Category: Informational

Y. Pavlenko
D. Isaenko
A. Andreev
Rig Expert Ukraine ltd.

Transceiver IP Link Protocol rev.1

Abstract

This document specifies an application-level Transceiver IP Link Protocol (TILP) to be used to monitor and control devices compatible with CAT & RS-485 protocols remotely.

Table of contents

Goals 3
Transceiver audio interface 3
CAT interface 3
FSK output 3
PTT and CW output functions 3
RS-485 3
WinKey emulation 3
Terminology 4
TILP protocol details 4
General structure of the data packet 4
Packet types overview 5
Authorization packet 6
Checksum calculation algorithm 7
Session keep alive 8
PTT packet 9
Audio packet 11
TTY packet 15
CONNERR packet structure 19
Access levels packet 20
Firmware version packet 22
Packet aggregation 23
Packet segmentation 23
Alignment 23
PTT/CW stream 23
Appendix A An example of the short session 24
Appendix B The short session with the aggregated packets 25
Appendix C The session with the audio streams example 26
Appendix D The session with the audio and CW/PTT streams 27
Appendix E The session keepalive example 28
Appendix F The serial ports data transfer example 29

Goals

Most popular ham radio applications for working with transceivers require multiple serial ports. When it comes to control the transceiver remotely it is necessary to use special devices for bind serial ports via the Internet or via a local area network. Or to use TILP-compatible devices to communicate with remote transceiver(s) via IP-based network.

Transceiver audio interface

TILP protocol provides audio stream encapsulation and delivery from the transceiver to control software and back.

CAT interface

CAT (Computer Aided Tuning) provides control of receiving and transmitting frequency, VFO, diversity reception, audio levels, memory and other operations by the computer software. Normally, modern transceivers have serial (with various signal levels) link providing CAT interface.

FSK output

FSK (Frequency Shift Keying) is a popular method of transmitting digital messages over radio primarily used in radioteletype (RTTY) mode. Most transceivers provide FSK modulator feature to make the RTTY signal stable and clear.

PTT and CW output functions

Transceivers provide PTT (Push To Talk) and CW (Continuous Wave) keyer inputs to allow setting the transmitter on or off and operating CW using external device (PTT pedal, CW bug or paddle, terminal node controller, or personal computer).

RS-485

Some types of equipment, such as antenna switches and relays can be controlled via a RS-485 port. Multiple devices may be connected in parallel and controlled by the computer software.

WinKey emulation

To operate CW the ham operator may use software which either directly manipulates the DTR line, or uses the WinKey protocol.

All the features mentioned above are supported by the TILP protocol.

Terminology

TILP device - any device that supports TILP protocol.

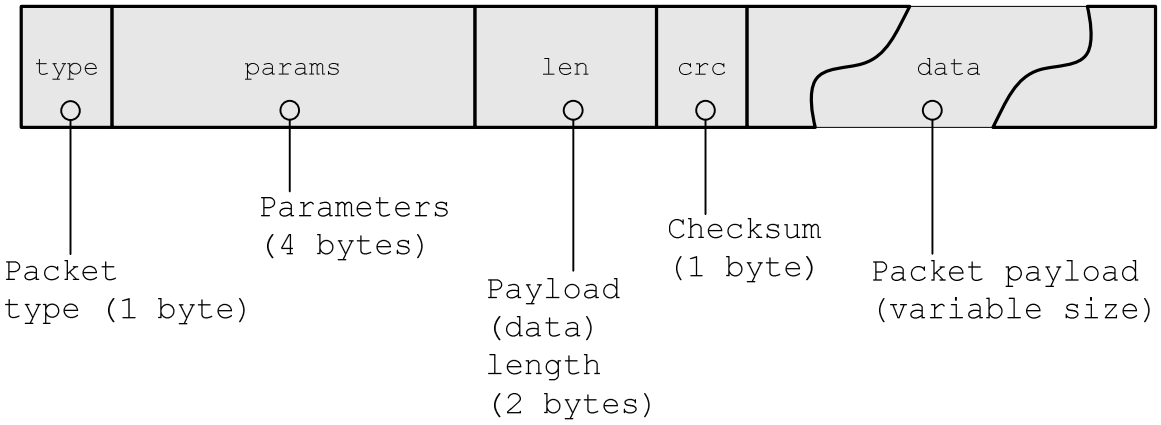
TILP protocol details

General structure of the data packet

RigExpert Wireless Transceiver Interfaces uses a proprietary protocol for exchanging data between the device and the computer.

This protocol has the following key features:
- data flow from PC software to TILP device and back is organized into packets
- packets are Checksum Protected
- The Payload is a variable length field

Packet structure:



Packet contents:			
Offset	Length (bytes)	Name	Description
0	1	type	Packet type. Field value depends on the packet type
1	4	params	Parameters (depend on the packet type)
5	2	len	Payload length (in bytes)
7	1	crc	Checksum value. The checksum is calculated over the packet excluding the CRC field
8	up to 0xFFFF	data	Packet payload

NOTE: packet header consists of “type”, “params” and “len” fields (first 7 bytes)

NOTE: The checksum is calculated over the packet excluding the CRC field. I.e. checksum algorithm takes into account only header and data fields (“type”, “params”, “len” and “data”).

