Lecture 1

Reference Architectures of IoT

IoT Reference Model

In the IoT domain, anyone can propose an own architecture and an own communication protocol, due to a wide variety of requirements to which each architecture should be compliant.

A common reference model for the IoT domain and the identification of reference architectures can help to a faster, more focused development and an exponential increase of IoT-related solutions.

The European Lighthouse Integrated Project has addressed for three years the Internet-of-Things Architecture (IoT-A) and created an architectural reference model together with the definition of an initial set of key building blocks. It wants to promote a common ground between architectures so that they can interoperate even a more levels. IoT-A has achieved that thanks to two steps:

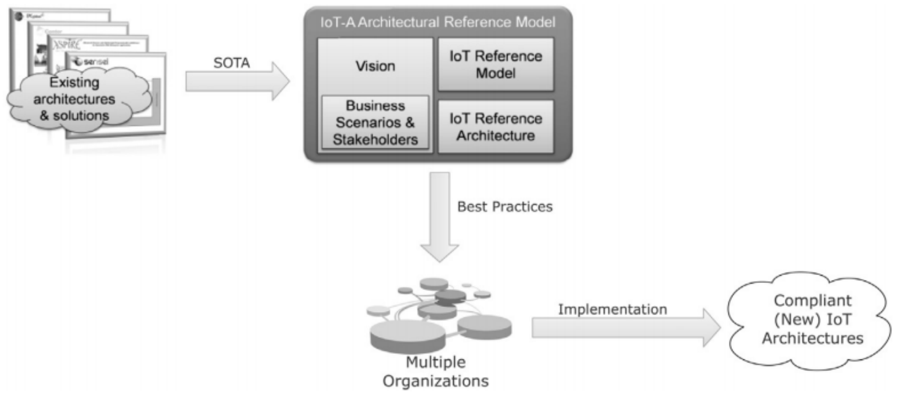
- Establishing a Reference Model
- Providing a Reference Architecture

IoT-A ARM

The IoT-A ARM (Architecture Reference Model) consists of four parts:

- **Vision**: summarizes the rationale for providing an architectural reference model for the IoT;
- **Business scenarios**: define requirements provided by stakeholders that are the drivers of the architecture. They allow the architecture to be validated;

- **The IoT Reference Model**, which provides the highest abstraction level for the definition of the IoT-A Architectural Reference Model. It promotes a common understanding of the IoT domain. It includes the description of domain, communication and information model;
- **The IoT Reference Architecture**, which is the reference for building compliant IoT architectures. It provides views and perspectives on different architectural aspects that are of concern to stakeholders of the IoT.

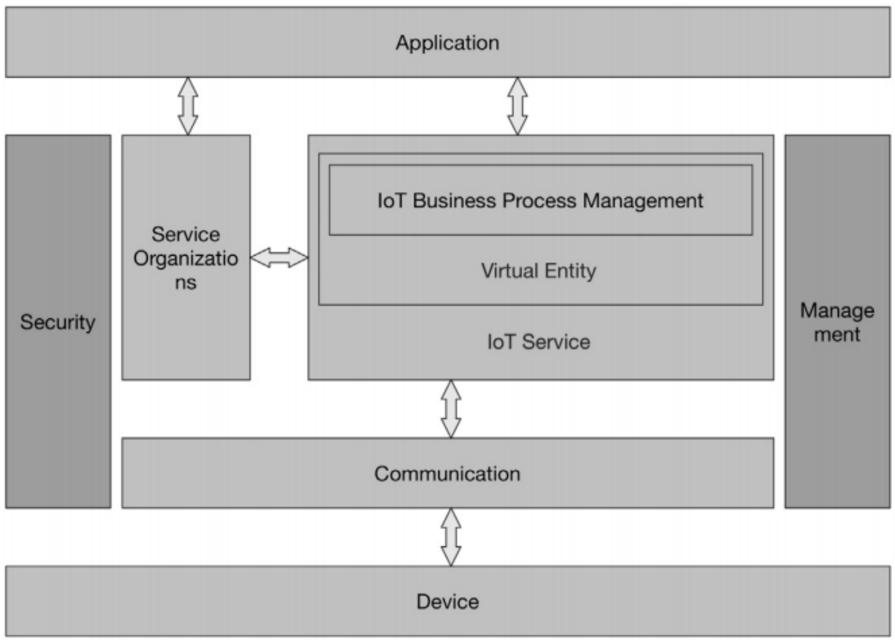

Existing architectures & solutions
SOTA
IoT-A Architectural Reference Model
Vision
IoT Reference Model
Business Scenarios & Stakeholders
IoT Reference Architecture
Best Practices
Multiple Organizations
Implementation
Compliant (New) IoT Architectures

Uses of IoT Reference Architecture

When it comes to product development and other activities, an architectural reference model is of fourfold use.

Cognitive Aid

- Firstly, it helps to guide discussions, since it provides a language everyone involved can use, and which is intimately linked to the architecture, the system, the usage domain
- Secondly, the high-level view provided in such a model is of high educational value, since it provides an abstract but also rich view of the domain. Such a



view can help people new to the field to ?find their way? and to understand the special features and intricacies of IoT.

- Thirdly, the IoT ARM can assist IoT project leaders in planning the work at hand and the teams needed.
- Fourthly, the IoT ARM helps to identify independent building blocks for IoT systems. This constitutes very valuable information when dealing with questions such as system modularity, processor architectures, third-vendor options, re-use of components already developed, etc.

Reference Model as a Common Ground

Establishing the common ground for the IoT encompasses defining IoT entities and describing their basic interactions and relationships with each other. The IoT ARM provides exactly such a common ground for the IoT field.

Generating Architectures

One of the main benefits is the use of the IoT ARM for generating compliant architectures for specific systems. This is done by providing best practices and guidance for translating the IoT ARM into concrete architectures. The benefit of this type of generation scheme for IoT architectures is not only a certain degree of automation in this process, and thus lower R&D efforts, but also that the decisions made follow a clear, documented pattern.

Identifying Differences in Derived Architectures

the IoT ARM defines a set of tactics and design choices for meeting qualitative system requirements. All of these facts can be used to predict whether two derived architectures will differ and where they will do so. The IoT ARM can also be used for reverse mapping. System architectures can be cast in the "IoT ARM? language and the resulting "translation? of the system architectures is then stripped of incompatible language and system partitions and mappings. The differences that remain are then true differences in architecture.

Achieving Interoperability

By comparing the design choices made when deriving two architectures, one can readily identify where in the architecture measures are necessary to achieve interoperability. Interoperability may be achieved a posteriori by integrating one IoT system as subsystem in another system, or by building a bridge through which key functionalities of the respective other IoT system can be used.

System Roadmaps and Product Life Cycles

the IoT ARM can be used to devise system roadmaps that lead to minimum changes between two product generations while still guaranteeing a noticeable enhancement in system capability and features. This approach also helps the designer to formulate clear and standardised, requirements-based rationales for the system roadmap chosen and the product life cycles that result from the system roadmap.

Benchmarking

While the reference model prescribed the language to be used in the systems/architectures to be assessed, the reference architecture stated the minimum (functional) requirements for the systems/ architectures. By standardising the description and also the ordering and delineation of system components and aspects, this approach also provided the benchmarking process with a high level of transparency and inherent comparability

Lecture 2

IoT Architectural Reference Model

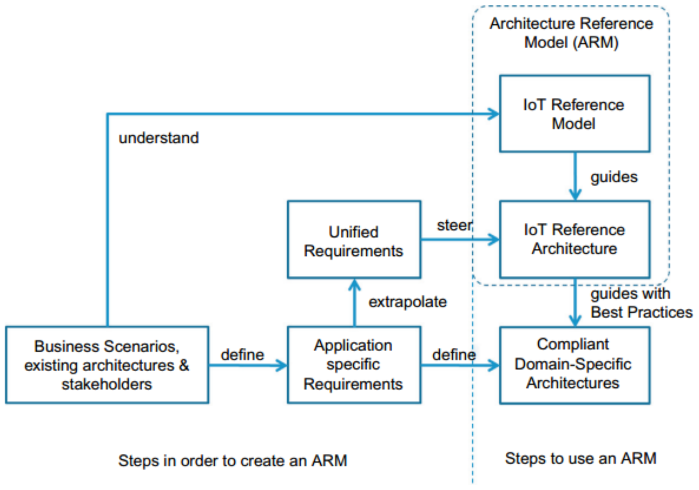
An ARM consists of two main parts: a **Reference model** and a **Reference Architecture**.

