

CHAPTER 6

PROGRAMMING WITH ARDUINO

- 6.1 Features of Arduino
- 6.2 Components of Arduino Board
- 6.3 Arduino IDE
- 6.4 Case Studies

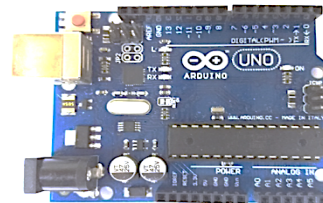
6.1 Features of Arduino

Types of Arduino Board

- Arduino boards based on ATMEGA328 microcontroller
- Arduino boards based on ATMEGA32u4 microcontroller
- Arduino boards based on ATMEGA2560 microcontroller
- Arduino boards based on AT91SAM3X8E microcontroller

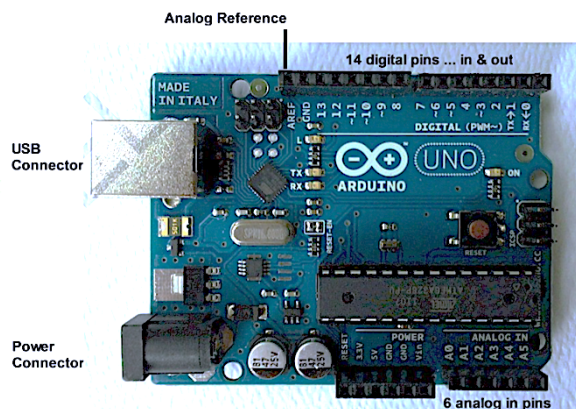
Arduino UNO

Feature	Value
Operating Voltage	5V
Clock Speed	16MHz
Digital I/O	14
Analog Input	6
PWM	6
UART	1
Interface	USB via ATmega16U2



Board Details

- Power Supply: USB or power barrel jack
- Voltage Regulator
- LED Power Indicator
- Tx-Rx LED Indicator
- Output power, Ground
- Analog Input Pins
- Digital I/O Pins

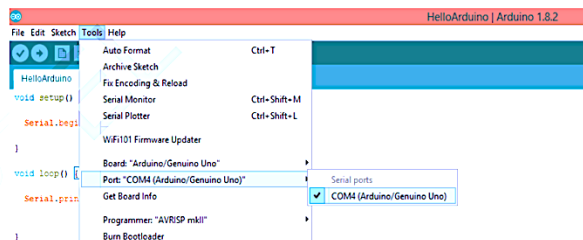
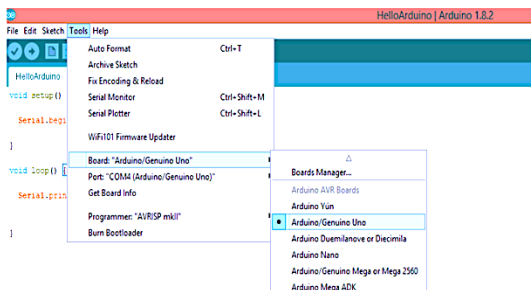


Arduino IDE

- Arduino IDE is an open source software that is used to program the Arduino controller board
- Based on variations of the C and C++ programming language
- It can be downloaded from Arduino's official [website](https://www.arduino.cc/) and installed into PC

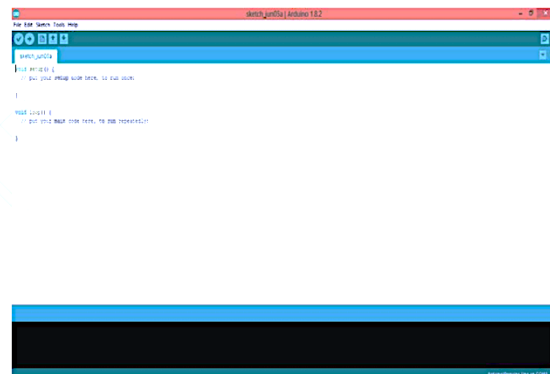
Set Up

- Power the board by connecting it to a PC via USB cable
- Launch the Arduino IDE
- Set the board type and the port for the board
- TOOLS -> BOARD -> select your board
- TOOLS -> PORT -> select your port

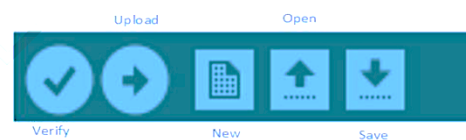
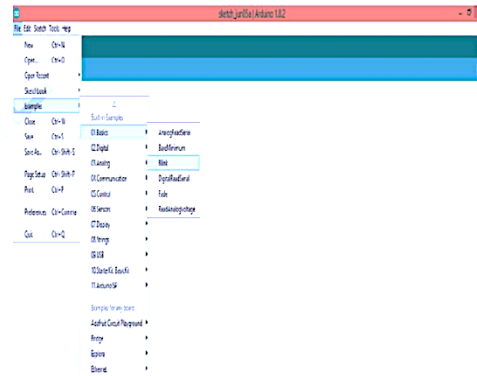


Arduino IDE Overview

Program coded in Arduino IDE is called a SKETCH



- To create a new sketch
 - File -> New
- To open an existing sketch
 - File -> open ->
- There are some basic ready-to-use sketches available in the EXAMPLES section
- File -> Examples -> select any program
- Verify: Checks the code for compilation errors
- Upload: Uploads the final code to the controller board
- New: Creates a new blank sketch with basic structure
- Open: Opens an existing sketch
- Save: Saves the current sketch
- Serial Monitor: Opens the serial console
- All the data printed to the console are displayed here



Sketch Structure

- A sketch can be divided into two parts:
 - Setup ()
 - Loop()
- The function setup() is the point where the code starts, just like the main() function in C and C++
- I/O Variables, pin modes are initialized in the Setup() function
- Loop() function, as the name suggests, iterates the specified task in the program



Supported Datatype

- Arduino supports the following data types-

Void	Long
Int	Char
Boolean	Unsigned char
Byte	Unsigned int
Word	Unsigned long
Float	Double
Array	String-char array
String-object	Short

Arduino Function Libraries

- Input/Output Functions:
 - The arduino pins can be configured to act as input or output pins using the pinMode() function

```
Void setup ()  
{  
    pinMode (pin , mode);  
}
```

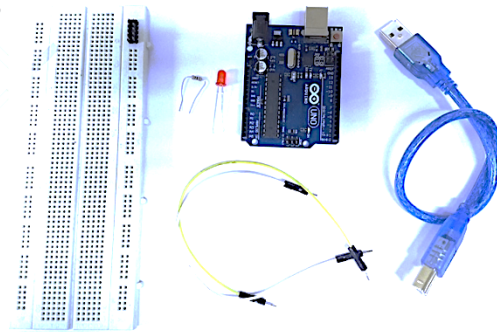
Pin- pin number on the Arduino board
Mode- INPUT/OUTPUT

Arduino Function Libraries (contd..)

- digitalWrite() : Writes a HIGH or LOW value to a digital pin
- analogRead() : Reads from the analog input pin i.e., voltage applied across the pin
- Character functions such as isdigit(), isalpha(), isalnum(), isxdigit(), islower(), isupper(), isspace() return 1(true) or 0(false)
- Delay() function is one of the most common time manipulation function used to provide a delay of specified time. It accepts integer value (time in milliseconds)

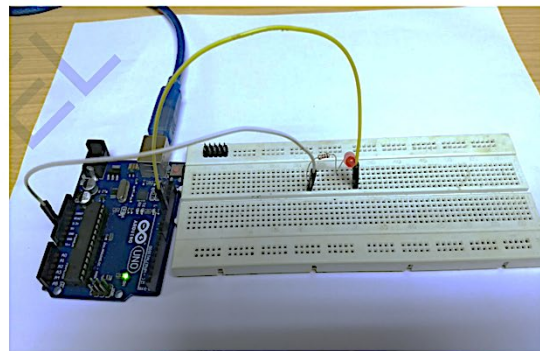
Example- Blinking LED

- Requirement:
 - Arduino controller board, USB connector, Bread board, LED, 1.4Kohm resistor, connecting wires, Arduino IDE
- Connect the LED to the Arduino using the Bread board and the connecting wires
- Connect the Arduino board to the PC using the USB connector
- Select the board type and port
- Write the sketch in the editor, verify and upload.



Example- Blink (contd..)

Connect the positive terminal of the LED to digital pin 12 and the negative terminal to the ground pin (GND) of Arduino Board



Example- Blink (contd..) image setup

```
void setup() {  
    pinMode(12, OUTPUT); // set the pin mode  
}  
void loop() {  
    digitalWrite(12, HIGH); // Turn on the LED  
    delay(1000);  
    digitalWrite(12, LOW); //Turn of the LED  
    delay(1000);  
}
```

Example- Blink (contd..)



```
File Edit Sketch Tools Help
sketch_arduino | Arduino 1.8.2

void setup() {
  // set the pin mode
  pinMode(12, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(12, HIGH); // Turn on the LED
  delay(2000);
  digitalWrite(12, LOW); // Turn on the LED
  delay(2000);
}
```

Done Uploading

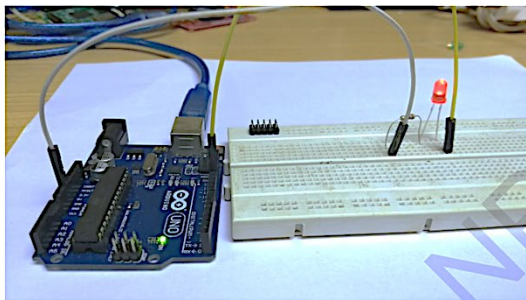
Sketch uses 800 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.

Set the pin mode as output which is connected to the led, pin 12 in this case.

Use digitalWrite() function to set the output as HIGH and LOW

Delay() function is used to specify the delay between HIGH-LOW transition of the output

Example- Blink (contd..) image setup



- Connect the board to the PC
- Set the port and board type
- Verify the code and upload, notice the TX – RX led in the board starts flashing as the code is uploaded.

Content

- Operators in Arduino
- Control Statement
- Loops
- Arrays
- String
- Math Library
- Random Number
- Interrupts
- Example Program

Operators

- Arithmetic Operators: =, +, -, *, /, %
- Comparison Operator: ==, !=, <, >, <=, >=
- Boolean Operator: &&, ||, !
- Bitwise Operator: &, |, ^, ~, <<, >> ,
- Compound Operator: ++, --, +=, -=, *=, /=, %=, |=, &=

Control Statement

- If statement
 - if(condition){
Statements if the condition is true ;
}
- If...Else statement
 - if(condition){
Statements if the condition is true;
}
else{
Statements if the condition is false;
}
- If.....Elseif.....Else
 - if (condition1){
Statements if the condition1 is true;
}
else if (condition2){
Statements if the condition1 is false and condition2 is true;
}
else{
Statements if both the conditions are false;
}

Control Statement (contd..)

- Switch Case
 - Switch(choice)
{
 case opt1: statement_1;break;
 case opt2: statement_2;break;
 case opt3: statement_3;break;
 .
 .
 .
 case default: statement_default; break;
}
- Conditional Operator.
 - Val=(condition)?(Statement1): (Statement2)

Loops

- For loop
 - for(initialization; condition; increment){
 Statement till the condition is true;
}
- While loop
 - while(condition){
 Statement till the condition is true;
}
- Do... While loop
 - do{
 Statement till the condition is true;
}while(condition);

Loops (contd..)

- Nested loop: Calling a loop inside another loop
- Infinite loop: Condition of the loop is always true, the loop will never terminate

Arrays

- Collection of elements having homogenous datatype that are stored in adjacent memory location.
- The conventional starting index is 0.
- Declaration of array:
 <Datatype> array_name[size];
 Ex: int arr[5];

Arrays (contd..)

- Alternative Declaration:
`int arre[]={0,1,2,3,4};`
`int arre[5]={0,1,2};`
- Multi-dimentional array Declaration:
`<Datatype> array_name[n1] [n2][n3]....;`
Ex: `int arre[row][col][height];`

String

- Array of characters with NULL as termination is termed as a String.
- Declaration using Array:
 - `char str[]="ABCD";`
 - `char str[4];`
 - `str[0]='A';`
 - `str[0]='B';`
 - `str[0]='C';`
 - `str[0]=0;`
- Declaration using String Object:
 - `String str="ABC";`

String (contd..)

- Functions of String Object:
 - `str.ToUpperCase()`: change all the characters of str to upper case
 - `str.replace(str1,str2)`: if str1 is the sub string of str then it will be replaced by str2
 - `str.length()`: returns the length of the string without considering null

Math Library

- To apply the math functions and mathematical constants, "**MATH.h**" header files is needed to be included.
- Functions:
 - `cos(double radian);`
 - `sin(double radian);`
 - `tan(double radian);`
 - `fabs(double val);`
 - `fmod(double val1, double val2);`

Math Library (contd..)

- Functions:
 - `exp(double val);`
 - `log(double val);`
 - `log10(double val);`
 - `square(double val);`
 - `pow(double base, double power);`

Random Number

- `randomSeed(int v):` reset the pseudo-random number generator with seed value v
- `random(maxi)=`gives a random number within the range [0,maxi]
- `random(mini,maxi)=`gives a random number within the range [mini,maxi]

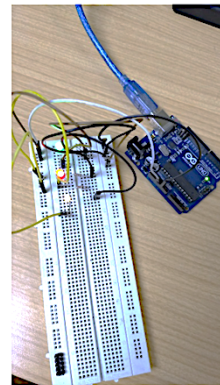
Interrupts

- An external signal for which system blocks the current running process to process that signal
- Types:
 - Hardware interrupt
 - Software interrupt
- `digitalPinToInterrupt(pin)`: Change actual digital pin to the specific interrupt number.
- `attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);`
 - ISR: a interrupt service routine have to be defined

Example: Traffic Control System

Requirement:

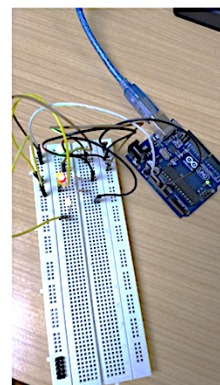
- Arduino Board
- 3 different color LEDs
- 330 Ohm resistors
- Jumper wires



Example: Traffic Control System (contd..)

Connection:

- Connect the positive terminals of the LEDs to the respective digital output pins in the board, assigned in the code.
- Connect the negative terminals of the LEDs to the ground



Example: Traffic Control System (contd..)

- Sketch

```
//LED pins
int r = 2;
int g = 3;
int y = 4;
void setup()
{
    Serial.begin(9600);
    pinMode(r, OUTPUT); digitalWrite(r,LOW);
    pinMode(g, OUTPUT); digitalWrite(g,LOW);
    pinMode(y , OUTPUT); digitalWrite(y, LOW);
}
```

Example: Traffic Control System (contd..)

```
void traffic()
{
    digitalWrite(g, HIGH);
    Serial.println("Green LED: ON, GO");
    // delay of 5 seconds
    delay(5000);
    digitalWrite(g, LOW);
    digitalWrite(y, HIGH);
    Serial.println("Green LED: OFF ; Yellow LED: ON, WAIT");
    delay(5000);
}
```

Example: Traffic Control System (contd..)

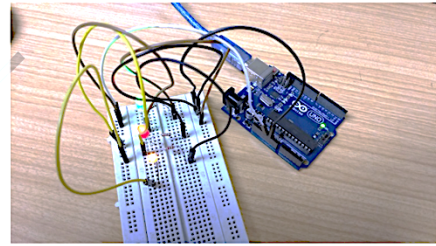
```
        digitalWrite(y, LOW);
        digitalWrite(r, HIGH);
        Serial.println("Yellow LED: OFF ; Red LED: ON, STOP");
        delay(5000);                // for 5 seconds
        digitalWrite(r, LOW);
        Serial.println("All OFF");
    }

void loop()
{
    traffic ();
    delay (10000);
}
```

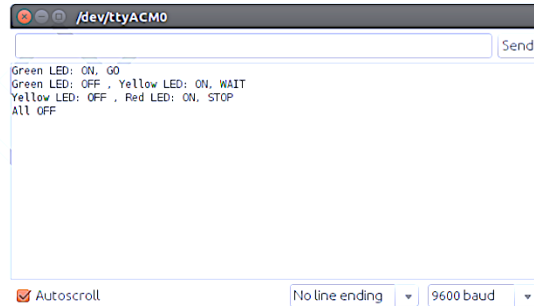
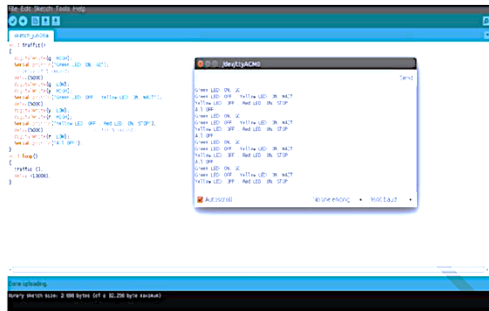
Example: Traffic Control System (contd..)

Output:

- Initially, all the LEDs are turned off
- The LEDs are turned on one at a time with a delay of 5 seconds
- The message is displayed accordingly
- Figure showing all the LEDs turned on

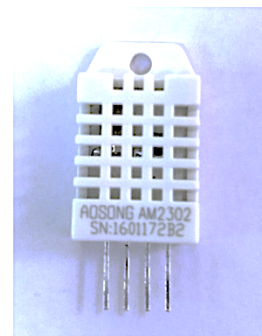


Output



Sensor Interface with Arduino

- Digital Humidity and Temperature Sensor (DHT)
- PIN 1, 2, 3, 4 (from left to right)
 - PIN 1- 3.3V-5V Power supply
 - PIN 2- Data
 - PIN 3- Null
 - PIN 4- Ground

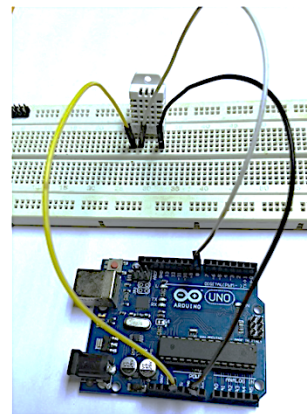


DHT Sensor Library

- Arduino supports a special library for the DHT11 and DHT22 sensors
- Provides function to read the temperature and humidity values from the data pin
`dht.readHumidity()`
`dht.readTemperature()`

Connection

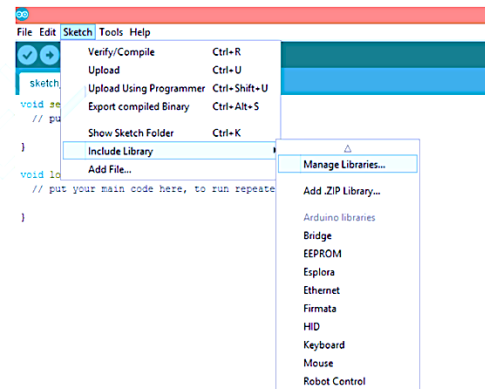
- Connect pin 1 of the DHT to the 3.3 V supply pin in the board
- Data pin (pin 2) can be connected to any digital pin, here 12
- Connect pin 4 to the ground (GND) pin of the board



Sketch: DHT_SENSOR

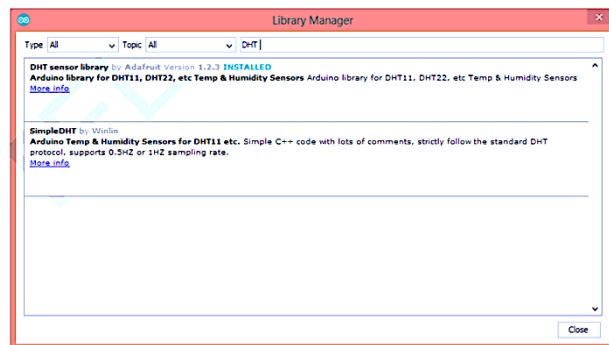
Install the DHT Sensor Library

- Go to Sketch -> Include Library -> Manage Library



Sketch: DHT_SENSOR (contd..)

