Trainset messages and communications

Larry Hughes, PhD

1 August 2017
Minor revisions: 31 October 2018; 5 November 2019

1 Introduction

The trainset and the trainset controller are two different systems that must exchange data to ensure the reliable functioning of the operating trains. The trainset is responsible for detecting Hall sensor events (cause by a locomotive passing over a Hall sensor), while the trainset controller employs a set of rules to control the locomotive state (its location, speed, and direction) and the switch state (thrown either straight or diverged).

Events detected by the trainset and state-changing commands issued by the trainset controller must be conveyed quickly and reliably to the train controller and trainset, respectively. This document describes the protocol used to convey messages (in the form of events or commands) in a layered communication architecture between the trainset and trainset controller.

The architecture is divided and discussed in terms of three distinct layers (see Figure 1):

Application Layer. Entities running application software on the trainset and trainset controller encode events and commands into *messages*. These messages are associated with a set of rules or protocol specific to the application (for example, Hall-sensor events or switch-throwing). The location of the remote entity is unknown in that messages are exchanged between entities on the remote system in much the same way they would be on the local system.

Data Link Layer: Messages sent between entities on different systems are encapsulated and sent reliably in *packets* using the underlying Data Link Layer. The contents of the message are ignored by the Data Link Layer. While there can be many application entities on a system, there is only a single Data Link Layer on each system responsible for exchanging packets.

Physical Layer: The transfer of packets between the two systems is the responsibility of the Physical Layer, which sends and receives packets encapsulated in *frames*. The Physical Layer is not reliable.

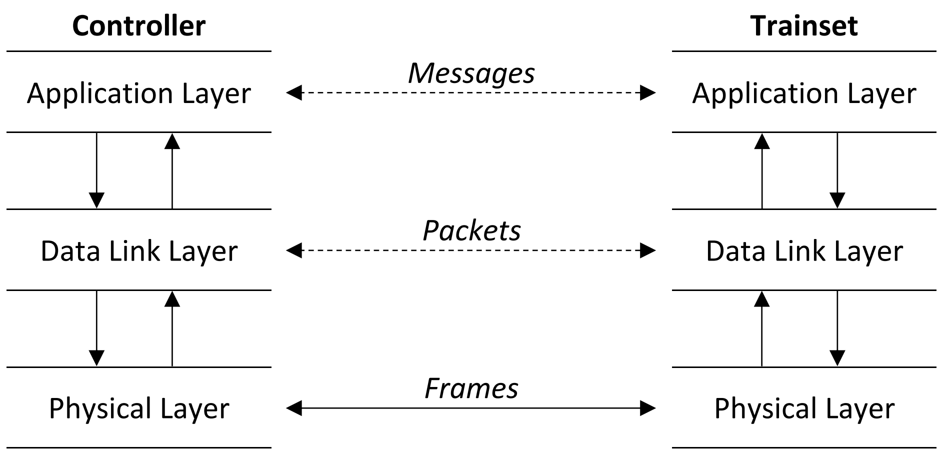


Figure 1: Trainset communications stack

2 Application Layer (message structure)

As the state of the system changes, the controller and trainset applications exchange data in *messages*. The data in a message can indicate:

- Changes to the trainset state detected by the trainset system;
- Changes to a locomotive or switch state requested by the controller, or
- An acknowledgement to a previous message from the trainset or the controller.

A message consists of a Code (indicating a detected or requested state change or an acknowledgement) and zero, one, or two Code-specific Arguments (see Figure 2).

Code	Argument 1	Argument 2

Figure 2: Message structure

The message structure can be represented in C as follows:

```
struct message
{
unsigned char code;    /* Message code (described below) */
unsigned char arg1;    /* First argument (optional) */
unsigned char arg2;    /* Second argument (optional) */
};
```

The different message codes and their arguments are described below.

2.1 Hall sensor messages

A locomotive's location on the layout is detected by one of the two magnets on the locomotive triggering one of the 24 Hall sensors. The sensor number is queued for the trainset to send it to the controller in a message. The controller must acknowledge its receipt of the Hall sensor message to allow the trainset to reset the Hall sensor.

There are four Hall-sensor messages (Table 1 specifies the message's codes and arguments):

Hall-sensor triggered message: When a Hall sensor is triggered, the trainset is responsible for sending the number of the sensor as Argument 1 to the controller.

Hall-sensor triggered acknowledgement message: The controller responds to Hall-sensor triggered messages with an acknowledgment message carrying the number of the sensor being acknowledged. All sensors can be acknowledged by specifying the sensor number as 0xFF.

Hall-sensor queue reset message: A queue holding the number of Hall-sensors numbers that have yet to be sent to the controller is maintained by the trainset. To ensure that the queue is properly initialized, the controller should send a queue reset to the trainset to initialize the Hall Sensor queue at start-up. There are no arguments for this message.

