KARADENİZ TEKNİK ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ BİLGİSAYAR AĞLARI LABORATUARI

KABLOSUZ SENSÖR AĞLAR DENEYİ

1. GİRİŞ

Son zamanlarda mikro elektro-mekanik sistemlerin gelişmesiyle birlikte, ebatları küçük, düşük maliyetli, az güç gerektiren ve kısa mesafelerde kablosuz haberleşebilen çok fonksiyonlu sensör düğümlerinin tasarlanması ve geliştirilmesi mümkün hale gelmiştir. Yetenekleri sürekli artan bu küçük sensör düğümleri çevrelerinde meydana gelen değişiklikleri **algılayabilir (sensing)**, elde ettikleri veriyi **işleyebilir (processing)** ve **haberleşebilirler (communicating)**. Birbirleriyle işbirliği içerisinde çalışan sensör düğümleri kablosuz sensör ağlarını meydana getirirler.

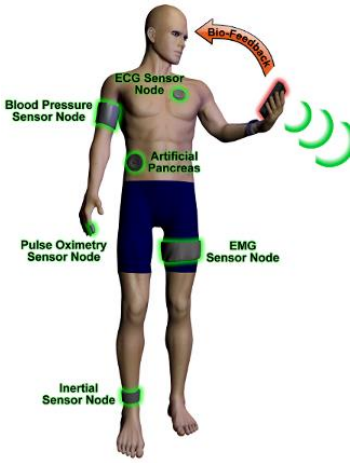
2. KABLOSUZ SENSÖR AĞLAR (WIRELESS SENSOR NETWORKS)

WSNs'in amacı sıcaklık, ses, basınç, nem gibi fiziksel ve çevresel değişiklikleri algılamak ve algılanan veriyi kurulan ağ üzerinden işbirliği içerisinde ana bir merkeze iletmektir. WSNs giderek hayatımızın önemli bir parçası haline gelmektedir. Bu ağlar geniş uygulama alanlarına sahiptir. Askeri, çevresel, sağlık, endüstriyel uygulamaların yanı sıra akıllı evler içinde uygulama alanları vardır. Bu ağları daha iyi anlayabilmek için en çok bilinen iki örnek aşağıda verilmiştir.

- **Sniper Algılama Sistemi:** Askeri alanda kullanılmak üzere geliştirilmiş bu sistemin amacı savaş alanında keskin nişancının yerini tam olarak tespit etmektir. Figür 1'de görüldüğü gibi bu sistem birçok sensör düğümü içeren bir kablosuz sensör ağdır. Bir araca bağlanan yada bir asker tarafından giyilebilen bu sistemde ateş edildiğinde sesi algılayabilmek için pasif akustik sensörler kullanılmıştır. Sensörler aracılığıyla algılanan ses verileri işlenerek ateş edilen yön tahmin edilmeye çalışılmaktadır. İnsan kulağının sesin nereden geldiğini algılayabildiği gibi, ortamda meydana gelen ses dalgaları sensörlere farklı zamanlarda ulaştığı için sesin geldiği yeri algılamak mümkündür. Böylece bu sistem ile keskin nişancı hareket halinde olsa dahi konumu algılanabilmektedir.

Figür-1: Sniper Algılama Sistemi

- **Hasta İzleme Sistemi:** Sağlık alanında kullanılan bu sistem, giyilebilir sensörler aracılığıyla hastanın EKG ve EMG gibi değerlerini ölçmeyi amaçlamaktadır. Bu sistem ile hastanın nabız atış oranı, kandaki oksijen miktarı, kalbin elektriksel aktiviteleri, kas aktiviteleri ve hastanın hareketi algılanabilmektedir. Bu sistem ile hastanın yukarıda sıralanan değerleri periyodik olarak izlenebilir ve acil durumlarda müdahale edilebilir.



Figür-2: Hasta İzleme Sistemi

3. KABLOSUZ SENSÖR AĞLAR VE ZIGBEE PROTOKOL YIĞINI

Bu bölümde kablosuz sensör düğümlerinin katmanları ve protokol yığınları kısaca incelenecek ve haberleşme standartları anlatılacaktır. Figür-3.a'da görüldüğü gibi bir WSN düğümü **fiziksel**, **veri iletim**, **ağ**, **taşıma** ve **uygulama** katmanlarından oluşur.

