Wanhive Systems Private Limited	**WANHIVE/JANUS/1001/PROTO/02**
Request Ref No. **N/A**	Dated: 10/23/2019

Communication Protocol

WANHIVE PROTOCOL

Document Version *1.0.0*

Created on October 23, 2019

(CC BY-SA 4.0)

FOR:	**CIRCULATED BY:**
Application developers	Wanhive Systems Private Limited www.wanhive.com

Document Control

Title	Specification of Wanhive protocol
Softcopy Name	wanhive protocol v1.0.0
Document Ref. No.	WANHIVE/JANUS/1001/PROTO/02
Version/Status	Version 1.0.0
Author	Amit Kumar
Distribution Date	October 23, 2019
No of Pages	17

Document Distribution

Company	**Name**	**Purpose**	**Number of Copies**
Total			

Document Approval

Author	**Approved By**	**Released By**
Amit Kumar (amit@wanhive.com)		

Document References

SL	**Name & Ref. Number**	**Softcopy**	**Hardcopy**

Revision History

Version	Change By	Comments	Date	Pages
0.1.0	Amit Kumar (amit@wanhive.com)	Initial draft	11/09/2018	19
1.0.0	Amit Kumar (amit@wanhive.com)	Updated to the new header specification	10/23/2019	17

Glossary- Read First

Byte	8 bits
IoT	Internet of things
IP	Internet protocol
TCP	Transport control protocol

Contents

This page intentionally left blank

1 Introduction and Purpose

Purpose of this document is to describe the "Wanhive protocol" (hereafter referred to as **PROTOCOL**), a binary message-based streaming and multicasting protocol for the implementation of the internet of things (IoT) and telemetry applications.

This document is not just an enumeration of features but also contains several informative sections designed to help both the developers and the stakeholders.

2 Overview

Use PROTOCOL to build IoT and telemetry applications over public IP networks with a small code footprint, high throughput, near-real-time response, and mobility.

2.1 Terminology

2.1.1 Connection

PROTOCOL uses TCP/IP to establish a connection and to transport data between the programs.

2.1.2 Message

A message is a packet of data carried by the PROTOCOL. Each message constitutes a quantum of information.

2.1.3 MTU (Maximum transmission unit)

MTU is the largest message size measured in bytes.

2.1.4 Communication

Communication is the process of sending and receiving messages.

2.1.5 Hub

Hub is a program or device that participates in communication. Capabilities of a hub:

1. Connect to another hub.
2. Accept incoming connections from another hub.
3. Create a message.
4. Send a message to a connection.
5. Accept a message from a connection.
6. Forward a message to its destination via an intermediary.
7. Answer: send out the response to a message.
8. Consume: accept a message but do not send back a response.
9. Disconnect from another hub.

2.1.6 Network

A network consists of several hubs that can communicate with one another.

2.1.7 Routing

Routing is the process of forwarding a message through the network until it is delivered.

2.1.8 Overlay Network

An overlay network is primarily responsible for routing the messages. A hub in the overlay network accepts and maintains multiple incoming connections.

2.1.9 Server

A server is a hub which constitutes the overlay network.

2.1.10Client

A client is a hub that doesn’t participate in message routing. A client generally doesn’t accept incoming connections.

2.1.11Identity

The identity of a hub is its unique numerical identifier to distinguish it from the rest of the network.

2.1.12Identification

Identification is the process of recognizing a hub.

2.1.13Authentication

Authentication is the process of confirming the identity of a hub.

2.1.14Authorization

Authorization is the process of assigning access rights and privileges to a hub.

2.1.15Registration

Registration establishes a functional contract between a hub and the network.

2.1.16Authentication hub

These are specialized hubs that facilitate identification, authentication, and authorization processes in the network.

2.2 Design goals

1. Simple
2. Small
3. Useful

2.2.1 Attributes

1. **Binary protocol**: Intended to be read by computer programs and devices rather than human beings.
2. **Multicasting**: Deliver a message to multiple destinations.
3. **Streaming**: In-order delivery of messages in an end-to-end fashion.

3 Message format

A message contains a fixed-length header and variable-length payload organized in the following order. Index offsets are in bytes (8 bits) unless otherwise mentioned.

