wireless control for everyone

BLOCK AND STREAM MESSAGES IN ONE-NET

Version 2.3.0

August 14, 2012

Table of Contents

1 Overview

Block and Stream messaging in ONE-NET can be used when the amount of data to be transferred cannot be transferred in a single packet. Networks will differ widely, but it is generally assumed that not all devices within the network will be able to or need to participate in Block and Stream communications. It is also generally assumed that many long Block and Stream transfers will have to be interrupted at some point to allow for Single transactions. Networks where this is not a problem should consider altering the core ONE-NET code to their needs.

2 Intended Audience

This document is geared towards both programmers and non-programmers. Certain sections will only be of interest to programmers. Network designers should also read this document to know what the issues are and what decisions need to be made.

3 Nomenclature

The following sections define terms used in this document. Note that many of these terms were defined in the **ONE-NET Specification** document.

3.1 Definitions

Block Transaction: A transfer of a fixed number of data bytes that do not fit into a single data packet, and are delivered reliably.

Single Transaction: A transfer of a small, fixed number of data bytes that fit into a single data packet, and are delivered reliably.

Stream Transaction: A transfer of an undetermined number of data bytes, which are not delivered reliably.

Client Repeater: A Client that listens for multi-hop packets and retransmits them. Such Devices are generally powered on at all times.

Repeater: Same as Client Repeater.

Fragment: The data payload in a single Block or Stream data packet.

Segment: Same as Fragment.

Chunk: One or more Segments in a Block Transaction. Block Transactions are sent one Chunk at a time before the sender waits for an acknowledgement from the recipient.

Chunk Size: The number of Segments in a Chunk of a Block Transaction.

Chunk Index: The packet number WITHIN A CHUNK that is being transferred. This will range from 0 to the Chunk Size minus 1.

Byte Index: The offset from the first byte in a block transfer

Fragment Delay: The time that the sender of a Block or Stream Transaction should wait between sending consecutive Segments of a Block or Stream Transaction.

Segment Delay: Same as Fragment Delay.

Chunk Pause: The time that the sender of a Block Transaction should wait after receiving a response from the recipient after sending a Chunk of data. The Chunk Pause is intended to prevent the Block Transaction from monopolizing all bandwidth on a particular channel.

Features: The capabilities of a device. Examples include Boolean bits representing whether a device can serve as a multi-hop repeater, whether the device sleeps, whether the device maintains a peer table, and whether the device can handle Block and Stream Transactions.

Capabilities: Same as Features.

Response Timeout: The maximum amount of time that a Device waits for a response for its message.

Timeout Time: The time that a Device will wait for a pertinent message in a Block or Stream transfer before assuming that communication has been lost and declaring that the Transfer has failed.

Retransmission Delay: The amount of time (2ms for high-priority Transactions, and 10ms for low-priority Transactions) that a Device waits before retransmitting a packet, after it has timed out awaiting a response. This delay incorporates an exponential back-off component to help avoid collisions (which cannot be detected).

Long Transfer: A Block Transfer of more than 2000 Bytes of payload or a Stream Transfer that lasts more than two seconds.

Short Transfer: A Block or Stream Transfer that is not a Long transfer.

Channel Change Policy: The policy of whether to change channels during a Long Block or Stream Transfer in order to alleviate network congestion.

Data Rate Change Policy: The policy of whether to change data rates during a Long Block or Stream Transfer in order to increase transfer speed and alleviate network congestion.

Priority Policy: The policy of whether Long Block and Stream Transactions should proceed at High or Low Priority.

Stream Response Interval: The amount of time that should be allowed to lapse before waiting for an ACK for a Stream Data packet.

4 Block Transfers

As mentioned in the Definitions section of this document, a block transfer is a transfer of a fixed number of data bytes that do not fit into a single data packet, and are delivered reliably. There are a few varieties of Block transfers and a few questions to answer.

1. Is the transfer long or short? A "long" transfer is the transfer of more than 2000 bytes. Short transfers are less complicated than long transfers. For long transfers, a data rate, channel, priority, and chunk size need to be negotiated. For short transfers, they do not.
2. Is the Master one of the end-devices in the transfer? If the Master is one of the end devices in the transfer, there is no extra step of asking the Master for permission to proceed with the transfer.
3. Are the two end-devices within range of each other? If the two end-devices are within range of each other, things are simplified immensely since no Multi-Hop Repeater Clients are needed.

4.1 Short Transfers Involving The Master Device Where The Two End-Devices Are Within Range

This is the easiest transfer type. The transfer is a short transfer and as such, the network isn't very worried about the Block transfer hogging all the network bandwidth for an extended period of time. Hence, regardless of the network's policy of selecting channels, data rates, priorities, and chunk sizes, none are

relevant in this transfer. The data rate will not be changed. The channel will not be changed. The message will proceed at high priority. The chunk size will be one, which means that every Block Data packet must be ACK'd. In addition, since the transfer involves the Master, there will be no extra step of asking the Master if it is OK to proceed. Finally, no Repeaters will be needed so they do not need to be reserved.

4.1.1 Data To Be Transferred

Let us assume that Device 002 wants to transfer 100 bytes of data to the Master. Block Data packet payloads contain 25 bytes of data, so if things go optimally, exactly four Block Data packets will need to be sent. Let's assume that the data to be sent in an array of 100 bytes with values 1 to 100. So the four packets in this transfer are as follows.

1. 0x0102030405060708090A0B0C0D0E0F10111213141516171819
2. 0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
3. 0x333435363738393A3B3C3D3E3F404142434445464748494A4B
4. 0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

The "byte index" of a packet in a Block transfer is the offset from the first byte in the transfer. Hence the byte index of packet 1 is 0. Since each packet has a payload of 25 data bytes, the offsets of packets 2, 3, and 4 are 25, 50, 75, or 0x19, 0x32, 0x49 in hex respectively. The byte index is inserted above before each data packet. In addition, the numbering below will start at 0, not 1, the numbering has been changed from 0 to 3 instead of 0 to 3.

0. 0x000000-0x0102030405060708090A0B0C0D0E0F10111213141516171819
1. 0x000019-0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
2. 0x000032-0x333435363738393A3B3C3D3E3F404142434445464748494A4B
3. 0x000049-0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

As mentioned before, in short transfers, all chunk sizes are 1. Thus we will place 1 before the byte index for all packets above.

0. 0x01-0x000000-0x0102030405060708090A0B0C0D0E0F10111213141516171819
1. 0x01-0x000019-0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
2. 0x01-0x000032-0x333435363738393A3B3C3D3E3F404142434445464748494A4B
3. 0x01-0x000049-0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

Chunk Indexes are largely irrelevant for short transfers since the chunk size is 1, and hence trivial. Thus our four packets are split into four "chunks" of one packet each. This is trivial for short transfers, but will make more sense for longer transfers where the chunk size may not equal 1. Chunk indexes will vary from 0 to the Chunk Size minus 1. Again, since the Chunk Size is 1 here, 1 minus 1 is 0, so this is again trivial and the Chunk Index will equal 0 for all packets. The Chunk Index will vary for Long transfers. Here we add a Chunk Index of 0 in front of all packets.

0. 0x00-0x01-0x000000-0x0102030405060708090A0B0C0D0E0F10111213141516171819
1. 0x00-0x01-0x000019-0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
2. 0x00-0x01-0x000032-0x333435363738393A3B3C3D3E3F404142434445464748494A4B
3. 0x00-0x01-0x000049-0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

Before proceeding with the Block Transfer, a Message ID will be negotiated between the two devices. This will be the same for all packets within the transfer. Let's say that the two devices negotiated a Message ID of 0x456.

0. 0x456-0x00-0x01-0x000000-0x0102030405060708090A0B0C0D0E0F10111213141516171819
1. 0x456-0x00-0x01-0x000019-0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
2. 0x456-0x00-0x01-0x000032-0x333435363738393A3B3C3D3E3F404142434445464748494A4B
3. 0x456-0x00-0x01-0x000049-0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

Block Data packets have the following makeup. The packets above match. They are missing the Payload CRC and the Encryption Technique. The Payload CRC and Encryption Technique fields are discussed in more detail in the **ONE-NET Specification** document and are thus not discussed in this document.