ZigBee, IEEE 802.15.4 standardı ile tanımlanan ve kişisel alan ağında haberleşmeyi sağlayan bir standarttır. Bundan dolayı ZigBee ve 802.15.4 kavramları karıştırılabilir. Figür-3.b'de görüldüğü gibi herbiri farklı katmanları tanımlayan farklı standartlardır. Fiziksel katman ve

MAC katmanı 802.15.4 tarafından tanımlanırken ağ ve uygulama katmanı ZigBee tarafından tanımlanmaktadır.

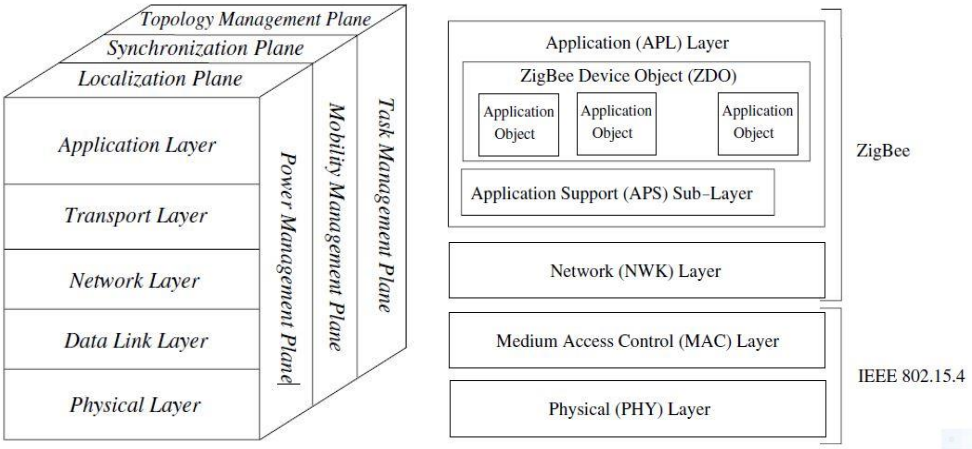
Fiziksel Katman ; frekans seçimi, taşıyıcı frekans üretimi, gelen sinyali algılama, modülasyon ve bit düzeyinde veri şifrelemekle görevlidir. **2.4 GHz** frekans bandında haberleşen ZigBee **250 Kb/s** gönderim hızına sahiptir. **10-100 m** arasında kapsama alanı vardır.

Veri İletim Katmanı ; veri akışını çoğullama (multiplexing), ortam erişim kontrolü (MAC) ve hata kontrolüyle görevlidir.

Ağ Katmanı ; WSN düğümlerini adresleme, yönlendirme (routing) ve tıkanıklık kontrolüyle görevlidir. **Normalde ZigBee IP adresiyle uyumlu değildir. Çünkü IPv6 başlık ve adres alanları 40 byte iken ZigBee toplam paketin 127 byte olmasına izin verir.** Bundan dolayı ZigBee cihazları internete açılamaz ve sunucu veya web tabanlı cihazlarla haberleşemezler. Bu problemi ortadan kaldırmak ve ZigBee cihazlarını IP adresiyle uyumlu hale getirmek için IETF (Internet Engineering Task Force) **6LowPAN** standardını geliştirmiştir. Bu çözüm ile paket başlıkları sıkıştırılmış ve boyutları küçültülmüştür. Ayrıca **6LowPAN** ile gerektiğinde paketler parçalanabilir ve hedefte birleştirilebilir. **RPL** ise ağ katmanında çalışan ve WSN düğümleri için geliştirilmiş bir yönlendirme protokolüdür.

Taşıma Katmanı ; datagramların güvenli olarak iletimiyle görevlidir. **TCP protokolü güvenli bir taşıma katmanı protokolü olmasına rağmen WSN için uygun bir protokol değildir.** Çünkü WSN düğümleri için harcanan güç çok önemlidir ve kısıtlı güce ve belleğe sahip olan WSN düğümleri ACK mesajlarıyla fazla enerji harcayacaktır. Bundan dolayı güvenli bir UDP protokolü WSN için daha uygundur.