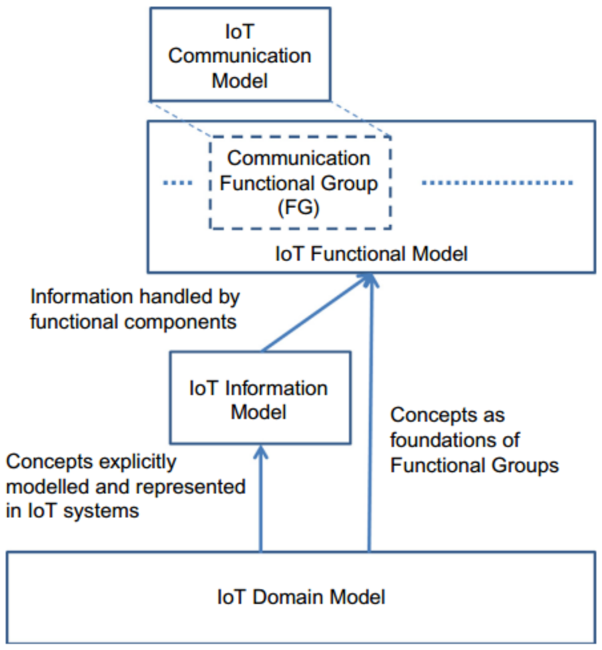
- A reference model describes the domain using a number of sub-models . The *domain model* of an architecture model captures the main concepts or entities in the domain in question, in this case IoT.
- When these common language references are established, the domain model adds descriptions about the relationship between the concepts. These concepts and relationships serve the basis for the development of an *information model* because a working system needs to capture and process information about its main entities and their interactions.
- A working system that captures and operates on the domain and information model contains concepts and entities of its own, and these need to be described in a separate model, the *functional model*.
- IoT system contain communicating entities, and therefore the corresponding *communication model* needs to capture the communication interactions of these entities.

Apart from the reference model, the other main component of an ARM is the Reference Architecture. A System Architecture is a communication tool for different stakeholders of the system.

- A Reference Architecture captures the essential parts of an architecture, such as design principles, guidelines, and required parts (such as entities), to monitor and interact with the physical world for the case of an IoT Reference Architecture.

- Concrete architectures are instantiations of rather abstract and high-level Reference Architectures.
- A concrete architecture can be further elaborated and mapped into real world components by designing, building, engineering, and testing the different components of the actual system.
- The whole process is iterative, which means that the actual deployed system in the field provides invaluable feedback with respect to the design and engineering choices, current constraints of the system, and potential future opportunities that are fed back to the concrete architectures. The general essentials out of multiple concrete architectures can then be aggregated, and contribute to the evolution of the Reference Architecture.


Architecture Reference Model (ARM)
IoT Reference Model
understand
guides
Unified Requirements
steer
IoT Reference Architecture
extrapolate
guides with Best Practices
Business Scenarios, existing architectures & stakeholders
define
Application specific Requirements
define
Compliant Domain-Specific Architectures
Steps in order to create an ARM
Steps to use an ARM


IoT Communication Model
Communication Functional Group (FG)
IoT Functional Model
Information handled by functional components
IoT Information Model
Concepts as foundations of Functional Groups
Concepts explicitly modelled and represented in IoT systems
IoT Domain Model

IoT domain model

A domain model defines the main concepts of a specific area of interest, in this case the IoT. These concepts are expected to remain unchanged over the course of time, even if the details of an ARM may undergo continuous transformation or evolution over time. The domain model captures the basic attributes of the main concepts and the relationship between these concepts. A domain model also serves as a tool for human communication between people working in the domain in question and between people who work across different domains.

The main concepts of ARM domain model are:

- *User and Physical Entity*: As interaction with the physical world is the key for the IoT. The first most fundamental interaction is between a human or an application with the physical world object or place. The physical interaction is the result of the intention of the human to achieve a certain goal (e.g. park the car).A Physical Entity, as the model shows, can potentially contain other physical entities (e.g building).

- *Virtual Entity*: A Physical Entity is represented in the digital world as a Virtual Entity. A Virtual Entity can be a database entry, a geographical model (mainly for places), an image or avatar, or any other *Digital Artifact*. Each Virtual Entity also has a unique identifier for making it addressable among other Digital Artifacts.

- A Virtual Entity representation contains several attributes that correspond to the Physical Entity current state (e.g. the parking spot availability). The Virtual Entity representation and the Physical Entity actual state should be synchronized whenever a User operates on one or the other, if of course that is physically possible. There are cases that state synchronization can occur only one way.

- *Physical Artifact*:In order to monitor and interact with the Physical Entities through their corresponding virtual entities, the Physical Entities or their surrounding environment needs to be instrumented with certain kinds of Devices, or certain Devices need to be embedded/attached to the environment. The Devices are physical artifacts with which the physical and virtual worlds interact. For the IoT Domain Model, three kinds of Device types are the most important - Sensors, actuators and Tags.

- *Resources*:Resources are software components that provide data for, or are endpoints for, controlling Physical Entities. Resources can be of two types, on-Device resources and Network Resources.

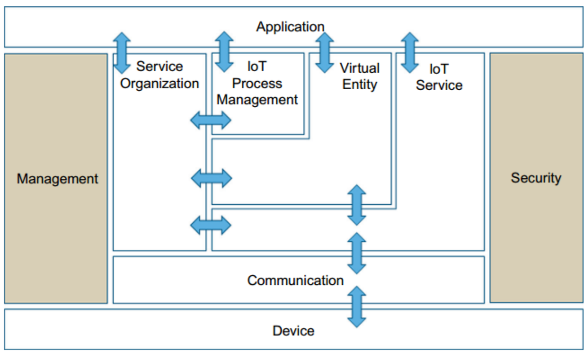
- An on-Device Resource is typically hosted on the Device itself and provides information, or is the control point for the Physical Entities that the Device itself is attached to. The Network Resources are software components hosted somewhere in the network or cloud.

- A Virtual Entity is associated with potentially several Resources that provide information or control of the Physical Entity represented by this Virtual Entity.
- Resources expose (monitor or control) functionality as Services with open and standardized interfaces, thus abstracting the potentially low-level implementation details of the resources.
- *Services* are therefore digital artifacts with which Users interact with Physical Entities through the Virtual Entities. Therefore, the Virtual Entities that are associated with Physical Entities instrumented with Devices that expose Resources are also associated with the corresponding resource Services. These associations are important to be maintained for lookup or discovery by the interested Users.
- IoT Services can be classified into three main classes according to their level of abstraction: (i)Resource-Level Services (ii) Virtual Entity-Level Services (iii) Integrated Services .

Information Model

Information is defined as the enrichment of data (raw values without relevant or usable context) with the right context, so that queries about who, what, where, and when can be answered.

- On a high-level, the IoT Information Model maintains the necessary information about Virtual Entities and their properties or attributes.
- These properties/attributes can be static or dynamic and enter into the system in various forms, e.g. by manual data entry or reading a sensor attached to



the Virtual Entity. Virtual Entity attributes can also be digital synchronized copies of the state of an actuator.

- In the presentation of the high-level IoT information model, we omit the attributes that are not updated by an IoT Device (sensor, tag) or the attributes that do not affect any IoT Device (actuator, tag), with the exception of essential attributes such as names and identifiers
- A Virtual Entity object contains simple attributes/properties: (a) entityType to denote the type of entity, such as a human, car, or room (the entity type can be a reference to concepts of a domain ontology, e.g. a car ontology); (b) a unique identifier; and (c) zero or more complex attributes of the class Attributes.

Functional model

In the IoT-ARM project, *Functional Decomposition* (FD) refers to the process by which the different *Functional Components* (FC) that make up the IoT ARM are identified and related to one another. The main purpose of Functional Decomposition is, on the one hand, to break up the complexity of a system compliant to the IoT ARM in smaller and more manageable parts, and to understand and illustrate their relationship on the other hand.

The most important functional group are:

- *Device functional group:* The Device FG contains all the possible functionality hosted by the physical Devices that are used for instrumenting the Physical Entities.
- *Communication functional group* The Communication FG abstracts all the possible communication mechanisms used by the relevant Devices in an actual system in order to transfer information to the digital world components or other Devices.
- *IoT Service functional group*: The IoT Service FG corresponds mainly to the Service class from the IoT Domain Model, and contains single IoT Services exposed by Resources hosted on Devices or in the Network (e.g. processing or storage Resources).
- *Virtual Entity functional group* The Virtual Entity FG corresponds to the Virtual Entity class in the IoT Domain Model, and contains the necessary functionality to manage associations between Virtual Entities with themselves as well as associations between Virtual Entities and related IoT Services, i.e. the Association objects for the IoT Information Model.
- *IoT Service Organization functional group* The purpose of the IoT Service Organization FG is to host all functional components that support the composition and orchestration of IoT and Virtual Entity services.
- *IoT Process Management functional group* The IoT Process Management FG is a collection of functionalities that allows smooth integration of IoT-related services (IoT Services, Virtual Entity Services, Composed Services) with the Enterprise (Business) Processes

- *Management functional group* The Management FG includes the necessary functions for enabling fault and performance monitoring of the system, configuration for enabling the system to be flexible to changing User demands, and accounting for enabling subsequent billing for the usage of the system.
- *Security functional group* The SecurityFG contains components for Authentication of Users (Applications,Humans), Authorization of access to Services by Users, secure communication (ensuring integrity and confidentiality of messages) between entities of the system such as Devices, Services, Applications, and last but not least,assurance of privacy of sensitive information relating to Human Users

Communication Model

The communication model for an IoT Reference Model consists of the identification of the endpoints of interactions, traffic patterns (e.g. unicast vs. multicast), and general properties of the underlying technologies used for enabling such interactions.

