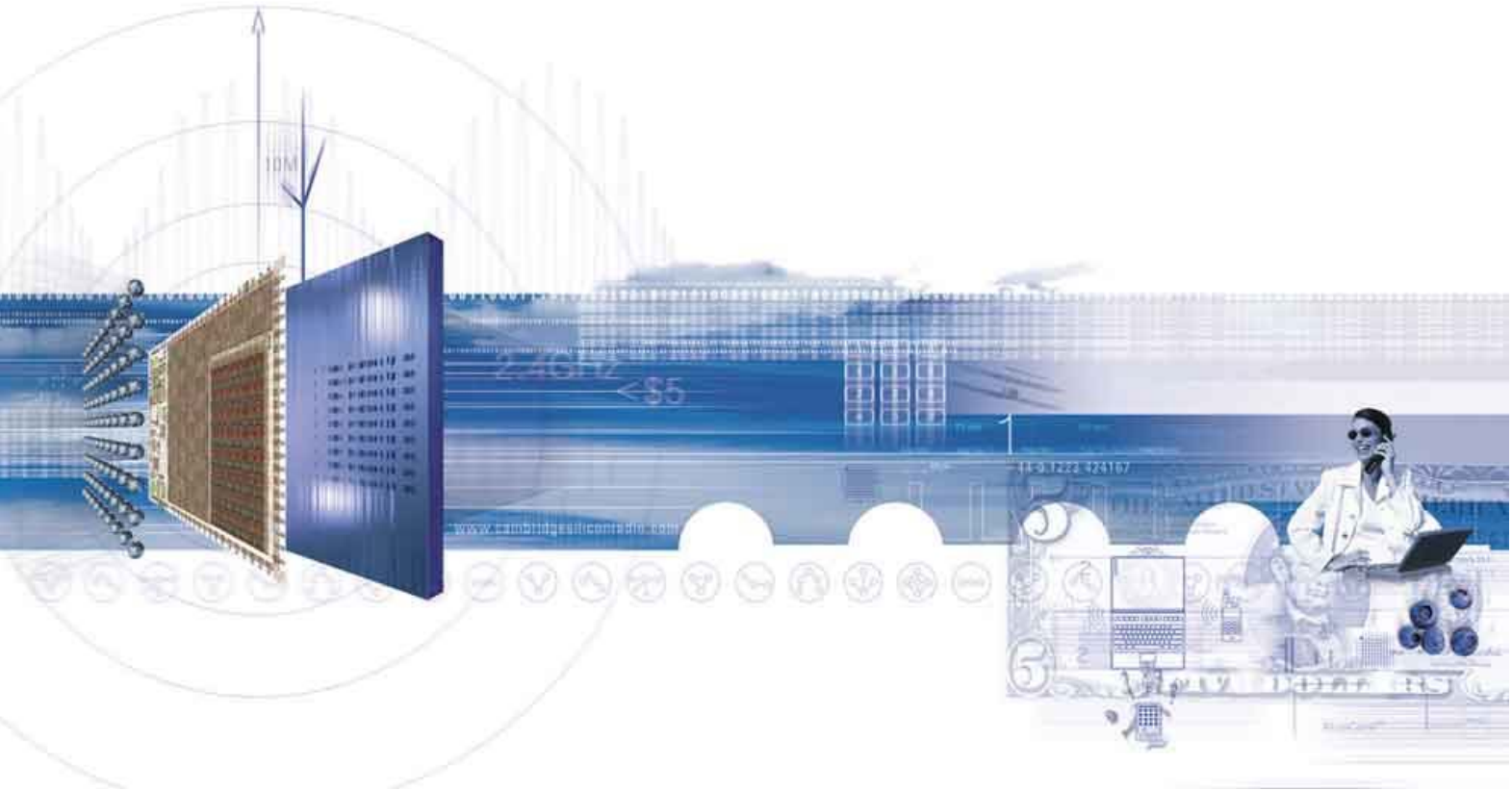
BlueCore™

BlueCore Serial Protocol (BCSP)

July 2004



CSR

Cambridge Science Park
Milton Road
Cambridge
CB4 0WH
United Kingdom

Registered in England 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

Contents **2**
1 Introduction **4**
2 Context **5**
3 Overview **6**
4 Packet Structure **7**
4.1 Flags Field 7
4.1.1 Seq Field 7
4.1.2 Ack Field 7
4.1.3 CRC Present Field 8
4.1.4 Protocol Type Field 8
4.2 The Protocol Identifier Field 8
4.3 Payload Length Field 8
4.4 Checksum Field 8
4.5 Payload 8
4.6 CRC Field 9
5 UART Driver Layer **10**
6 SLIP Layer **11**
6.1 Transmitting Packets 11
6.2 Receiving Packets 11
7 Packet Integrity Layer **12**
7.1 Transmitting Packets 12
7.2 Receiving Packets 12
8 MUX Layer **13**
8.1 MUX Layer Context 13
8.2 Receiving Packets 13
8.3 Transmitting Packets 14
8.4 Acknowledging Packets 14
8.5 Ack Packets 15
8.6 Choke 15
8.7 Reset 16
9 Sequencing Layer **17**
9.1 Acknowledgement and Retransmission 17
9.2 Sequencing Layer Context 18
9.3 Top Level View 19
9.3.1 tx_reliable_pkt() 19
9.3.2 rx_reliable_pkt() 19
9.3.3 link_failed() 19
9.4 Transmit State Machine 20
9.5 Receive State Machine 21
9.6 Reset 23
9.7 Protocol Identifier Field 23
10 Datagram Queue Layer **24**
10.1 Datagram Queue Layer Context 24
10.2 Datagram Queue Layer Functions 24
10.2.1 Receiving Messages 24
10.2.2 Sending Messages 24
10.2.3 Seq Field 24
10.2.4 Protocol Identifier Field 25

10.2.5 Reset 25
10.3 Top Level View 25
10.3.1 tx_unreliable_pkt() 25
10.3.2 rx_unreliable_pkt() 25
11 Configurable Items 26
12 Comments 27
Document References 28
Acronyms and Definitions 29
Record of Changes 30

List of Figures

Figure 1.1: UART Host Connection 4
Figure 2.1: BCSP Context 5
Figure 3.1: BCSP Stack Elements 6
Figure 4.1: BCSP Packet Structure 7
Figure 4.2: Flags Field 7
Figure 4.3: BCSP Packet Structure with CRC Field 9
Figure 6.1:SLIP Packets 11
Figure 8.1: MUX Layer Context 13
Figure 8.2: Acknowledging Packets 14
Figure 9.1: Acknowledgement and Retransmission 17
Figure 9.2: Sequencing Layer Context 18
Figure 9.3: Transmit State Machine 21
Figure 9.4: Receive State Machine 22
Figure 10.1: Datagram Queue Layer 24

List of Tables

Table 8.1: Ack Packet Fields 15
Table 9.1: Transmit State Machine Variables 20
Table 9.2: Transmit State Machine Constants 20
Table 9.3: Receive State Machine Variables 21
Table 11.1: Configurable Items 26

1 Introduction

This document defines BlueCore Serial Protocol (BCSP), a protocol used to carry a set of data flows through a highly reliable UART link.

