I. Introduction

WIMA is an API designed to support image transmission over Wirelles Sensor Networks (WSN), it provides a collection of classes to facilitate the implementation and experimentation of visual systems over WSN. This work is a practical extension of an image transmission research, the design is based on the main techniques found on the literature. The API design aims to support the simulation process, it is ready to use Contiki-ng/Cooja which is a popular academic OS and simulator tool for Internet Og Things applications.

The API has the fundamental characteristic the ease of adaptation and extension for using in IoT applications, tailored for experimentation of various identified scenarios. Due WSN technical constraints, compression, change detection and optimized transmission algorithms are explored on the literature to overcome the WSN technology limitations.

II. WIMA

The API is divided in the following main parts:

- **Image components**: Represents the image data information to be transmitted over the network, they are named **tImage**. The image component means an image data or a ready to use database of images, taking advantage of the simulator.

- **Image Slicing**: Normally an image represents a large block of data to be handled, in a WSN environment the computational power is limited to solve heavy mathematical algorithms, then researchers slice the image into small fragments to solve such problems, the API defines **Slicer** component for this task.
  Slicing is also necessary to transmit a large block of data on the network, it has a maximum size of data that can be transmitted at once. If the size of the transmitted data block is greater than a frame of the network, the fragmentation into smaller parts is required.

- **Image compression:** Compression is the main technique to reduce amount of data to be transferred in a constrained wireless network, **Compression** represents the image compression methods.

- **Change Detection**: Change detection techniques detect the parts changed in a sequence of messages, in this way only the changed areas needs to be transmitted, **ChangeDetection** is a component to implement the change detection algorithm. Similarly the compression component, change detection is an optional implementation.

- **Network**: An entity to interface the transport layer, in which the data will actually be transmitted, considering WSN built with constrained device, UDP is the recommended protocol, WIMA implements **TransferUDP** component for networking process.

- **Application Protocol**: There are not specialized protocols to transmit large amount of data for WSN, the design of optimized protocols between transmitter and receiver is a mandatory component in API, **AppProtocol** coordinates the transmission of data between the sender and receiver parts.

All components are detialed defined in section V.

III. WIMA Components relationship

The API is designed for both sides of the network client and server, on the components there are interfaces to transmit and to receive images. Considering two elements of the network, one to send (client) and another to receive (server), the possible relationships between the API components, their attributes and methods are described in the figure 1.

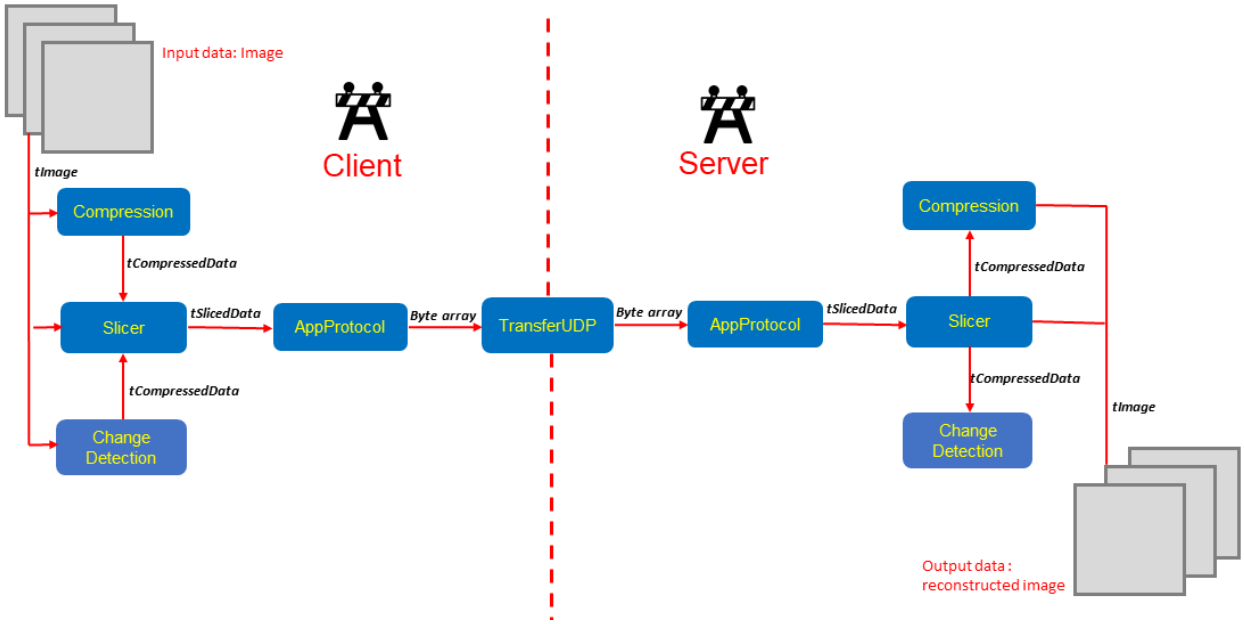

Figure 1: WIMA components behavior for client and server parts.

IV. WIMA implementation

The API is implemented in C language, since most of WSN devices only supports this low level language, it is designed based on pointers to functions inside the main structure, which has the named of the component, for each component there is also an "**init**" function.

To use WIMA firstly the component is created by declaring a variable with its type, at the same the component is initialized. The API doesn´t bring full implementation of the strategies for transmission like compression methods, the developer has to implement the according to the strategy evaluated.

After the transmission strategies implemented, theirs methods needs to be pointed to the component, similarly

In a practical example, to use **tCompression** component follows these steps:

Step 1: Source code analysis - The minimum API design as function prototypes and types are defined in the api source code, it is also can be extended:

```
typedef struct tCompression {
tCompressedData * (*compressChangeDetectionData)(struct tCompression*, tCompressedData*);
tCompressedData * (*compressImage)(struct tCompression*, tImage*);
} tCompression;
```

Step 2: Component creation - A component is declared and initialized by its create function :

```
tCompression Compression=tCompression_create();
```

Step 3: A compression method has to be implemented: The API defines only prototypes and types to the components interact each other, it is necessary to implement the strategies are going to be evaluated. It is not the aims of this work to develop the methods. At the API´s source code there are empty functions, at these places the strategies may be coded:

```
tCompressedData * tCompression_compressImage(tCompression*, tImage* img) {

 /* Compression method needs to be implemented here */
  return NULL;
}
```

Step 4: Compression method is associated to the components – The function has to be pointed to structure data:

```
compression->compressImage = &tCompression_compressImage;
```

Note that steps 3 and 4 are already done in init tCompression_create, but the developer may associate different methods any time after creation, in this way the API is very flexible to implement various scenarios to transmit images.

To complement the information provided in figure 1, a typical API component´s instantiation steps are given bellow :

Step 1: Image Files : Get image files, in case of simulation imageDB is initialized

```
tImageDb * DBImage=tImageDb_create();
```

Step 2: Initialize network :

```
TransferUDP * UDPTransfer=tTransferUDP_create();
```

Step 3: Defines the compression method to be applied to the image

```
tCompression= Compression tCompression_create();
```

Step 4: Initialize application protocol

```
tAppProtocol * tAppProtocol_create();
```

The sequence is the same for both, client and server sides.

V. WIMA UML Classes

In the section II the main API´s components are described regarding their roles, here all components are detailed define as UML diagram classes as follows:

- **Image data components**: defined in figure 2 by tImage and tImageDB.
- **Slicing process**: defined in figure 3 by tSlice and tSlicedData.
- **Image compression**: define in figure 4 by tCompression, tCompressedData.
- **Change Detection**: defined in figure 5 by tChangeDetection and tChangedStatus.
- **Network**: defined in figure 6 by tTransferUDP. TrawReceivedMsg.
- **Application Protocol**: defined in figures 7 and 8 by tAppProtocol, tSliceDataMsg, tReplyMsg, tHandShakeMsg, tImgSliceHeaderMsg, MessageID and ReplyStatus.

