taifatech TF-6x0
Viewer/Extender Protocol Specification

Revision 1.00
2010-04-01

Revision History

Revision Number	**Date**	**Description**
Rev 1.00	2010/04/01	Preliminary

I. Introduction

The *taifatech Viewer/Extender Protocol* (abbreviated as **TFVEP**) specifies Layer4 protocol over wireless or Ethernet networks. The TFVEP is defined for controlling and communicating between TF630/TF600 devices of taifatech. This protocol is based on the request-response communication scheme. Sender sends commands to target device. The target device receives these command packets, interprets these packets and replies information and status by sending back response packets to the Sender.

In this document, several words are used to signify the requirements of the specification. These words are often capitalized.

MUST

This word, or the adjective "required", means that the definition is an absolute requirement of the specification.

MUST NOT

This phrase means that the definition is an absolute prohibition of the specification.

SHOULD

This word, or the adjective "recommended", means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications must be understood and carefully weighed before choosing a different course.

MAY

This word, or the adjective "optional", means that this item is one of an allowed set of alternatives. An implementation which does not include this option MUST be prepared to interoperate with another implementation which does include the option.

II. Packet Format

There are several types of command sets in TFVEP; those command sets are separated by command type. There are many commands in each command set; those commands are separated by OPCODE. For each command, there are two types of packets, command packet and response packet. A summary of the packet structure is shown below (The data order of packet payload must be in *Big Endian* format.).

Layer4 TFVEP Packet Format

Destination Address	Source Address	Ether Type (**0x0800**)	IP/ UDP Header (Destination Port=**48689**)	TFVEP Message

- **Destination Address (6 bytes)**
  The Destination Address can be network broadcast MAC address or individual device's MAC address. Initially, sender uses broadcast packet to try to connect to a receiver. If sender receives response from receiver, sender uses receiver's MAC address for subsequent packet transmissions.
- **Source Address (6 bytes)**
  The Source Address specifies the packet sender's MAC address. It must be individual address.
- **Ether Type (2 bytes)**
  For Layer4 TFVEP, the Ether type is standard IP packet (0x0800).
- **UDP destination Port (2 bytes)**
  For Layer4 TFVEP, the UDP destination port has to be **48689**.

TFVEP Message Format

Signature	Product ID	Random Code	Sequence ID	COMMAND TYPE	OPCODE	LENGTH	DATA
4 Bytes	2 Bytes	2 byte	2 Bytes	2 Bytes	2 Bytes	2 Bytes	n Bytes

- **Signature (4 bytes)**

  For TFVEP, the signature has to be **0x5446265A ("TF6z")**.

- **Sequence ID (2 bytes)**

  The Sequence ID is used to keep the sequence number of the packets during handshaking. For example, receivers might receive a packet with the same sequence number of previously received packet if sender resends the same packet. This Sequence ID is used to identify the packet duplication or loss. When receiver replies response packet, the Sequence number must be copied to the field of Sequence ID in the response packet.

- **COMMAND TYPE (2 bytes)**

  It is used to specify the type of command packet. If one of most-significant four bits, bit-15, bit-14, bit-13 and bit-12, is set to “1”, it means the packet is a response packet. If bit15 is set, it means the execution of command is success. If bit14 is set, it means the execution of command failed. If bit13 is set, it means the command type is not supported. If bit12 is set, it means the OPCODE is not supported. For a command packet, these bits must be zero.

B15	B14	B13	B12	B11 B0
S	**RJ**	**WR**	**NS**	**COMMAND**

  S: Indicator bit for successfully executing command.

  RJ: Indicator bit for receiver rejects to execute command from sender.

  WR: Indicator bit for receiver is waiting for retry.

  NS: Indicator bit for unsupported COMMAND/OPCODE.

  COMMAND: TFVEP command.

- **OPCODE (2 bytes)**

  The OPCODE field specifies the real operation of command packet. The description of OPCODE values are defined in the next section.

- **LENGTH (2 bytes)**

  The Length field specifies the total byte length of Data field. The maximum length must not exceed the limitation of Ethernet packet.

- **DATA (1-n bytes)**

  The length of DATA field depends on the command type and the execution command. For detail information, refer the descriptions of following sections.

III. TFVEP Commands

TFVEP commands are classified into three types according to their functionality. Each command has several operation codes (OPCODEs) for different requirements. The TFVEP commands summary below could help you to browse the TFVEP commands and relative operations.

Summary of TFVEP Commands

Command Type	**Opcode**	**Description**
DISCOVERY (0x0001)	OPCODE_TF6x0_DISCOVER_ANNOUNCE (0x0100)	Senders (Viewers) send out this packet to inform Receivers (Extenders); The receivers must store the senders' information and not reply it.
	OPCODE_TF6x0_DISCOVER (0x0101)	Senders (Viewer) send out this packet to inform Receivers (Extenders); The receivers must store the senders' information and reply it.
	OPCODE_TF6x0_DISCOVER_ASK_REPLY_ONLY (0x0102)	Senders (Viewer) send out this packet to inform Receivers (Extenders); The receivers must reply it and not store the senders' information
	OPCODE_TF6x0_DISCOVER_AND_RESET (0x0103)	Senders (Software Extender) send out this packet to reset Receivers (Viewers) connection and force Receivers enter discovery mode. And the Viewers will reply reject packet if it is not in connect table.
	OPCODE_TF6x0_SEARCH (0x0104)	Senders (Viewers/Extenders) send out this packet to search dedicated Receivers (Viewers/Extenders).
CONNECTING (0x0002)	OPCODE_TF6x0_CONNECTING (0x0201)	Senders(Viewers) send out this packet to connect with Receivers(Extenders); The receivers must reply it. If the receivers' status is connected, it will send back wait retry reply; if the receivers' status is not connected and the multicast mode enabled, it will send back listen only allow reply; if the connect table is full, it will send back reject reply; if all condition is ok, it will send back connecting reply. If the