Hall-sensor queue reset acknowledgement message: The trainset responds to the Hall-sensor queue reset message with an acknowledgement message. There are no arguments for this message.

Table 1: Hall-sensor messages

Message			Source	Description
Code	**Argument 1**	**Argument 2**		
0xA0	Sensor number		Trainset	Hall sensor triggered message, specifying the number of sensor triggered (1 through 24)
0xA2	Sensor number		Controller	Hall sensor triggered acknowledge message (sensor number or 0xFF - all)
0xA8			Controller	Hall sensor queue reset request message
0xAA			Trainset	Hall sensor queue reset acknowledge message

2.2 Locomotive magnitude and direction messages

The magnitude (speed) and direction of each locomotive is specified by the controller.[1] There are two messages (shown in Table 2):

Magnitude and direction message: The controller can change the magnitude and direction of any locomotive in a direction and magnitude message (see Table 2). The message has two arguments, the first being the locomotive number to change and the second, a byte specifying the magnitude and direction of the locomotive:

```
struct mag_dir
{
unsigned magnitude : 4;    /* 0 – stop through 15 – maximum */
unsigned ignored : 3;      /* Zero */
unsigned direction : 1;    /* 1 for CCW and 0 for CW */
};
```

[1] Each locomotive is assigned a unique number, starting at 0x01. Each locomotive is labelled with its number.

The magnitude and direction of all locomotives can be set if the locomotive number is 0xFF. This allows the controller to issue a message to, for example, stop all locomotives; this can be used as a “panic” signal to stop all locomotives by sending a magnitude of 0x00.

Table 2: Locomotive magnitude and direction messages

Message			Source	Description
Code	**Argument 1**	**Argument 2**		
0xC0	Locomotive number	Speed/Direction	Controller	Magnitude and direction request message (locomotive number or 0xFF - all)
0xC2	Locomotive number	Result	Trainset	Magnitude and direction acknowledgement message, either success (0) or failure (>0). **This message is only sent to indicate failure.**

Magnitude and direction acknowledgement message: The trainset is to respond to the magnitude and direction with an acknowledgement message with the locomotive number as the first argument. The second argument indicates the success or failure of the magnitude and direction message (see Table 3).

Table 3: Magnitude and direction acknowledgement results

Value	Meaning
0	Success
1	Invalid locomotive number
2 to 255	Not implemented

2.3 Switch-throw messages

Switch-throw messages are sent from the controller to the trainset when a switch is to be thrown to straight or diverged. After a switch is thrown, the trainset is to respond with a switch-thrown acknowledgement message.

The switch-throw messages are defined as follows (messages codes and arguments are listed in Table 4):

Switch-throw message: The controller can throw any switch to straight or diverged using a switch-thrown message, specifying the switch number in the first argument and the direction of the throw in the second (1 – straight and 0 – diverged). A switch number of 0xFF throws all switches to the specified direction.

Switch-throw acknowledgement message: After the trainset has thrown the switch to the specified direction, it is to acknowledge the switch-thrown message, indicating the success of the action (see Table 5).

Table 4: Switch-throw messages

Message			Source	Description
Code	**Argument 1**	**Argument 2**		
0xE0	Switch number	Direction	Controller	Switch-throw message, specifying the switch (switch number or 0xFF – all) and specified direction (1 – straight; 0 – diverged)
0xE2	Switch number	Result	Trainset	Switch-throw acknowledgement message, either success (0) or failure (>0). **This message is not sent.**

Table 5: Switch-throw acknowledgement results

Value	Meaning
0	Success
1	Invalid switch number
2	Invalid direction
3 to 255	Not implemented

3 Communications

Messages are sent and received by applications on the trainset and controller. The actual process of sending and receiving is handled in two underlying layers:

- The Data Link Layer is responsible for transferring messages reliably in *packets*.
- The Physical Layer is responsible for the actual transfer of packets between the machines in *frames*.

3.1 Data Link Layer (packet structure)

The Data Link Layer transfers messages reliably between the two machines in packets. Reliability is attained using a sliding window protocol.

3.1.1 The sliding window protocol

There are three types of packet:

Data (DATA): A data packet carries an application supplied message. Each packet transmitted is assigned a unique sequence number between 0 and 7, referred to as Ns. The sequence number increases, modulo 8. Packets are not discarded until they are acknowledged (see below). The sequence number of the next data packet to be received is referred to as Nr.

Both sequence numbers (Ns and Nr) and the packet type (DATA) make up the packet's Control field. Data packets should only be discarded after they are acknowledged. If a data packet is sent and no acknowledgement is received, the transmitting Data Link layer should retransmit the message after a one-second delay.

The protocol is referred to as a sliding-window protocol for two reasons. First, the window refers to the number of data packets that can be transmitted before an acknowledgement is expected. A window size of two is quite common. Second, the window is "sliding" in that as acknowledgements are received, the window "slides" or moves to the next sequence number.