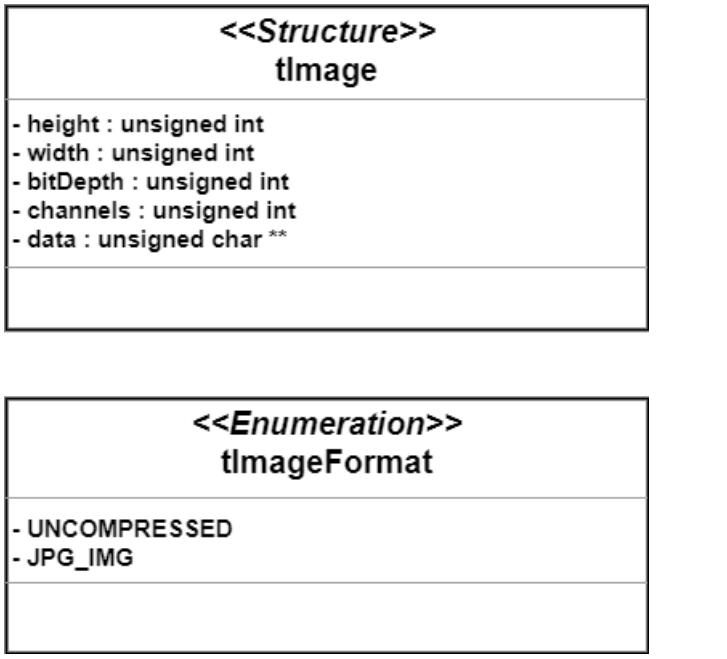

Figure 2: tImage,tImageFormat and tImageDB UML classes

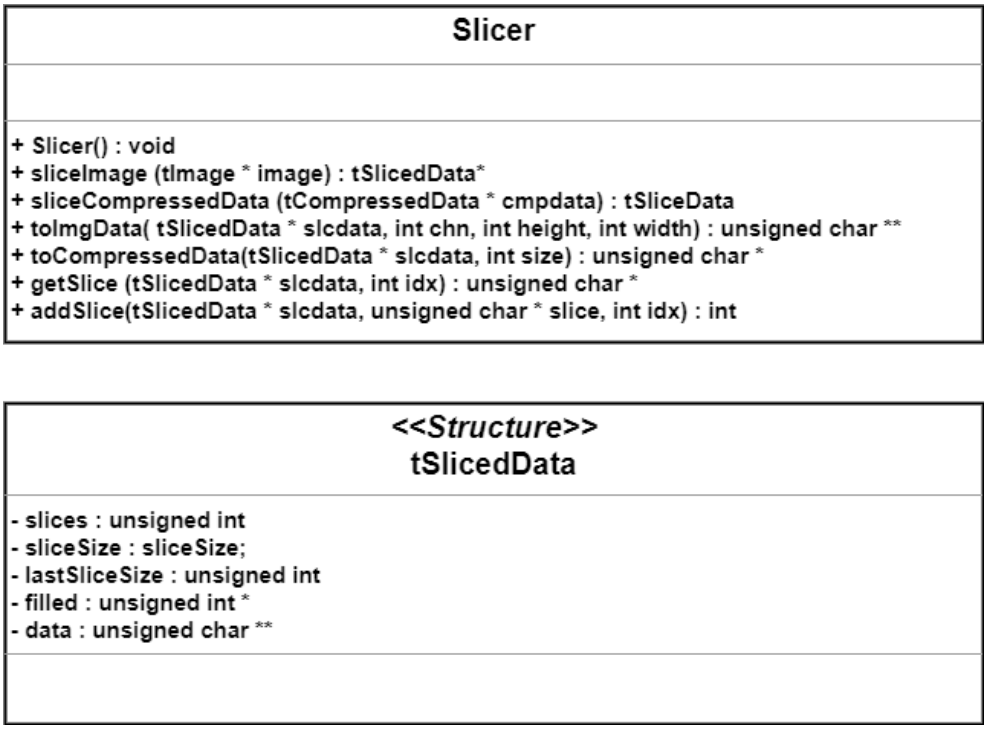

Figure 3: Slicer and tSlicedData UML classes

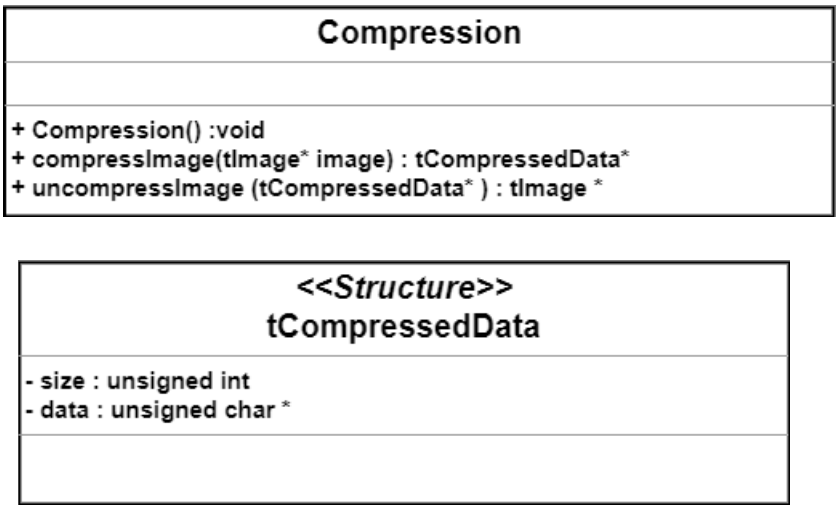

Figure 4: Compression and tCompressedData UML classes

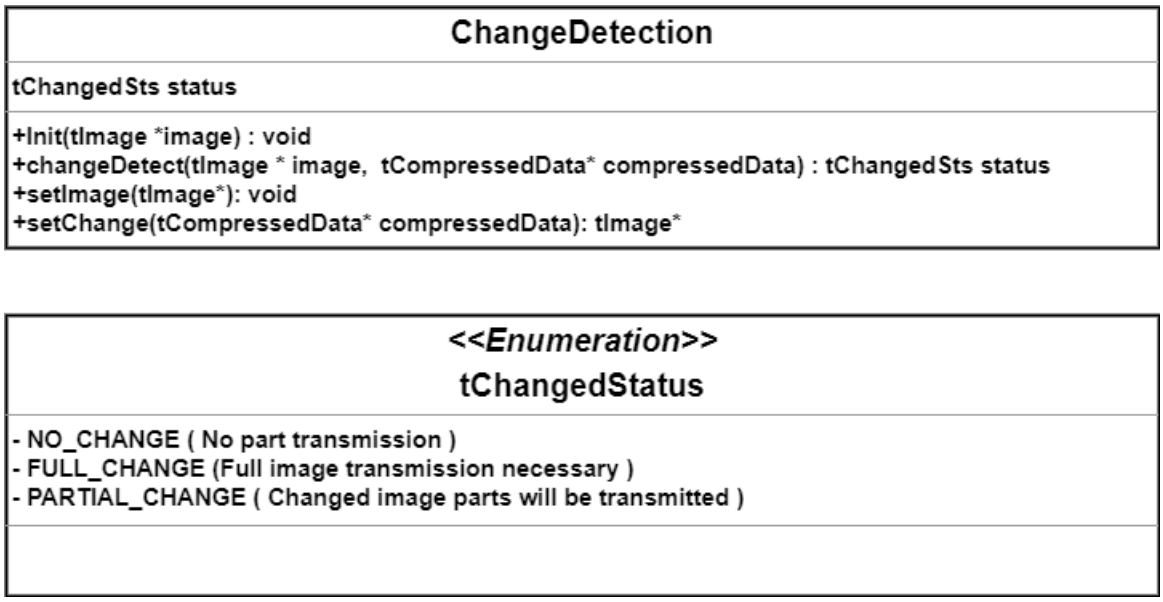

Figure 5: ChangeDetection and tChangedStatus UML classes

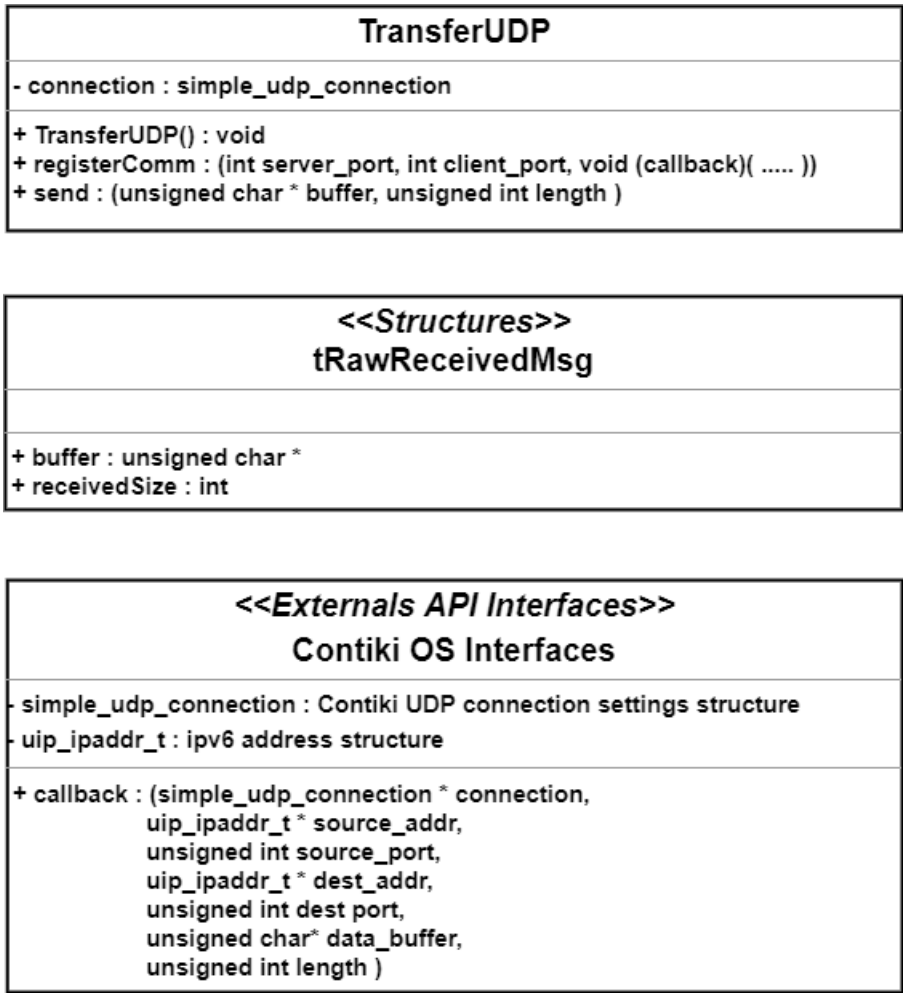

Figure 6: TransferUDP, tRawReceivedMsg and Contiki interface UML classes