The stack has been designed to transfer data between a Bluetooth Host and a Bluetooth Host Controller. The stack is intended to be used to carry the Bluetooth HCI (Host Controller Interface) protocols plus several others.

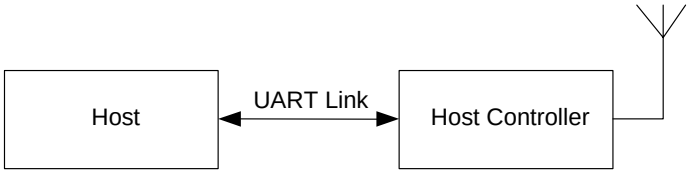


Figure 1.1: UART Host Connection

BCSP is intended to be used on the CSR BlueCore Host Controller, which provides hardware to support much of the stack's functionality.

2 Context

BCSP is used to control and format information that flows between a Bluetooth Host and a Bluetooth Host Controller, as described in [BT1.2]. The stack carries a set of parallel information flows between the two computers, multiplexing them over a single UART link.

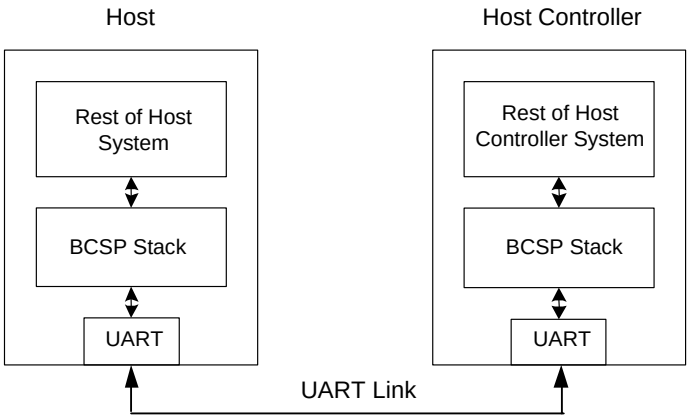


Figure 2.1: BCSP Context

An instance of the BCSP stack runs on the Host and the Host Controller.

The BCSP stack is layered above the UART on each computer.

The top of the BCSP stack presents:

- One bidirectional reliable datagram service
- One bidirectional unreliable datagram service

Higher protocol layers can be built upon the two datagram services.

The protocol is defined to run on a 3-wire UART connection (two data lines plus ground). However, BCSP can also be run on a 5-wire UART connection (with two flow control lines), and this is common, particularly at higher baud rates.

3 Overview

This section gives an overview of the elements of a BCSP stack.

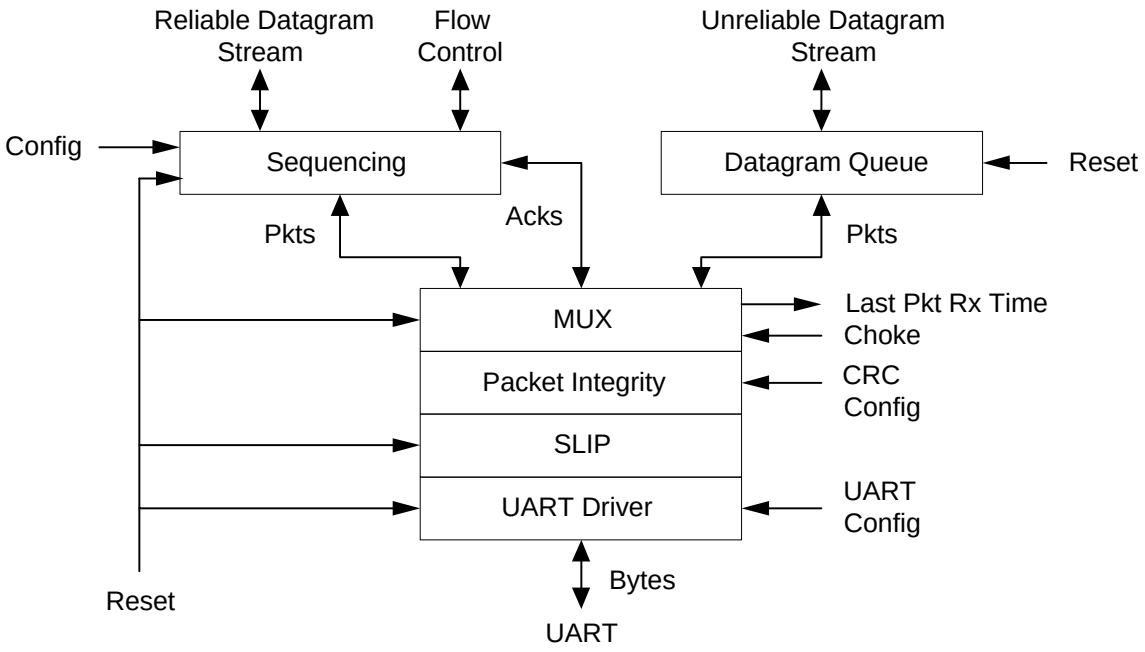


Figure 3.1: BCSP Stack Elements

Considering the diagram (Figure 3.1) from the bottom:

- The UART Driver Layer initialises and controls the local UART, translating the flows of bytes on the physical UART connection to the peer computer into flows of bytes at the base of the SLIP Layer.
- The SLIP Layer uses the Serial Link Internet Protocol (SLIP) to transform the flow of bytes into a flow of packets.
- The Packet Integrity Layer ensures that packets received from the SLIP layer are intact.
- The MUX Layer routes received packets either to the Sequencing Layer or to the Datagram Queue Layer. The MUX Layer also keeps a note of the time at which the last packet was last successfully received.
- The Sequencing Layer uses a windowing mechanism to provide a single reliable flow of packets to and from the peer. Code above the Layer can route packets using a Protocol Identifier value, which is carried with each packet.
- The Datagram Queue Layer provides a single unreliable flow of packets to and from the peer. As with the Sequencing Layer, code above this Layer can route packets using a Protocol Identifier value, which is carried with each packet.

Later sections of this document describe the layers in more detail.

4 Packet Structure

The BCSP uses only one packet type; this section describes the packet's structure.

Figure 4.1 shows the packet structure used by all BCSP layers above the SLIP Layer.

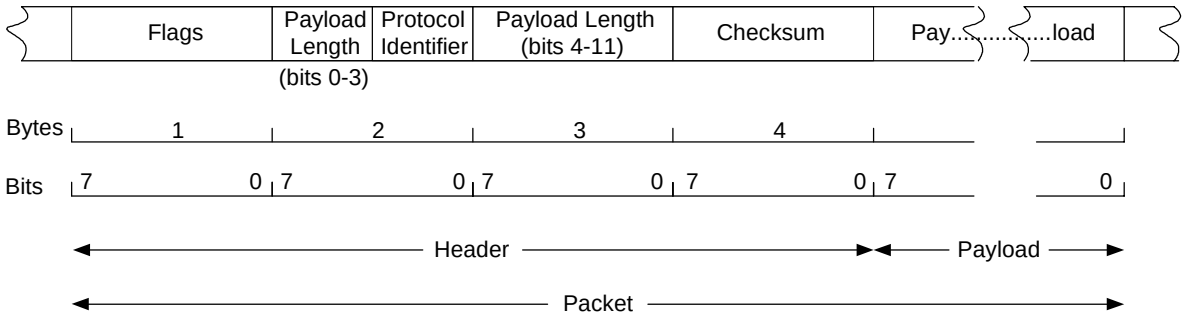


Figure 4.1: BCSP Packet Structure

The packet's fields are described in the following sections.