- The potential communicating endpoints or entities are the Users, Resources, and Devices from the IoT Domain Model. Users include Human Users and Active Digital Artifacts.
- *User-Service / Service-Service Interactions*: , the IoT Domain Model entities involved in this interaction are mainly two: User and Service. If a Service is constrained, or if it needs to provide access to constrained Users, it must be accessible with constrained protocols (e.g., 6LoWPAN, UDP, CoAP, etc.). Finally, when the two elements belong to different sub-networks, gateway(s) and/or proxy(ies) must be deployed for ensuring successful end-to-end communication.

- *Service / Resource / Device Interactions*: The complexity of this interaction is due to variety of different properties that a Device can have; in particular, a Device can be as simple and limited as a Tag and as complex and powerful as a server.

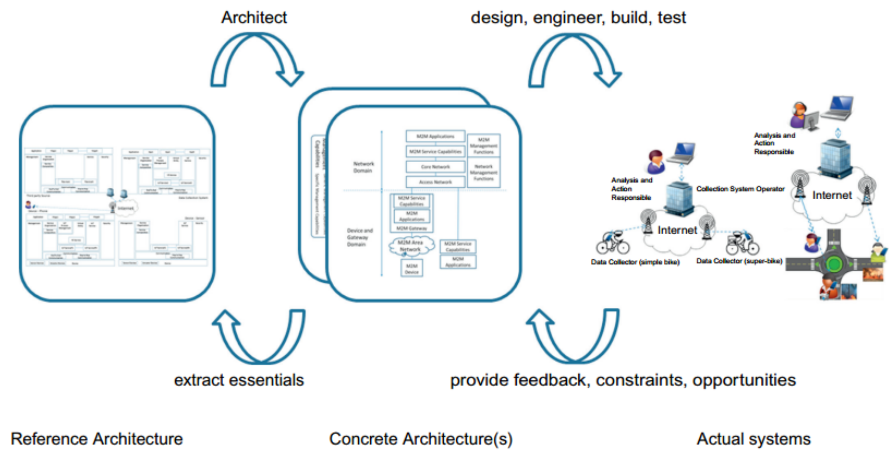
Safety, privacy, trust, security model

The fact that Human Users are part of the system that could potentially harm humans if malfunctioning, or expose private information, motivates the Safety and Privacy needs for the IoT Reference Model and Architecture.

- **Safety**: The IoT Reference Model can only provide IoT-related guidelines for ensuring a safe system to the extent possible and controllable by a system designer. A system designer of such critical systems typically follows an iterative process with two steps: (a) identification of hazards followed by, (b) the mitigation plan.
- **Privacy**: he IoT-A Privacy Model depends on the following functional components: Identity Management, Authentication, Authorization, and Trust & Reputation. The Trust and Reputation functional component maintain the static or dynamic trust relationships between interacting entities.
- **Security** The Security Model for IoT consists of communication security that focuses mostly on the confidentiality and integrity protection of interacting entities and functional components such as Identity Management, Authentication, Authorization, and Trust & Reputation.

IoT Reference Architecture

The Reference Architecture is a starting point for generating concrete architectures and actual systems. A Reference Architecture, serves as a guide for one or more


Architect
design, engineer, build, test
extract essentials
provide feedback, constraints, opportunities
Reference Architecture
Concrete Architecture(s)
Actual systems
Internet
Collection System Operator
Analysis and Action Responsible
Data Collector (simple bike)
Data Collector (super-bike)

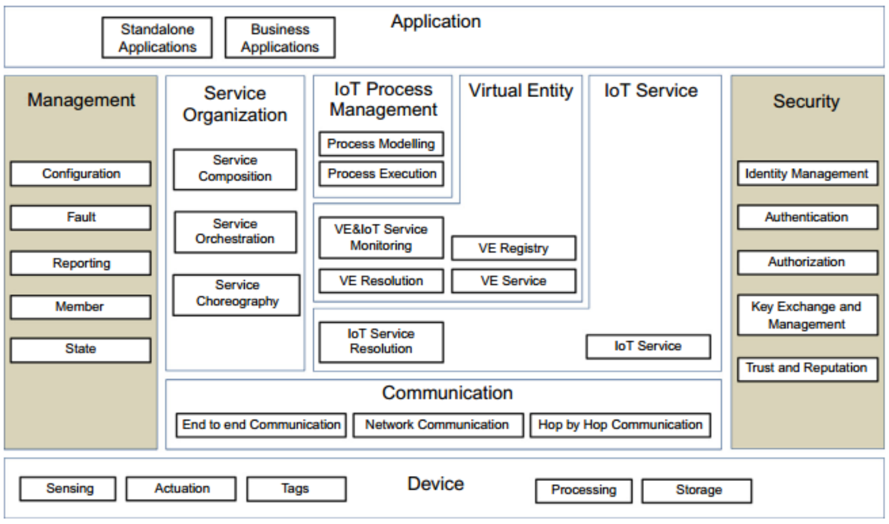
concrete system architects.

The Reference Architecture is presented as set of architectural views. Views are useful for reducing the complexity of the Reference Architecture blueprints by addressing groups of concerns one group at a time. In order to address the concerns of mainly the concrete IoT architect, and secondly the concerns of most of the above stakeholders following views are chosen.

- **Functional View**: Description of what the system does, and its main functions.
- **Information View**: Description of the data and information that the system handles.
- **Deployment and Operational View**: Description of the main real world components of the system such as devices, network routers, servers, etc.

Functional View

It consists of the Functional Groups (FGs) presented earlier in the IoT Functional Model, each of which includes a set of Functional Components (FCs).



Information View

The information view consists of (a) the description of the information handled in the IoT System, and (b) the way this information is handled in the system;

The pieces of information handled by an IoT system complying to an ARM such as the IoT-A are the following:

- Virtual Entity context information, i.e. the attributes (simple or complex) as represented by parts of the IoT Information model (attributes that have values and metadata such as the temperature of a room).
- IoT Service Descriptions, which contain associated Resources, interface descriptions, etc. IoT Service output itself is another important part of information generated by an IoT system.
- Resource Descriptions, which contain the type of resource (e.g. sensor), identity, associated Services, and Devices. Device Descriptions such as device capabilities (e.g. sensors, radios)
- IoT Business Process Model, which describes the steps of a business process

utilizing other IoT-related services (IoT, Virtual Entity, Composed Services).

- Management information such as state information from operational FCs used for fault/performance purposes, configuration snapshots, reports, membership information, etc.

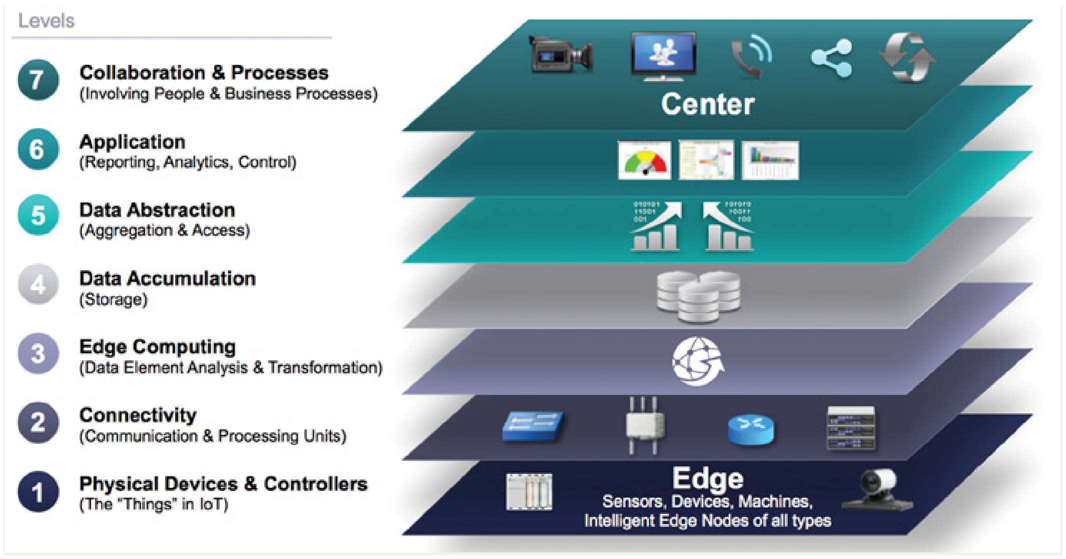
The presentation of information handling in an IoT system assumes that FCs exchange and process information. The exchange of information between FCs follows the interaction patterns below