- Search for DHT SENSOR
- Select the “DHT sensor library” and install it

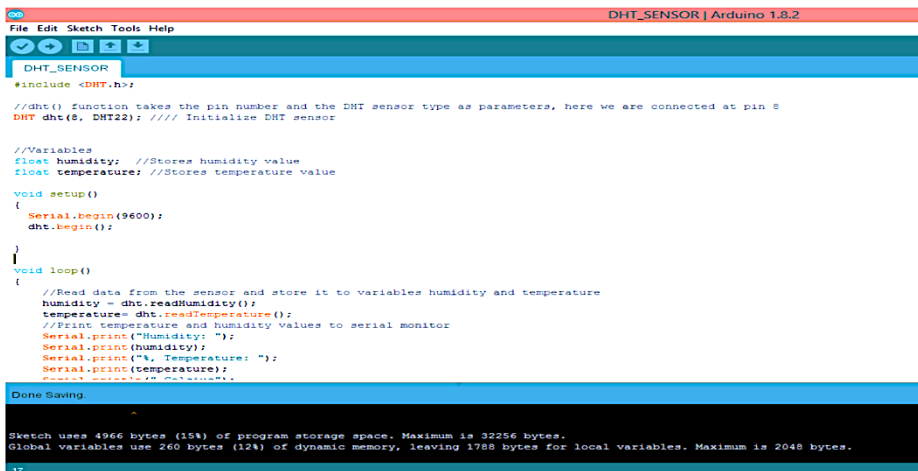


Sketch: DHT_SENSOR (contd..)

```
#include <DHT.h>
DHT dht(8, DHT22); //Initialize DHT sensor
float humidity; //Stores humidity value
float temperature; //Stores temperature value
void setup()
{
  Serial.begin(9600);
  dht.begin();
}

void loop()
{
  //Read data from the sensor and store it to variables
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
  //Print temperature and humidity values to serial
  monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print("%, Temperature: ");
  Serial.print(temperature);
  Serial.println(" Celsius");
  delay(2000); //Delay of 2 seconds
}
```

Sketch: DHT_SENSOR (contd..)



The screenshot shows the Arduino IDE interface with the sketch titled "DHT_SENSOR | Arduino 1.8.2". The code in the editor is as follows:

```
File Edit Sketch Tools Help
DHT_SENSOR
#include <DHT.h>

//dht() function takes the pin number and the DHT sensor type as parameters, here we are connected at pin 8
DHT dht(8, DHT22); //Initialize DHT sensor

//Variables
float humidity; //Stores humidity value
float temperature; //Stores temperature value

void setup()
{
  Serial.begin(9600);
  dht.begin();
}

void loop()
{
  //Read data from the sensor and store it to variables humidity and temperature
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
  //Print temperature and humidity values to serial monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print("%, Temperature: ");
  Serial.print(temperature);
  Serial.println(" Celsius");
  delay(2000); //Delay of 2 seconds
}
```

Below the code editor, the "Done Saving" message is visible. At the bottom, the memory usage summary is displayed:

```
Sketch uses 4966 bytes (15%) of program storage space. Maximum is 32256 bytes.
Global variables use 260 bytes (12%) of dynamic memory, leaving 1788 bytes for local variables. Maximum is 2048 bytes.
```

Sketch: DHT_SENSOR (contd..)

- Connect the board to the PC
- Set the port and board type
- Verify and upload the code

Output

The readings are printed at a delay of 2 seconds as specified by the delay() function

```
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
```

Actuators

- Mechanical/Electro-mechanical device
- Converts energy into motion
- Mainly used to provide controlled motion to other components

Basic Working Principle

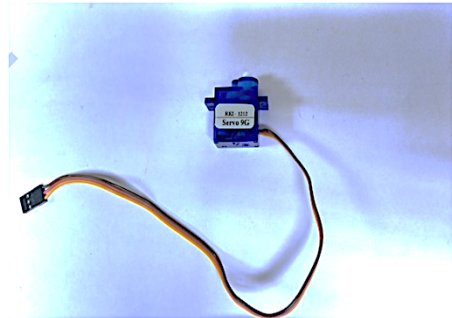
Uses different combination of various mechanical structures like screws, ball bearings, gears to produce motion.

Types of Motor Actuators

- Servo motor
- Stepper motor
- Hydraulic motor
- Solenoid
- Relay
- AC motor

Servo Motor

- High precision motor
- Provides rotary motion 0 to 180 degree
- 3 wires in the Servo motor
 - Black or the darkest one is Ground
 - Red is for power supply
 - Yellow for signal pin



Servo Library on Arduino

- Arduino provides different library- SERVO to operate the servo motor
- Create an instance of servo to use it in the sketch

Servo myservo;

Sketch: SERVO_ACTUATOR

```
#include <Servo.h>
//Including the servo library for the program
int servoPin = 12;

Servo ServoDemo; // Creating a servo object
void setup() {
    // The servo pin must be attached to the servo
    before it can be used
    ServoDemo.attach(servoPin);
}
```

```
void loop(){
    //Servo moves to 0 degrees
    ServoDemo.write(0);
    delay(1000);

    // Servo moves to 90 degrees
    ServoDemo.write(90);
    delay(1000);

    // Servo moves to 180 degrees
    ServoDemo.write(180);
    delay(1000);
}
```

Sketch: SERVO_ACTUATOR (contd..)

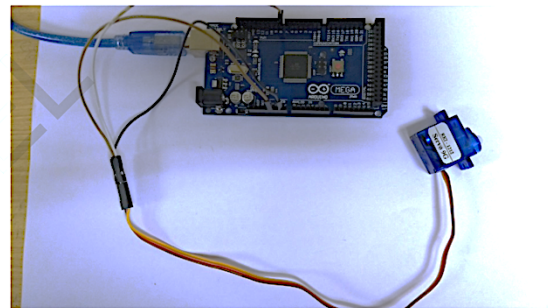
- Create an instance of Servo
- The instance must be attached to the pin before being used in the code
- Write() function takes the degree value and rotates the motor accordingly

```
File Edit Sketch Tools Help
SERVO_ACTUATOR
#include <Servo.h> //Including the servo library for the program
int servoPin = 8;

Servo ServoDemo; // Creating a servo object
void setup() {
  // The servo pin must be attached to the servo before it can be used
  ServoDemo.attach(servoPin);
}
void loop(){
  //Servo moves to 0 degrees
  ServoDemo.write(0);
  delay(1000);
  // Servo moves to 90 degrees
  ServoDemo.write(90);
  delay(1000);
  // Servo moves to 180 degrees
  ServoDemo.write(180);
  delay(1000);
}
```

Connection

- Connect the Ground of the servo to the ground of the Arduino board.
- Connect the power supply wire to the 5V pin of the board.
- Connect the signal wire to any digital output pin (we have used pin 8).

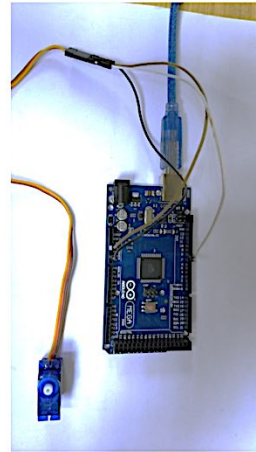


Board Setup

- Connect the board to the PC
- Set the port and board type
- Verify and upload the code

Output

The motor turns 0, 90 and 180 degrees with a delay of 1 second each.



Do more with the Servo library

Some other functions available with the Servo library:

- Knob()
- Sweep()
- write()
- writeMicroseconds()
- read()
- attached()
- detach()

Chapter-7

Programming with Raspberry Pi

7.1 Architecture and Pin Configuration

7.2 Case studies

7.3 Implementation of IoT with Raspberry Pi

7.0 What is Raspberry Pi?

- Computer in your palm.
- Single-board computer.
- Low cost.
- Easy to access.

Raspberry Pi

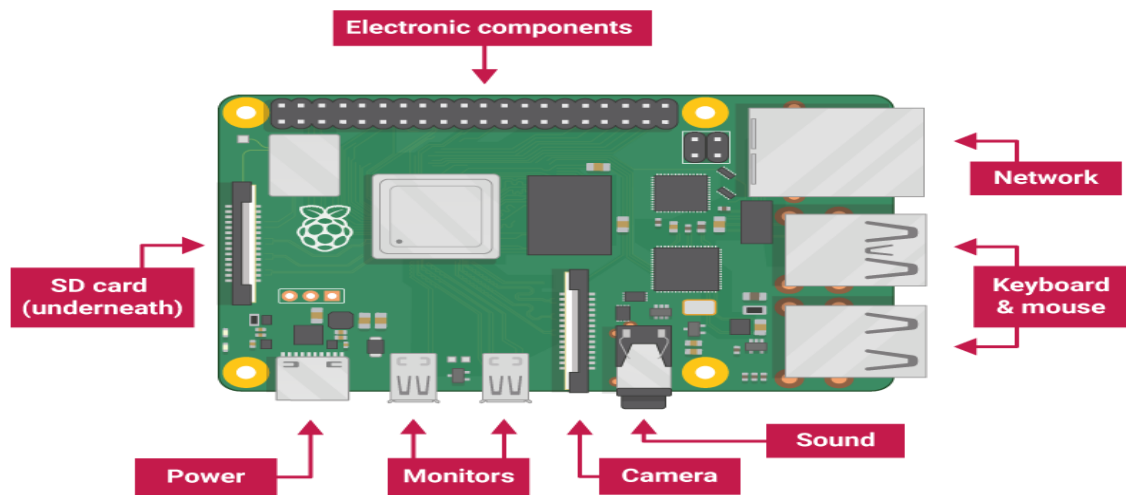
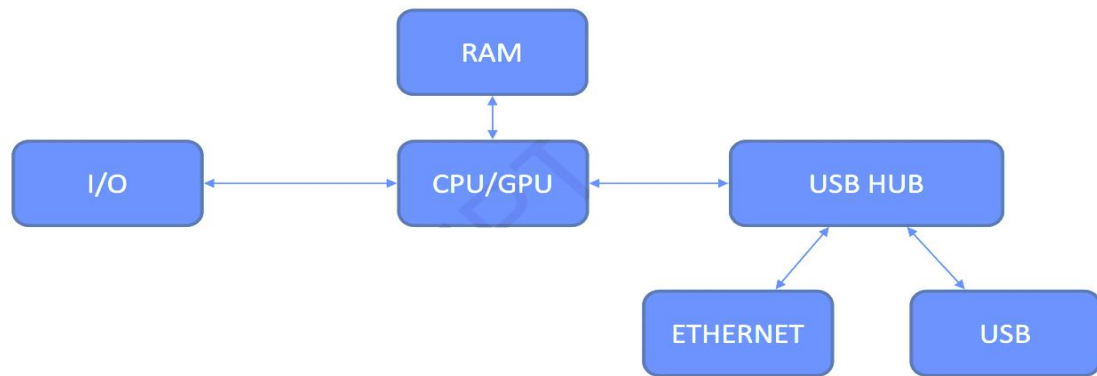


7.0.1 Specifications of commonly used Ver. Of Raspberry Pi

Key features	Raspberry pi 3 model B	Raspberry pi 2 model B	Raspberry Pi zero
RAM	1GB SDRAM	1GB SDRAM	512 MB SDRAM
CPU	Quad cortex A53@1.2GHz	Quad cortex A53@900MHz	ARM 11@ 1GHz
GPU	400 MHz video core IV	250 MHz video core IV	250 MHz video core IV
Ethernet	10/100	10/100	None
Wireless	802.11/Bluetooth 4.0	None	None
Video output	HDMI/Composite	HDMI/Composite	HDMI/Composite
GPIO	40	40	40

7.1 Architecture and Pin Configuration

Basic Architecture



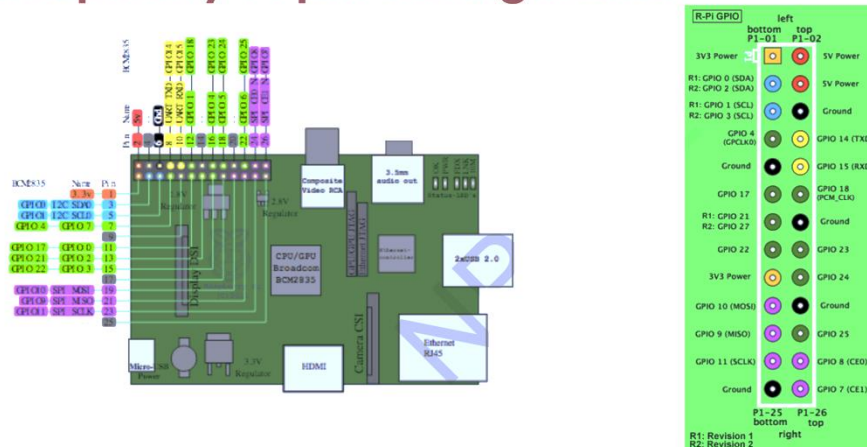
Start up raspberry pi



Raspberry Pi GPIO

- Act as both digital output and digital input.
- **Output:** turn a GPIO pin high or low.
- **Input:** detect a GPIO pin high or low.

Raspberry Pi pin configuration



Basic Set up for Raspberry Pi

- HDMI cable.
- Monitor.
- Key board.
- Mouse.
- 5volt power adapter for raspberry pi.
- LAN cable .
- Min- 2GB micro sd card

Basic Set up for Raspberry Pi



Operating System

Official Supported OS :

- Raspbian
- NOOBS

Some of the third party OS :

- UBUNTU mate
- Snappy Ubuntu core
- Windows 10 core
- Pinet
- Risc OS

Sl. No.	Arduino	Raspberry Pi
1.	Control unit of Arduino is from ATmega family.	While control unit of Raspberry Pi is from ARM family.
2.	Arduino is based on a microcontroller.	While Raspberry Pi is based on a microprocessor.
3.	It is designed to control the electrical components connected to the circuit board in a system.	While Raspberry Pi computes data and produces valuable outputs, and controls components in a system based on the outcome of its computation.
4.	Arduino boards have a simple hardware and software structure.	While Raspberry Pi boards have a complex architecture of hardware and software.
5.	CPU architecture: 8 bit.	CPU architecture: 64 bit.
6.	It uses very less RAM, 2 kB.	While Raspberry Pi requires more RAM, 1 GB.
7.	It clocks a processing speed of 16 MHz.	While Raspberry Pi clocks a processing speed of 1.4 GHz.
8.	It is cheaper in cost.	While Raspberry Pi is expensive.
9.	It has a higher I/O current drive strength.	While Raspberry Pi has a lower I/O current drive strength.
10.	It consumes about 200 MW of power.	While it consumes about 700 MW of power.

Raspberry Pi Setup

Download Raspbian:

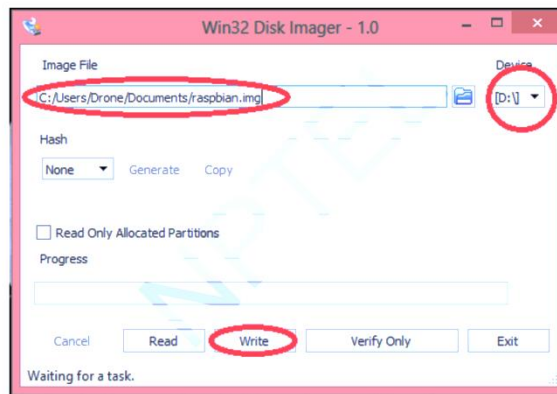
- Download latest Raspbian image from raspberry pi official site:
<https://www.raspberrypi.org/downloads/>
- Unzip the file and end up with an .img file.

Raspberry Pi OS Setup

Write Raspbian in SD card :

- Install “Win32 Disk Imager” software in windows machine .
- Run Win32 Disk Imager
- Plug SD card into your PC
- Select the “Device”
- Browse the “Image File”(Raspbian image)
- Write

Raspberry Pi OS Setup



Basic Initial Configuration

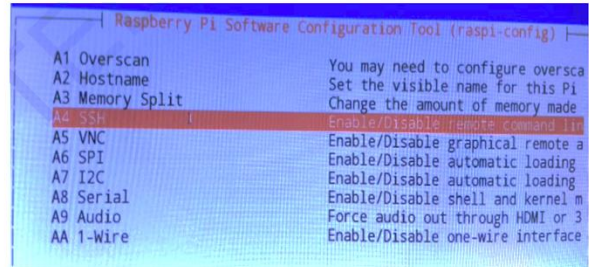
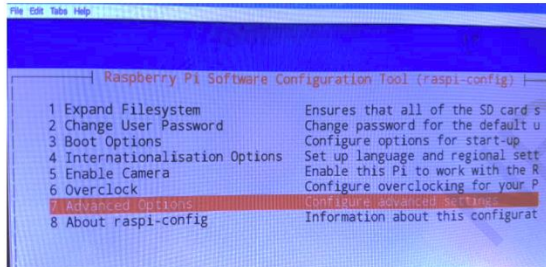
Enable SSH

Step1 : Open command prompt and type **sudo raspi-config** and press enter.

Step2: Navigate to SSH in the Advance option.

Step3: Enable SSH

Basic Initial Configuration



Basic Initial Configuration contd.

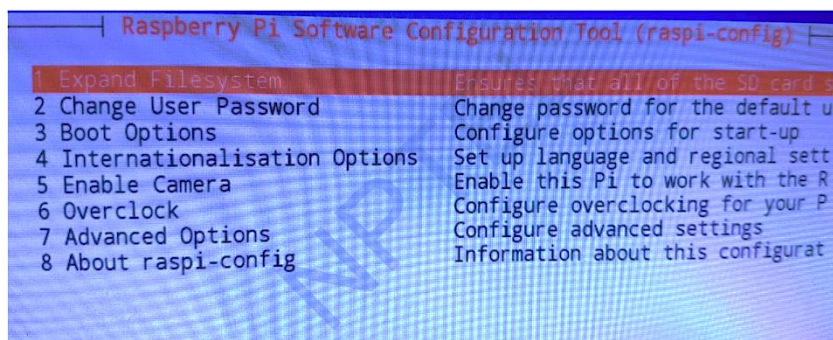
Expand file system :

Step 1: Open command prompt and type `sudo raspi-config` and press enter.

Step 2: Navigate to Expand Filesystem

Step 3: Press enter to expand it.

Basic Initial Configuration contd.



Programming

Default installed :

- Python
- C
- C++
- Java
- Scratch
- Ruby

Note : Any language that will compile for ARMv6 can be used with raspberry pi.

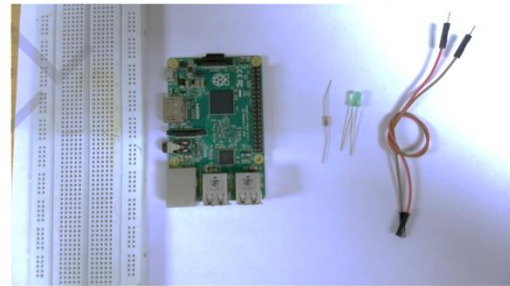
Popular Applications

- Media streamer
- Home automation
- Controlling BOT
- VPN
- Light weight web server for IOT
- Tablet computer

7.2 CASE STUDIES

7.2.1 Blinking LED

- **Requirement:**
- **Raspberry pi**
- **LED**
- **100 ohm resistor**
- **Bread board**
- **Jumper cables**

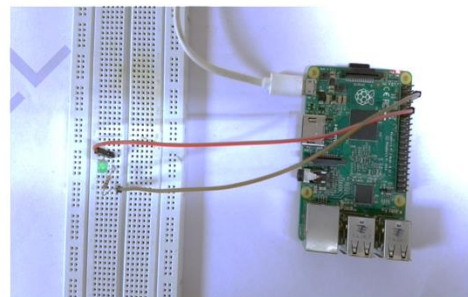


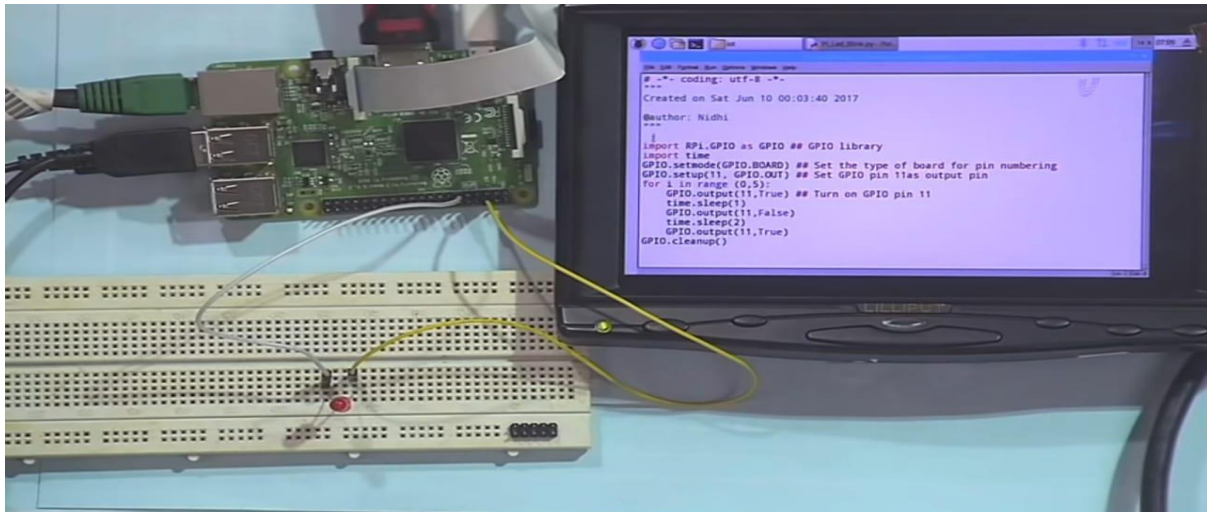
Installing GPIO library:

- Open terminal
- Enter the command “sudo apt-get install python-dev” to install python development
- Enter the command “sudo apt-get install python-rpi.gpio” to install GPIO library.