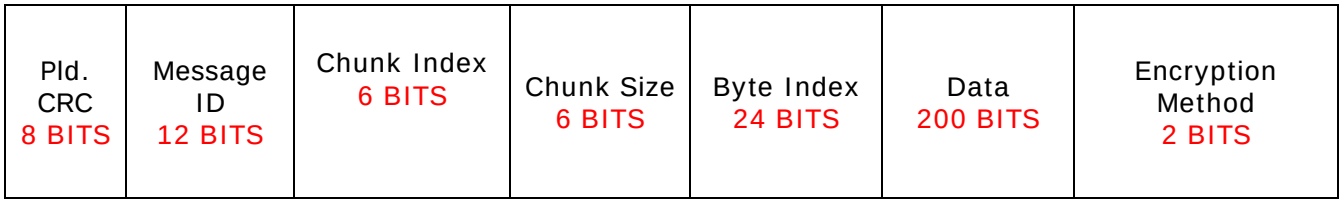

Figure 4.1 – PCON Structure for Block Data Packets

Note that the Chunk Indexes and Chunk Sizes are 6 bits, not 8, so combining those two fields into two six-bit fields, we get (combining red section – Chunk Index And Size).

Before Combining Chunk Index And Chunk Size

0. 0x456-**0x00-0x01**-0x000000-0x0102030405060708090A0B0C0D0E0F10111213141516171819
1. 0x456-**0x00-0x01**-0x000019-0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
2. 0x456-**0x00-0x01**-0x000032-0x333435363738393A3B3C3D3E3F404142434445464748494A4B
3. 0x456-**0x00-0x01**-0x000049-0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

After Combining Chunk Index And Chunk Size

0. 0x456-**0x001**-0x000000-0x0102030405060708090A0B0C0D0E0F10111213141516171819
1. 0x456-**0x001**-0x000019-0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
2. 0x456-**0x001**-0x000032-0x333435363738393A3B3C3D3E3F404142434445464748494A4B
3. 0x456-**0x001**-0x000049-0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

Again, the 25 bytes after the last hyphen is the data portion. That is the part the application code cares about. The CRC and the Message ID are both for security purposes and the Message ID also makes it easier to tell which message goes with which message. The Chunk Index, Chunk Size, and Byte Index are "housekeeping" variables that keep track of what part of the Block transfer is currently occurring.

So the Client device 002's goal is to transfer the 100 bytes above. There are several steps involved in doing so. Recall that Device 002 is the Source and Device 001 (the Master) is the recipient. Not all steps are relevant here. They are as follows…

0. Device 002 must check device 001's Features to determine whether 001 can handle a Block Transaction.
1. Device 002 must find a Route to 001 and determine the Repeaters required, if any.
2. Device 002 must decide what channel and data rate to use.

3. If the data rate and/or channel to use are different from the base data rates, Device 002 must tell all devices involved, including any Repeaters, to change data rates and / or channels. It must then re-establish that the Route chosen works on the new channel and data rate.
4. Device 002 must contact each device involved and give them the details of the proposed transfer. All devices must be capable and must agree.
5. If the Master is not involved, the Master should also be informed of the parameters of the transfer and must agree to it.
6. The data is transferred to the recipient device in a series of Block Data packets.
7. When the data is transferred successfully, a termination message is sent with a Success status.
8. If the data rate and / or channel was changed, the devices involved should return to the base data rate and the Network channel.

Numbers 2, 3, 5, and 8 are relevant only for Long transfers. In addition, number 5 would be irrelevant even for a Long transfer since the destination is the Master and hence after step 4, the Master would know the details of the transfer. Step 5 is relevant only for Client-to-Client transfers.

In this example, it has been stipulated that devices 001 and 002 are within range of each other. Hence the Route message will be successful and no repeaters are needed. Hence in our case, if everything goes right, there will be eight messages from Device 002 to 001.

1. Two Route Messages.
2. One Setup message.
3. Four Block Data message.
4. One “Termination With Success” Message.

Each of the eight messages will be ACK’d by the Master. This assumes that both devices are in sync with each other regarding Message IDs, neither needs to adjust the transfer properties, and all packets are received quickly with no errors and on the first attempt.

4.1.2 Block Data Transfer Administration Setup Message

Recall the following information for this transfer. Let's assume we are using Channel 6. Let's also assume a "Chunk Pause" of 50 milliseconds (more details on that later).

1. This is a Short Transfer
2. This is a Block Transfer
3. The Priority is High
4. 100 bytes are to be transferred
5. Chunk Size is 1
6. Chunk Pause is 50 milliseconds
7. Data Rate will be the base data rate (38,400 baud).
8. Source Device is 002
9. Destination Device is 001

See Section 4.2.9.1 of the ONE-NET Specification for more information on Single Packets. Device 002 will be sending Device 001 a Single Data packet. The Message Type of that Single Data Packet will be Administration (or ON_ADMIN_MSG to use the ONE-NET constant). The Administration Message Type will be ON_REQUEST_BLOCK_STREAM (which has code 0x10 – see Section 4.2.9.1.3.2 in the ONE-NET Specification). The makeup of that message is as follows.

Type / Priority / Hops 8 BITS	Number Of Bytes To Transfer 32 BITS	Chunk Size 8 BITS	Fragment Delay 16 BITS	Chunk Pause 16 BITS	Channel 8 BITS	Data Rate 8 BITS	Timeout 16 BITS
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 4.2 – Block Transfer Setup Message

4.1.3 Flags Field

Three pieces of information are contained in the flags field: Transfer type (Block Or Stream), Priority (High or Low), and the Number Of Hops required (0 – 7, relevant only if both devices are Multi-Hop-capable and not within range of each other).

Unused 2 BITS	Transfer Type (0 = Block, 1 = Stream) 1 Bit	Priority (1 = Low, 2 = High) 2 BITS	Number Of Hops 3 BITS

Figure 4.3 – Block / Stream Setup Message Flags Byte

For our example, this is a Block transaction, the priority is High(2 in decimal, 10 in binary), and the number of Hops is 0 since the devices are within range.

Unused 00	Transfer Type (0 = Block, 1 = Stream) 0	Priority (1 = Low, 2 = High) 10	Number Of Hops 000

Figure 4.4 – Block / Stream Setup Message Flags Byte

So the flag bits are 00010000. Hence the flags byte will be 0x10 for this example. Filling in the setup message to reflect that…

Type / Priority / Hops 0x10	Number Of Bytes To Transfer 32 BITS	Chunk Size 8 BITS	Fragment Delay 16 BITS	Chunk Pause 16 BITS	Channel 8 BITS	Data Rate 8 BITS	Timeout 16 BITS
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 4.5 –Block Transfer Setup Message With Type / Priority / Hops

Adding the number of bytes to transfer to the message, we have 100, which is 0x00000064 in hex. The chunk size is 1, which is 0x01. The chunk pause is 50 milliseconds, which is 0x0032 in hex. The channel is 6, which is 0x06. The data rate will be the base data rate of 38,400, which is 0 or 0x00. Updating our message, we get the following…

Type / Priority / Hops 0x10	Number Of Bytes To Transfer 0x00000064	Chunk Size 0x01	Fragment Delay 16 BITS	Chunk Pause 0x0032	Channel 0x06	Data Rate 0x00	Timeout 16 BITS
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 4.6 –Updated Block Transfer Setup Message

4.1.4 Chunk Size

The Chunk Size for the first and last 1000 bytes of a Block Transfer is always 1 for all Block transfers, regardless of size. The value sent in the message is the Chunk Size to use in the middle of the Block transfer (i.e. not the first 1000 or last 1000 bytes). Therefore, regardless of what is in the message, a Short Block transfer will always have a Chunk Size of 1 for the whole message. Hence the Chunk Size field in a Short transfer is irrelevant and hence ignored. Therefore the actual message may not actually contain a 1 in the Chunk Size field in the actual message.

