# Universal Serial Bus Communications Class Subclass Specifications for Host Socket Sharing

## Revision 0.2

## June 22, 2020

# Revision History

Table 1: Revision History

| Version | Date | Comments |
|---|---|---|
| 0.1 | 9/4/2019 | Initial Release |
| 0.2 | 6/22/2020 | Rename to HSS, general fixes |

# Contributors


| | |
|---|---|
| Daniel Berliner | Xaptum |
| David Bild | Xaptum |

# Contents



# List of Figures



# List of Tables

# 1 Introduction

## 1.1 Purpose

The Host Socket Sharing (HSS) subclass is a protocol by which USB devices can efficiently manage and use Berkeley/POSIX-style sockets on the host.

The specification currently defines support for the INET (IPv4) and INET6 (IPv6) protocol families and the TCP and UDP protocols. Future versions may add support for additional families and protocols.

This protocol offers an alternative method for a device to communicate with remote servers. Existing approaches that operate at the L3 layer (IP/USB) or L2 layer (CDC ECM, EEM, NCM) have several disadvantages. In particular, they require the device to be assigned an IP address. Since many networks will assign only a single IP address to a host, this requires implementing some form of network address translation (NAT) on the host. Setting up the NAT requires userspace configuration in most operating systems, so the USB device is no longer truly plug and play.

## 1.2 Scope

This document specifies new device subclasses intended for use with Communication devices, based on the Universal Serial Bus Class Definitions for Communication Devices specification [USBCDC].

The intention of this specification is that all material presented here be upwards-compatible extensions of the [USBCDC] specification. New numeric codes are defined for subclass codes, protocol codes, management elements, and notification elements.

In some cases material from [USBCDC] is repeated for clarity. In such cases, [USBCDC] shall be treated as the controlling document.

In this specification, the word 'shall' or 'must' is used for mandatory requirements, the word 'should' is used to express recommendations and the word 'may' is used for options.

# 2 Overview

Host Socket Sharing (HSS) is a specification for efficient management of sockets on a host by its devices. HSS sends Application Layer traffic and internal socket management commands from the device to the host over USB. This module requires host implementations for each supported Network and Transport layer protocols, but other layers are beyond its scope.

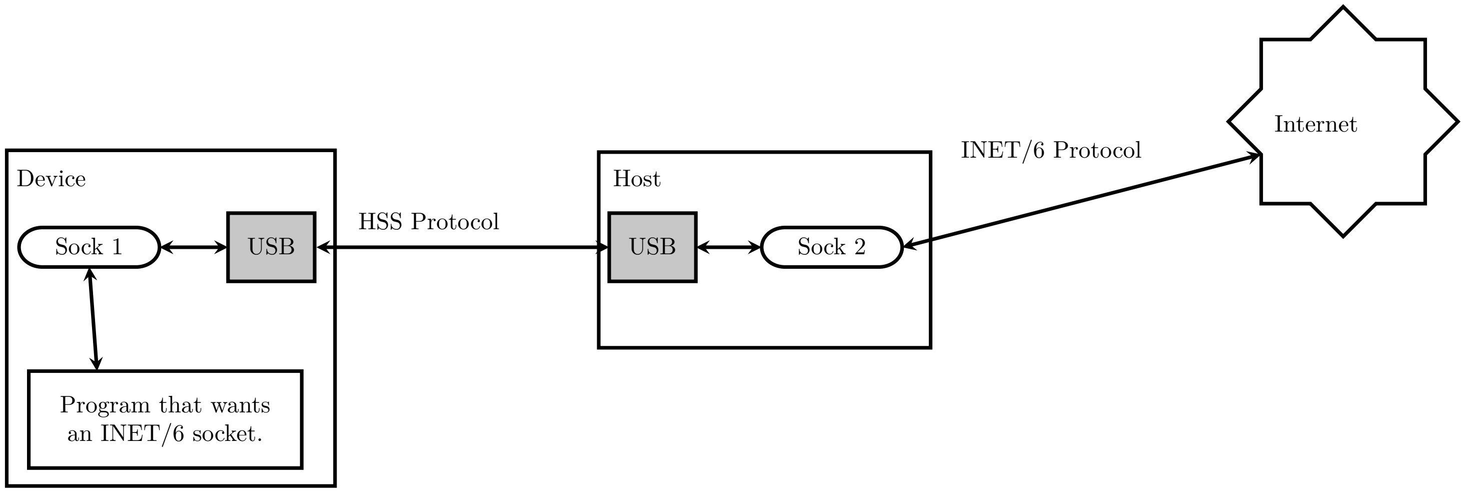
## 2.1 What is Host Socket Sharing (HSS)?