Connection:

- Connect the negative terminal of the LED to the ground pin of Pi
- Connect the positive terminal of the LED to the output pin of Pi





Basic python coding:

- Open terminal enter the command
sudo nano filename.py
- This will open the nano editor where you can write your code
- Ctrl+O : Writes the code to the file
- Ctrl+X : Exits the editor

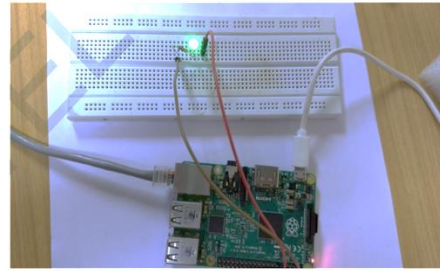
Code:

```
import RPi.GPIO as GPIO      #GPIO library
import time
GPIO.setmode(GPIO.BOARD)    # Set the type of board for pin numbering
GPIO.setup(11, GPIO.OUT)    # Set GPIO pin 11 as output pin
for i in range (0,5):
    GPIO.output(11,True)    # Turn on GPIO pin 11
    time.sleep(1)
    GPIO.output(11,False)
    time.sleep(2)
    GPIO.output(11,True)
GPIO.cleanup()
```

```
GNU nano 2.2.6                                File: BLINK_LED.py
import RPi.GPIO as GPIO ## GPIO library
import time
GPIO.setmode(GPIO.BOARD) ## Set the type of board for pin numbering
GPIO.setup(11, GPIO.OUT) ## Set GPIO pin 11 as output pin
for i in range (0,5):
    GPIO.output(11,True) ## Turn on GPIO pin 11
    time.sleep(1)
    GPIO.output(11,False)
    time.sleep(2)
    GPIO.output(11,True)
GPIO.cleanup()
```

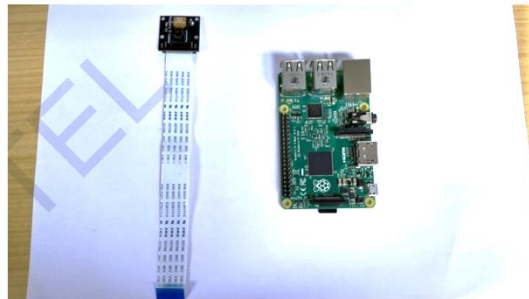

Result

The LED blinks in a loop with delay of 1 and 2 seconds.



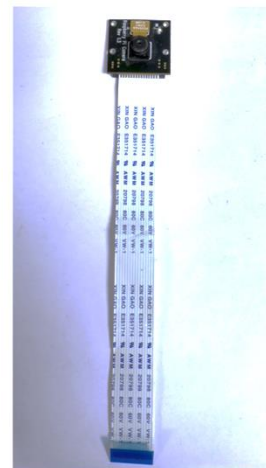
7.2.2 Taking Picture using PiCam Requirement

- Raspberry Pi
- Raspberry Pi Camera



Raspberry Pi Camera

- Raspberry Pi specific camera module
- Dedicated CSI slot in Pi for connection
- The cable slot is placed between Ethernet port and HDMI port



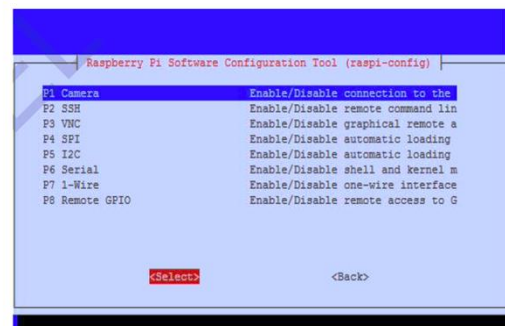
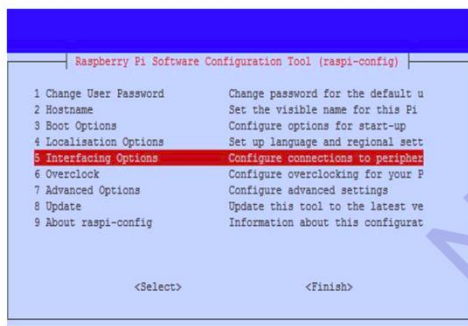
Connection

Boot the Pi once the camera is connected to Pi



Configuring Pi for Camera

- In the terminal run the command “sudo raspi-config” and press enter.
- Navigate to “Interfacing Options” option and press enter.
- Navigate to “Camera” option.
- Enable the camera.
- Reboot Raspberry pi.



Capture Image

- Open terminal and enter the command-

`raspistill -o image.jpg`

- This will store the image as 'image.jpg'

PiCam can also be processed using Python camera module `python-picamera`

```
sudo apt-get install python-picamera
```

Python Code:

```
import picamera  
camera = picamera.PiCamera()  
camera.capture('image.jpg')
```



Capture Image (contd..)

```
pi@raspberrypi:~$ raspistill -o image.jpg  
pi@raspberrypi:~$
```



7.3 Implementation of IoT with Raspberry Pi IOT

Internet Of Things

- Creating an interactive environment
- Network of devices connected together

Sensor

- Electronic element
- Converts physical quantity into electrical signals
- Can be analog or digital

Actuator

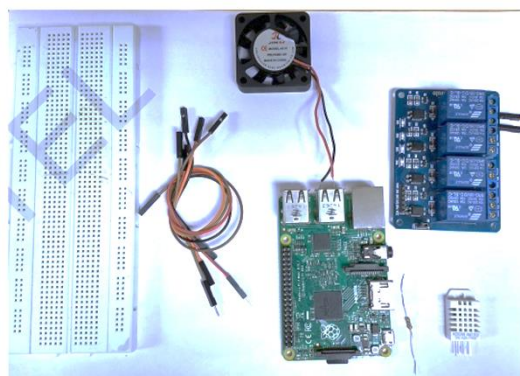
- Mechanical/Electro-mechanical device
- Converts energy into motion
- Mainly used to provide controlled motion to other components

System Overview

- Sensor and actuator interfaced with Raspberry Pi
- Read data from the sensor
- Control the actuator according to the reading from the sensor
- Connect the actuator to a device

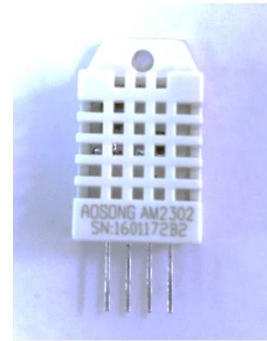
Requirements

- DHT Sensor
- 4.7K ohm resistor
- Relay
- Jumper wires
- Raspberry Pi
- Mini fan



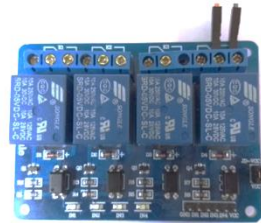
DHT Sensor

- Digital Humidity and Temperature Sensor (DHT)
- PIN 1, 2, 3, 4 (from left to right)
 - PIN 1- 3.3V-5V Power supply
 - PIN 2- Data
 - PIN 3- Null
 - PIN 4- Ground



Relay

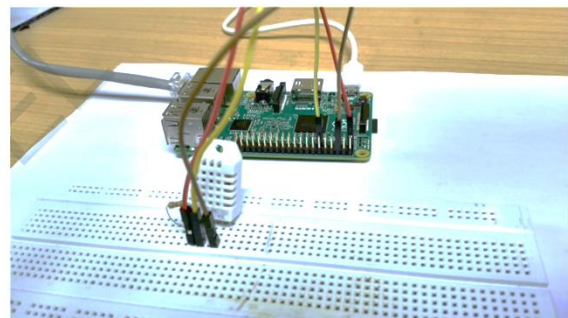
- Mechanical/electromechanical switch
- 3 output terminals (left to right)
 - NO (normal open):
 - Common
 - NC (normal close)



Temperature Dependent Auto Cooling System

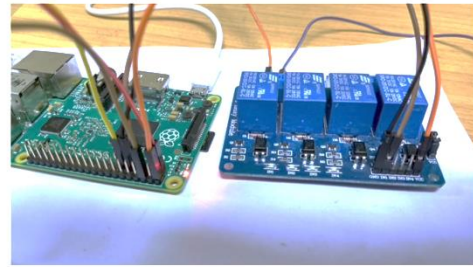
Sensor interface with Raspberry Pi

- Connect pin 1 of DHT sensor to the 3.3V pin of Raspberry Pi
- Connect pin 2 of DHT sensor to any input pins of Raspberry Pi, here we have used pin 11
- Connect pin 4 of DHT sensor to the ground pin of the Raspberry Pi



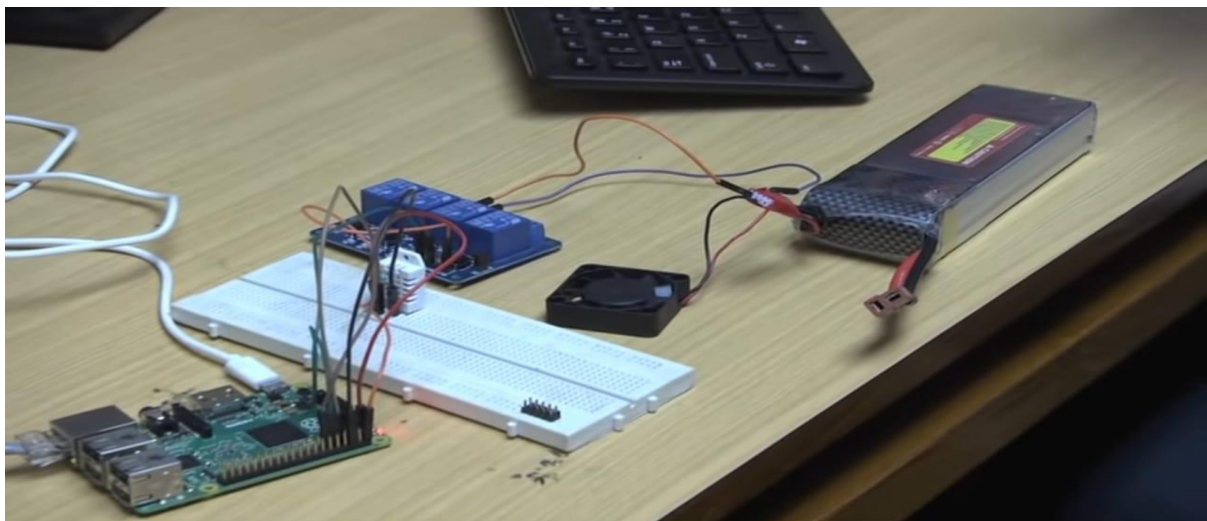
Relay interface with Raspberry Pi

- Connect the VCC pin of relay to the 5V supply pin of Raspberry Pi
- Connect the GND (ground) pin of relay to the ground pin of Raspberry Pi
- Connect the input/signal pin of Relay to the assigned output pin of Raspberry Pi (Here we have used pin 7)



Adafruit provides a library to work with the DHT22 sensor

- Install the library in your Pi-
 - Get the clone from GIT
git clone [https://github.com/adafruit/Adafruit_Python_DHT.g...](https://github.com/adafruit/Adafruit_Python_DHT)
 - Go to folder Adafruit_Python_DHT
cd Adafruit_Python_DHT
 - Install the library
sudo python setup.py install



Program: DHT22 with Pi

```
import RPi.GPIO as GPIO
from time import sleep
import Adafruit_DHT                                     #importing the Adafruit library

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
sensor = Adafruit_DHT.AM2302                            # create an instance of the sensor type
print ('Getting data from the sensor')
#humidity and temperature are 2 variables that store the values received from the sensor

humidity, temperature = Adafruit_DHT.read_retry(sensor,17)
print ('Temp={0:0.1f}*C humidity={1:0.1f}%'.format(temperature, humidity))
```

Program: DHT22 interfaced with Raspberry Pi

Code

```
GNU nano 2.2.6 File: IOTSR.py
import RPi.GPIO as GPIO
from time import sleep

import Adafruit_DHT

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

sensor = Adafruit_DHT.AM2302 # create an instance of the sensor type

print ('Getting data from the sensor')

#humidity and temperature are 2 variables that store the values received from the sensor
humidity, temperature = Adafruit_DHT.read_retry(sensor,17)

print ('Temp={0:0.1f}*C humidity={1:0.1f}%'.format(temperature, humidity))
```

Output

```
pi@raspberrypi:~ $ python IOTSR.py
Getting data from the sensor
Temp=26.1*C humidity=65.9%
pi@raspberrypi:~ $
```

Connection: Relay

- Connect the relay pins with the Raspberry Pi as mentioned in previous slides
- Set the GPIO pin connected with the relay's input pin as output in the sketch
GPIO.setup(13,GPIO.OUT)
- Set the relay pin high when the temperature is greater than 30
if temperature > 30: *pin is low*
GPIO.output(13,0) # Relay is active low *low*
print('Relay is on')
sleep(5)
GPIO.output(13,1) # Relay is turned off after delay of 5 seconds

```
GNU nano 2.2.6 File: IOTSR.py
import RPi.GPIO as GPIO
from time import sleep

import Adafruit_DHT

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(7,GPIO.OUT)

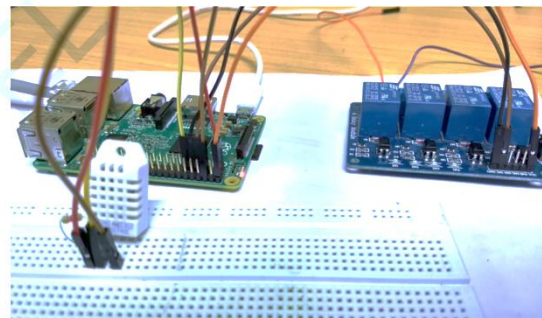
sensor = Adafruit_DHT.AM2302 # create an instance of the sensor type

print ('Getting data from the sensor')

#humidity and temperature are 2 variables that store the values received from the sensor
humidity, temperature = Adafruit_DHT.read_retry(sensor,17)

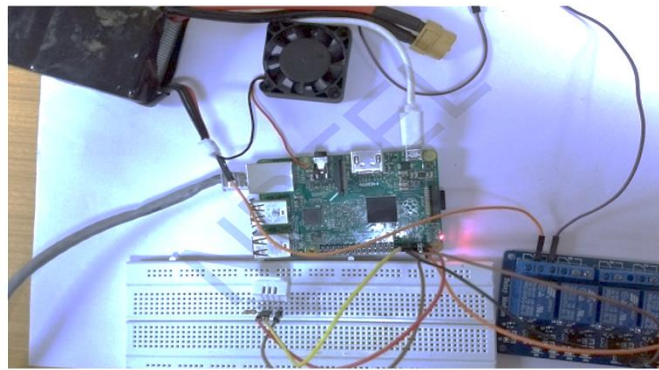
print ('Temp={0:0.1f}*C humidity={1:0.1f}%'.format(temperature, humidity))

if temperature > 20:
    GPIO.output(7,0) # Relay is active low
    print('Relay is on')
    sleep(5)
    GPIO.output(7,1) # Relay is turned off after delay of 5 seconds
```



Connection: Fan

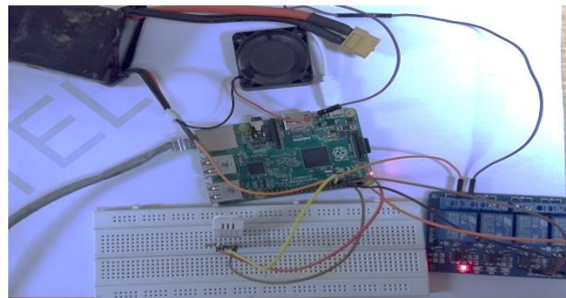
- Connect the Li-po battery in series with the fan
 - NO terminal of the relay -> positive terminal of the Fan.
 - Common terminal of the relay -> Positive terminal of the battery
 - Negative terminal of the battery -> Negative terminal of the fan.
- Run the existing code. The fan should operate when the surrounding temperature is greater than the threshold value in the sketch



Result

The fan is switched on whenever the temperature is above the threshold value set in the code.

Notice the relay indicator turned on.



IOT: Remote Data Logging

- Collect data from the devices in the network
- Send the data to a server/remote machine
- Control the network remotely

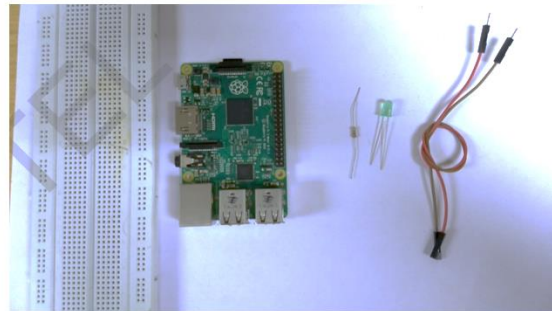
IOT: Remote Data Logging

System Overview:

- A network of Temperature and humidity sensor connected with Raspberry Pi
- Read data from the sensor
- Send it to a Server
- Save the data in the server

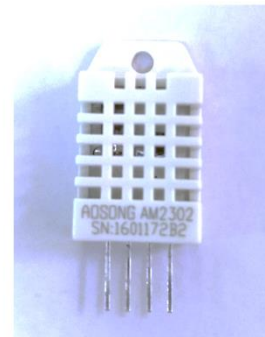
Requirements

- DHT Sensor
- 4.7K ohm resistor
- Jumper wires
- Raspberry Pi



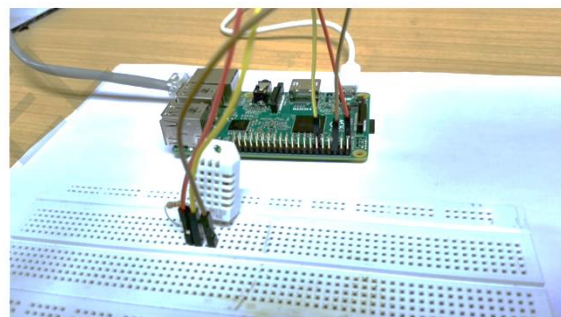
DHT Sensor

- Digital Humidity and Temperature Sensor (DHT)
- PIN 1, 2, 3, 4 (from left to right)
 - PIN 1- 3.3V-5V Power supply
 - PIN 2- Data
 - PIN 3- Null
 - PIN 4- Ground



Sensor- Raspberry Pi Interface

- Connect pin 1 of DHT sensor to the 3.3V pin of Raspberry Pi
- Connect pin 2 of DHT sensor to any input pins of Raspberry Pi, here we have used pin 11
- Connect pin 4 of DHT sensor to the ground pin of the Raspberry Pi



Read Data from the Sensor

Adafruit provides a library to work with the DHT22 sensor

Install the library in Raspberry Pi

Use the function `Adafruit_DHT.read_retry()` to read data from the sensor

Program: DHT22 interfaced with Raspberry Pi

Code

```
GNU nano 2.2.6 File: IOTSR.py
import RPi.GPIO as GPIO
from time import sleep

import Adafruit_DHT

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

sensor = Adafruit_DHT.AM2302 # create an instance of the sensor type
print ('Getting data from the sensor')

#humidity and temperature are 2 variables that store the values received from the sensor
humidity, temperature = Adafruit_DHT.read_retry(sensor,17)

print ('Temp={0:0.1f}*C humidity={1:0.1f}%'.format(temperature, humidity))
```

Output

```
pi@raspberrypi:~ $ python IOTSR.py
Getting data from the sensor
Temp=26.1*C humidity=65.9%
pi@raspberrypi:~ $
```

Sending Data to a Server

Sending data to Server using network protocols

- Create a server and client
- Establish connection between the server and the client
- Send data from the client to the server

Sending Data to a Server (contd..)

