6	17 (0x11)	2.435
7	18 (0x12)	2.440
8	19 (0x13)	2.445
9	20 (0x14)	2.450
10	21 (0x15)	2.455
11	22 (0x16)	2.460
12	23 (0x17)	2.465
13	24 (0x18)	2.470
14	25 (0x19)	2.475
15	26 (0x1A)	2.480

AT Command

- WR (Write)
  - 매개 변수 값을 비휘발성 플래시 메모리에 즉시 기록하여 전원주기 동안 지속하도록 함
  - 운영 네트워크 매개 변수는 영구적이며 장치가 네트워크에 다시 연결되는데 WR 명령이 필요하지 않음

AT Command

- 아래 코드 실행

```
01:  from pop import xnode
02:  xnode.atcmd('NI', 'Node01')
03:  xnode.atcmd('WR')
```

01: pop 라이브러리에서 xnode class import
02: Node Identifier를 Node01로 설정
03: 플래시 메모리에 기록

- 전원 재연결 후 아래 코드 실행

```
01:  from pop import xnode
02:  print('I am', xnode.atcmd('NI'))
```

01: pop 라이브러리에서 xnode class import
02: 플래시 메모리에 기록된 Node Identifier 출력

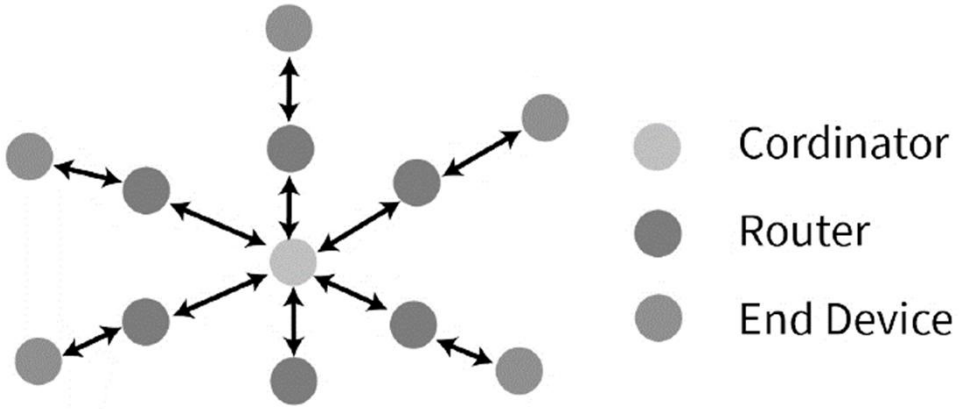
```
I am Node01
```

Transmission - 통신 방식

- Zigbee 데이터 패킷을 unicast 또는 broadcast 전송 가능
  - unicast 전송
    - 하나의 소스 장치에서 하나의 대상 장치로 데이터를 라우팅
  - broadcast 전송
    - 네트워크의 여러 또는 모든 장치로 전송

Transmission - 통신 방식

- Broadcast transmissions
  - 모든 노드가 전송을 수신하도록 전체 네트워크에 전파



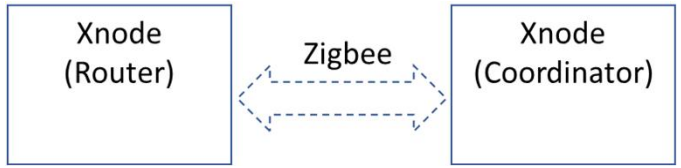
Transmission - 통신 방식

- Unicast transmissions
  - unicast 전송은 한 소스 장치에서 다른 대상 장치 1대로 전송
  - 대상 장치는 소스의 바로 이웃이거나 여러 홉 떨어져 있을 수 있음

데이터 송신, 수신

준비물

준비물	
1	PC 1ea
2	XNode 2ea
3	Micro type USB cable 1ea



예제 구성

- 예제 진행을 위해 USB> Library > CORE > lib 폴더를 XNode B Type에 복사하여 사용해야 함.