Packet types overview

Mnemonic	Hex value	Description
AUTHORIZ_TYPE	0x00	Authorization packet
PTT_TYPE	0x01	PTT state packet
AUDIO_TYPE	0x02	Audio parameters packet (sends by both PC and TILP device)
TTY1_TYPE	0x03	Parameters and data for CAT serial port
TTY2_TYPE	0x04	Parameters and data for RS-485 port
TTY3_TYPE	0x05	Parameters and data for FSK port
CONERR_TYPE	0x08	Connection error type (sends by TILP device)
PERMISS_TYPE	0x09	Access right parameters (sends by TILP device)
FWVER_TYPE	0x0A	Firmware revision

```
/**
 * @brief Packet types
*/
typedef enum TILProtocolPacketTypes {
AUTHORIZ_TYPE = 0x00,    // Authorization packet
PTT_TYPE,       // PTT state packet
AUDIO_TYPE,     // Audio parameters packet
TTY1_TYPE,      // Parameters and data for CAT serial port
TTY2_TYPE,      // Parameters and data for RS-485 port
TTY3_TYPE,      // Parameters and data for FSK port
RESERVED_6,
RESERVED_7,
CONERR_TYPE,    // Connection error type
PERMISS_TYPE,   // Access rights parameters
FWVER_TYPE,     // Firmware revision
};
```

Authorization packet

Payload contents:				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x00	Packet type
1	4	params	0x00	For Authorization packet this field value must be zeroed
5	2	len	0x00 .. 0x20	Passphrase length (in bytes). Passphrase length must not exceed 32 bytes
7	1	crc		CRC of the packet (except CRC field)
8	up to 32	data		Passphrase

If the password doesn’t match the TILP device’s one, the TCP connection will be terminated by the TILP device.

```
/**
 * @brief send_autorization
 * @param sd - socket descriptor
 * @param pass - pointer to password string
 * @return
 */
int send_autorization(int sd, const char *pass)
{
    const uint32_t slen = strlen(pass);
    char buff[sizeof(packet_t) + slen];
    int retval;
    struct packet_t *pack;

    pack = (struct packet_t*)buff;
    pack->type = AUTHORIZ_TYPE;
    pack->params = 0;
    pack->len = slen;
    pack->crc = 0;

    memcpy(&buff[sizeof(packet_t)], pass, slen);
    pack->crc = crc8(buff, sizeof(packet_t) + pack->len);

    retval = write(sd, buff, sizeof(buff));

    return retval;
}
```

Checksum calculation algorithm

The checksum is calculated over the packet excluding the CRC field. I.e. checksum algorithm takes into account only header and data fields (“type”, “params”, “len” and “data”).

```
struct packet_t {
    uint8_t  type;
    uint32_t params;
    uint16_t len;
    uint8_t  crc;
};

char buff[30];
struct packet_t *pack;
pack = (struct packet_t*)buff;
pack->len = sizeof(buff);
pack->crc = 0;
// Init other fields of struct packet_t
pack->crc = crc8(&pack, sizeof(packet_t) + pack->len);
```

For the CRC8 calculations use this function:

```
/*
  Name  : CRC-8
  Poly  : 0x31    x^8 + x^5 + x^4 + 1
  Init  : 0xFF
  Revert: false
  XorOut: 0x00
  Check : 0xF7 ("123456789")
*/
unsigned char crc8(unsigned char *pcBlock, unsigned int len)
{
    unsigned char crc = 0xFF;
    unsigned int i;

    while (len--)
    {
        crc ^= *pcBlock++;

        for (i = 0; i < 8; i++)
            crc = crc & 0x80 ? (crc << 1) ^ 0x31 : crc << 1;
    }

    return crc;
}
```

Session keep alive

Within 8 (eight) seconds, the application and the interface should exchange with at least one packet. Usually this is the PTT status packet (the interface transmits this packet asynchronously).
If within 8 (eight) seconds the interface does not take a single packet, the TCP/IP connection will be terminated and the interface will go into a safe state (switched off PTT, CW and FSK).
If within 8 (eight) seconds the user application has not received a single packet from the interface, it must break the connection.

PTT packet

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x01	Packet type
1	4	params	0	For PTT packet this field value must be zeroed
5	2	len	0x01	For the PTT packet the payload always consist of one byte
7	1	crc		CRC of the packet (header and data)
8	1	data	0 or 1	0 = PTT OFF 1 = PTT ON

PTT packet is sent both ways – to and from the interface. When the packet is sent from the application to the interface, the interface sets PTT as specified in the data field. Packets which send from the interface to the application have current status of PTT. Application can update PTT status from those packets if it needs.

Packet sending:

```
/**
 * @param sd - Socket descriptor
 * @param state - ptt state
 * @return - > 0 if Ok, <=0 if error
  */
int send_ptt(int sd, bool state)
{
     int retval;
    char buff[sizeof(packet_t) + 1];
    struct packet_t *pack;

    pack = (struct packet_t*)buff;
    pack->type = PTT_TYPE;
    pack->params = 0;
    pack->len = 1;
    pack->crc = 0;

    if (state) {
       buff[sizeof(packet_t] = 1;
    } else {
       buff[sizeof(packet_t] = 0;
    }
    /*Calculate CRC8*/
    pack->crc = crc8(&pack, sizeof(packet_t) + pack->len);
    /*Send data via TCP socket*/
```

```
    retval = write(sd, buff, sizeof(buff));
          return retval;
 }

Packet receiving:
int get_ptt(int sd)
{
    int retval;
    char buff[sizeof(packet_t) + 1];
    struct packet_t *pack;
    uint8_t  crc;

    retval = read(sd, buff, sizeof(buff));

    if (retval <= 0 || retval != sizeof(buff)) {
        return -1;
    }

    pack = (struct packet_t*)buff;

    if (pack->type != PTT_TYPE || pack->len != 1) {
        return -1;
    }

    crc = pack->crc;
    pack->crc = 0;

    pack->crc = crc8(&pack, sizeof(packet_t) + pack->len);

    if (crc != pack->crc) {
        return -1;
    }

    return (int)buff[sizeof(packet_t)];
}
```