		viewers get the correct connecting reply packet, viewers will go on to send out connect password request packet.
	OPCODE_TF6x0_DISCONNECTING (0x0202)	Senders(Viewers/Extenders) send out this packet to disconnect with Receivers(Extenders/Viewers); The receivers must reply it. If the connection is in the connected table, Receivers will send out the disconnect reply to Senders; if the connection is not in the connected table, Extenders will send data lost packet back to the Viewers.
	OPCODE_TF6x0_CONNECTING_INFORMATION (0x0203)	N/A
	OPCODE_TF6x0_CONNECTING_PASSWORD (0x0204)	Got connecting reply from Extenders, Senders(Viewers) send out this packet with connecting username to receivers(Extenders); The receivers must reply. If the connection is not allowed, the Extenders will send out a reject packet; If the connection is allowed, the Extenders send out a reply packet to viewers and send out status announce packet.
	OPCODE_TF6x0_CHANGE_PASSWORD (0x0205)	N/A
	OPCODE_TF6x0_LISTEN_ANNOUNCE (0x0208)	N/A
	OPCODE_TF6x0_LISTEN_ONLY_ALLOW (0x0209)	Senders(Extenders) send out this packet to let the Receivers(Viewers) be listen mode. The receivers change their status to listen mode and need not to reply.
	OPCODE_TF6x0_LISTEN_TYPE_CHANGED (0x020A)	N/A
OPERATION (0X0003)	OPCODE_TF6x0_ALIVE_REPORT (0X0301)	Senders(Viewers) send out alive report to receivers(Extenders). If the Receivers' status is normal, it need not to reply; if the Receivers' status changed, it will reply data lost report to Viewers.
	OPCODE_TF6x0_DATALOST_REPORT (0X0302)	Senders(Viewers) send out data lost report to receivers(Extenders).If the Receivers' status is normal, it need to reply alive report; else it will reply data lost report.

	OPCODE_TF6x0_SOURCE_STATUS_ANNOUNCE (0x0303)	Senders(Extenders) send out this packet to Receivers (Viewers) to announce status. And the Viewer need not to reply it.
	OPCODE_TF6x0_EDID_DATA (0X0304)	N/A
	OPCODE_TF6x0_IRPULSE_DATA (0X0305)	Senders(Viewers) send out this packet to give IR pulse data, Receivers(Extenders) will transmit the pulse data. The receivers no need to rely.
	OPCODE_TF6x0_PICTURE_STATUS (0X0306)	N/A
	OPCODE_TF6x0_VGA_POSITION_REQUEST (0x0307)	N/A
	OPCODE_TF6x0_PICTURE_BANDWIDTH_REQUEST (0x0308)	N/A
	OPCODE_TF6x0_MULTI_UNI_CAST_REQUEST (0x0309)	N/A
	OPCODE_TF6x0_STREAM_TYPE_REQUEST (0x030a)	N/A
	OPCODE_TF6x0_LOCK_PCA_REQUEST (0x030b)	N/A
	OPCODE_TF6x0_CONTROL_MESSAGE (0x030f)	N/A
	OPCODE_TF6x0_ADBUS_64BYTES (0x0310)	Please refer to OPCODE_TF6x0_ADBUS_64BYTES_AG
	OPCODE_TF6x0_ADBUS_64BYTES_AG (0x0311)	Senders(Viewers/Extenders)send out this packet to give ADBus infos, and Receivers need not to reply.
	OPCODE_TF6x0_ADBUS_GPIO5 (0x0312)	Senders(Viewers) send out this packet to announce AD bus init status. The receivers no need to reply.
	OPCODE_TF6x0_HDMI_SINK_STATUS (0x0601)	Senders(Viewers) send out this packet to sync sink status. Receivers get the packet and reply ack status.
	OPCODE_TF6x0_HDMI_CEC_MSG (0x0602)	Senders(Viewers/Extenders) send out this packet to send CEC msg, if ack is false, the receivers will send CEC msg packet back.

Command Type	Opcode	Description
NETWORK DETECTING (0x0004)	OPCODE_TF6x0_NETWORK_DETECTING (0x0401)	N/A
	OPCODE_TF6x0_NETWORK_DETECTING_RESULT (0x0402)	N/A
	OPCODE_TF6x0_NETWORK_DETECTING_DATA (0x0403)	N/A
	OPCODE_TF6x0_NETWORK_DETECTING_PASSOWRD (0x0404)	N/A
PS2 REPORT (0x0005)	OPCODE_TF6x0_PS2_MS_ALL (0x0500)	Senders(Extenders) send out this packet to give mouse infos. The Receivers need to change its mouse parameters and need not to reply.
	OPCODE_TF6x0_PS2_MS_TYPE (0x0501)	Senders(Viewers) send out this packet to give mouse type. The Receivers will change its mouse type and reset mouse, no need to reply.
	OPCODE_TF6x0_PS2_MS_SCALE (0x0502)	Senders(Extenders) send out this packet to give mouse scale values. The Receivers need to change its mouse scale value and need not to reply.
	OPCODE_TF6x0_PS2_MS_RATE (0x0503)	Senders(Extenders) send out this packet to give mouse rate values. The Receivers need to change its mouse rate value and need not to reply.
	OPCODE_TF6x0_PS2_MS_RESOLUTION (0x0504)	Senders(Extenders) send out this packet to give mouse resolution values. The Receivers need to change its mouse resolution value and need not to reply.
	OPCODE_TF6x0_PS2_MS_RESET (0x0505)	Senders(Extenders) send out this packet to reset mouse. The Receivers need to reset its mouse and need not to reply.
	OPCODE_TF6x0_PS2_KB_ALL (0x0508)	Senders(Extenders) send out this packet to give keyboard infos. The Receivers need to change its keyboard parameters and need not to reply.
	OPCODE_TF6x0_PS2_KB_ID (0x0509)	Senders(Viewers) send out this packet to give keyboard ID. The Receivers will change its keyboard ID, and no need to reply.
	OPCODE_TF6x0_PS2_KB_LED (0x050A)	Senders(Extenders) send out this packet to give keyboard led values. The Receivers need to change its keyboard led value