- **Push:** An FC A pushes the information to another FC B provided that the contact information of the component B is already configured in component A, and component B listens for such information pushes.
- **Request/Response**: An FC A sends a request to another FC B and receives a response from B after A serves the request.
- **Subscribe/Notify**: Multiple subscriber components (SA, SB) can subscribe for information to a component C, and C will notify the relevant subscribers when the requested information is ready. This is typically an asynchronous information request after which each subscriber can perform other tasks.
- **Publish/Subscribe**: In the Publish/Subscribe (also known as a Pub/Sub pattern), there is a third component called the broker B, which mediates subscription and publications between subscribers (information consumers) and publishers (or information producers)

Lecture 3

IoTWF Architecture

In 2014 the IoTWF architectural committee (led by Cisco, IBM, Rockwell Automation, and others) published a seven-layer IoT architectural reference model. While various IoT reference models exist, the one put forth by the IoT World Forum offers a clean, simplified perspective on IoT and includes edge computing, data storage, and access. It provides a succinct way of visualizing IoT from a technical perspective. Each of the seven layers is broken down into specific functions, and security encompasses the entire 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

As shown in figure the IoT Reference Model defines a set of levels with control flowing from the center (this could be either a cloud service or a dedicated data center), to the edge, which includes sensors, devices, machines, and other types of intelligent end nodes. In general, data travels up the stack, originating from the edge, and goes northbound to the center. Using this reference model, we are able to achieve the following:

- Decompose the IoT problem into smaller parts
- Identify different technologies at each layer and how they relate to one another
- Define a system in which different parts can be provided by different vendors
- Have a process of defining interfaces that leads to interoperability
- Define a tiered security model that is enforced at the transition points between levels

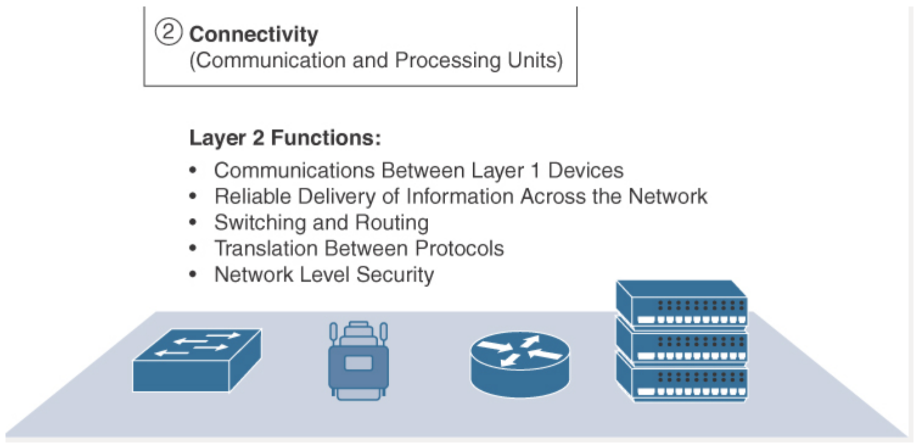
Layer 1: Physical Devices and Controllers Layer

The first layer of the IoT Reference Model is the physical devices and controllers layer. This layer is home to the “things? in the Internet of Things, including the various endpoint devices and sensors that send and receive information. The size of these “things? can range from almost microscopic sensors to giant machines in a factory. Their primary function is generating data and being capable of being queried and/or controlled over a network.

Layer 2: Connectivity Layer

In the second layer of the IoT Reference Model, the focus is on connectivity. The most important function of this IoT layer is the reliable and timely transmission of data. More specifically, this includes transmissions between Layer 1 devices and the network and between the network and information processing that occurs at Layer 3 (the edge computing layer).

As you may notice, the connectivity layer encompasses all networking elements of IoT and doesn?t really distinguish between the last-mile network (the network between the sensor/endpoint and the IoT gateway, discussed later in this chapter), gateway, and backhaul networks.


② Connectivity
(Communication and Processing Units)
Layer 2 Functions:
• Communications Between Layer 1 Devices
• Reliable Delivery of Information Across the Network
• Switching and Routing
• Translation Between Protocols
• Network Level Security

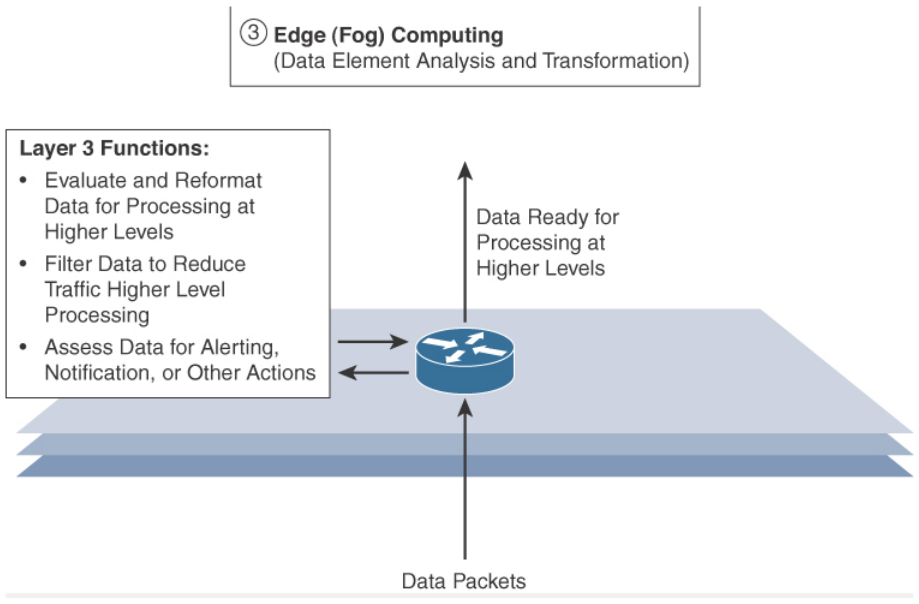
Layer 3: Edge Computing Layer

Edge computing is the role of Layer 3. Edge computing is often referred to as the “fog? layer and is discussed in the section “Fog Computing,? later in this chapter. At this layer, the emphasis is on data reduction and converting network data flows into information that is ready for storage and processing by higher layers. One of the basic principles of this reference model is that information processing is initiated as early and as close to the edge of the network as possible. Figure highlights the functions handled by Layer 3 of the IoT Reference Model.

Layer 3: Edge Computing Layer

Edge computing is the role of Layer 3. Edge computing is often referred to as the “fog? layer and is discussed in the section “Fog Computing,? later in this chapter. At this layer, the emphasis is on data reduction and converting network data flows into information that is ready for storage and processing by higher layers. One of the basic principles of this reference model is that information processing is initiated as early and as close to the edge of the network as possible. Figure highlights the functions handled by Layer 3 of the IoT Reference Model.

Another important function that occurs at Layer 3 is the evaluation of data to


③ Edge (Fog) Computing
(Data Element Analysis and Transformation)
Layer 3 Functions:
• Evaluate and Reformat Data for Processing at Higher Levels
• Filter Data to Reduce Traffic Higher Level Processing
• Assess Data for Alerting, Notification, or Other Actions
Data Ready for Processing at Higher Levels
Data Packets

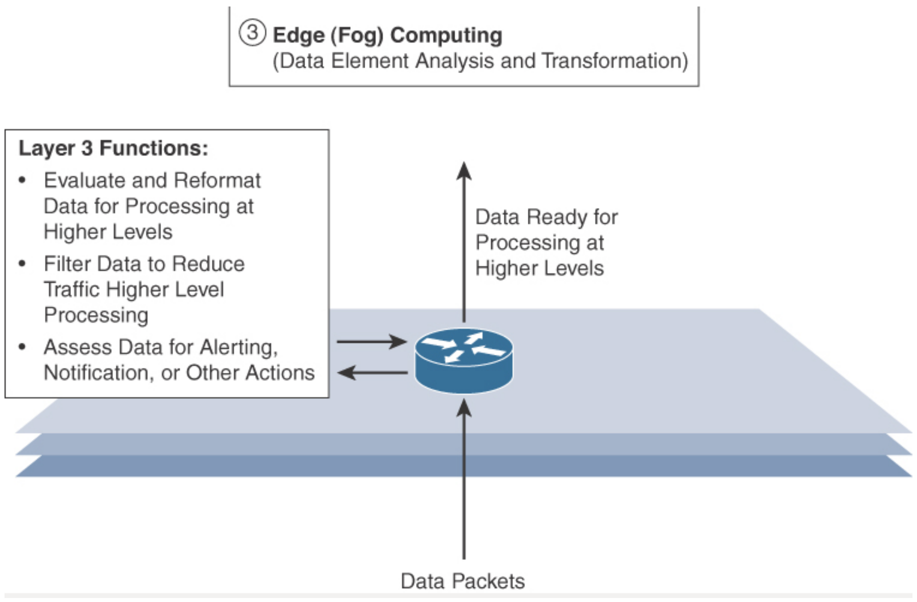
see if it can be filtered or aggregated before being sent to a higher layer. This also allows for data to be reformatted or decoded, making additional processing by other systems easier. Thus, a critical function is assessing the data to see if predefined thresholds are crossed and any action or alerts need to be sent.

