IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Implementation of IoT with Raspberry Pi: Part 2

Dr. Sudip Misra
Associate Professor
Department of Computer Science and Engineering
IIT KHARAGPUR
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/

IOT

Internet Of Things

- Creating an interactive environment
- Network of devices connected together

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

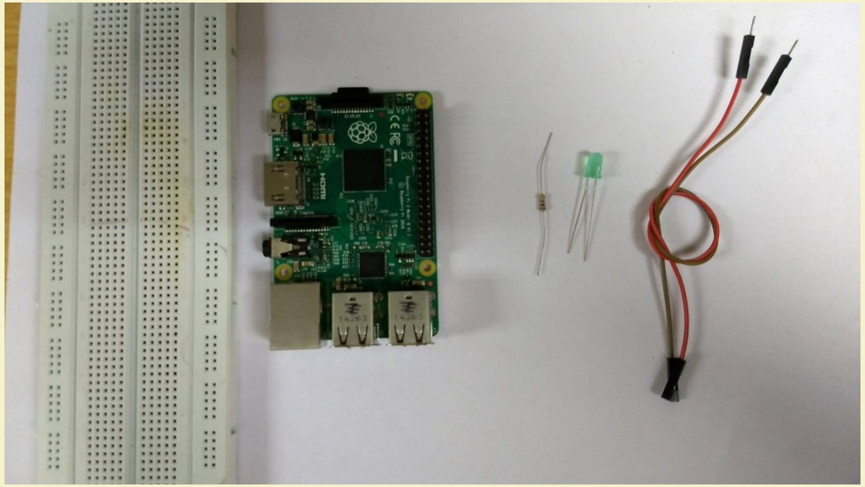
System Overview:

- 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

IOT: Remote Data Logging (contd..)

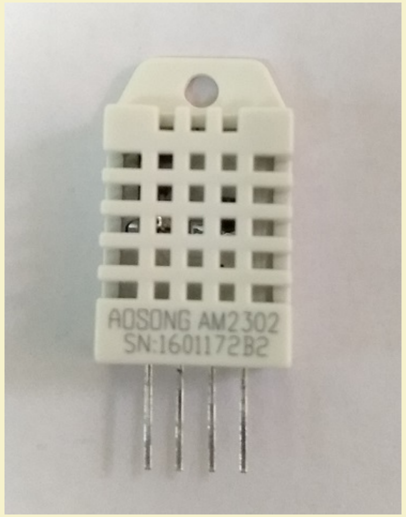
Requirements

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



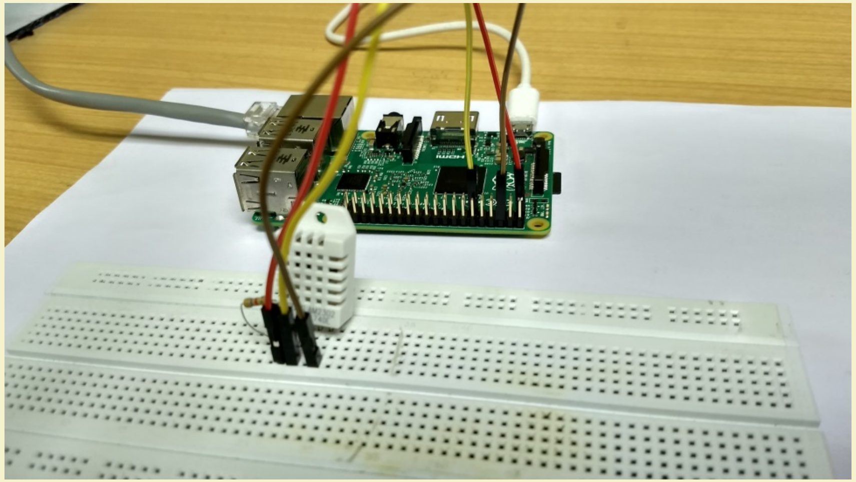
DHT Sensor

- 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

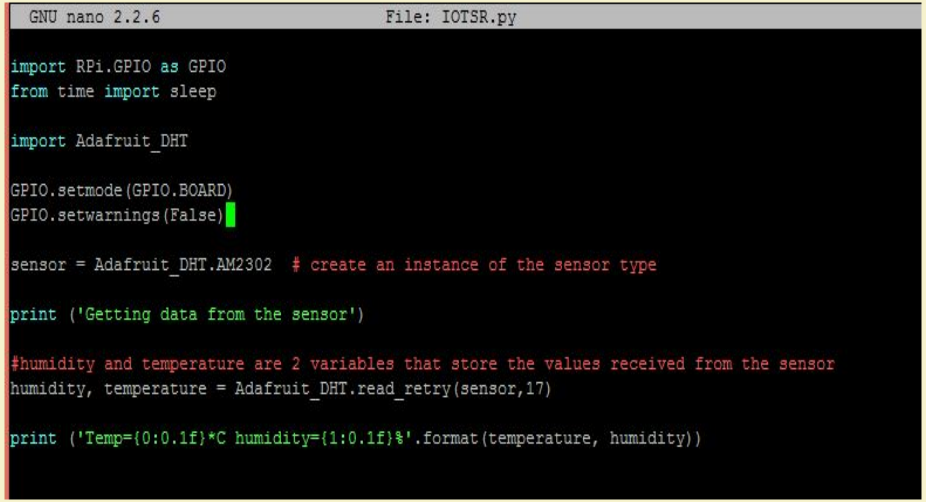
Install the library in Raspberry Pi

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

Source: ADAFRUIT DHTXX SENSORS, Lady Ada, 2012-07-29

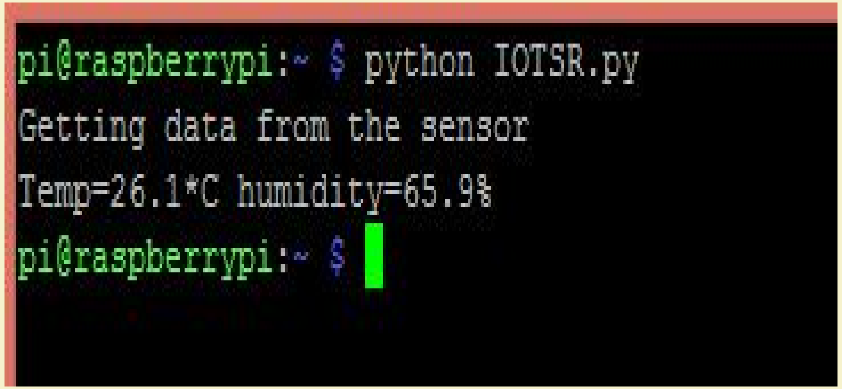
Program: DHT22 interfaced with Raspberry Pi

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

Source: PYTHON NETWORK PROGRAMMING, TutorialsPoint

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
```

Source: PYTHON NETWORK PROGRAMMING, TutorialsPoint

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
```

Source: PYTHON NETWORK PROGRAMMING, TutorialsPoint

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()
```

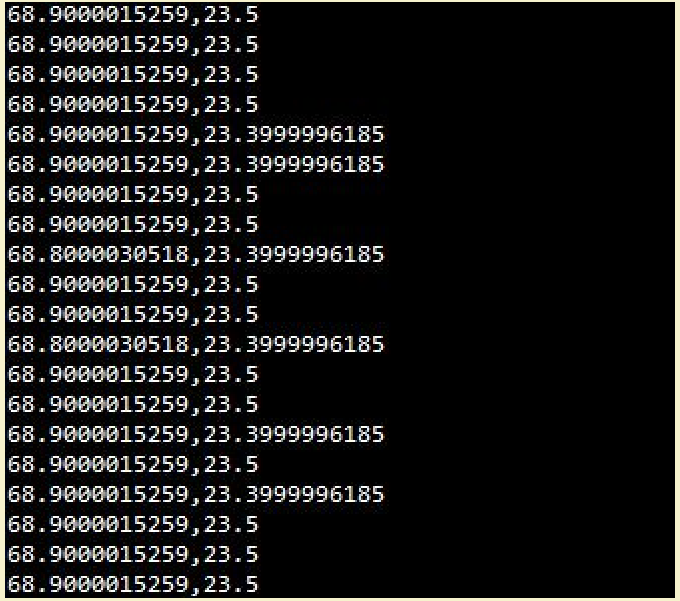
Sending Data to a Server (contd..)

Server Code: Receive data from client and save it