		and need not to reply.
	OPCODE_TF6x0_PS2_KB_SET_SCANCODE (0x050B)	Senders(Extenders) send out this packet to give scan code values. The Receivers need to change its keyboard scan code value and need not to reply.
	OPCODE_TF6x0_PS2_KB_GET_SCANCODE (0x050C)	N/A
	OPCODE_TF6x0_PS2_KB_RATE (0x050D)	Senders(Extenders) send out this packet to give keyboard rate values. The Receivers need to change its keyboard rate value and need not to reply.

IV. Command Details

Discovery Command Set

MNEMONIC	VALUE
CMDTYPE_TF6x0_DISCOVERY	0x0001

01. OPCODE_TF6x0_DISCOVER_ANNOUNCE

● **Purpose**

This command is used to announce *Viewer* devices connected to network. The *Extenders* could get *Viewers'* information from this op code including Product type, Firmware version, Group name, Machine name and Expected product type.

● **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0001	0x0100	**L**	**D**

L: sizeof (NET_PROTOCOL_DISCOVER_REQUEST_MSG).

D: structure of NET_PROTOCOL_DISCOVER_REQUEST_MSG

● **Response Packet**

NO NEED TO REPLY

Relevant:

Refer to NET_PROTOCOL_DISCOVER_ANNOUNCE_MSG

02. OPCODE_TF6x0_DISCOVER

● **Purpose**

Viewer devices use this command to discover how many *Extender* devices connected to network. The *Extender* devices received this op code must reply the corresponding message to *Viewer* devices and store the Viewers' information

● **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA

0x0001	0x0101	**L**	**D**

L: Size of (NET_PROTOCOL_DISCOVER_REQUEST_MSG)

D: Structure of NET_PROTOCOL_DISCOVER_REQUEST_MSG.

Relevant:

Refer to NET_PROTOCOL_DISCOVER_REQUEST_MSG

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8001(ACK)	0x0101	**L**	**D**
0x4001(NACK)	0x0101	**L**	**D**

L: Size of (NET_PROTOCOL_DISCOVER_REPLY_MSG)

D: Structure of NET_PROTOCOL_DISCOVER_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_DISCOVER_REPLY_MSG

03. **OPCODE_TF6x0_DISCOVER_ASK_REPLY_ONLY**

- **Purpose**

Viewer devices use this command to discover how many *Extender* devices connected to network. The *Extender* devices received this op code must reply the corresponding message to *Viewer* devices, but not store the Viewers' information

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0001	0x0102	**L**	**D**

L: Size of (NET_PROTOCOL_DISCOVER_REQUEST_MSG)

D: Structure of NET_PROTOCOL_DISCOVER_REQUEST_MSG.

Relevant:

Refer to NET_PROTOCOL_DISCOVER_REQUEST_MSG

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8001(ACK)	0x0102	**L**	**D**
0x4001(NACK)	0x0102	**L**	**D**

L: Size of (NET_PROTOCOL_DISCOVER_REPLY_MSG)

D: Structure of NET_PROTOCOL_DISCOVER_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_DISCOVER_REPLY_MSG

04. **OPCODE_TF6x0_DISCOVER_AND_RESET**

- **Purpose**

  Software Extender could use this command to reset the connection table of Viewer devices and force Viewer devices enter discovery mode.

  NOTE: Discovery mode means the *Viewer* devices send out the discovery message continually.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0001	0x0103	**X**	**X**

X: don't care

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x4001(NACK)	0x0103	**NA**	**NA**

05. **OPCODE_TF6x0_SEARCH**

- **Purpose**

  The Viewer/Extender devices could use this command to search for specified the Extender/Viewer devices.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0001	0x0104	**NA**	**NA**

NA: Not Available

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8001(ACK)	0x0003	**NA**	**NA**
0x4001(NACK)	0x0003	**NA**	**NA**

NA: Not Available.

06. **OPCODE_TF6x0_CONNECTING**

- **Purpose**

  The Viewer devices could use this command to connect with the Extender devices.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0002	0x0201	**L**	**D**

L: Size of (NET_PROTOCOL_CONNECTING_REQUEST_MSG)
D: Structure of NET_PROTOCOL_CONNECTING_REQUEST_MSG.
Relevant:
Refer to NET_PROTOCOL_CONNECTING_REQUEST_MSG

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8002(ACK)	0x0201	**L1**	**D1**
0x2002	0x0201	**L2**	**D2**
0x8002	0x0209	**L3**	**D3**
0x4002	0x0201	**L2**	**D2**

L1: Size of (NET_PROTOCOL_CONNECTING_REPLY_MSG)
D1: Structure of NET_PROTOCOL_CONNECTING_REPLY_MSG.
Relevant:
Refer to NET_PROTOCOL_CONNECTING_REPLY_MSG
L2: Size of (NET_PROTOCOL_REPLY_MSG)
D2: Structure of NET_PROTOCOL_REPLY_MSG.
Relevant:
Refer to NET_PROTOCOL_REPLY_MSG
L3: Size of (NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG)
D3: Structure of NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG.
Relevant:
Refer to NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG

07. **OPCODE_TF6x0_DISCONNECTING**

- **Purpose**

Both Extender and Viewer may send this command to do a disconnecting announce or request.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0002	0x0202	**L**	**D**

L: Size of (NET_PROTOCOL_DISCONNECTING_REQUEST_MSG)

D: Structure of NET_PROTOCOL_DISCONNECTING_REQUEST_MSG.

