WASP

Communication and Service-oriented Protocol for the Internet of Things

Program Specifications



Communication and Service-oriented Protocol for the Internet of Things

Program Specifications

-by

Subhajit Sahu

Debiprasad Priyabrata Acharya

National Institute of Technology, Rourkela

December, 2013

Copyright and Usage Notice

Copyright © 2013, National Institute of Technology, Rourkela.

The copyright and trademarks of respective owners is acknowledged.

This document remains the property of National Institute of Technology, Rourkela; however, a worldwide license is hereby granted for the use of this document by educational and non-profit organizations provided this copyright notice is included on all derived material.

Use of this document by for-profit commercial organizations, both in India and overseas, is expressly forbidden without the written permission of National Institute of Technology, Rourkela.

The information contained in this document is subject to change without notice and should not be construed as a commitment by the author unless such commitment is expressly given in a covering document.

Revision History

|  |  |  |
| --- | --- | --- |
| Release No. | Date | Revision Description |
| Rev. 0 | 05/12/2013 | Rough Layout |
|  |  |  |

Document Overview

This document contains the detailed description of WASP, and is expected to act as a manual for engineers implementing it on a system. It is also expected to act as a general guide for student and engineers to understand the working. There exist websites, servers, applications and SDKs, which can be used design the desired system. This document provides information about the protocol, but does not contain the process of using the above mentioned tools.

Contents

[1 Introduction 1-1](#_Toc374342895)

[1-1 Internet of Things 1-1](#_Toc374342896)

[1-2 Making Things Talk 1-2](#_Toc374342897)

[1-3 What is WASP? 1-2](#_Toc374342898)

[1-4 System Structure 1-3](#_Toc374342899)

[1-5 Applications 1-4](#_Toc374342900)

[2 Data Types 2-1](#_Toc374342901)

[3 Application Layer 3-1](#_Toc374342902)

[3-1 Packet Structure 3-2](#_Toc374342903)

[3-2 Sections 3-3](#_Toc374342904)

[3-3 Objects and Functions 3-4](#_Toc374342905)

[3-4 Packet Examples 3-7](#_Toc374342906)

[4 Network Layer 4-1](#_Toc374342907)

[4-1-A Heading 3 4-1](#_Toc374342908)

List of Tables

[Table 2‑1: Meaning of each Section Flag. 3-3](#_Toc374343007)

[Table 2‑2: Classes of functions, and their range of function identifiers. 3-7](#_Toc374343008)

[Table 2‑3: Meaning of each function class. 3-7](#_Toc374343009)

List of Figures

[Figure 1‑1: The Internet of Things impacts healthcare and manufacturing industries in particular [2]. 1-1](#_Toc374342909)

[Figure 1‑2: Layering Structure of WASP. WASP includes both its Network and Application Layer. 1-2](#_Toc374342910)

[Figure 1‑3: Features of WASP (including optional features). 1-3](#_Toc374342911)

[Figure 1‑4: System Structure of WASP. 1-4](#_Toc374342912)

[Figure 2‑1: The various different things that an object represents. 3-1](#_Toc374342913)

[Figure 2‑2: Structure of an Application layer packet. Black colored sections are encrypted. 3-2](#_Toc374342914)

[Figure 2‑3: Invocation of Functions in a Section is sequential, as shown. 3-2](#_Toc374342915)

[Figure 2‑4: Structure of a standard Section. Function invocations are present in Section Data. 3-3](#_Toc374342916)

[Figure 2‑5: Structure of Section Flags. It is used to represent various properties of a section. 3-3](#_Toc374342917)

[Figure 2‑7: Structure of a Skip Section. It is used to represent blank sections. 3-4](#_Toc374342918)

[Figure 2‑8: Object in OOP vs. Object in WASP 3-5](#_Toc374342919)

[Figure 2‑9: Format of a normal function invocation 3-5](#_Toc374342920)

[Figure 2‑10: Assignment of Function ID to a normal function, when a 16-bit number is assigned to it. 3-5](#_Toc374342921)

[Figure 2‑11: Assignment of Function ID to a field, when a 15-bit number is assigned to it. 3-6](#_Toc374342922)

[Figure 2‑12: In WASP, functions are classified into various groups. 3-6](#_Toc374342923)

# Introduction

## Internet of Things

With the advent of the Internet, it became possible to interconnect networks of computers together. It enabled people to share information in ways, never anticipated before. Today, we all know where it stands; the most successful communication system ever built. Despite all of its complexity and enormity, it works like an unbelievably flawless system.

