IoTivity Core Framework: Features & Opportunities

Kishen Maloor

Intel Open Source Technology Center

OPEN INTERCONNECT CONSORTIUM℠

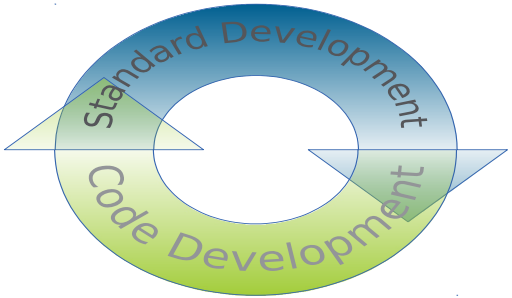
THE LINUX FOUNDATION

Outline

- What is IoTivity and why is it useful?
- IoTivity stack architecture
- IoTivity resource model and request-response flow
- Role in the IoT ecosystem
- Cross-platform support

What is IoTivity?

OPEN INTERCONNECT CONSORTIUM℠



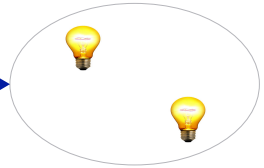
IoTivity

- Open source framework and SDK for building IoT applications
- Independently governed

Why is it useful?

Smartphone

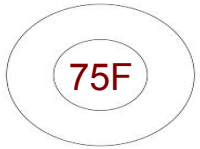
Turn Lights ON



Light bulbs with BLE radios

Notify Current Setting

Smart TV



Digital Thermostat

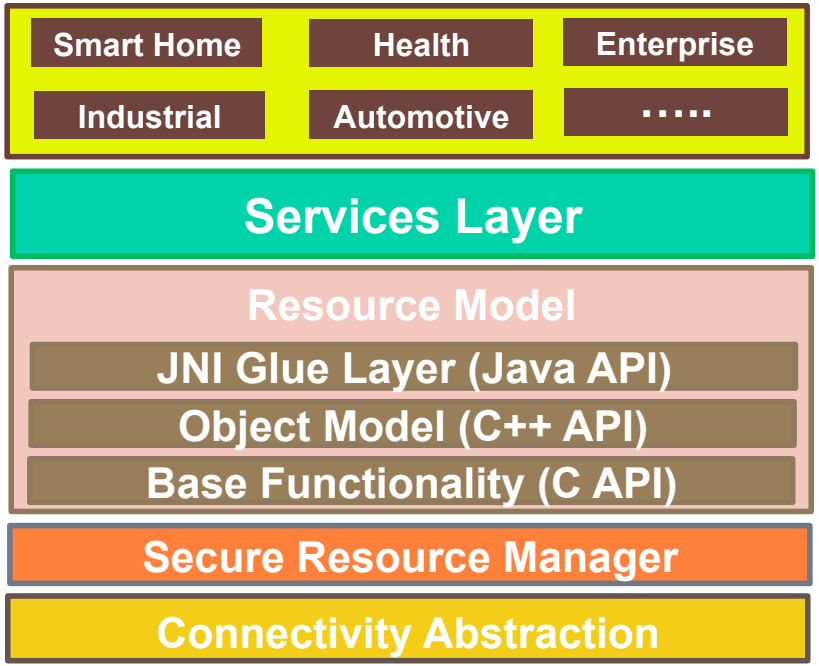
Tablet

Regulate Temperature

Why is it useful?

- Cross-platform support
- Uniform and easy-to-use APIs
- Based on open standards
- Support for multiple connectivity types
- Extensible to support proprietary protocols