```
# Create a UDP socket
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)
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
```

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Implementation of IoT with Raspberry Pi: Part 3

Dr. Sudip Misra
Associate Professor
Department of Computer Science and Engineering
IIT KHARAGPUR
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/

IOT

Internet Of Things

- Creating an interactive environment
- Network of devices connected together

IOT: Remote Data Logging

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

IOT: Remote Data Logging

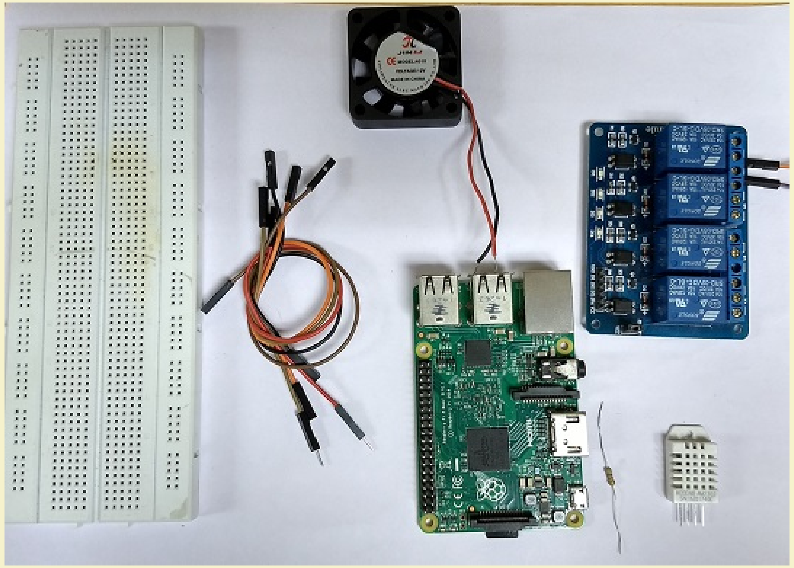
System Overview:

- 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
- **Data Splitting**
- **Plot the data**

IOT: Remote Data Logging (contd..)

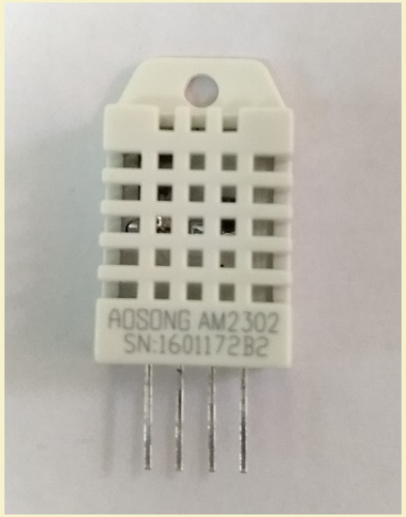
Requirements

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



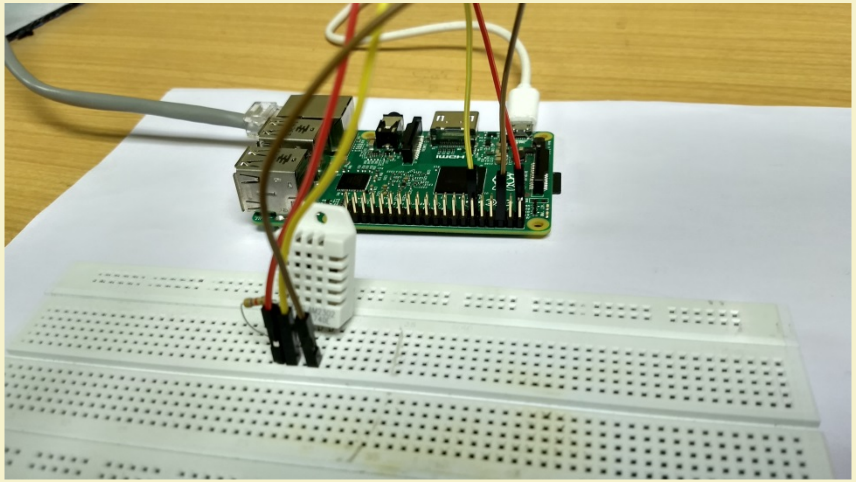
DHT Sensor

- 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



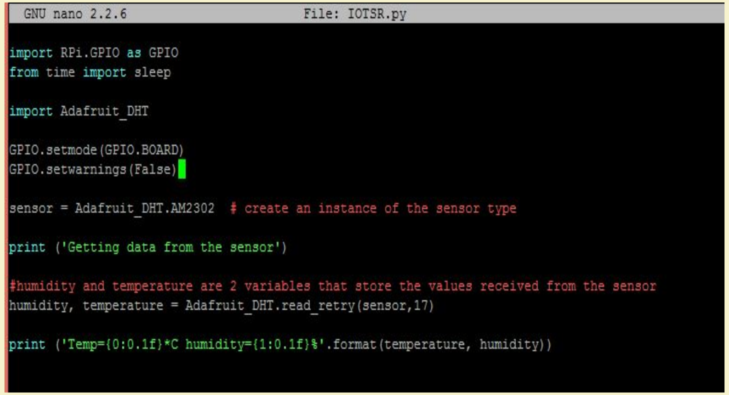
Sensor- Raspberry Pi Interface

- 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

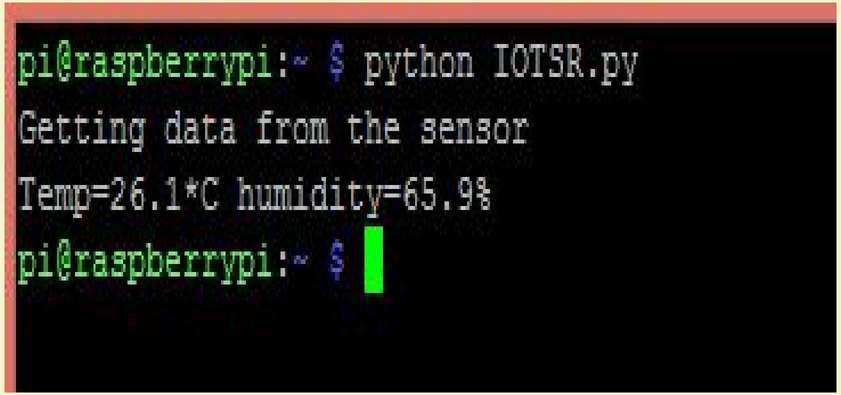


Read Data from the Sensor

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.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))
```



```
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
```

Source: HOW TO USE SPLIT IN PYTHON, PythonForBeginners, Sep 26, 2012

Data Processing

Plotting the data:

- MATPLOTLIB is a python library used to plot in 2D
  - 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

Source: MATPLOTLIB, John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team, 2012 - 2016

Data Processing (contd..)

Plotting the data:

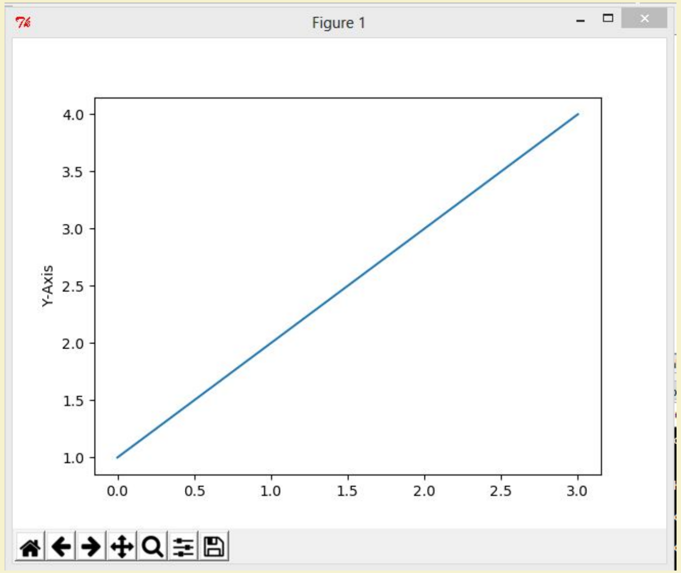
```
import matplotlib.pyplot as myplot
myplot.plot([1,2,3,4])
myplot.ylabel(‘Y-Axis’)
myplot.show()
```

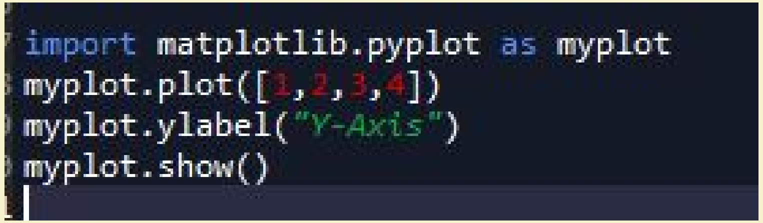
By default the values are taken for y-axis, values for x-axis are generated automatically starting from 0

Source: MATPLOTLIB, John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team, 2012 - 2016

Data Processing (contd..)

Basic Plot:





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

Data Proceessing (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

Source: MATPLOTLIB, John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team, 2012 - 2016

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]
    tem=data.split(",")[1]
    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 ($^0 C$)’)
    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)
```

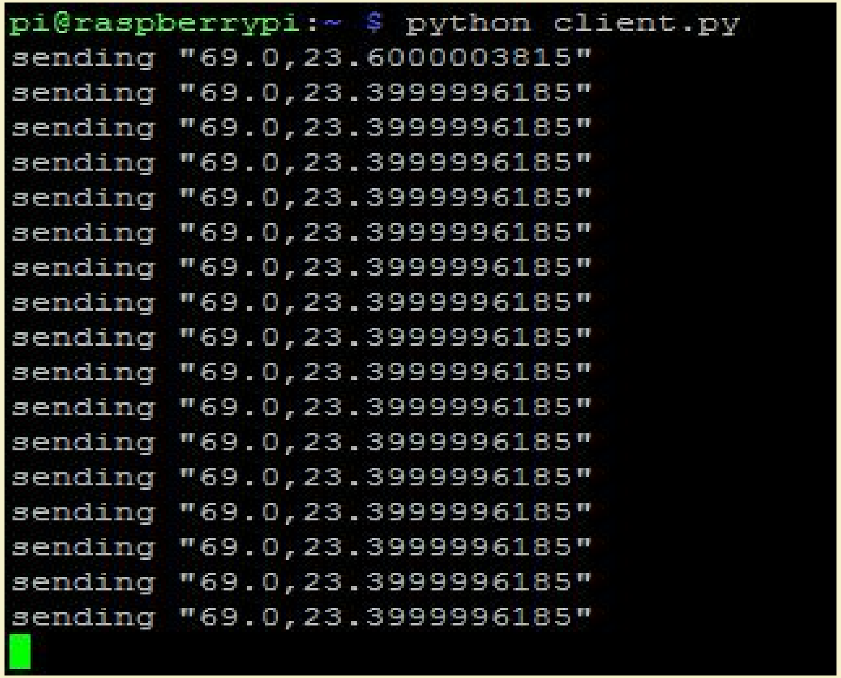
Sending Data to a Server (contd..)

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()
```