NI	추가 라이브러리
Coordinator	USB > Library > CORE > lib > pop.py, core_b.py
Router	USB > Library > CORE > lib > pop.py, core_b.py

데이터 송신, 수신

- XNode 설정
  - 데이터 송신, 수신하기 위해 1개의 Router와 1개의 Coordinator 필요
    - 한 공간 안에 반드시 단 1개의 Coordinator만 있어야 함
  - Xnode2개를 다음과 같이 설정 후 'WR'command 실행

Command	Coordinator	Router
NI	'Coordinator'	'Router'
CE	0x01	0x00
ID	0x15	0x15
JV	-	0x01
Role	수신	송신

데이터 송신, 수신

- 모든 설정이 완료된 후 reset 버튼을 눌러 네트워크를 검색
- 코드에서는 Coordinator 장치를 수신 장치, Router 장치를 송신장치로 사용

Coordinator

```
01:  from pop import xnode
02:
03:  xnode.atcmd('NI', 'Coordinator')
04:  xnode.atcmd('CE', 0x01)
05:  xnode.atcmd('ID', 0x15)
06:  xnode.atcmd('WR')
```

Router

```
01:  from pop import xnode
02:
03:  xnode.atcmd('NI', 'Router')
04:  xnode.atcmd('CE', 0x00)
05:  xnode.atcmd('ID', 0x15)
06:  xnode.atcmd('JV', 0x01)
07:  xnode.atcmd('WR')
```

데이터 송신, 수신

- 데이터 수신(Coordinator)
  - 수신된 메시지를 receive()로 읽은 후 표준 출력 버퍼로 출력

```
01:  from pop import xnode
02:  import time
03:
04:  print("Receiving data...")
05:  print("Press CTRL+C to cancel.")
06:
07:  while True:
08:      p = xnode.receive()
09:      if p:
10:          print(p)
11:      else:
12:          time.sleep(0.5)
```

데이터 송신, 수신

- Coordinator 장치에서 코드 실행
  - 0.5초 단위로 수신된 메시지가 있는지 확인하여 그 결과에 따라 terminal에 출력

```
Receiving data...
Press CTRL+C to cancel.
{'profile': 49413, 'dest_ep': 232, 'broadcast': True, 'sender_nwk': 25611, 'source
_ep': 232, 'payload': b'hello', 'sender_eui64': b'\x00\x13\xa2\x00A\xae\\\x90', 'c
luster': 17}
```

데이터 송신, 수신

- 데이터 송신(Router)
  - Broadcast로 데이터를 송신하는 코드
  - xnode.transmit()의 첫 번째 인자에 xnode.ADDR_BROADCAST 전달
  - Broadcast로 데이터 송신

```
01:     from pop import xnode
02:     msg = "hello"
03:
04:     print("Sending msg, "to Broadcast")
05:     xnode.transmit(xnode.ADDR_BROADCAST, msg)
06:     print("complete")
```

데이터 송신, 수신

- Router 장치에서 코드 실행
  - 네트워크내의 모든 모듈에게 'hello' 메시지 전송
  - terminal에서 출력 확인

```
Sending msg :  hello to Broadcast
complete
```

- Coordinator 장치에서 수신된 메시지

```
{'profile': 49413, 'dest_ep': 232, 'broadcast': True, 'sender_nwk': 25611, 'source
_ep': 232, 'payload': b'hello', 'sender_eui64': b'\x00\x13\xa2\x00A\xae\\\x90', 'c
luster': 17}
```

데이터 송신, 수신

- 데이터 송신(unicast)
  - xnode.transmit()의 첫 번째 인자에 xnode.ADDR_COORDINATOR 전달
  - Coordinator에게만 데이터 송신 가능

```
01:     from pop import xnode
02:     msg = "hello"
03:
04:     print("Sending msg : ", msg, "to Coordinator")
05:     xnode.transmit(xnode.ADDR_COORDINATOR,msg)
06:     print("complete")
```

데이터 송신, 수신

- Router 장치에서 코드를 실행
  - 네트워크 내의 Coordinator에게 'hello'를 전송
  - terminal에서는 출력 확인

```
Sending msg :  hello to Coordinator
complete
```

- Coordinator 장치에서 수신된 메시지

```
{'profile': 49413, 'dest_ep': 232, 'broadcast': True, 'sender_nwk': 25611, 'source
_ep': 232, 'payload': b'hello', 'sender_eui64': b'\x00\x13\xa2\x00A\xae\\\x90', 'c
luster': 17}
```

데이터 송신, 수신

- 데이터 송신(unicast)
  - discover()로 검색된 장치에게 transmit()로 'hello'전송

```
01:     from pop import xnode
02:     msg = "hello"
03:
04:     print("Discovering network devices...\n")
05:
06:     for device in xnode.discover():
07:         addr = device['sender_eui64']
08:         node_id = device['node_id']
09:         print("Sending msg : ", msg, "to", node_id)
10:         xnode.transmit(addr, msg)
11:         print("complete")
```

데이터 송신, 수신

- Router 장치에서 코드를 실행
  - 네트워크 내의 검색된 모듈에'hello'를 전송
  - terminal에서는 출력 확인

```
Discovering network devices...

Sending msg :  hello to Coordinator
complete
```