However, it is almost wholly dependent on human beings for information. Nearly all of the data was captured and created by human beings - by typing, pressing a record button, taking a digital picture or scanning a bar code. The problem is, people have limited time, attention and accuracy—all of which means they are not very good at capturing data about things in the real world. And that's a big deal. We're physical, and so is our environment ... You can't eat bits, burn them to stay warm or put them in your gas tank. Ideas and information are important, but things matter much more. Yet today's information technology is so dependent on data originated by people that our computers know more about ideas than things. If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so [1].

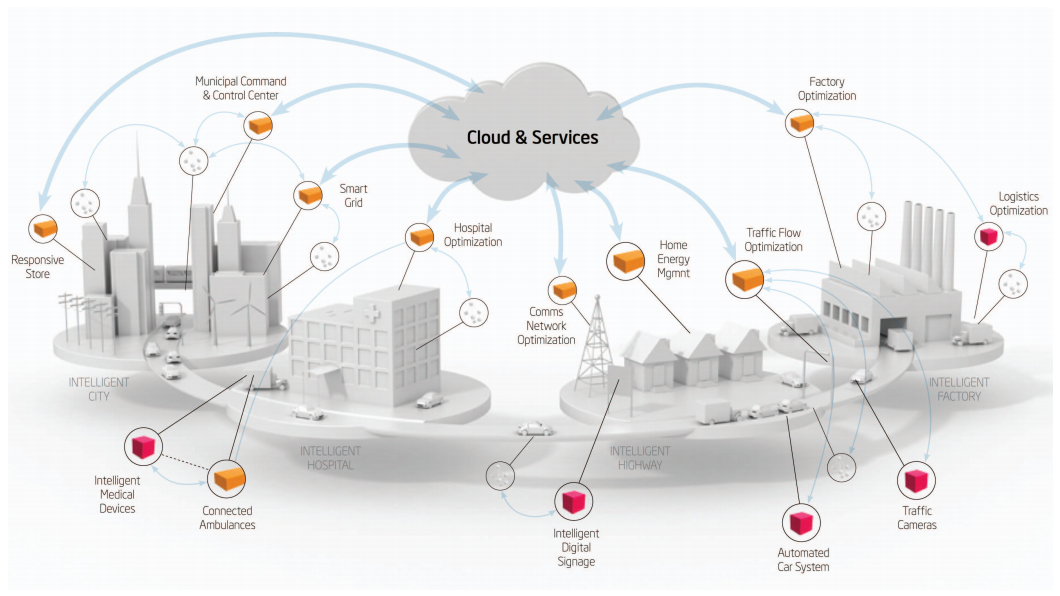


Figure ‑: The Internet of Things impacts healthcare and manufacturing industries in particular [2].

## Making Things Talk

How would it be, if your watch could inform you that your meeting was postponed, your mobile could ask the charger to recharge it, your fridge would tell the home delivery food service what to bring, or your room heater could heat up before you reached home? It would be a world where every-day physical objects could work intelligently, and keep you informed about everything, whenever you wanted. It will usher a new era of automation, thus allowing us to reduce waste, loss, and cost.

The IoT[[1]](#footnote-1) is all about making the things talk, among themselves, and to us. It is this sharing of information, which would allow every-day physical objects to act intelligently. This is however not possible unless all devices and applications follow one common protocol. The protocol that they choose to follow must be sufficiently expressive to allow a variety of devices, designed for a variety of applications to work together, and provide services. It must also be compact enough, so as not to burden the communication networks.

## What is WASP?

Wirelessly Attached Sensor Protocol, or simply WASP, is a communication and service-oriented protocol for the Internet of Things. It is a set of rules, which defines the method of exchange of information between devices, which includes packet formats, routing techniques, and packet processing methods. Designed for scalability, ease of use, efficiency and low power; WASP easily handles communication and services through the use of dynamic network topologies. It provides carefree handling of the Network, Transport, and Application OSI[[2]](#footnote-2) layers in case of WSNs[[3]](#footnote-3), and only the Application layer, in case of devices directly connected to the internet. The layering structure of WASP is shown in Figure 1‑2.

|  |  |
| --- | --- |
| WASP | WASP – Application Layer |
| WASP – Network Layer |
| Lower Layers (Physical / Link / Network / Transport) | |

Figure ‑: Layering Structure of WASP. WASP includes both its Network and Application Layer.