Table 1

Parts	Header	Payload
Index	0-31	32-<MTU-1>
Total	Min: 32 bytes, Max: MTU bytes	

IMPLEMENTATION NOTES

1. MTU in the reference implementation is 1024 bytes.

3.1 Header

The message header is as follows:

Table 2

Fields	Label	Source	Destination	Length	Sequence number	Session	Command	Qualifier	Status
Index	0-7	8-15	16-23	24-25	26-27	28	29	30	31
Bytes	8	8	8	2	2	1	1	1	1
Total		0-31 (total size: 32 bytes)							

1. **Label**: Implementation dependent.
2. **Source**: Identity of the message's origin. It is a 64-bit unsigned integer packed in network byte order.
3. **Destination**: Identity of the message's target. It is a 64-bit unsigned integer packed in network byte order.
4. **Length**: Total length of the message in bytes. It is a 16-bit unsigned integer packed in network byte order.
5. **Sequence number**: Sequence number of the message. In the reference implementation, response to a message usually carries the same sequence number as the request. It is a 16-bit unsigned integer packed in network byte order.
6. **Session**: It is an 8-bit unsigned integer.
   1. Messages are delivered in-order if they have the same source and session.
   2. Messages are interleaved and delivered out-of-order in all the other cases.
7. **Command**: the command of the message. It is an 8-bit unsigned integer.
   1. The best analogy is a *class declaration* in OOP.
8. **Qualifier**: the qualifier of the message. It is an 8-bit integer.
   1. The best analogy is a *public method declaration* in OOP.
9. **Status**: the status of the message. It is an 8-bit unsigned integer.
   1. The best analogy is the *exit code* returned by a program.

3.1.1 Header data

The following table lists the header fields with their permissible values in their order of occurrence:

Table 3

Field	**Minimum value**	**Maximum value**
Label	-	-
Source	0	9223372036854775807
Destination	0	9223372036854775807

Field	**Minimum value**	**Maximum value**
Length	32	Minimum Of (MTU, 65535)
Sequence number	0	65535
Session	0	255
Command	0	255
Qualifier	0	255
Status	0	255

IMPLEMENTATION NOTES

1. The overlay network correctly sets the "source" field during routing in the reference implementation. In short, a client can always trust the source field in an incoming message's header.
2. The "source" field with a value of 0 has a special meaning in the reference implementation. It means that the message is a communication from the server.
3. The "destination" field with a value of 0 has a special meaning in the reference implementation. It means that the client wishes to communicate with the server. It also means that the message is multicast.
4. The "sequence number" field with a value of 0 has a special meaning in the reference implementation. It advises the recipient to ignore the sequence number of the incoming message during validation.
5. Use the "session" field as
   1. The topic identifier during publish-subscribe-based multicasting.
   2. The stream identifier during message-based multistreaming.

3.2 Payload

The total length of a message cannot exceed the MTU. Hence the maximum number of bytes in the payload is restricted to (MTU - HEADER_SIZE).

The minimum payload size is zero (0) bytes. In this case, the message carries a header but no payload.

3.2.1 Payload data

PROTOCOL doesn't assign any special meaning to the payload data and treats the payload as an array of unsigned bytes. Here are the two basic operations on the payload data:

1. **Serialization**: Write application data into the message payload in a portable format.
2. **Deserialization**: Read a chunk of data (bytes) from the payload and transform it into a machine-readable format.

Reference implementation has built-in support for serialization and de-serialization of certain data types. Collectively they form the **Standard data types**.

3.2.2 Standard data types

Reference implementation handles the following data types transparently.

Table 4

Type	**Size (bytes)**	**Serialized format**
Unsigned long integer	8	Serialized in network byte order
Unsigned integer	4	Serialized in network byte order

Type	**Size (bytes)**	**Serialized format**
Unsigned short integer	2	Serialized in network byte order
Unsigned byte	1	
Double precision double	8	Encoded in IEEE-754 format, serialized in network byte order. Sign is preserved.
Single precision float	4	Encoded in IEEE-754 format, serialized in network byte order. Sign is preserved.
Half precision float	2	Encoded in IEEE-754 format, serialized in network byte order. Sign is preserved.
String	Variable	A C-style ASCII string. String is prefixed with 16-bit unsigned value representing its length, NUL terminator is not included.
Blob	Variable	A variable length array of unsigned bytes. Blob is prefixed with 16-bit unsigned value representing its length.