Upper Layers: Layers 4-7

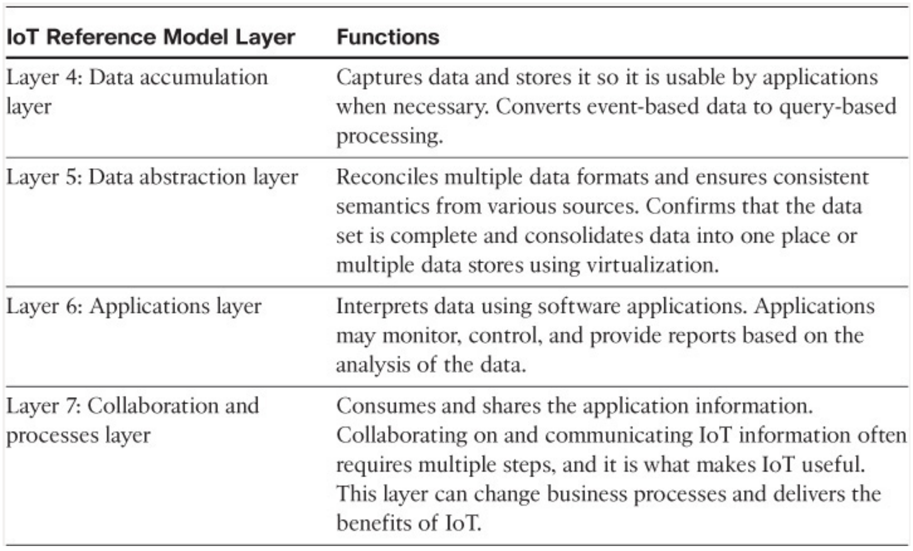
The upper layers deal with handling and processing the IoT data generated by the bottom layer. For the sake of completeness, Layers 4?7 of the IoT Reference Model are summarized

IT and OT Responsibilities in the IoT Reference Model

IT and OT Responsibilities in the IoT Reference Model An interesting aspect of visualizing an IoT architecture this way is that you can start to organize responsibilities along IT and OT lines. Figure 2-5 illustrates a natural demarcation point between IT and OT in the IoT Reference Model framework.


③ Edge (Fog) Computing
(Data Element Analysis and Transformation)
Layer 3 Functions:
• Evaluate and Reformat Data for Processing at Higher Levels
• Filter Data to Reduce Traffic Higher Level Processing
• Assess Data for Alerting, Notification, or Other Actions
Data Ready for Processing at Higher Levels
Data Packets

As demonstrated in Figure , IoT systems have to cross several boundaries beyond just the functional layers. The bottom of the stack is generally in the domain of OT. For an industry like oil and gas, this includes sensors and devices connected to pipelines, oil rigs, refinery machinery, and so on. The top of the stack is in the IT area and includes things like the servers, databases, and applications, all of which run on a part of the network controlled by IT. In the past, OT and IT have generally been very independent and had little need to even talk to each other. IoT is changing that paradigm. At the bottom, in the OT layers, the devices generate real-time data at their own rate?sometimes vast amounts on a daily basis. Not only does this result in a huge amount of data transiting the IoT network, but the sheer volume of data suggests that applications at the top layer will be able to ingest that much data at the rate required. To meet this requirement, data has to be buffered or stored at certain points within the IoT stack. Layering data management in this way throughout the stack helps the top four layers handle data at their own speed. As a result, the real-time ?data in motion? close to the edge has to be organized and stored so that it becomes ?data at rest? for the applications in the IT tiers.



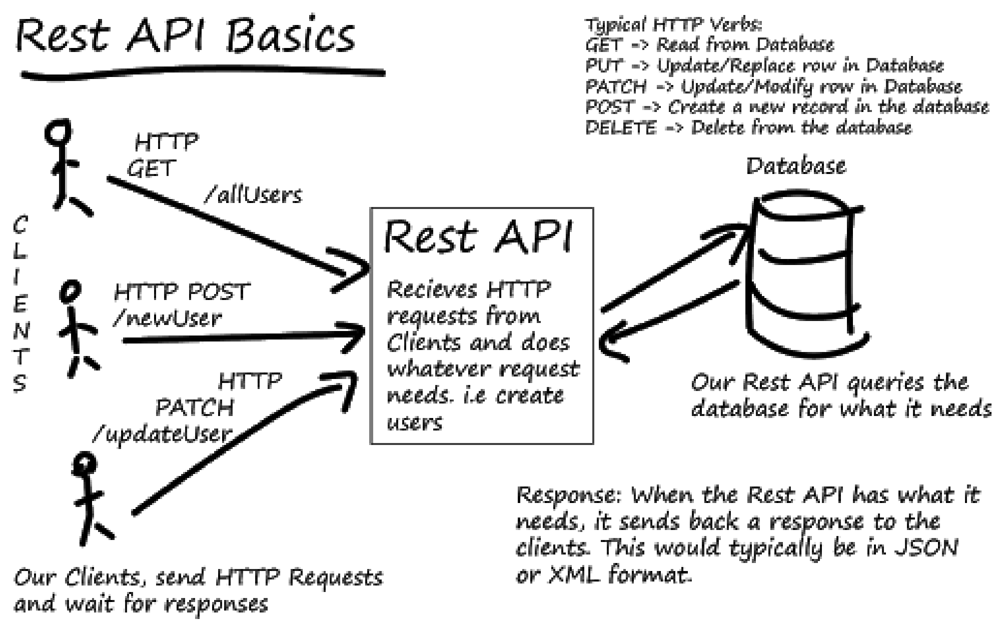
IoT Reference Model Layer	Functions
Layer 4: Data accumulation layer	Captures data and stores it so it is usable by applications when necessary. Converts event-based data to query-based processing.
Layer 5: Data abstraction layer	Reconciles multiple data formats and ensures consistent semantics from various sources. Confirms that the data set is complete and consolidates data into one place or multiple data stores using virtualization.
Layer 6: Applications layer	Interprets data using software applications. Applications may monitor, control, and provide reports based on the analysis of the data.
Layer 7: Collaboration and processes layer	Consumes and shares the application information. Collaborating on and communicating IoT information often requires multiple steps, and it is what makes IoT useful. This layer can change business processes and delivers the benefits of IoT.

The IT and OT organizations need to work together for overall data management.

Lecture 4

REST Architectures

One promising approach that is being brought to the IoT is the idea that it should be built in a similar way to the Internet. There are several reasons to use a web-based approach in the IoT. The web has been around for decades and lots of experience has been gained. Since its public release in 1991, the World Wide Web has dramatically evolved and has become an infrastructure upon which to store documents, resources, and to build distributed applications. The most important aspects of the introduction of the web were the referencing of resources through uniform resource identifiers (URIs) and the introduction of the Hypertext Transfer Protocol (HTTP) as the application-layer protocol for hypermedia systems. Along with these two major pillars, other essential standards and technologies have been developed, such as the Hypertext Markup Language (HTML) for web documents, web browsers, and web servers. As the web became more and more popular, browsers integrated dynamic behavior through Javascript and Cascading Stylesheets (CSS). After all these years, HTTP is by far the most common application-layer protocol and software libraries implementing web protocols are available (web servers, HTTP clients and so on) for any programming language. Ways of building web applications are widely known and used: adopting similar approaches for the IoT could therefore take advantage of the expertise of existing developers. Moreover, the web has proved to scale extremely well: this is extremely important for the IoT, where billions of connected devices are expected to operate. The IoT can greatly benefit from all the experience gained in the development of the web and thus the use of a similar architecture would seem to be a wise design choice.


Rest API Basics
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database
Database
HTTP GET /allUsers
CLIENTS
HTTP POST /newUser
HTTP PATCH /updateUser
Rest API
Recieves HTTP requests from Clients and does whatever request needs. i.e create users
Our Rest API queries the database for what it needs
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.
Our Clients, send HTTP Requests and wait for responses

REST: The Web as a Platform

The web was born to be an easy to use, distributed, and loosely coupled (see below) system for sharing documents. The architecture of the web is simple enough to make it easy to build applications and manage content. The web is based on a small set of principles, yet it has proved to scale and evolve wonderfully. Thanks to these principles, the web has evolved to become a platform for building distributed systems using HTTP. REpresentational State Transfer (REST) is the architectural style behind the web. Defined in 2000 in Roy Fielding?s PhD thesis [29], REST defines a set of rules and principles that all the elements of the architecture must conform to in order to build web applications that scale well, in terms of:

- scalability (number of interacting clients)
- robustness (long-term evolution of systems).

Loose coupling means that the endpoints should contain as little information about each other as needed to work. All necessary missing information should be collected while interacting. The client must know very few things a-priori. The server will drive the client and pass in the information required to progress and to perform the intended operations. The more a client knows about the server, the more closely it depends on the server implementation. This is a weakness for an application because any change on the server must be matched by a change in the client, which would otherwise just break. In a highly dynamic, evolving, and gigantic environment such as the IoT, design principles that lead to create robust applications must be adopted.