Figure 7: tAppProtocol, tSliceData, tReply and tHandShakeMsg UML classes

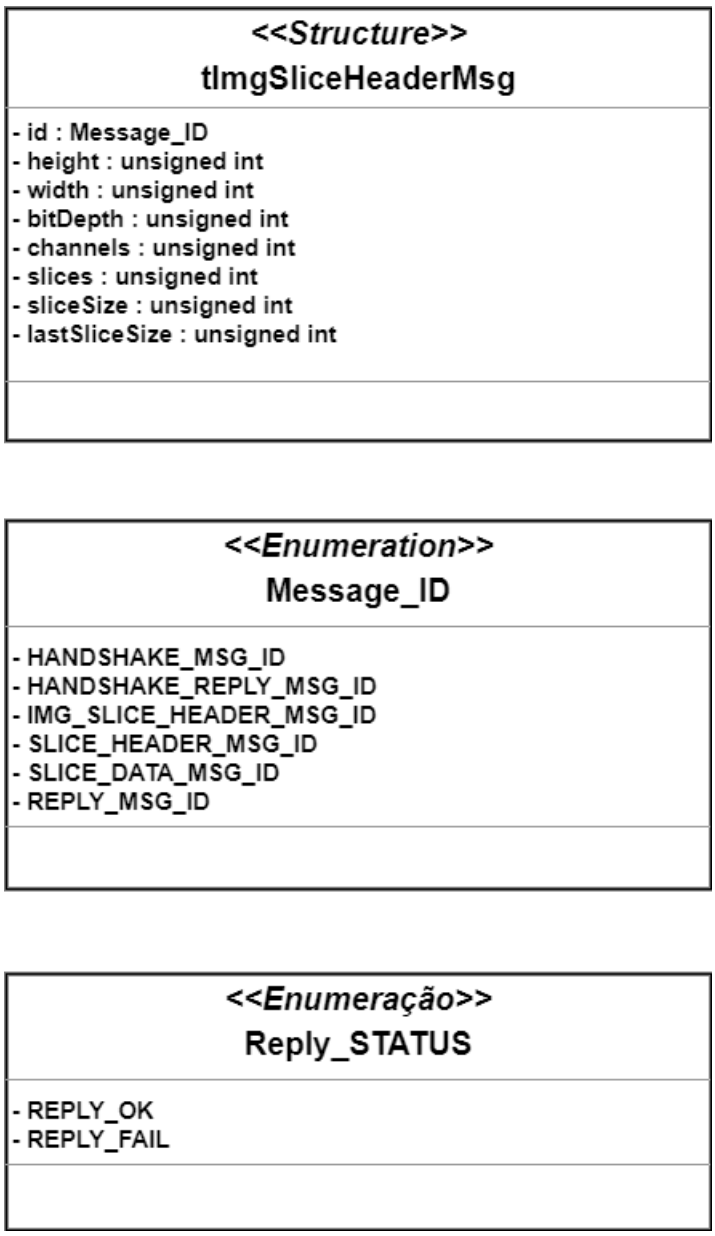

Figure 8: tImgSliceHeaderMsg, MessageID and ReplyStatus UML classes

VI. Detailed use cases examples

In this section three use cases are described by a detailed sequence diagrams:

- **Raw image transmission**: described by figures 9,10,11 and 12.
- **Raw image with change detection transmission**: described by figures 13,14,15 and 16.
- **Compressed image transmission**: described by figures 17,18,19 and 20.