- Coordinator 장치에서 수신된 메시지

```
{'profile': 49413, 'dest_ep': 232, 'broadcast': False, 'sender_nwk': 25611, 'sourc
e_ep': 232, 'payload': b'hello', 'sender_eui64': b'\x00\x13\xa2\x00A\xae\\\x90', '
cluster': 17}
```

데이터 송신, 수신

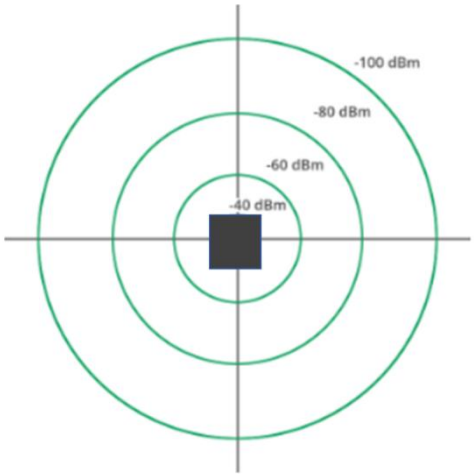
- 송신, 수신 power
  - 전력소모량
    - 센서 네트워크를 구현할 때 고려 해야 할 점
    - Battery로 운용이 가능해야 하므로 제한적인 Power를 효율적으로 사용 필요
  - XNode는 통신의 송신 세기를 설정('PL')하여 효율적인 Battery 운영 가능

데이터 송신, 수신

- 수신 신호 강도 표시기 (RSSI)
  - 무선 신호에 존재하는 전력량 측정
  - 안테나에서 수신된 신호 강도의 대략적인 값
  - 수신 안테나에서 신호 강도를 측정하는 것은 통신 링크의 품질을 결정하는 한 가지 방법
  - 송신기와 수신기 거리가 가까울수록 수신 안테나의 신호 강도 증가
  - 송신기가 멀어지면 수신 안테나의 신호 강도 감소
  - RSSI는 dBm 단위로 측정. 더 큰 음수 값 (dBm) 은 더 약한 신호 의미
    - -50dBm이 -60dBm보다 높은 신호 강도
  - RSSI는 안테나 포트에서 감지된 RF 에너지 표시

데이터 송신, 수신

- RSSI는 안테나 포트에서 감지된 RF 에너지 표시
  - 보고된 전력 수준은 배경 잡음 및 간섭의 에너지와 원하는 신호의 에너지를 포함 가능하므로 인위적으로 높일 수 있음
  - 이 상황은 지속해서 높은 RSSI 판독 값을 얻을 수 있으나 간섭 발생하기 쉬운 환경에서 더 나쁨



데이터 송신, 수신

- Tx Power변경에 따른 수신 강도 변화를 측정하는 예제
  - 예제를 위해서는 2개의 XNode가 필요하고 동일한 네트워크에 접속된 장치 필요
  - 2개의 XNode를 다음과 같이 설정

Command	Coordinator	Router
CE	0x01	0x00
ID	0x15	0x15
JV	-	0x01

데이터 송신, 수신

- 아래의 코드는 Tx Power를 지정된 시간마다 변경하는 코드
- 'PL' 값을 0x00~0x04로 변경. 변경할 때 NI 값을 변경하여 현재의 PL 값을 확인

```
01:     from pop import xnode
02:     import time
03:
04:     pl = 0x00
05:
06:     while True:
07:         if pl <= 0x04:
08:             msg = 'PL' + str(pl)
09:             xnode.atcmd('PL',pl)
10:             xnode.atcmd('NI',msg)
11:             print(msg)
12:             pl += 0x01
13:             time.sleep(15)
14:
15:         else:
16:             pl = 0x00
```

데이터 송신, 수신

- XNode(Router) 장치에서 코드를 실행
  - 현재 설정된 msg값에 의해 설정된 PL 값을 확인

```
PL0
PL1
PL2
PL3
PL4
PL0
```

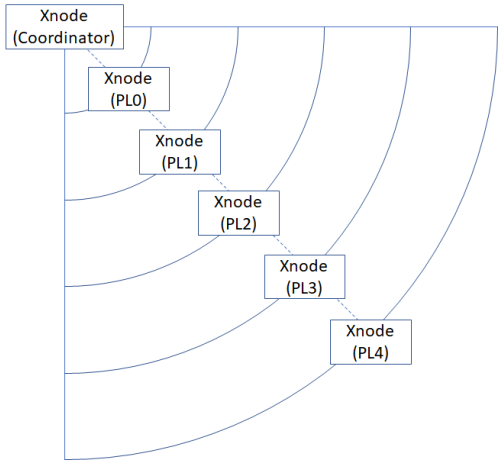
데이터 송신, 수신

- 지속해서 네트워크에 속한 장치 검색
- 검색된 장치의 NI와 rssi, 그리고 'DB' 확인

```
01:     from pop import xnode
02:
03:     while True:
04:         try:
05:             for device in xnode.discover():
06:                 print("NI : ", device['node_id'], ", rssi : ",device['rssi'],", DB : ", xnode.atcmd('DB'))
07:         except:
08:             pass
```