The default timeout period is 3000 milliseconds or 3 seconds. As with all default values, the application code can change this value. For our example, well leave it as 3000 milliseconds. The timeout period (3000 milliseconds) is interpreted as follows: if either device has not heard from the other device within 3000 milliseconds, it is assumed that communication has been lost and the Block transfer should be considered a failure and abort. If data rates and / or channels have been changed, the device should revert back to the original data rate and channel. 3000 is 0x0BB8 in hexadecimal. Filling the message in with that information, we get…

Type / Priority / Hops 0x10	Number Of Bytes To Transfer 0x00000064	Chunk Size 0x01	Fragment Delay 16 BITS	Chunk Pause 0x0032	Channel 0x06	Data Rate 0x00	Timeout 0x0BB8
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 4.7 –Updated Block Transfer Setup Message With Timeout Filled In

Each device has two fragment delay times, both measured in milliseconds: one for high-priority transfers, one for low-priority transfers. The high-priority fragment delay is always less than or equal to the low-priority fragment delay. The master device will assign fragment delays to all other devices and can change them at any time. The default times are 25 milliseconds for high-priority and 125 milliseconds for low-priority. This is a high-priority message, so let's assume we are using the default. 25 in decimal is equal to 0x0019 in hexadecimal. The fragment delay is the time that the sending device pauses between sending successive data packets when it is NOT waiting for an acknowledgement. It is therefore irrelevant in this case, since when the Chunk Size is 1, all Block Data packets will wait for an ACK (see examples later in the Document where the Chunk Size is greater than one for how to decide whether a packet should be ACK'd. Again, in this case, all packets need to be acknowledged, so the fragment delay is irrelevant here. However it is still filled in). Let's update our packet with the fragment delay (0x0019). The destination is the Master Device (001). Encoded, this value is 0xB4BC.

Type / Priority / Hops 0x10	Number Of Bytes To Transfer 0x00000064	Chunk Size 0x01	Fragment Delay 0x0019	Chunk Pause 0x0032	Channel 0x06	Data Rate 0x00	Timeout 16 BITS
Encoded Dst. DID 0xB4BC	Estimated Transfer Time 32 BITS						

Figure 4.8 –Updated Block Transfer Setup Message With Fragment Delay And Destination Filled In

Finally, the sending device makes a very rough estimate of the time that it anticipates the entire transfer to take in milliseconds. This is a very rough estimate based on the data rate, number of bits to transfer, and the fragment delays and chunk pause. Let's just say that we estimate the transfer time to equal 1250 milliseconds, which is 0x000004E2 in hexadecimal. Our final message to send to the Master is the following…

Type / Priority / Hops 0x10	Number Of Bytes To Transfer 0x00000064	Chunk Size 0x01	Fragment Delay 0x0019	Chunk Pause 0x0032	Channel 0x06	Data Rate 0x00	Timeout 16 BITS
Encoded Dst. DID 0xB4BC	Estimated Transfer Time 0x000002E2						

Figure 4.9 –Final Block Transfer Setup Message

This message will be set up by the core ONE-NET code as defaults. Before actually being sent to the recipient (001), the application code may decide to change whatever it likes. Let's assume it is satisfied with the defaults. It will now send the message above to the recipient device, which is the Master, and wait for a response.

4.1.5 Sending Device Sends The Setup Message To The Recipient Device

The setup device pushes the message above onto the outgoing message queue. The message is popped and sent. Flags are set on the sending device that a Block Message is in progress and a timer is set for the last response from the recipient. If the recipient does not reply back within the allotted time, the block message will abort.

4.1.6 Recipient Device Receives The Setup Message

The recipient device receives the request from Device 002 to start a Block transfer. The recipient, recall, is the Master. It will quickly check a few things…

1. Am I, the Master, capable of receiving a Block transfer?
2. Am I in the middle of another Block or Stream transfer at the moment?

If the Master is not capable or is in the middle of another Block or Stream transfer at the moment, it will NACK the message. Assuming neither is the case, the Master will look at a few other things and pass on the decision to the application code to decide whether to grant permission for this transfer. Some of the things it may look at are the following…

1. Generally if the Master is in the middle of any of the following, it will ask the sender to delay the transaction.
    a. The network is currently adding a device.
    b. The network is currently deleting a device.
    c. The network is currently in the middle of a key change or expects to very soon.
    d. A device that normally sleeps is about to check in or is expected to at some point during the Block transfer.
2. The Master will also look at all the devices involved and check their Features to make sure all devices are capable of fulfilling the request. If not, the request will be NACK'd.
3. There are also a variety of other reasons that the Master might reject the transfer.
    a. If the transfer requires a Repeater, the Master might not be willing to reserve that Repeater for an extended period of time.
    b. If one or more of the devices involved is requesting a channel or data rate change and is on some other device's per table, the Master may not allow the transfer.
    c. The Master might ask the sending device to delay the transfer until a later time when the network is less congested.

In addition, the Master or the recipient (if not the Master) may not completely reject the message, but instead change the priority, the Chunk Size, the Chunk Pause, the fragment delay, etc. It will then NACK the request, but give a NACK Reason explaining why along with an alternate value. The sending device can decide whether to repackage the request and try again.

In this case, we will assume that the recipient (Master) approves the request as-is and ACKs it.

4.1.7 Sending Device Receives An ACK For The Transfer

At this point the sender has received an ACK for the transfer and can actually start sending data. Recall that it is not switching channels or data rates, that its timeout time is 3000 milliseconds, its fragment delay is 25 milliseconds, the chunk size is 1, the chunk pause is 50 milliseconds, the priority is high, and there are no repeaters involved. Recall also that the fragment delay is irrelevant for this transfer. Device 002 needs to send the following four data packets to Device 001, as discussed earlier.

Chunk Index	Chunk Size	Byte Index	Data
0	1	0	0x0102030405060708090A0B0C0D0E0F10111213141516171819
0	1	25	0x1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F303132
0	1	50	0x333435363738393A3B3C3D3E3F404142434445464748494A4B
0	1	75	0x4C4D4E4F505152535455565758595A5B5C5D5E5F6061626364

Figure 4.10 –Block Data Packets To Send

It will start sending immediately. It will wait for an ACK or NACK if and only if the Chunk Index is exactly one less than the Chunk Size. The Receiving Device should send an ACK or NACK if and only if the Chunk Index is exactly one less than the Chunk Size. Hence, as mentioned, when the Chunk Size is 1, all data packets will wait for an Acknowledgement.

4.1.8 Storage Buffers And Loading Data Into Packets

ONE-NET makes few assumptions as far as how the application code stores the data to be sent and how much memory the application code has available. In this example, the transfer size is 100 bytes. ONE-NET does NOT assume that the bytes to be transferred are stored in an array of 100 contiguous bytes. That is a very common way of storing things, but it is not assumed. The application code may be very tight on RAM or it may not. The message may involve some formula where each subsequent packet is determined by the last packet and hence not all packets need to be stored. Again, ONE-NET makes no assumptions about what the data represents and how it is stored. Therefore all ONE-NET does when sending a Block data packet is call an application-level function called *one_net_block_get_next_payload()*. The sending device application code (in this case, 002), copies in the next 25 bytes to send based on the parameters passed to this function. To figure out which bytes to copy, a mathematical formula is used based on packet size, byte index, and chunk index.

4.1.8.1 Determining The Relevant Bytes from Indexes

The relevant bytes being transferred are quite easy to determine based on the byte index and the chunk index. Recall that the packet size is 25. That is, 25 bytes of data are transferred in each Block Data packet. The formula is therefore as follows…

Relevant Bytes Formula
Starting Byte = Byte Index + (25 x Chunk Index) *Ending Byte = Minimum (Starting Byte + 24, Transfer Size – 1)*

Figure 4.11 –Formula To Calculate Starting And Ending Byte Indexes

Calculating the ending byte should be obvious. Assume a 90 byte transfer. Each packet contains 25 bytes. The last packet may or may not include garbage bytes. Byte indexes are 0 to 89. There will be four packets…

1. Bytes 0 to 24
2. Bytes 25 to 49
3. Bytes 50 to 74

4. Bytes 75 to 89

Each packet, including the last packet, transfer 25 bytes. However, the last 10 bytes of the last packet will be garbage bytes, so just ignore them. Hence the formula gives you the starting byte. You then add 24 to get the last byte. Compare that value to the number of bytes being transferred and you decide whether this represents a full packet or a partial packet.

Back to our example…

Chunk Index	Chunk Size	Byte Index	Start Byte
0	1	0	0 + (0 x 25) = 0
0	1	25	25 + (0 x 25) = 25
0	1	50	50 + (0 x 25) = 50
0	1	75	75 + (0 x 25) = 75

Figure 4.12 –Calculating The Start Byte From Indexes

This is a trivial example since the chunk index is always 1. A more complex example would be in the middle of a Long transfer of chunk size 6.