Each data packet sent must be responded to within a specified time limit, typically one or two seconds. If a response (acknowledgement or negative acknowledgement) is not received, the pending packets are transmitted again.

Acknowledgement (ACK**)**: A receiving Data Link Layer can respond to a data packet with an acknowledgement packet. The acknowledgement packet carries the sequence number of the next expected data packet, Nr. Acknowledgement packets are not associated with timers and do not change the sequence number Ns. Nr is incremented modulo 8 whenever a data packet is received with the next expected sequence number.

For example, if data packets 5 and 6 have been transmitted and received corrected, the receiver responds with an acknowledgement packet and Nr equal to 7, indicating the next expected sequence number.

Negative acknowledgement (NACK**)**: If a data packet is received out-of-sequence (for example, packet 2 is expected, but packet 3 is received instead), the receiving Data Link Layer responds with a negative acknowledgement and the sequence number of the expected packet, Nr (2, in this example). Negative acknowledgement packets are not associated with timers and do not change the sequence number.

The structure of a packet is shown in Figure 3.

Control	Length	Message

Figure 3: Packet structure

The fields in a packet are defined as follows:

Control: The Control field specifies the packet type, its sequence number, Ns (DATA packets only), and the next expected sequence number from the remote Data Link Layer, Nr (all three packet types).

```
enum PktType {DATA, ACK, NACK};

struct control
{
unsigned nr : 3;
unsigned ns : 3;
enum PktType type : 2;
};
```

The initial values of Nr and Ns are zero. The Ns field is only defined for Data packets, all three packet types carry the current value of Nr.

If a data packet is received correctly (i.e., its Ns equals the expected Ns), the value of the expected Ns is incremented modulo 8.

Length: If the packet type is DATA, the length field contains the number of bytes in the message (1 to 255). The Length field is omitted if the packet type is ACK or NACK.

Message: The message to be sent to the remote entity. The Message field is omitted if the packet type is ACK or NACK.

Table 6: Summary of packet types

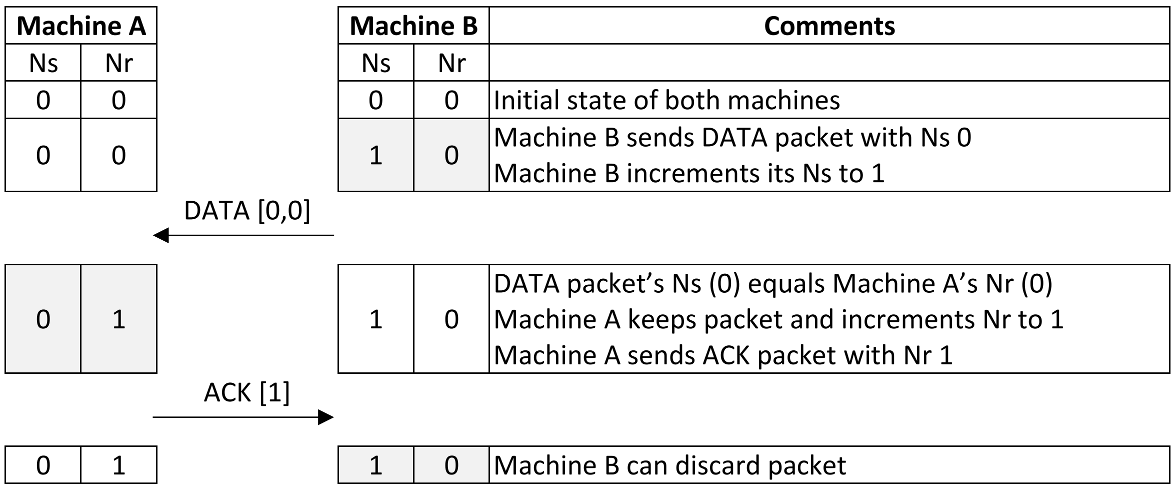
Packet type	Control			Length	Message	Description
	Type	Ns	Nr			
DATA	00					Packet carries all Control fields, the Length and the Message
ACK	01	▨		▨	▨	Packet only carries Type and Nr field
NACK	10	▨		▨	▨	Packet only carries Type and Nr field

3.1.2 Examples

DATA-ACK sequence between two machines

Each data packet is sent with its own sequence number (Ns) and the number of the next expected packet from the remote (Nr). Data packets are acknowledged with acknowledgement packets if a data packet is not available to carry the next expected packet number (Nr).

In this example, Machine B sends a data packet with sequence number Ns equal to 0. Machine A responds with an acknowledgement packet with Nr equal to 1, the next expected sequence number. See Figure 4.