IoTivity stack architecture



OIC protocol & connectivity types

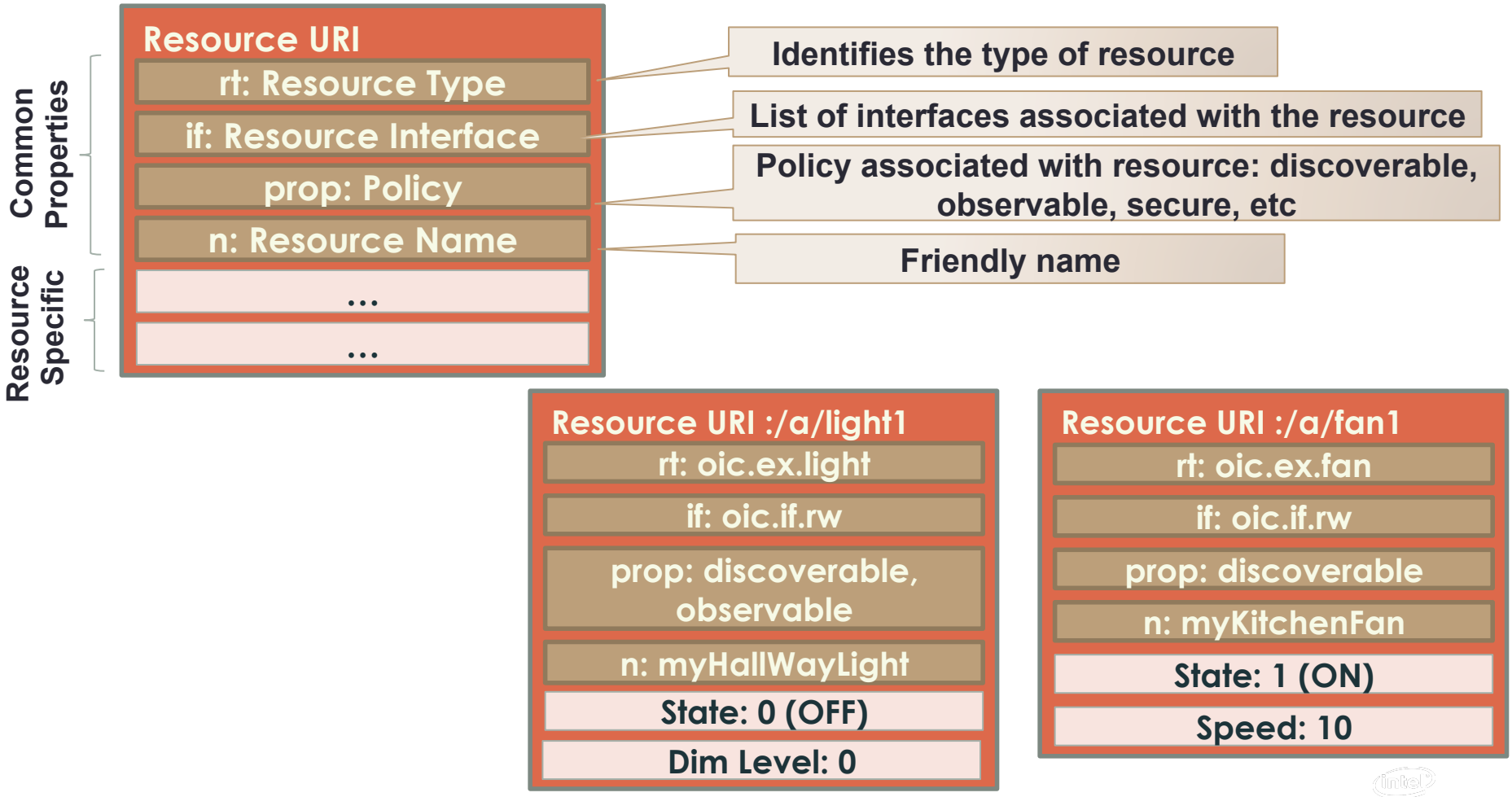
- Messaging protocol
  - Currently based on CoAP (RFC 7252)
- OIC payloads encoded using CBOR (RFC 7049)
- Adapter abstraction
  - Handle multiple connectivity types
    - Dual-stack IPv4 / IPv6
    - Bluetooth Low Energy using GATT
    - Bluetooth EDR using RFCOMM

IoTivity resource model

- RESTful design -> Things modeled as resources
- Server role: Exposes hosted resources
- Client role: Accesses resources on a server
- Intermediary role: Bridges messaging between client and server