Chunk Index	Chunk Size	Byte Index	Start Byte
0	6	1000	1000 + (0 x 25) = 1000
1	6	1000	1000 + (1 x 25) = 1025
2	6	1000	1000 + (2 x 25) = 1050
3	6	1000	1000 + (3 x 25) = 1075
4	6	1000	1000 + (4 x 25) = 1100
5	6	1000	1000 + (5 x 25) = 1125
0	6	1150	1150 + (0 x 25) = 1150
1	6	1150	1150 + (1 x 25) = 1175
2	6	1150	1150 + (2 x 25) = 1200
3	6	1150	1150 + (3 x 25) = 1225
4	6	1150	1150 + (4 x 25) = 1250
5	6	1150	1150 + (5 x 25) = 1275

Figure 4.13 –Calculating The Start Byte From Indexes

Block Data is transferred one Chunk at a time. As mentioned, when the Chunk Size is 1, which it always is for Short Transfers, this is equivalent to one packet at a time. The general approach to block transfers is to reserve a buffer of memory at least the size of a full Chunk. However, again, this is entirely up to the application

code. ONE-NET notifies the application code every time a Block Data packet is received. It also notifies the application code when a full Chunk has been received. A common scheme is to do nothing with the individual packets, but to only handle a chunk at a time. If the receiving device has enough RAM to store an entire Block Data transfer, it may do nothing until it receives the whole message. Another common setup is to have the ONE-NET device pass have limited storage, but pass the data along to a much more powerful desktop computer which actually processes the data. Again, ONE-NET makes no assumptions regarding how the data is stored or used.

4.1.8.2 Block Chunks Sent / Received Buffer

Again, in our example, the Chunk Size is 1. Let's pretend the Chunk Size was something larger: 6. Let's assume that we are in the middle of a Long transfer. Let's assume that we are about to transfer bytes 1000 to 1149. Refer to Figure 4.12 above. Hence all of the setup messages have already been handled successfully and we have transferred 40 packets of 25 data bytes each so far, for a total of 1000 data bytes. As mentioned preciously, for the first 1000 bytes, the Chunk Size is 1. Each packet must be ACK'd. Starting with byte 1000, we will start sending data one chunk at a time. Again, the Chunk Size is now 6.

Both the sender and receiver must initialize an array of 6 bits. The payload will actually be a five byte / 40-bit array, but only 6 of the 40 bits will be relevant. A 1 bit represents true, a 0 it represents false. The 6 bits represent each of he six packets that make up a Chunk, the number of relevant bits equaling the Chunk Size. A bit is true if the packet has been successfully transmitted. Hence at the start of the Chunk transfer, both sides initialize the bits to all 0…

000000

The full 40-bit / five byte array is as follows. The right-most (red) 34 bits are irrelevant. The six left bits are relevant. No packets have been sent.

000000**0000000000000000000000000000000000**

The sending device will rapidly send all six packets without waiting for a response. After sending the last packet, it will wait for a response from the recipient. After each packet is sent, the sender will mark the corresponding packet with a 1. Recall that the chunk we are sending is bytes 1000 to 1149. This is six packets, 25 bytes each.

Let's assume the fragment delay is 25 milliseconds. Let's assume that each packet takes 10 milliseconds to transmit. Hence a packet is transferred every 10

milliseconds + 25 milliseconds = 35 milliseconds total time. The chart below represents the first attempt at the Chunk Transfer with times.

Chunk Index	Chunk Size	Byte Index	Start Byte	Send Time (ms)	Start Byte	Array Before Sending	Array After Sending
0	6	1000	1000	0	1000	000000	100000
1	6	1000	1025	35	1025	100000	110000
2	6	1000	1050	70	1050	110000	111000
3	6	1000	1075	105	1075	111000	111100
4	6	1000	1100	140	1100	111100	111110
5	6	1000	1125	175	1125	111110	111111

Figure 4.14 –Chunk Index Array And Send Times

Note that the sender DOES NOT wait for responses and does not listen for responses. The recipient should therefore not SEND a response. As mentioned, responses are expected after the LAST packet in a chunk is sent and only then.

In an ideal world, all six packets will have been received and accepted by the recipient. The sender has calculated the expected time that it should wait to receive a response. It must wait at least this long before assuming communication has been lost. If it does not hear a response within this time period, it will resend the last packet, hoping to get a response.

The recipient, upon receiving the last packet, should send back an array of the packets it has received and accepted. Let's assume that the recipient received and accepted packets 1, 2, and 5. The receiver should send back an ACK containing the following array…

011001 → Note that bits 1, 2, and 5 are set to true. Bits 0, 3, and 4 were not received or were rejected, so the recipient wants them sent again. Upon receiving this array, the sending device will update its array and resend any missing packets (i.e. packets flagged with 0). Note that since the LAST packet is the trigger for a response, it is ALWAYS sent even if it has already been successfully received.

The next attempt at sending the chunk is as follows. Packets 0, 3, 4, and 5 (the bits with 0's plus the last bit) will be sent.

Chunk Index	Chunk Size	Byte Index	Start Byte	Send Time (ms)	Start Byte	Array Before Sending	Array After Sending
0	6	1000	1000	0	1000	011001	111001
3	6	1000	1075	35	1075	111001	111101
4	6	1000	1100	70	1100	111101	111111
5	6	1000	1125	105	1125	111111	111111

Figure 4.15 –Chunk Index Array And Send Times

Again, the receiving device will keep track of what has been received and send back an array. Let's assume that this time it received packets 0 and 4, but still needs packet 3. The corresponding array would be 111011. The sending device starts the process all over again. It will send packets 3 and 5.

Chunk Index	Chunk Size	Byte Index	Start Byte	Send Time (ms)	Start Byte	Array Before Sending	Array After Sending
3	6	1000	1075	35	1075	111011	111111
5	6	1000	1125	70	1125	111111	111111

Figure 4.16 –Chunk Index Array And Send Times

Let's assume that this time we get no response. We have calculated a time that seems reasonable to wait. Let's pretend that time is 100 milliseconds. If we get no response, we will simply resend the last packet every 100 milliseconds until we get a response. If we never get another response, there is a timeout time that was agreed upon in the initial message (3000 milliseconds in this case). If 3000 milliseconds go by with no response, the message will be aborted.

Let's assume now that the recipient has finally received all the packets in the chunk, so we have successfully sent this chunk. At this point, the recipient will actually NACK the message, but it is a good NACK. Nothing has gone wrong. The NACK will have a NACK Reason of "Invalid Byte Index" and will be accompanied by a new byte index. In this case, the byte index will be 1150, the

first byte of the next chunk. So we've been NACK'd, but it is the one NACK which means that something good has happened. We sent a packet with Byte Index 1000 and the receiving device is telling us that it no longer needs any data in this chunk because it's ready for the next one.

At this point, we go into "Chunk Pause" mode, discussed earlier. We've successfully sent a chunk of the message and we now pause to allow any other single messages to be sent in the network so things do not get too congested. The length of the chunk pause will vary from network to network. Recall that we chose 50 milliseconds to be the chunk pause, so the sending device will wait this amount of time before sending the next Chunk. At that time, the process will be repeated and it will send bytes 1150 to 1299. Then 1300 to 1449, etc. finally it will reach the last 1000 bytes and the chunk size will revert to 1, as discussed.

4.1.8.3 Possible Responses From Recipient Block/Stream Messages

Besides sending back a bit array representing the packets received or a NACK pointing out the correct byte index, any of the following responses can be sent. Note that the list includes the byte array and byte index messages.

4.1.8.3.1 Array Of Packets Received

Block only. See Section 4.1.8.2. This is an ACK. The ACK handle is ON_ACK_BLK_PKTS_RCVD. The payload is a bitwise Boolean array of packets that have been received and accepted as valid. A 0 bit represents the packet has not been received or is invalid. A 1 bit represents the packet has been received and is valid.

4.1.8.3.2 Invalid Byte Index

Block only. See Section 4.1.8.2. This is a NACK. The NACK Reason is ON_INVALID_BYTE_INDEX. The NACK handle is ON_NACK_VALUE. The NACK payload is the new byte index to use divided by 25 (since each packet has 25 bytes).

4.1.8.3.3 Change Chunk Pause

Block only. This is a NACK. The NACK Reason is ON_INVALID_CHUNK_DELAY. The NACK handle is ON_NACK_TIME_MS. The NACK payload is the new chunk pause in milliseconds.

4.1.8.3.4 Change Chunk Size

Block only. This is a NACK. The NACK Reason is ON_INVALID_CHUNK_SIZE. The NACK handle is ON_NACK_VALUE. The NACK payload is the new chunk size. Recall that chunk sizes are not relevant (are always 1) in the first and last 1000 transfer bytes.

4.1.8.3.5 Change Fragment Delay

Block or Stream. This is a NACK. The NACK Reason is ON_INVALID_FRAG_DELAY. The NACK handle is ON_NACK_TIME_MS. The NACK payload is the new fragment delay in milliseconds.