Audio packet

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x02	Packet type
1	4	params		Sampling rate & codec used
5	2	len	0x00	Initial packet contains no payload
7	1	crc		CRC of the packet (header and data)

[Params] field structure for Audio packet				
Offset	Length (bytes)	Name	Value	Description
0	2	samplerate		The sampling rate (Hz)
2	2	codec		Codec types: 0 = PCM 1 = μ-law 2 = A-law

The application must send this packet to properly initialize the audio codec after a connection has been established.

```
typedef union {
    uint32_t data;
    struct{
        uint16_t samplerate;
        uint16_t codec;
    };
} audio_head_t;


/**
 * @brief send_audio_init
 * @param sd - socket descriptor
 * @param srate - Samplerate (8000, 12000, 16000)
 * @param codec - 0–PCM, 1–uLAW, 2-aLAW
 * @return
 */
bool send_audio_init(int sd, uint16_t srate, uint16_t codec)
{
    int retval;
    char buff[sizeof(packet_t)];
    struct packet_t *pack;
    audio_head_t ahead;
```

```
    ahead.samplerate = srate;
    ahead.codec = codec;

    pack = (struct packet_t*)buff;
    pack->type = AUDIO_TYPE;
    pack->params = ahead.data;
    pack->len = 0;
    pack->crc = 0;

    /*Calculate CRC8*/
    pack->crc = crc8(&pack, sizeof(packet_t));
    /*Send data via TCP socket*/
    retval = write(sd, buff, sizeof(buff));

    if (retval < sizeof(buff)) {
        return false;
    }

    return true;
}
```

The interface in turn sends the packet that contains the input and output levels of the codec's amplifiers:

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x02	Packet type
1	4	params		Sampling rate & codec used
5	2	len	0x03	
7	1	crc		CRC of the packet (header and data)
8	1	outlvl	0..118	Out level. Interface's audio output level. Range from 0 to 118 (0 = 0dB, 118 = 78.3dB)
9	1	inLlvl	0..110	In level. Left channel amplifier level. Range from 0 to 110. (0 = 0dB, 110 = 55dB).
10	1	inRlvl	0..110	In level. Right channel amplifier level. Range from 0 to 110. (0 = 0dB, 110 = 55dB)

```
/**
 * @brief receive_audio_params
 * @param [in] sd - socket descriptor
 * @param [out] outlvl - current output level in the TILP device
 * @param inLlvl [out] - current input level in the TILP device
(left channel)
 * @param inRlvl [out] - current input level in the TILP device
(right channel)
 * @return true of false
 */
bool receive_audio_params(int sd, uint8_t* outlvl, uint8_t*
inLlvl, uint8_t* inRlvl)
{
    int retval;
    char buff[sizeof(packet_t)+3];
    struct packet_t *pack;
    audio_head_t ahead;

    retval = read(sd, buff, sizeof(buff));

    if (retval <= 0 || retval != sizeof(buff)) {
        return false;
    }

    pack = (struct packet_t*)buff;

    if (pack->type != AUDIO_TYPE || pack->len != 3) {
        return false;
    }

    crc = pack->crc;
    pack->crc = 0;

    pack->crc = crc8(&pack, sizeof(packet_t) + pack->len);

    if (crc != pack->crc) {
        return false;
    }

    *outlvl = buff[sizeof(packet_t)];
    *inLlvl = buff[sizeof(packet_t)+1];
    *inRlvl = buff[sizeof(packet_t)+2];

    return true;
}
```

To set levels the application must send this packet. If the levels are not changed again for 5 seconds, current values will be stored in the non-volatile RAM.

Setting audio levels example:

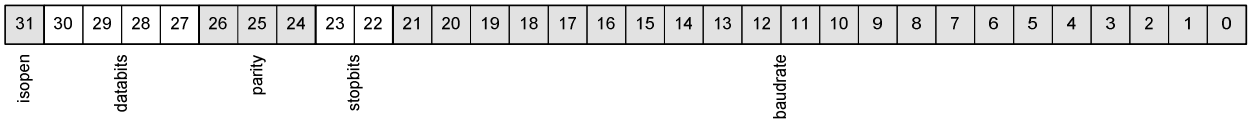
```
/**
 * @brief send_audio_params
 * @param sd - socket descriptor
 * @param srate - Samplerate (8000, 12000, 16000)
 * @param codec - 0–PCM, 1–uLAW, 2-aLAW
 * @param outlvl - output level for the TILP device
 * @param inLlvl - input level for the TILP device (left
channel)
 * @param inRlvl - input level for the TILP device (right
channel)
 * @return true of false
 */
bool send_audio_params(int sd, uint16_t srate, uint16_t codec,
                       uint8_t outlvl, uint8_t inLlvl, uint8_t
inRlvl)
{
    int retval;
    char buff[sizeof(packet_t)+3];
    struct packet_t *pack;
    audio_head_t ahead;

    ahead.samplerate = srate;
    ahead.codec = codec;

    pack = (struct packet_t*)buff;
    pack->type = AUDIO_TYPE;
    pack->params = ahead.data;
    pack->len = 3;
    pack->crc = 0;

    buff[sizeof(packet_t)] = outlvl;
    buff[sizeof(packet_t)+1] = inLlvl;
    buff[sizeof(packet_t)+2] = inRlvl;

    /*Calculate CRC8*/
    pack->crc = crc8(&pack, sizeof(packet_t)+pack->len);
    /*Send data via TCP socket*/
    retval = write(sd, buff, sizeof(buff));

    if (retval < sizeof(buff)) {
        return false;
    }
    return true;
}
```