Socket Programming:

- Creates a two-way communication between two nodes in a network
- The nodes are termed as Server and Client
- Server performs the task/service requested by the client

Sending Data to a Server (contd..)

Creating a socket:

```
s = socket.socket(SocketFamily, SocketType, Protocol=0)
```

- ✓ SocketFamily can be AF_UNIX or AF_INET
- ✓ SocketType can be SOCK_STREAM or SOCK_DGRAM
- ✓ Protocol is set default to 0

Sending Data to a Server (contd..)

Server:

```
s = socket.socket()          # creating a socket object
host = socket.gethostname()  # local machine name/address
port = 12321                 # port number for the server
s.bind((host, port))         # bind to the port
s.listen(5)                  # waiting for the client to connect
while True:
    c, addr = s.accept()      # accept the connection request from the client
    print 'Connected to', addr
    c.send('Connection Successful')
c.close()                    #close the socket
```

Sending Data to a Server (contd..)

Client:

```
s = socket.socket()          # creating a socket object
host = socket.gethostname()  # getting local machine name
port = 12345                 # assigning a port
s.connect((host, port))
print s.recv(1024)
s.close
```

Sending Data to a Server (contd..)

Client Code: Obtain readings from the sensor

```
def sensordata():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)
    sensor = Adafruit_DHT.AM2302
    humidity, temperature = Adafruit_DHT.read_retry(sensor,17)
    return(humidity, temperature)
```

This function returns the values from the DHT sensor

Sending Data to a Server (contd..)

Client Code: Connecting to the server and sending the data

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)           #create UDP socket
server_address = ('10.14.3.194', 10001)
try:
    while (1):
        h,t = sensordata()
        message = str(h)+' '+str(t)
        #Send data
        print >>sys.stderr, 'sending "%s"' % message
        sent = sock.sendto(message, server_address)
finally:
    print >>sys.stderr, 'closing socket'
    sock.close()
```

75.27

Sending Data to a Server (contd..)

Server Code: Receive data from client and save it

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)           # Create a UDP socket
server_address = ('10.14.3.194', 10001)
sock.bind(server_address)                                         # Bind the socket to the port
while True:
    data, address = sock.recvfrom(4096)
    with open("DataLog.txt", "a") as f:
        mess=str(data)
        f.write(mess)
        print mess
    f.close()
```



Result

- The client takes reading from the sensor and sends it to the server
- The server receives the data from the client and saves it in a text file DataLog.txt

```
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.3999996185
68.9000015259,23.3999996185
68.9000015259,23.5
68.9000015259,23.5
68.8000030518,23.3999996185
68.9000015259,23.5
68.9000015259,23.5
68.8000030518,23.3999996185
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.3999996185
68.9000015259,23.5
68.9000015259,23.3999996185
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
```

IOT: Remote Data ~~Logging~~ ^{Plotting}

- Collect data from the devices in the network
- Send the data to a server/remote machine
- Processing the data ✓
- Respond to the network ✓

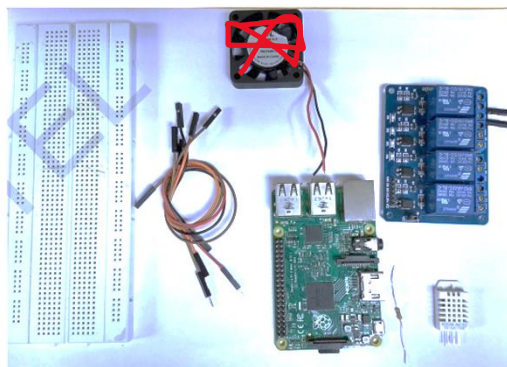
System Overview:

- A network of Temperature and humidity sensor connected with Raspberry Pi
- Read data from the sensor
- Send it to a Server
- Save the data in the server
- **Data Splitting**
- **Plot the data**

Requirements

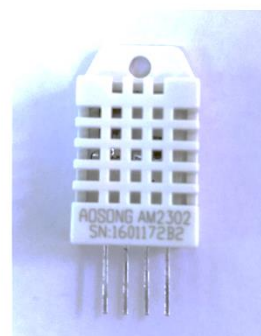
- DHT Sensor
- 4.7K ohm resistor
- Jumper wires
- Raspberry Pi

• Bread board



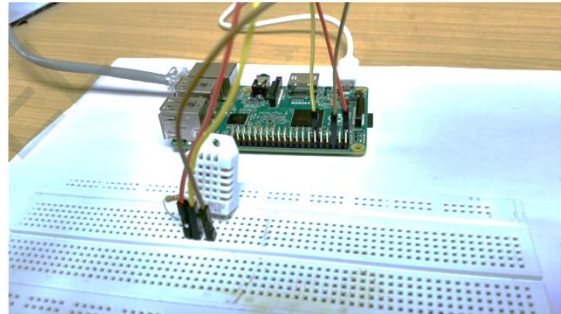
DHT Sensor

- Digital Humidity and Temperature Sensor (DHT)
- PIN 1, 2, 3, 4 (from left to right)
 - PIN 1- 3.3V-5V Power supply
 - PIN 2- Data
 - PIN 3- Null
 - PIN 4- Ground



Sensor- Raspberry Pi Interface

- Connect pin 1 of DHT sensor to the 3.3V pin of Raspberry Pi
- Connect pin 2 of DHT sensor to any input pins of Raspberry Pi, here we have used pin 11
- Connect pin 4 of DHT sensor to the ground pin of the Raspberry Pi



Read Data from the Sensor

Use the Adafruit library for DHT22 sensor to read the sensor data

```
GNU nano 2.2.6 File: IOTSR.py
import RPi.GPIO as GPIO
from time import sleep

import Adafruit_DHT

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

sensor = Adafruit_DHT.DHT22 # create an instance of the sensor type

print ('Getting data from the sensor')

# Humidity and temperature are 2 variables that store the values received from the sensor
humidity, temperature = Adafruit_DHT.read_retry(sensor, 17)

print ('Temp={0:0.1f}*C humidity={1:0.1f}%'.format(temperature, humidity))
```

```
pi@raspberrypi:~ $ python IOTSR.py
Getting data from the sensor
Temp=26.1*C humidity=65.9%
pi@raspberrypi:~ $
```

Sending Data to a Server

- Sending data to server using socket programming
 - Create a client and server
 - Establish connection between the two
 - Send data from the client to the server
 - Save the data in a file

Data Processing

Data from the client needs to be processed before it can be used further

- Data splitting/filtering
- Data plotting

Data Processing

✓ Data splitting/filtering:

- Data from the client is saved in a text file
- The values are separated by a comma(' , ')

`message = str(h)+' '+str(t)`

- Split() function can be used to split a string into multiple strings depending on the type of separator/delimiter specified.

Example:

`Data= 'sunday,monday,tuesday'`

`#Data is a string with 3 words separated by a comma`

`Data.split(",")`

`# split the data whenever a "," is found`

`['sunday','monday','tuesday']`

`# Gives 3 different strings as output`

Data Processing

Plotting the data:

- MATPLOTLIB is a python library used to plot in 2D *graph*
 - Plot(x,y): plots the values x and y
 - xlabel('X Axis'): Labels the x-axis
 - ylabel('Y Axis'): Labels the y-axis
 - title("Simple Plot"): Adds title to the plot

Data Processing (contd..)

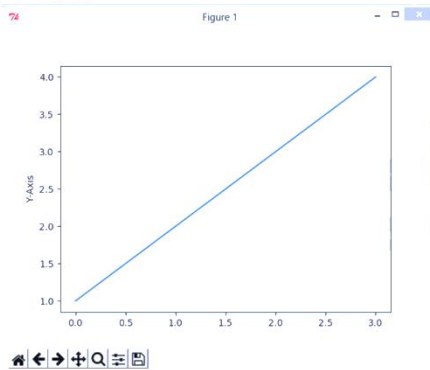
✓ Plotting the data:

```
import matplotlib.pyplot as myplot myplot  
myplot.plot([1,2,3,4])  
myplot.ylabel('Y-Axis')  
myplot.show()
```

By default the values are taken for y-axis, values for x-axis are generated automatically starting from 0

Data Processing (contd..)

Basic Plot:



```
import matplotlib.pyplot as myplot
myplot.plot([1,2,3,4])
myplot.ylabel("Y-Axis")
myplot.show()
```

Data Processing (contd..)

Some other common functions used in plotting:

- `figure()`: Creates a new figure
- `grid()`: Enable or disable axis grids in the plot
- `ion()`: turns on the interactive mode
- `subplot()`: Adds subplot in a figure
- `Close()`: Close the current figure window
- `Scatter()`: make a scatter plot of the given points



Sending Data to a Server (contd..)

Client:

```
def sensordata():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)
    sensor = Adafruit_DHT.AM2302
    humidity, temperature =
    Adafruit_DHT.read_retry(sensor,17)
    return(humidity, temperature)
```

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM) #create UDP socket
server_address = ('10.14.3.194', 10001)
try:
    while (1):
        h,t = sensordata()
        message = str(h)+'_'+str(t) #Send data
        print >>sys.stderr, 'sending "%s"' % message
        sent = sock.sendto(message, server_address)
finally:
    print >>sys.stderr, 'closing socket'
    sock.close()
```


Sending Data to a Server (contd..)

Server:

```
def coverage_plot(data,i):  
    hum=data.split(",")[0] temp = 18  
    tem=data.split(",")[1] iden = 100  
    print 'temp='+str(tem)+'iter='+str(i)  
    plt.ion()  
    fig=plt.figure(num=1,figsize=(6,6))  
    plt.title(' IoT Temperature and Humidity Monitor')  
    ax = fig.add_subplot(121)  
    ax.plot(tem,i, c='r', marker=r'$\Theta$')  
    plt.xlabel('Temp ($^{\circ}$C$)')  
  
    ax.grid()  
    ax = fig.add_subplot(122)  
    ax.plot(hum,i, c='b', marker=r'$\Phi$')  
    plt.xlabel('Humidity ($\%$)')  
    ax.grid()  
    fig.show()  
    fig.canvas.draw()  
  
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
  
# Bind the socket to the port  
server_address = ('10.14.3.194', 10001)  
sock.bind(server_address)
```

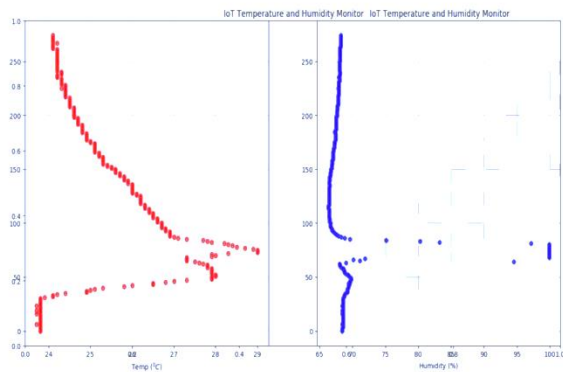
Sending Data to a Server (contd..)

Server:

```
i=0  
while True:  
    data, address = sock.recvfrom(4096)  
    with open("DataLog.txt","a") as f:  
        mess=str(data)  
        f.write(mess)  
        coverage_plot(mess,i)  
        print mess  
        i+=1  
    f.close()
```

Output

- The Reading from the sensor is sent to the Server and saved in a text file.
- Two different plots for temperature and humidity data

[illegible]

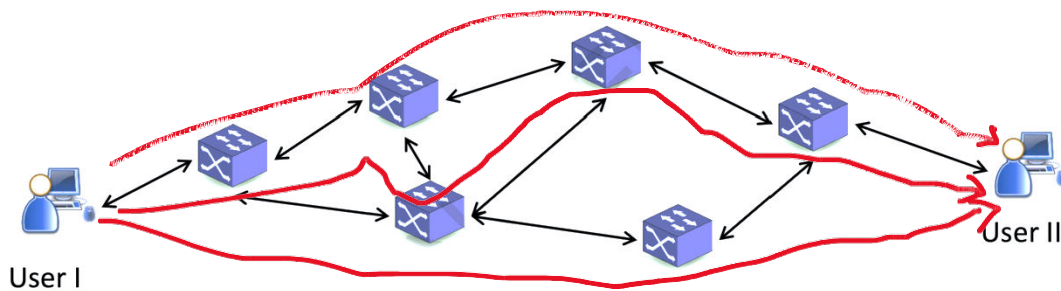
```
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.39999996185
68.9000015259,23.39999996185
68.9000015259,23.5
68.9000015259,23.5
68.8000030518,23.39999996185
68.9000015259,23.5
68.9000015259,23.5
68.8000030518,23.39999996185
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.39999996185
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
68.9000015259,23.5
```

Chapter 8.

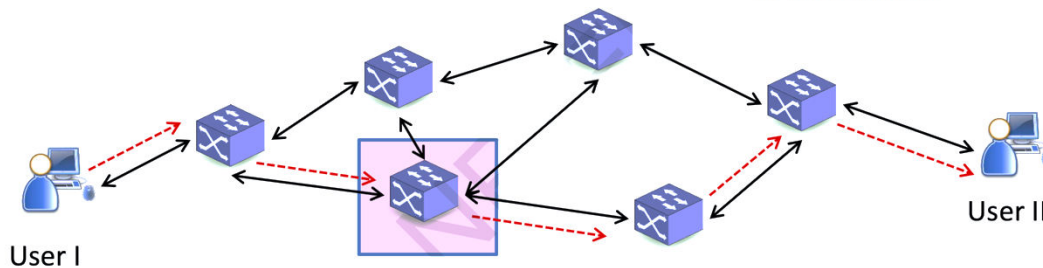
Software defined Networking

- 8.1 Limitation of current network
- 8.2 Origin of SDN
- 8.3 SDN Architecture
- 8.4 Rule Placement, Open flow Protocol
- 8.5 Controller placement
- 8.6 Security in SDN
- 8.7 Integrating SDN in IoT

Overview of Current Network

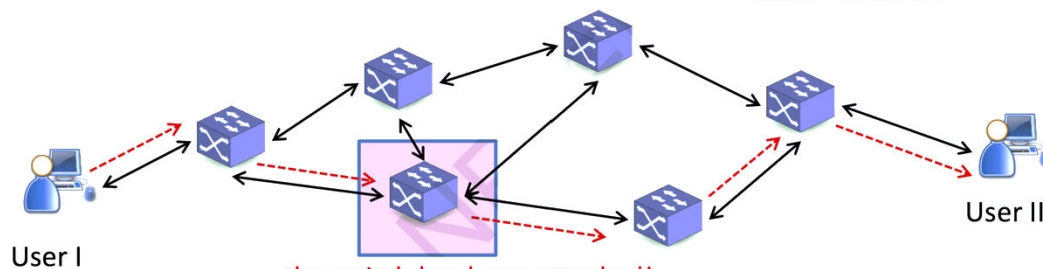


Open shortest Path First
OSPF Protocol executing at the switches



the switch has been attacked!

OSPF Protocol executes at the switches

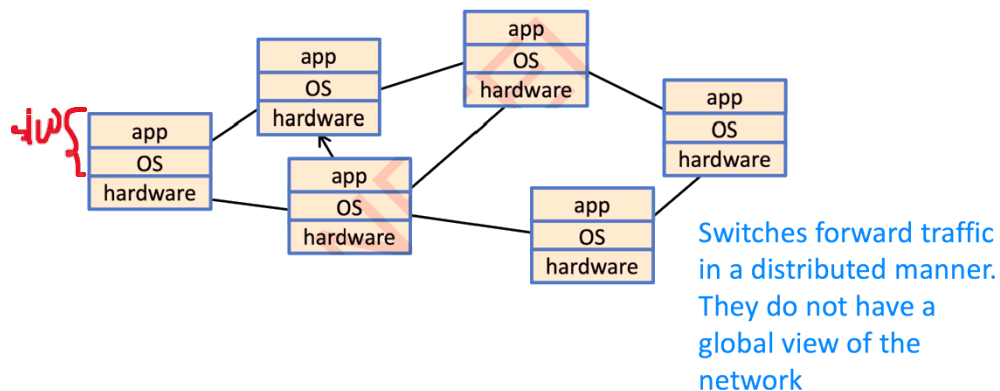


the switch has been attacked!

needs to route through an alternate path!

Present: No centralized control.

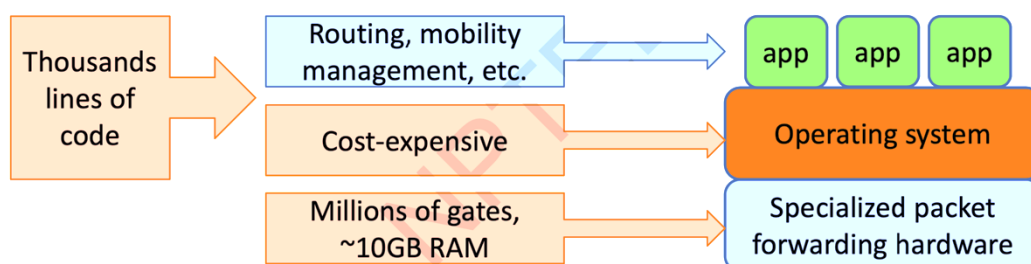
Limitations in Current Network



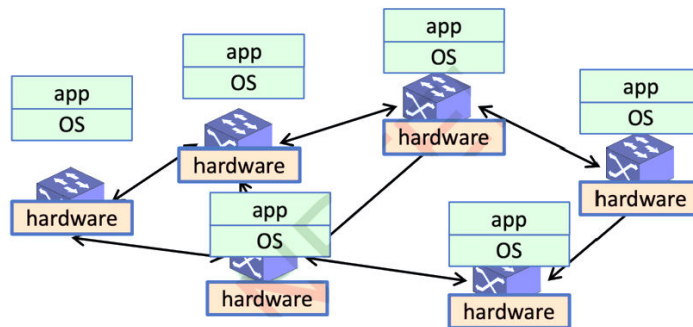
Limitations in Current Network

- ✓ Vendor-specific architecture of switches limits dynamic configuration according to application-specific requirements.
- ✓ Switches are required to configure according to the installed operating system (OS).
- ✓ Centralized control is not feasible in traditional network.

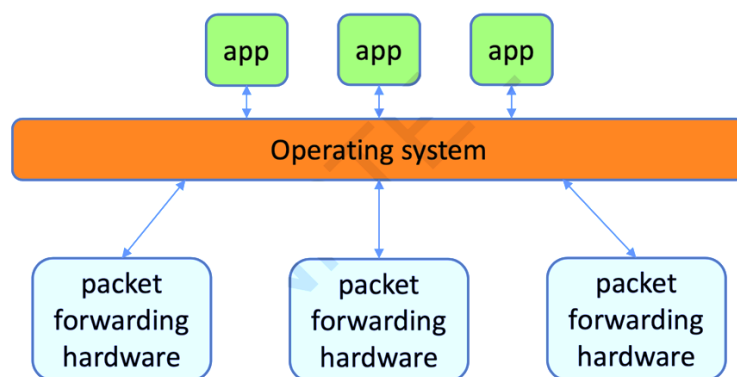
Limitations in Current Network



Current Network to SDN



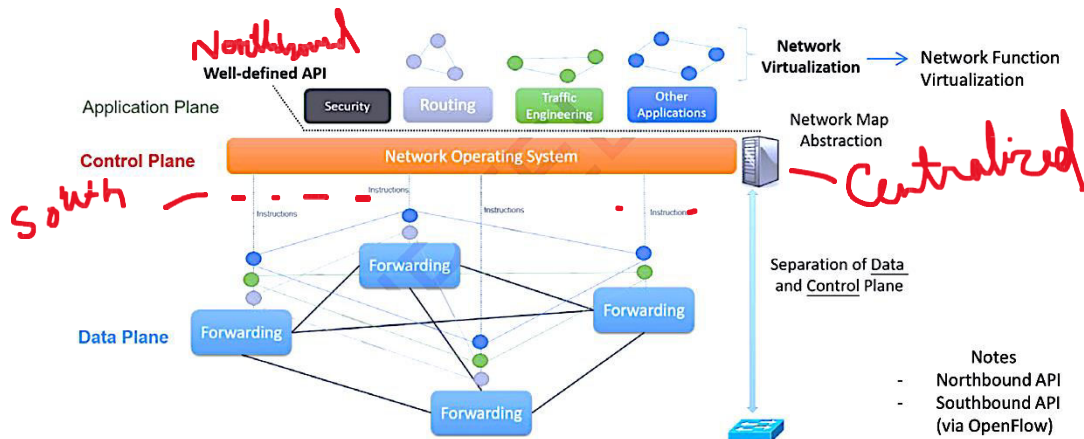
Current Network to SDN



Origin of SDN

- ✓ 2006: At Stanford university, a team proposes a clean-slate security architecture (SANE) to control security policies in a centralized manner instead of doing it at edges.
- ✓ 2008: The idea of *software-defined network* is originated from OpenFlow project (ACM SIGCOMM 2008).
- ✓ 2009: Stanford publishes OpenFlow V1.0.0 specs.
- ✓ June 2009: Nicira network is founded.
- ✓ March 2011: Open Networking Foundation is formed.
- ✓ Oct 2011: First Open Networking Summit. Many Industries (Juniper, Cisco announced to incorporate.

SDN Architecture



Basic Concepts of SDN

- ✓ Separate control logic from hardware switches
- ✓ Define the control logic in a centralized manner
- ✓ Control the entire network including individual switches
- ✓ Communication between the application, control, and data planes are done through APIs

Components/Attributes of SDN

- ✓ Hardware switches
- ✓ Controller
- ✓ Applications
- ✓ Flow-Rules
- ✓ Application programming interfaces (APIs)

Current Status of SDN

- ✓ Companies such as Google have started to implement SDN at their datacenter networks.
- ✓ It is required to change the current network with SDN in a phased manner.
- ✓ Operational cost and delay caused due to link failure can be significantly minimized.

Challenges

- ✓ Rule placement
- ✓ Controller placement

Rule Placement

- ✓ Switches forward traffic based on a rule – ‘Flow-Rule’ – defined by the centralized controller.
 - Traditionally, **Routing Table** in every switch (L3 switch/router). SDN maintains **Flow Table** at every switch.
 - Flow-Rule: Every entry in the Flow Table.
- ✓ Each rule has a specific format, which is also defined by a protocol (e.g., OpenFlow).

OpenFlow

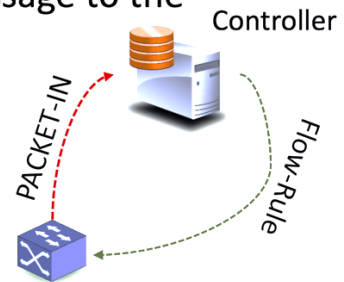
Match SDN Applications First and Use Normal For Unmatched Packets (Hybrid Default Forwarding)										
Priority	Ingress Port	MAC Source Address	MAC Destination	Protocol	Vlan ID	IP Source Address	IP Destination	Source Port	Destination Port	Instructions
10000	*	*	*	TCP	*	*	10.1.1.20/32	*	80	Forward to Port 1
5000	*	*	*	*	*	*	10.1.1.0/24	*	*	Forward to Port 2
300	*	*	*	*	2500	*	*	*	*	Send to Controller
0	*	*	*	*	*	*	*	*	*	OF Normal

Example of a flow-rule based on OpenFlow protocol

Rule Placement Challenges |

- ✓ Size of ternary content-addressable memory (TCAM) is limited at the switches.
 - Limited number of rules can be inserted.
- ✓ Fast processing is done using TCAM at the switches.
- ✓ TCAM is very cost-expensive.

- ✓ On receiving a request, for which no flow-rule is present in the switch, the switch sends a *PACKET-IN* message to the controller.
- ✓ The controller decides a suitable flow-rule for the request.
- ✓ The flow-rule is inserted at the switch.
- ✓ Typically, **3-5ms delay** is involved in a new rule placement



- ✓ How to define/place the rules at switches, while considering available TCAM.
- ✓ How to define rules, so that less number of *PACKET-IN* messages are sent to controller.

OpenFlow Protocol I

- ✓ Only one protocol is available for rule placement – OpenFlow.
- ✓ It has different versions – 1.0, 1.1, 1.2, 1.3, etc. – to have different number of match-fields.

Priority	Ingress Port	MAC Source Address	MAC Destination	Protocol	Vlan ID	IP Source Address	IP Destination	Source Port	Destination Port	Instructions
10000	*	*	*	TCP	*	*	10.1.1.20/32	*	80	Forward to Port 1
5000	*	*	*	*	*	*	10.1.1.0/24	*	*	Forward to Port 2
300	*	*	*	*	2500	*	*	*	*	Send to Controller
0	*	*	*	*	*	*	*	*	*	OF Normal

✓ Different match-fields

- Source IP
- Destination IP
- Source Port
- Priority
- etc.

How much time a flow-rule is to be kept at the switch?

✓ Hard timeout

- All rules are deleted from the switch at hard timeout.
- This can be used to reset the switch.

✓ Soft timeout

- If NO flow is received associated with a rule for a particular time, the rule is deleted.
- This is used to empty the rule-space by deleting an unused rule.

✓ SDN is NOT OpenFlow

- SDN is a technology/concept
- OpenFlow is a protocol used to communicate between data-plane and control-plane.
- We may have other protocols for this purpose. However, OpenFlow is the only protocol present today.

OpenFlow Switch Software

- ✓ **Indigo**: Open source, it runs on Mac OS X.
- ✓ **LINC**: Open source, it runs on Linux, Solaris, Windows, MacOS, and FreeBSD.
- ✓ **Pantou**: Turns a commercial wireless router/access point to an OpenFlow enabled switch. OpenFlow runs on OpenWRT.
- ✓ **Of13softswitch**: User-space software switch based on Ericsson TrafficLab 1.1 softswitch.
- ✓ **Open vSwitch**: Open Source, it is the MOST popular one present today.

Summary

- ✓ Basics of SDN
- ✓ Challenges present in SDN
- ✓ Rule Placement with OpenFlow
- ✓ **Controller Placement – to be discussed in next lecture**

SDN - Recap

- ✓ SDN – restructuring current network infrastructure
- ✓ Architecture of SDN – Application, Control and Infrastructure layers
- ✓ Rule Placement – TCAM and Delay
- ✓ OpenFlow protocol – flow-rule and math-fields

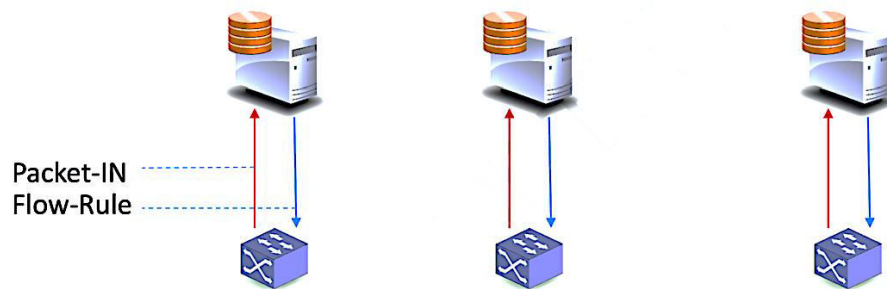
APIs in SDN

- ✓ Southbound API
 - Used to communicate between control layer and infrastructure layer.
 - OpenFlow protocol is used.
- ✓ Northbound API
 - Used to communicate between control layer and application layer.
 - Standard APIs are used.
- ✓ East-Westbound APIs
 - Used to communicate among **multiple controllers** in the control layer.

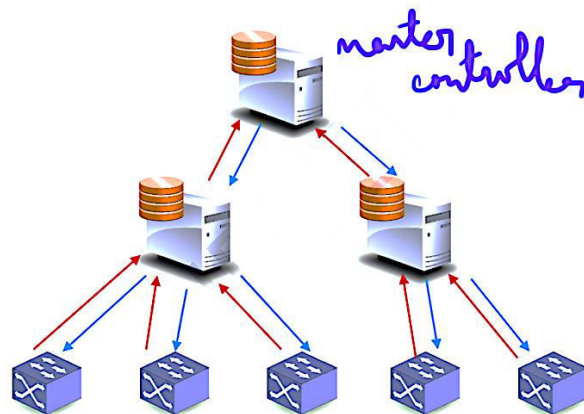
Controller Placement I

- ✓ Controllers define flow-rule according to the application-specific requirements.
 - ✓ The controllers must be able to handle all incoming requests from switches.
 - ✓ Rule should be placed without incurring much delay.
 - ✓ Typically, a controller can handle 200 requests in a second (through a single thread).
-
- ✓ The controllers are logically connected to the switches in one-hop distance.
 - Physically, they are connected to the switches in multi-hop distance.
 - ✓ If we have a very small number of controllers for a large network, the network might be congested with control packets (i.e., PACKET-IN messages).

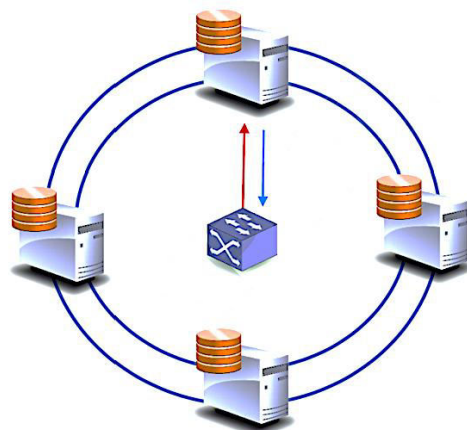
Flat Architecture



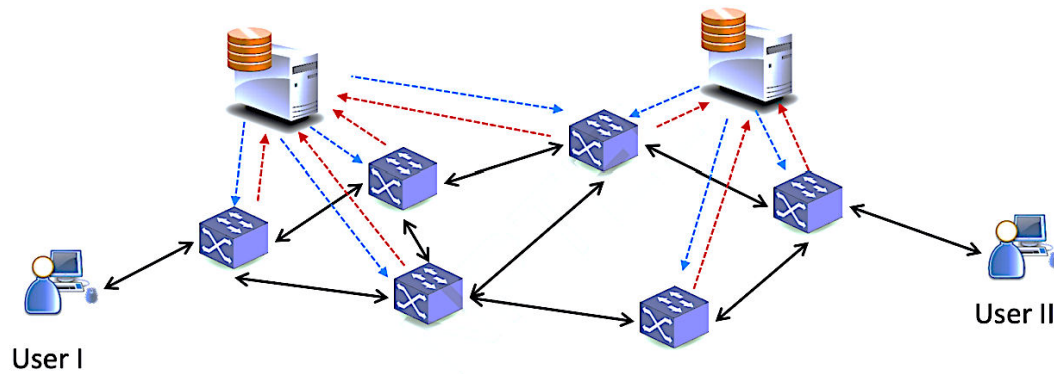
Hierarchical (tree) Architecture



Ring Architecture



Mesh Architecture



Control Mechanisms

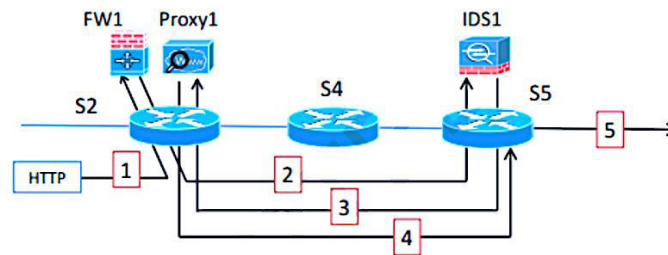
- ✓ Distributed
 - The control decisions can be taken in a distributed manner
 - Ex: each subnetwork is controlled by different controller
- ✓ Centralized
 - The control decisions are taken in a centralized manner.
 - Ex: A network is controlled by a single controller.

Backup Controller

- ✓ If a controller is down, what will happen?
 - Backup controller is introduced
 - Replica of the main controller is created
 - If the main controller is down, backup controller controls the network to have uninterrupted network management.

Security I

- ✓ Enhanced security using SDN
 - Firewall
 - Proxy
 - HTTP
 - Intrusion detection system (IDS)



Example of potential data plane ambiguity to implement the policy chain Firewall-IDS-Proxy in the example topology.

Experimenting with SDN

✓ Simulator/Emulator

- Infrastructure deployment – MUST be supported with OpenFlow
- Controller placement – MUST support OpenFlow
 - Remote – controller can be situated in a remote place, and communicated using IP address and port number
 - Local

Switch Deployment

✓ Mininet (s/w)

- Used to create a virtual network with OpenFlow-enabled switches
- Based on Python language
- Supports remote and local controllers

Controller Configuration Software

- ✓ Pox
- ✓ Nox
- ✓ FloodLight
- ✓ OpenDayLight [Popular!]
- ✓ ONOS [Popular!]

Summary

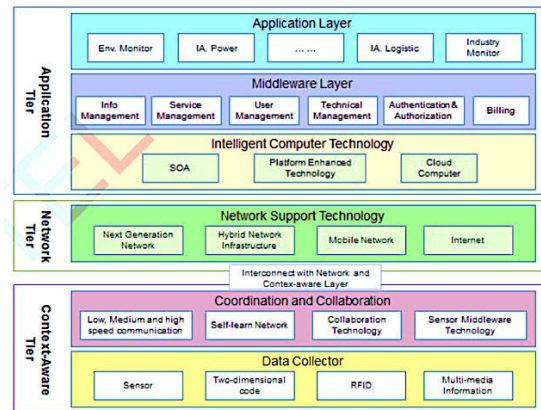
- ✓ Performance of SDN depends on rule placement and controller placement in the network.
- ✓ Control message overhead may be increased due to additional number of packets (PACKET-IN messages).
- ✓ Unified network management is possible using SDN, while leveraging global view of the network.

IoT Architecture

IoT World Forum Reference Model



Source: <https://image.slidesharecdn.com>

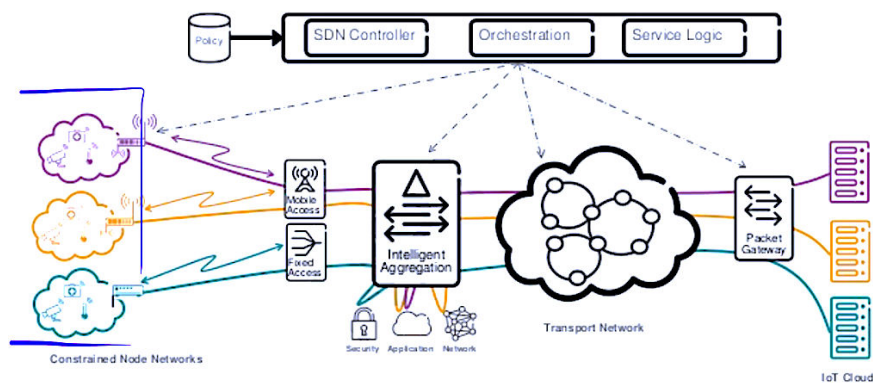


Source: <http://www.luxhotels.info/p/46800/internet-of-things-iot/>

Benefits of Integrating SDN in IoT

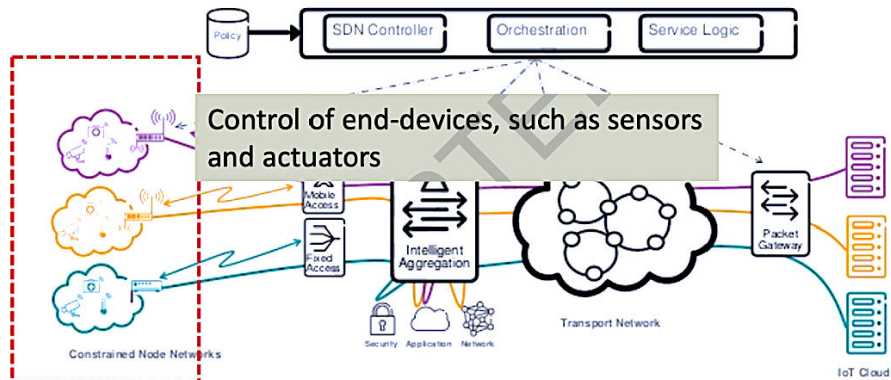
- ✓ Intelligent routing decisions can be deployed using SDN
- ✓ Simplification of information collection, analysis and decision making
- ✓ Visibility of network resources – network management is simplified based on user, device and application-specific requirements