Relevant:

Refer to NET_PROTOCOL_DISCONNECTING_REQUEST_MSG

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8002(ACK)	0x0202	**L1**	**D1**
0x8003(ACK)	0x0302	**L2**	**D2**

L1: Size of (NET_PROTOCOL_DISCONNECTING_REPLY_MSG)

D1: Structure of NET_PROTOCOL_DISCONNECTING_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_DISCONNECTING_REPLY_MSG

L2: Size of (NET_PROTOCOL_DATALOST_REPLY_MSG)

D2: Structure of NET_PROTOCOL_DATALOST_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_DATALOST_REPLY_MSG

08. **OPCODE_TF6x0_CONNECTING_INFORMATION**

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0203		

NA: Not Available

- **Response Packet**

NA: Not Available.

09. **OPCODE_TF6x0_CONNECTING_PASSWORD**

- **Purpose**

  Viewer uses this command to login and get permission of receiving A/V streaming from Extender.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8002	0x0204	**L**	**D**

L: Size of (NET_PROTOCOL_ CONNECTING_PASSWORD_MSG)

D: Structure of NET_PROTOCOL_CONNECTING_PASSWORD_MSG.

Relevant:

Refer to NET_PROTOCOL_CONNECTING_PASSWORD_MSG

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8002(ACK)	0x0204	**L**	**D**

L: Size of (NET_PROTOCOL_CONNECTING_PASSWORD_REPLY_MSG)

D: Structure of NET_PROTOCOL_CONNECTING_PASSWORD_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_CONNECTING_PASSWORD_REPLY_MSG

10. **OPCODE_TF6x0_CHANGE_PASSWORD**

- **Purpose**

  N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0205		

NA: Not Available

- **Response Packet**

NA: Not Available.

11. OPCODE_TF6x0_LISTEN_ANNOUNCE

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0208		

NA: Not Available

- **Response Packet**

NA: Not Available.

12. OPCODE_TF6x0_LISTEN_ONLY_ALLOW

- **Purpose**

Extenders send out this packet to let the viewers be listen mode.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8002	0x0209	**L**	**D**

L: Size of (NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG)

D: Structure of NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG.

Relevant:

Refer to NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG.

- **Response Packet**

NO NEED TO REPLY

13. **OPCODE_TF6x0_LISTEN_TYPE_CHANGED**

- **Purpose**

  N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x020A		

NA: Not Available

- **Response Packet**

NA: Not Available.

14. **OPCODE_TF6x0_ALIVE_REPORT**

- **Purpose**

  Viewers/Extenders send out this packet to announce alive status.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0301	**L**	**D**

L: Size of (NET_PROTOCOL_ALIVE_REPORT_MSG)

D: Structure of NET_PROTOCOL_ALIVE_REPORT_MSG.

Relevant:

Refer to NET_PROTOCOL_ALIVE_REPORT_MSG.

- **Response Packet**

If the status is normal, need not to reply, else reply data lost report.

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8003(ACK)	0x0302	**L**	**D**

L: Size of (NET_PROTOCOL_DATALOST_REPLY_MSG)

D: Structure of NET_PROTOCOL_DATALOST_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_DATALOST_REPLY_MSG.

15. **OPCODE_TF6x0_DATALOST_REPORT**

- **Purpose**

  Viewers send out this packet to announce data lost.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0302	**L**	**D**

L: Size of (NET_PROTOCOL_DATALOST_REPORT_MSG)

D: Structure of NET_PROTOCOL_DATALOST_REPORT_MSG.

Relevant:

Refer to NET_PROTOCOL_DATALOST_REPORT_MSG.

- **Response Packet**

If the status is normal, reply alive report:

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8003(ACK)	0x0301	**L**	**D**

L: Size of (NET_PROTOCOL_STATUS_OK_REPLY_MSG)

D: Structure of NET_PROTOCOL_STATUS_OK_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_STATUS_OK_REPLY_MSG.

Else reply data lost report:

COMMAND TYPE	OPCODE	LENGTH	DATA
0x8003(ACK)	0x0302	**L**	**D**

L: Size of (NET_PROTOCOL_DATALOST_REPLY_MSG)

D: Structure of NET_PROTOCOL_DATALOST_REPLY_MSG.

Relevant:

Refer to NET_PROTOCOL_DATALOST_REPLY_MSG.

16. **OPCODE_TF6x0_SOURCE_STATUS_ANNOUNCE**

- **Purpose**

  Extender sends out this packet to announce status when connection between Viewer and Extender is established.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0303	**L**	**D**

L: Size of (NET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG)

D: Structure of NET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG.

Relevant:

Refer to NET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG

- **Response Packet**

NO NEED TO REPLY

17. **OPCODE_TF6x0_EDID_DATA**

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0304		

NA: Not Available

- **Response Packet**

NA: Not Available.

18. **OPCODE_TF6x0_IRPULSE_DATA**

- **Purpose**

Viewers send out this packet to give IR pulse data.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0305	**L**	**D**

L: Size of (NET_PROTOCOL_IRPULSE_DATA_MSG)

D: Structure of NET_PROTOCOL_IRPULSE_DATA_MSG.

Relevant:

Refer to NET_PROTOCOL_IRPULSE_DATA_MSG

- **Response Packet**

NO NEED TO REPLY

19. OPCODE_TF6x0_PICTURE_STATUS

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0306		

NA: Not Available

- **Response Packet**

NA: Not Available.

20. OPCODE_TF6x0_VGA_POSITION_REQUEST

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0307		

NA: Not Available

- **Response Packet**

NA: Not Available.