4.1 Flags Field

The single byte Flags Field has the following structure:

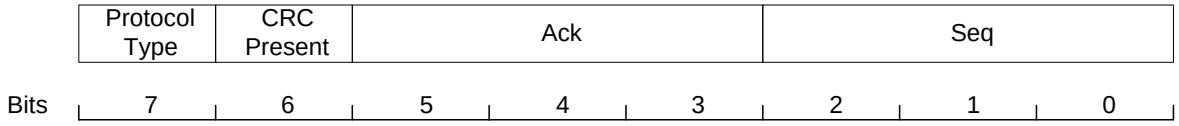


Figure 4.2: Flags Field

The fields within the Flags Field are described below.

4.1.1 Seq Field

The Seq Field holds a 3-bit, little-endian sequence number. This is of interest only to packets that pass into the Sequencing Layer.

The Sequencing Layer sets the Seq Field to carry an incrementing sequence number, range 0 to 7, on each (fresh) packet that it sends to the peer computer.

Packets that have the Protocol Type Field set to "Unreliable Datagram Stream" (see section 4.1.4 below) have their Seq Field set to zero.

4.1.2 Ack Field

The Ack Field holds a 3-bit, little-endian sequence acknowledgement number. This field normally holds the value of the Seq Field in the last packet successfully received by the peer's Sequencing Layer, plus one. Put more simply, this is the value of the Seq Field that the peer expects to see in the next packet it receives.

For example, if one computer's Sequencing Layer last successfully received a packet with its Seq Field set to 3, then that computer will set the Ack Field in all packets that it sends to the peer to 4 (until it successfully receives another packet, etc.).

The Sequencing Layer processes the value of the Ack Field in *all* packets received from the peer.

4.1.3 CRC Present Field

The CRC Present Field indicates whether the packet payload carries a CRC Field; this is described below. This Field's values are:

0	CRC Field not present
1	CRC Field present

4.1.4 Protocol Type Field

The Protocol Type Field is used only by the MUX Layer, and indicates whether the packet is part of the reliable or unreliable data stream. The Field's values are:

0	Unreliable Datagram Stream
1	Reliable Datagram Stream

4.2 The Protocol Identifier Field

Each packet that is sent into the stack, either on the reliable or unreliable datagram stream, carries with it a value for the Protocol Identifier Field. This value travels with the packet to the peer computer and is delivered with the packet.

The MUX Layer generates and consumes packets with the Protocol Identifier Field of value zero. Otherwise, BCSP takes no interest in the value of the Protocol Identifier Field in packets it transfers, other than to carry and deliver the Field's value with the packet's payload.

The higher layers of the stack may use the value of the Field to distinguish parallel flows of data that pass through a BCSP connection.

4.3 Payload Length Field

The 12-bit Payload Length Field holds the length of the packet's Payload Field in bytes.

The 12-bit Field is split over two bytes: the least significant 4 bits of the Field lie in the most significant 4 bits of the Header's second byte, the remaining 8 bits of the Field form the Header's third byte.

Values in the 4 and 8-bit parts of the Field are little-endian.

A value of 0 indicates that a Payload Field of length zero is present, so a payload of length zero should be passed up/down the stack.

4.4 Checksum Field

The Checksum Field is the bit inverse of the sum of the first three bytes in the packet, modulo 256.

This Field is used to provide assurance of the integrity of the packet's header.

4.5 Payload

The Payload Field carries the data of the packet. This is passed unaltered through a BCSP connection.

The Payload Field is an integer number of bytes in length, as indicated by the Payload Length Field.

4.6 CRC Field

If the CRC Present Field indicates that the packet carries a CRC Field, then the packet structure is slightly different:

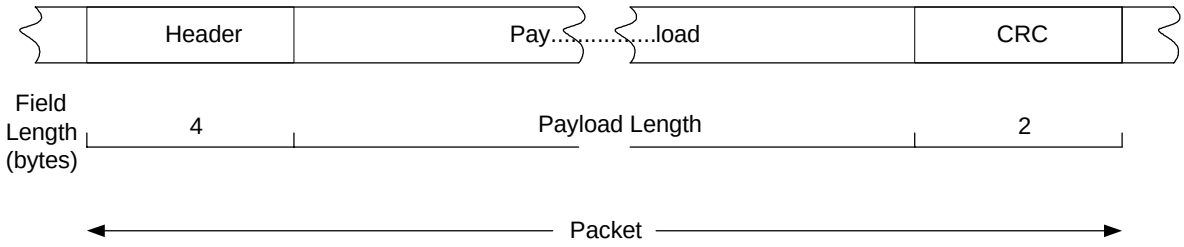


Figure 4.3: BCSP Packet Structure with CRC Field

The two byte CRC Field is only included if the CRC Present Field indicates its presence.

The definition of the Payload Length Field is unchanged if a CRC Field is present: it indicates only the length of the Payload Field.

The value of the CRC field is calculated over the Header and Payload fields.

The CRC is defined using the CRC-CCITT generator polynomial:

```
g(D) = D^16 + D^12 + D^5 + 1
```

with the following comments:

- The CRC shift register is filled with 1's before calculating the CRC for each packet.
- Bytes are fed through the CRC generator least significant bit first.
- The most significant parity byte is transmitted first (where the CRC shift register is viewed as shifting from the least significant bit towards the most significant bit). Therefore, the transmission order of the parity bytes within the CRC shift register is as follows:

  ```
  x[8] (first), x[9],..., x[15], x[0], x[1],..., x[7] (last)
  ```

  where `x[15]` corresponds to the highest power CRC coefficient and `x[0]` corresponds to the lowest power coefficient.

5 UART Driver Layer

The UART Driver Layer of the stack initialises and controls the local UART. The Layer transfers bytes between the local UART and the bottom of the SLIP Layer.

The local UART connects to the peer computer via a physical link with the following default characteristics:

- A three-wire link: data in both directions plus a common ground
- No hardware flow control signals
- 8 data bits, transmitted least significant bit first
- Even parity
- One stop bit
- 38.4kbaud

If a received byte fails its parity test then the byte is silently discarded.

The baud rate, number of stop bits, parity on/off, parity even/odd and use of hardware flow control may be changed by local configuration.

If the Layer is reset, then it restores the configuration listed above.

6 SLIP Layer

The SLIP (Serial Link Internet Protocol) Layer implements a version of the SLIP protocol described in Internet standard RFC 1055.

Fundamentally, SLIP translates between a byte stream and a packet stream. The UART Driver Layer transfers a pair of byte streams, so the SLIP Layer converts this into a pair of packet streams. More exactly, the UART link deals with *unreliable* byte streams, so the SLIP Layer turns these into *unreliable* packet streams.

6.1 Transmitting Packets

The SLIP Layer performs the following procedure for each packet as it passes from the Packet Integrity Layer to the UART Driver Layer:

A byte of value `0xc0` is transmitted. (This denotes the start of the packet.)

for each byte in the packet, reading from its start:

if the byte's value is `0xc0`:

Two bytes are transmitted: values {`0xdb`, `0xdc`}

else if the byte's value is `0xdb`:

Two bytes are transmitted: values {`0xdb`, `0xdd`}

else:

The original byte is transmitted.