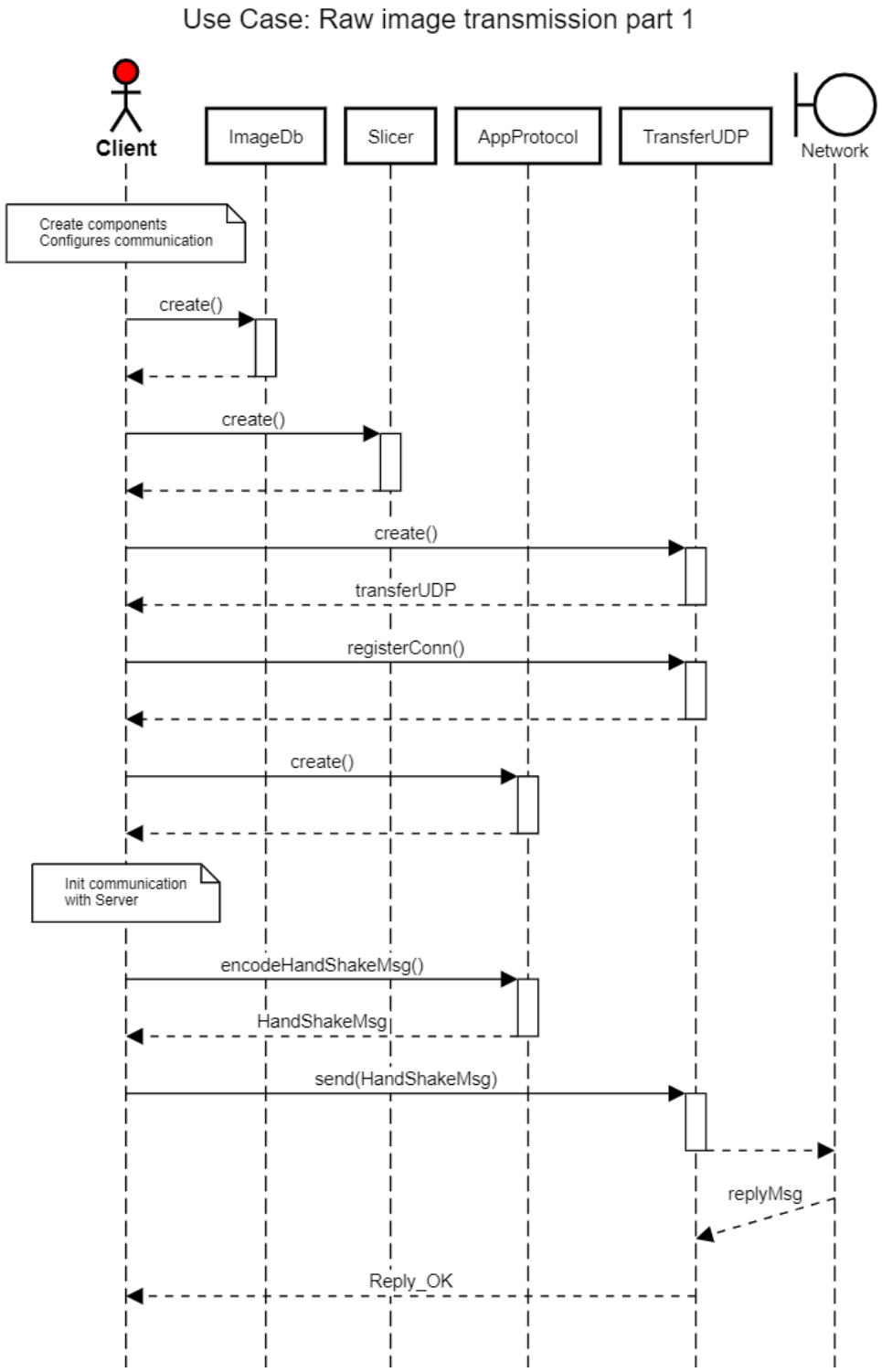

Figure 9: Raw image transmission detailed diagram part 1

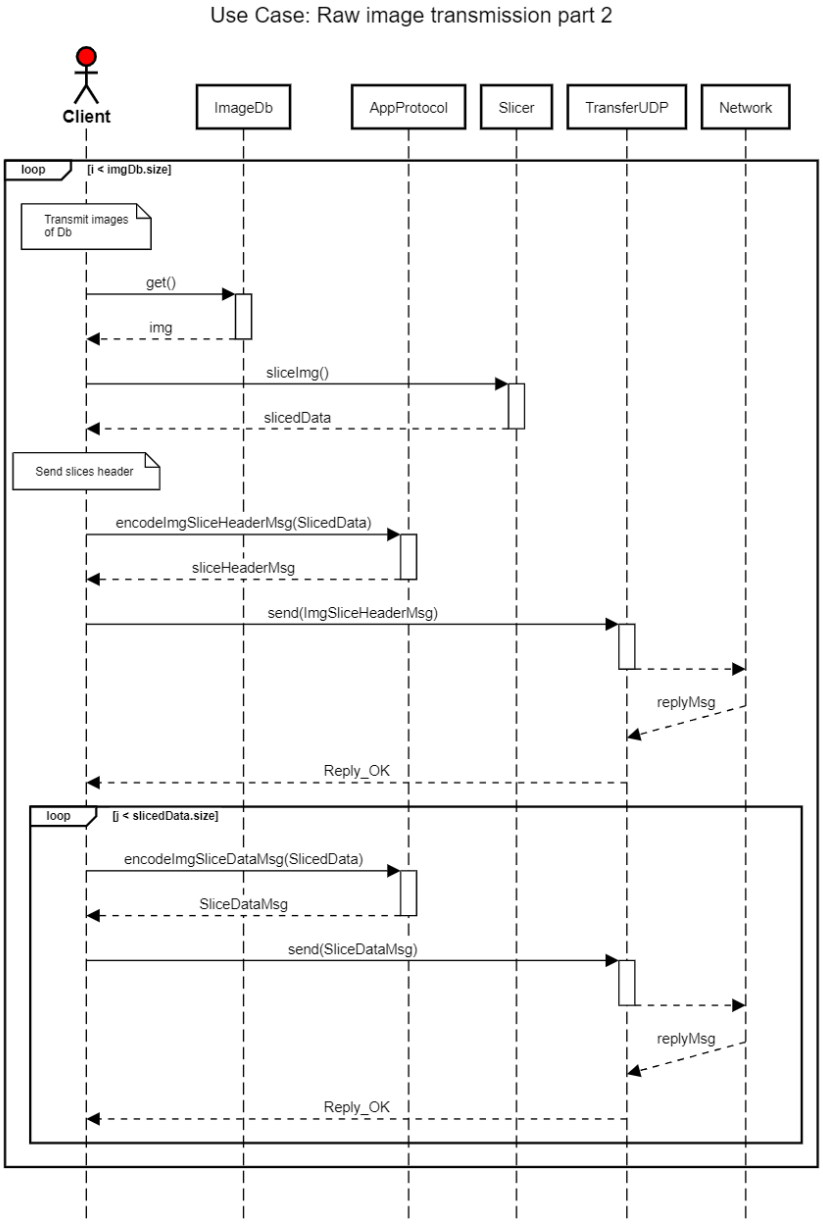

Figure 10: Raw image transmission detailed diagram part 2

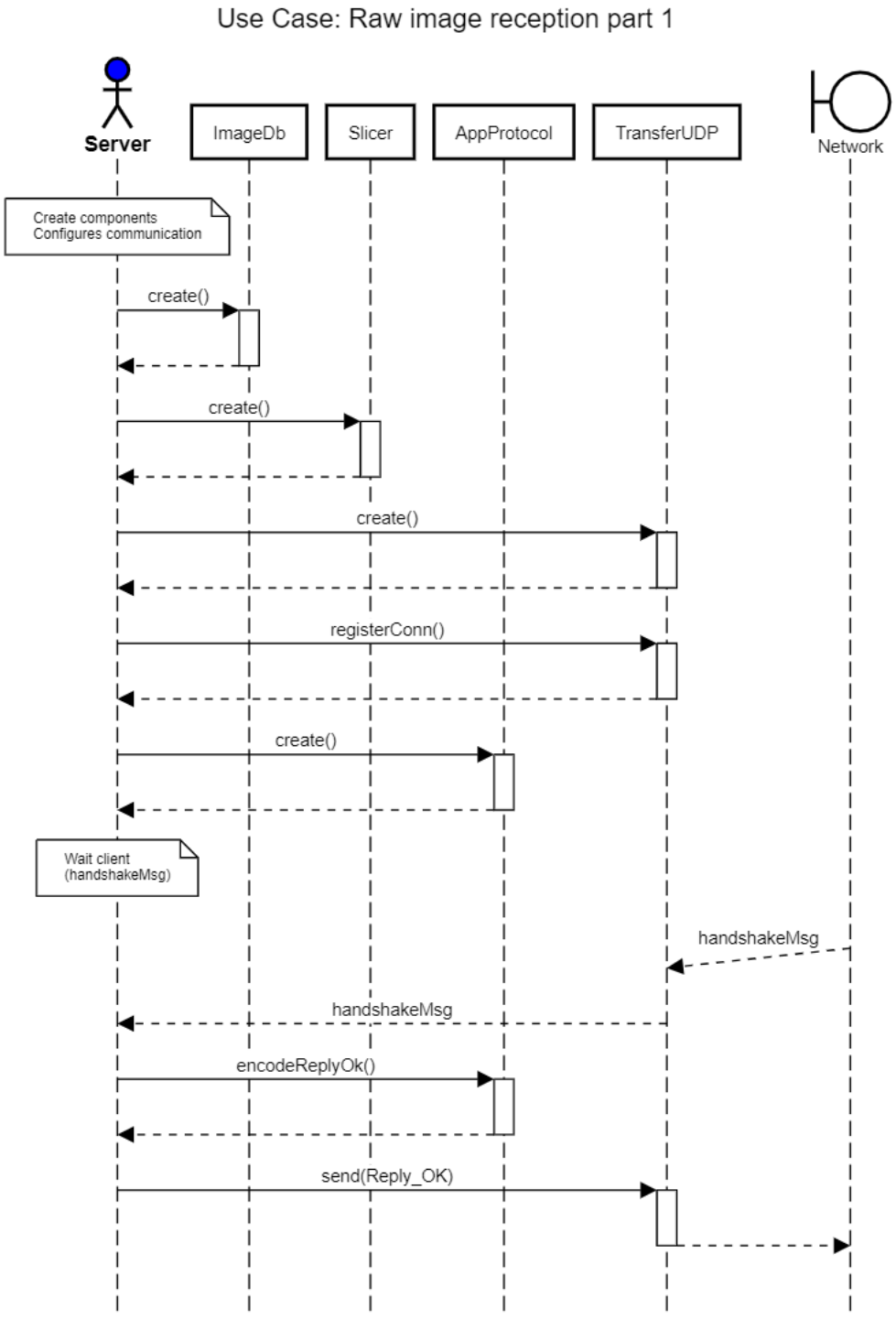