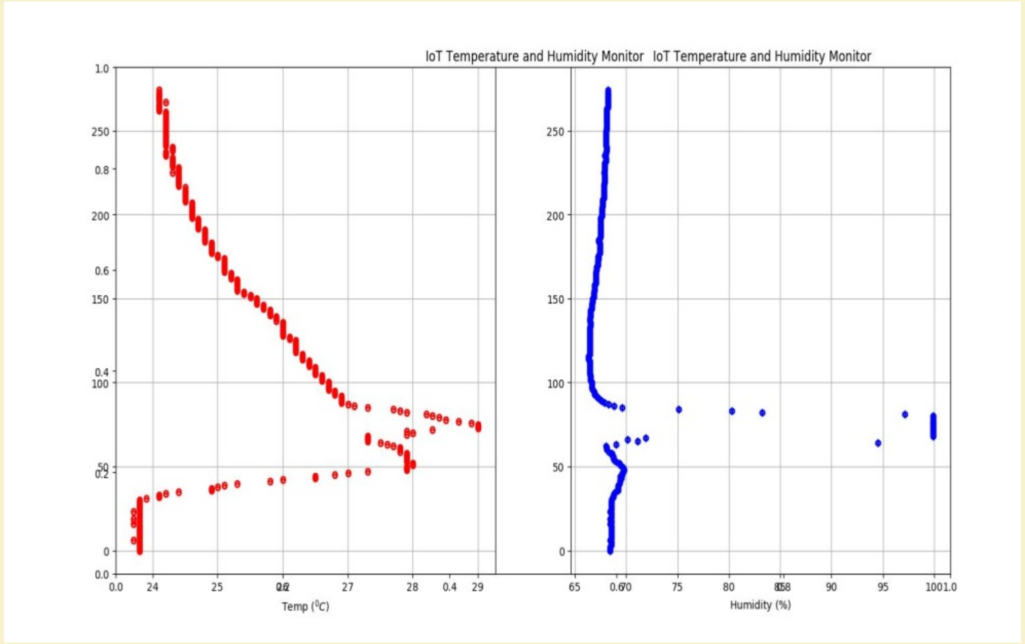
Output

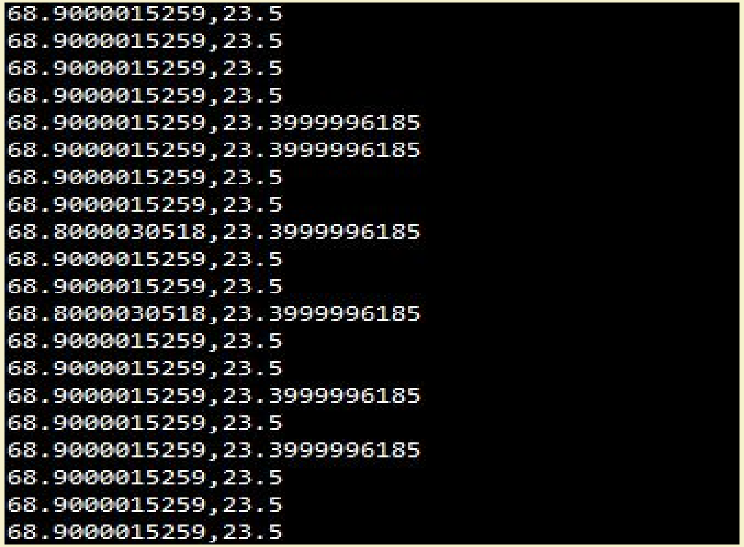
- The Reading from the sensor is sent to the Server and saved in a text file.
- Two different plots for temperature and humidity data



```
pi@raspberrypi:~ $ python client.py
sending "69.0,23.6000003815"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
sending "69.0,23.3999996185"
```

Output





```
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
```

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

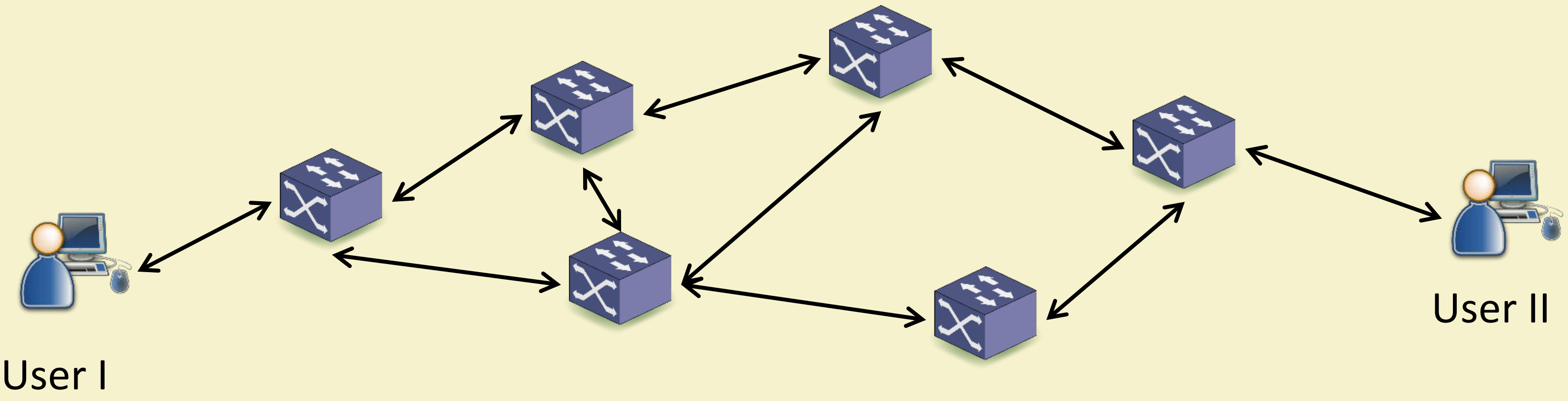
NPTEL

Software-Defined Networking – Part I

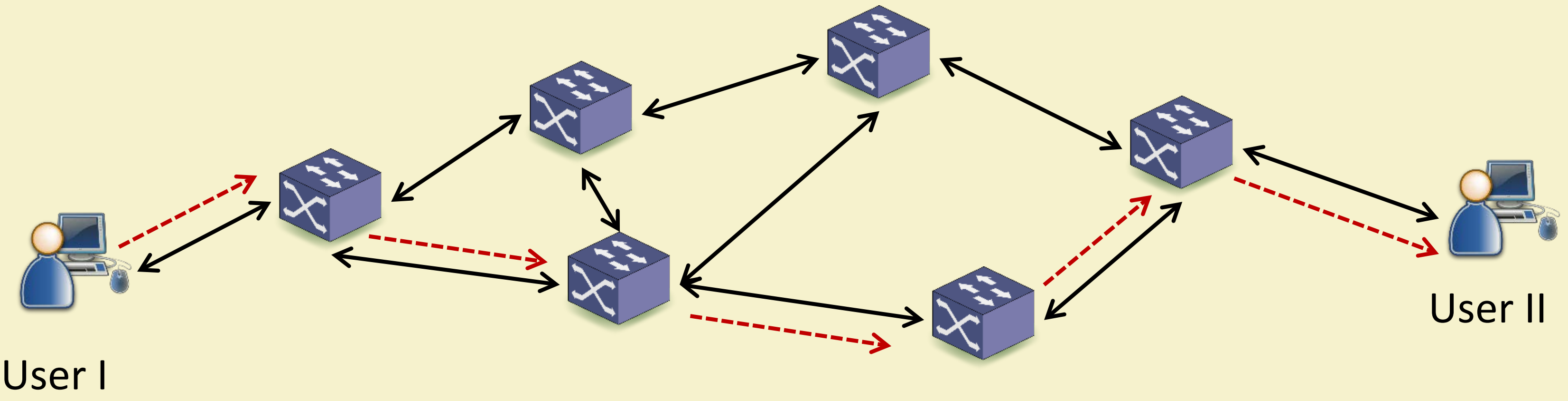
Restructuring the Current Network Infrastructure

Dr. Sudip Misra
Associate Professor
Department of Computer Science and Engineering
IIT KHARAGPUR
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/

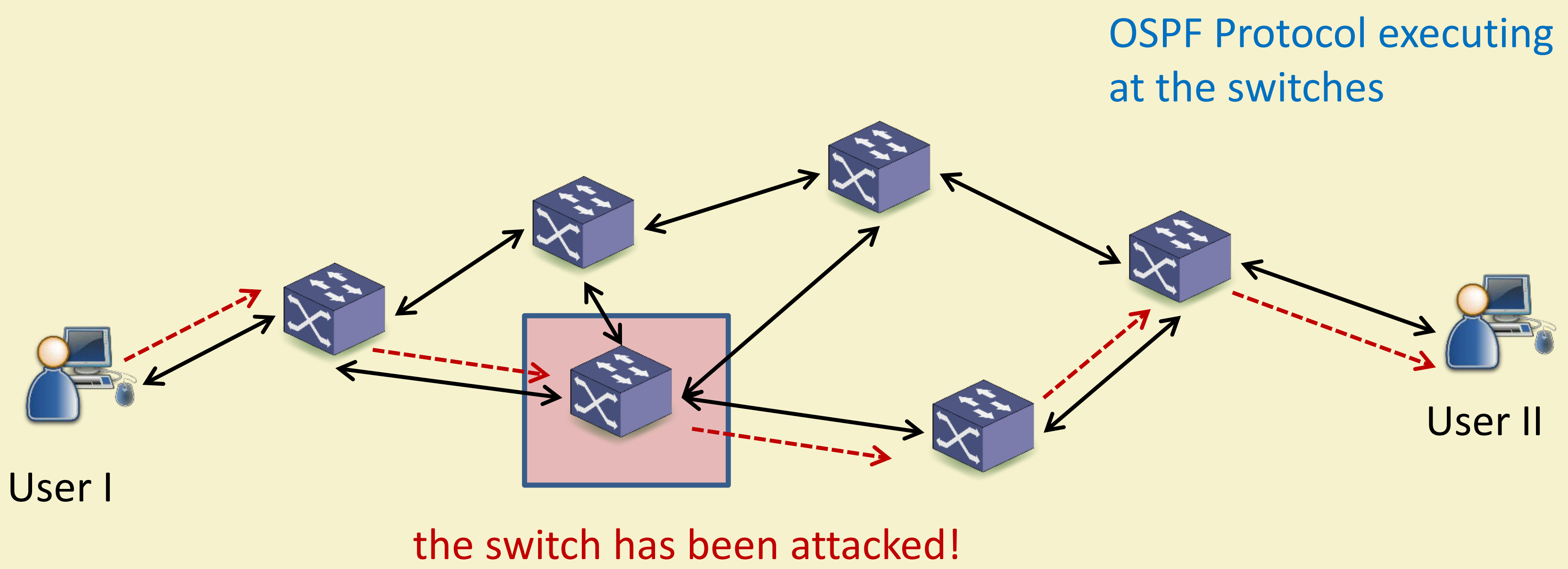
Overview of Current Network



Overview of Current Network

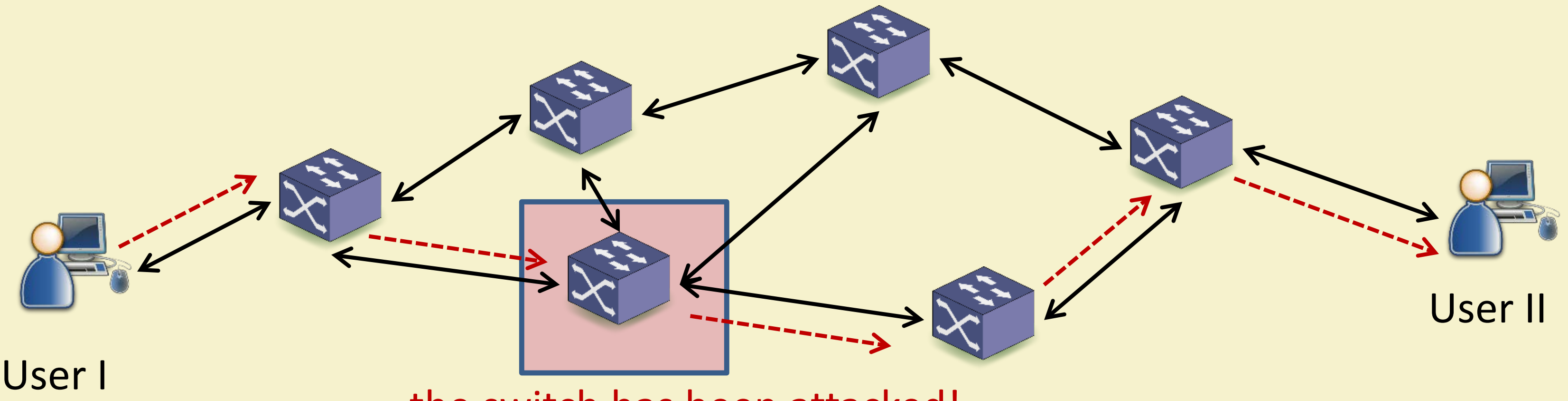


Overview of Current Network


OSPF Protocol executing at the switches
User I
User II
the switch has been attacked!