A byte of value `0xc0` is transmitted. (This denotes the end of the packet.)

The result of the first and last lines of this procedure is to frame each packet with a pair of marker bytes of value `0xc0`, as shown in the following diagram:

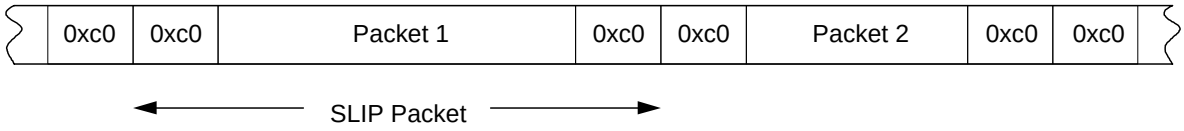


Figure 6.1: SLIP Packets

The remainder of the procedure ensures that occurrences of the marker byte are escaped within the packet body.

6.2 Receiving Packets

The SLIP Layer applies a corresponding procedure on received bytes, translating the byte stream into a packet stream. The SLIP Layer passes the received packets up to the Packet Integrity Layer.

If the SLIP Layer receives unexpected bytes from the UART Driver Layer, it silently discards bytes until it resynchronises on the expected flow of bytes.

When the Layer is reset it behaves initially as if it has lost packet synchronisation.

7 Packet Integrity Layer

The Packet Integrity Layer examines packets received from the SLIP layer and only passes them up to the MUX Layer if it determines that they are intact.

7.1 Transmitting Packets

The Packet Integrity Layer sets the following fields in each packet that it transmits to the peer:

- Payload Length Field
- Checksum Field
- CRC Present Field
- CRC Field (only set if the CRC Present Field indicates its presence)

The value of the CRC Present Field is a configurable item. By default, no CRC Field is transmitted.

7.2 Receiving Packets

The Packet Integrity Layer performs the following tests on all packets it receives from the SLIP Layer:

- Checks that the Checksum Field has the expected value.
- Checks that the packet length is as expected, after examining the Payload Length and CRC Present Fields.
- If the CRC Present Field indicates that the packet carries a CRC Field the Packet Integrity Layer checks that the CRC Field holds the expected value.

If any of these tests fail on a packet, the packet is silently discarded.

Packets passed up to the MUX Layer are regarded as being intact.

8 MUX Layer

The MUX Layer routes packets to/from the reliable and unreliable branches at the top of the BCSP.

The MUX Layer also deals with some elements of packet flow control on behalf of the Sequencing Layer.

8.1 MUX Layer Context

The diagram gives a context for the MUX Layer:

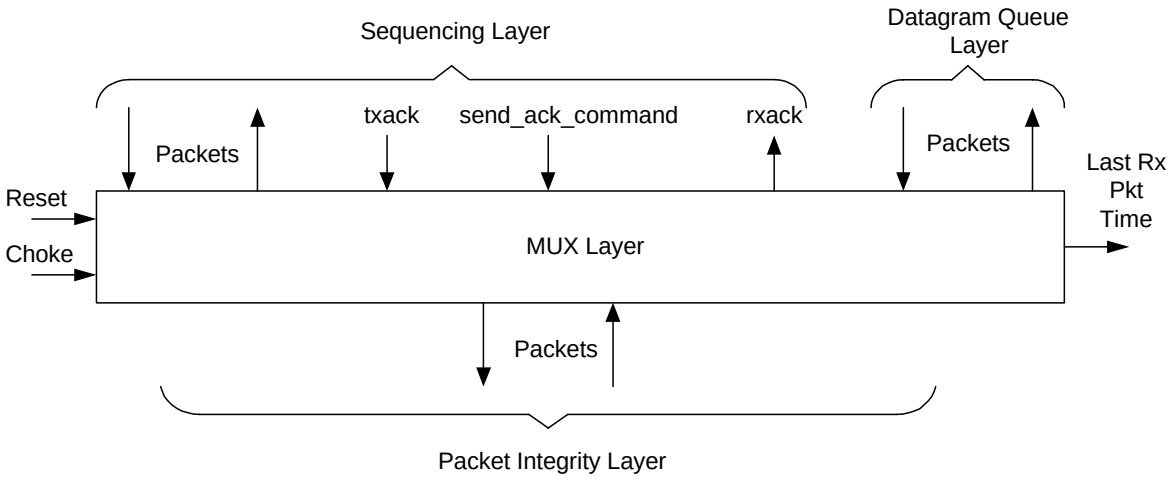


Figure 8.1: MUX Layer Context

Packets from the Packet Integrity Layer are passed up to the Sequencing and Datagram Queue layers.

Packets from the Sequencing and Datagram Queue layers are passed down to the Packet Integrity Layer.

The MUX Layer reads the value of the Ack Field from every packet received from the Packet Integrity Layer and presents this to the Sequencing Layer (rxack).

The MUX Layer reads a txack value from the Sequencing Layer. It writes this value into the Ack Field of every packet it sends to the Packet Integrity Layer.

By sending a send_ack_command, the Sequencing Layer is able to force the MUX Layer to send the current txack value to the peer.

The Layer publishes the time at which it last received a packet from the Packet Integrity Layer.

While the Boolean Choke signal is asserted only particular (UART link management) packets are passed through the Layer.

The Reset signal initialises its knowledge of the last Ack Field value read from packets from the peer (rxack). The signal also clears internal mechanisms used to send packets generated within the Layer to the peer.

8.2 Receiving Packets

For each packet received from the peer, the MUX Layer passes the value of its Ack Field up to the Sequencing Layer as signal rxack.

The MUX Layer publishes the local system time at which it last received a packet from the peer. This value allows local processes to determine that traffic is flowing freely from the peer. The recorded value must have a

resolution of at least one second and a range of at least ten minutes. This information is not maintained if the host system does not have a clock which provides the required range and resolution.

If the MUX Layer receives an Ack Packet (see section 8.5), it silently discards the packet. (The job of this packet was to transfer an Ack value from the peer.)

If the MUX Layer receives a packet that is not an Ack Packet it examines the Protocol Type Field:

- If the Field indicates that the packet is part of the reliable datagram stream then it is passed to the Sequencing Layer.
- If the Field indicates that the packet is part of the unreliable datagram stream then it is passed to the Datagram Queue Layer.

8.3 Transmitting Packets

Packets come into the top of the MUX Layer from the Sequencing and Datagram Queue layers. All of these packets are sent to the Packet Integrity Layer.

The MUX Layer sets the value of the Protocol Type Field for each packet sent; the value written depends on whether the packet came from the Datagram Queue Layer or from the Sequencing Layer.

The Sequencing Layer continuously passes a value txack into the MUX Layer, which it writes into the Ack Field of all packets transmitted.

The MUX Layer always gives priority to packets from the Datagram Queue Layer over the Sequencing Layer.

8.4 Acknowledging Packets

The MUX Layer is responsible for passing acknowledgements of the receipt of reliable datagrams to the peer. The Layer performs this on behalf of the Sequencing Layer.