데이터 송신, 수신

- XNode(Coordinator) 장치에서 코드를 실행 (XNode(Router) 동작 중)
  - 결과를 확인
  - 두 장치 간에 거리가 멀어질수록 높은 'PL'이 설정된 경우에만 검색 가능




```
NI :  PL3 , rssi :  -90 , DB :  95
NI :  PL4 , rssi :  -91 , DB :  95
NI :  PL4 , rssi :  -90 , DB :  95
```

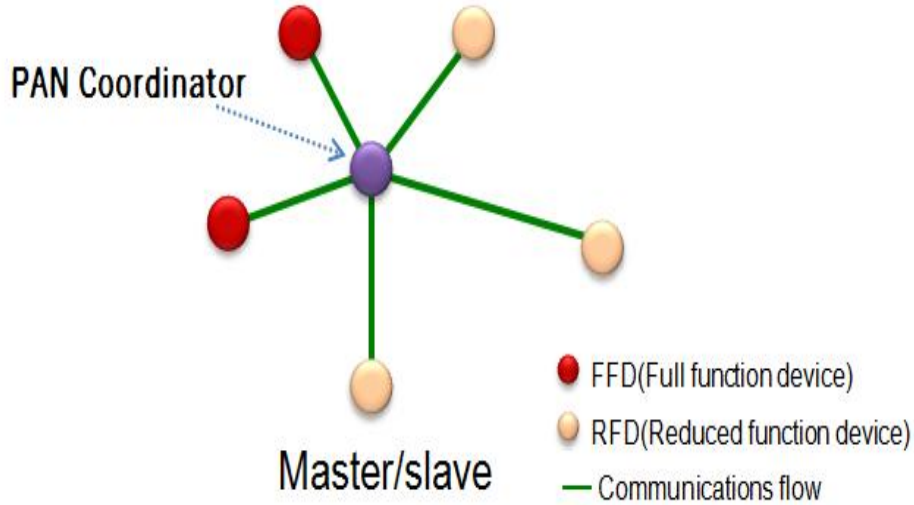
네트워크 토폴로지

- 네트워크 토폴로지
  - 네트워크에 참여한 노드들의 기하학적인 연결 형태
  - IEEE 802.15.4 응용 요구사항에 따라 스타(Star)와 피어투피어(Peer to peer) 토폴로지에서 작동 가능

네트워크 토폴로지

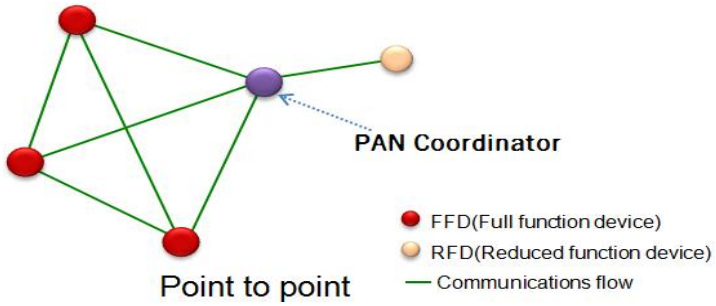
- 스타 토폴로지 통신
  - PAN 코디네이터와 디바이스 간의 연결
  - 일부 디바이스에 설치된 응용프로그램은 네트워크 통신의 시작이나 종료에 관여
  - PAN 코디네이터는 네트워크의 초기화나 종료 및 네트워크 간의 경로 지정을 위해 특별한 응용프로그램을 사용
  - PAN 코디네이터는 PAN의 메인 컨트롤러이며 특정 토폴로지의 네트워크에서 운영되는 모든 디바이스는 64비트의 고유 주소를 갖습니다.
  - 하나의 PAN 코디네이터에 다수의 네트워크 디바이스가 참여하는 가장 일반적인 형태의 로컬 네트워크 구현
  - PAN 코디네이터를 중심으로 통신이 수행되므로 네트워크 구현 역시 가장 간단

네트워크 토폴로지


PAN Coordinator
Master/slave
FFD(Full function device)
RFD(Reduced function device)
Communications flow