Figure 11: Raw image reception detailed diagram part 1

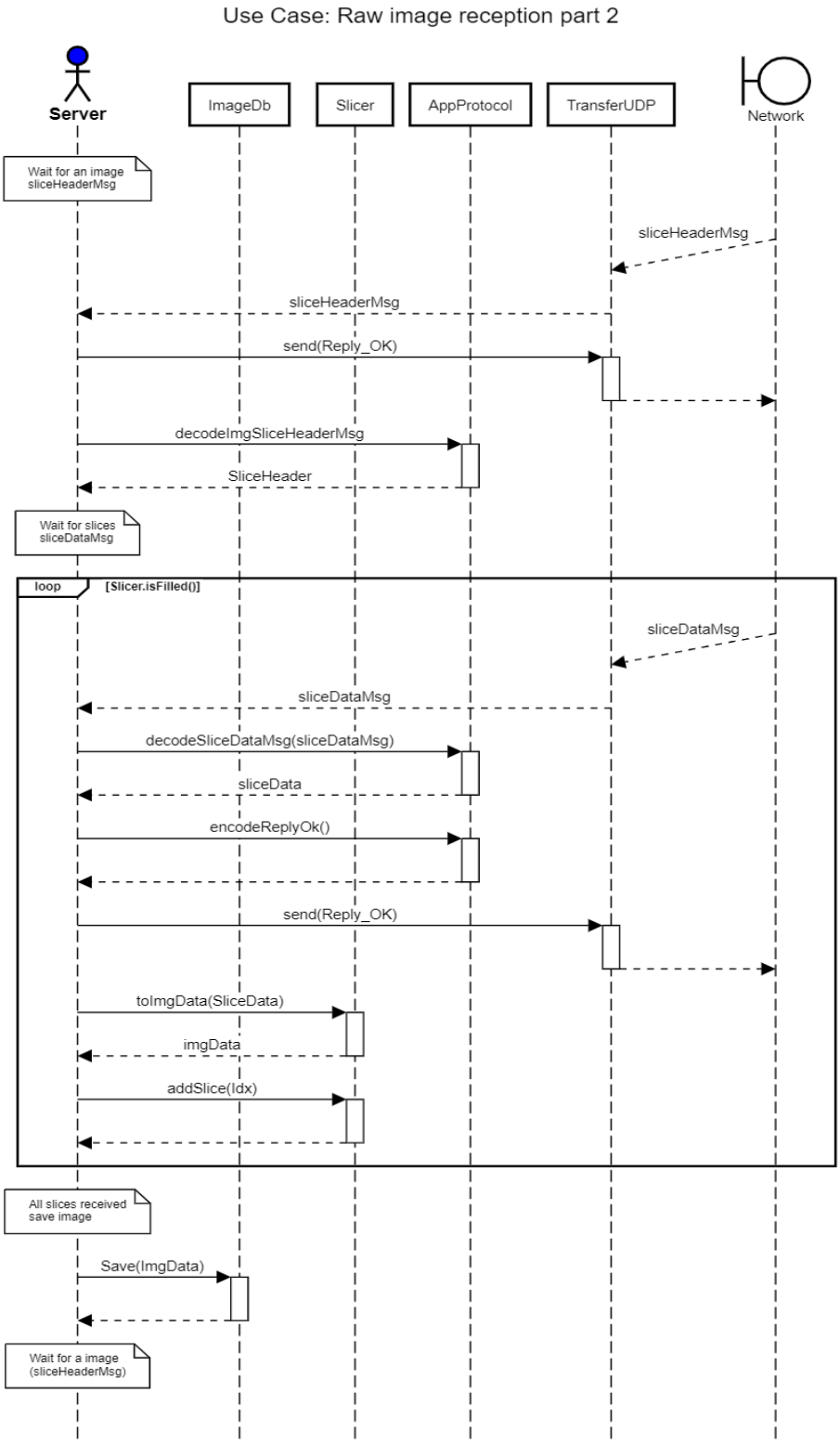

Figure 12: Raw image reception detailed diagram part 2

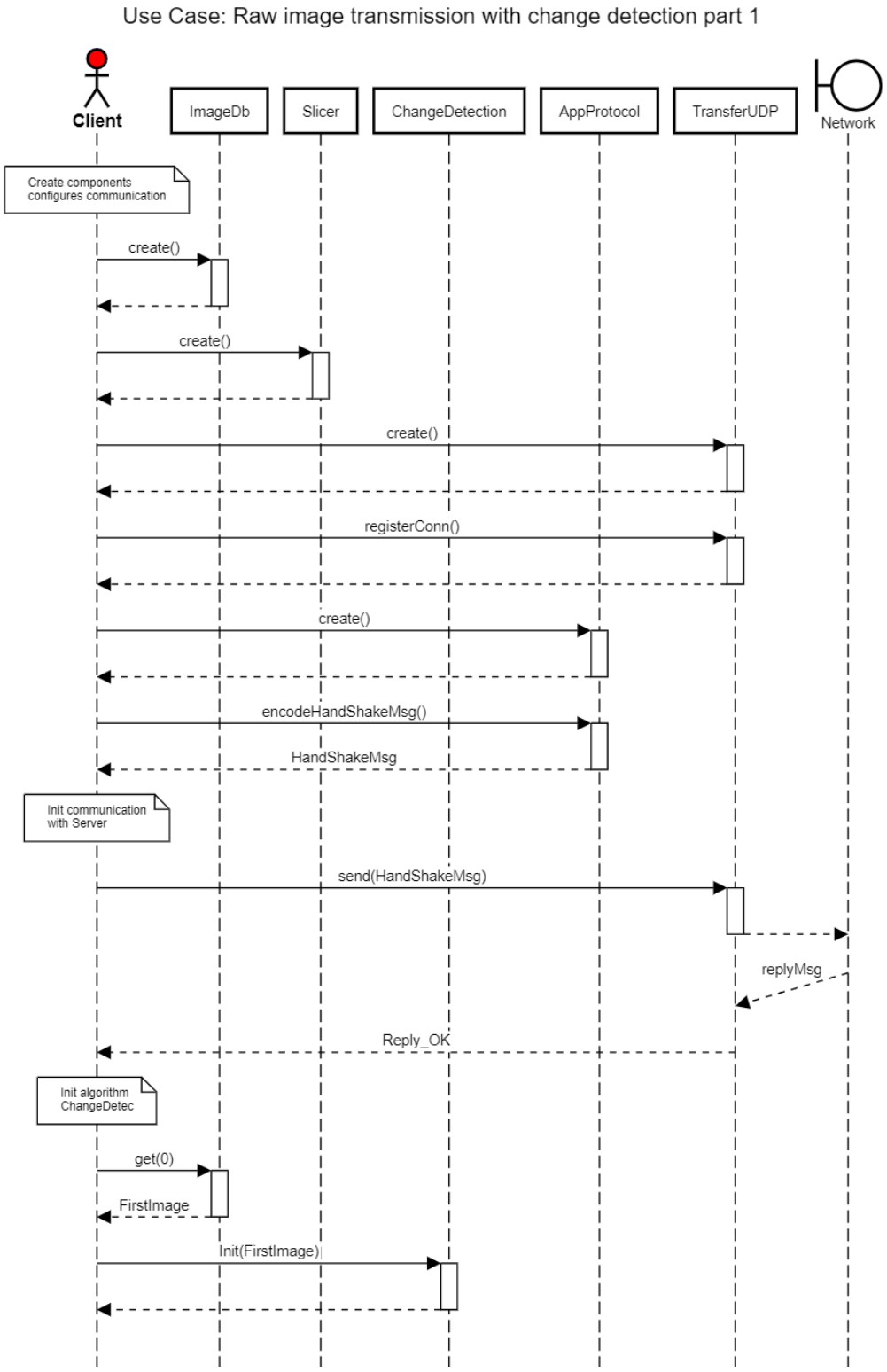

Figure 13: Raw image transmission with change detection detailed diagram part 1