The Sequencing Layer can command the MUX Layer to send an acknowledgement to the peer. If the Layer is currently sending a packet, this must complete before it can consider sending the new Ack value. The following state machine describes the mechanism for acknowledging the receipt of packets:

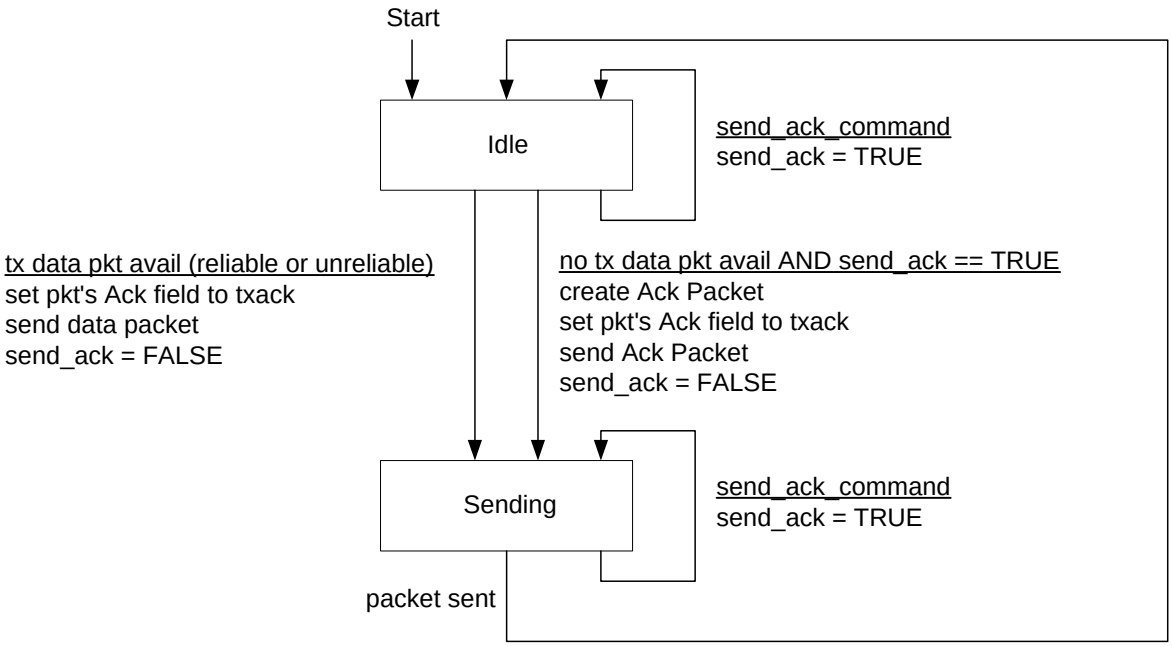


Figure 8.2: Acknowledging Packets

The MUX Layer alternates between the states of waiting for instructions and sending packets to the peer.

Whenever the Layer is asked to send a data packet (from either the Sequencing Layer or the Datagram Queue Layer) it writes the current txack value from the Sequencing Layer into the Ack field of the packet before it sends it on to the Packet Integrity Layer.

The Sequencing Layer asks the MUX Layer to send an acknowledgement packet by issuing the "send_ack_command" signal. The state machine records this in the local variable "send_ack".

If the state machine finds it is required to send the txack value to the peer but it has no normal data packet in which to send it, the state machine creates and sends an Ack Packet (described below).

Note:

The description of this Layer exists only to describe elements of behaviour required of the overall BCSP. It is expected that an implementation will not provide a message queue in this Layer, and that the Layer will need knowledge of the availability of transmit packets from higher layers.

Consequently, whenever the Sequencing Layer issues the "send_ack_command" signal the MUX Layer sends the txack value in at least one outbound packet.

The role of acknowledging the receipt of reliable packets may seem out of place in the MUX Layer, however:

- Acknowledgements are carried in both reliable and unreliable data packets, and the MUX Layer is the highest point in the stack that sees both packet types.
- Only the MUX Layer is able to detect that there are no reliable or unreliable datagrams waiting to be sent to the peer, and so it is best sited to send Ack Packets.

8.5 Ack Packets

The BCSP has only one packet type, which it uses to carry both payload data and delivery acknowledgement over the link. When an exchange of data packets comes towards its end there is no payload data packet via which to acknowledge receipt of the last data packet received from the peer. To overcome this the MUX Layer generates a packet simply to carry an acknowledgement value to the peer. This is an Ack Packet.

An Ack Packet has the following fields:

Field	Value
Ack Field	`txack` - from the Sequencing Layer
Seq Field	`0`
Protocol Identifier Field	`0`
Protocol Type Field	Any value
Payload Length Field	`0`

Table 8.1: Ack Packet Fields

8.6 Choke

While the Boolean Choke signal is not asserted the Layer behaves as described in the remainder of this Layer's description.

While the Choke signal is asserted the Layer only passes packets with:

- Protocol Type Field set to Unreliable Datagram Stream
- Protocol Identifier Field set to `1`

This control allows external software to ensure that the link is behaving correctly (by exchanging packets that pass the choke filter) before enabling the flow of normal traffic.

8.7 Reset

When the MUX Layer is reset it sets the rxack value, which it passes up to the Sequencing Layer, to the value zero. (The value will be overwritten with the value of the Ack Field from first packet received from the peer, as normal.)

When the MUX Layer is reset, it initialises the mechanism that generates Ack Packets to prevent generation of those packets until txack changes.

9 Sequencing Layer

The Sequencing Layer uses a conventional packet windowing mechanism to provide a single reliable bidirectional flow of packets, with flow control, to and from the peer.

Flow control within the Sequencing Layer is achieved by not acknowledging packets from the peer.

9.1 Acknowledgement and Retransmission

The Sequencing Layer applies a windowed acknowledgement mechanism to the message stream in order to obtain transfer reliability. The Layer also provides a simple flow control mechanism to allow a receiver of packets to signal to the sender whether it is ready to receive packets.

As is normal for such systems, independent mechanisms apply to data passed to and from the peer:

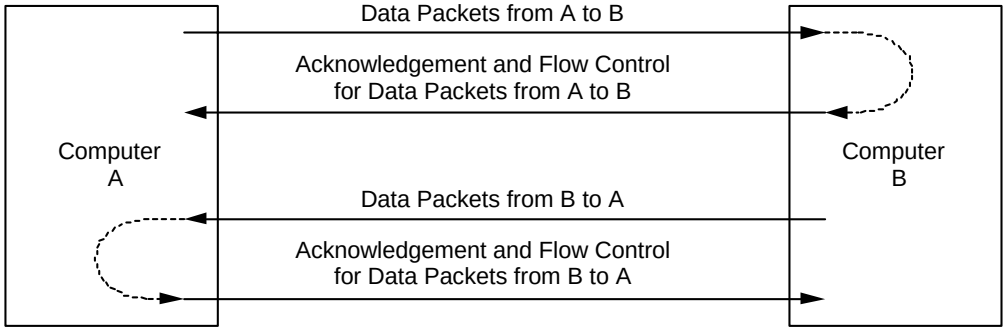


Figure 9.1: Acknowledgement and Retransmission

A data packet sent from A to B provokes an acknowledgement from B to A, which A uses to determine that B has successfully received the data packet. Also, B sends flow control information to A to signal whether A should currently send packets to B.