HSS allows a USB connected device to remotely create and control sockets on its host. A device may only control sockets it has created, and it shall expect exclusive control over this socket. Devices cannot control sockets created by other devices on the same host, and the host will not expose a devices socket for use by other applications.

Prior to HSS, connected devices would have to be assigned an IP address requiring the host to bridge the device or use NAT which also requires userspace configurations to the host. HSS allows a host to act on behalf of its device and provide it's network connection in a truly plug and play manner. This implicitly subjects the device to the same firewall rules and external network considerations without specific configuration.

The below figure illustrates a possible use case where a new socket type (Sock 1) is created on the device to interface with HSS. This allows userspace applications to interact with a standard protocol.

Figure 1: HSS Overview.




## 2.2 USB Endpoints

HSS's endpoint requirements consist of a bulk pair (In and Out) and command pair (In and Out). The HSS Device Class uses the standard Endpoint descriptor, as defined in chapter 9 of the USB Specification.
Additionally, HSS requires a pair of bulk-in/bulk-out endpoints and a control-in endpoint for receiving commands.

# 3 Host Socket Sharing (HSS)

The payload of the USB packet contains any combination of a single HSS packet, two or more HSS packets or a split HSS packet. HSS transfers can be split across USB packets but shall not be split across USB transfers.

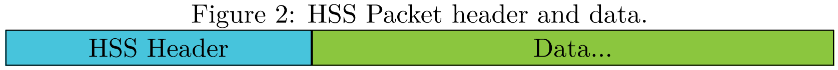
## 3.1 HSS Packet Format

The packet format defines an HSS packet. For information regarding USB packets refer to [USB2.0] specification. Details packets formats can be found in section 8.4 of the [USB2.0] specification.

All HSS packets begin with a one-byte opcode (see Table 3), a one-byte message ID and a one-byte Socket ID.

While represented differently, both types of HSS packets will indicate the length of data being sent and then the data being sent. A packets data segment will always immediately proceed the header and this packet will always be sent in one USB transfer (although USB may split the HSS packet into many USB packets).

Figure 2: HSS Packet header and data.



## 3.2 HSS Error Codes

Table 2: ACK Codes for OPEN

| Value | Name | Description |
|---|---|---|
| 0 | ESUCCESS | Success |
| 1 | EHOSTERR | An error has occurred on the host |
| 2 | EINVAL | Invalid parameters sent |
| 3 | EPROTONOSUPPORT | The protocol type is not supported on the domain. |
| 4 | ECONNREFUSED | No one is listening on remote address |
| 5 | ENETUNREACH | Network is unreachable to the host |
| 6 | ETIMEDOUT | Timeout while host attempted to perform operation. |
| 7 | EMISMATCH | Family for CONNECT to not match socks type on OPEN. |
| 8 | ENOTCONN | An operation has occurred on a non-connected socket which requires a connection. |
| 9 | ENOSOCK | Socket not found on the host |

## 3.3 HSS Packet Types

HSS has two different types of commands: Command and Payload. Command supports quickly sending short messages and Payload supports sending large messages.

### 3.3.1 Common Fields

All HSS packets begin with the following three bytes:

1. **Opcode**: 8-bit unsigned integer identifying what type of packet follows. All Op Codes are defined as either a Command or Data type so this field is sufficient to decide which type of structure to expect.
2. **Message ID**: 8-bit unsigned integer provided to allow the receiver to identify the received message when replying. This is always generated by the sender, and the receiver may only reply using the given ID once. Sending an ACK or REPLY to an unknown Message ID causes undefined behavior.
3. **bSockID**: 8-bit unsigned integer identifying which sock is being sent to. Sock IDs are always created by the host during an OPEN command.

While represented differently, both packets also include a length parameter indicating how much data follows the header. Both length fields are the length of the payload, not including the header.