Overview of Current Network

OSPF Protocol executes at the switches

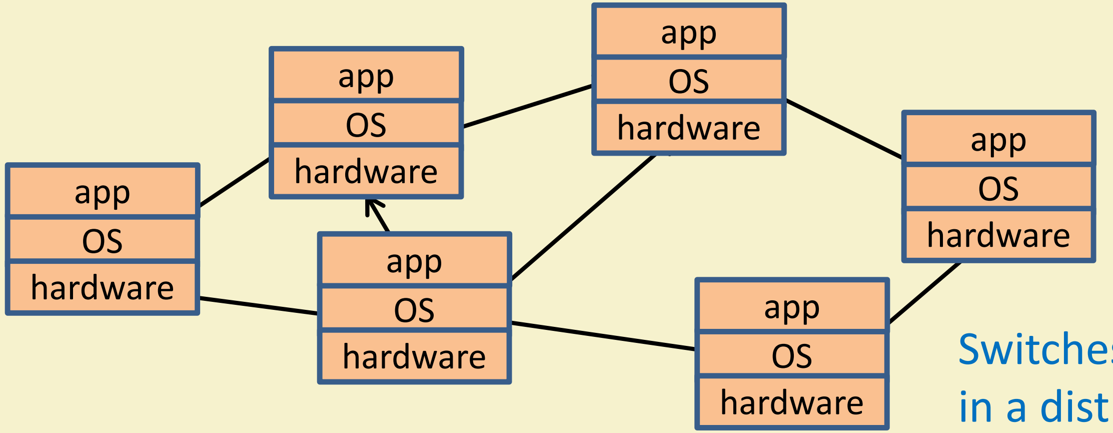


the switch has been attacked!

needs to route through an alternate path!

Present: No centralized control.

Limitations in Current Network

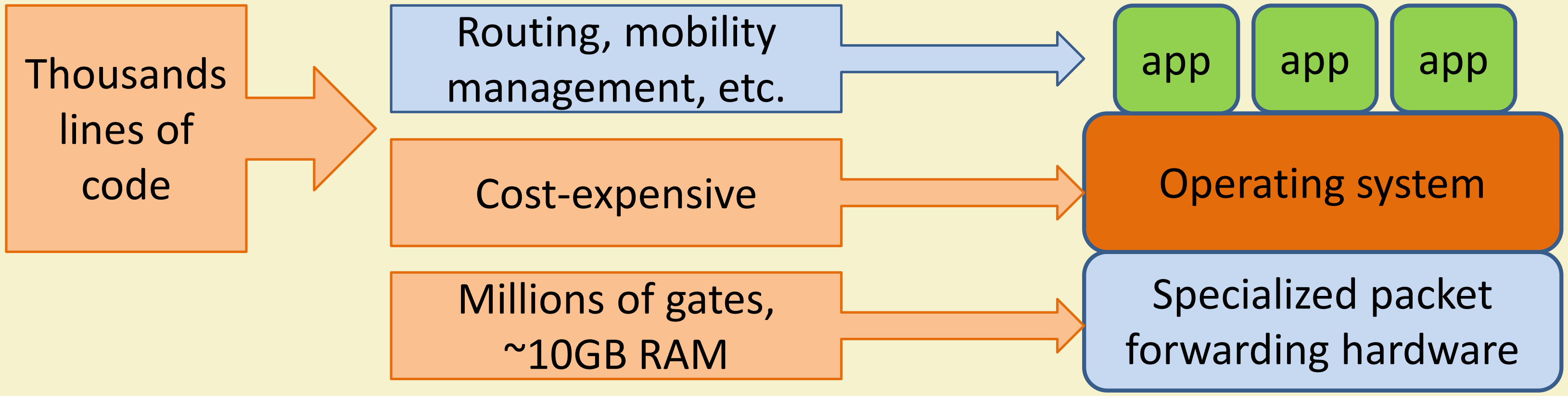


Switches forward traffic in a distributed manner. They do not have a global view of the network

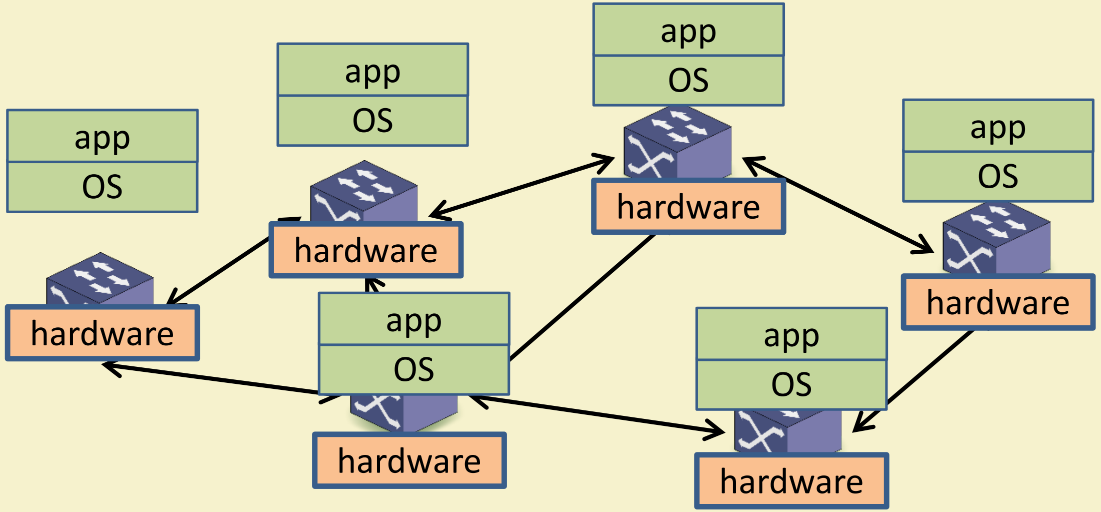
Limitations in Current Network

- ✓ 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.

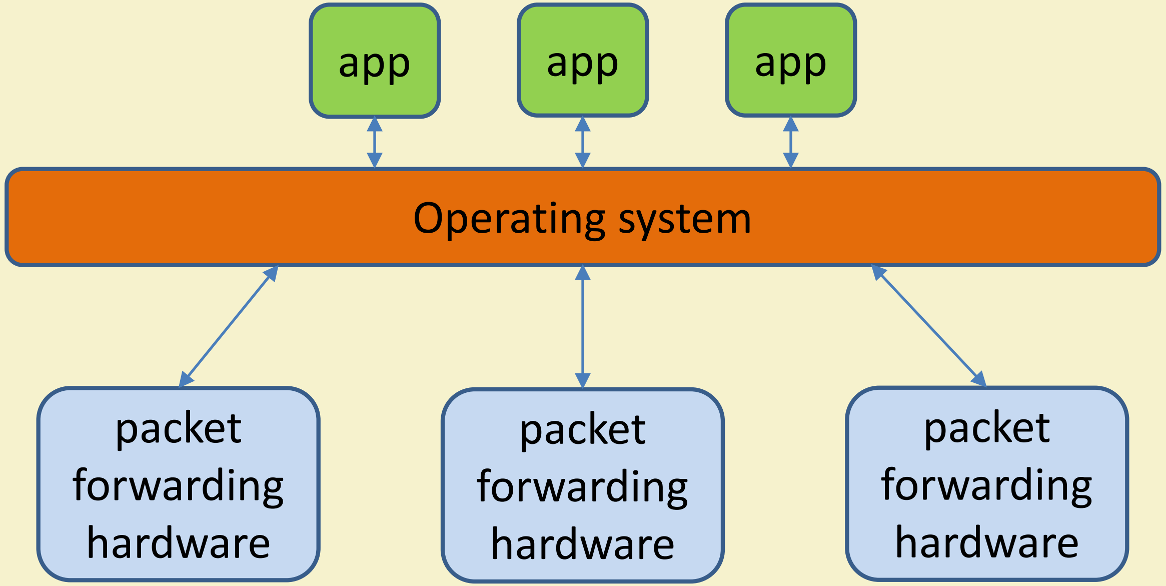
Limitations in Current Network


Thousands lines of code
Routing, mobility management, etc.
Cost-expensive
Millions of gates, ~10GB RAM
app
app
app
Operating system
Specialized packet forwarding hardware

Current Network to SDN



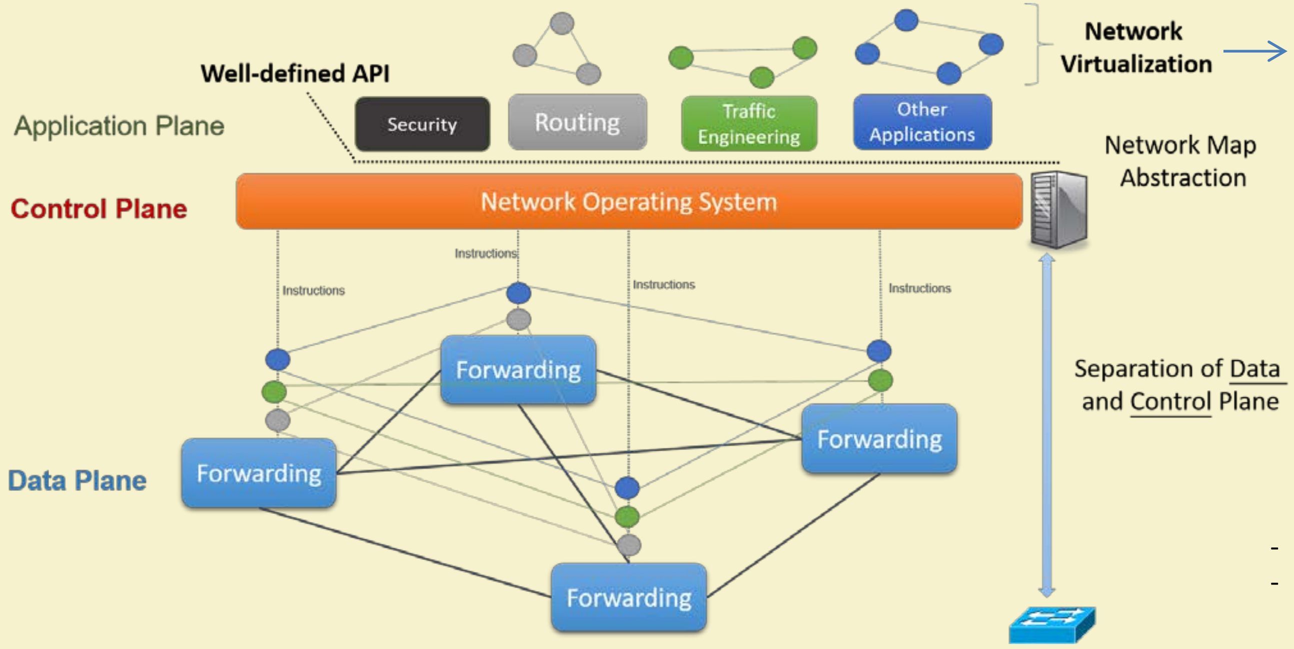
Current Network to SDN


app
app
app
Operating system
packet forwarding hardware
packet forwarding hardware
packet forwarding hardware

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



→ Network Function Virtualization

Notes
- Northbound API
- Southbound API (via OpenFlow)

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.

