Internet Draft
Document: draft-casagranda-vignaroli-btftp-01.txt
Category: Experimental

P.Casagranda
L.Vignaroli
RAI - CRIT

December 2000

Broadcast Trivial File Transfer Protocol

Status of this Memo

Abstract

This document describes the Broadcast Trivial File Transfer Protocol (BTFTP), a protocol designed to be used as a datagram based, multicast enabled protocol with optional back-channel. The main purpose of the protocol is data and file transfer over broadcast, wireless channels (e.g. Digital Video Broadcasting channels, like terrestrial DVB-T and satellite DVB-S [ISO13818]) without the need of a back-channel (which is optional). The efficiency of the protocol over a satellite and terrestrial wireless link has been widely tested.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

Acknowledgements

The BFTP protocol has some functionality of TFTP protocol [RFC1350] but it has very few structural similarities with it.

Table of Contents

1. Overview of the Protocol

The BTFTP protocol was created for the purpose of data and file transfer over digital broadcast channels, DVB like (DVB-S, DVB-T). The peculiar features of the protocol are:
a) optional back-channel: the protocol works without back-channel, in a pure broadcasting way; an optional back-channel is nevertheless supported to increase reliability
b) very low header overhead: the headers are kept as small as possible
c) simplicity: the specification and implementation are very simple (more straightforward than DVB Object Carousel for example)
d) extensibility: the BTFTPWriteRequest packet can easily be improved for particular needs.
e) compatibility: is an IP based protocol (instead of DVB Data Carousel and Object Carousel Protocols)

The protocol was thought to be implemented over User Datagram Protocol (UDP) but this does not exclude other datagram protocols. In that way it can be encapsulated in Digital Video Broadcasting transport stream (Multi-Protocol Encapsulation Profile [ISO13818]) and sent over wireless, broadcasting channels. BTFTP can work in two ways: without the back-channel and with a low bit-rate back-channel.

- Operation without back-channel
The protocol can be used without a back-channel, in a pure broadcasting manner, with no interactivity. This is the main purpose of the protocol.

In that way the contents are fixed, and the user selects a channel and then begins downloading all or part of the data of that channel. Reliability is obtained in two ways: with the Forward Error Correction (FEC) of the underlying layers (e.g. Viterbi and Reed Solomon protection of a DVB channel) and/or redundancy, that is repeating the transmission.

- Operation with back-channel

The protocol can be used with a low bit-rate back-channel. This adds reliability at the expense of another channel that should carry the packet requests. The protocol in itself does not implement interactivity (it was not designed for this scenario): for this purpose HTTP or other protocols can be possibly used instead of BTFTP. The check of data integrity is provided in both scenarios by a CRC32 code.

2. Protocol Specification

This section describes the format of the BTFTP packets. Below is shown the syntax of each type of packet and the semantic of each field.

2.1. BTFTPPacket

This is the primary packet which contains the common header of all BTFTP packets. The packet format is:

```
BTFTPPacket()                Length in bits (big endian notation)
{
TID;                                        32
OPCODE;                                     8
BlockNumber;                                32
BlockSize;                                  16
FutureUse;                                  5
LastPacketFlag;                             1
RedundancyFlags;                            2
if  (OPCODE==0)
             BTFTPWriteRequest();
if (OPCODE==1)
             BTFTPData();
if (OPCODE==2)
             BTFTPNak();
if (OPCODE==3)
             BTFTPWriteRequestExtension();
```

```
if (RedundancyFlags==01)
            Chekcsum;                       32
If (RedundancyFlags==10)
            CRC32;                          32
If (RedundancyFlags==00)
            Ignored;                        32
}
```

Semantics:
TID: it is the Transmission Identifier which is used to differentiate each file transmission from others, in this way we can have different multiplexed files transmissions. The value of this field is the same for each packet of the same file transmission, and must be different from other file transmissions at the same time.

OPCODE: this field contains the operation code of current packet, this value specified the packet type which can be BTFTPWriteRequest (value = 0), BTFTPData (value = 1), BTFTPNak (value = 2) or BTFTPWriteRequestExtension (value = 3).
BlockNumber: this is the packet counter of the protocol, it is used to enumerate packets of same type; the start value is one.
BlockSize: It is the size in bytes of the current packet. There are no restrictions in the use of this field but it should be used a constant BlockSize value for BTFTPData packets for each file transmission.
FutureUse: five bits field for future use.
LastPacketFlag: this field is used only with OPCODE = 1 (BTFTPData), the value 1 indicates that this is the last data packet of a trasmitted file. If OPCODE is not BTFTPData the field is ignored.
RedundancyFlags: these two bits specify the redundancy type of the BTFTP packet.
Checksum: this is a 32 bits field which contain the sum of all bytes in the packet without carry.
CRC32: this is an 32 bits field which contain the CRC calculated over this packet, which must be calculated in according to MPEG 2 system specifications, refer to [ISO13818]
Ignored: if RedundancyFlags is 00 these 32 bits are ignored by the decoding procedure.