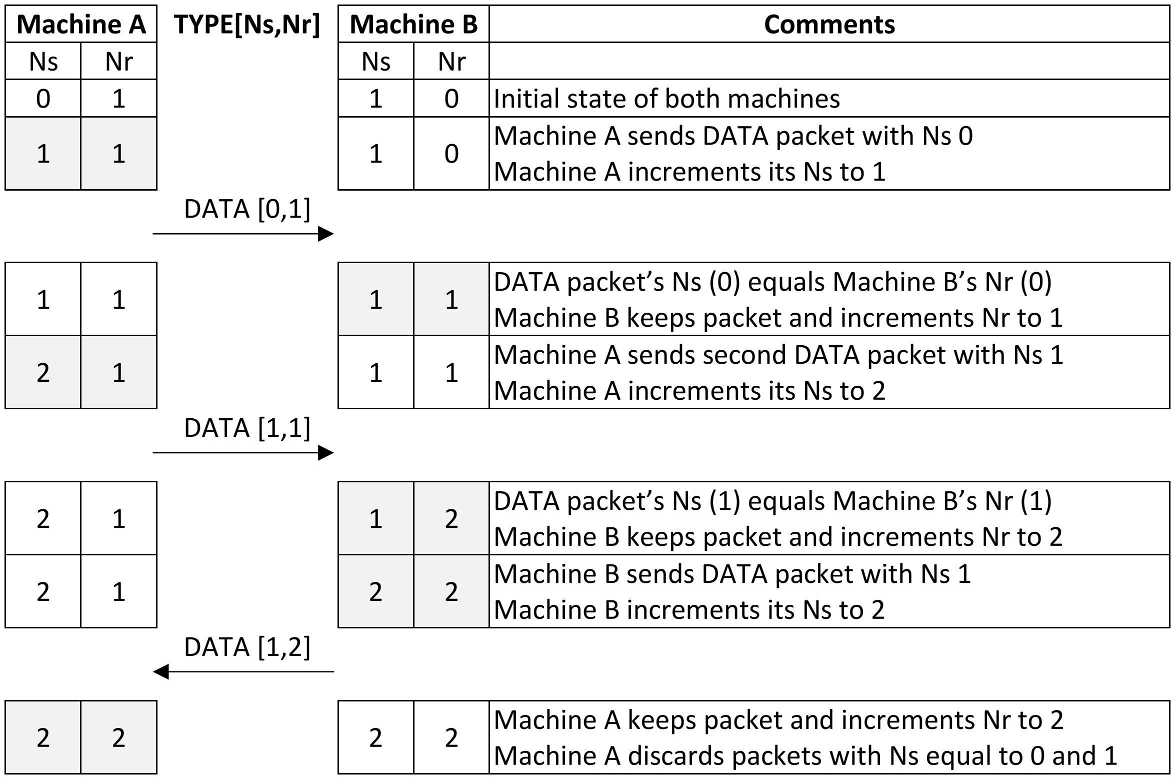
Machine A			Machine B		Comments
Ns	Nr		Ns	Nr	
0	0		0	0	Initial state of both machines
0	0		1	0	Machine B sends DATA packet with Ns 0 Machine B increments its Ns to 1
		← DATA [0,0]			
0	1		1	0	DATA packet's Ns (0) equals Machine A's Nr (0) Machine A keeps packet and increments Nr to 1 Machine A sends ACK packet with Nr 1
		ACK [1] →			
0	1		1	0	Machine B can discard packet

Figure 4: DATA-ACK sequence

Multiple packets and piggybacking acknowledgements

Depending on the window size, multiple packets can be sent before the transmitter waits for an acknowledgement. An acknowledgement (in the form of the next expected sequence number, Nr) can be piggybacked on a data packet.

In this example, Machine A sends two packets (with sequence numbers, Ns, 0 and 1) to Machine B; the data packet includes the sequence number of the next expected packet (Nr, 1). Machine B receives both packets and responds with a data packet with sequence number, Ns, of 1 and piggybacks its acknowledgement to Machine A's two data packet with Nr of 2. See Figure 5.



Machine A		TYPE[Ns,Nr]	Machine B		Comments
Ns	Nr		Ns	Nr	
0	1		1	0	Initial state of both machines
1	1		1	0	Machine A sends DATA packet with Ns 0 Machine A increments its Ns to 1
		DATA [0,1] →			
1	1		1	1	DATA packet's Ns (0) equals Machine B's Nr (0) Machine B keeps packet and increments Nr to 1
2	1		1	1	Machine A sends second DATA packet with Ns 1 Machine A increments its Ns to 2
		DATA [1,1] →			
2	1		1	2	DATA packet's Ns (1) equals Machine B's Nr (1) Machine B keeps packet and increments Nr to 2
2	1		2	2	Machine B sends DATA packet with Ns 1 Machine B increments its Ns to 2
		← DATA [1,2]			
2	2		2	2	Machine A keeps packet and increments Nr to 2 Machine A discards packets with Ns equal to 0 and 1