TTY packet

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x03 .. 0x05	Packet type 0x03 = CAT data 0x04 = RS485 data 0x05 = FSK data
1	4	params		Serial port settings bitfield
5	2	len		Payload length (in bytes)
7	1	crc		CRC of the packet
8	Variable	data		Payload

Type – packet type. Values:
3 for serial CAT
4 for serial RS485
5 for serial FSK

Port settings bitfield:



[Params] field structure for TTY packet		
Length (bits)	Name	Description
1	isopen	0 = closed 1 = open If any application opens this serial port, this bit must be set to 1. If port is closed – set to 0
4	databits	data width in bits
3	parity	0 = No parity 1 = Odd 2 = Even 3 = Mark 4 = Space
2	stopbits	0 = 1 stop-bit 1 = 1.5 stop-bits 2 = 2 stop-bits
22	baudrate	baud rate

The serial port buffer size in the TILP device is 256 bytes. The interface replies with the TTY packet setting the field “Param” to the buffer’s free space. Also this packet may contain data, received by the interface from the transceiver. This packet may be sent by the interface asynchronously.

```
typedef union {
    uint32_t data;
    struct {
        uint32_t isopen      :1;
        uint32_t databits    :4;
        uint32_t parity      :3;
        uint32_t stopbits    :2;
        uint32_t baudrate    :22;
    };
} tty_head_t;


static const uint32_t bufsize = 256;
static uint32_t buff_fill = 0;


/**
 * @brief send_serial
 * @param sd - socket descriptor
 * @param spd - serial port descriptor
 * @param pd - pinter to data
 * @param len - data len
 * @return - how many bytes was sent
 */
int send_serial(int sd, int spd, const void* pd, uint32_t len)
{
    char *buff;
    uint32_t wlen;
    struct packet_t *pack;
    tty_head_t serhdr;

    if (bufsize == buff_fill) {
        return 0;
    }

    wlen = len <= (bufsize - buff_fill) ? len : (bufsize -
buff_fill);

    buff = malloc(sizeof(struct packet_t) + wlen);

    if (buff == NULL) {
        return -1;
    }

    pack = (struct packet_t *) buff;
```

```
    // Fill serial port paremeters
    serhdr.isopen = serialport_state(spd);
    serhdr.databits = serialport_databits(spd);
    serhdr.parity = serialport_parity(spd);
    serhdr.stopbits = serialport_stop(spd);
    serhdr.baudrate = serialport_baud(spd);

    pack->params = serhdr.data;
    pack->type = TTY1_TYPE;
    pack->len = wlen;
    pack->crc = 0;

    memcpy(&buff[sizeof(struct packet_t)], pd, wlen);

    pack->crc = crc8(bufsize, sizeof(packet_t)+pack->len);

    retval = write(sd, buff, sizeof(packet_t)+pack->len);

    if (retval != sizeof(packet_t)+pack->len) {
        wlen = 0;
    }

    free(buff);

    buff_fill += wlen;

    return wlen;
}


/**
 * @brief receive_serial
 * @param sd - socket descriptor
 * @param pd - pointer to buffer
 * @param len - buffer size
 * @return - how many bytes was sent
 */
int receive_serial(int sd, void* pd, uint32_t len)
{
    char buff[sizeof(packet_t)+bufsize];
    uint32_t rlen;
    struct packet_t *pack;
    uint8_t crc;
    uint32_t wlen;

    retval = read(sd, buff, sizeof(buff));

    if (retval < sizeof(packet_t)) {
        return 0;
    }

    pack = (struct packet_t*)buff;
```

```
    if (pack->type != TTY1_TYPE || pack->len > bufsize) {
        return 0;
    }

    crc = pack->crc;
    pack->crc = 0;

    pack->crc = crc8(buff, sizeof(packet_t)+pack->len);

    if (pack->crc != crc) {
        return 0;
    }

    if (len < pack->len) {
        pack->len = len;
    }

    memcpy(pd, &buff[sizeof(packet_t)], pack->len);

    buff_fill = pack->params;

    return pack->len;
}
```

CONNERR packet structure

When connection error occurs, the interface sends the CONNERR packet with the error code.

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x08	Packet type
1	4	params		Error code
5	2	len	0x00	contains no payload
7	1	crc		CRC of the packet

The interface may send next error codes:

Error code	Description
0x00	No errors (NoError)
0x01	Multiple connections (MultipleConnection). The error occurs when more than one application attempt to connect to the interface
0x02	Wrong password (WrongPassword). The error occurs when the password in the authentication packet does not match stored in TILP device password
0x03	Session timeout (Timeout). The error occurs when the application did not communicate with interface for more than 8 seconds
0x04	Unknown packet type (UnkonownPacket). Data in the packet has unknown format

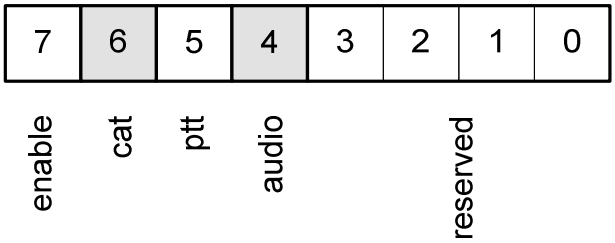
Access levels packet

Access levels packets are transmitted by the interface only. The interface supports multiple access profiles. Data exchange according to the access level is performed by the application.