21. OPCODE_TF6x0_PICTURE_BANDWIDTH_REQUEST

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0308		

NA: Not Available

- **Response Packet**

NA: Not Available.

22. OPCODE_TF6x0_MULTI_UNI_CAST_REQUEST

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0309		

NA: Not Available

- **Response Packet**

NA: Not Available.

23. OPCODE_TF6x0_STREAM_TYPE_REQUEST

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x030a		

NA: Not Available

- **Response Packet**

NA: Not Available.

24. **OPCODE_TF6x0_LOCK_PCA_REQUEST**

- **Purpose**

  N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x030b		

NA: Not Available

- **Response Packet**

NA: Not Available.

25. **OPCODE_TF6x0_CONTROL_MESSAGE**

- **Purpose**

  N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x030f		

NA: Not Available

- **Response Packet**

NA: Not Available.

26. **OPCODE_TF6x0_ADBUS_64BYTES**

- **Purpose**

Is replaced by OPCODE_TF6x0_ADBUS_64BYTES_AG.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0310		

NA: Not Available

- **Response Packet**

NA: Not Available.

27. OPCODE_TF6x0_ADBUS_64BYTES_AG

- **Purpose**

Viewers/Extenders send out this packet to give ADBUS data.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0311	**L**	**D**

L: Size of (NET_PROTOCOL_ADBUS_64BYTES_AG_MSG)

D: Structure of NET_PROTOCOL_ADBUS_64BYTES_AG_MSG.

Relevant:

Refer to NET_PROTOCOL_ADBUS_64BYTES_AG_MSG

- **Response Packet**

NO NEED TO REPLY

28. OPCODE_TF6x0_ADBUS_GPIO5

- **Purpose**

Viewer sends out this packet to announce status of GPIO5. (for device on ADBUS to refer).

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0312	**L**	**D**

L: Size of (NET_ADBUS_REPORT_MSG)

D: Structure of NET_ADBUS_REPORT_MSG.

Relevant:

Refer to NET_ADBUS_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

29. OPCODE_TF6x0_HDMI_SINK_STATUS

- **Purpose**

Viewers send out this packet to sync sink status.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0601	**L**	**D**

L: Size of (NET_PROTOCOL_HDMI_SINK_STATUS_MSG)

D: Structure of NET_PROTOCOL_HDMI_SINK_STATUS_MSG.

Relevant:

Refer to NET_PROTOCOL_HDMI_SINK_STATUS_MSG

- **Response Packet**

NO NEED TO REPLY

30. OPCODE_TF6x0_HDMI_CEC_MSG

- **Purpose**

Viewers/Extenders send out this packet to send CEC msg.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0602	**L**	**D**

L: Size of (NET_PROTOCOL_HDMI_CEC_CMD_MSG)

D: Structure of NET_PROTOCOL_HDMI_CEC_CMD_MSG.

Relevant:

Refer to NET_PROTOCOL_HDMI_CEC_CMD_MSG

- **Response Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0003	0x0602	**L**	**D**

L: Size of (NET_PROTOCOL_HDMI_CEC_CMD_MSG)

D: Structure of NET_PROTOCOL_HDMI_CEC_CMD_MSG.

Relevant:
Refer to NET_PROTOCOL_HDMI_CEC_CMD_MSG

31. OPCODE_TF6x0_NETWORK_DETECTING

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0401		

NA: Not Available

- **Response Packet**

NA: Not Available.

32. OPCODE_TF6x0_NETWORK_DETECTING_RESULT

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0402		

NA: Not Available

- **Response Packet**

NA: Not Available.

33. OPCODE_TF6x0_NETWORK_DETECTING_DATA

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0403		

NA: Not Available

- **Response Packet**

NA: Not Available.

34. OPCODE_TF6x0_NETWORK_DETECTING_PASSOWRD

- **Purpose**

N/A.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x0404		

NA: Not Available

- **Response Packet**

NA: Not Available.

35. OPCODE_TF6x0_PS2_MS_ALL

- **Purpose**

Extenders send out this packet to give mouse infos.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0500	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

36. OPCODE_TF6x0_PS2_MS_TYPE

- **Purpose**

Viewers send out this packet to give mouse type value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0501	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

37. OPCODE_TF6x0_PS2_MS_SCALE

- **Purpose**

Extenders send out this packet to give mouse scale value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0502	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

38. OPCODE_TF6x0_PS2_MS_RATE

- **Purpose**

Extenders send out this packet to give mouse rate value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0503	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

39. OPCODE_TF6x0_PS2_MS_RESOLUTION

- **Purpose**

Extenders send out this packet to give mouse resolution value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0504	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

40. OPCODE_TF6x0_PS2_MS_RESET

- **Purpose**

Extenders send out this packet to reset mouse.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0505	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

  NO NEED TO REPLY

41. OPCODE_TF6x0_PS2_KB_ALL

- **Purpose**

  Extenders send out this packet to give keyboard info.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0508	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

  NO NEED TO REPLY

42. OPCODE_TF6x0_PS2_KB_ID

- **Purpose**

  Viewers send out this packet to give keyboard IDs.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x0509	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

  NO NEED TO REPLY

43. OPCODE_TF6x0_PS2_KB_LED

- **Purpose**

  Extenders send out this packet to give keyboard led value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x050a	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

44. **OPCODE_TF6x0_PS2_KB_SET_SCANCODE**

- **Purpose**

Extenders send out this packet to give keyboard scan code value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x050b	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

NO NEED TO REPLY

45. **OPCODE_TF6x0_PS2_KB_GET_SCANCODE**

- **Purpose**

N/A

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
	0x050c		

- **Response Packet**

N/A

46. **OPCODE_TF6x0_PS2_KB_RATE**

- **Purpose**

  Extenders send out this packet to give keyboard rate value.