### 3.3.2 Op Codes

Table 3: OP Codes

| Opcode | Name | Type | Purpose |
|---|---|---|---|
| 0x00 | OPEN | Cmd | Open a socket on the host |
| 0x01 | CONNECT | Cmd | Connect an open socket to a given address |
| 0x02 | SHUTDOWN | Cmd | Indicates that the sender will not send any more traffic. |
| 0x03 | TRANSMIT | Data | Write data to a connected socket |
| 0x04 | ACK | Cmd | Acknowledge a command and indicate success |
| 0x05 | ACKDATA | Data | Similar to ACK but contains data. |
| 0x06 | CLOSE | Cmd | Close the socket. |

### 3.3.3 Command

HSS Command operations relate to opening, closing and managing sockets across USB. These requests are infrequent and short in length so they do not need to be mixed with larger data transmissions that can take a long time to process. Command packets are always sent over the USB Control EP and have a maximum length of 64 bytes including the header.

<table>
<tr><td>0</td><td>8</td><td>16</td><td>24</td><td>31</td></tr>
<tr><td colspan="2">wOPCode</td><td colspan="3">wMsgID</td></tr>
<tr><td colspan="5">dSockID</td></tr>
<tr><td colspan="5">dPayloadLen</td></tr>
<tr><td colspan="5">Data (Up to 52 bytes)...</td></tr>
</table>

### 3.3.4 Data

HSS Data packets are meant for forwarding data across USB bulk endpoints. These requests can contain large amounts of data. These requests are either for sending data transmitted across sockets or acknowledging messages with a response longer than 52 bytes.

<table>
<tr><td>0</td><td>8</td><td>16</td><td>24</td><td>31</td></tr>
<tr><td colspan="2">wOPCode</td><td colspan="3">wMsgID</td></tr>
<tr><td colspan="5">dSockID</td></tr>
<tr><td colspan="5">dPayloadLen</td></tr>
<tr><td colspan="5">Payload Data...</td></tr>
</table>

HSS Packets All values in HSS packets are little endian unless otherwise noted. All IDs and integer values are unsigned unless otherwise noted.

### 3.3.5 HSS Command Packet Formats

#### 3.3.5.1 OPEN

The OPEN command is always initiated by the device to the host. The device will send an OPEN packet with no Sock ID in the header but a 32-bit handle in the payload. The host will reply to all operations on this sock with this handle.

Table 4: Values for Family.

| ID | Name |
|---|---|
| 0x01 | IP |
| 0x02 | IP6 |

Table 5: Values for Protocol.

| ID | Name |
|---|---|
| 0x01 | TCP |
| 0x02 | UDP |

Table 6: Values for Type.

| ID | Name |
|---|---|
| 0x01 | Stream |
| 0x02 | Datagram |

| 0 | 8 | 16 | 24 31 |
|---|---|---|---|
| wOPCode (0x00) | | wMsgID | |
| dSockID | | | |
| dPayloadLen (0x09) | | | |
| dHandle | | | |
| wAddrFamily | | wProtocol | |
| bType | | | |

ACK will return a code with no payload.

Table 7: ACK Codes for OPEN

| **Name** | **Description** |
|---|---|
| ESUCCESS | Success |
| EHOSTERR | An error has occurred on the host |
| EINVAL | Host does not understand protocol or protocol family |
| EPROTONOSUPPORT | The protocol type is not supported on the domain. |

#### 3.3.5.2 CLOSE

When sent from device to host, this command disconnects (if connected) and closes a socket using the ID given during creation. If sent from host to device this will serve as a notification that the socket has been closed by the remote peer.

| 0 | 8 | 16 | 24 31 |
|---|---|---|---|
| wOPCode (0x06) | | wMsgID | |
| dSockID | | | |
| dPayloadLen (0x00) | | | |

Upon completion an ACK command will be sent with no return data.

#### 3.3.5.3 SHUTDOWN

Shutdown indicates that the sender will no longer transmit data to the receiver, but the receiver may still send messages.

| 0 | 8 | 16 | 24 31 |
|---|---|---|---|
| wOPCode (0x02) | | wMsgID | |
| dSockID | | | |
| dPayloadLen (0x00) | | | |

Upon completion an ACK command will be sent with no return data.