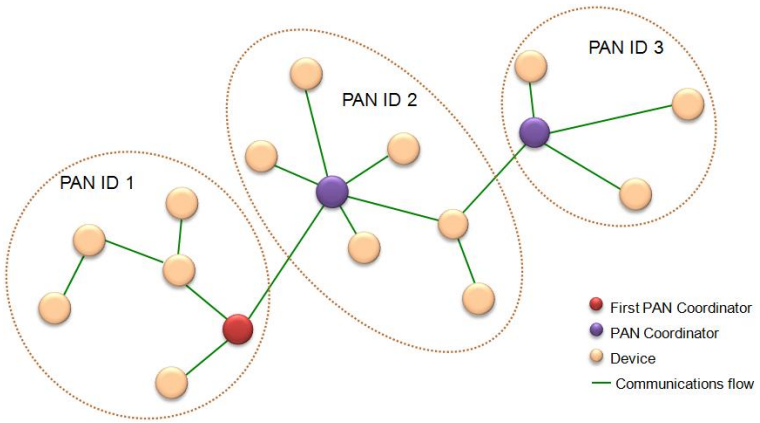
네트워크 토폴로지

- 피어투피어(Peer to Peer) 토폴로지
  - FFD로 구성되며, RFD가 포함될 때는 PAN 코디네이터를 추가
  - 로컬 네트워크에 주로 사용
  - 대부분의 네트워크 디바이스가 FFD로 구성되므로 전력 소모가 많다는 단점이 있음
  - 모든 네트워크 디바이스간 통신 경로를 설정하기 위해 라우팅 알고리즘의 적용 필요
  - 피어투피어는 메쉬(Mesh) 토폴로지로도 불림



네트워크 토폴로지

- 클러스터 트리 토폴로지
  - 스타 토폴로지로 각각의 로컬 네트워크를 구성한 후 로컬 네트워크와 로컬 네트워크를 피어투피어 형태로 연결해 대규모 네트워크를 구성하는 토폴로지
  - 각 로컬 네트워크마다 PAN 코디네이터가 존재
  - 각 로컬 네트워크를 피어투피어로 연결하기 위해 복잡한 라우팅 알고리즘 적용

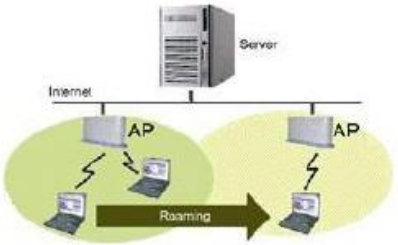


무선 통신 모드

- 무선 LAN이나 LR-WPAN과 같은 무선 기기 간의 두 가지 통신 모드
  - 중계기나 기지국과 같은 별도의 기반 시설을 이용하는 인프라스트럭처 (Infrastructure)
    - 무선 공유기나 액세스 포인터(Access Pointer)를 휴대폰은 기지국(Base station)을 기반 시설로 이용해 기기 간에 통신을 수행
    - 기반 시설은 일정 거리 이상에서도 통신이 수행되도록 높은 안테나 출력을 유지
    - 대규모 무선 기기 간의 통신에 필요한 관리 기능을 제공
    - 주로 대량의 고속 통신 운영에 활용

무선 통신 모드

- 기반 시설 없이 직접 통신이 가능한 애드혹(Ad-Hoc)
  - 애드혹 모드는 기반 시설 없이 무선 기기 간에 직접 통신하는 방법
  - 기반 시설을 이용한 중앙 집중화된 관리가 필요 없음
    - Xnode와 같이 네트워크 구성이 유동적이며 안테나 출력이 제한된 환경에서 주로 사용
  - 노트북에 내장된 무선 LAN을 애드혹 모드로 운영하면 액세스 포인터나 무선 공유기 없이 노트북 간에 직접 통신을 수행하는 것이 가능
    - 전송 속도나 통신에 참여 가능한 노트북 개수는 제한



애드혹 네트워크

- 애드혹 기반의 네트워크 특징
  - 보안
    - 통신 기기 간에 주고받는 정보에 대한 무결성 및 도청방지를 제공하는 기능
    - 정보에 대한 암호화 구현
  - 라우팅(Routing)
    - 라우팅은 통신 거리 확장 및 장애 회피를 위해 정보의 이동을 관리하는 기능으로 다양한 라우팅 알고리즘을 이용해 구현
  - 이동성

애드혹 네트워크

- 애드혹 네트워크의 보안 요구 조건은 다른 통신 네트워크와 동일
  - 무선 통신은 매체를 신뢰할 수 없는 상황에서 보안에 필요한 암호를 사용
  - 암호 키에 크게 의존
  - 키 사이에 신뢰할 수 있는 관계를 형성하고 이를 애드혹 네트워크 전반에 분배하는 것이 중요