- **Command Packet**

COMMAND TYPE	OPCODE	LENGTH	DATA
0x0005	0x050c	**L**	**D**

L: Size of (NET_PS2_REPORT_MSG)

D: Structure of NET_PS2_REPORT_MSG.

Relevant:

Refer to NET_PS2_REPORT_MSG

- **Response Packet**

  NO NEED TO REPLY

APPENDIX – I (Data Structure Definition)

NET_PROTOCOL_DISCOVER_ANNOUNCE_MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DISCOVER_ANNOUNCE_MSG */
typedef struct tagNET_PROTOCOL_DISCOVER_ANNOUNCE_MSG {
      u32_t product_type;
      u16_t firmware_protocol_version;
      u8_t   my_groupname[NET_PROTOCOL_GROUPNAME_SIZE];
      u8_t   machine_name[NET_PROTOCOL_GROUPNAME_SIZE];
} NET_PROTOCOL_DISCOVER_ANNOUNCE_MSG;
```

NET_PROTOCOL_DISCOVER_REQUEST_MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DISCOVER_REQUEST_MSG*/
typedef struct tagNET_PROTOCOL_DISCOVER_REQUEST_MSG {
           u32_t product_type;
           u16_t firmware_protocol_version;
           u8_t   my_groupname[NET_PROTOCOL_GROUPNAME_SIZE];
           u8_t   machine_name[NET_PROTOCOL_GROUPNAME_SIZE];
           u32_t ask_type;   // define as product type
} NET_PROTOCOL_DISCOVER_REQUEST_MSG;
```

NET_PROTOCOL_DISCOVER_REPLY_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DISCOVER_REPLY_MSG */
typedef struct tagNET_PROTOCOL_DISCOVER_REPLY_MSG{
      u32_t product_type;
      u16_t firmware_protocol_version;
      u8_t   my_groupname[NET_PROTOCOL_GROUPNAME_SIZE];
      u8_t   machine_name[NET_PROTOCOL_GROUPNAME_SIZE];
      u16_t reply_sequence_id;
} NET_PROTOCOL_DISCOVER_REPLY_MSG;
```

NET_PROTOCOL_CONNECTING_REQUEST _MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_CONNECTING_REQUEST_MSG */
typedef struct tagNET_PROTOCOL_CONNECTING_REQUEST_MSG{
    u8_t   my_ip[NET_PROTOCOL_IP_SIZE];
    u16_t my_cmd_port;
    u8_t   my_reserved[NET_PROTOCOL_IP_SIZE];
    u32_t product_type;
    u16_t firmware_protocol_version;
    u8_t   my_groupname[NET_PROTOCOL_GROUPNAME_SIZE];
    u8_t   machine_name[NET_PROTOCOL_GROUPNAME_SIZE];
    u8_t   username[NET_PROTOCOL_USERNAME_SIZE]; // encrypted
by rand_code
    u32_t my_ticks;
} NET_PROTOCOL_CONNECTING_REQUEST_MSG;
```

NET_PROTOCOL_CONNECTING_REPLY_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_CONNECTING_REPLY_MSG */
typedef struct tagNET_PROTOCOL_CONNECTING_REPLY_MSG{
    u8_t   my_ip[NET_PROTOCOL_IP_SIZE];
    u16_t my_cmd_port;
    u8_t   my_reserved[NET_PROTOCOL_IP_SIZE];
    u32_t product_type;
    u16_t firmware_protocol_version;
    u8_t   my_groupname[NET_PROTOCOL_GROUPNAME_SIZE];
    u8_t   machine_name[NET_PROTOCOL_GROUPNAME_SIZE];
    u32_t my_ticks;
    u32_t reply_ticks;
    u16_t reply_sequence_id;
} NET_PROTOCOL_CONNECTING_REPLY_MSG;
```

NET_PROTOCOL_REPLY_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_REPLY_MSG */
typedef struct tagNET_PROTOCOL_REPLY_MSG{
        u16_t status;              // 0: success, 1: , 2: , 3: ,
        u32_t my_ticks;
        u32_t reply_ticks;
        u16_t reply_sequence_id;
} NET_PROTOCOL_REPLY_MSG;
```

NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG */
typedef struct tagNET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG{
        //u8_t    my_mac[NET_PROTOCOL_MAC_SIZE];
        u8_t    my_ip[NET_PROTOCOL_IP_SIZE];
        u16_t my_cmd_port;
        u8_t    my_reserved[NET_PROTOCOL_IP_SIZE];
        u32_t product_type;
        u16_t firmware_protocol_version;
        u8_t    my_groupname[NET_PROTOCOL_GROUPNAME_SIZE];
        u8_t    machine_name[NET_PROTOCOL_GROUPNAME_SIZE];

        u8_t    av_dst_mac[NET_PROTOCOL_MAC_SIZE];
        u8_t    av_dst_ip[NET_PROTOCOL_IP_SIZE];
        u16_t cmd_port;
        u8_t    cmd_streamtype;
        u16_t ps2_port;
        u8_t    ps2_streamtype;
        u16_t audio_port;
        u8_t    audio_streamtype;
        u16_t video_port;
        u8_t    video_streamtype;
        u16_t reserve1_port;
        u8_t    reserve1_streamtype;
        u16_t reserve2_port;
        u8_t    reserve2_streamtype;

        u32_t my_ticks;
        u32_t reply_ticks;
        u16_t reply_sequence_id;
        u8_t    support_list[12];    // [0]: 0- master receiver, others- slave receiver
                                              // [1]: 0- remote off, others- remote on.
                                              //           1:IR remote on, 2:PS2 remote on,
4:audio remote on
                                              // [2]: network scaledown compress rate
                                              //           0:default, 1:low, 2:mid, 3:high
                                              // [3]: bit[0]: video stream exist, bit[1]: audio
stream exist
} NET_PROTOCOL_LISTEN_ONLY_ALLOW_MSG;
```

NET_PROTOCOL_DISCONNECTING_REQUEST_MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DISCONNECTING_REPLY_MSG */
typedef struct tagNET_PROTOCOL_DISCONNECTING_REQUEST_MSG{
        u8_t    username[NET_PROTOCOL_USERNAME_SIZE];    // encrypted
by rand_code
        u32_t my_ticks;
} NET_PROTOCOL_DISCONNECTING_REQUEST_MSG;
```