4.1.8.3.6 Slow Down The Message

Block or Stream. This is a NACK. The NACK handle is ON_NACK_SLOW_DOWN_TIME_MS. The NACK payload is the amount of time to slow down in milliseconds. The fragment delay will be increased by this amount.

4.1.8.3.7 Speed Up The Message

Block or Stream. This is a NACK. The NACK handle is ON_NACK_SPEED_UP_TIME_MS. The NACK payload is the amount of time to speed up in milliseconds. The fragment delay will be decreased by this amount.

4.1.8.3.8 Pause The Message

Block or Stream. This is a NACK. The NACK handle is ON_PAUSE_TIME_MS. The NACK payload is the amount of time to pause in milliseconds. This is generally sent when the recipient gets busy and wants to make a one-time pause of the message for, say, five seconds, while it attends to some other business. It is again a ONE-TIME pause. Upon receiving this reply, the sender goes into Chunk Pause mode. The time of the Chunk Pause is the time specified in the message. Again, this message DOES NOT change the overall Chunk Pause time for subsequent Chunk Pauses.

4.1.8.3.9 Change Priority

Block or Stream. This is a NACK. The NACK Reason is ON_INVALID_FRAG_PRIORITY. The NACK handle is ON_NACK_VALUE. The NACK payload is the new priority for the transfer.

4.1.8.3.10 Terminate The Transaction

Block And Stream. This may or may not be in the form of an ACK or a NACK. If possible, devices wishing to terminate are encouraged to send a Single Administration message. Block And Stream messages can be terminated by the recipient, the Master, a Repeater, or some other device saying that the device wished to terminate the transaction for whatever reason. This message will occur during a Chunk Pause.

If the recipient device does not have time to gracefully terminate the message, it can simply either stop responding, in which case the message will timeout, or it can respond with a NACK Reason of ON_NACK_RSN_ABORT.

See Section 5 for more on Termination messages.

4.1.8.3.11 Application-Specific Acknowledgement Messages

All other ACKs and NACK Reasons are allowed, but only the ones above are interpreted by the core ONE-NET code. All other ACKs and NACKs are passed to the Application-Level code. It will interpret and handle the ACK or NACK. Note that all ACKs and NACKs are passed to the Application-Level code regardless of type, including the ones above. The Application-Level code always has a chance to be informed of and change any response. In most cases, however, the Application-Level code will not do anything with the cases that the core ONE-NET code handles.

4.1.9 Back To Our Original Example

To recap, device 002 is sending 100 bytes to device 001 in a Block transfer. Much of the issues above are irrelevant. There are no Repeaters involved. The Master is involved so there is no extra step of requesting permission from Repeaters or the Master after the recipient has agreed since the master IS the recipient. Similarly there are no issues involving priorities, channel changes, data rate changes, or fragment delays. Optimally the transfer will work as follows if no messages are lost and there is no other traffic in the network. The packet sizes involved are as follows.

Packet Type	Number Of Bits	Data Rate	Transfer Time(ms) Bits / Baud
Route Packet	416	38.4 Bits/ms	10.83
Block Data Packet	496	38.4 Bits/ms	12.92
Block Setup Admin Message	416	38.4 Bits/ms	10.83
Block Termination Admin Message	328	38.4 Bits/ms	8.54
Response Packet	240	38.4 Bits/ms	6.25

Figure 4.17 –Packet Send Times

The packets to be sent are as follows. Assuming it takes 10 seconds on each end to process the information and get a clear channel, the following packets will be sent under ideal circumstances. Recall that the Chunk Pause is 50 milliseconds. Hence the 50 millisecond pauses in addition the after each Acknowledgement of a block data packet.

Action	Packet Type	Start Time (ms)	Transfer Time (ms)	End Time (ms)
Route Packet From 001 to 002	Route Packet	0.00	10.83	10.83
Return Route Packet From 002 to 001	Route Packet	20.83	10.83	31.66
Route Packet From 001 to 002	Route Packet	41.66	10.83	52.49
Return Route Packet From 002 to 001	Route Packet	62.49	10.83	73.32
Block Message Setup Packet From 001 to 002	Block Setup Admin Packet	83.32	8.54	91.86
Response To Setup Message From 002 To 001	Response Packet	101.86	6.25	108.11
Transfer Bytes 0 To 24 From 001 to 002	Block Data Packet	118.11	12.92	131.03

ACK For Bytes 0 To 24 From 002 To 001	Response Packet	141.03	6.25	147.28
Transfer Bytes 25 To 49 From 001 to 002	Block Data Packet	207.28	12.92	220.20
ACK For Bytes 25 To 49 From 002 To 001	Response Packet	230.20	6.25	236.45
Transfer Bytes 50 To 74 From 001 to 002	Block Data Packet	296.25	12.92	309.37
ACK For Bytes 50 To 74 From 002 To 001	Response Packet	319.37	6.25	325.62
Transfer Bytes 75 To 99 From 001 to 002	Block Data Packet	385.62	12.92	398.54
ACK For Bytes 75 To 99 From 002 To 001	Response Packet	408.54	6.25	414.79
Termination Message From 002 to 001	Block Termination Admin Message	424.79	8.54	433.33
ACK For Termination Message From 001 To 002	Response Packet	443.33	6.25	449.58

Figure 4.18 –Block Transaction From Start To Finish

Thus under optimal circumstances, the Block Transfer will involve sixteen packets, eight each from devices 001 and 002, and will have a duration of 450 milliseconds.

The ten seconds for processing and looking for a clear channel can be reduced in fast chips. Looking for a clear channel will take about five seconds regardless of the chip. The Chunk Pause of 50 milliseconds can be reduced dramatically to close to zero. If it was reduced to 0, 150 milliseconds (50 ms times (Number of chunks(4) minus 1)) could be shaved off, or a total transfer time of 300 milliseconds. Changing the ten seconds processing and clear channel time to five seconds will take off an additional 5 seconds off of all of the packets except for the first, so the time can be reduced another 735 milliseconds to 225 milliseconds.

Adding up all of the transfer times, we get 72.90 milliseconds to transmit the administration messages, then 19.17 seconds to transfer each block data packet and its response. This is all at the 38,400 baud data rate. The 72.90 milliseconds will be constant regardless of the transfer size. Thus for Long transfers, the number of Administration messages (four) is the same as for short messages. Hence the transfer time can be calculated largely from the number of bytes to be transferred. The administration overhead largely disappears from the equation. It should also

be quickly noted that doubling the data rate will cut the 19.17 seconds figure in half and quadrupling the data rate will cut it in half again. The processing time and time to look for a Clear Channel will remain constant. Thus raising the data rate can significantly reduce the overall transfer time. Similarly, since under ideal circumstances there is only one Acknowledgement message per Chunk, increasing the Chunk Size can also greatly reduce the length of the transfer. These issues are discussed later in the document.

5 Block / Stream Termination Messages

The Master, Source, Recipient, or a Repeater can terminate a Block or Stream transaction. The reason may be that the Message is no longer needed for whatever reason, connectivity has been reduced, the airwaves have become congested, or a device becomes busy or is needed for some other purpose. The device terminating the transfer should give a reason for the termination and a possible status code. If the sending device of the transfer is not the device terminating the transfer, the terminating device should send a Single Administration Message to the sending device. The sending device should then send this termination device to all relevant devices, including the recipient, the Master, and any Repeaters. All devices should ACK the message, pause slightly for any follow-up messages, then return to the base data rate and channel. See Sections 15 and 4.2.9.1.3.2 of the **ONE-NET Specification** document for a list of Message Status codes and the makeup of the ON_TERMINATE_BLOCK_STREAM administration message. It is also below. If the message was successful, no NACK Reason is needed, so ON_NACK_RSN_NO_ERR should be specified as the NACK Reason. The terminating device is the device initiating the termination of the message.