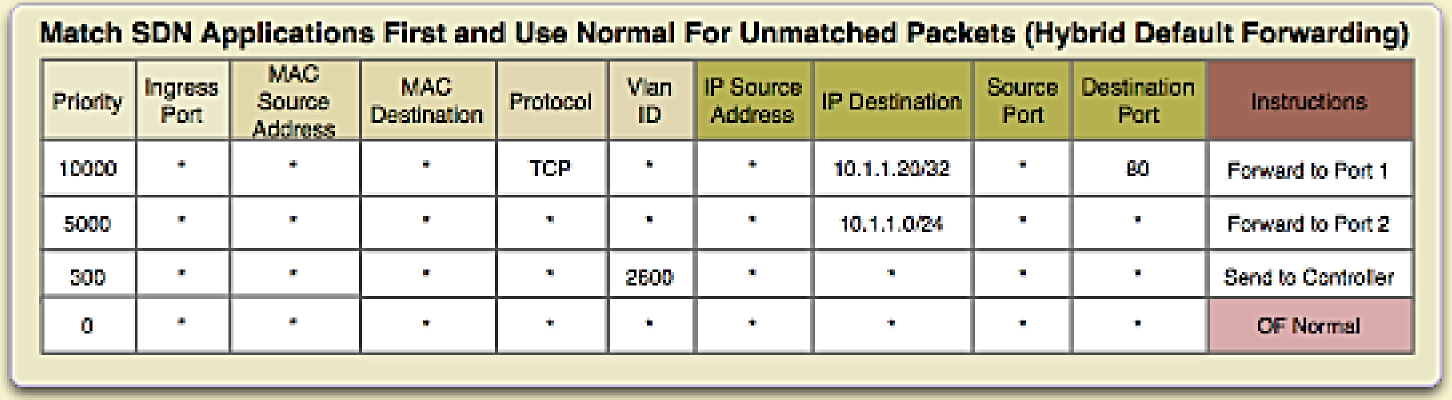
Challenges

- ✓ Rule placement
- ✓ Controller placement

Rule Placement I

- ✓ 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).

Rule Placement II



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

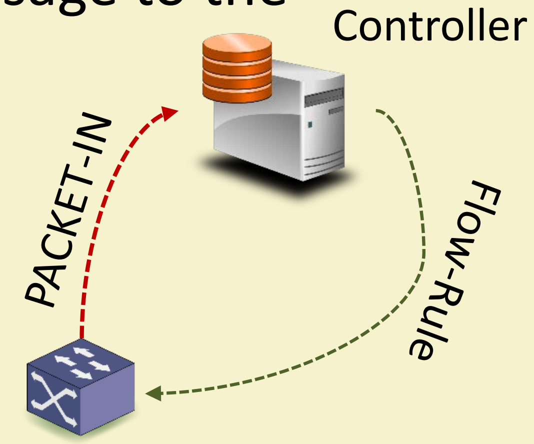
Source: http://networkstatic.net/wp-content/uploads/2013/06/OFP_normal_rules.png

Rule Placement Challenges I

- ✓ 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.

Rule Placement Challenges II

- ✓ 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

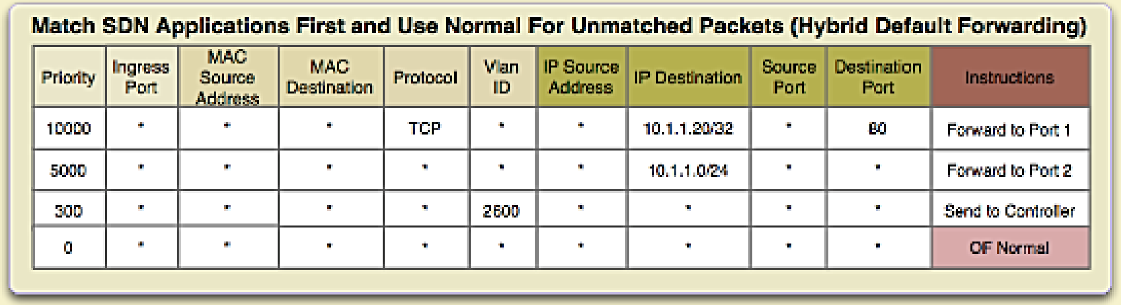


Rule Placement III

- ✓ 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.



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	*	*	*	*	2600	*	*	*	*	Send to Controller
0	*	*	*	*	*	*	*	*	*	OF Normal

Source: http://networkstatic.net/wp-content/uploads/2013/06/OFP_normal_rules.png

OpenFlow Protocol II

- ✓ Different match-fields
  - Source IP
  - Destination IP
  - Source Port
  - Priority
  - etc.

OpenFlow Protocol III

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 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.

OpenFlow Protocol IV

✓ 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

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Software-Defined Networking – Part II

Restructuring the Current Network Infrastructure

Dr. Sudip Misra
Associate Professor
Department of Computer Science and Engineering
IIT KHARAGPUR
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/

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.

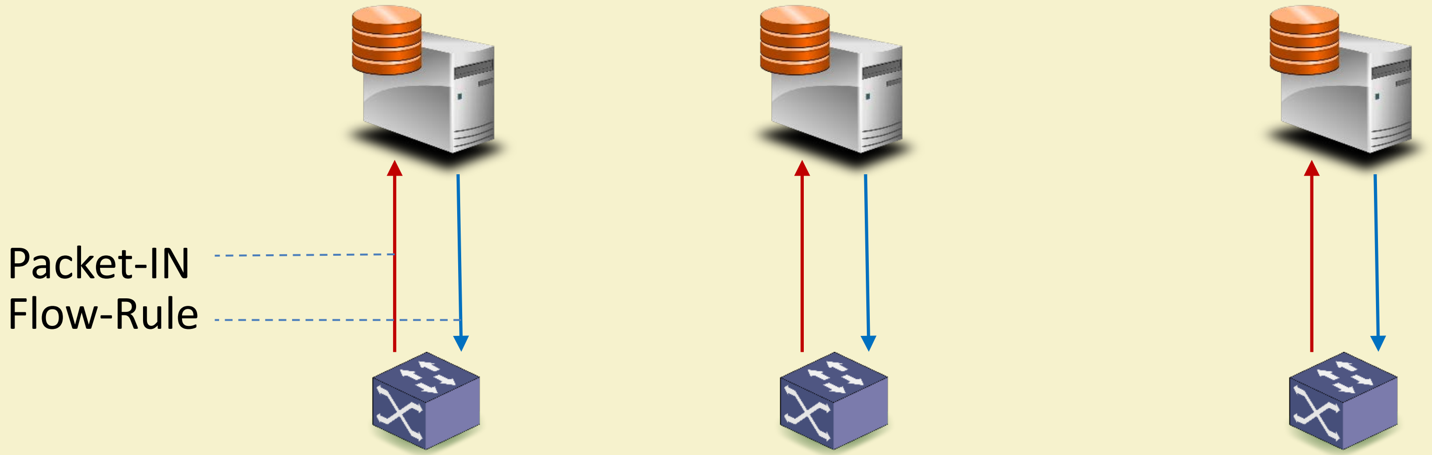
Controller Placement I

- ✓ 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).

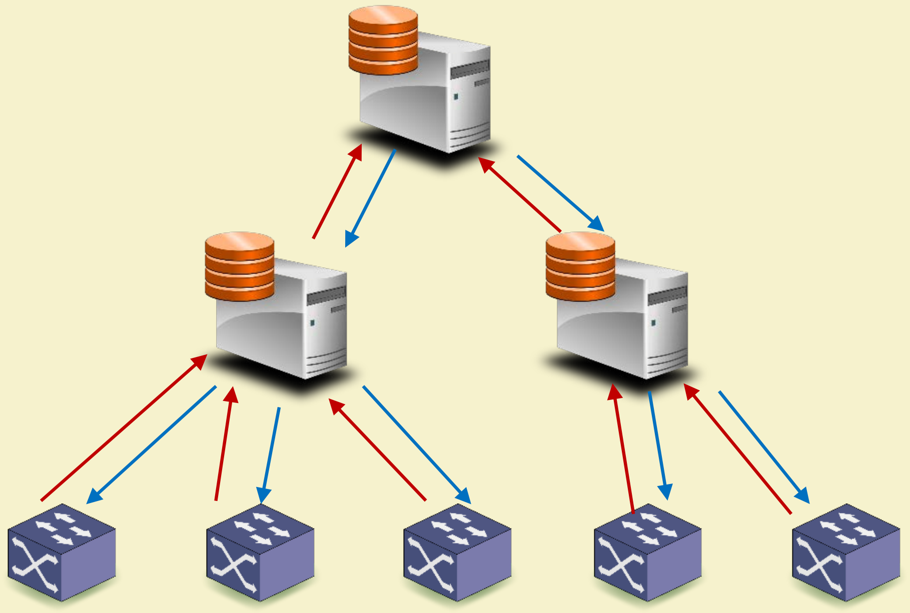
Controller Placement II

- ✓ The controllers are logically connected to the switches in <u>one-hop</u> 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).