NET_PROTOCOL_DISCONNECTING_REPLY_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DISCONNECTING_REPLY_MSG */
typedef struct tagNET_PROTOCOL_DISCONNECTING_REPLY_MSG{
        u8_t    username[NET_PROTOCOL_USERNAME_SIZE];    // encrypted
by rand_code
        u32_t my_ticks;
        u32_t reply_ticks;
        u16_t reply_sequence_id;
} NET_PROTOCOL_DISCONNECTING_REPLY_MSG;
```

NET_PROTOCOL_ALIVE_REPORT_MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_ALIVE_REPORT_MSG */
typedef struct tagNET_PROTOCOL_ALIVE_REPORT_MSG{
        u8_t    frame_counter;
        u8_t    success_frames;
        u8_t    fail_frames;
        u32_t my_ticks;
        u16_t display_width;
        u16_t display_height;
        u8_t    kb_push_flag;
        u8_t    kb_push_no;
        u8_t    kb_scan_code[20];
        u8_t    audio_status;    // b7: audio abnormal, [b6:b0]: audio buffer count
if b7 is set.
        u8_t    next_flag;
} NET_PROTOCOL_ALIVE_REPORT_MSG;
```

NET_PROTOCOL_STATUS_OK_REPLY_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_STATUS_OK_REPLY_MSG */
typedef struct tagNET_PROTOCOL_STATUS_OK_REPLY_MSG{
        u32_t my_ticks;
        u32_t reply_ticks;
        u16_t reply_sequence_id;
} NET_PROTOCOL_STATUS_OK_REPLY_MSG;
```

NET_PROTOCOL_DATALOST_REPORT_MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DATALOST_REPORT_MSG */
```

```
typedef struct tagNET_PROTOCOL_DATALOST_REPORT_MSG{
        u32_t my_ticks;
} NET_PROTOCOL_DATALOST_REPORT_MSG;
```

NET_PROTOCOL_DATALOST_REPLY_MSG

```
/* RESPONSE MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_DATALOST_REPLY_MSG */
typedef struct tagNET_PROTOCOL_DATALOST_REPLY_MSG{
        u32_t my_ticks;
        u32_t reply_ticks;
        u16_t reply_sequence_id;
} NET_PROTOCOL_DATALOST_REPLY_MSG;
```

NET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG

```
/* COMMAND MESSAGE FORMAT of OPCODE:
NET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG */
typedef struct
tagNET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG{
        u8_t    av_dst_mac[NET_PROTOCOL_MAC_SIZE];
        u8_t    av_dst_ip[NET_PROTOCOL_IP_SIZE];
        u16_t input_source;
        u16_t source_width;
        u16_t source_height;
        u16_t source_framerate;
        u16_t xmt_width;
        u16_t xmt_height;
        u16_t xmt_audiorate;
        u32_t my_ticks;
        u8_t    remote_onoff;      // 0- remote off, others- remote on.
                                    //       1: IR remote on, 2: PS2 remote on, 4:
audio remote on
        u8_t    compress_highlow;    // 0:default, 1:low, 2:mid, 3:high
        u8_t    hdcp_onoff;          // 0- hdcp disable, 1- hdcp enable
        u8_t    macrovision_onoff; // 0- macrovision off,    others- macrovision
mode
        u8_t    avi_info_db2;         // AVI Infoframe Data Byte2    (M1 M0 is
refered currently.)
} NET_PROTOCOL_SOURCE_STATUS_ANNOUNCE_MSG;
```

NET_PROTOCOL_SOURCE_CHANGE_REQUEST_MSG

```
typedef struct
tagNET_PROTOCOL_SOURCE_CHANGE_REQUEST_MSG{
        u16_t input_source;
        u16_t xmt_width;
        u16_t xmt_height;
        u16_t xmt_audiorate;
        u32_t my_ticks;
} NET_PROTOCOL_SOURCE_CHANGE_REQUEST_MSG;    // from TF600
```

NET_PROTOCOL_SOURCE_CHANGE_REPLY_MSG

```
typedef struct tagNET_PROTOCOL_SOURCE_CHANGE_REPLY_MSG{
        u16_t input_source;
        u16_t source_width;
        u16_t source_height;
        u16_t source_framerate;
        u16_t xmt_width;
        u16_t xmt_height;
        u16_t xmt_audiorate;
        u32_t my_ticks;
        u32_t reply_ticks;
        u32_t reply_sequence_id;
        u8_t remote_onoff;    // 0- remote off, others- remote on.
                                  //      1: IR remote on, 2: PS2 remote on, 4: audio
remote on
        u8_t    compress_highlow;    // 0:default, 1:low, 2:mid, 3:high
        u8_t    hdcp_onoff;          // 0- hdcp disable, 1- hdcp enable
        u8_t    macrovision_onoff; // 0- macrovision off,    others- macrovision
mode
} NET_PROTOCOL_SOURCE_CHANGE_REPLY_MSG;    // from TF630
```

NET_PROTOCOL_EDID_DATA_MSG

```
typedef struct tagNET_PROTOCOL_EDID_DATA_MSG{
        u32_t my_ticks;
        u8_t    edid_data[512];
} NET_PROTOCOL_EDID_DATA_MSG;    // from TF600
```