IMPLEMENTATION NOTES

1. Standard data types listed above are neither limiting nor mandatory.

4 Message signing and verification

A message may be digitally signed and verified using asymmetric cryptography. Reference implementation signs and verifies X509_SIG ASN1 object inside PKCS#1 padded 2048 bit RSA encryption.

4.1 Message signing

The message signing process:

1. **Input**: A message.
2. **Output**: A digitally signed message.
3. **Precondition**: The length of the signed message can't exceed the MTU.
4. **Postcondition**: Update the message length to <Original message length>+<Digital signature length>. Use a private key to compute a signature for the updated message. Append the signature at the end of the payload.

4.2 Message verification

The message verification process:

1. **Input**: Digitally signed message.
2. **Output**: Accept or reject the message's claim of authenticity.
3. **Precondition**: Message's payload must contain a valid digital signature in its tail.
4. **Postcondition**: Use a public key to verify the authenticity of the message.

5 Special Messages

In general, the network doesn't assign special meaning to the messages and routes them without prejudice. However, the proper functioning of the network requires specialized messages. Most of them have been listed in this section.

IMPLEMENTATION NOTES

1. These specialized messages follow a request-response pattern. A hub sends a request to another hub and receives a response synchronously.
2. The response to a failed request does not carry any payload.
3. The status field takes one the following values:
    1. Request (127)
    2. Success (1)
    3. Failure (0)

5.1 GetKey message

Establish a unique session with a server to guard the registration process against replay and man-in-the-middle attacks. The response carries the **session key**.

5.1.1 Header

Table 5

Field	Request	Response
Label	-	-
Source	Optional, ignored	0
Destination	Optional, ignored	0
Length	32/96/288	96/160/416
Sequence number	Optional	Same as in the request
Session	Optional	Same as in the request
Command	1	1
Qualifier	1	1
Status	Optional, ignored	0 on failure, 1 on success

5.1.2 Payload

Table 6

Request 1	Request 2	Request 3	Response to request 1	Response to Request 2	Response to Request 3
	64-byte challenge key	256-byte: 64 byte challenge key encrypted with server's public key	64-byte session key	64-byte challenge key	64-byte challenge key
				64-byte session key	64-byte session key
					256 byte digital signature

IMPLEMENTATION NOTES

1. Request formats 1 and 2 are safe for secure connections (SSL).
2. Request format 3 uses costly cryptographic operations resulting in performance degradation.

5.2 Registration message

Register on a server with the given identity. The **session key** is the one that was retrieved from the GetKey response.

5.2.1 Header

Table 7

Field	Request	Response
Label	-	-
Source	Desired identity	0
Destination	Optional, ignored	0
Length	32/96/352	32
Sequence number	Optional	Same as in the request
Session	Optional	Same as in the request
Command	1	1
Qualifier	0	0
Status	Optional, ignored	1 on success, connection terminated on failure

5.2.2 Payload

Table 8

Request 1	Request 2	Request 3	Response
	64-byte session key	64-byte session key	
		256 byte digital signature	

IMPLEMENTATION NOTES

1. A private key is required to sign the registration request.
2. A hub can ask an authentication hub to sign the request after completing the identification and authentication steps.
3. Request formats 1 and 2 are completely insecure.

5.3 FindRoot message

Find a server in the overlay network to connect.

5.3.1 Header

Table 9

Field	Request	Response
Label	-	-
Source	Optional, ignored	0
Destination	0	0
Length	40	48
Sequence number	Optional	Same as in the request
Session	Optional	Same as in the request
Command	1	1
Qualifier	2	2
Status	127	1 on success, always succeeds

5.3.2 Payload

Table 10

Request	Response
8-byte identity for which the root is being searched	8-byte identity from the request
	8-byte identity of the root server

IMPLEMENTATION NOTES

1. Reference implementation doesn't allow the clients to connect with arbitrary servers.

5.4 Identification message

Identify the self at the authentication hub. The request succeeds if the desired identity exists in the network's database. This request succeeds only once over a connection.