Uygulama Katmanı ; kullanıcı ihtiyaçlarına göre tasarlanmış protokolleri içeren katmandır. Örneğin, **CoAP**, WSN düğümleri için geliştirilmiş bir web transfer protokolüdür.



a) b)

Figür-3: a) Sensör ağlar protokol yığını b) IEEE 802.15.4 ve Zigbee protokol yığını.

4. CONTİKİ İŞLETİM SİSTEMİ

4.1. Contiki İşletim Sistemi Nedir?

2002 yılında Adam Dunkels yönetiminde C programlama diliyle geliştirilmiş açık kaynaklı bir işletim sistemidir. Contiki işletim sistemi, **kısıtlı hafızaya** ve **düşük güce** sahip olan **IoT**(internet of things) cihazları ve **WSN** düğümleri için geliştirilmiştir. Normal bir Contiki konfigürasyonu 2 kilobyte RAM ve 40 kilobyte ROM belleğe sahip bir mikrokontrolcü içindir. Contiki IPv4 ve IPv6 üzerinden haberleşmeyi sağlayabilmektedir. Bu işletim sisteminin WSN düğümleri üzerinde koşulmasıyla birlikte birçok farklı uygulama geliştirmek mümkündür.

4.2.Contiki Dizin Yapısı

Contiki dizinleri aşağıda kısaca açıklanmıştır.

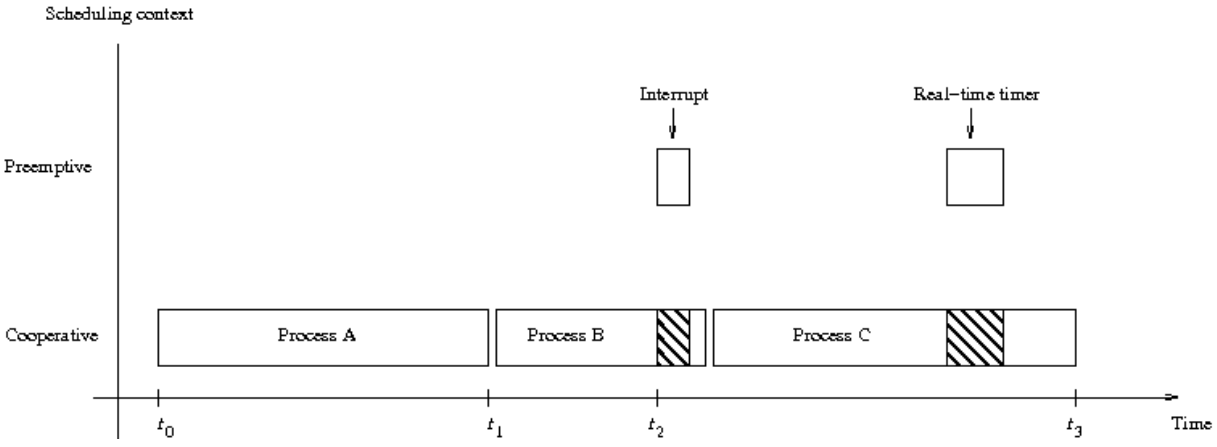
- **contiki/core:** Sistemin kaynak kodlarını içerir. Sistemin farklı kısımları için alt dizinler bulunur.
- **contiki/examples:** Örnek uygulamaların kaynak dosyalarını içerir. Her bir örneğe ait dosyalar farklı dizinlerde bulunmaktadır.
- **contiki/apps:** Kullanmaya hazır Contiki uygulamalarını içerir.

- **contiki/platform:** Her bir mote (sensor düğümü) için spesifik sürücüleri içerir.
- **contiki/cpu:** Spesifik MCU (microcontroller unit) dosyalarını içerir. Örneğin cpu/msp430/ dizini Z1 mote için gereken MCU sürücülerini içerir.
- **contiki/tools:** Contiki için yazdığınız bir uygulamanın simülasyonunu yapmak yada debug etmek için gereken araçları barındırır. Örneğin COOJA simülatörü bu dizin içerisindedir.