Resource-oriented Architectures

REST is based on the concept of a resource. A resource can be defined as any relevant entity in an application?s domain that is exposed on the network. A webpage, a video, and an order on an e-commerce website can all be considered web resources. A resource is anything with which a user interacts while progressing toward some goal. Anything can be mapped as a resource, as long as it is meaningful for the application. Resources are characterized by some data, such as the title of the page or the items in an order.

An alternative to a resource-oriented architecture (ROA) is a service-oriented architecture (SOA). SOAs have been around for many years and have become a reference for many legacy businessoriented systems. A SOA refers to an architecture where two endpoints communicate through a pre-defined set of messaging contracts. A client starts interacting with a server by retrieving the list of available services and how these can be mapped to HTTP messages, in a Web Service Definition Language (WSDL) document. In essence, the WSDL maps a message to a method call on the server. Remote method calls are contained in a SOAP (an XML specification) included in the body of messages. The presence of a WSDL document is needed to add semantics to messages. However, this is a weakness: if a server changes its services, a client needs to get access to the new WSDL or its functionalities are invalidated. In a ROA, on the other hand, there is no endpoint exposing services; there are only resources that can be manipulated. This is critical for the robustness of the client application.

Lecture 5

Resource Identification and Representation

The principle of separation of concerns is a fundamental of the REST architecture. According to this principle, each element of a system is responsible for a specific concern. Well-separated concerns allow for modularity, reusability, and independent evolution of the system?s elements. A REST architecture builds on:

- clients (or user agents, such as browsers), which are the application interface and initiate the interactions
- servers (origin servers) host resources and serve client requests.

Intermediaries act as clients and servers at the same time. Forward proxies (known to clients) are ?exit points? for a request. Reverse proxies appear as origin servers to a client, but actually relay requests.

A REST architecture is characterized by uniform interfaces: all connectors within the system must conform to this interface?s constraints. Collectively, REST defines the following principles:

- identification of resources
- manipulation of resources through representations
- self-descriptive messages
- hypermedia as the engine of application states

An application that follows the above principles is termed RESTful.

Representation of Resources

Resources are never exchanged directly by endpoints. Instead, representations of resources are exchanged between endpoints. A representation is a view of the state of the resource at a given time. This view can be encoded in one or more transferable formats, such as XHTML, Atom, XML, JSON, plain text, comma-separated values, MP3, or JPEG. Typically, the type of representation is specified in one header of the message containing the resource; for example, HTTP defines the Content-Type header. For the sake of compactness, from now on we will refer to the representation of a resource simply as a resource. Resources are exchanged back and forth between clients and servers. In order to be exchanged, resources must be serialized/deserialized properly at each endpoint, as shown in

The same resource can have many different representations (1:N relationship): the state of the same sensor can be described using JSON, XML, or any other suitable format.

Resource Identifierstitle

In order to ensure that an application is handling the correct resource, a mechanism to identify a resource univocally in the network is necessary. uniform resource identifiers (URIs), defined in RFC 3986, serve this specific need.

A URI identifies a resource univocally. A URI can be used to address a resource, so that it can be located, retrieved, and manipulated. There is a 1:N relationship between a resource and URIs: a resource can be mapped to multiple URIs, but a URI points exactly to one resource.

URIs can be of two kinds:

- a uniform resource name (URN) specifies the name of a resource (e.g., urn:ietf:rfc:2616);

- a uniform resource locator (URL) specifies how to locate the resource, (e.g., http://example.com/books/123)

All URIs take the following form: scheme:scheme-specificpart. The scheme part defines how the rest of the URI is to be

interpreted ? it typically serves as an indication of the communication protocol that should be used to target the resource. For instance, URNs use the urn scheme, while web resources use the http scheme. URLs include all the information needed to successfully address the resource.

The optional [username:password] part specifies credentials to use

for authenticated access to the resource. The host and optional port include networking information needed to reach to the resource. The host can be be either an IP address or a fully qualified domain name, which must be resolved using the DNS system. The path provides information to locate the resource inside the host. The query contains matching information to filter out the result. Finally, the fragment can be used to identify a specific portion of the resource. URIs should be opaque and not expose any specific notion of the format used to represent the targeted resource. For example, http://example.com/people/123 is a good URI, while http://example.com/people/123.xml and http://example.com/people/123.json are not.

Statelessness

An important principle of REST is statelessness. Statelessness implies that no state information must be kept on the client and server sides, thus avoiding the need to use cookies or to introduce the concept of sessions, which demand a stricter coupling between the endpoints. All requests must therefore be stateless. In order to preserve statelessness, each message must be self-descriptive. This means that all requests must contain all the information to understand the request so that servers can process them without context (about the state of the client). There is no state information maintained between clients and servers: the state of the client is contained in the request, so the server is relieved from the burden of keeping track of the client state.

Applications as Finite-state Machines

RESTful applications make forward progress by transitioning from one state to another, just like a finite-state machine (FSM), as shown in Figure 3.4.

Due to the loose coupling between clients and servers, the states of the FSM and the transitions between them are not known in advance. A state is reached when the server transfers a representation of the resource as a consequence of a client request. The next possible transitions are discovered when the application reaches a new state (gradual

reveal). Resource representations that embed links are called hypermedia. These links represent the possible transitions to the next states. In essence, the state of the a resource identified by a URI is contained in the data section of the resource representation and the transition to the next states are contained in the links.

Hypermedia as the Engine of Application State

The final principle of REST is hypermedia as the engine of application state?, or HATEOAS for short. HATEOAS states that resource representation should include all the information to drive the client

to perform the next requests to progress in the application. By doing so, a client just needs to follow the instructions that the server transmits in order to reach its goal. This guarantees that, should the server change its implementation and introduce new functionality (states and links in the FSM), the client would be unaffected by these changes and could continue to operate.

In summary, RESTful applications progress according to the following steps:

1. The client starts from an entry URI or a bookmark.
2. The response to a GET request includes a hypermedia representation.
3. The representation contains links that define the possible transitions to the next states of the FSM.
4. The client selects a link to follow and issues the next request; that

   is, it triggers a transition to the next state.
5. The client can also go back.

An important contribution of REST is the fact that it allows the web to be modeled as an FSM. The web is a globally distributed application, with web browsers as the clients and millions of servers that can serve requests. In the web, resources are documents (such as HTML pages). HTML includes links that can point to other documents (inside ¡a¿ tags) and is therefore a hypermedia format. The application is entered through a first URI (entered in the address bar). By clicking on a link, a new state is reached (the new document).

Lecture 6

DESIGN CHALLENGES IN IOT

This section discusses major design challenges and potential threats caused due to those challenges.

Heterogeneity and Interoperability

The supporting backend logic of many IoT solutions are custom designed with a particular solution in mind. However due to version upgrades and addition of new vendor products, the backend logic has to cater to a multi vendor or multi version deployment of IoT devices. This can be achieved using an API translator. The API translation component can effectively expose a consistent set of APIs and message formats in one direction while using adapter sub-components to manage interoperability between heterogeneous devices.This component thus has to be reachable from the field area devices and must therefore necessarily expose a public IP. By bombarding this component with bogus requests, it could be possible to induce the backend system into dedicated precious bandwidth towards executing bogus workloads, thus resulting in a service outage leading to DOS attacks. Spoofing is possible in this heterogeneous environment where a malicious node might impersonate another node possibly a router to route data through itself and get unauthorized access to sensor data.In many IoT applications the protocol suits verify the field devices. Due to lack of mutual authentication, the schemes are susceptible against MITM

Connectivity

IoT applications typically require two forms of connectivity, and either of these have their own set of challenges. The first form is at a physical level, where the sender and receiver need to communicate using the same PHY and MAC. Unfortunately peripheral IOT devices tend to communicate over low power radio standards like Bluetooth, ZigBee, Z-Wave, NFC etc The information from these peripheral devices have to be bridged to an IP network over traditional Ethernet networks.The bridging devices act as proxies for the peripheral devices which may be a cause for MITM.

The second form of connectivity is in terms of service connectivity. Even if packets from a physical device can reach the backend, it needs to be appropriately registered so that the backend can deliver these messages to the appropriate services. Any kind of changes to the availability of the services has to be notified to the respective devices. Otherwise devices will unknowingly flood requests to an unavailable server leading to DOS attacks.