WASP – Network Layer ensures data integrity through the use of a checksum for each and every network packet. Reliability and flow-control are also provided, but they are optional, and the use of these features is decided by the sender, who creates the packet. WASP incorporates optional security features, in its Application Layer, that prevent any form of information leakage or manipulation, and ensure that data is delivered to the desired receiver only. The encryption method can either be a symmetric-key encryption technique, such as the AES, or an asymmetric-key encryption technique, such as the ?. The only restriction is that the desired receiver must be aware of the encryption technique being used, and must know its key.

Figure ‑: Features of WASP (including optional features).

It is the Object-Function model followed by the Application Layer of WASP, which allows devices to provide any kind of service. According to this model, sensors, actuators, devices (including agents), users, and their groups are all called as objects, and any operation to be performed by these objects are called functions. WASP has been designed to satisfy the following requirements:

* To enable an object to communicate with any other object, anywhere.
* To enable an object to request/provide service from/to another object.

## System Structure

WASP is an Agent-based system, where Agents[[4]](#footnote-4) perform the task of managing the activities of a set of objects[[5]](#footnote-5). In this system, there exists two types of Agents, a Service Agent, and a Communication Agent. A Service Agent actually manages a set of devices, and helps them provide services. A Service Agent knows everything about a sensor, stores its recorded data, and keeps tracking it. On the other hand, a Communication Agent does not manage or store information about a particular device, but knows how to communicate with any desired device, i.e., it knows how to find the Service Agent for any given device. Both the types of Agents work together and act as the IoT infrastructure.

Each Communication Agent is assigned a hierarchy level depending upon its knowledge of association of devices to Service Agents. Each Communication Agent may seek the help of a higher level communication agent, if it cannot determine the Service Agent of a particular device. However, Service Agents have no such hierarchy system, and each such agent works individually, or seeks the help of a communication agent. An example system structure of WASP is shown in Figure 1‑4.