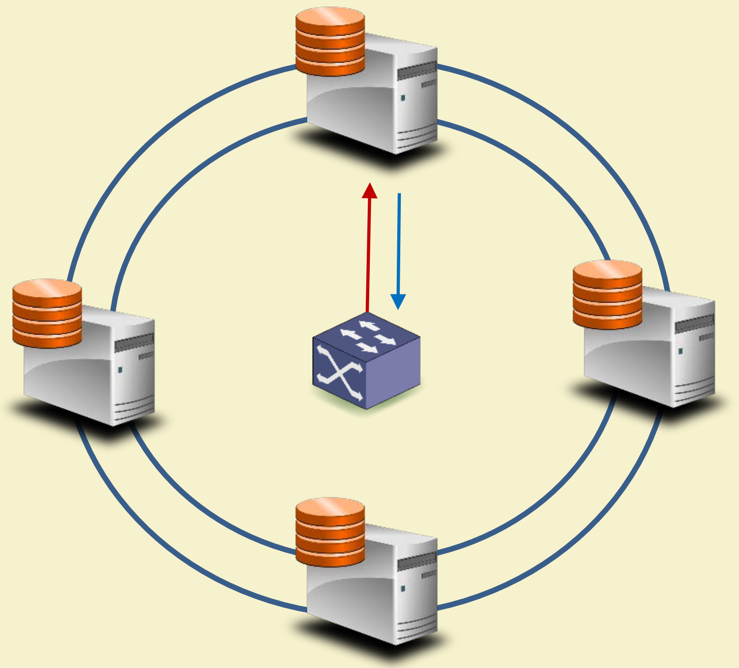
Flat Architecture



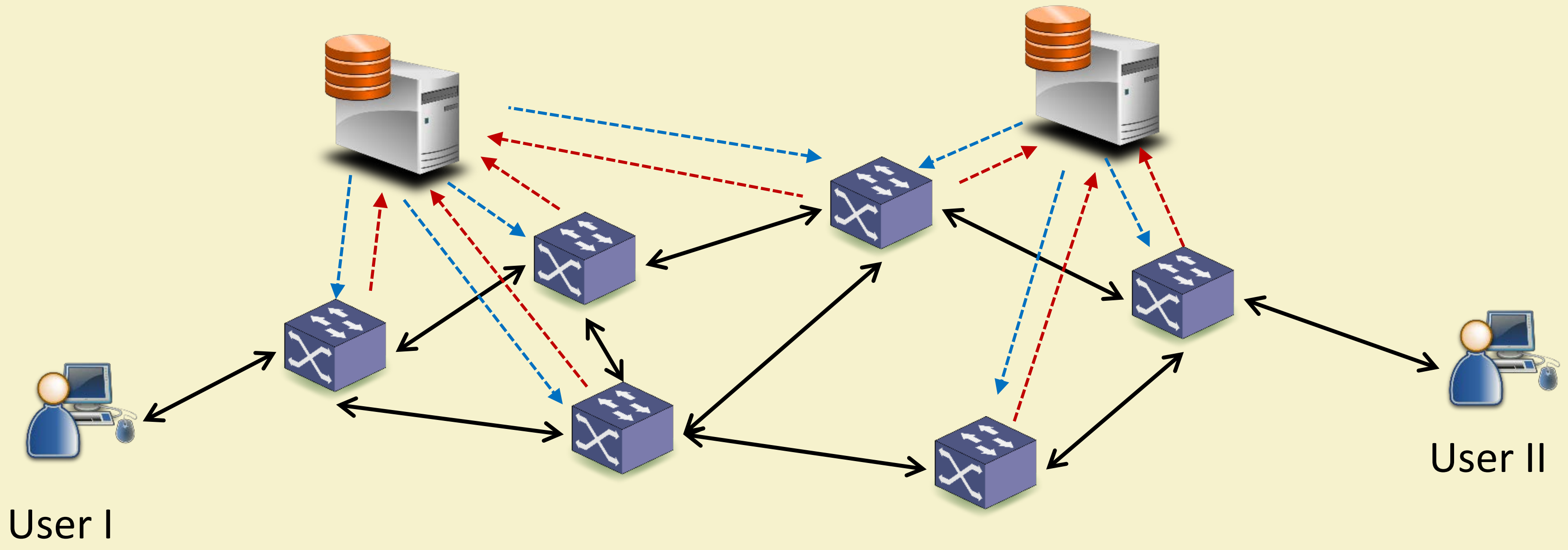
Hierarchical (tree) Architecture



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.

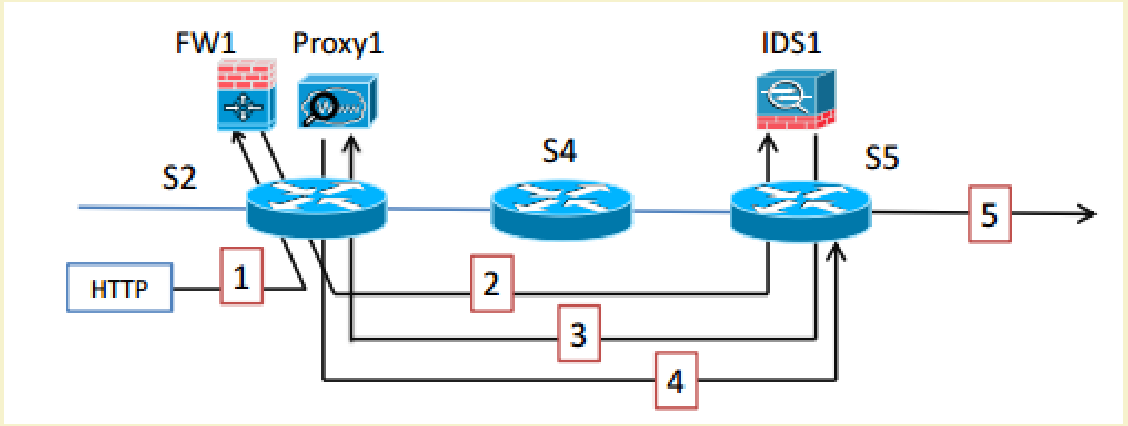
Backup Controller

- ✓ 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)

Security II



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

Source: SIMPLE-fying Middlebox Policy Enforcement Using SDN, SIGCOMM 2013

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

- 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.

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

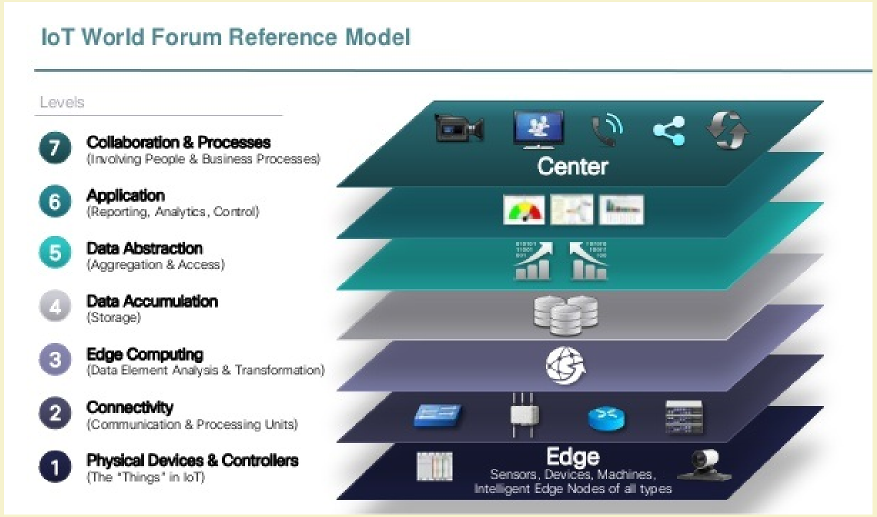
NPTEL

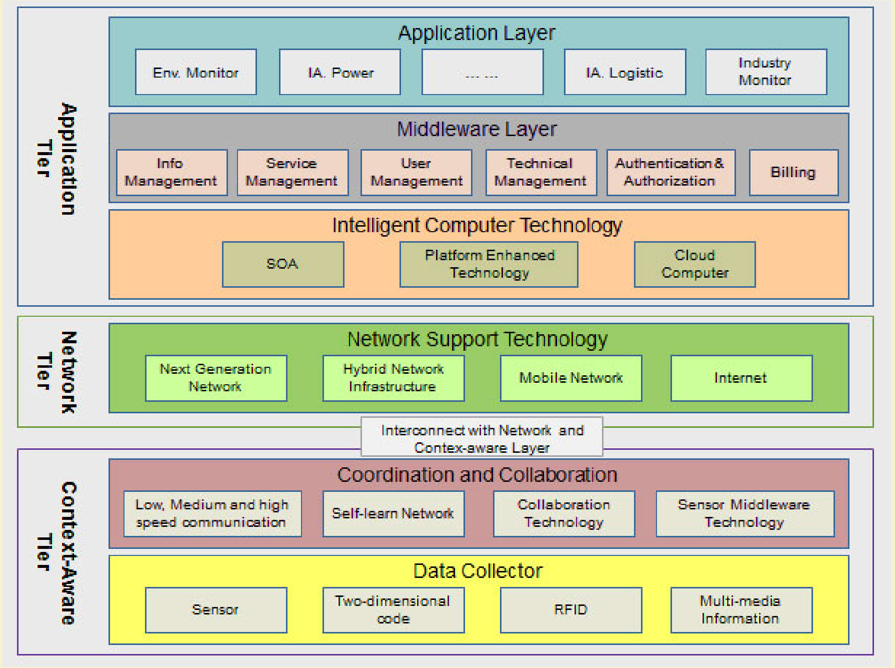
Software-Defined IoT Networking – Part I

Recent Advances of SDN in IoT

Dr. Sudip Misra
Associate Professor
Department of Computer Science and Engineering
IIT KHARAGPUR
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/

IoT Architecture


IoT World Forum Reference Model
Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines, Intelligent Edge Nodes of all types