#### 3.3.5.4 ACK

Upon completion of a message the receiver will send this back to acknowledge receipt and indicate whether the operation was a success or a failure. Once USB has acknowledged receipt, the sender of an ACK will not wait for further confirmation that the recipient has received the message.

| 0 | 8 | 16 | 24 |
|---|---|---|---|
| wOPCode (0x06) | | wMsgID | |
| dSockID | | | |
| dPayloadLen | | | |
| wOrigOPCode | | Return Code | Return Data... |
| ...Return data (Up to 49 bytes) | | | |

#### 3.3.5.5 CONNECT

Connect tells the host to connect a created socket to a given address. The address information passed will vary by protocol, the host and device should know which structs the other side will send and process those (these are typically the same on both ends).

**IP Connect Packet**

| 0 | 8 | 16 | 24 |
|---|---|---|---|
| wOPCode (0x01) | | wMsgID | |
| dSockID | | | |
| dPayloadLen (0x08) | | | |
| bFamily (0x00) | Reserved | wPort | |
| IP Address (in Network Byte Order[1]) | | | |

**IP6 Connect Packet**

| 0 | 8 | 16 | 24 |
|---|---|---|---|
| wOPCode (0x01) | | wMsgID | |
| dSockID | | | |
| dPayloadLen (0x28) | | | |
| bFamily (0x01) | Reserved | wPort | |
| dFlowInfo | | | |
| dScopeID | | | |
| IP6 Addr In Network Byte Order... | | | |
| ...IP6 Addr In Network Byte Order... | | | |
| ...IP6 Addr In Network Byte Order... | | | |
| ...IP6 Addr In Network Byte Order | | | |

Upon completion an ACK command will be sent with no return data.

Table 8: ACK Codes for CONNECT

| Name | Description |
|---|---|
| ESUCCESS | Success |
| EHOSTERR | An error has occurred on the host |
| ECONNREFUSED | No one is listening on remote address |
| ENETUNREACH | Network is unreachable |
| ETIMEDOUT | Timeout while attempting connection |
| EMISMATCH | Family or protocol for CONNECT to not match socks type on OPEN. |
| ENOSOCK | Sock ID not found on host. |

[1] POSIX.1-2017 expects the address and port to be in network byte order regardless of the host byte order.

### 3.3.6 HSS Data Packet Formats

Data packets contain arbitrary data immediately after the header of whatever length is contained in the headers length field.

Data Length will always be an unsigned 32-bit integer.

#### 3.3.6.1 TRANSMIT

Sends data over the Bulk endpoint containing an arbitrary amount of data for a connected socket.

| 0 | 8 | 16 | 24 | 31 |
|---|---|---|---|---|
| wOPCode (0x03) | | wMsgID | | |
| dSockID | | | | |
| dPayloadLen | | | | |
| Payload data... | | | | |

ACK will return with a 4-byte signed integer. On error the a negative number will be returned. Non-negative return codes indicate a successful transfer while positive return codes also indicate flow indicators (TODO: IMPLEMENT POSITIVE CODE BEHAVIOR).

Table 9: ACK Codes for TRANSMIT

| Name | Description |
|---|---|
| ESUCCESS | Success |
| -EHOSTERR | An error has occurred on the host |
| -ENOTCONN | The socket on the remote side is not open and connected |
| -ENOSOCK | Sock ID not found on host. |

#### 3.3.6.2 REPLY

Similar to ACK but is a DATA type command. This should only be used for commands that can expect more than 60 bytes of data back.

| 0 | 8 | 16 | 24 | 31 |
|---|---|---|---|---|
| wOrigOPCode (0x05) | | wMsgID | | |
| dSockID | | | | |
| dPayloadLen | | | | |
| wOrigOPCode | | Return Code | Payload Data... | |
| Payload data... | | | | |

## 3.4 HSS Packet Ordering

Synchronization must be maintained to ensure that all commands arrive at the destination in the order they were requested. In particular, CLOSE commands shall not be transmitted while there is still unsent data and more data shall not be queued to transmit while a CLOSE is waiting to be sent.

## 3.5 Multiple Devices and Separation

A host may provide sockets to many devices, but a device may never manage a socket which is controlled by another device. It is the responsibility of the host to ensure packets from one device are never executed on sockets belonging to another.