4.3.Contiki İşletim Sisteminde Prosesler

Bütün contiki programları proseslerden oluşmaktadır. Bir proses, contiki işletim sistemi tarafından çalıştırılan kod bloklarından ibarettir. Contiki prosesleri, sistem yüklenince yada proses içeren bir modül sisteme yüklendiğinde başlatılır. Contiki işletim sistemi için yazılan kodlar iki şekilde çalışır. Bunlar;

- Cooperative Kod: Şekilden de görüldüğü gibi bu kodlar sırayla birbiri ardınca çalışan kodlardır.
- Preemptive Kod: Bu kodlar önceliğe sahip kodlardır. Herhangi bir anda cooperative kodun çalışmasını durdurabilir. Preemptive kodların koşulması tamamlandıktan sonra cooperative kod kaldığı yerden çalışmasına devam eder.



Figür-4: Cooperative ve Preemptive Kodların Çalışma Mantığı

Bir contiki prosesi iki kısımdan oluşur. Bunlar;

- **Proses Kontrol Bloğu:** Ram de tutulur. Proses ile ilgili genel bilgileri içerir. Bunlar; prosesin ismi, prosesin durumu ve proses thread'ini gösteren işaretçi. Proses kontrol bloğu aşağıda gösterildiği gibidir.

```
struct process {
  struct process *next;
  const char *name;
  int (* thread)(struct pt *,
  process_event_t,
  process_data_t);
  struct pt pt;
  unsigned char state, needspoll;
};
```

- **Proses Thread:** Prosesin asıl kodlarını içeren kısımdır. Bir proses thread aslında hafızanın boşa harcanmadığı bir protothread'dir.

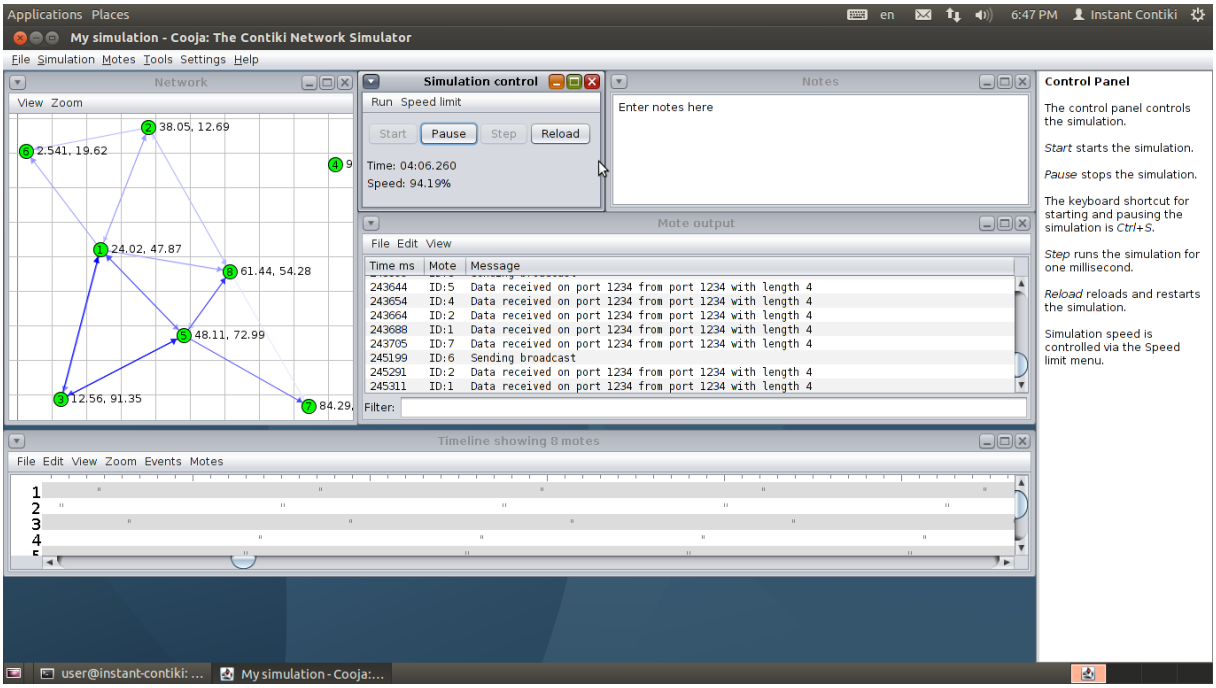
```
PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();
  printf("Hello, world\n");
  PROCESS_END();
}
```