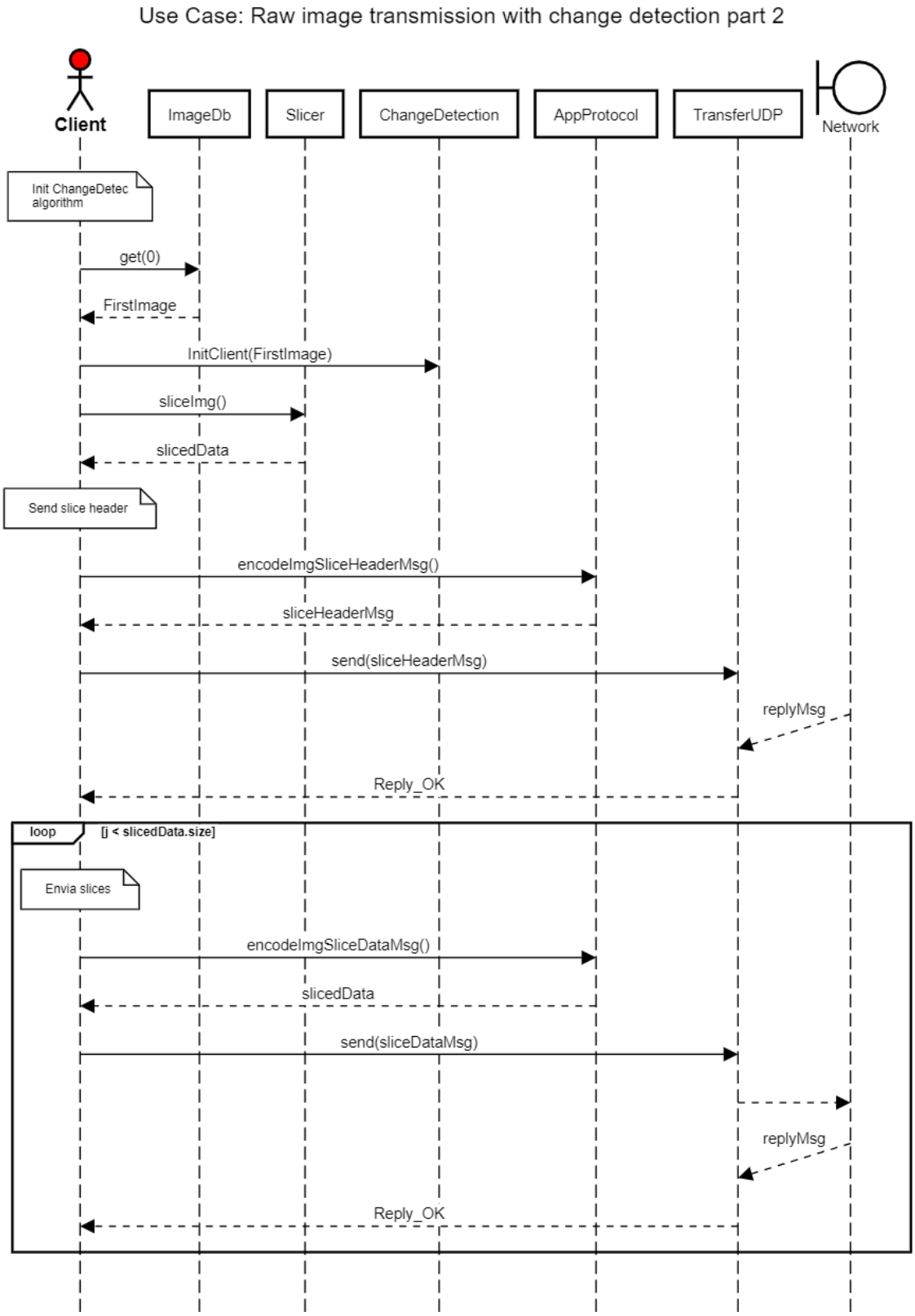

Figure 13: Raw image transmission with change detection detailed diagram part 2

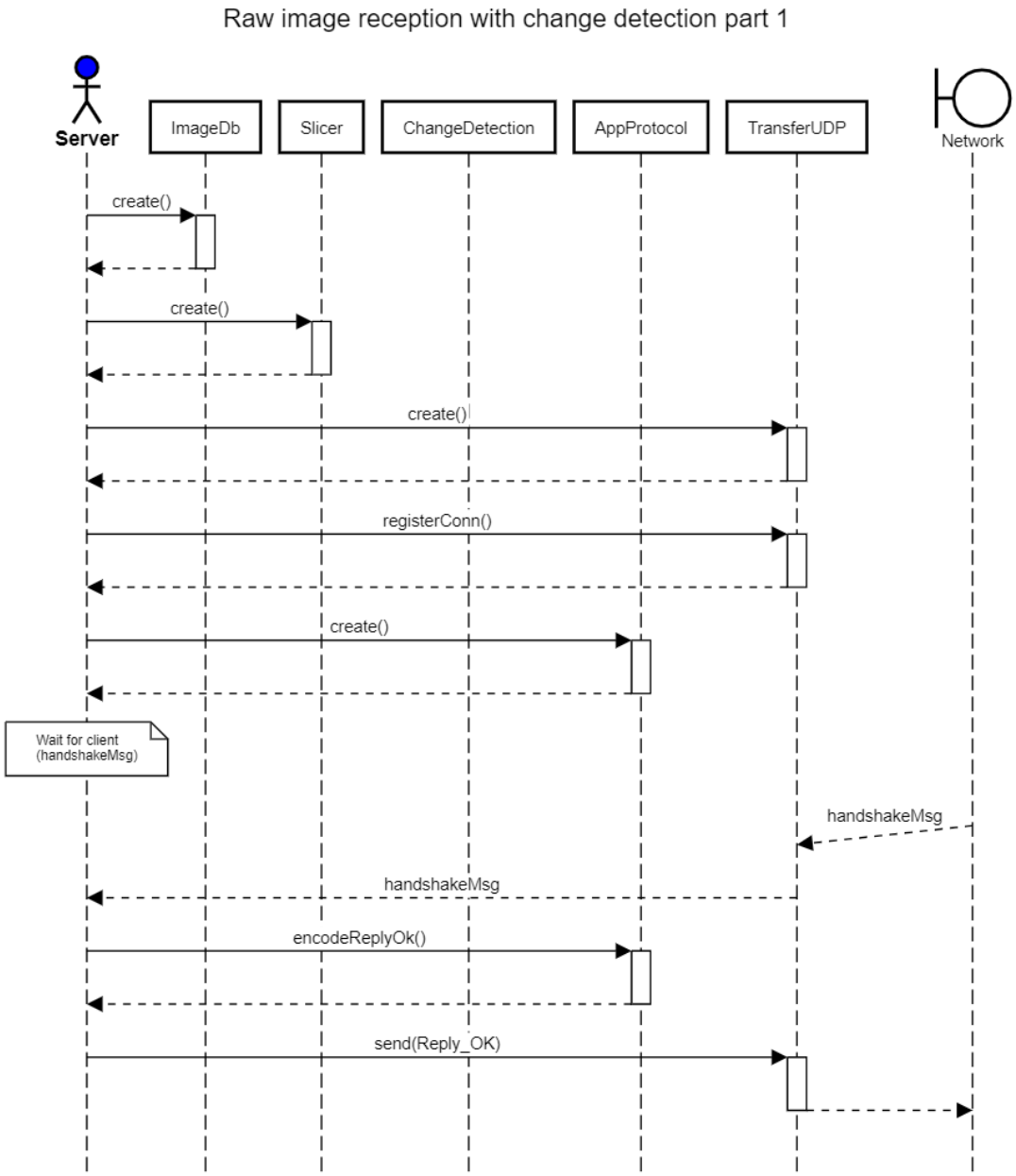

Figure 14: Raw image reception with change detection detailed diagram part 1

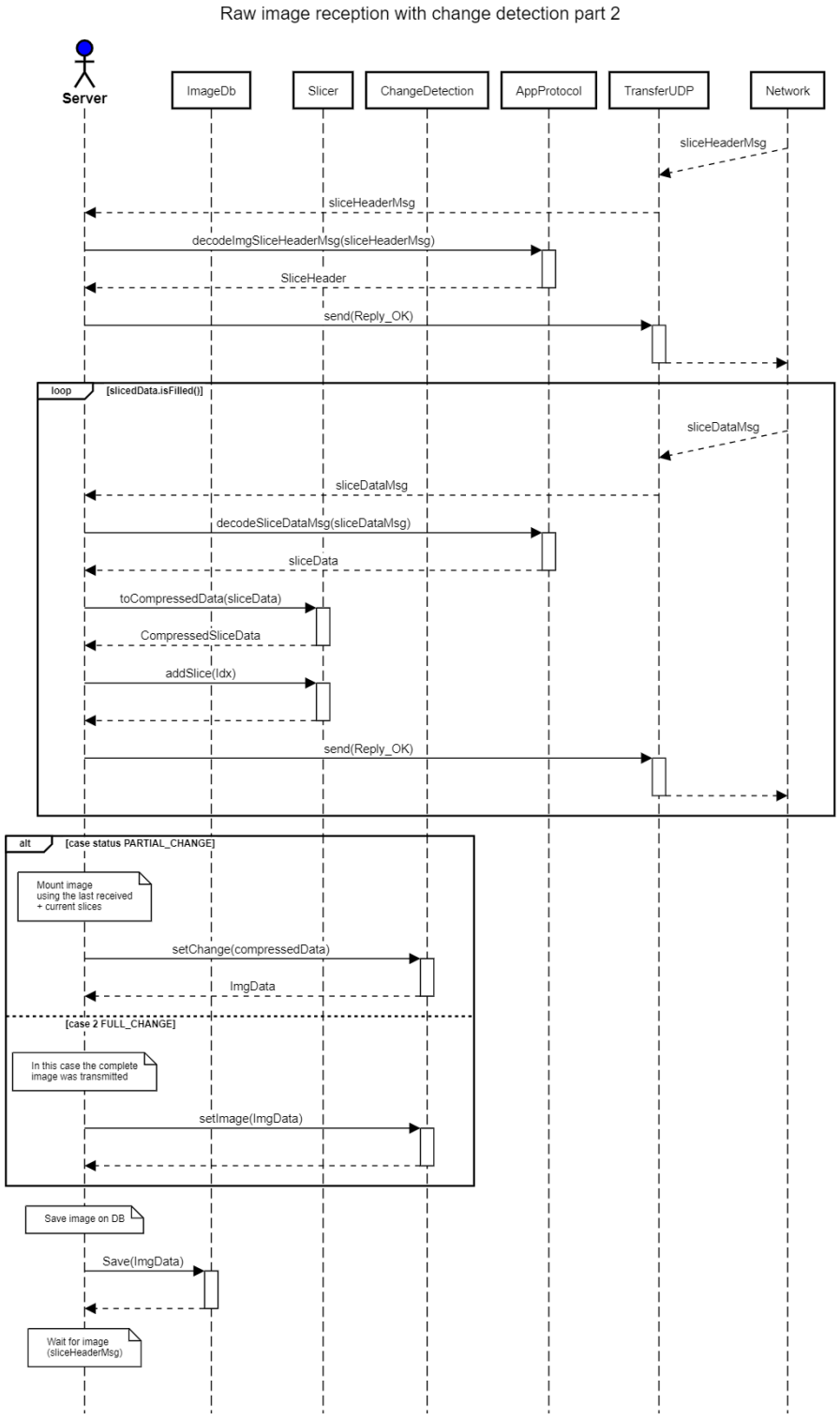