The same mechanism operates for data packets sent from B to A.

BCSP uses only one packet type. This carries acknowledgement and flow control information from A to B in packets that carry data from B to A and vice versa.

The Sequencing Layer also notes sequence acknowledgement numbers in packets received from the peer stack, and uses this to provoke retransmission of unacknowledged packets.

9.2 Sequencing Layer Context

The diagram shows the context of the Sequencing Layer:

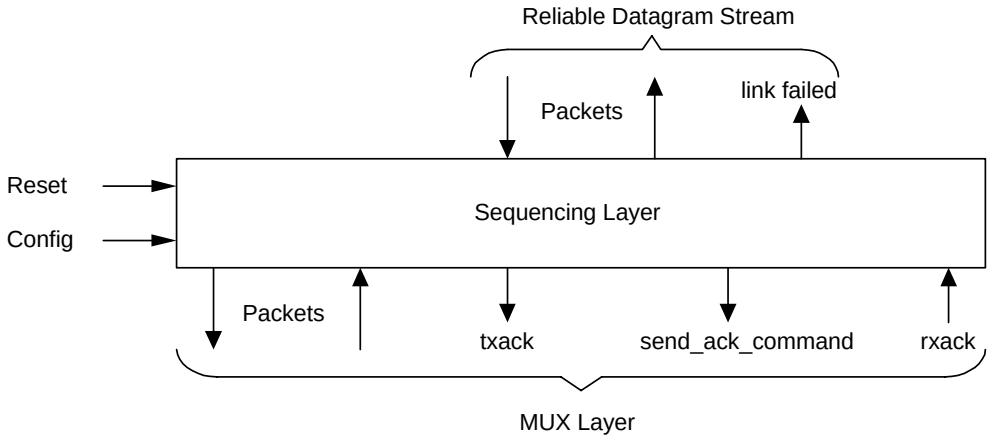


Figure 9.2: Sequencing Layer Context

Data packets are sent into the Sequencing Layer from higher layered code. These are passed to the MUX Layer, which sends them towards the peer.

Data packets received from the peer are passed up from the MUX Layer; the Sequencing Layer passes these up to the higher layers of code.

The Sequencing Layer continuously passes an acknowledgement value (txack) down into the MUX Layer. This reveals the sequence number of the last packet received from the peer by the Sequencing Layer. (The value of txack is one more than the value of the Seq Field in the last received packet, i.e. it is the Seq Field value that it expects to see in the next packet it receives.) The MUX Layer writes this into the Ack Field of *all* outbound packets.

The Sequencing Layer may also insist that the MUX Layer forwards the current txack value to the peer via the send_ack_command.

The MUX Layer continuously passes an acknowledgement value rxack up into the Sequencing Layer. This carries the value of the Ack Field from the last packet *of any type* received by the MUX Layer. The Sequencing Layer uses this to note which packets have been successfully received by the peer.

If the Sequencing Layer determines that the connection to the peer has failed, this is indicated to higher level code.

The Reset control resets the Sequencing Layer's windowing mechanism, causes the next packet to be transmitted to the peer to carry a Seq Field with value zero and sets the Layer to expect the next data packet from the peer to carry a Seq Field with value zero.

9.3 Top Level View

This section gives a notional view of how code above the Sequencing Layer might view its services. These function descriptions are for illustration only.

9.3.1 tx_reliable_pkt()

To send a data packet, higher layer code might call a function of the form:

```
bool tx_reliable_pkt(char *buf, unsigned n, unsigned protocol_id);
```

which attempts to send the `n` byte message in the buffer `buf` to the peer, tagged with the protocol identifier `protocol_id`.

The range of `n` is zero to `0xfff`, though the system's configuration is likely to limit the value more severely.

The value of `protocol_id` can range from `1` to `15`. The value zero must not be used.

The function returns TRUE if the packet has been sent on its way, or FALSE if transmission was not possible (presumably from lack of resources). This provides transmit flow control. If the function returns FALSE, the caller may attempt to send a different message and `protocol_id` with the next call.

This functional view of the interface presumes the Sequencing Layer stores a copy of the message for initial transmission and for any required retransmission. It is recognised that implementations are unlikely to be structured in this way.

9.3.2 rx_reliable_pkt()

For the Sequencing Layer to deliver a packet to higher level code, the Sequencing Layer might call a function of the form:

```
bool rx_reliable_pkt(char *buf, unsigned n, unsigned protocol_id);
```

This attempts to deliver to the higher level code the `n` byte message at `buf`, tagged with protocol identifier `protocol_id`. The message and identifier will match parameters passed into `tx_reliable_pkt()` on the peer.

The higher level may accept or refuse the message by returning TRUE or FALSE. The Sequencing Layer uses the returned Boolean value to choose whether to acknowledge delivery of the message to the peer, i.e., the Sequencing Layer uses this indication internally to support its flow control.

9.3.3 link_failed()

The Sequencing Layer might indicate to higher level code that it has determined that the link to the peer has failed with a call of the form:

```
void link_failed(void);
```

9.4 Transmit State Machine

The Sequencing Layer runs a state machine for transmitting packets. This uses the following variables:

Variable	Initial Value	Description
txseq	0	Sequence number of next transmitted packet.
txack	0	Acknowledgement number of the next transmitted packet.
rxack	(from MUX Layer)	Acknowledgement number from the current received packet.
winspace	winsize	The number of extra packets that can be sent to the peer before receiving any acknowledgements, i.e. the capacity remaining in the transmit window.
retries	0	Count of number times a message has been resent.

Table 9.1: Transmit State Machine Variables

The range of the variables txseq, txack and rxack is determined by the widths of fields Seq and Ack in the BCSP packet format: both fields are 3 bits wide, so the variables can take values between 0 and 7. Variables in these fields wrap on overflow and underflow.

The state machine uses the following constants, all of which are configurable items:

Constant	Value	Description
timeout	250 milliseconds	Packet acknowledgement timeout.
winsize	4	Transmit window size (in packets). (Minimum value 1. Maximum value is the range of the Ack Field, minus one.)
retry_limit	20	Number of times a message is resent before declaring the link has failed.

Table 9.2: Transmit State Machine Constants

Figure 9.3 depicts the state machine to show the mechanism used to transmit packets.