4.4. COOJA SİMÜLATÖRÜ

Cooja simülatörü sayesinde contiki işletim sisteminde geliştirilen bir uygulamanın simülasyonu yapılabilir, test edilebilir. Cooja simülatörünün detaylarını 7.1 bölümünde verilen link üzerinden inceleyebilirsiniz. Figür-5'de Cooja ile oluşturulan bir kablosuz sensör ağ ortamı gösterilmiştir. Cooja simülatörü contiki/tools/cooja dizininde bulunmaktadır. Simülatörü aşağıdaki konsol komutlarıyla çalıştırabilirsiniz;

cd contiki/tools/cooja/

sudo ant run

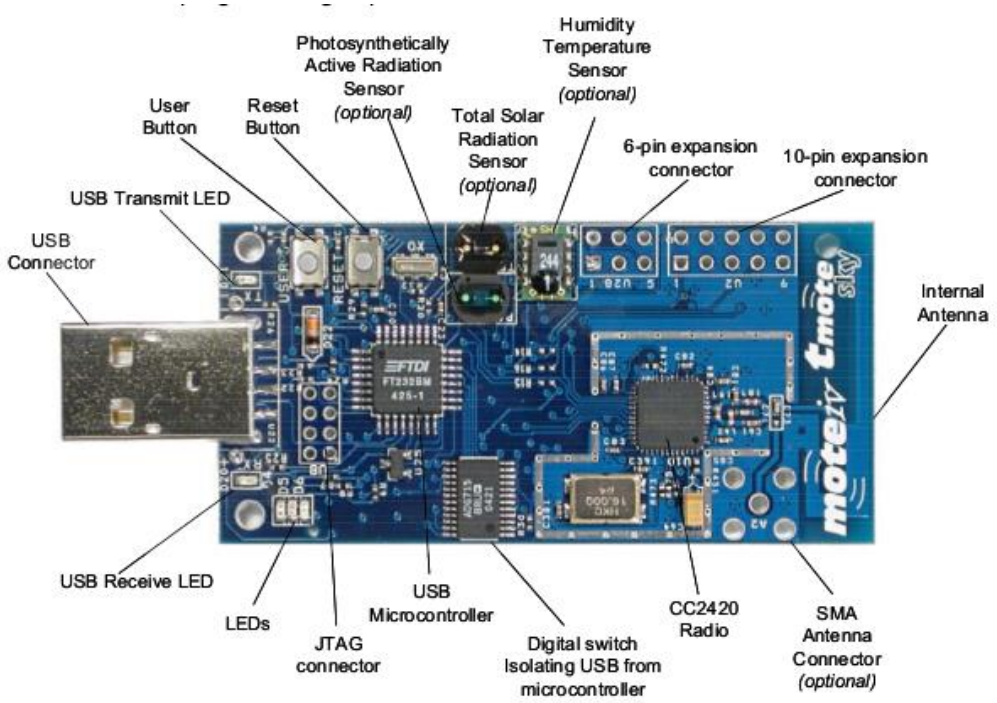


Figür-5: Cooja simülatörüyle oluşturulan bir kablosuz sensör ağı

5. CONTIKI OS İLE TMOTE SKY SENSÖR MODÜLÜ PROGRAMLAMA

5.1.Tmote Sky

Tmote Sky, sensör ağlarda kullanılmak için geliştirilmiş çok düşük güç ile çalışan kablosuz bir modüldür. Bu modül Contiki OS ile hızlı bir şekilde programlanabilir ve yazılan uygulamaların çıktıları izlenebilir. Bu modül IEEE 802.15.4 standardını kullanarak diğer modüllerle işbirliği içinde çalışabilir. Üzerinde nem, sıcaklık ve ışık sensörleri bulunur. Bu modül ile ilgili detaylı bilgiye http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf adresinden ulaşılabilir.