애드혹 네트워크

- 애드혹 통신에서 사용하는 암호는 공개키 방식을 주로 사용
  - A, B, C 세 개의 그룹이 존재한다고 할 때 그룹 A의 대표 노드가 서버 노드 역할을 수행하여 신뢰 위임(Trust delegation) 절차를 주도
  - B, C 그룹의 대표 노드들은 자기 그룹이 정한 공개키를 다른 그룹과 교환해 그룹 간 신뢰 관계를 형성

애드훅 네트워크

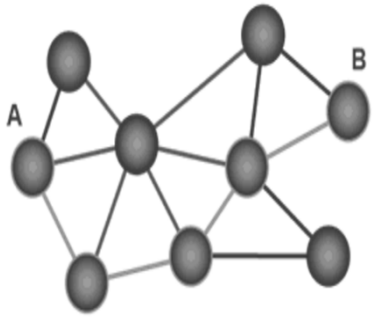
- 노드가 이동할 수 있는 애드훅 네트워크의 경우
  - 노드 간에 주고받는 정보가 네트워크 내에서 원활하게 교환하도록 관리하는 라우팅 기능 필요
    - 라우팅 : 네트워크에 참여한 노드들이 출발지에서 보낸 정보의 목적지까지 경로 배정 기능
  - 노드의 이동성이 심한 경우
    - 네트워크에 일반적인 정보 대신 라우팅 정보만 가득 찰 수 있음
    - 노드의 배터리 소모가 급속도로 증가하기도 함
    - 해결책 : 라우팅이 필요한 경우에만 경로 배정을 하는 반응에 따른 경로 배정 기법 필요

Mesh 네트워크

- 메시 네트워크
  - 네트워크의 각 노드가 주변의 다른 노드에 연결되는 토폴로지
- 메시 네트워크의 세 가지 기능
  - Routing : 메시지가 최종 목적지에 도달할 때까지 노드에서 호핑하여 경로를 따라 메시지가 전파됨
  - Ad-hoc network creation : 사람의 개입 없이 전체 노드 네트워크를 자동 생산하는 프로세스
  - Self-healing : 네트워크에서 누락된 노드를 자동 파악하고 복구하도록 네트워크 재구성

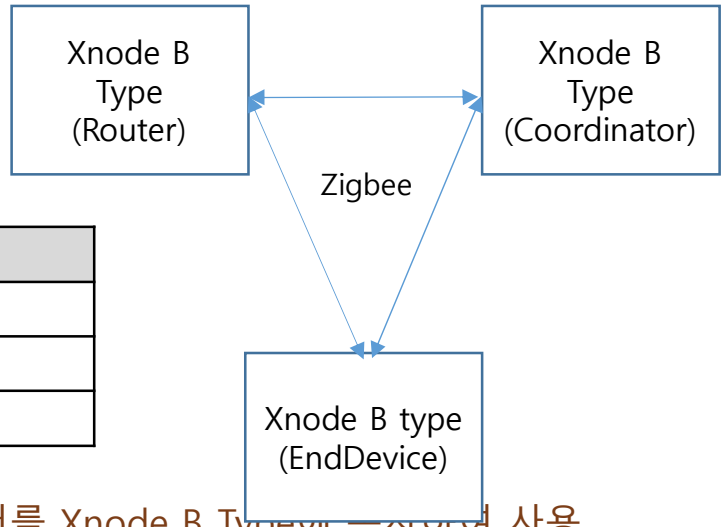
Mesh 네트워크

- 메시지를 전달하기에 충분한 노드가 있다면 두 노드 사이의 거리는 중요하지 않음
  - 노드 간 통신 시 네트워크는 자동으로 최상의 경로 계산
- 메시 네트워크는 안정적이며 중복성 제공
  - 노드가 네트워크에서 더 이상 작동할 수 없는 경우
    - 나머지 노드는 여전히 직접 또는 중간 노드를 통해 서로 통신 가능



Mesh 네트워크 통신

- Mesh 네트워크 통신 예제
  - 준비물



준비물	
1	PC 1ea
2	XNode B Type 3ea
3	Micro type USB cable 2ea

- 예제 진행을 위해 USB>Library>CORE>lib 폴더를 Xnode B Type에 복사하여 사용

NI	추가 라이브러리
Coordinator	USB > Library > CORE > lib > pop.py, core_a.py, core_b.py
Router	USB > Library > CORE > lib > pop.py, core_a.py, core_b.py
EndDevice	USB > Library > CORE > lib > pop.py, core_a.py, core_b.py

Mesh 네트워크 통신

- Mesh 네트워크 통신 예제 순서

| 1 | Xnode 3대 준비

| 2 | 각각 Coordinator, Router, End Device로 설정.