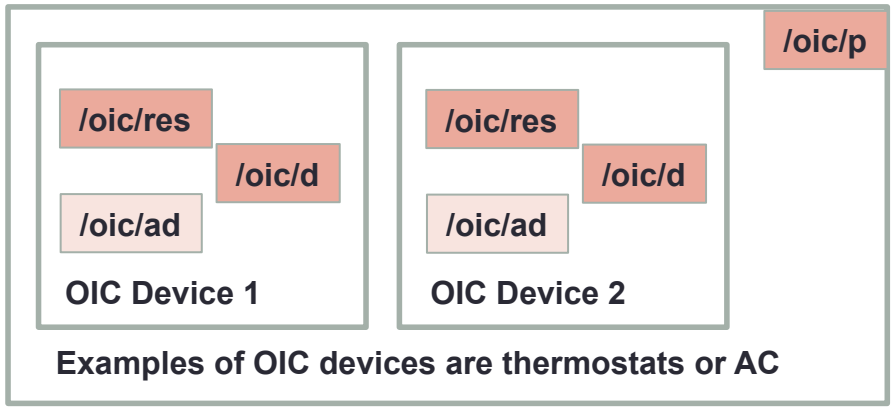
Model 1

Model 2


Common Properties
Resource Specific
Resource URI
rt: Resource Type
if: Resource Interface
prop: Policy
n: Resource Name
...
...
Identifies the type of resource
List of interfaces associated with the resource
Policy associated with resource: discoverable, observable, secure, etc
Friendly name
Resource URI :/a/light1
rt: oic.ex.light
if: oic.if.rw
prop: discoverable, observable
n: myHallWayLight
State: 0 (OFF)
Dim Level: 0
Resource URI :/a/fan1
rt: oic.ex.fan
if: oic.if.rw
prop: discoverable
n: myKitchenFan
State: 1 (ON)
Speed: 10

“Well-Known” resources

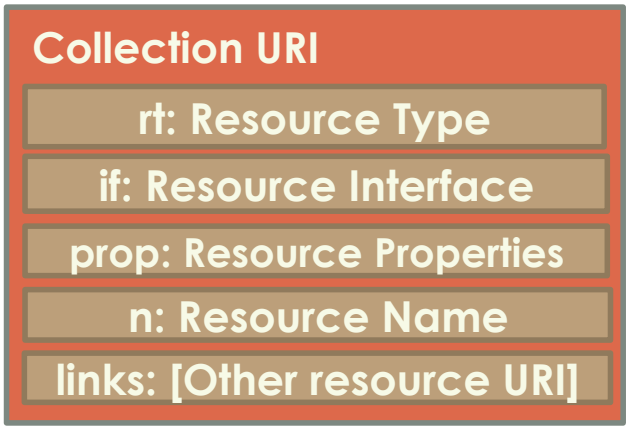
Functionality	Fixed URI
Discovery	/oic/res
Device	/oic/d
Platform	/oic/p
Presence	/oic/ad
Security	…

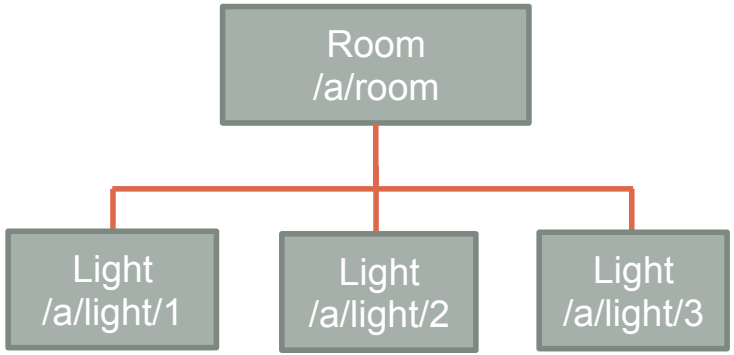


- Resources are associated with “Entity Handlers”
- Execute OIC methods on resources

Resource collections

- Links to other resources (RFC 5988)
- Express hierarchy, groups, indexes





IoTivity request-response flow

Client

Application

Resource Model

CoAP over UDP

L2 Connectivity + IP

URI: /a/light; rt = ‘oic.ex.light’,
if = ‘oic.ex.rw’,
prop = discoverable,
observable

Server

Application EH

Resource Model

CoAP over UDP

L2 Connectivity + IP

Resource discovery

Multicast GET coap://224.0.1.187:5683/oic/res

Unicast response

[URI: /a/light; rt = ‘oic.ex.light’, if = ‘oic.ex.rw’, prop = discoverable, observable]