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x09	Packet type
1	4	params	0x00	For Access Level packet this field value must be zeroed
5	2	len	0x09	Payload length (in bytes)
7	1	crc		CRC of the packet
8	1	flags		Permissions bit field
9	4	worktime		Active session time limit in minutes. After the expiration of the time limit, the application must close the connection
13	4	pausetime		Connecting pause time in minutes. After closing the connection the application must wait a specified time before the new connection. When all fields are set to 1 (full access), the time limit fields must be ignored

Flags bitfield described as follows:

[Flags] bitfield structure		
Length (bits)	Name	Description
1	enable	Field is set to 1 when the profile is active. In case this field set to 0, the application must terminate the connection
1	cat	Field is set to 1 when the application is allowed CAT data transfer (CAT serial port). If this field set to 0, any data received through the CAT interface must be ignored
1	ptt	Field is set to 1 when the application is allowed to turn on the transmitter by PTT. If this field set to 0, the application must ignore PTT commands from user
1	audio	field is set to 1 when the application is allowed change audio levels. If this field set to 0, the application must ignore the audio levels change commands from user




```
union flags_t {
    quint8 data;
    struct {
        quint8 enable:1;
        quint8 cat:1;
        quint8 ptt:1;
        quint8 audio:1;
        quint8 unused:4;
    };
};
```

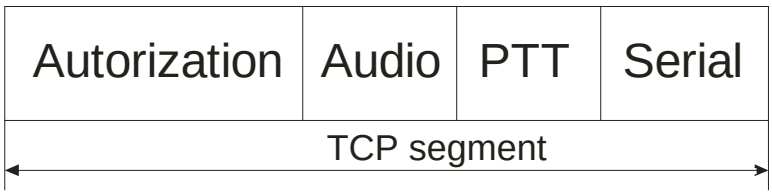
Worktime field is a uint32_t variable, contains the active session time limit in minutes. After the expiration of the time limit, the application must close the connection.

Pausetime is a uint32_t variable, contains the connecting pause time in minutes. After closing the connection the application must wait a specified time before the new connection.
When all fields are set to 1 (full access), the time limit fields must be ignored.

Firmware version packet
This packet transmits only by the interface.

Payload contents				
Offset	Length (bytes)	Name	Value	Description
0	1	type	0x0A	Packet type
1	4	params	0x00	For Firmware packet this field value must be zeroed
5	2	len	0x0C	Payload length (in bytes)
7	1	crc		CRC of the packet (header and data)
8	4	ver		Field contains the firmware version
12	4	subversion		Field contains the firmware subversion
16	4	pointversion		Field contains the firmware modification

Packet aggregation
Packet segmentation
For sending multiple packets at once, the application needs to place those packets one-by-one in the TCP segment. Total size of packets must not exceed 1446 bytes (MTU is 1500, minus header size 54 bytes). The TILP device can also combine packets in the one TCP segment.



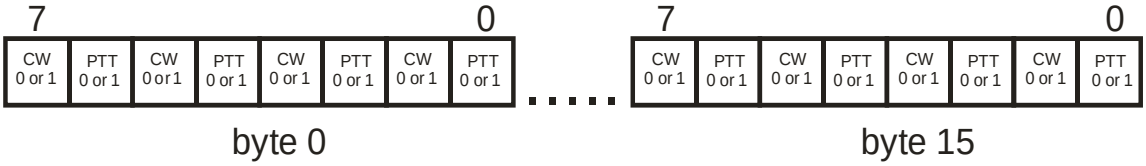
Alignment
All structures must be aligned on 1-byte boundaries.
Audio stream
Audio data is sent by the UDP protocol. The audio stream is RTP packets by RFC 3550. The audio stream from the interface to the application is always stereo. The audio stream from the application to the interface is always mono. The audio stream is full duplex. The interface sends audio data on that port and address, from which audio stream was received. When the TCP connection is closed, the audio stream must be terminated.

PTT/CW stream
For sending the CW manipulation state next packet type is used:

Num	Ts	Data
4 bytes	4 bytes	16 bytes

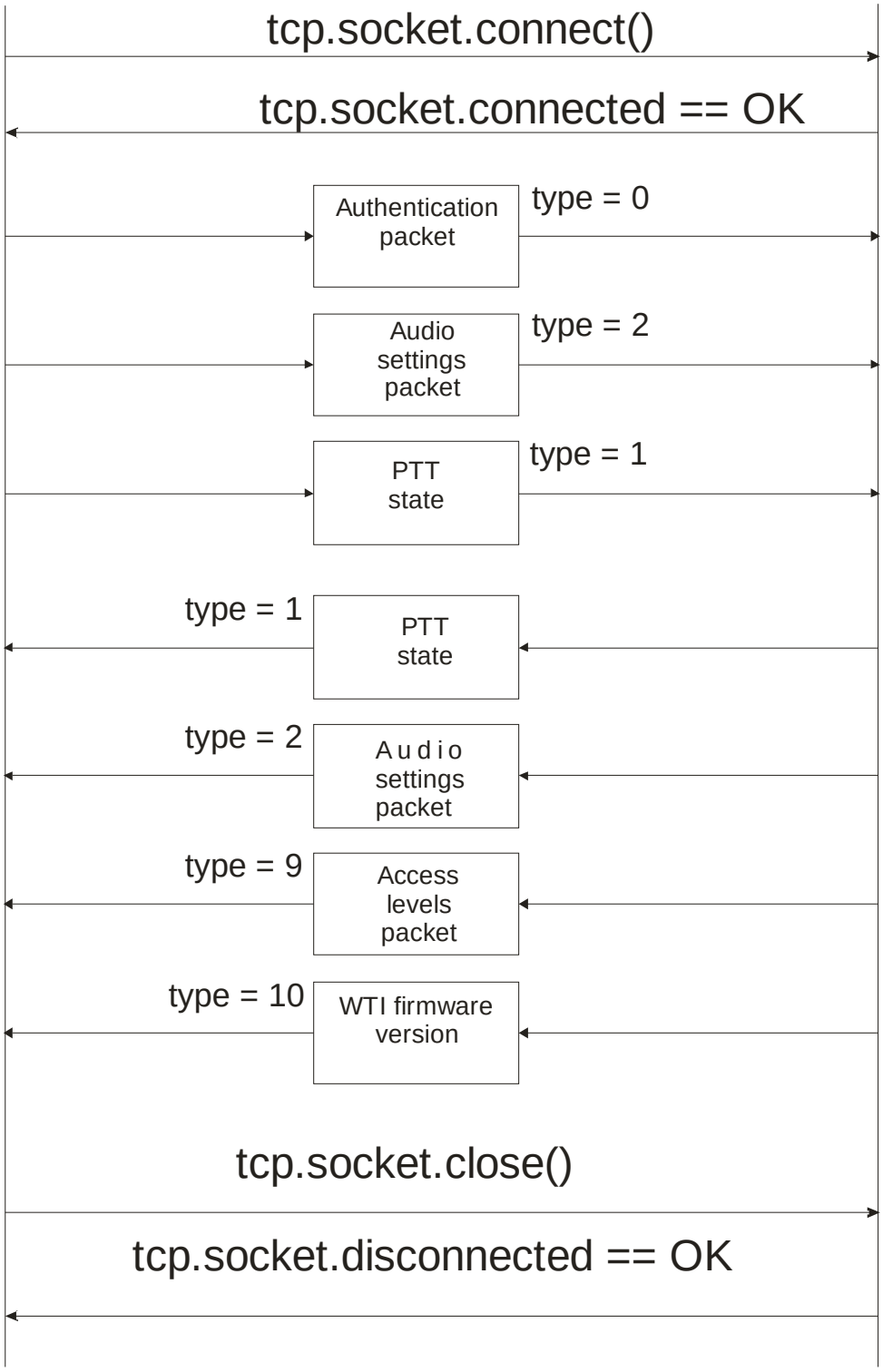
Num – packet number
Ts – timestamp in milliseconds
Data – PTT and CW state

This stream is sending onto the separate UDP port.
The PTT and CW states are stored in the following format.

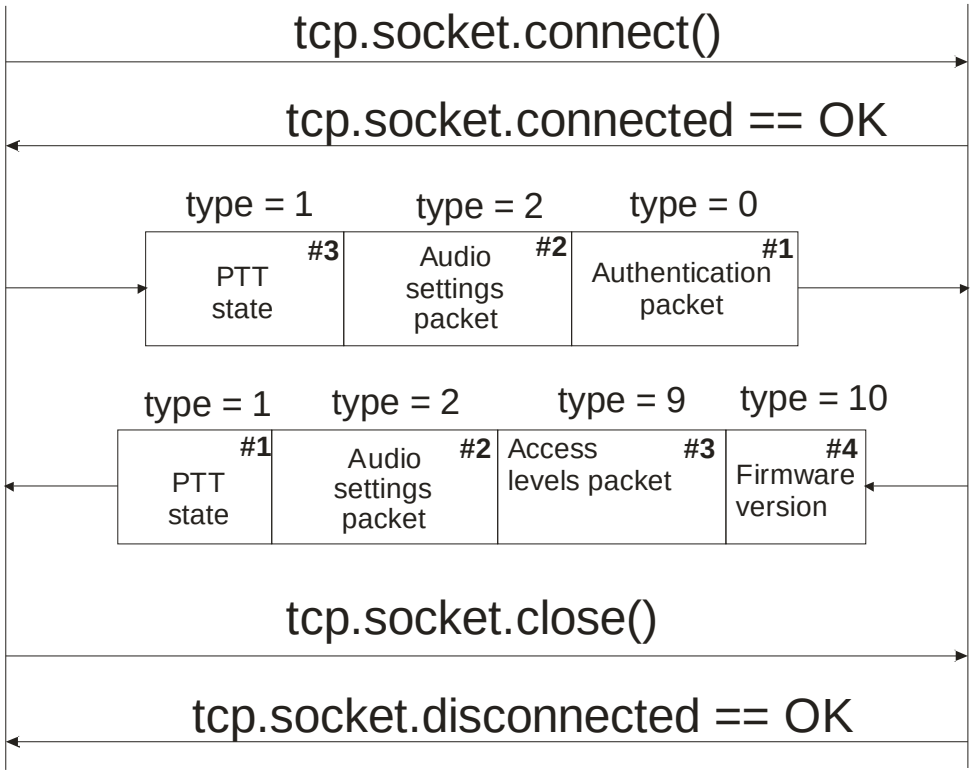


DTR and RTS pins of the serial port are polled by timer with the 1ms interval. When the packet is filled with 16 states, the CW state packet is sending by the UDP to the address and port, as specified in settings. After sending the packet, the previous packet is resending (for redundancy backup). The current packet will be resend after sending the next packet. The polling and sending packets start after the connection is set, and terminated after the connection is closed.