각각의 NI를 Coordinator, Router, End Device로 설정

| 3 | Coordinator는 End Device로 지속해서 데이터를 전송하는 코드와 수신된 데이터를 출력하는 코드 실행

| 4 | End Device는 수신된 데이터를 반송하는 코드를 실행

Mesh 네트워크 통신

| 5 | Router는 코드 실행 없음

| 6 | Coordinator와 End Device가 서로 데이터를 주고받을 수 없을 정도의 거리에 위치시킴

| 7 | Coordinator와 End Device 사이에 Router를 위치 시켜 전원을 켜고, 껐을 때 Coordinator에서 출력되는 메시지 확인하여 Mesh 네트워크 동작 여부 확인

Mesh 네트워크 통신

- Xnode 설정
  - 1개의 Coordinator, 1개의 Router, 1개의 End Device 필요
  - Xnode 3개를 다음과 같이 설정한 후 'WR' command 실행
  - 모든 설정이 완료된 후 reset 버튼을 눌러 네트워크 검색

Command	Coordinator	Router	End Device
NI	'Coordinator'	'Router'	'EndDevice'
CE	0x01	0x00	0x00
ID	0x15	0x15	0x15
JV	-	0x01	0x01
SM	-	-	0x04

Mesh 네트워크 통신

Coordinator

```
01:  from pop import xnode
02:
03:  xnode.atcmd('NI', 'Coordinator')
04:  xnode.atcmd('CE', 0x01)
05:  xnode.atcmd('ID', 0x15)
06:  xnode.atcmd('WR')
```

Router

```
01:  from pop import xnode
02:
03:  xnode.atcmd('NI', 'Router')
04:  xnode.atcmd('CE', 0x00)
05:  xnode.atcmd('ID', 0x15)
06:  xnode.atcmd('JV', 0x01)
07:  xnode.atcmd('WR')
```

End Device

```
01:  from pop import xnode
02:
03:  xnode.atcmd('NI', 'EndDevice')
04:  xnode.atcmd('CE', 0x00)
05:  xnode.atcmd('ID', 0x15)
06:  xnode.atcmd('JV', 0x01)
07:  xnode.atcmd('SM', 0x04)
08:  xnode.atcmd('WR')
```

Mesh 네트워크 통신

- 설정이 완료된 후 Coordinator에서 네트워크를 검색
- 2개의 장치가 검색되는 것 확인

Coordinator

```
01:     from pop import xnode
02:
03:     for i in xnode.discover():
04:         print(i)
```

```
{'rssi': -20, 'node_id': 'EndDevice', 'device_type': 1179648, 'parent_nwk': 0, 'sender_nwk': 8607, 'sender_eui64': b'\x00\x13\xa2\x00A\xae\\\x07', 'node_type': 2}
{'rssi': -26, 'node_id': 'Router', 'device_type': 1179648, 'parent_nwk': 65534, 'sender_nwk': 42278, 'sender_eui64': b'\x00\x13\xa2\x00A\xae\\\x90', 'node_type': 1}
```

Mesh 네트워크 통신

- Coordinator 코드
  - Coordinator에서 End Device로 1초 간격으로 데이터를 전송
  - 수신된 메시지가 있다면 출력

Mesh 네트워크 통신

```
01:  from pop import xnode
02:  import time
03:
04:  for i in xnode.discover():
05:      if i['node_id'] == EndDevice':
06:          addr = i['sender_eui64']
07:          print("End Device addr : ", addr)
08:
09:  pre_stime = time.ticks_ms()
10:  Count = 0
11:
12:  while True:
13:      try:
14:          if time.ticks_ms()-pre_stime>1000:
15:              payload = 'Count' + str(Count)
16:              xnode.transmit(addr, payload)
17:              print('send msg : ', payload)
18:              Count += 1
19:              pre_stime = time.ticks_ms()
20:
21:          msg = xnode.receive()
22:          if msg['sender_eui64']==addr:
23:              print('received msg : ', msg['payload'])
24:
25:      except:
26:          pass
27:
28:      time.sleep(0.1)
```

Mesh 네트워크 통신

- End Device 코드
  - End Device에 수신된 payload를 송신 주소에 다시 전송

```
01:  from pop import xnode
02:  import time
03:
04:  while True:
05:      try:
06:          msg = xnode.receive()
07:          if msg:
08:              addr = msg['sender_eui64']
09:              payload = msg['payload']
10:              xnode.transmit(addr, payload)
11:              print("send addr / msg : ", addr, "/", payload)
12:
13:      except:
14:          pass
15:
16:      time.sleep(0.1)
```