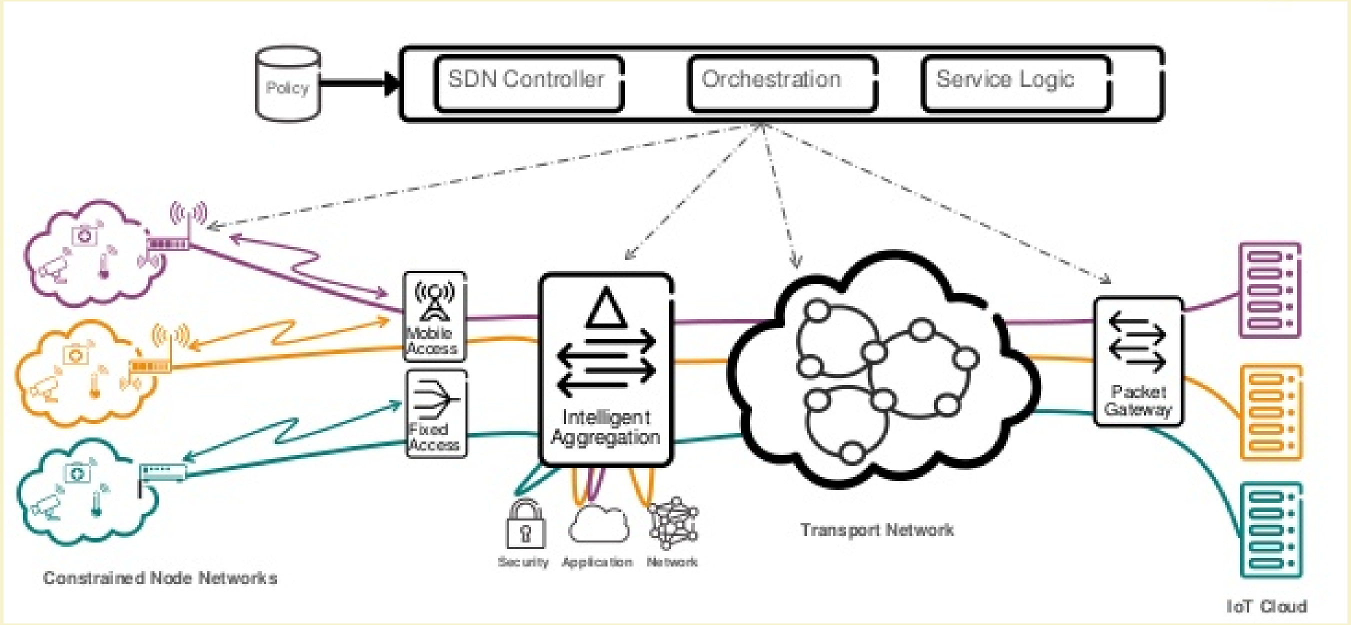
Source: https://image.slidesharecdn.com

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

Benefits of Integrating SDN in IoT

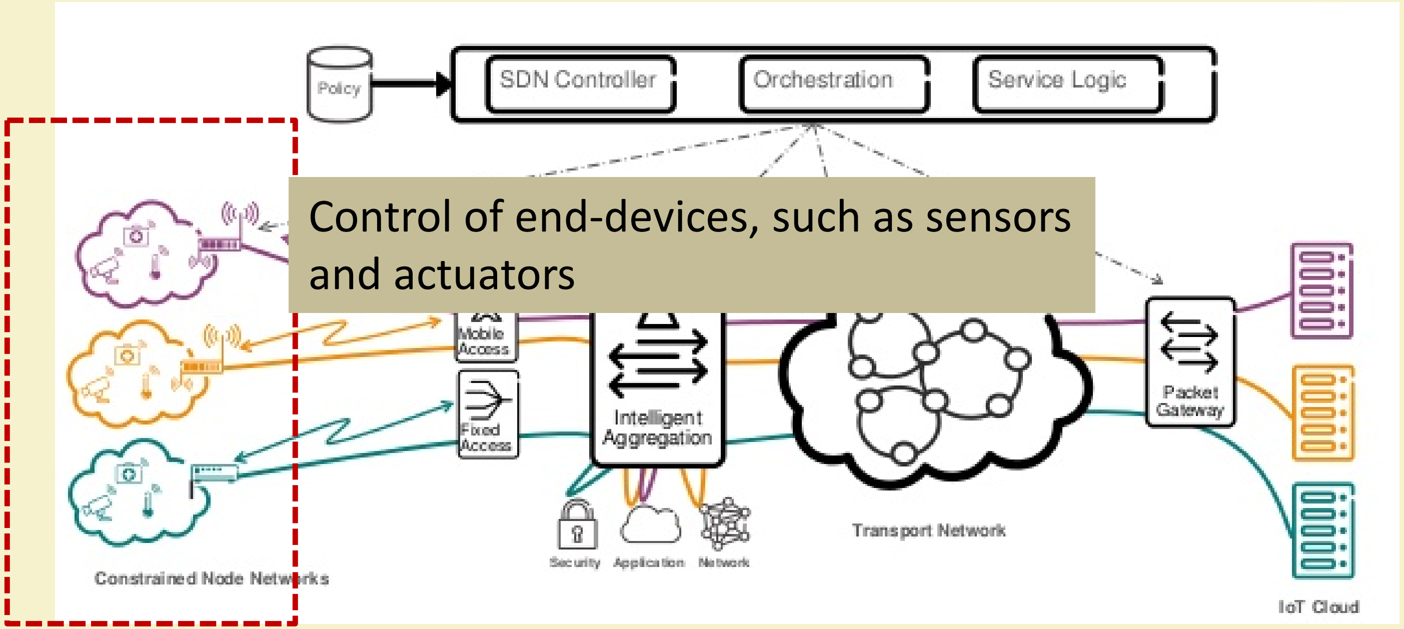
- ✓ Intelligent routing decisions can be deployed using SDN
- ✓ Simplification of information collection, analysis and decision making
- ✓ Visibility of network resources – network management is simplified based on user, device and application-specific requirements
- ✓ Intelligent traffic pattern analysis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

SDN for IoT I



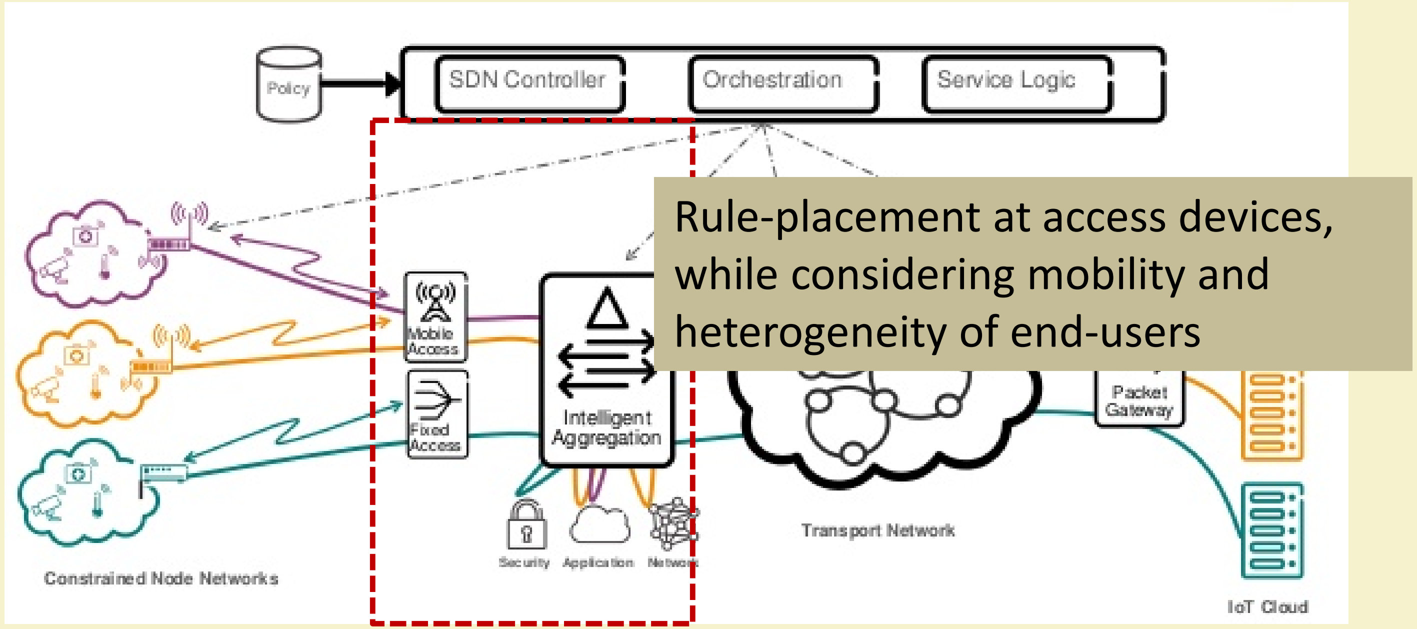
Source: https://image.slidesharecdn.com

SDN for IoT II



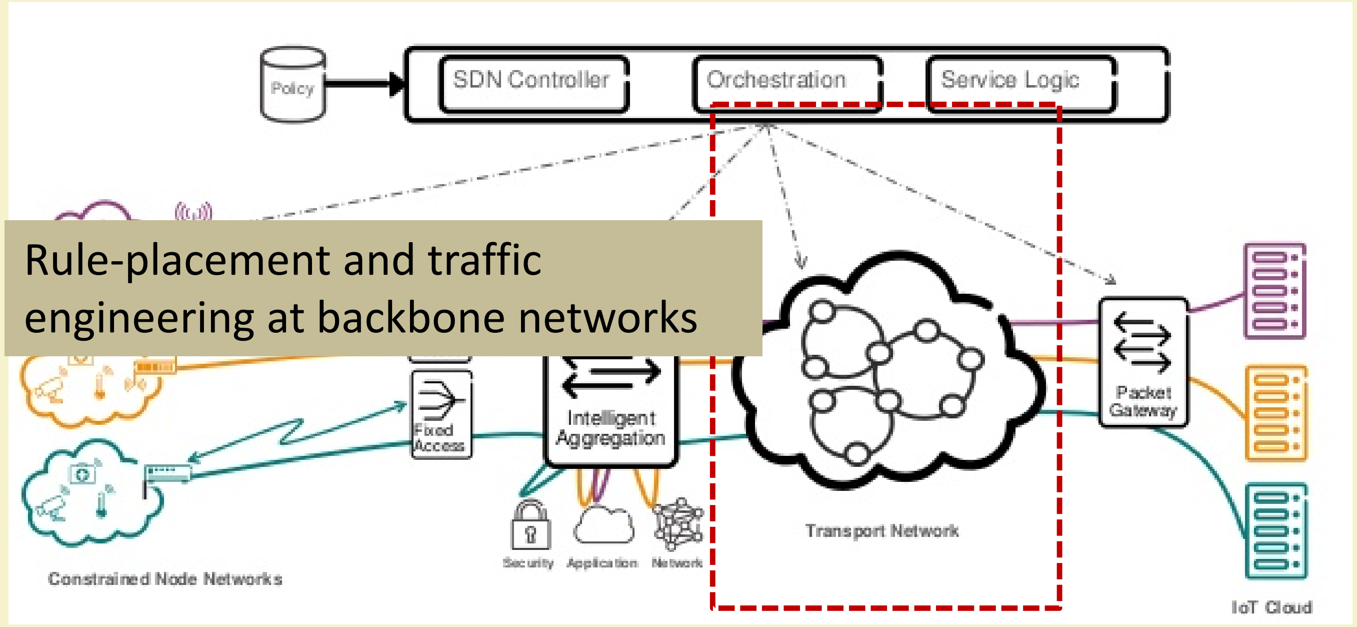
Control of end-devices, such as sensors and actuators