Mobility and Scalability

As the mobility of devices in the field area increases, there could be frequent disruption in the physical connectivity between the devices with their local bridges. To mitigate rogue devices from latching on to unauthorized services, peripheral devices that move to a new location have to maintain service continuity through secure handoff mechanisms. On the other hand, if the service continuity is disrupted, owing to the unavailability of the bridging infrastructure or other reasons, the isolated nodes must be extensively re-verified before the resumption of services. In most IoT applications it is sufficient to verify just the field devices, but in a few sensitive situations it might require two way verification of the backend system as well. An adversary node might inject some malicious data in a roaming network before moving back to the home network and hide itself in the roaming network to promote repudiation attacks. Further a cloned node may illegally want to exploit the mobility to co-exist in another network

Field deployments of IoT applications tend to grow either horizontally or vertically. In the case of horizontal scalability,hardware load balancers can be used extensively to improve service availability to the bridging infrastructure. It is thus important to allow only authorized devices to pass through.Otherwise, there is a potential for rouge nodes to launch DOS attacks by overwhelming the load balances. Alternatively in the case of vertical scalability, the network can extend multiple hops. It is thus important to be able to trust intermediate nodes in the field area network. Otherwise, it has the potential to introduce sinkhole or wormhole attacks.

Addressing and Identification

Field devices in IoT applications prefer to use low power radios for the last mile connectivity. As per current practices, coordinator nodes allocate local addresses to peer devices and these addresses do not follow a common standard. Moreover the addressing scheme of a field network remains hidden behind the FAN gateway/bridge, and makes it difficult to isolate a rogue node. For example, if a rogue node launches an attack to disable the intermediate bridge, it would be difficult to identify the erring node. Since the node becomes untraceable, at a later point of time it might deny for sending a message and thus lead to repudiation attacks. Further the node can attempt to access unauthorised privileges remaining screened from the outside network. Even it might take the advantage of not being traceable from the outside network and spoof within its network to other nodes claiming itself to be a genuine bridge of the network. To address this problem, low power radios must support a common addressing scheme like 6LoWPAN that would enable each node to obtain a unique IPv6 address. For further security, there should be a global repository where device-IP mappings can be queried for authenticity by interested services or certificate providers.

Spatio-temporal services

Any event in the IoT world can be characterized by the amplitude of a spatio-temporal impulse. Therefore it is necessary that the nodes in an IoT deployment are reasonably time synchronized and that they are able to tag any data with a spatial geolocation. To achieve this end, these devices could leverage location services like GPS or Cell Tower Triangulation or WiFi triangulation. But due to this feature, the user location data should not be revealed to unauthorised users as this might lead to privacy issues. Also to ensure the time drifts within the FAN cannot be exploited for replay attacks, the APIs exposed by the backend system should ideally be idempotent. Additionally the backend system must take into account the variance in the location data that is available from the end nodes. Otherwise it results in an inaccuracy of fault reporting leading to incorrect decisions.

Resource constraints

A vast majority of peripheral IoT devices are resource constrained. The constraints could be in terms of available computational resources, onboard memory(RAM and ROM), network bandwidth, energy availability, etc. As these nodes are open in the environment and easily accessible physically, they can be cloned and tampered. Also as these nodes have very little computation and storage capabilities, they preferably upload the data into the cloud for huge computations and storage. For this reason the user data privacy issues also creep in. So strong cryptographic encryption and integrity mechanisms need to be applied.

Data Interchange

Each of the different devices in an IoT application can potentially have a different data packing mechanism because of specific optimizations on their hardware platform. If an encrypted packet is decrypted and repacked at multiple points in the communication chain, it opens up a security vulnerability including information leakage and user privacy because the keys are being shared between multiple devices. Moreover, as the intermediary nodes are involved in crypto works they in turn become prone towards resource exhaustion and DOS attacks. For this reason end to end encryptions are better preferred.

Resource and service discovery

In an IoT application it is expected that a large number of end devices are going to be deployed in the field. Therefore it is imperative that IoT devices can function autonomously and that they can discover the necessary services that they need to consume. Therefore the coordinators in an IoT deployment must implement resource and service directories that can be queried on a public interface. The mechanisms like , two way authentication of the coordinators must be done to ensure that a rogue coordinator does not implement fake services and subsequently redirect traffic through itself and may lead to spoofing. Also due to this discovery mechanism, unnecessary requests to pretend to discover a resource causes DOS like attacks. As the IoT architecture is hugely distributed, the user data becomes distributed in the clouds. So without proper security measures it may lead to information leakage and user privacy issues. Accounting information inconsistencies may also arise here unless correct measures have been taken

Lecture 7

Security Challenges in IOT

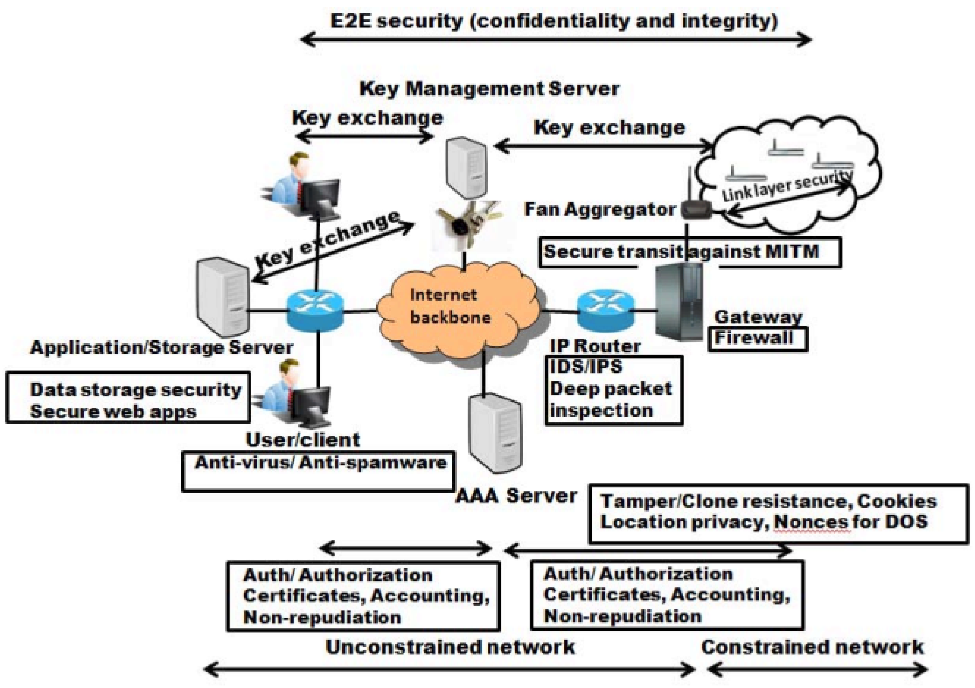
This section introduces an effective security framework to prevent various threats considered in the previous section.

Device or node end point physical security

As the IoT nodes are very easily accessible in the open environment, they have the possibility to be tampered or cloned. To prevent this, tamper resistant hardware should be provided for these nodes. Even if a node is tampered or compromised, there should be enough resiliency so that other nodes remain unaffected. Upon detection of a node being tampered, future communications with the node might be blocked to prevent further information loss. In the extreme case, the node may be needed to flush all data remotely so that the critical data are by no means accessed by illegal malicious nodes. Cloning of nodes can be prevented by allowing no access to its memory or crypto information from external sources.

Bootstrapping and setup security

While in the process of latching on to a network in the bootstrapping phase, the cipher and other critical information in the nodes should be kept secret. Typical cryptographic information including pre-shared keys and other private keys must not be compromised or stolen. After the bootstrapping phase is done safely, the device may participate in establishing shared keys.



Authentication, Access control and Accounting

Before any node communicates with a server, it should properly authenticate itself using certificates as well as the destination node to prevent open access to node data. Also justified authorization and access control rules are to be implemented on these nodes to limit them from super user access. At the end, detailed accounting information of the transaction should be logged. This way it will prevent to get exploited from spoofing, repudiation and privilege elevation attacks.

Data transmission and storage security

Data while being transmitted over the network should be made secure by using the light-weight crypto algorithms tailored for these resource constrained networks. Symmetric key encryptions are faster but PKI infrastructure though being slow allows dynamic key generation and distribution. For integrity checks and to prevent against MITM attacks light-weight hashes and integrity check codes may be used. Since the nodes have very less memory they can?t store huge amounts of data or are capable of doing huge computations and thus off-load them to powerful cloud servers. Stored data should be properly signed and encrypted so that they are genuine data and not readable by an anonymous entity.

Proxy security

The proxies become a major target because data are transformed from one form to another. This bridging infrastructure is thus expected to provide decent transport level security by implementing something like a secure DTLS over 6LoWPAN on their FAN side and TLS on their ethernet side. Secure interfaces and mechanisms of secure transit are required. Application level security can be built in addition to the transport layer security, depending on the capability of the hardware. Also cookie or nonce based mechanisms should be implemented to guard them against improper resource

allocations and DOS attacks. Highly trust based systems are to work in these middle nodes and proxies.

Network and routing security

The data while leaving a constrained network and being routed to the web from the router needs to be secure. Data in transit in the network backbone should also be prevented from malicious activities. Corresponding encryption and integrity checks need to be implemented to guard against eaves dropping and data distortion attacks. Intrusion detection systems are essential to detect when a network has been adversely attacked and compromised by an attacker. Menwhile, corresponding prevention and rollback algorithms should be implemented to prevent the overall network from further damage. Also the affected nodes should be isolated and investigated for the security breach.