Figure 5: Multiple packets and piggybacked acknowledgements

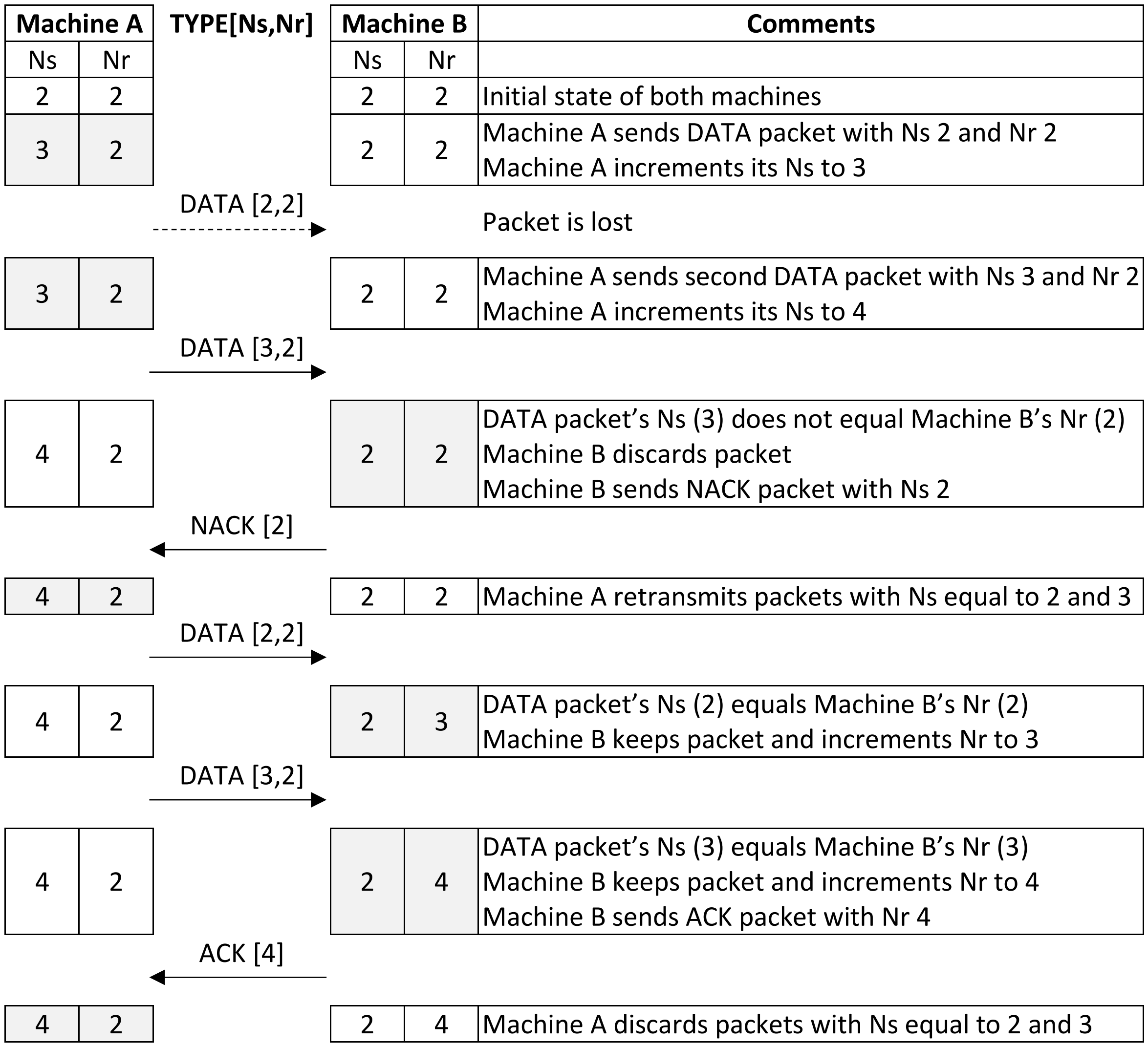
Negative Acknowledgements

If a data packet is received with an out-of-sequence sequence number, it means that at least one packet has been lost. The receiving Data Link Layer responds with a negative acknowledgement packet with the sequence number of the next expected packet (Nr).

The Data Link Layer protocol described here does not have the capability to do selective retransmission of packets, meaning that all packets must be retransmitted, including any received correctly.

In this example, Machine A sends two data packets with sequence numbers, Ns, 2 and 3. Machine B only receives the data packet with sequence number 3 because the data packet with sequence number 2 is lost. The out-of-sequence packet means that Machine B must respond with a negative acknowledgement packet, specifying the expected packet number, Nr; in this case, 2.

Machine A responds to the negative acknowledgement packet by retransmitting the two data packets; the same sequence numbers are used. When the acknowledgment is received from Machine B, Machine A discards the two packets. The sequence numbers can be reused again. See Figure 6.