Mesh 네트워크 통신

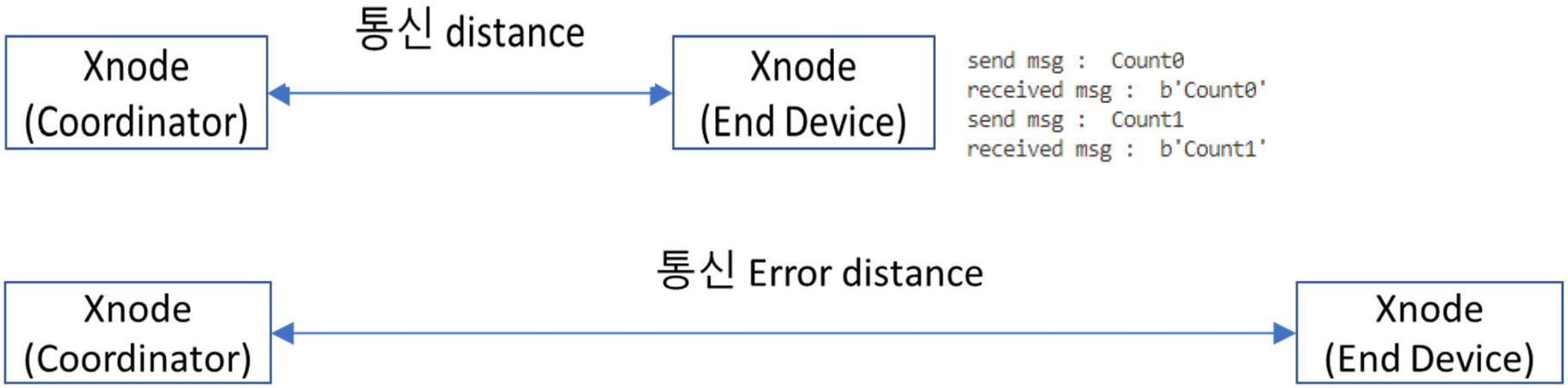
- Coordinator와 End Device가 서로 통신 가능한 거리 내에 있는 경우

  Coordinator 출력

```
End Device addr :  b'\x00\x13\xa2\x00A\xae\\\x07'
send msg :  Count0
received msg :  b'Count0'
send msg :  Count1
received msg :  b'Count1'
```

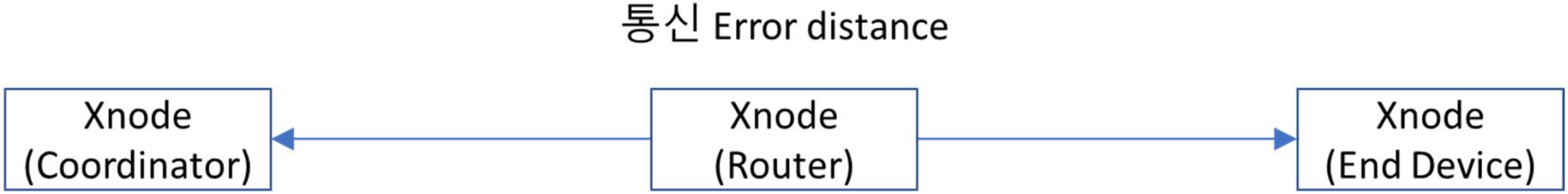
Mesh 네트워크 통신

- Mesh 네트워크
  - Coordinator와 End Device가 실행 중인 상태에서 서로 간의 거리를 점점 멀리 떨어뜨려 더 이상 데이터통신이 되지 않도록 두 장치를 위치시킴



Mesh 네트워크 통신

- Router 장치를 Coordinator와 End Device 장치 사이에 놓고 전원을 켰을 때 Coordinator와 End Device 장치 간에 통신이 이루어지는지 확인




```
send msg :  Count0
received msg :  b'Count0'
send msg :  Count1
received msg :  b'Count1'
```

Mesh 네트워크 통신

- End Device 복구
  - End Device 장치는 sleep 모드로 동작하기 때문에 이후 예제 진행 시 불편함
  - Mesh 네트워크 통신 예제를 진행한 후에는 반드시 Router로 변경해야 함

```
01: from pop import xnode
02:
03: print("Before SM : ", xnode.atcmd('SM'))
04:
05: xnode.atcmd('SM', 0x00)
06: xnode.atcmd('WR')
07:
08: print("After SM : ", xnode.atcmd('SM'))
```

```
Before SM :  4
After SM :  0
```