2.2. BTFTPWriteRequest (OPCODE = 0)

The Write Request Packet is used to inform the receiver that a transmission is beginning, this packet carries the information necessary to file decoding, it contains information like file name

(must be present if a file is trasmitted), file type or generic file information.
All information in the Write Request part of packet are encapsulated as TLV (Type Length Value) format.
There is only one Write Request packet for each file transmission (same TID), if the packet is to short to encapsulate file information or if there is the necessity to transmit extra file description it must be done using the BTFTPWriteRequestExtension.
The format of the Write Request is:

```
BTFTPWriteRequest()          Length in bits (big endian notation)
{
     TotalBlockNumberWRE;                  32
     TotalBlockNumberData;                 32
     TLV();
}
```

Semantics:
TotalBlockNumberWRE: this field carries to the receiver the total block number of Write Request Extension packets. If the value is 0 there are not Extension packets. Otherwise the value specify how many Write Request Extension packets the receiver must read to completely rebuild data description.

TotalBlockNumberData: this field indicates the number of packets containing data, the value specifies how many BTFTPData packets the receiver has to read to rebuild the transmitted file.
TLV(): indicates a sequential list of TLV fields which contains the data description, the file name TLV is mandatory. This TLV list terminates with a type '0' TLV. Look at TLV section for more information. So, the general form of a TLV() is:
<Type><Length><TLV content (when specified can be omitted)>

2.3. BTFTPData (OPCODE =1)

This is the data packet of the protocol. A portion of the file to transmit is stored in the payload. The length of payload depends on the BlockSize value and on the extra information optionally encapsulated.
The format of this packet is:

```
BTFTPData()
{
     TLV();
     Payload;
}
```

Semantics:
TLV(): indicates an optional sequential list of TLV field which contains the extra data description, the type '0' TLV is mandatory. This TLV list must terminate with a type '0' TLV. See TLV section for more information.
Payload: it contains a portion of transmitted file, the size depends on the BlockSize value and the length of previous TLV list (look at the meaning of BlockSize).

2.4. BTFTPNak (OPCODE = 2)

This type of packet is used only in operation with back channel. BTFTPNak packets are sent by the receiver to the transmitter. BTFTPNak packets must be sent with the same TID of the requested packet and must be sigle packets (the BTFTPNak cannot be on multiple packets). For example, if packet n of a transmission with TID m has been lost, the BTFTPNak request is a packet with the same TID m. BTFTPNak packets contain requests of lost packets.
The format of this packet is:

```
BTFTPNak()
{
     TLV();
}
```

Semantics:
TLV(): this is one Nak TLV field following by a type æ0Æ TLV. The Nak TLV indicate the list of lost packets.

The TLV with the request(s) is of type 128: it contains a list of intervals of packets to be retransmitted (Nak list). Each entry of the list is 9 octets long. The format of the list is:
     <OPCODE, 1 byte><Block number of first packet, 4 bytes><Number of packets, 4 bytes>
     <OPCODE><Block number of first packet><Number of packets> à
The notation is big endian. For example, if data packet 0x0A31 (hexadecimal) had been lost, the TLV content would be:
     0x01 0x00 0x00 0x0A 0x31 0x00 0x00 0x00 0x01
If there were 0x39 lost packet from packet number 0x0A31, and 0x28 from data packet number 0xBA13, the content of the TLV would be:
     0x01 0x00 0x00 0x0A 0x31 0x00 0x00 0x00 0x39
     0x01 0x00 0x00 0xBA 0x13 0x00 0x00 0x00 0x28

2.5. BTFTPWriteRequestExtension (OPCODE = 3)

This type of packet is optional and is used only if one BTFTPWriteRequest packet can not carry all the file and transmission description (because the description length is greater than the payload size of the BTFTPWriteRequest packet).
The format is:

```
BTFTPWriteRequestExtension()
{
     TLV();
}
```

Semantics:
TLV(): it contains a list of TLV field following by a type '0' TLV.