# 4 Packet Processing

Fig A describes the process for the receiver on both ends to assemble and transmit packets from USB to their respective proxies. The Send Command To Proxy procedure can be assumed to be non-blocking, after which the data passed in can be safely freed.

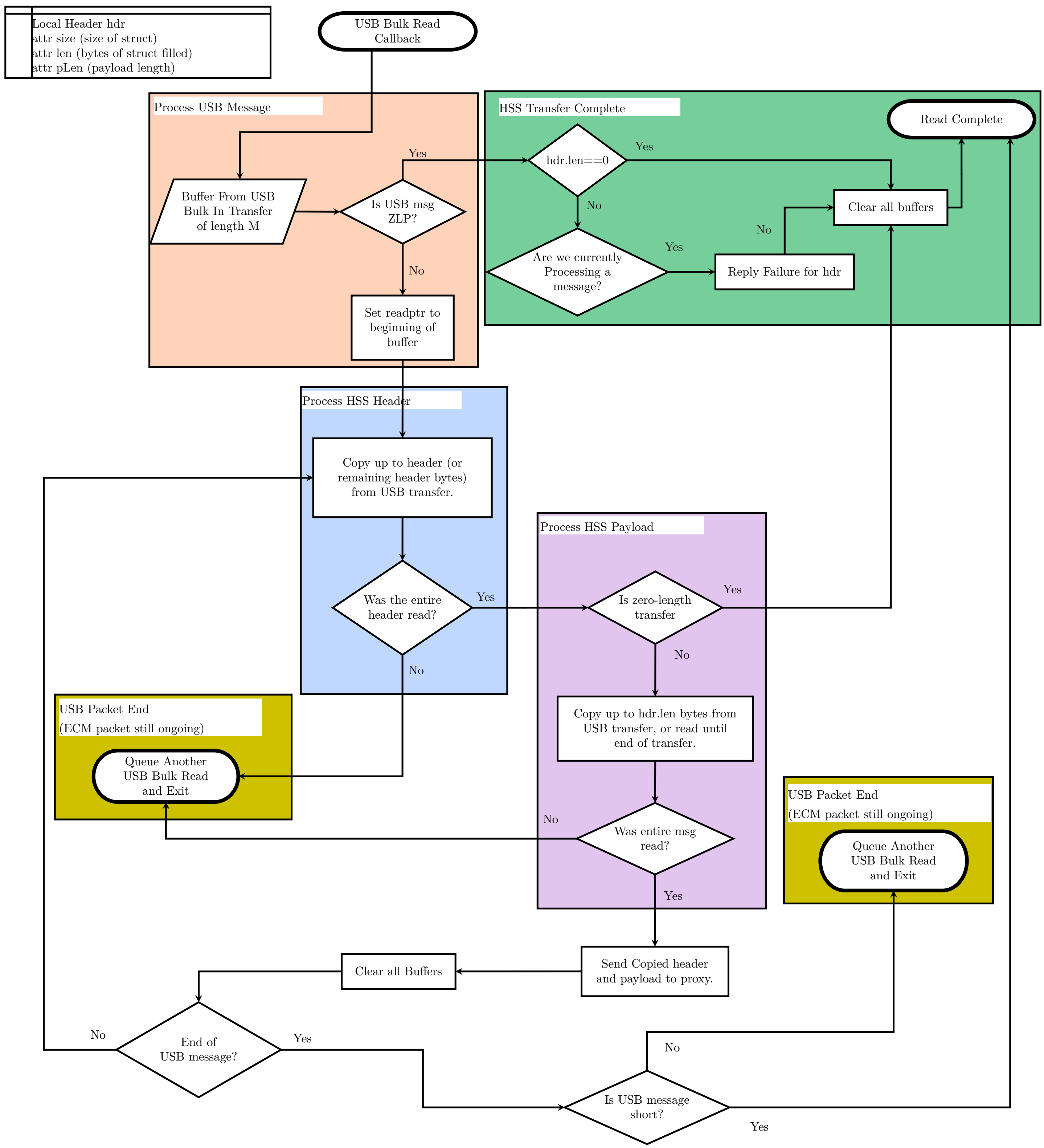
## 4.1 Processing Flow

### 4.1.1 Receiving HSS Packets From USB

HSS transmission packets are sent over USB bulk transfers. HSS transfers may not be split over multiple bulk transfers, but may be split over USB Bulk Packets within a single transfers. Many HSS transmission packets may be sent in one bulk transfer. This means that HSS packet headers may also be split between HSS packets.