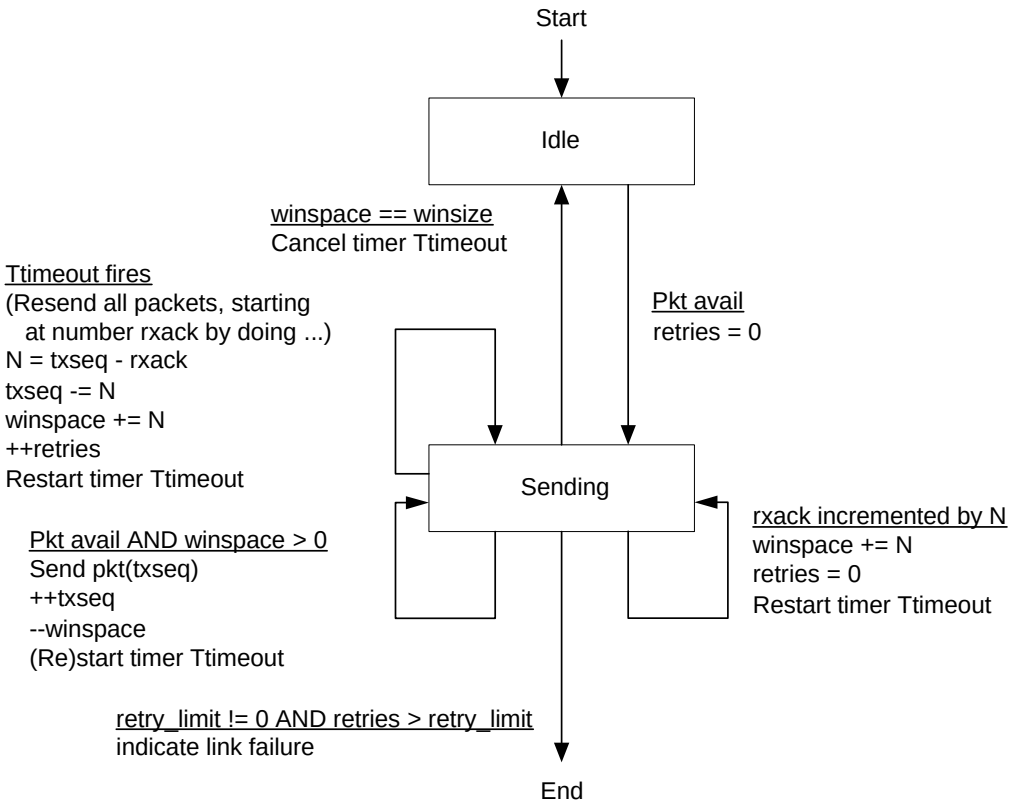


Figure 9.3: Transmit State Machine

Assuming a glut of packets to be sent to the peer, these will be sent until the transmit window fills (winspace == 0). Then one extra packet will be sent for each packet acknowledged. Under normal circumstances, the state machine thus keeps up to winsize unacknowledged packets in transit to the peer.

If the peer acknowledges no packets for a while the timeout will provoke a retransmission of all unacknowledged packets, starting with the oldest, i.e., it uses a conventional “go back n” mechanism to recover from the peer not acknowledging receipt of a packet.

The link is considered to have failed if more than retry_limit attempts are made to obtain acknowledgement of delivery of any packet. This mechanism is only applied if the value of retry_limit is non-zero. This gives a means of preventing the software from marking the link as failed.

Only one timer is used for detecting that the peer has not acknowledged receipt of a packet.

9.5 Receive State Machine

The Sequencing Layer runs a state machine for receiving packets. This uses the following variables:

Variable	Initial Value	Description
expected_rxseq	0	The Seq Field value of the next packet that will be accepted.
txack	0	Acknowledgement number of the next transmitted packet.

Table 9.3: Receive State Machine Variables

The range of the variables expected_rxseq and txack is determined by the widths of fields Seq and Ack in the BCSP packet format: both fields are 3 bits wide, so the variables can take values between 0 and 7. Variables in

these fields wrap on overflow and underflow. (The values of these two variables are always the same, so an implementation need use only one variable.)

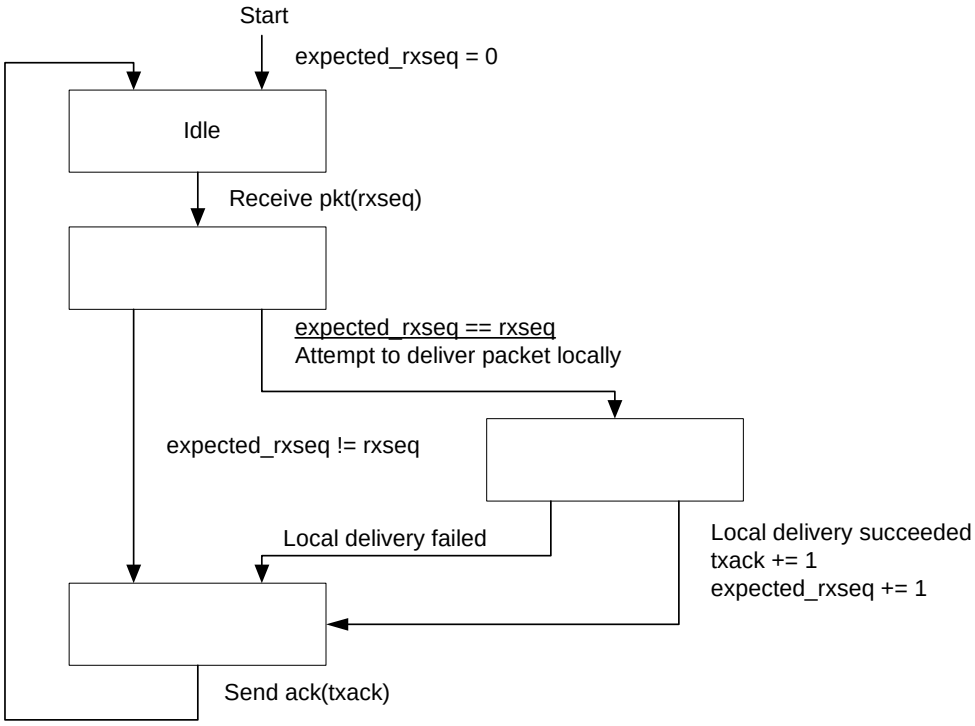


Figure 9.4: Receive State Machine

At any instant the receive state machine will only accept a packet with one particular sequence number (i.e., the state machine runs a sliding *receive* window of size one).

The receive state machine spends most of its time in the state “idle”, where it waits for a packet to be received. When a packet arrives, its sequence number (the value from the packet’s Seq Field) is compared with the expected_rxseq value.

If this matches, the state machine attempts to deliver the packet to higher layer code. This may accept or refuse the packet; refusal is the basis of the Sequence Layer’s internal flow control of the inbound packet stream.

If the packet is accepted then the state machine is set to seek the next sequence number on the next packet received.

After any packet is received, the state machine sends back an acknowledgement to the peer. This will be the value of the Seq Field plus one to indicate acceptance, or the value of the packet’s Seq Field to indicate that the packet was not accepted.

The sending of the acknowledgement forces the value of txack to be sent to the peer, even if the local machine has already sent an acknowledgement of that value.

9.6 Reset

When the Sequencing Layer is reset it clears all state associated with sending and receiving packets, and sets variables to their initial values.

9.7 Protocol Identifier Field

Each data packet sent into the Sequencing Layer from higher layered code is associated with a protocol identifier.

The Sequencing Layer writes this value into the Protocol Identifier Field of all data packets its sends down to the MUX Layer.

Similarly, the Sequencing Layer extracts the value from the Protocol Identifier Field of all packets it transfers from the MUX Layer to the higher level code and delivers the value with the packets.

The Field's value must not be zero; this value is used by the MUX Layer.

10 Datagram Queue Layer

The Datagram Queue Layer provides a single unreliable bidirectional packet stream to higher level code. The stream provides no flow control.

The Layer provides (queue) storage for a number of messages waiting to be sent to the peer.