Machine A Ns	Machine A Nr	TYPE[Ns,Nr]	Machine B Ns	Machine B Nr	Comments
2	2		2	2	Initial state of both machines
3	2		2	2	Machine A sends DATA packet with Ns 2 and Nr 2 Machine A increments its Ns to 3
		DATA [2,2] →			Packet is lost
3	2		2	2	Machine A sends second DATA packet with Ns 3 and Nr 2 Machine A increments its Ns to 4
		DATA [3,2] →			
4	2		2	2	DATA packet's Ns (3) does not equal Machine B's Nr (2) Machine B discards packet Machine B sends NACK packet with Ns 2
		← NACK [2]			
4	2		2	2	Machine A retransmits packets with Ns equal to 2 and 3
		DATA [2,2] →			
4	2		2	3	DATA packet's Ns (2) equals Machine B's Nr (2) Machine B keeps packet and increments Nr to 3
		DATA [3,2] →			
4	2		2	4	DATA packet's Ns (3) equals Machine B's Nr (3) Machine B keeps packet and increments Nr to 4 Machine B sends ACK packet with Nr 4
		← ACK [4]			
4	2		2	4	Machine A discards packets with Ns equal to 2 and 3

Figure 6: Negative acknowledgements

Lost packets and timeouts

If a packet is sent but not received, the transmitter must resend the packet; this requires that each packet sent is associated with a timer that can signal that a packet has not been received within a specified time limit.

In this example, Machine A sends two packets to Machine B (sequence numbers, Ns, of 4 and 5). Since no acknowledgements are forthcoming, the timer for the packet with sequence number 4 expires and Machine A retransmits the two packets. See Figure 7.

Machine A		TYPE[Ns,Nr]	Machine B		Comments
Ns	Nr		Ns	Nr	
4	2		2	4	Initial state of both machines
5	2		2	4	Machine A sends DATA packet with Ns 4 and Nr 2 Machine A increments its Ns to 5
		DATA [4,2] ⇢			Packet is lost
6	2		2	4	Machine A sends second DATA packet with Ns 5 and Nr 2 Machine A increments its Ns to 6
		DATA [5,2] ⇢			Packet is lost
6	2		2	4	Machine A does not get a response to either packet After a period of time, Machine A times-out and retransmits the packets with Ns equal to 4 and 5
		DATA [4,2] →			
6	2		2	5	DATA packet's Ns (4) equals Machine B's Nr (4) Machine B keeps packet and increments Nr to 5
		DATA [5,2] →			
6	2		2	6	DATA packet's Ns (5) equals Machine B's Nr (5) Machine B keeps packet and increments Nr to 6 Machine B sends ACK packet with Nr 6
		← ACK [6]			
6	2		2	4	Machine A discards packets with Ns equal to 4 and 5

Figure 7: Lost packets and timeouts

3.2 Physical layer (frame structure)

Packets are sent between the two machines using the services of the Physical Layer in *frames*. Although the Physical Layer has a simple checksum, it is a best-effort service in that the frame could be lost or corrupted. It is the responsibility of the Data Link Layer to detect these errors. In addition to defining the frame structure, the Physical Layer also specifies the physical interconnection characteristics.

3.2.1 Frame structure

Packets are enclosed in frames, defined as follows:

STX	`Packet`	`Checksum`	**ETX**

Figure 8: Frame structure

where:

STX: ASCII start-of-transmission control byte (0x02)

`Packet`: The packet bytes supplied by the Data Link Layer.

`Checksum`: The ones-complement of the sum of the packet bytes.

ETX: ASCII end-of-transmission control byte (0x03)

If the value of a packet byte or the checksum is the same value as either the start-of-transmission byte (**STX**) or end-of-transmission byte (**ETX**), it must be prefixed with the ASCII Data Link Escape control byte (**DLE** or 0x10). Escape bytes are not to be included in the checksum. Note that if a packet byte or the checksum has a value of **DLE**, the byte must also be prefixed with **DLE** control byte.

3.2.2 Transmitting frames

The Data Link Layer supplies the Physical Layer with a packet and a count of the number of bytes to be transmitted. The Physical Layer is responsible for transmitting the packet bytes, checksum, and any escape bytes in a frame. The transmit state-diagram is shown in Figure 9.