HSS transfers will end with one of a short USB packet, zero-length packet over USB (if supported) or a zero-length HSS transfer. Sending a zero-length USB packet when the receiver is expecting the continuation of a HSS packet causes undefined behavior.

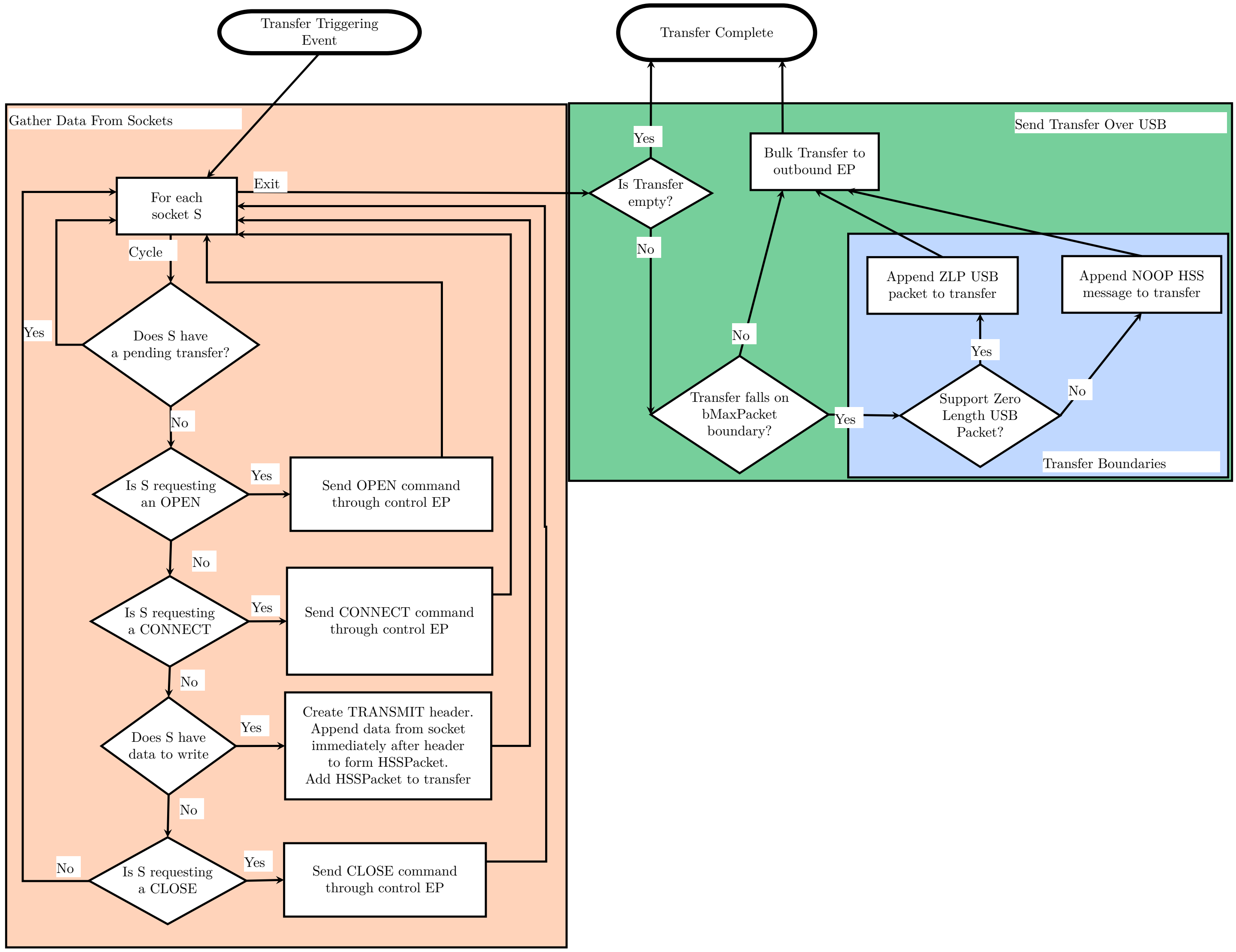
Figure 3: HSS Packet receive Flow.