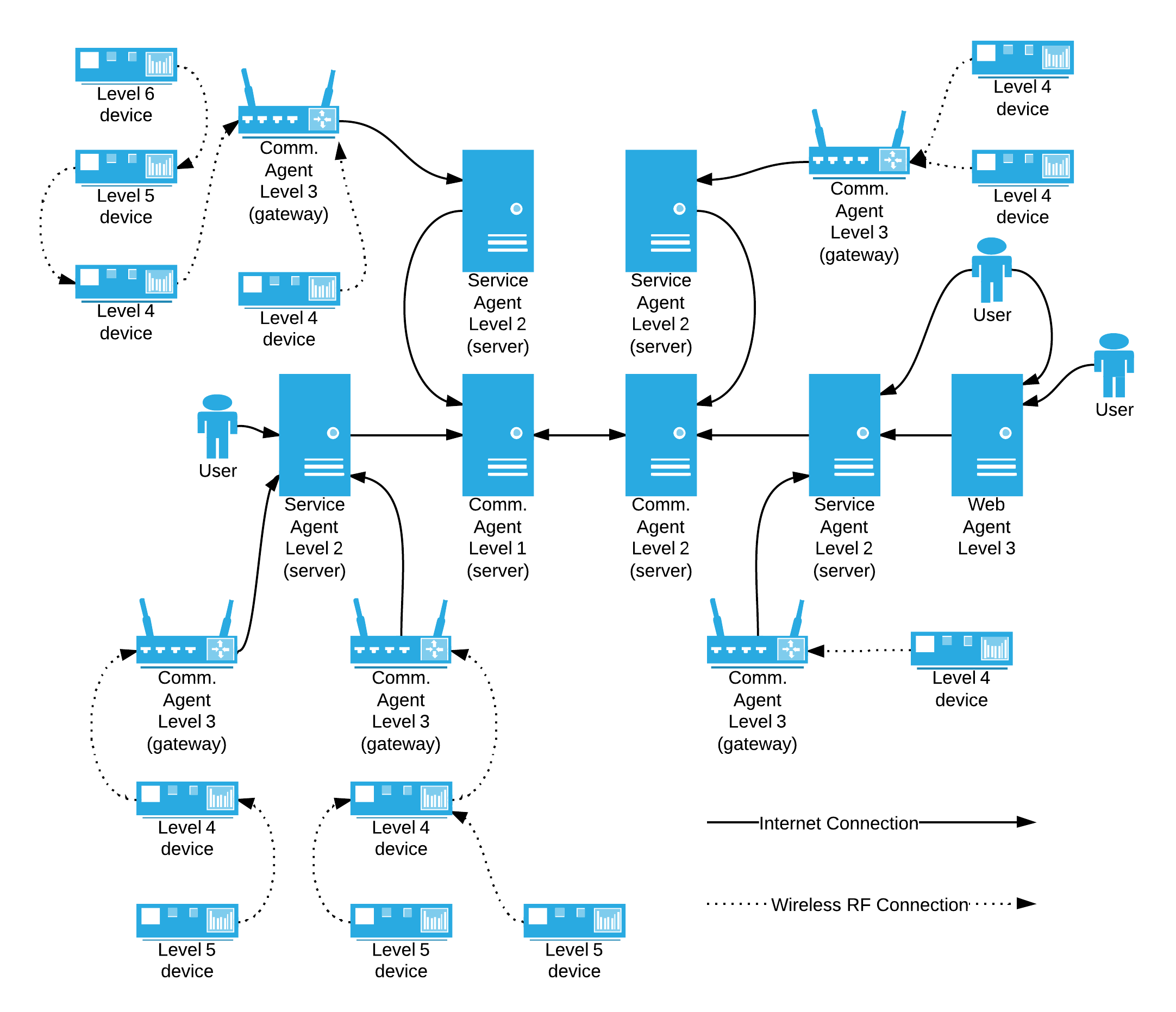


Figure ‑: System Structure of WASP.

## Applications

WASP can used for designing any system that requires connecting a set of devices, such as a wireless sensor network, to a service. Examples of possible systems which can developed are mentioned below:

* Automated Disaster Forecasting System.
* Automated Weather Monitoring System.
* Text / Video Chatting Service for all devices.
* Internet based phone
* All applications of Internet of Things

# Application Layer

WASP – Application layer is based on the object-function model, where sensors, actuators, devices (including agents), users, and their groups are all referred to as objects, and each object has a set of associated functions. The various different things that an object represents are shown in Figure 2‑1. Apart from that, there exist standard global functions, which can be executed by an object. Every packet in WASP only contains a set of functions to be executed by an intended destination. The return values from the functions can be caught, and sent back to the source object, if it desires so. If an error occurs during the execution of a function, it may either be suppressed, or reported back to the source. It is this ability to perform a remote function invocation, which allows WASP devices to provide any kind of service.

Figure ‑: The various different things that an object represents.

## Data Types

Functions can be invoked in WASP using any data type, such as, integer, floating-point, array, structure, etc. Functions, having variable parameters can also be used in WASP. WASP follows a specific protocol for specifying data for function parameters. Data types, such as short, long, etc. which have a predefined type and size are directly used in WASP. Arrays, strings, etc. which have variable lengths, but are of fixed types make use of a length field before the actual data. Object type which is neither has a type, nor size, must be used with a length field, followed by the type field.

WASP Packet format makes use of unsigned integer (UINT. The unsigned integer, though considered as a single data type, actually can be of different sizes, which can vary from 8-bits to 128-bits. These are named in short as, UINT8 to UINT128.

The other data type that WASP uses, is the length-specified data-block (BLK), which can be used for object, or function specification. When this data type is used, the first field is a UINT16 defining the size of the value in bytes, which is then followed by the value. The main purpose of introducing this data type is to allow a particular function to be executed on a group of entities.

The Type field, which is a set of 3-bits, is used to represent the data-type of a particular value in the packet format. The binary values of the Type field, and their meanings, and short names are shown in Table 3-1. When the data type of a value is not specified in the packet format, then the default type is implied, which is UINT16, or simply UINT.

## Packet Structure

Each Application layer packet is divided into one or more sections, where each section contains a number of function invocations. Each section represents a logical group of functions, and is meant for performing a particular set of sequential tasks. If any errors occur in function invocation of a particular section, the section is not continued any further. If the section which encountered an error is a transactional section, then if possible, the object is reverted back to its original state before the section execution started. If errors are not suppressed, it is reported back to the sender in the same section number. However, any errors occurring in function invocation of one section does not affect function invocations in any other section.

By convention, the first section is considered as the header section, and should not be encrypted. Figure 2‑2 shows the structure of an Application layer packet, where a packet contains multiple (one or more) sections, and each section has one or more function invocations. Invocation of each function in a section is performed serially, as show in Figure 2‑3. Each section contains a maximum of 255 variables, which can be used for passing parameters to functions, or getting return value from them. Variables of the return-type are sent back to the sender inside same section numbers.