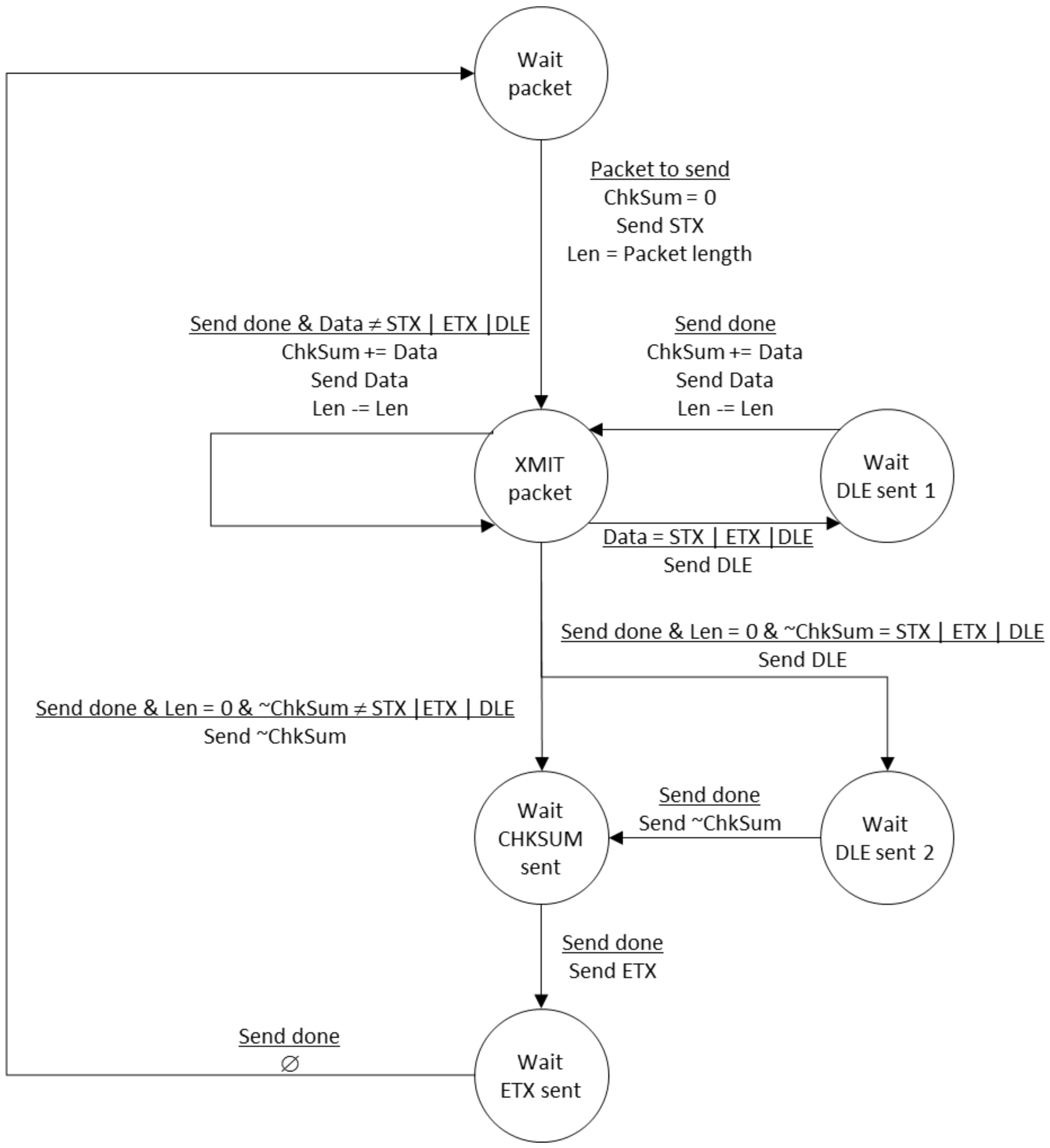


Figure 9: Transmit state-diagram
(Note: Data refers to the current byte to be sent from Packet at index Len)

3.2.3 Receiving frames

The Physical Layer receives frames from the remote machine (either the controller or the trainset). A valid frame must start with an **STX**, end with an **ETX**, have the correct checksum, and, where necessary, use the **DLE** control byte to prefix non-control **STX**, **ETX**, or **DLE** bytes. When a valid frame is received, its contents (i.e., the packet) is passed to the Data Link layer. The receive state-diagram is shown in Figure 10.