Client
192.168.1.2

- Application
- Resource Model
- CoAP over UDP
- L2 Connectivity + IP

Server
192.168.1.1

- Application EH
- Resource Model
- CoAP over UDP
- L2 Connectivity + IP

IPv4	**224.0.1.187: 5683**
IPv6	**FF0X::FD: 5683**

GET operation

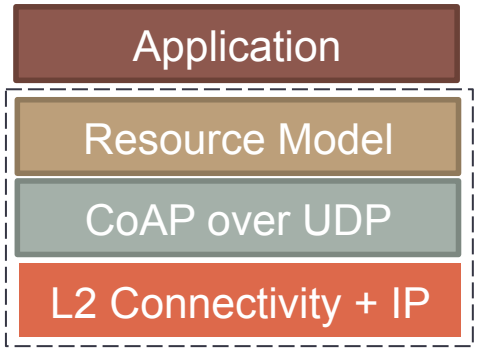
Unicast GET coap://192.168.1.1:9000/a/light

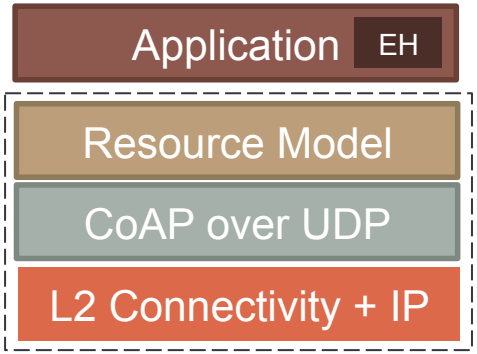
Unicast response

[URI: /a/light; state = 0, dim = 0]

Client
192.168.1.2

Server
192.168.1.1





PUT operation

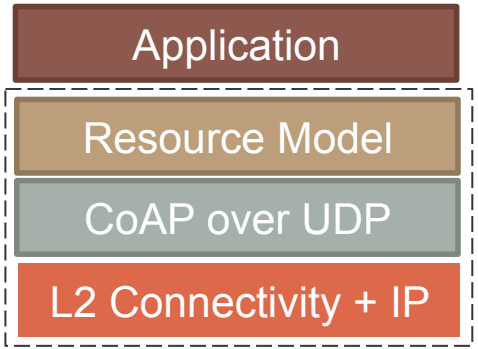
Unicast PUT coap://192.168.1.1:9000/a/light
PayLoad: [state=1;dim=50]

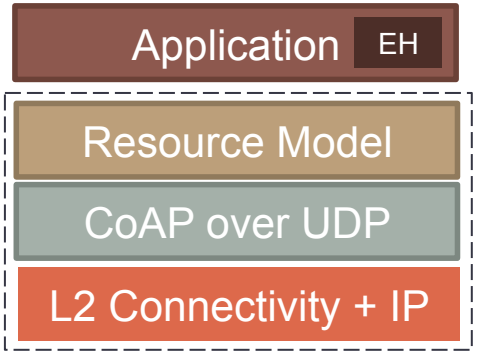
Unicast response

Status = Success

Client
192.168.1.2

Server
192.168.1.1





OBSERVE operation

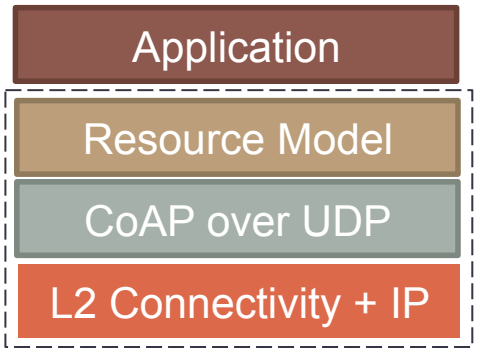
Unicast GET coap://192.168.1.1:9000/a/light; ObserveFlag = 1

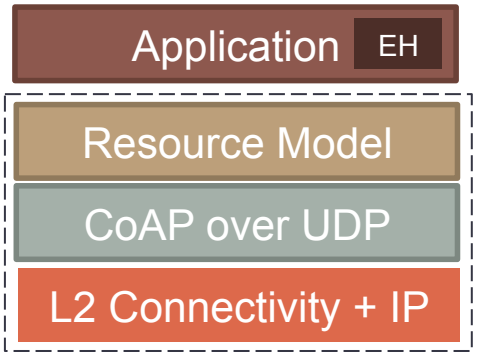
Unicast response

[URI: /a/light; state = 1, dim = 50]