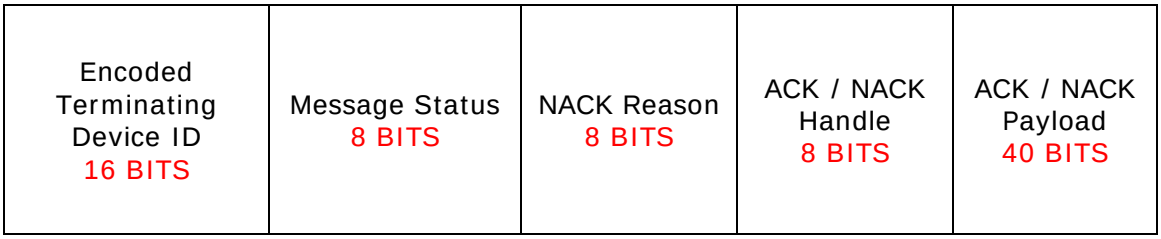

Figure 5.1 –Block/Stream Terminating Message

6 Stream Data Transfers

Stream Data Transfers are similar to Block Transfers. There are some differences. Due to the real-time nature of Stream communications, data is not repeated. There are no Chunks or Chunk Pauses. Data is simply sent rapidly with a timestamp. Every five seconds there is a slight pause while the Sender waits for an Acknowledgement.

See Section 4.2.9.6 in the **ONE-NET Specification** document for more information on Stream Data transfers. The Stream Data Packet's format, repeated here, is repeated.

Pld. CRC 8 BITS	Message ID 12 BITS	Unused 11 BITS	Response Needed 1 BIT	Elapsed Time 24 BITS	Data 200 BITS	Encryption Method 2 BITS

Figure 6.1 – PCON Structure for Stream Data Packets

As with the Block Data packet, 25 bytes are transferred at a time. The Elapsed Time is the Elapsed Time from the start of the message. The "Response Needed" bit is set true every five seconds. At that point, the Recipient, as with Block transfers, will wait for a response just so it knows that the connection is still alive. The receiving device should send a response if and only if the Response Needed flag is set. It can send a simple ACK or it can adjust the fragment delay, just as it can with the Block Message. It can also terminate the message, as with Block Data Transfers. The setup, routing, permission, and termination messages are identical to the Block Data messages. The difference is that the Chunk Size, Data Transfer Size, and Chunk Pause fields are irrelevant for Stream Transfers. What IS much more relevant is the Estimated Time Field. In the case of a Stream transfer, this is not an estimated time as it is with a Block Transfer. It is an exact time. If the time is unknown (i.e. for an intercom message), then the Estimated Time will be 0. That signifies that the Estimated Time is unknown. The Recipient will know that the Stream Message is complete when the Termination message is sent.

6.1 Short Versus Long Transfers

As with Block Data Transfers, there are Long And Short Data transfers. A Long Stream Data Transfer is any Steam transfer that has an unspecified time or has a time of more than two seconds. As with Short Block transfers, permission from

the Master (unless the Master is the recipient) is not required for Short Stream transfers, priority will be high, and data rates and channels are not changed.

6.2 Short Transfer From Device 002 To 001 (Master)

As with the previous example of a Short Block Transfer, the following is the process when sending a 2 Second Short Stream Transfer to the Master from device 002. The packet lengths are the same as for the Block transfer.

Packet Type	Number Of Bits	Data Rate	Transfer Time(ms) Bits / Baud
Route Packet	416	38.4 Bits/ms	10.83
Stream Data Packet	496	38.4 Bits/ms	12.92
Stream Setup Admin Message	416	38.4 Bits/ms	10.83
Stream Termination Admin Message	328	38.4 Bits/ms	8.54
Response Packet	240	38.4 Bits/ms	6.25

Figure 6.2 –Packet Send Times

Assume a high-priority transfer from 002 to 001 lasting two seconds with a fragment delay of 10 milliseconds.

Type / Priority / Hops 8 BITS	Number Of Bytes To Transfer 32 BITS	Chunk Size 8 BITS	Fragment Delay 16 BITS	Chunk Pause 16 BITS	Channel 8 BITS	Data Rate 8 BITS	Timeout 16 BITS
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 6.3 – Stream Transfer Setup Message

As mentioned, the type is Stream, the Priority is High, and the number of Hops is 0. The flags are there fore as follows.

Unused 00	Transfer Type (0 = Block, 1 = Stream) 1	Priority (1 = Low, 2 = High) 10	Number Of Hops 000

Figure 6.4 – Block / Stream Setup Message Flags Byte

The flags bits will therefore be 00110000, so the flags byte is 0x30. Filling that in, we get.

Type / Priority / Hops 0x30	Number Of Bytes To Transfer 32 BITS	Chunk Size 8 BITS	Fragment Delay 16 BITS	Chunk Pause 16 BITS	Channel 8 BITS	Data Rate 8 BITS	Timeout 16 BITS
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 6.5 – Stream Transfer Setup Message

The number of bytes, chunk size, and chunk pause are irrelevant, so we'll fill them with zeroes.

Type / Priority / Hops 0x30	Number Of Bytes To Transfer 0x00000000	Chunk Size 0x00	Fragment Delay 16 BITS	Chunk Pause 0x0000	Channel 8 BITS	Data Rate 8 BITS	Timeout 16 BITS
Encoded Dst. DID 16 BITS	Estimated Transfer Time 32 BITS						

Figure 6.6 – Stream Transfer Setup Message

The number of bytes, chunk size, and chunk pause are irrelevant, so we'll fill them. Assume a timeout of 3000 milliseconds, which is 0xBB8, and an Estimated Transfer Time of 2000 milliseconds, which is 0x7D0. Filling these values in…

Type / Priority / Hops 0x30	Number Of Bytes To Transfer 0x00000000	Chunk Size 0x00	Fragment Delay 16 BITS	Chunk Pause 0x0000	Channel 8 BITS	Data Rate 8 BITS	Timeout 0x0BB8
Encoded Dst. DID 16 BITS	Estimated Transfer Time 0x000007D0						

Figure 6.7 – Stream Transfer Setup Message

The destination DID is 001 which is 0xB4BC encoded. Filling it in…

Type / Priority / Hops 0x30	Number Of Bytes To Transfer 0x00000000	Chunk Size 0x00	Fragment Delay 16 BITS	Chunk Pause 0x0000	Channel 8 BITS	Data Rate 8 BITS	Timeout 0x0BB8
Encoded Dst. DID 0xB4BC	Estimated Transfer Time 0x000007D0						

Figure 6.8 – Stream Transfer Setup Message

The data rate is 38,400, which is Data Rate 0. Assume the channel is channel 6. These are both irrelevant for Short transfers. We use the base data rate and the network channel. We'll fill it in anyway. Finally, the fragment delay is 10 milliseconds, which is 0x0A. filling everything in…

Type / Priority / Hops 0x30	Number Of Bytes To Transfer 0x00000000	Chunk Size 0x00	Fragment Delay 0x000A	Chunk Pause 0x0000	Channel 0x06	Data Rate 0x00	Timeout 0x0BB8
Encoded Dst. DID 0xB4BC	Estimated Transfer Time 0x000007D0						

Figure 6.9 – Stream Transfer Setup Message

The data rate is 38,400, which is Data Rate 0. Assume the channel is channel 6. These are both irrelevant for Short transfers. We use the base data rate and the network channel. We'll fill it in anyway.

As with the Block transfer, this packet will be sent to the Master device, which is the recipient, which will either accept the parameters, reject them with a NACK and propose a change to one of the parameters, or reject the transfer completely. We assume that the Master will accept them as is.

The total transfer is as follows: Two route messages will be sent to establish connectivity, then the setup message will be sent, then two seconds of Stream Data packets will be sent, then a final termination message will be sent. No responses are needed for the Stream Data packets since this is a Short transfer.

Action	Packet Type	Start Time (ms)	Transfer Time (ms)	End Time (ms)
Route Packet From 001 to 002	Route Packet	0.00	10.83	10.83
Return Route Packet From 002 to 001	Route Packet	20.83	10.83	31.66
Route Packet From 001 to 002	Route Packet	41.66	10.83	52.49
Return Route Packet From 002 to 001	Route Packet	62.49	10.83	73.32
Block Message Setup Packet From 001 to 002	Block Setup Admin Packet	83.32	8.54	91.86

Response To Setup Message From 002 To 001	Response Packet	101.86	6.25	108.11
Transfer Bytes With Timestamp 0.00 From 002 To 001	Stream Data Packet	118.11	12.92	131.03
Transfer Bytes With Timestamp 22.92 From 002 To 001	Stream Data Packet	141.03	12.92	153.95
Transfer Bytes With Timestamp 45.84 From 002 To 001	Stream Data Packet	163.95	12.92	176.87
Transfer Bytes With Timestamp 68.76 From 002 To 001	Stream Data Packet	186.87	12.92	199.79
.....	.....	.....	.....	.....
More Stream Data Packets Sent Every 22.92 Milliseconds From 002 To 001.				
.....	.....	.....	.....	.....
Transfer Bytes With Timestamp 2000.00 From 002 To 001	Stream Data Packet	2118.11	12.92	2131.03
Termination Message From 002 to 001	Stream Termination Admin Message	2141.03	8.54	2149.57
ACK For Termination Message From 001 To 002	Response Packet	2159.57	6.25	2165.82

Figure 6.10 –Stream Transaction From Start To Finish

7 Long Block And Stream Messages

Long Block And Stream messages are the same as short Block and Stream messages, with some added considerations, some of which have been discussed both in this document and on the **ONE-NET Specification** document. To recap and expand, some of the issues are…

1. Is the network congested?
2. Can / should the devices involved devote themselves exclusively to the pending transfer?
3. Will this prevent other devices from communicating?