- ✓ Intelligent traffic pattern analysis and coordinated decisions

SDN for IoT I

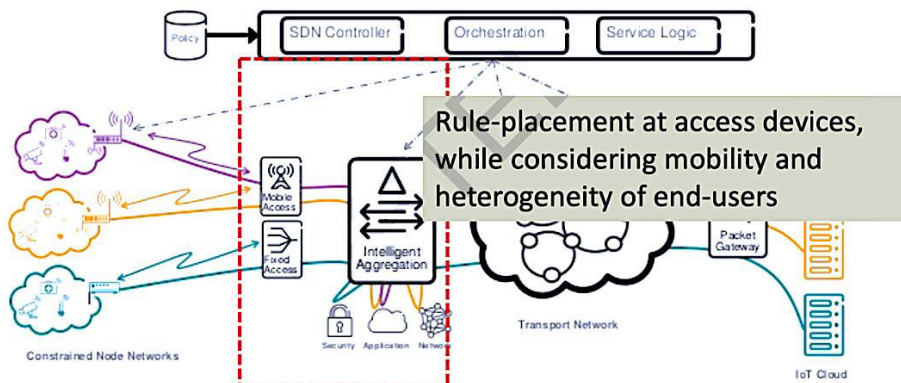


Source: <https://image.slidesharecdn.com>

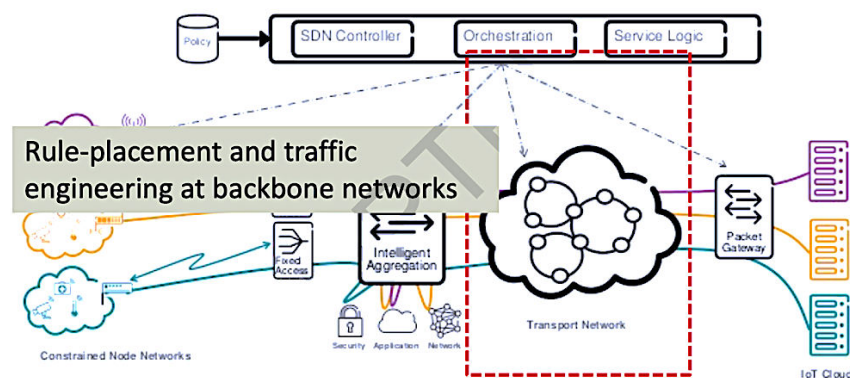
SDN for IoT II



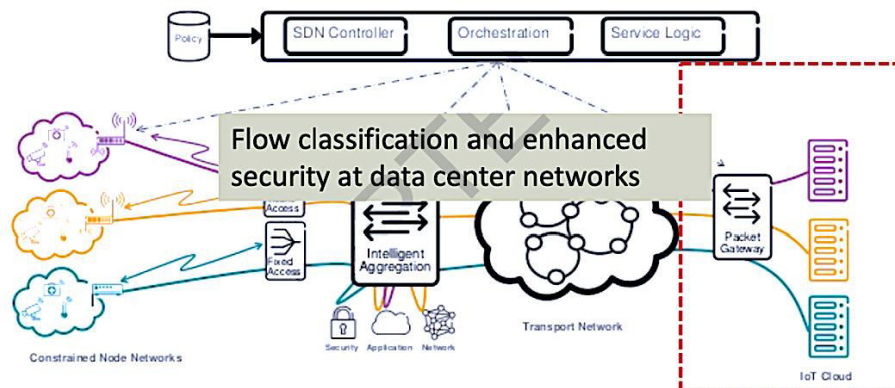
SDN for IoT III



SDN for IoT IV



SDN for IoT V



Wireless Sensor Network

✓ Challenges

- Real-time programming of sensor nodes
- Vendor-specific architecture
- Resource constrained – heavy computation cannot be performed
- Limited memory – cannot insert too many control programs

✓ Opportunities

- Can we program the sensor nodes in real-time?
- Can we change the forwarding path in real-time?
- Can we integrate different sensor nodes in a WSN?

Software-Defined WSN I

✓ Sensor OpenFlow (Luo et al., IEEE Comm. Letters '12)

- Value-centric data forwarding
 - Forward the sensed data if exceeds a certain value
- ID-centric data forwarding
 - Forward the sensed data based on the ID of the source node

Real-life implementation of such method NOT done

✓ Soft-WSN (Bera et al., IEEE SJ '16)

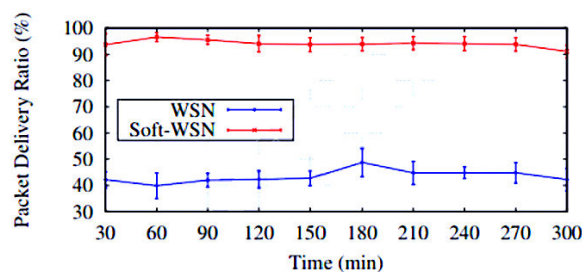
- Sensor Device Management
 - Sensor management
 - Multiple sensors can be implemented in a single sensor board
 - Sensors can be used depending on application-specific requirements
 - Delay management
 - Delay for sensing can be changed dynamically in real-time
 - Active-Sleep Management
 - States of active and sleep mode can be changed dynamically

✓ Soft-WSN

- Topology Management
 - Node-specific management – forwarding logic of a particular sensor can be modified
 - Network-specific management
 - Forward all traffic of a node in the network
 - Drop all traffic of a node in the network

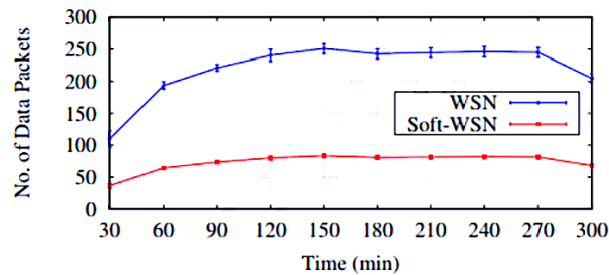
Experimental results show that network performance can be improved using software-defined WSN over traditional WSN

Soft-WSN: Result I



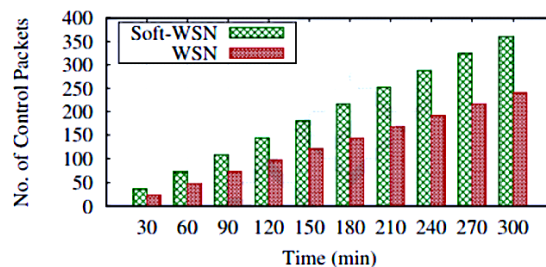
Packet delivery ratio in the network increases using Soft-WSN compared to the traditional WSN.

Soft-WSN: Result II



Number of replicated data packets is reduced using Soft-WSN over the traditional WSN.

Soft-WSN: Result III



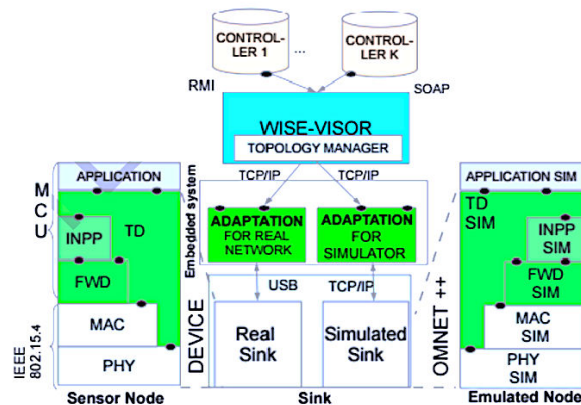
Number of control messages in the network is higher using Soft-WSN over the traditional WSN. This is due to the PACKET-IN message in the network. Each time a node receives a new packet, it asks the controller for getting adequate forwarding logic.

Software-Defined WSN III

- ✓ SDN-WISE (Galluccio et al., IEEE INFOCOM '15)
 - A software-defined WSN platform is designed
 - Flow-table for rule placement at sensor nodes is designed
 - Any programming language can be used through API to program the nodes in real-time

SDN-WISE Protocol Stack

- ✓ Sensor node includes
 - IEEE 802.15.4 protocol
 - Micro control unit (MCU)
 - Above IEEE 802.15.4 stack, *Forwarding layer* consists of *Flow-rules*.
 - *INPP* – In Network Packet Processing



Summary

- ✓ SDN is useful to manage and control IoT network
- ✓ Wireless sensor nodes and network can be controlled using SDN-based applications
- ✓ Network performance can be improved significantly using SDN-based approaches over the traditional approaches

SDN for IoT - Recap

- ✓ Software-Defined WSN
- ✓ Different approaches – Sensor OpenFlow, Soft-WSN, SDN-WISE
- ✓ Software-Defined WSN can improve overall performance over traditional WSN

Traditional (Wireless) Mobile Network

- ✓ Problems in Traditional Mobile Network
 - Difficult to Scale – static over-provisioned network are inflexible to manage the mobile traffic with high demand
 - Difficult to manage – many times lead to misconfigurations
 - Inflexible – Requires too much time to introduce a new service as the hardware architecture is inflexible
 - Cost-expensive – Both capital expenditure and operational expenditure are high

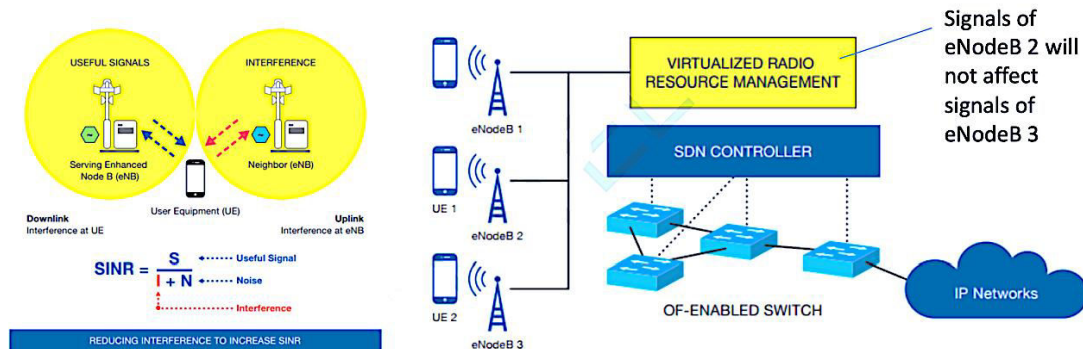
*Based on information from Open Networking Foundation (ONF)

SDN for Mobile Networking

- ✓ Flow-Table Paradigm of SDN
 - Well suited for end-to-end communication over multiple technologies such as WiFi, 3G, 4G, etc.
- ✓ Logically Centralized Control
 - Particularly useful for efficient base-station coordination for addressing inter-cell interference
- ✓ Path Management
 - Data can be routed based on service requirements without depending on core routing policies
- ✓ Network Virtualization
 - Abstracts the physical resources from the network services
 - Helps in providing seamless connectivity and service differentiation among users

*Based on information from Open Networking Foundation (ONF)

SDWMN-Use Case: Interference Management

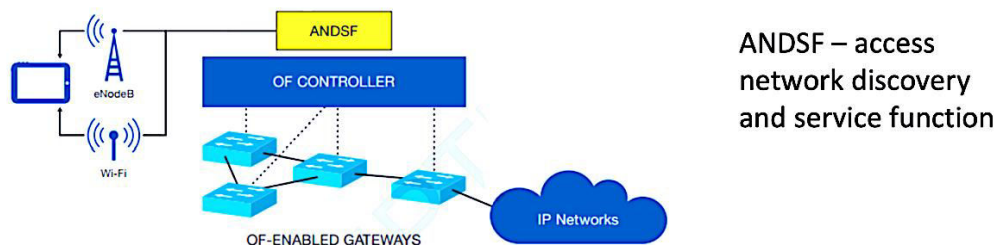


Traditional Mobile Network

Software-Defined Mobile Network

*Based on information from Open Networking Foundation (ONF)

SDWMN-Use Case: Mobile Traffic Management



Mobile traffic offloading based on OpenFlow

*Based on information from Open Networking Foundation (ONF)

Key Benefits

- ✓ Centralized control of devices manufactured by multiple vendors
- ✓ Higher rate of integration of new services
- ✓ Abstracted network control and management
 - Network abstracted from the user

Rule Placement at Access Devices

✓ Challenges

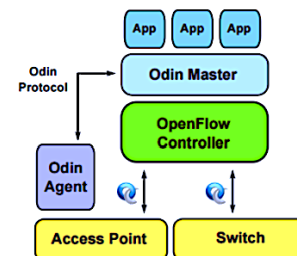
- General OpenFlow does not support wireless network
 - Modified version of OpenFlow is required
- Typically, users are mobile in nature – network is highly dynamic
- Frequent changes in rule placement is also required
- Presence of heterogeneous devices in the network
 - How to support such heterogeneous devices in a single platform

Approaches

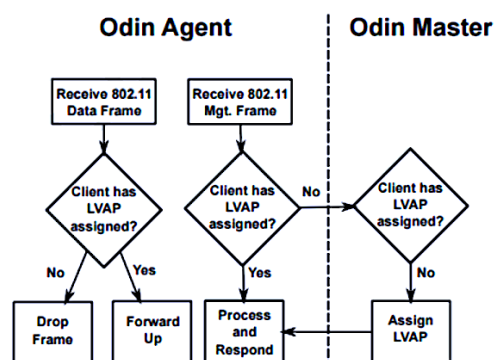
- ✓ ODIN
- ✓ Ubi-Flow
- ✓ Mobi-Flow

ODIN I

- ✓ An agent is placed at access points to communicate with controller
- ✓ Two components are present
 - Odin agent – placed with the physical devices
 - Odin master – placed at the controller end



- ✓ Conversion of 802.11
- ✓ LVAP – Light virtual AP



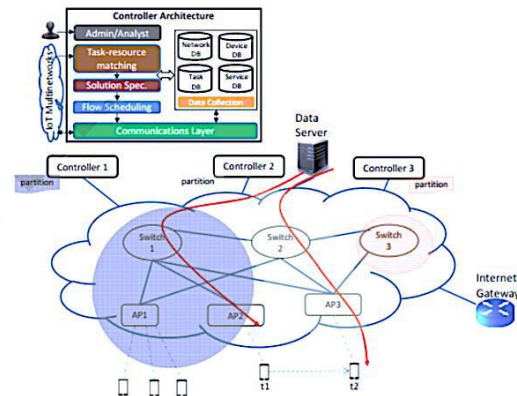
Ubi-Flow I

✓ Mobility management in SDIoT

- Scalable control of the APs
- Fault tolerance

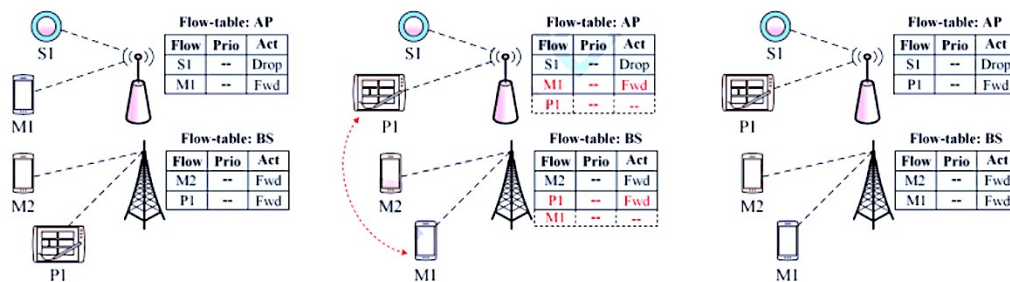
✓ Flow-Scheduling

- Network partition
- Network matching
- Load balancing



Mobi-Flow

✓ Mobility-aware flow-rule placement in SDIoT



✓ Proactive rule placement depending on users' movement in the network

✓ Approach

- Predict location of end-users at $(t+1)$ time, while the users are at (t) time
- Place flow-rules at the APs which can be associated to the users based on their predicted locations

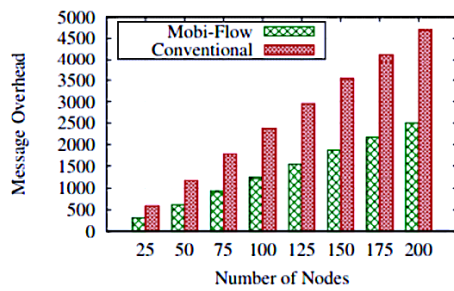
✓ Location prediction

- Order-K Markov predictor – takes last k-th location instances to predict next location

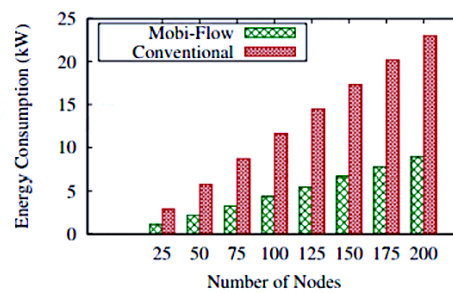
✓ Flow-rule placement

- Linear programming can be used to select optimal AP

Mobi-Flow IV



Message Overhead in the Network



Energy consumption in the Network

Control message overhead and energy consumption can be minimized significantly using Mobi-Flow compared to the conventional flow-rule placement schemes.

Rule Placement at Backbone Network

- ✓ Existing rule placement schemes for wired network can be used
- ✓ Load balancing is an important issue due to the dynamic nature of the IoT network
- ✓ Dynamic resource allocation can also be integrated

Data Center Networking

- ✓ Mice-Flow – Wildcard rules can be placed to deal with mice-flows
- ✓ Elephant Flow – Exact match rules are useful
- ✓ We need to classify the flows before inserting flow-rules at the switches to adequately forward them in the network

Anomaly Detection in IoT Network

- ✓ Monitor the network through OpenFlow to detect any anomaly in the network
 - This can be done by monitoring each flow in the network
 - We can also collect the port statistics of the switches
 - If there is any anomaly, it may generate large number of packets in the network – it can be detected by monitoring the flows

Experimenting with Wireless Network

- ✓ Mininet-WiFi
 - Can be used to deploy a network
 - Supports both wired and wireless network
 - Wired – Ethernet protocol
 - Wireless – WiFi protocol (IEEE 802.11 group)
- ✓ ONOS
 - Can be used to place the controllers

Summary

- ✓ SDN-based solution approaches are useful to deal with different challenges present in an IoT network
- ✓ Issues with interference and mobility can be managed using software-defined approaches

CHAPTER 9

SMART HOMES

9.1 Origin and example of Smart Home Technologies

9.2 Smart Home Implementation

9.3 Home Area Networks (HAN)

9.4 Smart Home benefits and issues

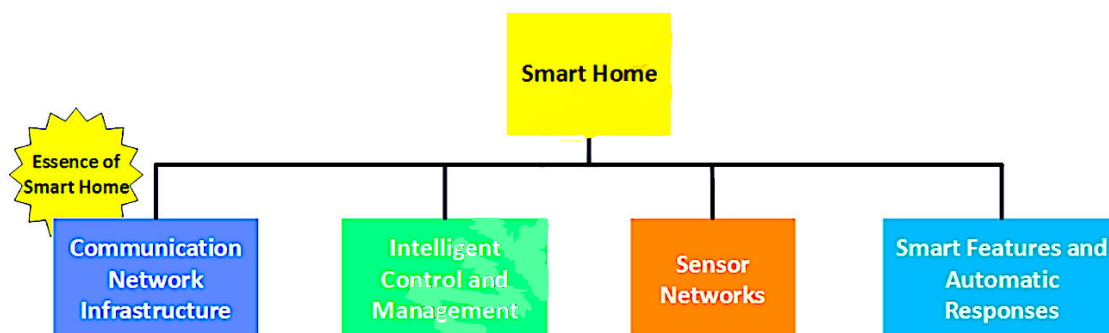
Introduction

- ✓ Smart home infrastructure consists of:
 - Intelligent networking device infrastructure
 - Seamless integration of various devices using wired/wireless technologies
- ✓ Allows ease of use for household systems.
- ✓ Creates a highly personalized and safe home space
- ✓ Corporations seriously indulging in smart home systems include GE, Cisco, Google, Microsoft, and others.

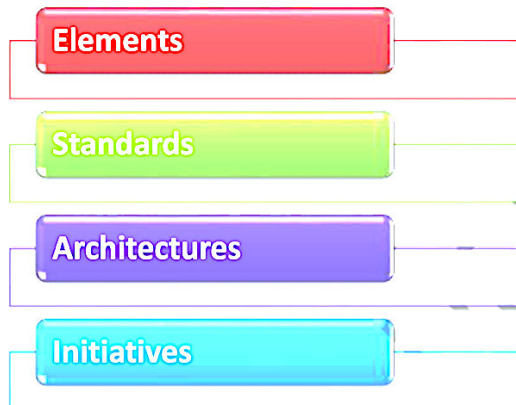
Smart Home

- ✓ Provides productive and cost-efficient environment.
- ✓ Maximizes the effectiveness of the occupants.
- ✓ Provides efficient management with minimum life-time costs of hardware and facilities.
- ✓ Optimizes-
 - Structures
 - Systems
 - Services and management
 - Interrelationships between the above three

Smart Home Aspects



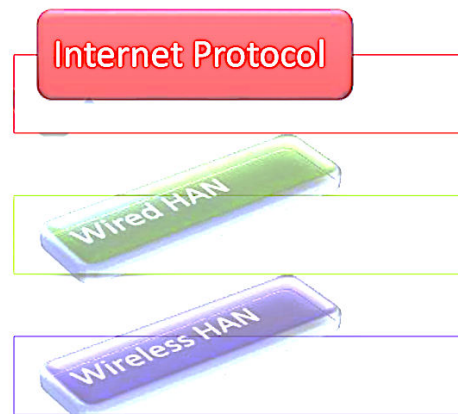
Home Area Networks (HANs)



- ✓ Network contained within a home.
- ✓ Enables remote access and control of devices and systems.
- ✓ Provides amalgamation of various systems within a home, such as – security systems, home automation systems, personal media, communication, etc.

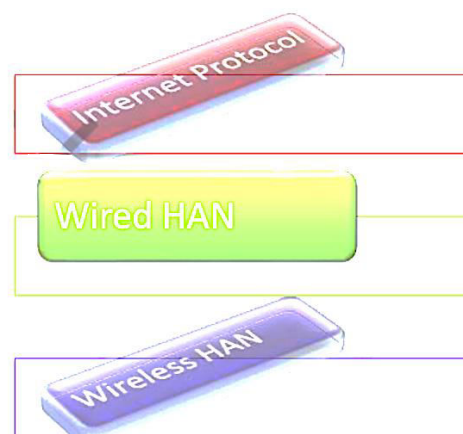
HAN Elements

- ✓ Internet Protocol (IP)
 - Multi-protocol gateway bridges non-IP network to IP network.
 - Bridging between new technologies is limited.
 - For new technologies or networks, a new mapping is required for bridging to perform satisfactorily.



HAN Elements

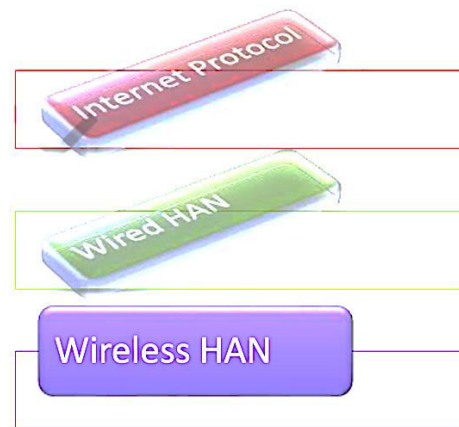
- ✓ Wired HAN
 - Easy integration with pre-existing house infrastructure.
 - Low cost.
 - Can use power lines, coaxial cables, telephone lines, optical fibers, and other such technologies for communication.



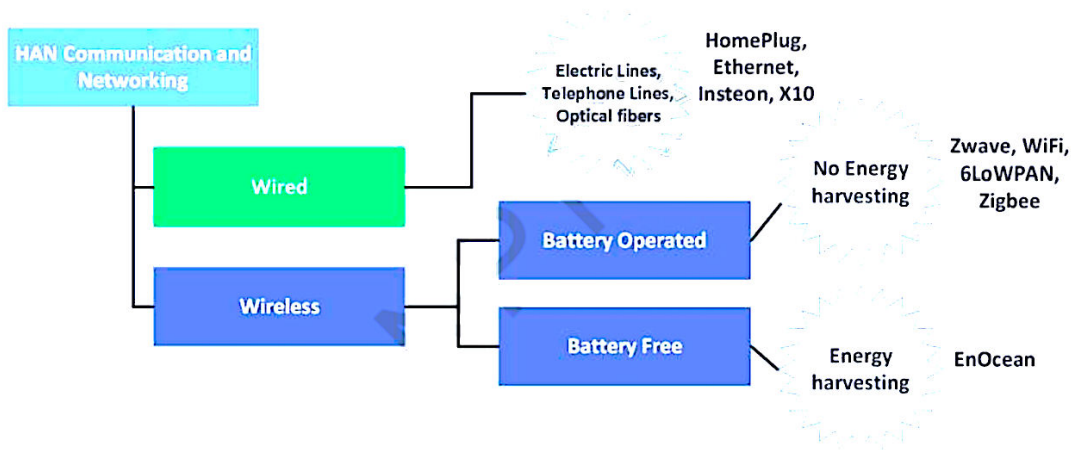
HAN Elements

✓ Wireless HAN

- Can use popular home Wi-Fi, ZigBee, and even new standards, such as 6LoWPAN.
- Wireless makes implementation easy.



HAN Medium Classification



HAN Standards

- ✓ Universal Plug and Play (UPnP).
- ✓ Application layer technology, mainly web-based.