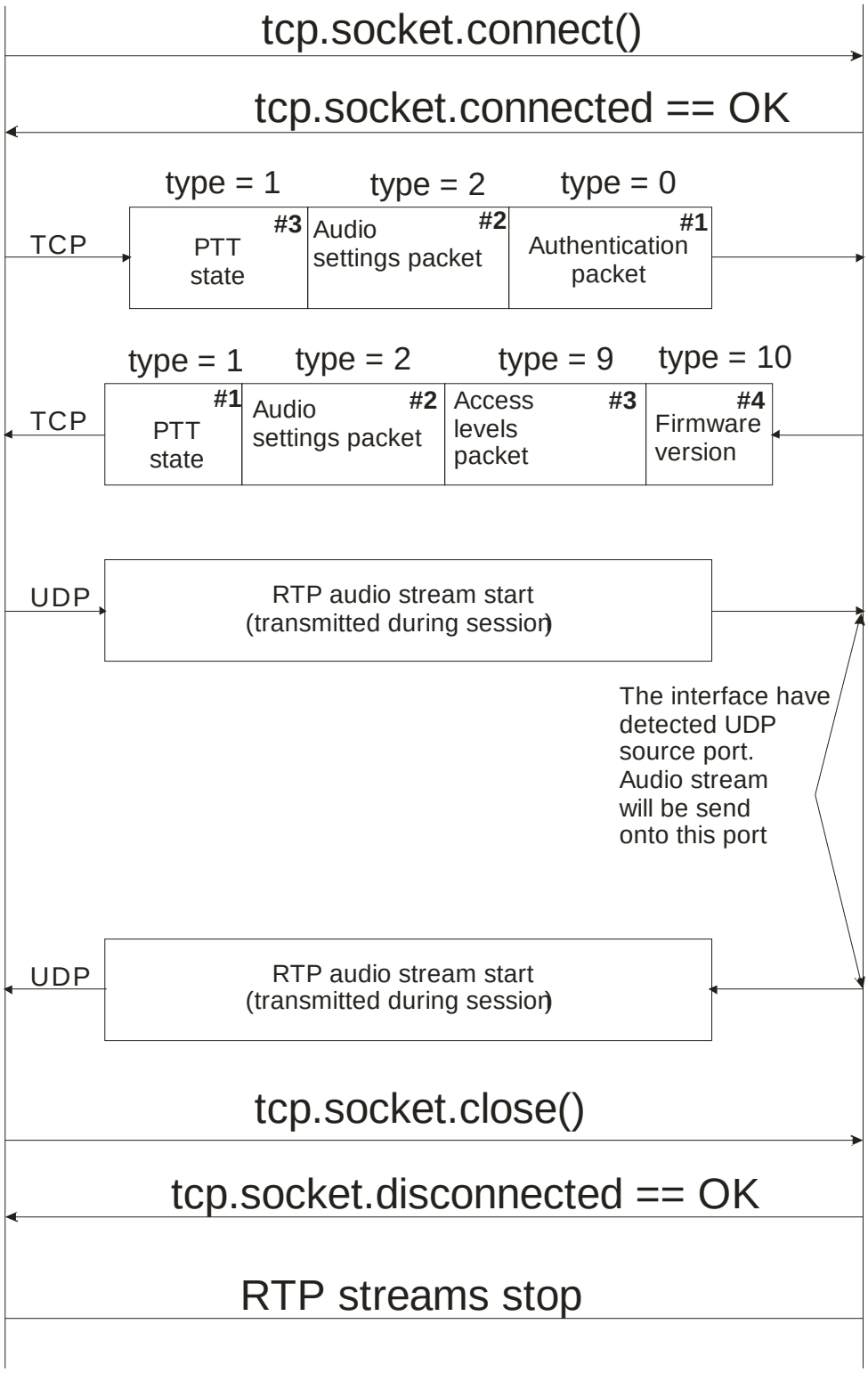
Appendix A An example of the short session



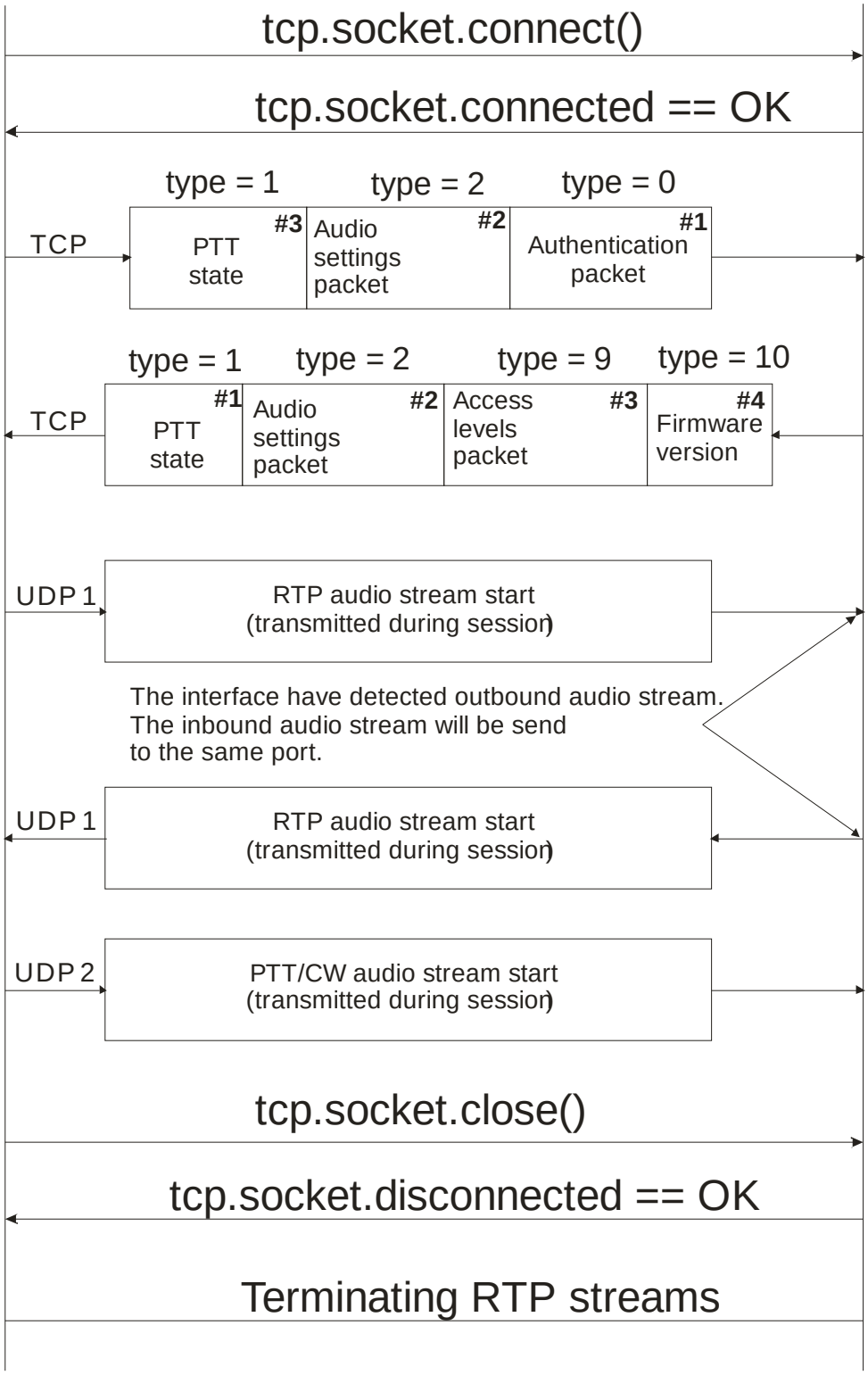
Appendix B The short session with the aggregated packets