4. Can the transfer be paused?

As mentioned, very often Block And Stream messages will be denied permission if the network is in the middle of a key change, in the middle of adding or removing a device, or a sleeping device is about to check in. Beyond this, networks where delayed and missed messages have high negative consequences need to consider that when allowing Long Block and Stream transfers. Long high-priority transfers with short fragment delays, large chunk sizes, and short chunk pauses can prevent many single messages from going through. One or all of these factors may need to be adjusted to accommodate the network. In largely dormant networks where lost messages are not a critical factor, it is generally acceptable to devote all resources to the block or stream transfer without interruption.

Some parameters that must be agreed upon are the following…

1. Priority of the transfer – Low priority transfers are expected to pause whenever single messages need to go through. High priority transfers do not pause.
2. Chunk Size (Block Only) – High Chunk Sizes means that data can be transferred very rapidly. The downside is that there may be RAM limitations on either the destination or source devise that prevent this. In addition, while sending a chunk of data, the network bandwidth may be dominated by the Block Transfer for a long time. Pick a low chunk size if RAM is an issue or if Single Packet latency is an issue.
3. Fragment Delay – Low Fragment delays means that data can be transferred very rapidly. The downside is that the network bandwidth may be dominated by the Block Transfer for a long time. In addition, a low fragment delay can cause extra problems when using Multi-Hop due to a message being repeated colliding with a later data packet. Pick a low fragment delay when not using Multi-Hop and when it is acceptable for the transfer to dominate the bandwidth.
4. Chunk Pause (aka Chunk Delay) – Low Chunk Pauses means that data can be transferred very rapidly. However, it increases the likelihood of missing Single Messages.
5. Channel Changes – To prevent the Long Block or Stream Transfer traffic from interfering with the other network traffic and vice versa, sometimes channels are changed. This results in fewer network collisions. However, it takes the devices largely offline and thus can prevent them from

participating in other communications. In addition, it adds several other messages to the transaction and there is more to keep track of. All devices need to know the correct channel. If the network is communicating at more than one channel, devices which are NOT on the same channel either cannot communicate with each other or must find out what channel is being used and switch back and forth. Simple devices will NOT be able to do this.

6. Data Rate Changes – Transfer times can be vastly improved simply by changing data rates. This can be particularly important with high-density stream messages such as sound messages. Sound messages require rapidly transferring a lot of data. Sending sound data at 38,400 baud, even with a low fragment delay, may not work. The downside is that, one, as with changing channels, changing data rates take devices offline, two, not all devices can communicate at all data rates, and three, higher data rates can significantly reduce range and increase transmission errors. Hence the number of hops required may increase. Multi-Hop communication always slows things down and increases error percentages, so that inefficiency might offset any gain from data rate changes.

There is no one "right" answer to these questions. The right answer will vary by device, network, and circumstances. Each client device has a flags byte set by the master. The master also has this flags byte for itself. There are four pertinent flags for Long transfers. Table 7.1 of the **ONE-NET Specification** document is reprinted here.

Bit	Name / Meaning / Interpretation
7	**ON_JOINED**: True if device is in a network, false otherwise
6	**ON_SEND_TO_MASTER**: True if the device should send status change updates to the master, false otherwise.
5	**ON_REJECT_INVALID_MSG_ID**: True if the device should reject messages that contain invalid Message IDs, false otherwise. See the "**Preventing_Replay_Attacks_And_Other_Security_Considerations_In_ONE-NET**" document for more details.
4	**ON_BS_ELEVATE_DATA_RATE**: True if the device should change elevate data rates for "long" block and stream messages, false otherwise. See the "**Block And Stream Messages in ONE-NET**" document for more details.

3	**ON_BS_CHANGE_CHANNEL**: True if the device should change channels for "long" block and stream messages, false otherwise. See the "**Block And Stream Messages in ONE-NET**" document for more details.
2	**ON_BS_HIGH_PRIORITY**: If true, the device should use high priority queuing for "long" block and stream messages, If false, low priority should be used. See the "**Block And Stream Messages in ONE-NET**" document for more details.
1	**ON_BS_ALLOWED**: If false, "long" block and stream transfers are not allowed at all or this device. See the "**Block And Stream Messages in ONE-NET**" document for more details.
0	Unused

Table 7.1 – ONE-NET Flags Byte

We are concerned with bits 1 through 4.

Bit	Name / Meaning / Interpretation
4	**ON_BS_ELEVATE_DATA_RATE**: True if the device should change elevate data rates for "long" block and stream messages, false otherwise. See the "**Block And Stream Messages in ONE-NET**" document for more details.
3	**ON_BS_CHANGE_CHANNEL**: True if the device should change channels for "long" block and stream messages, false otherwise. See the "**Block And Stream Messages in ONE-NET**" document for more details.
2	**ON_BS_HIGH_PRIORITY**: If true, the device should use high priority queuing for "long" block and stream messages, If false, low priority should be used. See the "**Block And Stream Messages in ONE-NET**" document for more details.
1	**ON_BS_ALLOWED**: If false, "long" block and stream transfers are not allowed at all or this device. See the "**Block And Stream Messages in ONE-NET**" document for more details.

Table 7.2 – ONE-NET Flags Bits

The sending device uses these master-issued flags to determine what the default parameters should be for a Long Block or Stream message. It will also generally have a predetermined preferred chunk size and chunk pause. Default fragment delay values have been assigned to each device by the Master. Bit 1 specifies

whether the device is allowed to participate in Long transfers at all. If it is false, the sending device cannot proceed. In addition, if either the sending device's or receiving device's features or flags do not support the transfer, it cannot proceed.

Note that the application code is informed of all parameters and can also change any parameter.

The sending device establishes a route with two route messages, as described earlier, then builds the administration setup message, then sends it to the recipient. The recipient will check its own features, flags, etc., and check with the application code, then reject, accept, or change the parameters and send back the response. Some very common changes it might make is to change the fragment delay, the chunk pause, the chunk size, or the priority of the transfer. The most common change occurs when the sending device requests a Chunk Size that is larger than the receiving device can handle, particularly due to RAM constraints. It will send back a NACK and specify the correct chunk size. The sending device can repackage the request an send it again.

7.1 Compile-Time Options

Look at both the one_net_port_const.h and config_options.h files. You may possibly have to also look at the one_net_master_port_const.h or one_net_client_port_const.h files as well. There are several constants that are used to help determine the default flags and fragment delays.

In particular, there is a compile-time #define option called BLOCK_STREAM_MASTER_PERMISSION. It is relevant ONLY for clients, and if this is defined, client devices should contact the Master for permission whenever initiating a Long Block or Stream transfer. If not, permission is not required. See Section 8 for more information regarding getting the Master's permission.

The other relevant compile-time options are eight options, four each for the default Master bit flags and the default Client bit flags. Note that these are only relevant for the Master. They correspond to the four bit flags listed above. Client devices get their flags from the Master when joining the network. All of these #define options should be defined in the code as TRUE or FALSE. Again, these are the DEFAULT flag values. See the one_net_master_port_const.h code for the Evaluation Module project for an example.

The following four compile-time flags are relevant to the MASTER device's flags.

1. ONE_NET_MASTER_MASTER_ALLOW_LONG_BLOCK_STREAM

    a. If true, the master can participate in long block and stream transfers. Otherwise, it cannot.

2. ONE_NET_MASTER_MASTER_BLOCK_STREAM_HIGH_PRIORITY
    a. If true, long block and stream transfers involving the master can proceed at high priority. Otherwise, they should occur at low priority.
3. ONE_NET_MASTER_MASTER_ELEVATE DATA_RATE
    a. If true, long block and stream transfers involving the master can proceed at data rates higher than 38,400 baud.
4. ONE_NET_MASTER_MASTER_CHANGE_CHANNEL
    a. If true, long block and stream transfers involving the master can change channels from the base channel.