SDN for IoT III



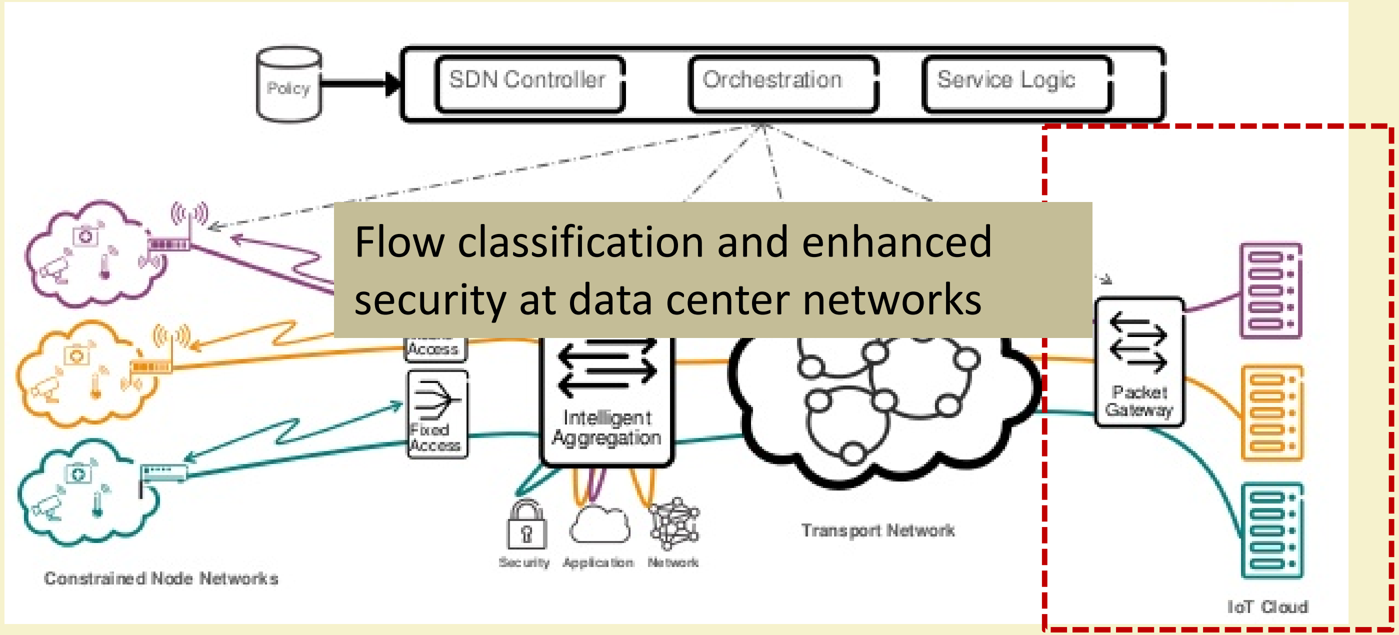
Rule-placement at access devices, while considering mobility and heterogeneity of end-users

SDN for IoT IV



Rule-placement and traffic engineering at backbone networks

SDN for IoT V



Flow classification and enhanced security at data center networks

Wireless Sensor Network I

- ✓ 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

Wireless Sensor Network II

- ✓ 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

Software-Defined WSN II

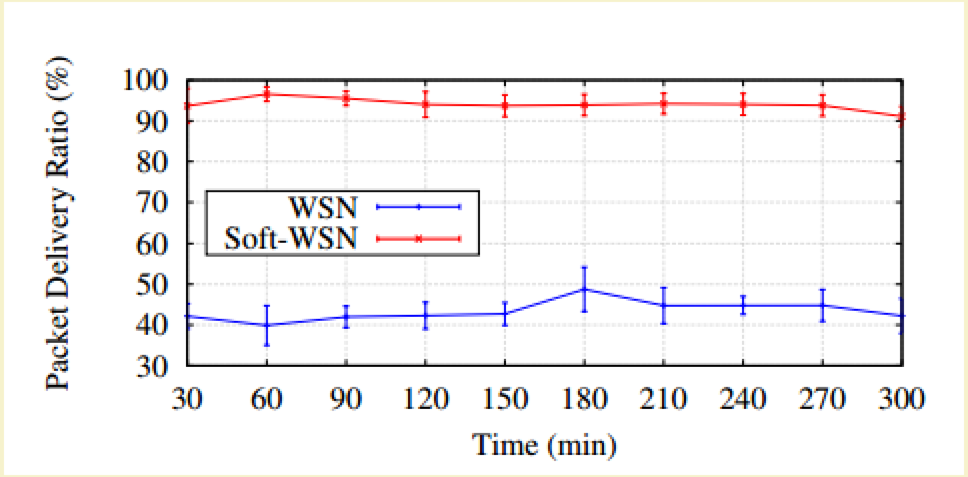
- ✓ Soft-WSN (Bera et al., IEEE SJ ’16)
  - Sensor Device Management
    - Sensor management
      - Multiple sensors can be implemented in a single sensor board
      - Sensors can be used depending on application-specific requirements
    - Delay management
      - Delay for sensing can be changed dynamically in real-time
    - Active-Sleep Management
      - States of active and sleep mode can be changed dynamically

Software-Defined WSN III

- ✓ 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

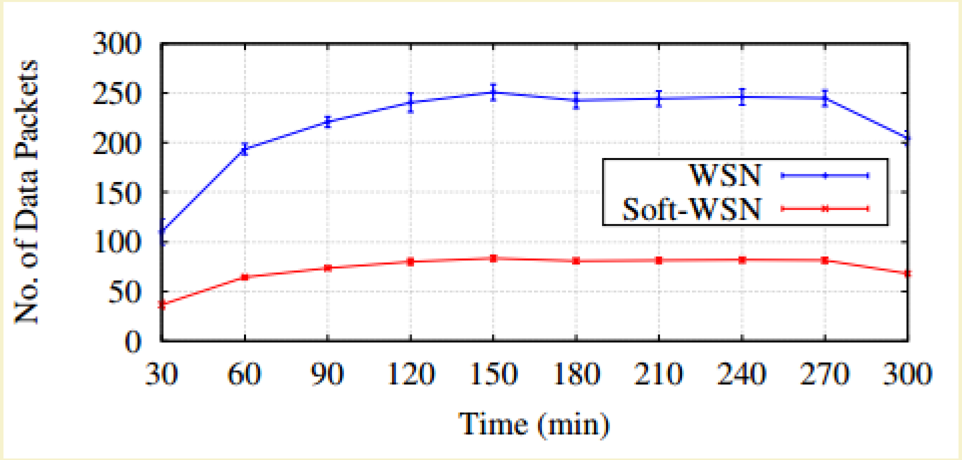
Experimental results show that network performance can be improved using software-defined WSN over traditional WSN

Soft-WSN: Result I



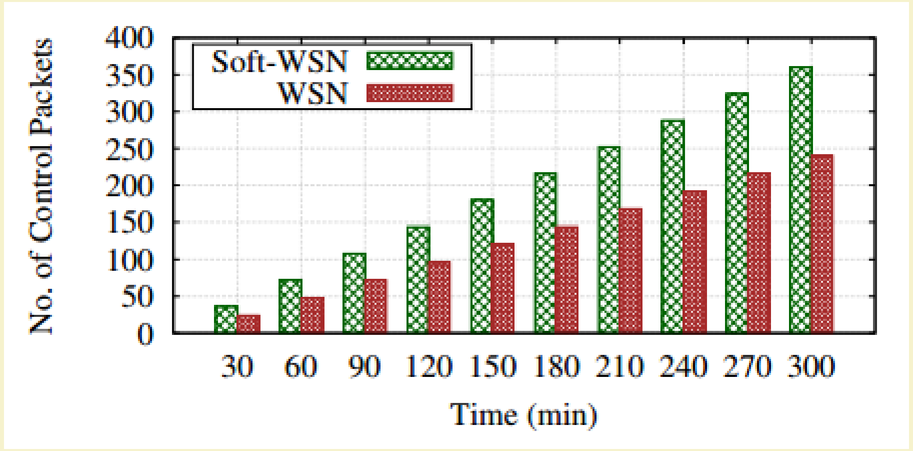
Packet delivery ratio in the network increases using Soft-WSN compared to the traditional WSN.

Soft-WSN: Result II



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

Soft-WSN: Result III



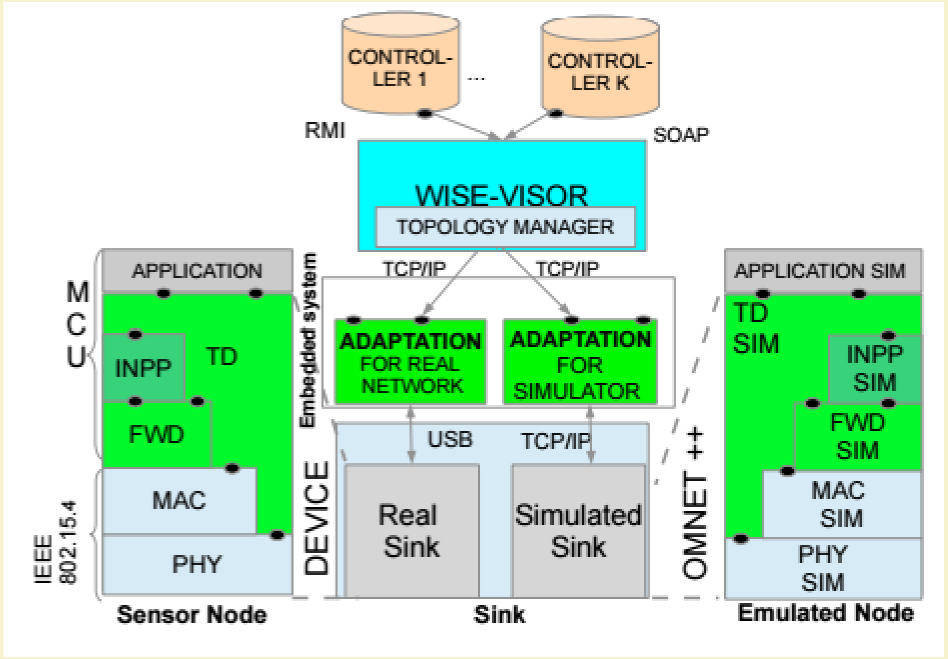
Number of control messages in the network is higher using Soft-WSN over the traditional WSN. This is due to the PACKET-IN message in the network. Each time a node receives a new packet, it asks the controller for getting adequate forwarding logic.

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



Source: Galluccio et al., IEEE INFOCOM '15

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

Thank You!!