Figure ‑: Structure of an Application layer packet. Black colored sections are encrypted.

Figure ‑: Invocation of Functions in a Section is sequential, as shown.

## Sections

The structure of a standard section is shown in Figure 2‑4. Each section starts with a UINT8 containing a set of flags, which indicate various properties of the section, and its structure is shown in Figure 2‑5. The meaning of each section flag is shown in Table 2‑1. It is then followed by a UNIT16, which indicates the size of the section (N), in bytes.

|  |  |  |
| --- | --- | --- |
| Section | | |
| Flags | **Size (N)** | **Data** |
| UINT8 | UINT16 | N bytes |

Figure ‑: Structure of a standard Section. Function invocations are present in Section Data.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Section Flags | | | | | | | |
| UINT8 | | | | | | | |
| Encrypted Section | 0 | Suppress Return | Suppress Error | Transactional Section | 0 | 0 | Section Skip |

Figure ‑: Structure of Section Flags. It is used to represent various properties of a section.

|  |  |
| --- | --- |
| Section Flag | Meaning |
| Encrypted Section | If 1, it indicates that this section is password encrypted, and is intended to be read only by the desired destination. |
| Suppress Return | If 1, it indicates that the return values of function inside the section should not be returned back to the sender, by default. |
| Suppress Error | If 1, any error that occurs while reading or executing a function in the section will not be reported back to the sender. |
| Transactional Section | If 1, any error that occurs inside the section will cause the object’s state to be restored to the state it originally was, before execution of the section. |
| Section Skip | If 1, it indicates that this is a Skip Section, which is used to represent a number of blank sections. |

Table ‑: Meaning of each Section Flag.

The section data, which is of N bytes in size contains a set of function invocations. When the section-skip flag in section flags is set, a skip section is used, which is used to represent blank sections. Figure 2‑7 shows the structure of a skip section.

|  |  |
| --- | --- |
| Skip Section | |
| Flags | **Number of sections to skip (N)** |
| UINT8 | UINT16 |

Figure ‑: Structure of a Skip Section. It is used to represent blank sections.

It is important to note that since each section operates independently, the receiver may execute a section even when the complete application layer packet has been received. This technique can be very useful for device, which do not have sufficient memory space to store an entire packet. As mentioned above, by convention the first section is considered as the header section and can contain important information, such as, unique identifier of the packet or whether an acknowledgement packet is required.

## Objects and Functions

In OOP[[6]](#footnote-6), objects are instances of classes[[7]](#footnote-7), which have a set of associated properties or attributes called “fields”, and a set of associated operations or procedures, known as “methods” [3] [4]. The Object-Function model of WASP is very similar to OOP; however, it differs in the point that WASP treats fields and methods alike; together they are referred to as “functions”. This is shown in Figure 2‑8. Hence, in order to represent a field in WASP two separate functions, such as, Get and Put functions, are used. Apart from that, WASP also provides standard procedures (that are independent of objects), which can be used by objects to perform a non-instance operation. Also, WASP considers namespaces[[8]](#footnote-8), classes, and functions alike.

Figure ‑: Object in OOP vs. Object in WASP

WASP represents each object by a unique 128-bit identifier. Functions, of type Object, Standard, Namespace or Class, are represented by a 16-bit identifier. An object can only invoke its “own” functions, or functions of objects “contained” in it. The actual parameters of a functions are passed directly after it. The format of a normal function invocation is shown in Figure 2‑9.

|  |  |
| --- | --- |
| Function ID | Actual Parameters |
| UINT16 | Depends upon the function, and variables, if used |

Figure ‑: Format of a normal function invocation

Figure 2‑10 and Figure 2‑11 show how function identifier is assigned to a normal function or a field. It is also possible to pass variables, to a function, when it is required to send or receive data from a function. However, by default, if a function that is invoked returns a value, it is automatically returned back to the sender unless its return value is suppressed. If a function returns an instance, then by default, continued function invocation is expected. A special function can be used to specify the properties of a function invocation. In WASP, functions are classified into various groups, as shown in Figure 2‑12.

|  |
| --- |
| Function ID |
| UINT16 |
| Function Number |
| UINT16 |

Figure ‑: Assignment of Function ID to a normal function, when a 16-bit number is assigned to it.

|  |  |
| --- | --- |
| Function ID | |
| UINT16 | |
| Field Number | **Get Field (if 0) / Set Field (if 1)** |
| 15-bit (MSBs) | 1-bit (LSB) |

Figure ‑: Assignment of Function ID to a field, when a 15-bit number is assigned to it.