NET_PROTOCOL_IRPULSE_DATA_MSG

```
typedef struct tagNET_PROTOCOL_IRPULSE_DATA_MSG{
        u32_t my_ticks;
        u16_t irpulse_data[255];
} NET_PROTOCOL_IRPULSE_DATA_MSG;    // from TF600
```

NET_PROTOCOL_AV_DATA_MSG

```
typedef struct tagNET_PROTOCOL_PICTURE_STATUS_MSG{
        u16_t picture_status;      // PAUSE, PLAY, UNSTABLE, STABLE
        u32_t my_ticks;
} NET_PROTOCOL_AV_DATA_MSG;    // from TF630
```

NET_PROTOCOL_VGA_POSITION_REQUEST_MSG

```
typedef struct tagNET_PROTOCOL_VGA_POSITION_REQUEST_MSG{
        u16_t stepx;             // bit[15] is signed bit
        u16_t stepy;             // bit[15] is signed bit, stepx and stepy both == 0
means auto sync
        u32_t my_ticks;
} NET_PROTOCOL_VGA_POSITION_REQUEST_MSG;    // from TF600
```

NET_PROTOCOL_PICTURE_QUALITY_REQUEST_MSG

```
typedef struct
tagNET_PROTOCOL_PICTURE_QUALITY_REQUEST_MSG{
      u8_t rule;      // 0:JPEG quality,    1:Scaledown rate
      u8_t quality;    // 0:default, 1:low, 2:mid, 3:high
      u32_t my_ticks;
} NET_PROTOCOL_PICTURE_QUALITY_REQUEST_MSG;    // from
TF600
```

NET_PROTOCOL_CONTROL_REQUEST_MSG

```
typedef struct tagNET_CONTROL_REQUEST_MSG{
      u8_t method;
      u32_t my_ticks;
} NET_PROTOCOL_CONTROL_REQUEST_MSG;    // from TF600
```

NET_PROTOCOL_CONTROL_MESSAGE_MSG

```
typedef struct tagNET_CONTROL_MESSAGE_MSG{
      u16_t op_code;
      u8_t    message_id;
      u16_t status;
      u32_t my_ticks;
} NET_PROTOCOL_CONTROL_MESSAGE_MSG;    // from TF630
```

NET_PROTOCOL_ADBUS_64BYTES_MSG

```
typedef struct tagNET_PROTOCOL_ADBUS_64BYTES_MSG{
        u32_t my_ticks;
        u8_t    ADBus_port;
        u8_t    ADBus_no;
        u8_t    ADBus_data[MAX_ADBUS_LEN];
} NET_PROTOCOL_ADBUS_64BYTES_MSG;    // from TF600
```

NET_PROTOCOL_ADBUS_64BYTES_AG_MSG

```
typedef struct
tagNET_PROTOCOL_ADBUS_64BYTES_AG_MSG{        //BRT_ADBUS_P
_AG
        u32_t my_ticks;
        u32_t    ADBus_port;
        u16_t    ADBus_no;
        u8_t    ADBus_data[MAX_ADBUS_LEN];
} NET_PROTOCOL_ADBUS_64BYTES_AG_MSG;    // from TF600
```

NET_PROTOCOL_DETECTING_RESULT_MSG

```
typedef struct tagNET_PROTOCOL_DETECTING_RESULT_MSG{
     u32_t start_packet_ticks;
     u32_t last_packet_ticks;
     u16_t start_packet_sequence;
     u16_t last_packet_sequence;
     u16_t total_packets;
     u16_t total_error_packets;
     u32_t my_ticks;
} NET_PROTOCOL_DETECTING_RESULT_MSG;    // from TF600
```

NET_PS2_REPORT_MSG

```
typedef struct tagNET_PS2_REPORT_MSG{
        u8_t content[8];
} NET_PS2_REPORT_MSG;    // from TF6x0
```

NET_ADBUS_REPORT_MSG

```
typedef struct tagNET_ADBUS_REPORT_MSG{
        u8_t adbus_sm;                  //this will record the current state machine!,
used for re-transmit UDP machanism
        u8_t adbus_status;      //0: init done, 1: not init done!
} NET_ADBUS_REPORT_MSG;    // from TF6x0
```

NET_PROTOCOL_DETECTING_RESULT_REPLY_MSG

```
typedef struct tagNET_PROTOCOL_DETECTING_RESULT_REPLY_MSG{
        u32_t my_ticks;
     u32_t reply_ticks;
     u32_t reply_sequence_id;
} NET_PROTOCOL_DETECTING_RESULT_REPLY_MSG;       // from
TF630
```

NET_PROTOCOL_DETECTING_RESULT_ACK_MSG

```
typedef struct tagNET_PROTOCOL_DETECTING_RESULT_ACK_MSG{
        u16_t status;
        u16_t reply_requence_id;
} NET_PROTOCOL_DETECTING_RESULT_ACK_MSG;       // from TF600
```

NET_PROTOCOL_HDMI_SINK_STATUS_MSG

```
typedef struct tagNET_PROTOCOL_HDMI_SINK_STATUS_MSG{
        u8_t ack; // for PCA to response
        u8_t sinkStatus;
        u8_t phyAddr[2];
} NET_PROTOCOL_HDMI_SINK_STATUS_MSG;
```

NET_PROTOCOL_HDMI_CEC_CMD_MSG

```
typedef struct tagNET_PROTOCOL_HDMI_CEC_CMD_MSG{
        u8_t ack;                    // reserved for future use
        u16_t cmd_opcode;         // refer to CEC spec.    //TF_CEC_OPCODE
        u8_t para_msg[2];       // should map to another structure according to
cmd_opcode
} NET_PROTOCOL_HDMI_CEC_CMD_MSG;
```

APPENDIX – II (Symbolic Constant Definition)