Client
192.168.1.2

Server
192.168.1.1





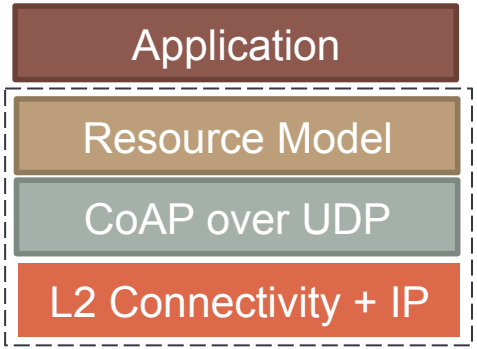
OBSERVE notification

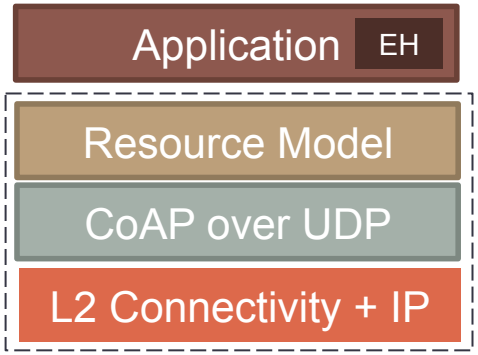
Notify Observers

[URI: /a/light; state = 0, dim = 0, sequence #: 1]

Client
192.168.1.2

Server
192.168.1.1





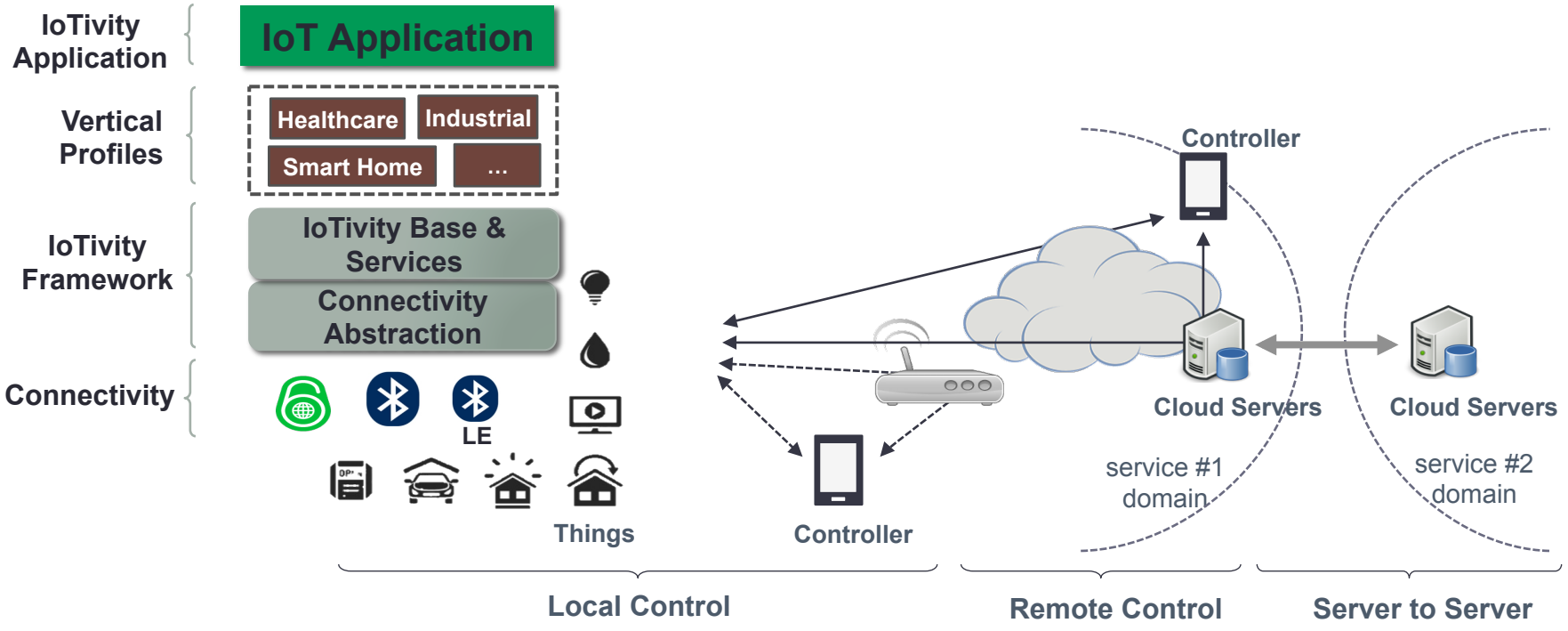
PRESENCE: “Active discovery”