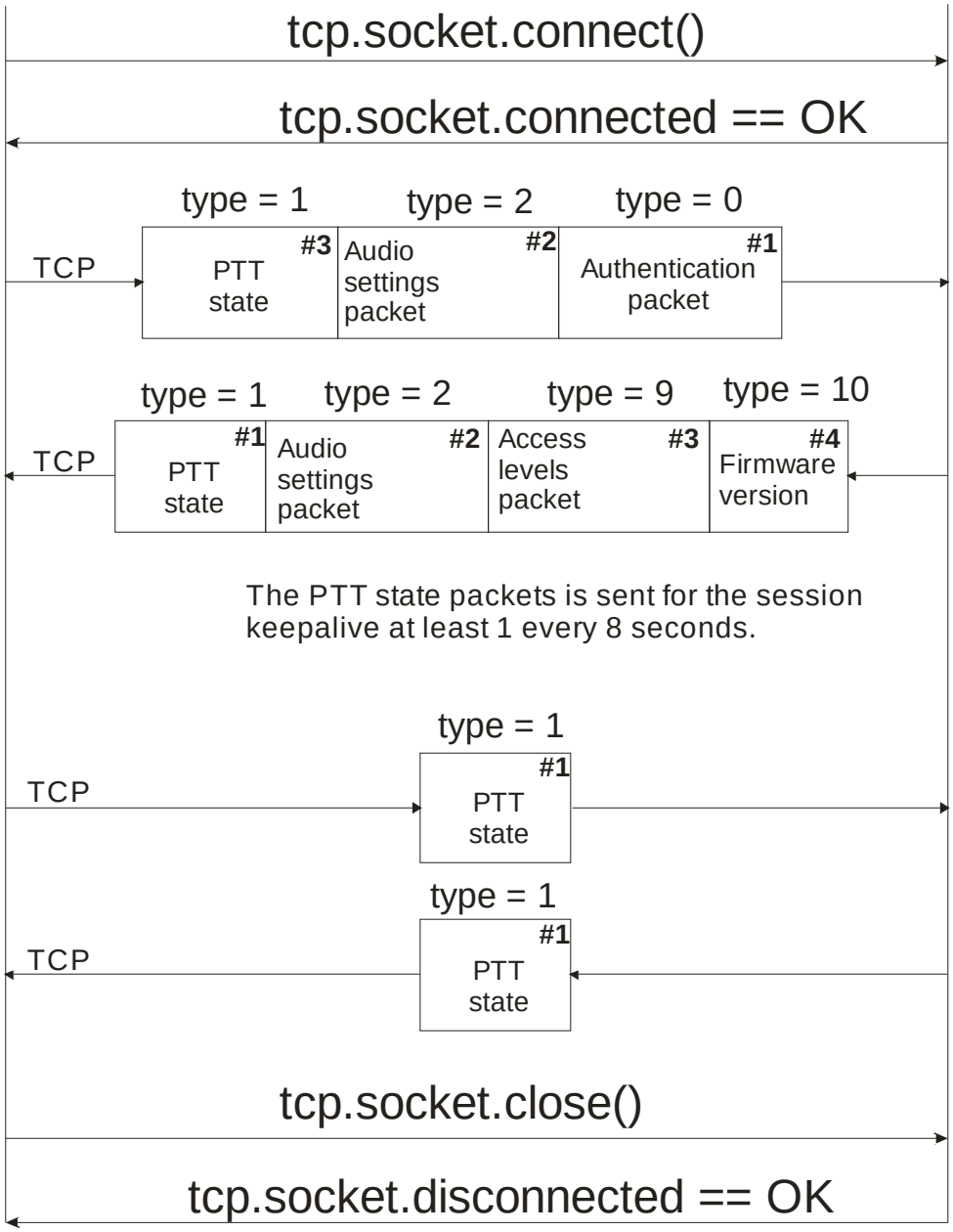
Appendix C The session with the audio streams example



Appendix D The session with the audio and CW/PTT streams



Appendix E The session keepalive example



Appendix F The serial ports data transfer example