10.1 Datagram Queue Layer Context

Figure 10.1 shows the context of the Datagram Queue Layer:

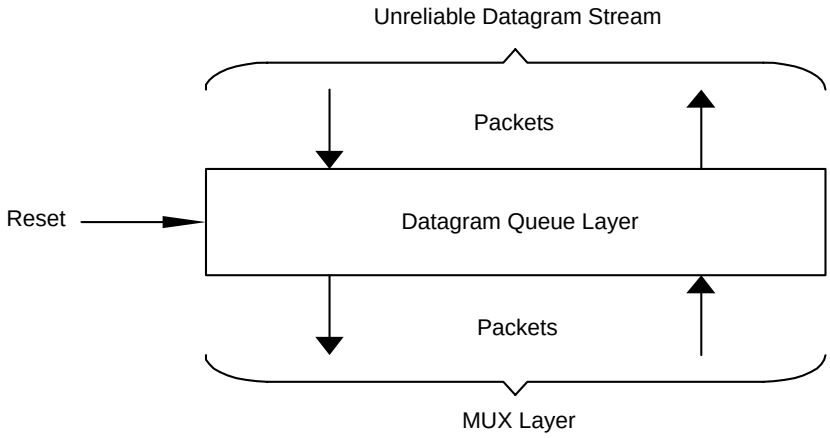


Figure 10.1: Datagram Queue Layer

Data packets are passed into the Datagram Queue Layer from higher layered code. These are sent on to the MUX Layer, which sends them towards the peer.

Data packets received from the peer are passed up from the MUX Layer; the Datagram Queue Layer passes these on up to the higher layers of code.

The Reset signal initialises the Layer.

10.2 Datagram Queue Layer Functions

10.2.1 Receiving Messages

The Layer delivers packets received from the MUX Layer to higher layered code.

10.2.2 Sending Messages

The Layer accepts packets to be sent to the peer from higher layered code.

The Layer maintains a queue of such messages, which it attempts to deliver in sequence order to the MUX Layer, for transfer to the peer. If the queue is full when a message is passed into the Layer, the oldest message(s) in the queue may be silently discarded to make room for it.

10.2.3 Seq Field

The Datagram Queue Layer sets the Seq Field to value zero in each packet passed to the MUX Layer.

10.2.4 Protocol Identifier Field

Each data packet sent into the Datagram Queue from higher layered code is associated with a protocol identifier. The Datagram Queue Layer writes this value into the Protocol Identifier Field of all data packets its sends down to the MUX Layer.

Similarly, the Datagram Queue Layer extracts the value from the Protocol Identifier Field from all packets it receives from the MUX Layer, and delivers this with data packets it sends to the higher code layers.

Higher layer code may use protocol identifier values in the range `1` to `15`; the value zero must not be used.

10.2.5 Reset

When the Datagram Queue Layer is reset it discards any messages waiting to be sent to the MUX Layer.

10.3 Top Level View

This section gives a notional view of how code above the Datagram Queue Layer might view its services. These function descriptions are for illustration only.

10.3.1 tx_unreliable_pkt()

To send a data packet, higher layer code might call a function of the form:

```
void tx_unreliable_pkt(char *buf, unsigned n, unsigned protocol_id);
```

This attempts to send the `n` byte message in the buffer `buf` to the peer, tagged with protocol identifier `protocol_id`.

The range of `n` is zero to `0xfff`, though the system's configuration is likely to limit the value more severely.

The value of `protocol_id` can range from `1` to `15`. The value zero must not be used.

This view presumes the Datagram Queue Layer locally stores a copy of the message for transmission. It is recognised that implementations are unlikely to be structured in this way.

10.3.2 rx_unreliable_pkt()

For the Datagram Queue Layer to deliver a packet to higher level code, the Layer might call a function of the form:

```
void rx_unreliable_pkt(char *buf, unsigned n, unsigned protocol_id);
```

This delivers to the higher level code the `n` byte message at `buf`, tagged with protocol identifier `protocol_id`. The message and identifier will match parameters passed into `tx_unreliable_pkt()` on the peer.

The higher level code may accept or discard the message; the Datagram Queue Layer is not concerned with this action.

11 Configurable Items

Some elements of the BCSP are programmable. This section gathers these items together.

It is expected that the BCSP on both ends of a connection will be statically configured to have the same settings.

Variable	Default value	Layer	Description
baud_rate	38.4kbaud	UART Driver	Baud rate of the UART link.
parity	enabled	UART Driver	Parity enabled on UART.
parity_type	even	UART Driver	Even or odd parity used on UART.
n_stop	1	UART Driver	Number of stop bits used on UART.
crc_present	FALSE	Packet Integrity	Is a CRC Field added to packets?
timeout	250ms	Sequencing	Delay before resending an unacknowledged packet.
winsize	4	Sequencing	Size of the transmit window (counted in number of packets).
retry_limit	20	Sequencing	Number of times a message is resent before declaring the link has failed.
hardware_flow_control	disabled	UART Driver	Hardware flow control on UART link.

Table 11.1: Configurable Items

12 Comments

The following comments are on the BCSP design presented in this document.

BCSP has been written to work with a highly reliable UART link. The CRC Field will not normally be present.

The stack does not constrain the size of packets transmitted, other than by the range of the Payload Length Field. It is probable that higher layers of code will constrain packets' sizes further, possibly with different limits for each Protocol Identifier Field value, and possibly with different limits in the two directions of the link.

The Sequencing Layer's flow control mechanism is intended to be used as a last line of defence of the receiving machines' resources. It is expected that higher level protocols will apply their own flow control mechanisms, possibly with different window sizes, and possibly with adaptive window sizing.

The use of fixed values for "timeout" in the Sequencing Layer may give uneven timing recovery behaviour under differing traffic loads. It may be appropriate to make the timing values adapt to link speed and maximum packet sizes, but there is unlikely to be a simple solution.

By convention each BCSP packet carries only a single higher-layer packet.

BCSP is defined to use a 3-wire UART. However, the design was originally intended for a system with a low baud rate: 38.4 kbaud. It is normal to use a 5-wire UART connection (with two hardware flow control signals), particularly at higher baud rates.

Document References

Document ID	Document Title	CSR Reference
[BT1.2]	Specification of the Bluetooth System, Version 1.2, Core Package, 5 November 2003	n/a

Further References

Document Title	CSR Reference
BCSP Channel Allocation	bcore-sp-007P
BCSP Link Establishment Protocol	bcore-sp-008P

Acronyms and Definitions

BCSP	BlueCore Serial Link Protocol, described in this document
BlueCore™	Group term for CSR's range of Bluetooth chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CRC	Cyclic Redundancy Check
CSR	Cambridge Silicon Radio
HCI	Host Controller Interface; part of Bluetooth
MUX	Multiplexor
SLIP	Serial Link Internet Protocol
UART	Universal Asynchronous Receiver Transmitter

Record of Changes

Date	Revision	Comment
8 Jan 03	a	Document originally published as CSR reference bc01-an-004 (revisions a through b; versions through HCIStack1.1v15.x builds). New revision control number allocated to align with HCIStack1.1v16.x builds.
15 Jul 04	b	Updated formatting, corrected typos.

BlueCore™ Serial Protocol (BCSP)

bcore-sp-012Pb

July 2004