Figure 14: Raw image reception with change detection detailed diagram part 2

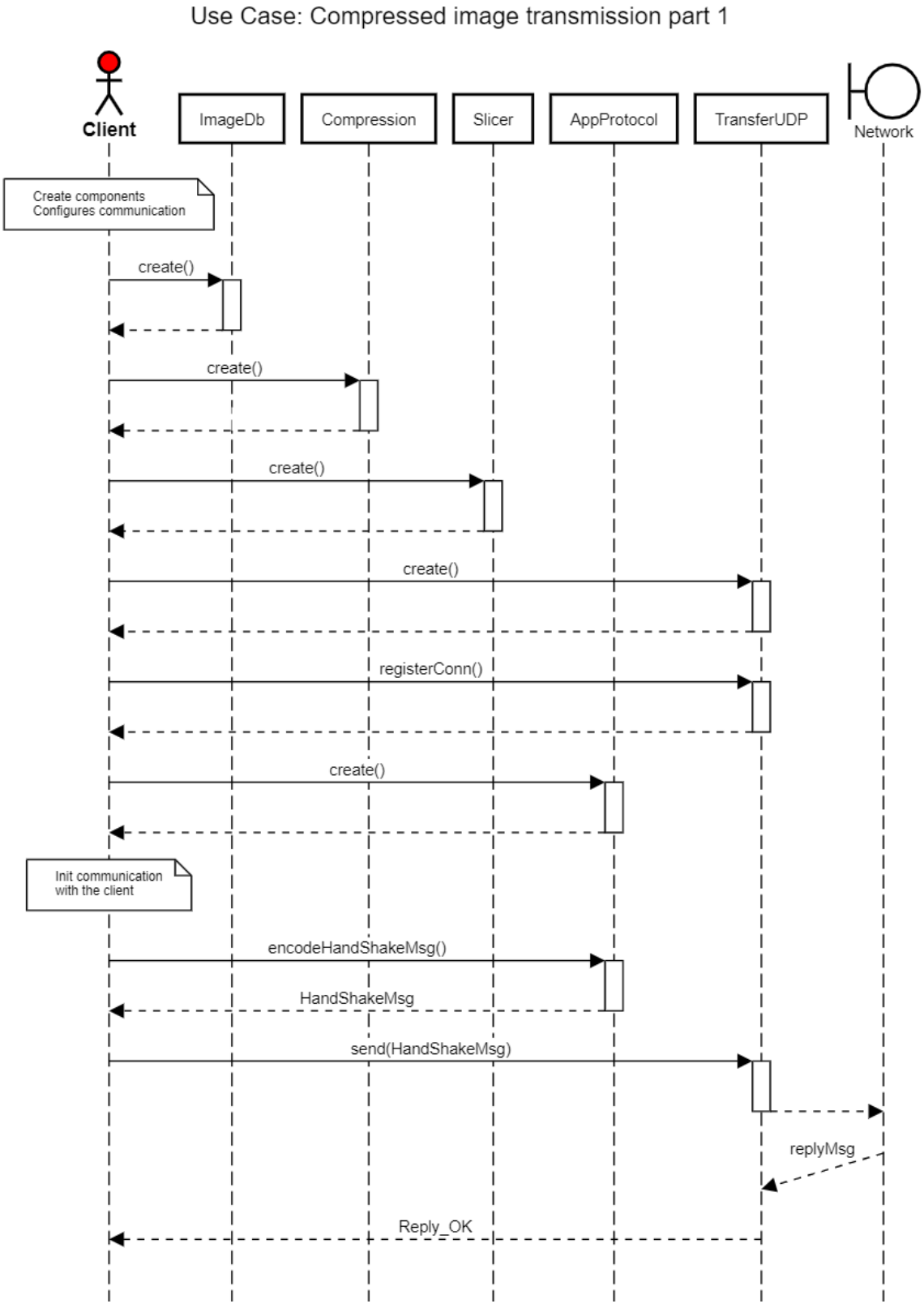

Figure 15: Compressed image transmission detailed diagram part 1

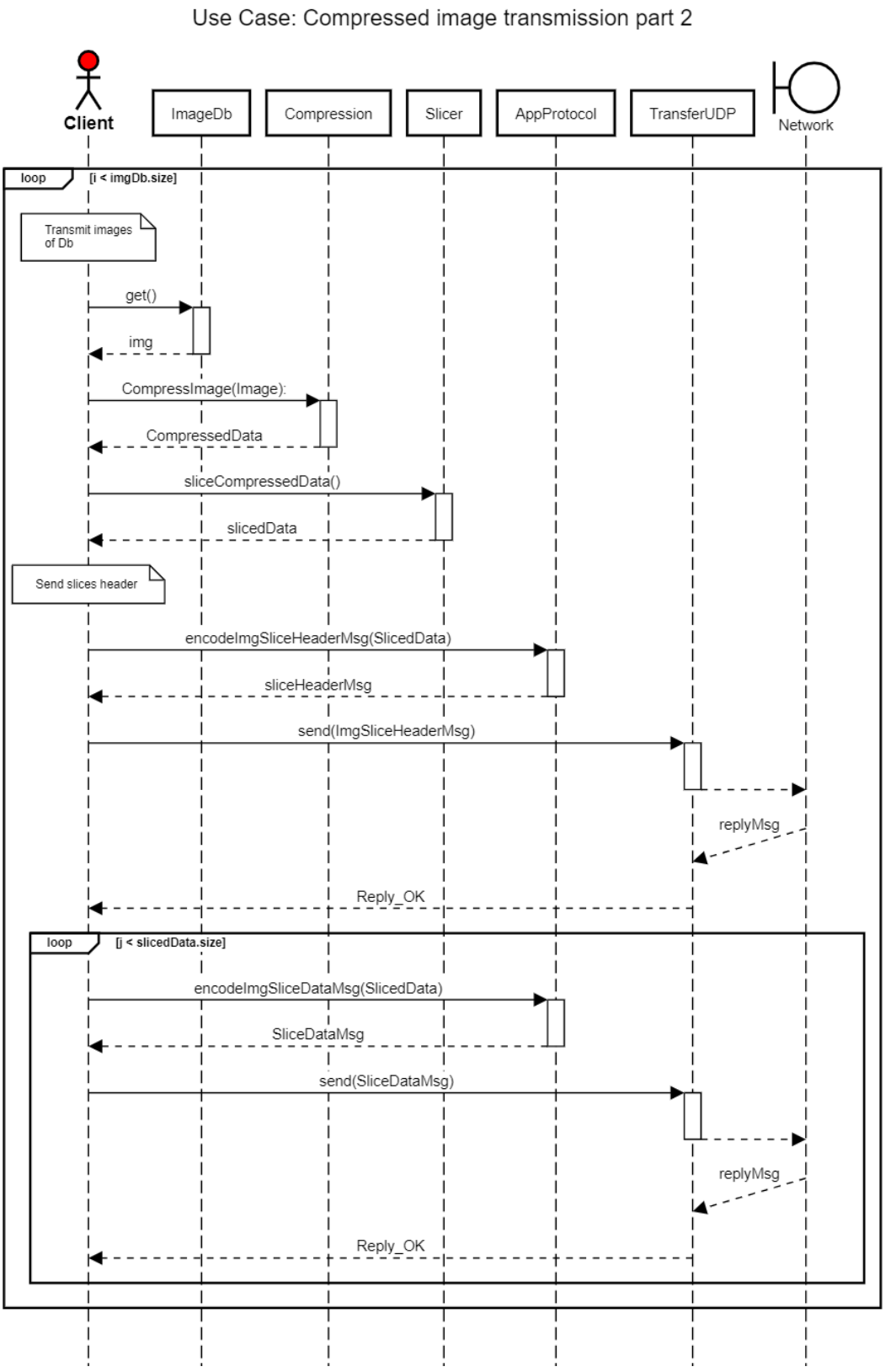