Figür-6: Tmote Sky Sensör Düğümü

5.2.Tmote Sky Sensör Fonksiyonları

Öncelikli olarak yukarıda bahsedilen sensörleri kullanabilmek ve ölçümler yapabilmek için aşağıdaki kod bloğunun programa eklenmesi gerekir.

```
//Button, humidity and light sensors for sky mote
#if CONTIKI_TARGET_SKY
#include "dev/button-sensor.h"
#include "dev/sht11/sht11-sensor.h"
#include "dev/light-sensor.h"
#endif /* CONTIKI_TARGET_SKY */
```

Daha sonra kod içerisinde sensörlerin aktif edilmesi gerekir.

```
#if CONTIKI_TARGET_SKY
SENSORS_ACTIVATE(button_sensor ) ; // Buton sensörünü aktif yapar.

SENSORS_ACTIVATE(sht11_sensor) ; // Nem ve sıcaklık sensörünü aktif yapar.

SENSORS_ACTIVATE(light_sensor) ; // Işık sensörünü aktif yapar.

#endif /* CONTIKI_TARGET_SKY */
```

Işık, sıcaklık ve nem sensörlerinin değerlerini okuyan ve döndüren kodlar aşağıdadır.

```
#if CONTIKI_TARGET_SKY
uint16_t light_p = light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC);
```

```
uint16_t light_s = light_sensor.value(LIGHT_SENSOR_TOTAL_SOLAR);
uint16_t temperature = sht11_sensor.value(SHT11_SENSOR_TEMP);
uint16_t humudity = sht11_sensor.value(SHT11_SENSOR_HUMIDITY);
#endif /* CONTIKI_TARGET_SKY */
```

Sensor.value(...) ile elde edilen değerler ham değerlerdir. Gerçek değerleri elde etmek için bazı dönüşümlerin yapılması gerekmektedir. Bu dönüşümler aşağıda gösterilmiştir.

Işık Sensörü Dönüşümü: 10 * light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC) / 7

Sıcaklık Sensörü Dönüşümü: ((sht11_sensor.value(SHT11_SENSOR_TEMP) / 10) - 396) / 10

6. DENEYDE GERÇEKLEŞTİRİLECEK UYGULAMALAR

6.1. COOJA simülatörü kullanarak contiki/examples/ipv6/simple-upd-rpl/broadcast-example.c örneği gerçekleştirilecektir.

6.2. COOJA simülatörü kullanılarak bir server/client uygulaması yapılacaktır. Bu uyguama için examples/ipv6/rpl-udp/udp-server.c ve examples/ipv6/rpl-udp/udp-client.c kaynak dosyaları kullanılacaktır.

6.3. COOJA simülatörü kullanarak Sky Mote sensör düğümlerinde bulunan sensörler ile sıcaklık, nem ve ışık değerleri okunacak ve server düğüme gönderilecektir. Bu uygulamada 6.2 de yapılan uygulamadan farklı olarak server cihazına "hello" mesajı yerine sernsörlerden okunan değerler gönderilecektir.

6.4. İkinci adımda gerçekleştirilen simülatör uygulaması Sky Mote donanımlarıyla gerçeklenecektir. Sensörlerden okunan değerler "minicom" isimli metin tabanlı modem kontrol programıyla görüntülenecektir.

7. DENEY HAZIRLIĞI

7.1. **http://www.contiki-os.org/start.html** sayfasında anlatılan adımları sırasıyla takip ederek gerekli ortamı sağlayınız ve 3. adımda anlatıdığı gibi COOJA simülatörünü kullanarak broadcast-example.c örneğini gerçekleştiriniz.

7.2. Minicom programını sanal makina ortamına kurarak çalışma mantığını araştırınız.