The following four compile-time flags are relevant to what the master should assign as the DEFAULT client device's flags.

1. ONE_NET_MASTER_CLIENT_ALLOW_LONG_BLOCK_STREAM
    a. If true, the client can participate in long block and stream transfers. Otherwise, it cannot.
2. ONE_NET_MASTER_CLIENT_BLOCK_STREAM_HIGH_PRIORITY
    a. If true, long block and stream transfers involving the client can proceed at high priority. Otherwise, they should occur at low priority.
3. ONE_NET_MASTER_CLIENT_ELEVATE DATA_RATE
    a. If true, long block and stream transfers involving the client can proceed at data rates higher than 38,400 baud.
4. ONE_NET_MASTER_CLIENT_CHANGE_CHANNEL
    a. If true, long block and stream transfers involving the client can change channels from the base channel.

Again, these are the DEFAULT flags. The master can change them at any time. In addition, the master can have different flags for different clients.

Default fragment delays, chunk sizes, and chunk delays can also be defined. See the Evaluation Board project for an example.

7.2 Changing Data Rates And Channels

If data rates and channels are to be changed, all relevant devices (source, destinations, and any Repeaters), after the initial setup message is sent, will be sent a ON_CHANGE_DATA_RATE_CHANNEL administration message to change data rates and channels. The message makeup is described in Section 4.2.9.1.3.2 of the **ONE-NET Specification** document. Re-quoting here…

- *"Payload is one byte for the new channel, one byte for the new data rate, two bytes for the number of milliseconds to wait before changing, two bytes for the number of milliseconds to wait for a message before reverting to the base data rate and channel (6 bytes total)."*

The slight wait is necessary in case an acknowledgement is missed. If the device ACKs the request, then immediately changes data rates, the ACK could be missed and a retry sent, which would be missed. To prevent this, a slight delay is needed. The second time allows the device to return back to the original data rate and channel if communication is lost. When communication is NOT lost, a timer is constantly reset every time the device receives a relevant packet. If that timer expires, a timeout has occurred and the devices should revert. In this way, all devices can efficiently and smoothly change between data rates and channels. The timeout time specified should be long enough to ensure that the sending device can bounce around between data rates and channels sending messages to other devices, including the master, setting up the transfer, without any devices timing out. At the same time, it must be reasonably short enough to allow reasonable detection of a timeout.

Note that the reason that there are TWO route messages instead of one is to test connectivity at both the base data rate and channel and the proposed new data rate and channel. If the route fails at the new data rate and channel, the sending device will need to revert to the base data rate and route.

8 Multi-Hop Block And Stream

If a direct connection between the two end devices cannot be established, Multi-

Hop must be used. The route message test at the start of the message will determine whether Multi-Hop is needed. If it is needed, the Repeater devices will be determined from the route message. Multi-Hop Block and Stream is the same as non-Multi-Hop Block and Stream, only more involved.

8.1 Long Versus Short Transfers And Reserving Repeaters

For Short transfers, Multi-Hop works exactly as it does for Single Messages. Any Repeater in the area will repeat any Multi-Hop message. Whenever Repeaters are involved, fragment delays, timeouts, and chunk pauses need to be chosen more carefully because the probability of collisions goes up significantly. Intelligent guesses based on packet length and transfer time can be made, but most likely the fragment delay will need to be adjusted during the transfer when using Multi-Hop to avoid collisions. The application code should do this by trial and error until it finds the right values. Repeaters do not need to be reserved for Short transfers. For long transfers, the Repeaters involved must be reserved. That means that if data rates an channels are to be changed, the Repeaters must also change data rates and channels. Since the repeaters are needed to transmit the message, data rates and channels must be changed in the correct order. This is solved by intelligently choosing the proper order and delay times when sending the data rate / channel change administration messages. For example, Suppose the following route is established…

001-002-003-004

For all the devices to change channels and retain connectivity, they must do so in the following order (assuming 001 is the sender): 004, then 003 then 002, 001). Fortunately, the core ONE-NET code handles this and hence the application writers need not concern themselves with how this is done. Just note that if several repeaters are needed, the elaborate data changes may take a few seconds.

When using Multi-Hop, the setup message is exactly the same as it is when not using Multi-Hop. The big difference is that the source device is now sending an extra message to each repeater asking that it be taken offline and reserve itself exclusively for use in the Block or Stream transfer. The Block and Stream Setup messages are also sent to each Repeater. Like the Destination Device, each Repeater has a chance to accept, reject, or change parameters. In addition, the Master needs to approve the Repeaters being reserved for the transfer. If the Repeaters are needed elsewhere, then either the Repeaters themselves or the

Master will reject the request.

9 Informing The Master

For transfers NOT involving the Master as either the source device or the recipient device, the Master should be informed of any Long transfers, including any Repeaters involved. ONE-NET does this automatically without interference from the application code. This step is not done if either the source or the recipient is the Master, nor is it done for Short transfers. Some networks may not require this step. If the network does NOT want client-to-client Long transfers to check-in with the Master, the developer should undefined the following compile-time constant on all clients in the client's config_options.h file:

BLOCK_STREAM_REQUEST_MASTER_PERMISSION

If this value is defined, the Master device will be consulted. If it is not, the master will not be consulted.

10 Application-Level Code

Developers implementing Block And Stream must write the following application-level code. See the one_net_port_specific.h, one_net_client_port_specific.h, and one_net_master_port_specific.h files for more details. See the Evaluation Board project code for examples. Note that not all of these functions will need to be written and many implementations will be empty.

1. one_net_client_handle_block_pkt: Called every time a block packet is received. Needed only for client devices.
2. one_net_master_handle_block_pkt: Called every time a block packet is received. Needed only for master devices.
3. one_net_client_handle_stream_pkt: Called every time a stream packet is received. Needed only for client devices implementing stream messaging.
4. one_net_master_handle_stream_pkt: Called every time a stream packet is received. Needed only for master devices implementing stream messaging.

5. one_net_client_handle_bs_ack_nack_response: Called every time an ACK or NACK is received for a Block or Stream Data packet. Needed only for client devices.
6. one_net_master_handle_bs_ack_nack_response: Called every time an ACK or NACK is received for a Block or Stream Data packet. Needed only for master devices.
7. one_net_client_block_chunk_received: Called every time a full chunk of a block transfer is received. Needed only for client devices.
8. one_net_master_block_chunk_received: Called every time a full chunk of a block transfer is received. Needed only for master devices.
9. one_net_client_get_default_block_transfer_values: Called when a block message is initiated to allow the application code to select / change default parameters. Needed only for client devices.
10. one_net_master_get_default_block_transfer_values: Called when a block message is initiated to allow the application code to select / change default parameters. Needed only for master devices.
11. one_net_client_get_default_stream_transfer_values: Called when a stream message is initiated to allow the application code to select / change default parameters. Needed only for client devices implementing stream messaging.
12. one_net_master_get_default_stream_transfer_values: Called when a stream message is initiated to allow the application code to select / change default parameters. Needed only for master devices implementing stream messaging.
13. one_net_client_block_txn_status: Called when a block message is complete. Needed only for client devices.
14. one_net_master_block_txn_status: Called when a block message is complete. Needed only for master .
15. one_net_client_stream_txn_status: Called when a stream message is complete. Needed only for client devices implementing stream messaging.
16. one_net_master_stream_txn_status: Called when a stream message is complete. Needed only for master devices implementing stream messaging.
17. one_net_client_repeater_requested: Called when a client device has been requested as a repeater for a Long transfer. Needed only for client repeater devices.

18. one_net_master_repeater_requested: Called when a client device has been requested as a repeater for a Long transfer. Needed only for master devices.
19. one_net_data_rate_channel_changed: Called when a device has changed channels and / or data rates. Needed only for devices that can change data rates or channels.
20. one_net_get_alternate_channel: Called when an elternate channel is needed for a channel change. Needed only for devices that can change channels.
21. one_net_block_stream_transfer-requested: Called when another device is requesting that a Block or Stream transfer be allowed to commence. Allows the application code to accept, reject, or change any parameters.
22. one_net_block_get_next_payload: Called every time the source in a Block transfer needs to send a Block Data payload.
23. one_net_stream_get_next_payload: Called every time the source in a Stream transfer needs to send a Stream Data payload. Needed only by devices implementing stream messaging.