Figure 16: Compressed image transmission detailed diagram part 2

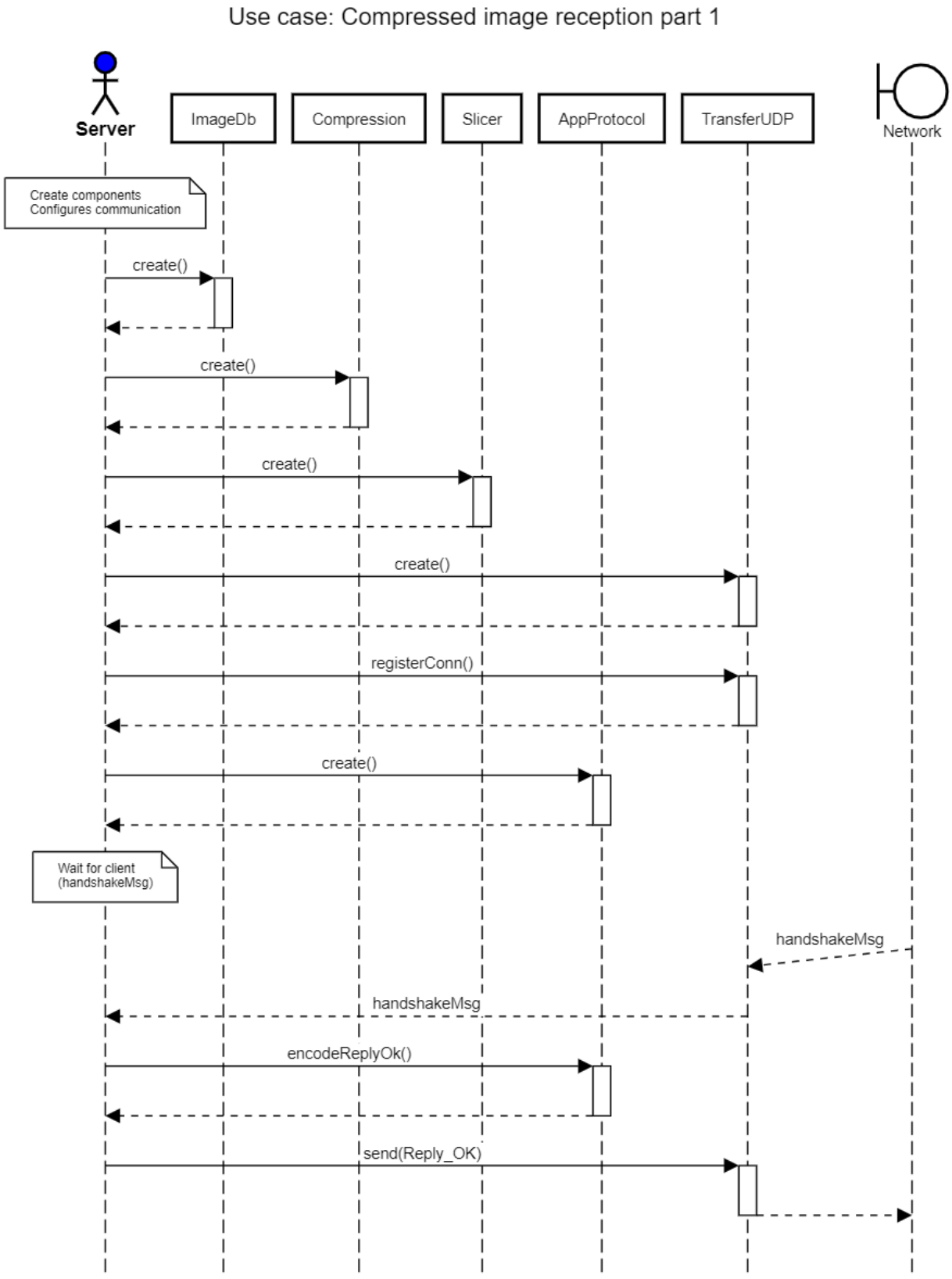

Figure 16: Compressed image reception detailed diagram part 1

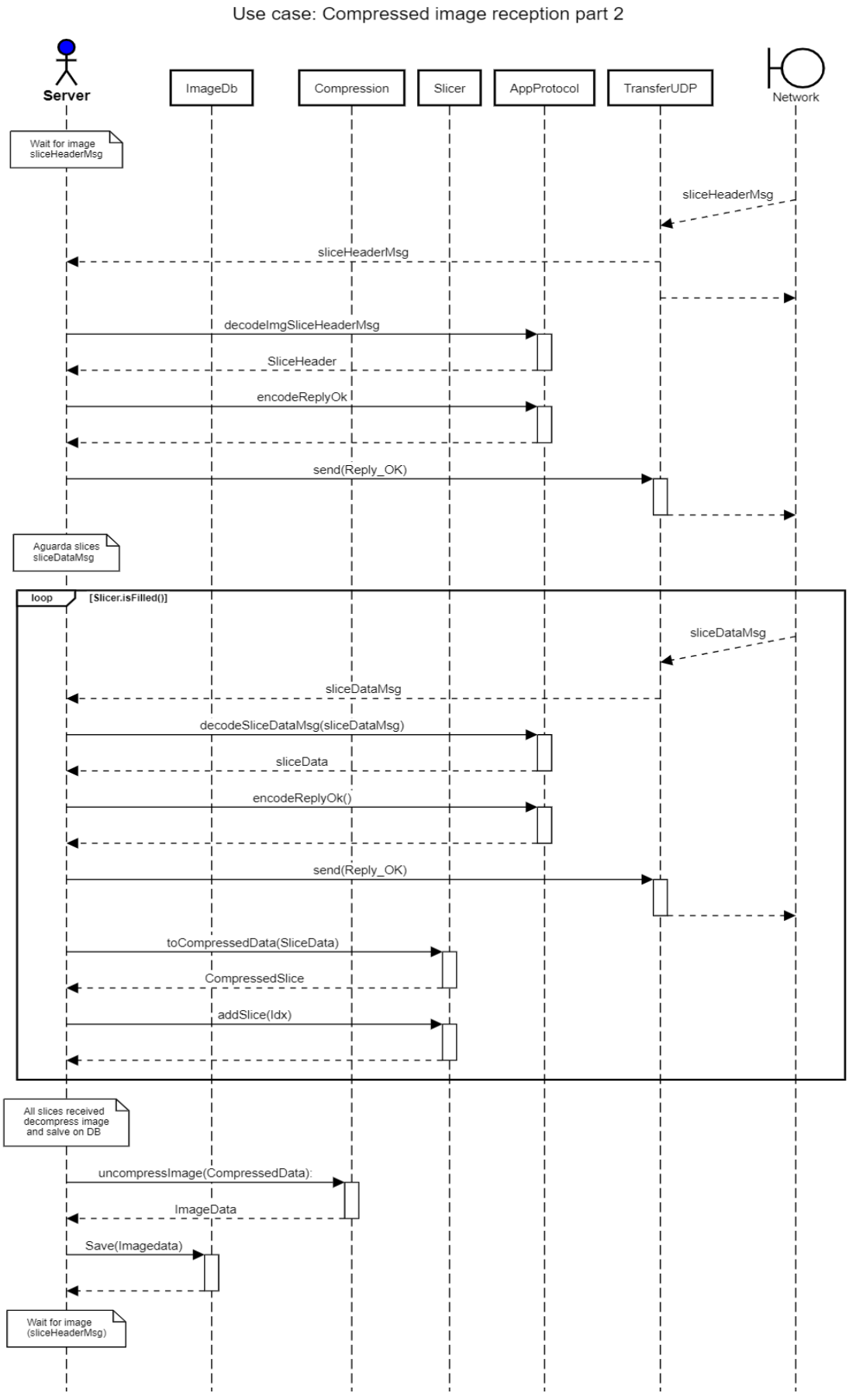

Figure 17: Compressed image reception detailed diagram part 2