- Servers can advertise themselves to clients
- Clients can request unicast or multicast notifications
  - Server coming online
  - Server going off-line
  - Changes to resources
- Clients may indicate interest in specific resource types

Notable IoTivity features

- Discovery
- Messaging and data model
- Message switching
- Remote access
- Services
  - Protocol plug-ins
  - Group management
- Security

Role in the IoT ecosystem



Cross-platform support

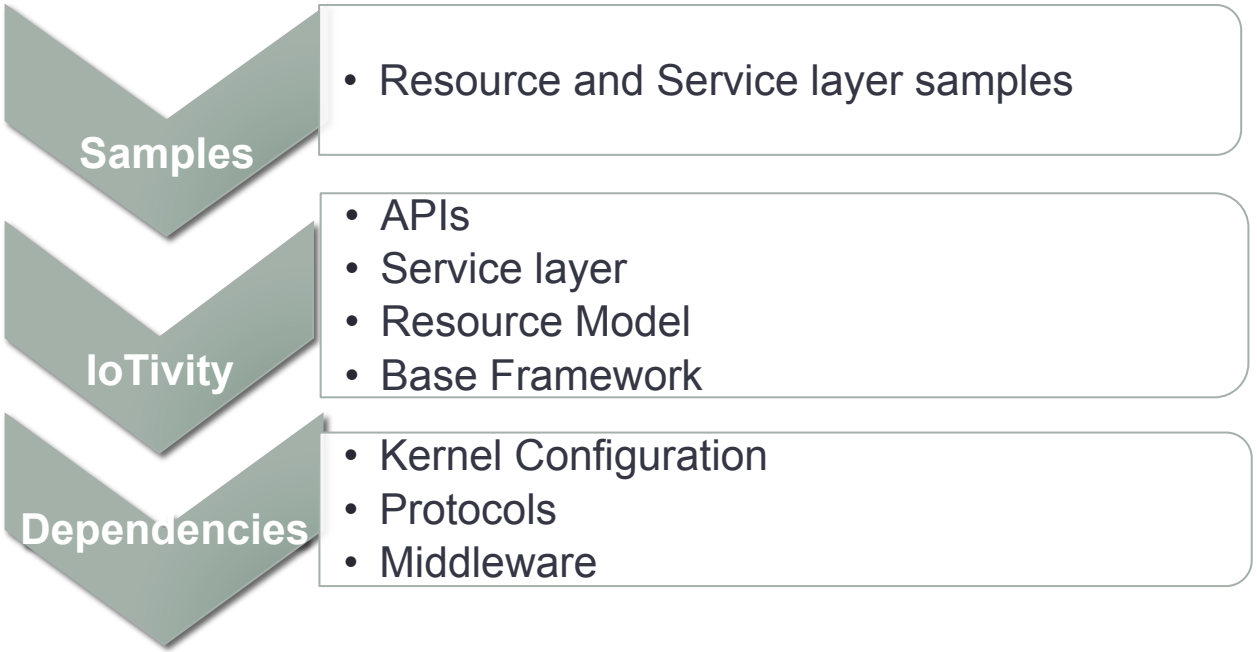
- Linux (Ubuntu 12.04)
- Arduino: Due, ATMega 2560
- Android
- Tizen

Embedded support: Yocto Project

- http://www.yoctoproject.org/
  - Hosted at the Linux Foundation
  - Create customized OS images for embedded targets
  - Ready-to-use BSPs for multiple platforms
  - Supports major CPU architectures
- Layers and recipes

meta-oic software layer for Yocto

- git://git.yoctoproject.org/meta-oic



Constrained peripherals

- Storage and memory constraints
- Lightweight IoTivity server stack
  - Base framework, resource model, messaging
- Work in progress…

How can you participate?

- Adopt IoTivity as the framework of choice for IoT projects
- Contribute to the IoTivity project: www.iotivity.org
- IoTivity mailing list: iotivity-dev@lists.iotivity.org

Thanks for your time!

Q&A