Figure ‑: In WASP, functions are classified into various groups.

The range of function identifiers for each function class is shown in Table 2‑2, and the meaning of each function class is shown in Table 2‑3. The functions associated with each function identifier is predetermined for all function classes, except object type, and is shown in the Appendix. Special functions are used for specifying function call flags, passing variables, returning variables to sender, return error status, or blocking a continued function call. Functions of associated with the object are defined by the creator of the object. Please note that the list of functions are continually updated, and hence it is necessary to refer to WASP website.

|  |  |
| --- | --- |
| Function Class | Function ID |
|  | UINT16 |
| Special Function | 0x0000 – 0x00FF |
| Standard Packet Function | 0x0100 – 0x1FFF |
| Standard Grouped Function | 0x2000 – 0x2FFF |
| Standard Function | 0x3000 – 0xBFFF |
| Object Grouped Function | 0xC000 – 0xCFFF |
| Object Function | 0xD000 – 0xFFFF |

Table ‑: Classes of functions, and their range of function identifiers.

|  |  |
| --- | --- |
| Function Class | Meaning |
| Special Function | Used to perform special operations, such as, to work with variables, alter the working of a function invocation, and many more. |
| Standard Packet Function | Used to indicate various properties of a packet. |
| Standard Grouped Function | Used for a variety of standard non-instance operations. |
| Standard Direct Function | Used for a variety of common standard non-instance operations. |
| Object Grouped Function | Used for performing operations associated with the object. |
| Object Direct Function | Used for performing common operations with the object. |

Table ‑: Meaning of each function class.

## Packet Examples

Gimme the examples

# Network Layer

Error, return, transaction skip, encrypted.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Network Layer Packet | | | | |
| **Size (N)** | **Flags** | **ID** | **Source?** | **Destination?** |
| UINT16 | UINT16 | UINT16 | UINT? | UINT? |
| **Checksum?** | **Offset?** | **Field?** | **Data?** | |
| UINT16 | UINT16 | UINT16 | N bytes | |

|  |  |  |  |
| --- | --- | --- | --- |
| Network Flags | | | |
| UINT16 | | | |
| Transaction [2] | Section Skip [2] | Error Reporting [2] | Function Return [2] |
| Encryption [2] | Reserved [2] | Section Type [4] | |

|  |  |
| --- | --- |
| Address Type | Meaning |
| 0 | No Address (used in lower layers) |
| 1 | Local Address (16-bit) |
| 2 | Global Address (64-bit) |
| 3 | Universal Address (128-bit) |

|  |  |
| --- | --- |
| Sequencing | Meaning |
| 0 | No sequencing |
| 1 | Sequence Start / Continue |
| 2 | Sequence Termination |
| 3 | ? |

|  |  |
| --- | --- |
| Encryption | Meaning |
| 0 | No encryption |
| 1 | AES (symmetric-key) encryption |
| 2 | Asymmetric-key encryption |
| 3 | ? |

|  |  |
| --- | --- |
| Priority | Meaning |
| 0 | Low |
| 1 | Medium |
| 2 | High |
| 3 | Very High |

### Heading 3

#### Heading 4

1. The Internet of Things (or IoT for short) refers to uniquely identifiable objects and their virtual representations in an Internet-like structure [6]. [↑](#footnote-ref-1)
2. The Open Systems Interconnection (OSI) model is a conceptual model that characterizes and standardizes the internal functions of a communication system by partitioning it into abstraction layers [5]. [↑](#footnote-ref-2)
3. A wireless sensor network (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions, and to cooperatively pass their data through the network to a main location [7]. [↑](#footnote-ref-3)
4. Agents are servers, which are directly connected to the internet, and running WASP server application. [↑](#footnote-ref-4)
5. Sensors, actuators, devices, (including agents), users, and their groups are all called as objects. [↑](#footnote-ref-5)
6. Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs [3] [4]. [↑](#footnote-ref-6)
7. In object-oriented programming, a class is a construct that is used to define a distinct type. The class is instantiated into instances of itself – referred to as class instances, class objects, instance objects or simply objects. A class defines constituent members that enable its instances to have state and behavior [9] [10]. [↑](#footnote-ref-7)
8. A namespace is nothing but a group of assemblies, classes, or types. A namespace acts as a container—like a disk folder—for classes organized into groups usually based on functionality [8]. [↑](#footnote-ref-8)