5.4.1 Header

Table 11

Field	Request	Response
Label	-	-
Source	Desired identity	0
Destination	0	0
Length	At least 32	At least 32
Sequence number	Optional	Same as in the request
Session	Optional	Same as in the request
Command	0	0
Qualifier	1	1
Status	127	0 on failure, 1 on success

5.4.2 Payload

Table 12

Request	Response
Variable length nonce (A)	2-bytes salt length
	2-bytes nonce length
	Variable length salt (S)
	Variable length nonce (B)

5.5 Authentication message

Authenticate the self at the authentication hub (*actually mutual authentication*). The request succeeds if the proof of identity matches the one in the network's database. Always fails if the previous identification request failed.

5.5.1 Header

Table 13

Field	Request	Response
Label	-	-
Source	Optional, ignored	0
Destination	0	0

Field	**Request**	**Response**
Length	At least 32	At least 32
Sequence number	Optional	Same as in the request
Session	Optional	Same as in the request
Command	0	0
Qualifier	2	2
Status	127	0 on failure, 1 on success

5.5.2 Payload

Table 14

Request	**Response**
Variable length proof of identity (M)	Variable length proof of the host (AMK)

5.6 Authorization message

Request the authentication hub to sign an arbitrary message. The only practical use is to get the registration message signed. Always fails if the previous authentication request failed.

5.6.1 Header

Table 15

Field	**Request**	**Response**
Label	-	-
Source	Optional, ignored	Identity of the client
Destination	Optional, ignored	Same as in the request
Length	At least 32	At least 32
Sequence number	Optional	Same as in the request
Session	Optional	Same as in the request
Command	Not 0	Same as in the request
Qualifier	Ignored	Same as in the request
Status	127	0 on failure, 1 on success

5.6.2 Payload

Table 16

Request	**Response**
Variable length	Digital signature appended at the end of the payload.

IMPLEMENTATION NOTES

1. The request will fail if there is insufficient space in the message for appending the digital signature.

5.7 Publish message

Publish a message to a topic on the server. The publisher must be registered on the server.

5.7.1 Header

Table 17

Field	**Request**	**Publish**
Label	-	-
Source	Optional, ignored	Identity of the publisher
Destination	0	0
Length	At least 32	Same as in the request
Sequence number	Optional	Same as in the request
Session	0-255 (topic identifier)	Same as in the request
Command	2	2
Qualifier	0	0
Status	127	1

IMPLEMENTATION NOTES

1. The message is delivered to all the subscribers except the source of the message. No response is sent to the source of the message.
2. Messages published to a topic by the same client are delivered in-order.
3. Messages may be lost or dropped due to network issues.

5.8 Subscribe message

Subscribe to a topic on the server. All the messages published on the topic will be delivered to its subscribers. The subscriber must be registered on the server.

5.8.1 Header

Table 18

Field	**Request**	**Response**
Label	-	-
Source	Optional, ignored	0
Destination	0	0
Length	32	32
Sequence number	Optional	Same as in the request
Session	0-255 (topic identifier)	Same as in the request
Command	2	2
Qualifier	1	1
Status	127	0 on failure, 1 on success

IMPLEMENTATION NOTES

1. A client can subscribe to multiple topics on the server.
2. A topic can have multiple subscribers.

5.9 Unsubscribe message

Unsubscribe from a topic on the server. Multicast messages in transit may still get delivered. The caller must be registered on the server.

5.9.1 Header

Table 19

Field	**Request**	**Response**
Label	-	-
Source	Optional, ignored	0

Field	**Request**	**Response**
Destination	0	0
Length	32	32
Sequence number	Optional	Same as in the request
Session	0-255 (topic identifier)	Same as in the request
Command	2	2
Qualifier	2	2
Status	127	1 on success, never fails

6 Source and Destination

PROTOCOL supports message-based end-to-end multi-streaming, in which case each message is delivered to a specific destination.

PROTOCOL also supports publish-subscribe based multicasting, in which case a message can be delivered to multiple destinations.

The overlay network always sets the “source” field in the message header correctly. Conventionally, the clients set the source field in the message header to zero (0).