Multi layer security

Higher layer protocols like (D)TLS, IpSec, etc. provide E2E security. But low powered devices latching with the gateway might run on 6LoWPAN and they in their network might need to adapt 802.15.4 link layer security. So the protocol stack should be flexible and accustomed with multiple security solutions while still maintaining the normal security requirements.

Common security measures

In addition to the above attack prevention scenarios, few common security measures are to be adapted to resist against the generous attacks. To avoid DOS attacks stateless protocols would be a suitable alternative. The loss due to spoofing can be limited by reducing the trust relationships, deep packet inspection and use anti-spoofing techniques.Parties involved n MITM attacks should be guarded by using certificate authentications wherever possible and message integrity checks using MAC and MIC should be used. Software securities like firewall, IDS, context aware security measures and deep packet inspection techniques should be applied. Anti-virus and antispamwares need to be deployed. Also application based web security protocols are valuable. Possible modifications are to be applied on the existing protocols to suite them in LLNs and also non-ip based systems. Encryption, authentication, integrity, anti-replay, non-repudiation which stands as the basic mechanisms for security needs to be applied correspondingly but in a light-weight fashion to protect against the attacks.

Lecture 8

Development and Other Challenges in IOT

IoT based systems are usually complex due to a tremendousimpact on all aspects of human lives as well as its various technologies deployed to enable autonomous data exchange between

embedded devices. Development of IoT has an impact on various aspects of human lives (e.g. security, safety, health, mobility, energy efficiency, environmental sustainability, etc.). Therefore, IoT related issues and challenges need to be considered from various aspects such as enabling technologies, services and applications, business models, social and environmental impacts. Analysis of recent contributions and research papers show that the most of the open issues arise due to an increasing number of connected devices which causes increased traffic demands with new

traffic models. Other issues are related to the integration of various technologies, heterogeneous environment (e.g. various devices,

data types, and network technologies), increased data storage and

processing demands, privacy and security risks, etc.

Standardization

Diversities in technologies and standards are identified as one of the major challenges in the development of IoT applications . Standardization of IoT architecture and communication technologies is considered as a backbone for the IoT development in the future. These facts imply that open standards are one of the key factors for the successful deployment of IoT. This type of standards is an important facilitator for innovation because of their availability to the public. They are being developed, approved, and maintained by a collaborative consensus-based decision-making process to provide better interoperability for systems using different technologies. Also, by using open standards there is less chance

of being limited to a specific vendor or technology which is very important factor for IoT development. The main standardization bodies such as ITU, ETSI, IETF, IEEE, W3C, OneM2M, OASIS, NIST, etc. are involved in the effort to make a framework for IoT standards. The scopes of standardization activities are various in order to provide open standards and architectures, seamless connectivity, interoperability, etc. But despite enormous efforts from standardization bodies and alliances, there is still no reference standard for IoT platform. There is a problem to integrate various standards and contributions to be consistent and coherent. All these open issues and challenges need to be considered in future

to enable seamless connectivity as well as the integration and interoperability among various IoT enabling technologies.

Architecture

IoT systems should be framed within open IoT architecture and set of standards to enable integration of various technologies and to support full interoperability. IoT architecture needs to provide interoperability and to support full mobility to ensure service continuity (without interruption). Accordingly, one of the main challenges for IoT systems is to use an open, integrated and standardized architecture with separated application logic

and hardware infrastructure. The corresponding architectures need to support heterogeneous nature of things, networks, data and applications to support full interoperability. Key design requirements for IoT architecture are scalability, interoperability, openness, and modularity in a heterogeneous environment. IoT architecture should enable multi-systems integration, cross-domain interactions, simple and scalable management functionalities, data analytics and user-friendly applications as well as the possibility to include the intelligence and automation across the IoT system. It may be treated as a system or paradigm which may consist physical objects (e.g. sensors, actuators), virtual objects (e.g. fog/cloud

services, communication layers and protocols) or a hybrid of these two perspectives. According to this perspective, IoT architecture can be classified into four groups: general/system-level, hardware/network, software and process.

General architecture considers a conceptual model to meet IoT requirements. There are numerous proposed IoT reference models

but still, there is no general architecture which provides full interoperability. The existing approaches and efforts to solve this issue

are based on layered frameworks and architecture. Hardware/Network architectures have to enable interoperability among various networks and communication technologies to provide full connectivity between IoT objects. IoT devices can be grouped into two groups based on TCP/IP protocol suite support. In order to solve this heterogeneity related issue, there have been proposed several approaches such as APIs, gateway solutions,

SDN-based solutions, NFV (Network Functions Virtualization), CCN (Content-Centric Networking), etc. Also, there are some other proposals such as a wearable IoT architecture with the ability to offer traceability of streamed data from source and the devices engaged with [188]. Other challenges are related to various technologies deployed in different networks including communication interfaces and access controls. All these issues must be considered in hardware and network architectures.

Software architectures should provide a common set of services to enable processing (aggregation, computation, etc.) large

amounts of data for service composition. IoT software architecture and framework need to be used to overcome the complexity of

systems and to provide an environment for IoT services composition. IoT software platform should be created as OAP (Open Application Platform) to enable modular design as well as providing an open APIs (Application Programming Interface) to sensors and other devices. Current IoT applications are mostly domain-specific with fragmented architectures that cannot integrate the data from different sources . In order to enable integration of various services with other infrastructure (e.g. Fog and/or Cloud) it can be used APIs, gateways, virtualization , SDN (Software Defined Networking), cloud centric architecture , etc. There are severalapproaches to provide application framework and another set of services for IoT such as SOA, RESTful, architectures based on fog and cloud computing, web application framework based on Google Toolkit, etc. These architectures cover Operating systems, IoT middleware, APIs, Data management, Big data, etc. Examples of Operating Systems (OS) for IoT support are Contiki, Riot OS, Android, TinyOS, LiteOS, etc. Future researches need to focus on improvements of RTOS (Real-Time Operating Systems), APIs and other components within IoT architectures to adapt these systems to IoT

Interoperability and integration

Interoperability is the ability of multiple devices and systems to interoperate regardless of deployed hardware and software. A variety of standards and technologies used for IoT development as well as various solutions from different vendors leads to massive heterogeneity which causes interoperability issues. IoT interoperability issue is considered across all layers. To overcome this issue, it need to be used a layered framework with standardized architecture. Technical interoperability is usually associated with communication infrastructure and protocols. IoT systems need to provide interoperability over heterogeneous devices, networks and a variety of communication protocols such as IPv6, IPv4, 6LoWPAN/RPL,

CoAP/CoRE, ZigBee, GSM/GPRS, Wi-Fi, Bluetooth, RFID, etc. The existing

Internet architecture doesn?t support full connectivity of

the heterogeneous devices and this is still a challenging task that causes complex integration issues Gateways provide several solutions such as protocol conversion or centralizing remote connectivity depending on purpose and

layer where they are implemented. Interoperable gateways at object layer support multiple interfaces to enable devices to be con

Other Challanges

Business models, new currencies in IoT and trust

A key area of research lies in building procedures

and protocols for decision making that are not based on the premise of speed. In a

real-time world there is no longer gain being the ?first? to have the data. Instead, the

internet of things favors a daily situation of full traceability. There is so much contextual information about what you are wearing ? this jacket or this pair of jeans ? that

neither the customer nor the merchant, require a Point of Sale/Point of Transaction as

a ?closure?. And yet ?closure? is of great importance to us as human beings, as it signals

the ?right? kind of feedback in a procedure enhancing levels of trust. It may be the case

that IoT will favor a situation where different forms of currencies, standards of banking

and money will exist together

Ethics, control society, surveillance, consent and data driven life

Privacy was named by the originator of ubicomp, Mark Weiser, the late chief scientist at Xerox Parc as a key issue (Weiser, 1991). Machina Research, in association with Latitude, Council and Info.nl - a trio of web 2.0 consultancy companies ? recently ran

an web survey, polling views on the future internet of things. One of the questions was

related to concerns that people may have about living in a future connected environment. Privacy was mentioned by a clear majority as a key challenge. Privacy Enhancing

Technologies (PET) is a partial solution. The Privacy Coach, produced by a small

Dutch consortium of RFID experts, is an application running on a mobile phone that

supports customers in making privacy decisions when confronted with RFID tags

(Broeninjk and others, 2011). It functions as a mediator between customer privacy preferences (Fischer-Hübner, 2011) and corporate privacy policies, trying to find a match

between the two, and informing the user of the outcome. Gérald Santucci (head of unit

Knowledge Sharing), key IOT architect at the European Commission explains: ”in the

future, the right to privacy, whatever we do to implement it with technology and/or

regulations (“right to be forgotten”, “right to oblivion”, “right to silent chips”, etc.), will

become a subset of ethics. The future is (largely) about ethics-by-design?