2.6. Types used in TLV fields

The TLV fields need a coherent convention to use the Type field. In general, the number of bytes of the field Length depends on the field Type; in other words it is not fixed for all Types.
The Type field has to be specified with the following convention:
Type = 0: There are no Length and Value fields. This must be the last TLV of the list.
Type = 1: Length of 2 bytes. The Value contains the file name.
Type = 2: Length of 2 bytes. The Value contains the type of the file.
Type = 3: Length of 2 bytes. The Value contains information on the file.
Type = 4: Length of 2 bytes. The Value contains a command file, to be executed on a local machine.
Type = 128: Length of 2 bytes. The Value contains a Nak list.

In greater detail we have:
Type = 0: This Type tells the decoder that the TLV sequence has come to an end.

Type = 1: The file name of the file that is going to be transmitted. The filename is encoded in standard 8 bits ASCII code.
Type = 2: The type of the file. The possible types (e.g. gzip, text, html, png...) are not standardized in this document. The type is encoded in standard 8 bits ASCII code.
Type = 3: Some useful information on the file. This can be an ASCII string that describes the file. It is, like the other TLV, optional. The information is encoded using standard 8 bits ASCII code.
Type = 4: Sometimes it is useful to have a command file to be

executed at application level and to be transmitted very rapidly, possibly on a single packet, which is the purpose of this TLV. This should be a text file.
Type = 128: It contains a list of intervals of packets to be retransmitted (Nak list). Each entry of the list is 9 octets long. The format of the list is:
<OPCODE, 1 byte><Block number of first packet, 4 bytes> <Number of packets, 4 bytes>
<OPCODE><Block number of first packet><Number of packets> ...

Other values for the Type field are reserved for future use.

3. Typical Sequence of Operation

This chapter describes two typical sequences of operation and it should clarify the previously stated concepts. Note that this document is not intended to specify a policy for requests (time-out algorithm, request reiteration, et al.)

3.1. Operation without Back-Channel

In a typical scenario without back-channel, let's suppose the Receiver has begun listening on a channel (address and port). The Broadcaster begins transmitting a file:

Transmitter	Receiver
BTFTPWriteRequest	Receiving BTFTPWriteRequest, checking CRC32
BTFTPData	Receiving BTFTPData, checking CRC32 and sequence number of the packet.
	The packet is out of sequence, set a timeout for previous packets.
	The packet is in sequence, but CRC32 is wrong: that TID is discarded.
	The timeout for that TID is reached: that TID is discarded.
	The packet is in sequence and CRC32 is right: continue until the last packet is correctly received.

3.2. Operation with Back-Channel

In a typical scenario with back-channel, let's suppose the Receiver has begun listening on a channel (address and port). There is also a

channel for request packets, that can be chosen as a low bit-rate channel. The Broadcaster begins transmitting a file:

Transmitter	Receiver
BTFTPWriteRequest	Receiving BTFTPWriteRequest, checking CRC32
BTFTPData	Receiving BTFTPData, checking CRC32 and sequence number of the packet. If the packet is out of sequence, set a timeout for previous packets. If the packet is in sequence, but CRC32 is wrong: add to the list of packets to be requested. Periodically send a BTFTPAcknowledgement packet to request lost or wrong packets. If timeout for that TID is reached: that TID is discarded. If the packet is in sequence and CRC32 is right: continue until the last packet is correctly received.

The implementation can chose a clever policy to request the lost packets and minimize the band of the back-channel; this policy is not specified in this document.

4. Security Considerations

The protocol has to work without a back-channel too (and it is the main purpose for which it has been developed). So a session canÆt be established in all scenarios. The content can be protected through a mechanism like this:

- the Receiver creates a private key and a public key pair
- the Receiver sends the public key to the Transmitter, out of band if there is no back-channel
- the Transmitter creates periodically symmetric keys (session keys). These keys are sent to the Receiver, encrypted with the ReceiverÆs public key
- the Transmitter sends content (files) to the Receiver encrypting it with the session key

The protocol works on an UDP/IP stack, so many security considerations concerning protocol on this stack can be applied to the BTFTP protocol.

5. References

[RFC1350] K. Sollins: "The TFTP Protocol - Revision 2", RFC1350, July 1992

[RFC2119] S. Bradner: ôKey words for use in RFCs to Indicate Requirement Levelsö, RFC2119, March 1997
[ISO13818] "ISO/IEC 13818 Specification"

6. Authors' Addresses

Paolo Casagranda
RAI - CRIT
Corso Giambone 68
10135 û Torino
ITALY
Email: p.casagranda@rai.it

Luca Vignaroli
RAI - CRIT
Corso Giambone 68
10135 û Torino
ITALY
Email: l.vignaroli@rai.it

Full Copyright Statement