- ✓ TCP/IP protocol stack provides support for the lower layers, and enables seamless integration of various technologies.
- ✓ Provides transparent networking with support for zero-configuration networking and automatic discovery of devices.



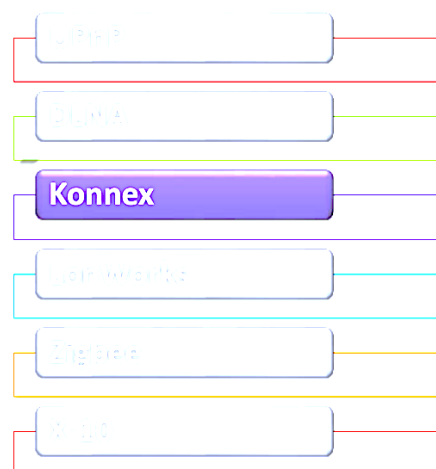
HAN Standards

- ✓ Digital Living Network Alliance (DLNA)
- ✓ Trade organization created by Sony, Intel, and Microsoft.
- ✓ Connects cable-based networks with wireless networks for increased sharing of media, control and access.
- ✓ Domestically shares network media resources.



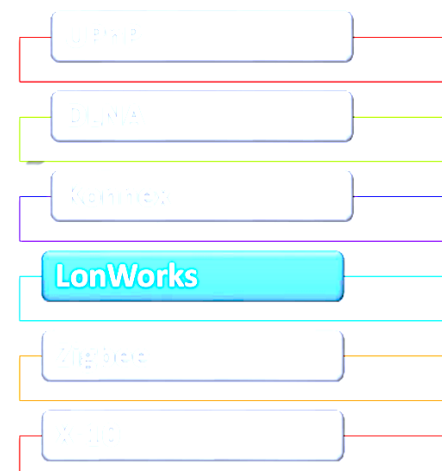
HAN Standards

- ✓ Konnex (KNX): an open important standard for home and building networks.
- ✓ Utilizes the full range of home communication infrastructure – Power lines, coaxial cables, twisted pair, RF, etc.
- ✓ Must be setup and configured via a software before its proper usage.



HAN Standards

- ✓ Local Operation Networks (LonWorks).
- ✓ Every device includes a Neuron Chip, a transceiver and the application electronics. *on Chip*
- ✓ Neuron chip is a SOC with multiple microprocessors, RAM, ROM and IO interface ports.
- ✓ Splits device groups into intelligent elements, which can communicate through a physical communication medium.



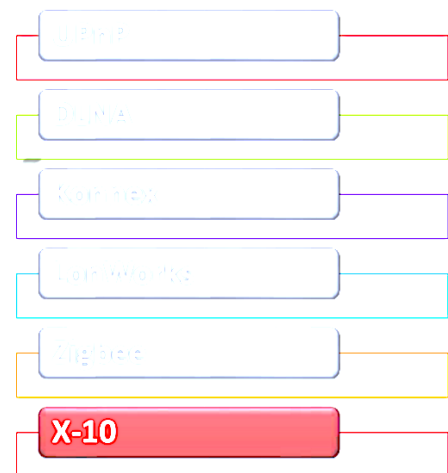
HAN Standards

- ✓ Zigbee consists of four layers – Physical, Medium Access Control, Network, and Application.
- ✓ Physical and MAC layers are defined by IEEE802.15.4, whereas Network and Application are defined by Zigbee.
- ✓ Aims at low-cost, low-energy devices.
- ✓ ZigBee Alliance is composed of Mitsubishi, Honeywell, Invensys, Motorola and Philips



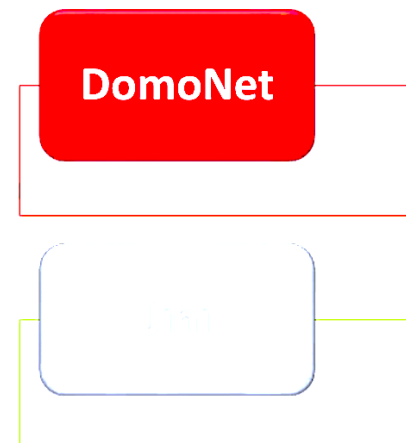
HAN Standards

- ✓ X-10 enables remote control of compliant transmitters and receivers over power lines and electrical wirings present in the house.
- ✓ Adopted by GE and Philips.
- ✓ Standard defines procedures for transmission of bits over AC carrier signals.
- ✓ Low-speed and low data rate.
- ✓ Mainly used for control of lighting, appliance networks and security sensors.



HAN Architectures

- ✓ Uses XML for description and web-services for control.
- ✓ Follows a Service oriented Architecture (SOA).
- ✓ Not tied to any software, language or architecture.
- ✓ A central gateway connects different technologies.
- ✓ A tech Manager for each technology provides web services for control and access.



HAN Architectures

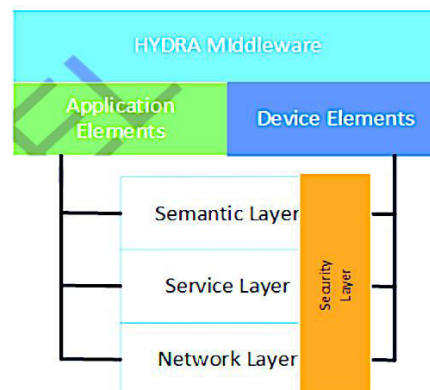
- ✓ Connects various devices sharing their resources with auto-configuration and auto-installation.
- ✓ Based on JAVA environment and pursued by Sun Microsystems (Now, Oracle).
- ✓ Constructs an organized distribution system without a central node (federation).
- ✓ Jini apps use bytecode to run JVM, and are portable.
- ✓ Follows Object Oriented Paradigm.



HAN Initiatives

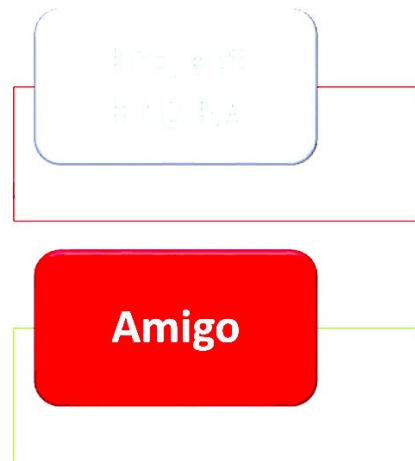
- ✓ Middleware for embedded intelligent systems.
- ✓ Connects a Service Oriented Architecture Network.
- ✓ Connected devices may have limited resources, low processing power, memory or energy consumption.
- ✓ Each device has an embedded HYDRA client which acts as a proxy between the device and the middleware.

HYDRA Protocol Stack



HAN Initiatives

- ✓ Aimed at:-
 - Ambient intelligent systems
 - For networked home systems
- ✓ Features user-friendly interfaces, interoperability, and automatic discovery of devices and services.



Chapter 10-Smart Cities

- 10.1 Characteristics of Smart Cities
- 10.2 Smart city Frameworks
- 10.3 Challenges in Smart cities
- 10.4 Data Fusion
- 10.5 Smart Parking
- 10.6 Energy Management in Smart cities

10.1 Characteristics of Smart Cities

Introduction

- ✓ A Smart City is-
 - An urban system
 - Uses Information & Communication Technology (ICT)
 - Makes infrastructure more interactive, accessible and efficient.
- ✓ Need for Smart Cities arose due to-
 - Rapidly growing urban population
 - Fast depleting natural resources
 - Changes in environment and climate

Analogy

Humans	Smart Cities
Skeleton	Buildings, Industries, People
Skin	Transportation, Logistics
Organs	Hospital, Police, Banks, Schools
Brain	Ubiquitously embedded intelligence
Nerves	Digital telecommunication networks
Sensory Organs	Sensors, Tags
Cognition	Software

Application Focus Areas



10.2 Smart city Frameworks

Smart Economy



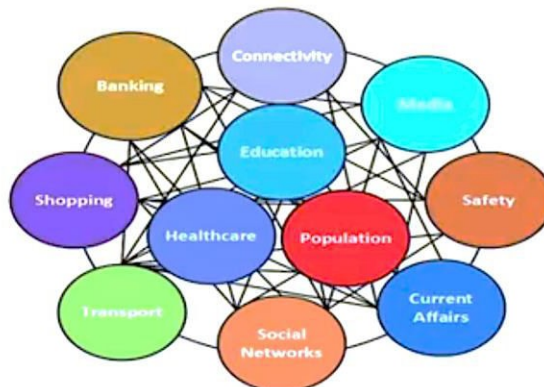
Smart Governance



Smart Environment



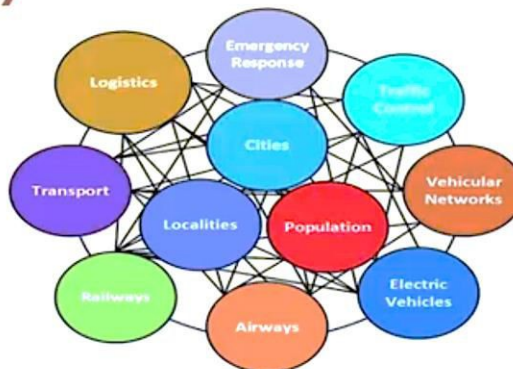
Smart People



Smart Living



Smart Mobility



Current Focus Areas

Smart Homes

- Health monitoring.
- Conservation of resources (e.g. electricity, water, fuel).
- Security and safety.

Smart Parking Lots

- Auto routing of vehicles to empty slots.
- Auto charging for services provided.
- Detection of vacant slots in the parking lot.

Smart Vehicles

- Assistance to drivers during bad weather or low-visibility.
- Detection of bad driving patterns or driving under the influence of substances.
- Auto alert generation during crashes.
- Self-diagnostics.

Smart Health

- Low cost, portable, at-home medical diagnosis kits.
- Remote check-ups and diagnosis.
- On-body sensors for effortless and accurate health monitoring.
- Auto alert generation in case of emergency medical episodes (e.g. Heart attacks, seizures).

Pollution and Calamity Monitoring

- Monitoring for weather or man-made based calamities.
- Alert generation in case of above-threshold pollutants in the air or water.
- Resource reallocation and rerouting of services in the event of calamities.

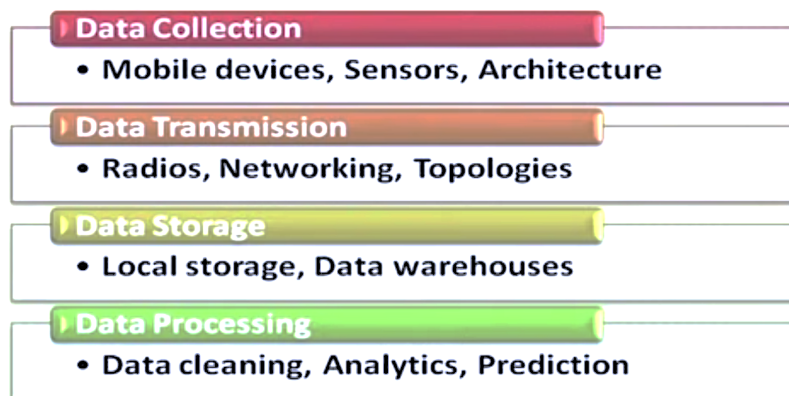
Smart Energy

- Smart metering systems.
- Smart energy allocation and distribution system.
- Incorporation of traditional and renewable sources of energy in the same grid.

Smart Agriculture

- Automatic detection of plant water stress.
- Monitoring of crop health status.
- Auto detection of crop infection.
- Auto application of fertilizers and pesticides.
- Scheduling harvesting and arranging proper transfer of harvests to warehouses or markets.

Technological Focus Areas



10.3 IOT CHALLENGES IN SMART CITIES

Security and Privacy

- Exposure to attacks (e.g. cross-site scripting, side channel, etc.).
- Exposure to vulnerabilities.
- Multi-tenancy induces the risk of data leakage.

Heterogeneity

- Integration of varying hardware platforms and specifications.
- Integration of different radio specifications.
- Integration of various software platforms.
- Accommodating varying user requirements.

Reliability

- Unreliable communication due to vehicle mobility.
- Device failures still significant

Large scale

- Delay due to large scale deployments.
- Delay due to mobility of deployed nodes.
- Distribution of devices can affect monitoring tasks.

Legal and Social aspects

- Services based on user provided information may be subject to local or international laws.
- Individual and informed consent required for using humans as data sources.

Big data

- Transfer, storage and maintenance of huge volumes of data is expensive.
- Data cleaning and purification is time consuming.
- Analytics on gigantic data volumes is processing intensive.

Sensor Networks

- Choice of appropriate sensors for individual sensing tasks is crucial.
- Energy planning is crucial.
- Device placement and network architecture is important for reliable end-to-end IoT implementation.
- Communication medium and means play an important role in seamless function of IoT in smart cities.

10.4 DATA FUSION

- Enormous volume of data is produced periodically in a smart city environment.
- Challenges include making the available/ incoming large data volume precise and accurate.
- Quality of data precision and accuracy affects the quality of decision making in IoT-enabled smart cities.
- Data fusion enables optimum utilization of massive data gathered from multiple sources, and across multiple platforms.

10.4.1 Multi-sensor Data Fusion

- Combines information from multiple sensor sources.
- Enhances the ability of decision-making systems to include a multitude of variables prior to arriving at a decision.
- Inferences drawn from multiple sensor type data is qualitatively superior to single sensor type data.
- Information fusion generated from multiple heterogeneous sensors provides for better understanding of the operational surroundings.

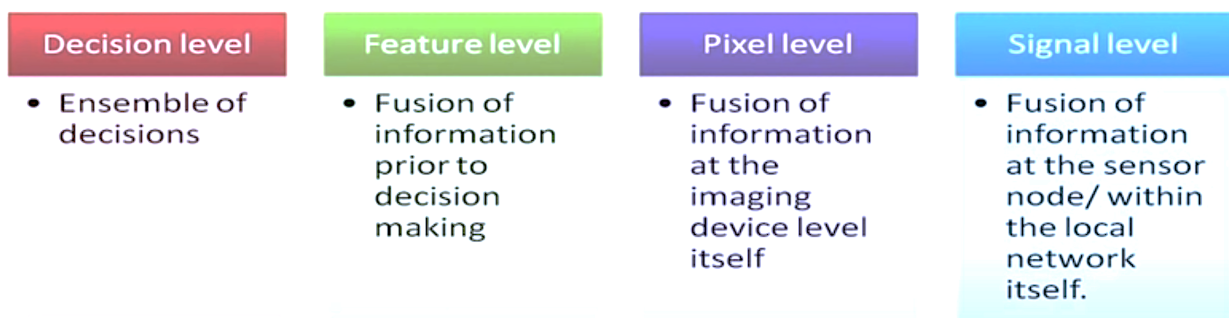
10.4.2 Challenges

- Imperfection** Inaccurate or uncertain WSN sensor data
- Ambiguity** Outliers, missing data
- Conflicts** Same sensor type reports different data for the same location.
- Alignment** Arises when sensor data frames are converted to a singular frame prior to transmission
- Trivial Features** Processing of trivial data features may bring down the accuracy of the whole system.

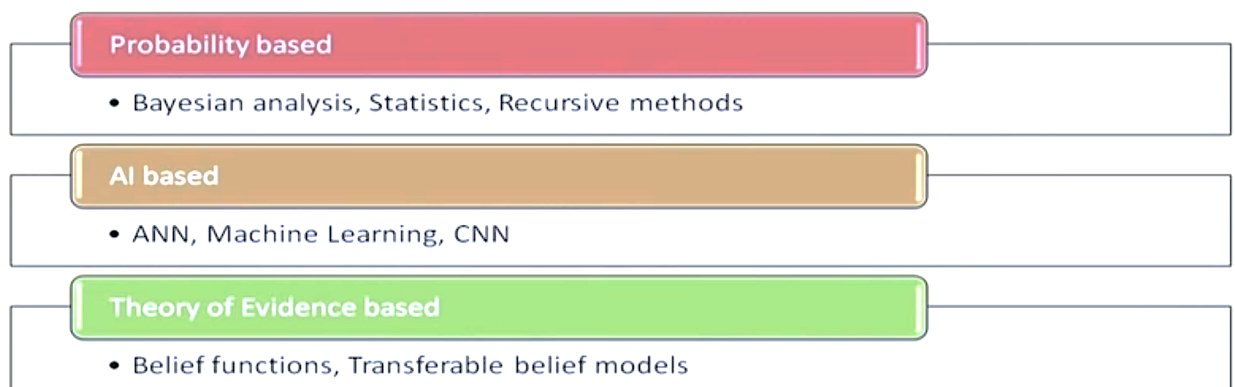
10.4.3 Data Fusion Opportunities in IoT

- Collective data is rich in information and generates better intelligence compared to data from single sources.
- Optimal amalgamation of data.
- Enhancing the collective information content obtained from multiple low-power, low-precision sensors.
- Enables hiding of critical data sources and semantics (useful in military applications, medical cases, etc.).

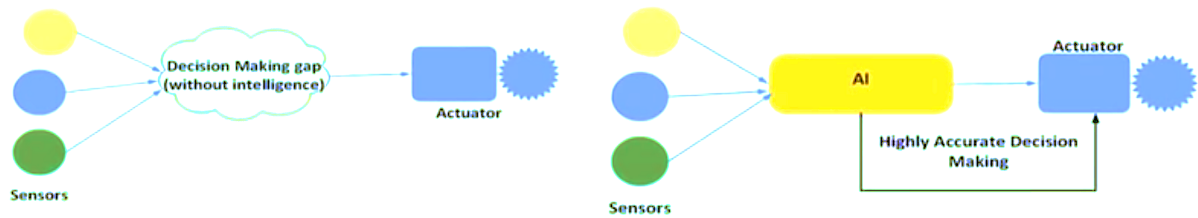
10.4.4 Stages of Data Fusion



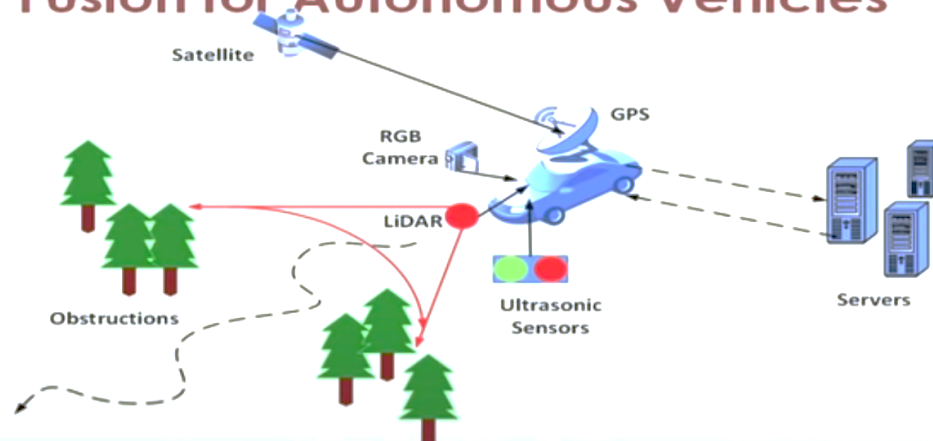
10.4.5 Mathematical Methods of Data Fusion



AI in IoT Decision Making



Data Fusion for Autonomous Vehicles



10.5 SMART PARKING

- Shortens parking search time of drivers. Reduces traffic congestion.
- Reduces pollution by keeping unnecessarily lingering vehicles off the roads.
- Reduces fuel consumption and costs. Increases urban mobility.
- Shorter parking search time results in more parked time, and hence, more revenue.

10.5.1 Functional Layers in Smart parking



10.5.2 Smart Parking: Information Collection



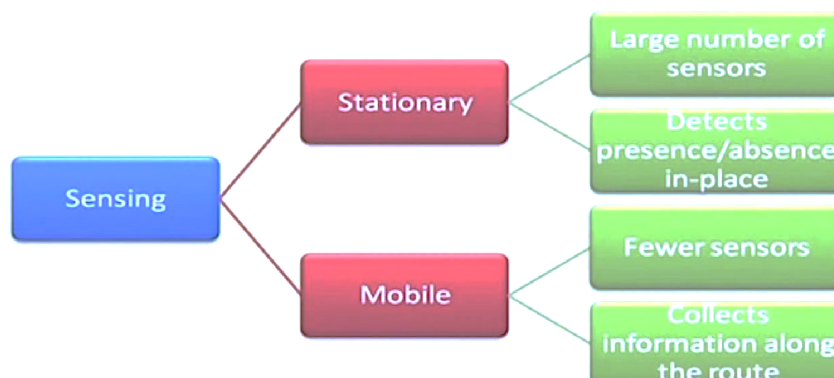
10.5.3 Smart Parking: System Deployment



10.5.4 Smart Parking: Service Dissemination



10.5.5 Information Sensing in Smart Parking



Energy Management in Smart Cities

- ✓ Energy efficient solutions
 - Lightweight protocols
 - Scheduling optimization
 - Predictive models for energy consumption
 - Cloud-based approach
 - Low-power transceivers
 - Cognitive management framework
- ✓ Energy harvesting solutions
 - Ambient energy harvesting
 - RF sources
 - Wind
 - Sun
 - Heat
 - Vibration
- ✓ Energy harvesting solutions
 - Dedicated energy harvesting
 - Energy sources intentionally deployed near IoT sources.
 - Amount of energy harvested depends upon:
 1. Sensitivity of the harvesting circuit
 2. Distance between the device and source