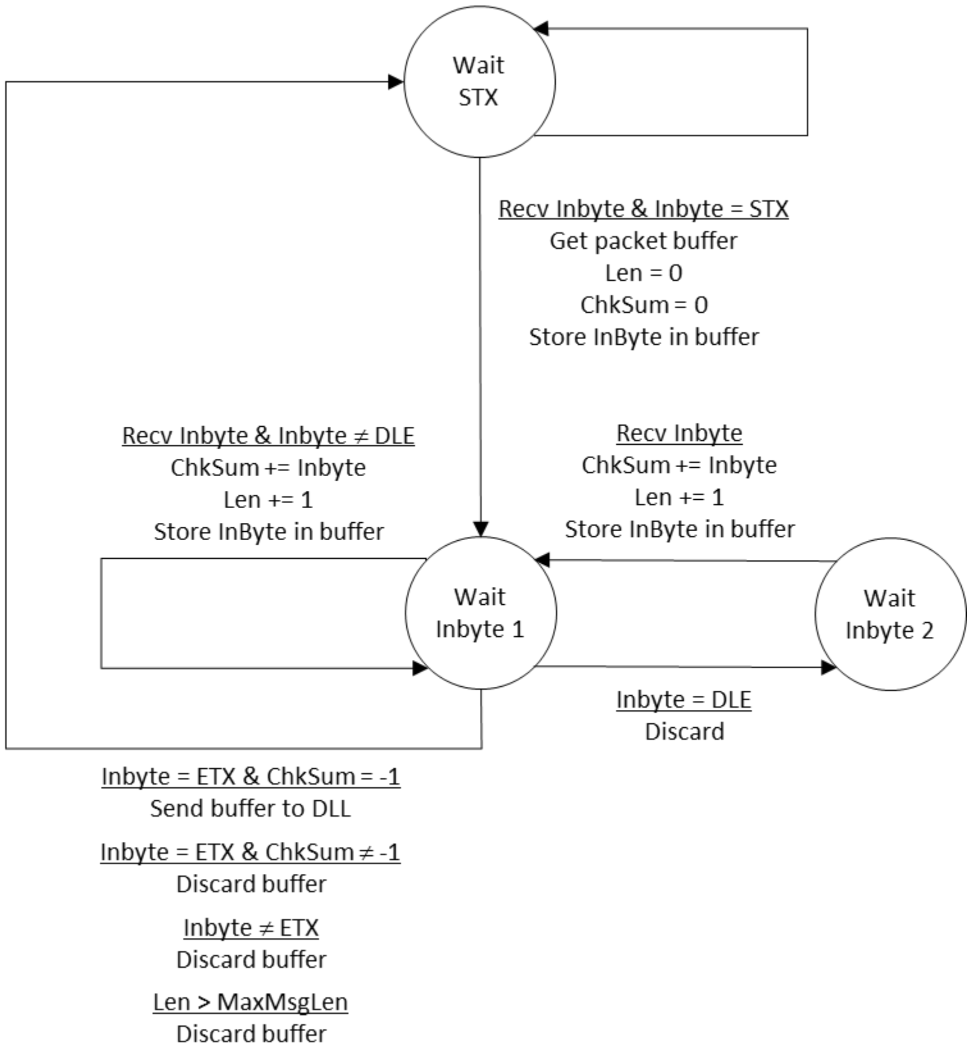


Figure 10: Receive state-diagram
(Note: Inbyte is the most recently received byte from the UART)

There is no timeout associated with the receive state-machine as it is assumed that the transmitting Data Link Layer will timeout and retransmit any unacknowledged packets. This could lead to problems if a frame is partially received and the next frame's **STX** is received as it means that the receive state-machine returns to the Wait-STX state rather than entering the state Wait-Inbyte-1.

3.2.4 UART configuration

The controller and trainset hardware are physically connected using UARTs. All communications between the two machines takes place across this channel in RS-232 frames. The frame-size is 10-bits, configured using the machine's UART control register(s):

- 8-bit data – can carry a byte of the Physical Layer frame (section 3.2.1).
- No parity – parity is not needed as the Physical Layer frame (section 3.2.1) determined the checksum.
- One stop-bit (the start-bit is default)

Communications are bi-directional, allowing simultaneous transmission and reception by both entities.

The line speed is 115,200 bits-per-second.

4 Implementation notes and suggestions

To keep the amount of byte-copying to a minimum, the Data Link Layer should supply the Physical Layer with three separate arguments (Control, Length, and the address of the Message).

Ideally, it should be possible to send multiple messages per packet; however, this is not supported at present.

5 Summary of Trainset Messages (Corrected 29 July 17)

The following table is an updated and rationalized version of the existing trainset messages intended for use with the bi-directional exchange of packets between the controller (Tiva) and trainset (Atmel).

There are now three message grouping: Hall sensor, locomotive magnitude and direction, and switch state. Each grouping includes an explicit acknowledgement message.

Message			Source	Description
Code	Argument 1	Argument 2		
0xA0	Sensor number		Trainset	Hall sensor triggered message, specifying the number of sensor triggered (1 through 24)
0xA2	Sensor number		Controller	Hall sensor triggered acknowledge message (sensor number or 0xFF - all)
0xA8			Controller	Hall sensor queue reset request message
0xAA			Trainset	Hall sensor queue reset acknowledge message
0xC0	Locomotive number	Speed/Direction	Controller	Change of locomotive speed and direction request message (locomotive number or 0xFF - all)
0xC2	Locomotive number	Result	Trainset	Change of locomotive speed and direction acknowledgement message, either success (0) or failure (>0). **This message is only sent when a failure is detected.**
0xE0	Switch number	Direction	Controller	Throw-switch request message, specifying the switch (switch number or 0xFF - all) and specified direction (1 - straight; 0 - diverged)
0xE2	Switch number	Result	Trainset	Throw-switch acknowledgement message, either success (0) or failure (>0). **This message is never sent.**