 3. Environment

CHAPTER 11- INDUSTRIAL INTERNET OF THINGS

11.1 IIoT requirements

11.2 Design considerations

11.3 Applications of IIoT

11.4 Benefits of IIoT

11.5 Challenges of IIoT

Introduction

- ✓ The main aim of Internet of Things (IoT) is
 - ✓ to globally connect smart 'things' or 'objects' .
 - ✓ objects are uniquely identified.
 - ✓ interoperability among the objects.
- ✓ The Industrial Internet of Things (IIoT) is an application of IoT in industries to modify the various existing industrial systems. IIoT links the automation system with enterprise, planning and product lifecycle.



Fig 1(a) : IIoT as an intersection of industries and IoT

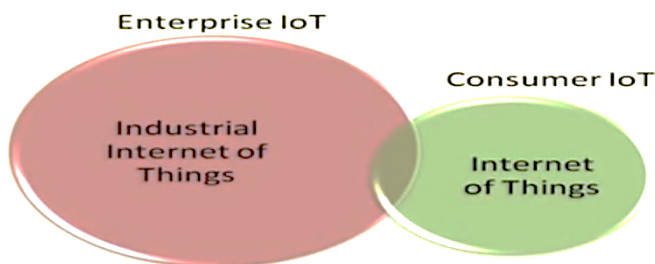


Fig 1(b) : IIoT ≠ IoT

- ✓ IIoT includes –
 - ✓ machine learning
 - ✓ big data technology
 - ✓ machine - to - machine interaction (M-2-M)
 - ✓ automation.
 - ✓ IIoT is supported by huge amount of data collected from sensors. It is based on “wrap & re-use” approach, rather than “rip & replace” approach.
- (Source : <http://www.mhi.org>)

- ✓ 1st Industrial Revolution : Mechanized production
- ✓ 2nd Industrial Revolution : Mass production
- ✓ 3rd Industrial Revolution : Internet evolution and automation
- ✓ 4th Industrial Revolution : IIoT

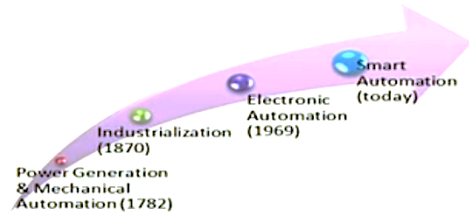
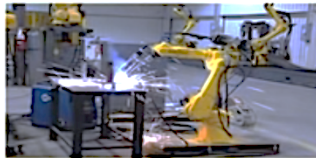


Fig 2: Industry 4.0

Source: <http://www.industry40wood.com>



Source: <https://www.artika.info>



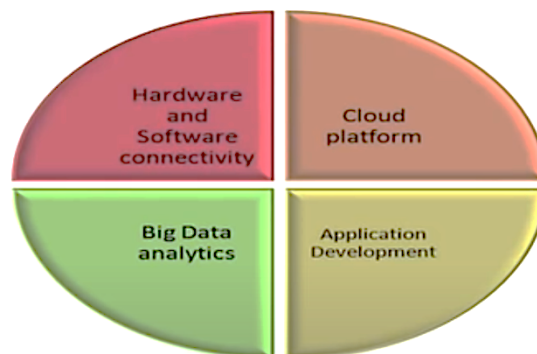
Source: <http://www.rehm-group.com>



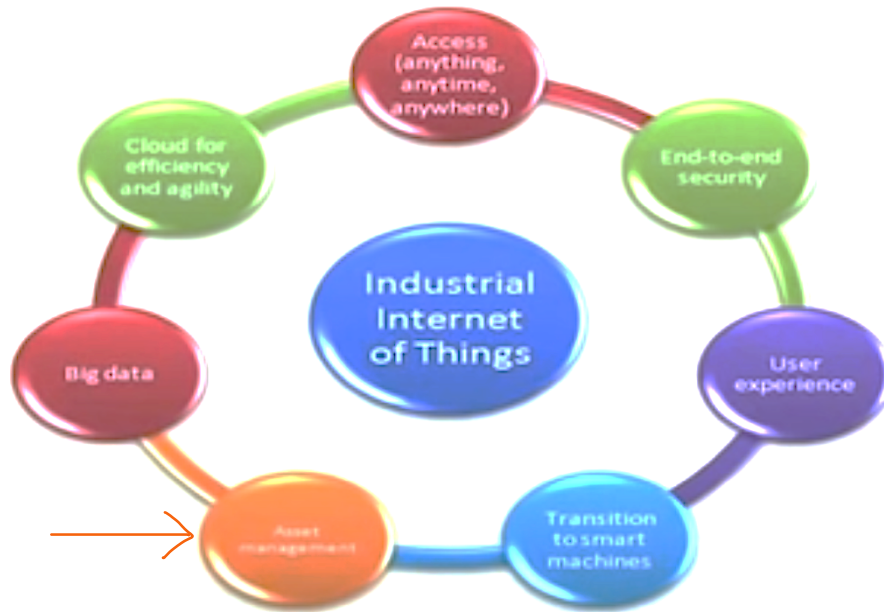
IIoT : 2nd generation of Internet evolution and 4th Industrial Automation

- ✓ IIoT is a network of
 - ✓ physical objects
 - ✓ systems
 - ✓ platforms
 - ✓ applications
- ✓ These networks can communicate with each other, external environment and other people.
- ✓ The acquisition of IIoT has led to availability and affordability of sensors, processors, and other technologies which facilitates capture and access to real-time information

IIoT Requirements



Asset
mgmt.

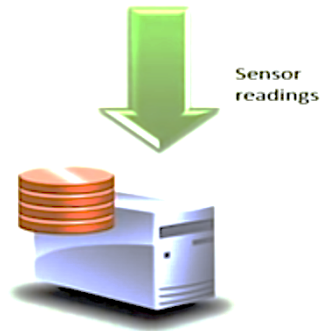


Physical Plant

Machine
instructions



Virtual Plant

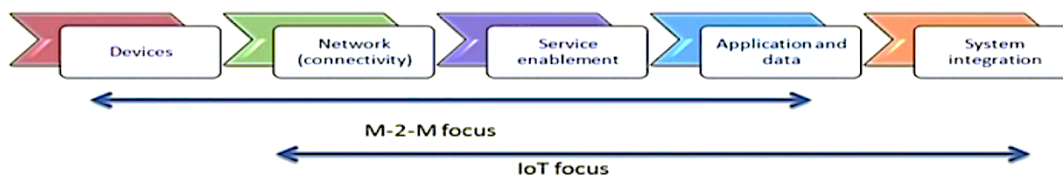
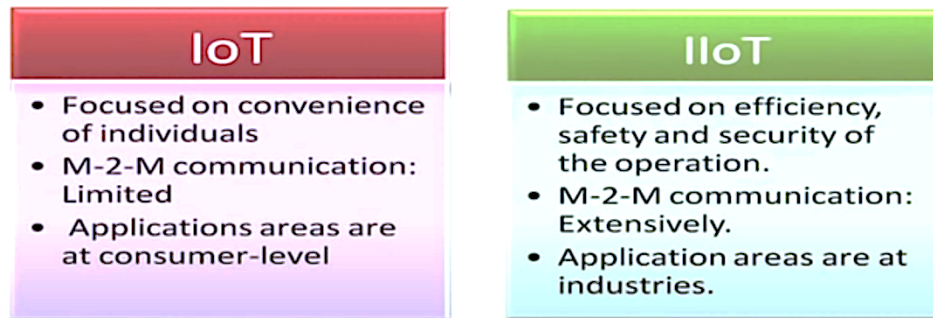


Design Considerations

- ✓ To use an IoT device for industrial applications, the following design objectives are to be considered –
 - ✓ Energy : Time for which the IoT device can operate with limited power supply.
 - ✓ Latency : Time required to transmit the data.
 - ✓ Throughput : Maximum data transmitted across the network.
 - ✓ Scalability : Number of devices supported.
 - ✓ Topology: Communication among the devices, i.e. interoperability.
 - ✓ Safety and Security: Degree of safety and security of the application.

Difference between IoT and IIoT

The main differences between IoT and IIoT are :



Service Management in IIoT

- ✓ “Service management refers to the implementation and management of the quality of services which meets the end-users demand”
- ✓ “Service is a collection of data and associated behaviors to accomplish a particular function or feature of a device or portions of a device”.
- ✓ Service can be of two types, which are -
 - ✓ Primary service - The basic services which are responsible for the primary node functions are termed as primary service.
 - ✓ Secondary service - The auxiliary functions which provide services to the primary service or secondary services are termed as secondary service.

Applications of IIoT

- ✓ The key application areas of IIoT are -
 - ✓ Manufacturing industry
 - ✓ Healthcare Service industry
 - ✓ Transportation & logistics
 - ✓ Mining
 - ✓ Firefighting

Manufacturing Industry

- ✓ The devices, equipment, workforce, supply chain, work platform are integrated and connected to achieve smart production. This will lead to –
 - ✓ reduction in operational costs
 - ✓ improvement in the productivity of the worker
 - ✓ reduction in the injuries at the workplace
 - ✓ resource optimization and waste reduction
 - ✓ end-to-end automation.

Healthcare Service Industry

- ✓ Patients can be continuously monitored due to the implanted on-body sensors. This has led to –
 - ✓ improved treatment outcome
 - ✓ costs have reduced
 - ✓ improved disease detection
 - ✓ improved accuracy in the collection of data
 - ✓ improved drugs management.

Transportation & logistics

- ✓ To improve safety, efficiency of transportation, Intelligent Transportation system (ITS) is developed which consists of connected vehicles. ITS provides –
 - ✓ Vehicle – to – sensor connectivity
 - ✓ Vehicle – to – vehicle connectivity
 - ✓ Vehicle – to – internet connectivity
 - ✓ Vehicle – to – road infrastructure
 - ✓ Dedicated short-range communications (DSRC) is the key enabling technology for V2V and V2R communications.
-
- ✓ In IIoT scenario the physical objects are provided with
 - ✓ bar codes
 - ✓ RFID tagshence, real-time monitoring of the status and location of the physical objects from destination to the origin, across the supply chain is possible.
 - ✓ Security and privacy of the data should be maintained.

Mining

- ✓ To prevent accidents inside the mines - RFID, Wi-Fi and other wireless technologies are used, which
 - ✓ provides early warning of any disaster
 - ✓ monitors air-quality
 - ✓ detects the presence of poisonous gases inside the mines
 - ✓ oxygen level inside the mines.

Firefighting

- ✓ Sensor networks, RFID tags are used to perform
 - ✓ automatic diagnosis
 - ✓ early warning of disaster
 - ✓ emergency rescue
 - ✓ provides real-time monitoringHence, improves public security.

Examples of IIoT

- ✓ Examples of IIoT are -
 - ✓ unmanned aerial vehicles (UAVs) to inspect oil pipelines.
 - ✓ monitoring food safety using sensors.
 - ✓ minimizing workers' exposure to noise, chemicals and other hazardous gases.
 - ✓ unmanned marine vehicle which can collect data up to a year without fuel or crew.

Connected Ecosystems in IIoT scenario

- ✓ Traditional supply chains in industries are linear in nature.
- ✓ To shift the business focus from products to outcomes, new ecosystem should be followed.
- ✓ Digital ecosystems progress at a much faster rate than physical industries. Hence, it can quickly adapt to the changes in the external environments.

Integration of Digital and Human Workforce

- ✓ In IIoT, machines become more intelligent. Hence, the automated tasks can be done in the industries at lower costs and higher quality level.
- ✓ Humans will work with machines, the outcome will be higher overall productivity.
- ✓ IIoT will reform and redefine the skills of the workers.

Creation of New Jobs

- ✓ The creation of new composite industries, such as precision agriculture, digital healthcare system, digital mines etc., will lead to development of new job opportunities.
- ✓ Highly automated machines will require lesser number of unskilled workers, but will require skilled experts with digital and analytical skills.

Reformation of Robots

- ✓ In IIoT environment, robots are featured with three capabilities : sensing, thinking and acting. They will be reformed with the ability to carry out repetitive tasks.
- ✓ Robots will be more intelligent but will work under the supervision of human beings. Their availability will increase.
- ✓ Robots will be reprogrammable to perform new tasks. They have the capability to 'learn' faster.

Challenges in IIoT

- ✓ Primary challenges



- ✓ Safety Challenges



- ✓ Hazards (related)



Standardization

- ✓ Standardization plays an important role in the development of the system.
- ✓ Goal: To improve the interoperability of the different systems/ applications and allow the products/services to perform better.

Standardization

- ✓ The problems related to standardization are:
 - ✓ Interoperability
 - ✓ Semantic interoperability (data semantics)
 - ✓ Security and privacy
 - ✓ Radio access level issues.

Privacy and security issues

- ✓ The two most important concerns related with IIoT are -
 - ✓ information security
 - ✓ data privacy protection
- ✓ The devices/things can be tracked, monitored and connected. So there are chances of attack on the personal and private data.

Privacy and security issues

- ✓ Examples –
 - ✓ Healthcare industry – the medical data of a patient must not be tampered, or altered by any person in the middle.
 - ✓ Food industry – the deterioration of any food item being sent to the company must be kept confidential as it will affect the reputation of the company.

Risks associated with IIoT in Manufacturing

- ✓ Though IIoT provides new opportunities, but few factors may cause hindrance in the path to success, which are :
 - ✓ lack of vision and leadership
 - ✓ lack of understanding of values among management employees
 - ✓ costly sensors
 - ✓ inadequate infrastructure.

Meet the challenges: Sensor improvement

- ✓ Improvement in sensor technologies –
 - ✓ miniaturization
 - ✓ performance
 - ✓ cost and energy consumption.

Meet the challenges : Manufacturing

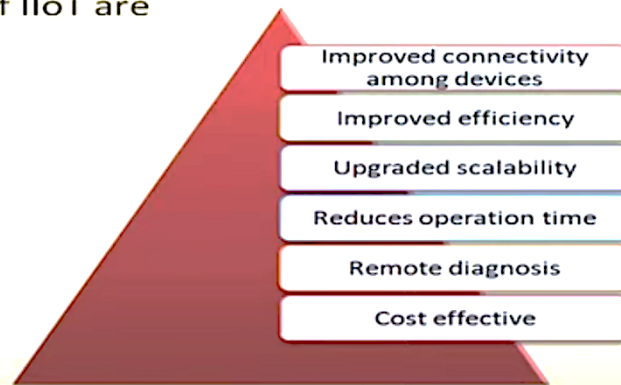
- ✓ Manufacturers use software capabilities to improve operational efficiency through –
 - ✓ predictive maintenance
 - ✓ savings on scheduled repairs
 - ✓ reduced maintenance costs
 - ✓ reduced number of breakdowns.

Case study : Rt Tech Software

- ✓ Rt Tech particularizes in software which –
 - ✓ improves industrial facilities' efficiency
 - ✓ improves productivity.
- ✓ Energy management solution, which leads to reduction in the plant's highest variable cost.
- ✓ Rt Tech automates the process of mapping and managing energy consumption.

Benefits of IIoT

- ✓ The benefits of IIoT are



Recent Research trends in IIoT

- ✓ Recent research challenges in IIoT are -
 - ✓ To improve the communications among the different things or objects.
 - ✓ To develop energy-efficient techniques so as to reduce power consumption by sensors.
 - ✓ To develop context-aware IoT middleware for better understanding of the sensor data.
 - ✓ To create smart objects with larger memory, processing and reasoning capabilities.

PRODUCTS DEVELOPED

- ✓ M-2-M communication : Intelligent Radio Modem (IRM)
 - ✓ IRM 1500 & ACE 1000 - IRM
 - ✓ simple
 - ✓ M-2-M connectivity
 - ✓ data transmission
 - ✓ These devices provide easy maintenance and installation. They can be connected to IP and non-IP serial devices to extend the capability to monitor and communicate with other technologies.

Conclusion

- ✓ IIoT system requires the following :
 - ✓ Smaller, less expensive sensors which makes them easily accessible.
 - ✓ Distributed control of assembly line, automated monitoring, control and maintenance.