#### 4.1.2 Transmitting Data

The figure below illustrates how the host and device should send messages to each other.

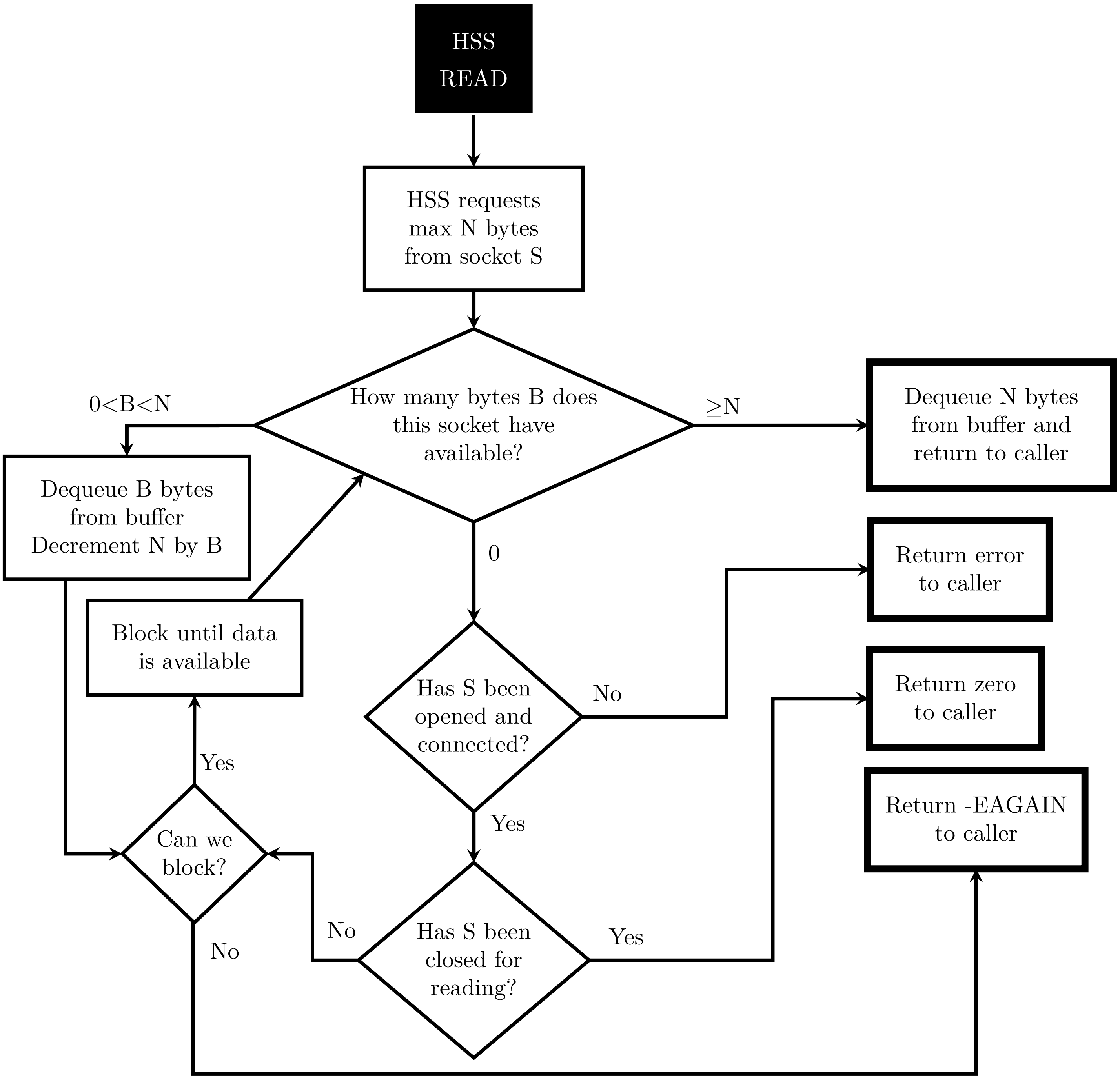
Figure 4: Sending To Peer.




### 4.1.3 Reading Transmitted Data

The figure below illustrates how either the host or device should its read functions. Read can only work with data that has already been transmitted, it does not ask for data from its peer because any data that can be sent will be sent as soon as possible.

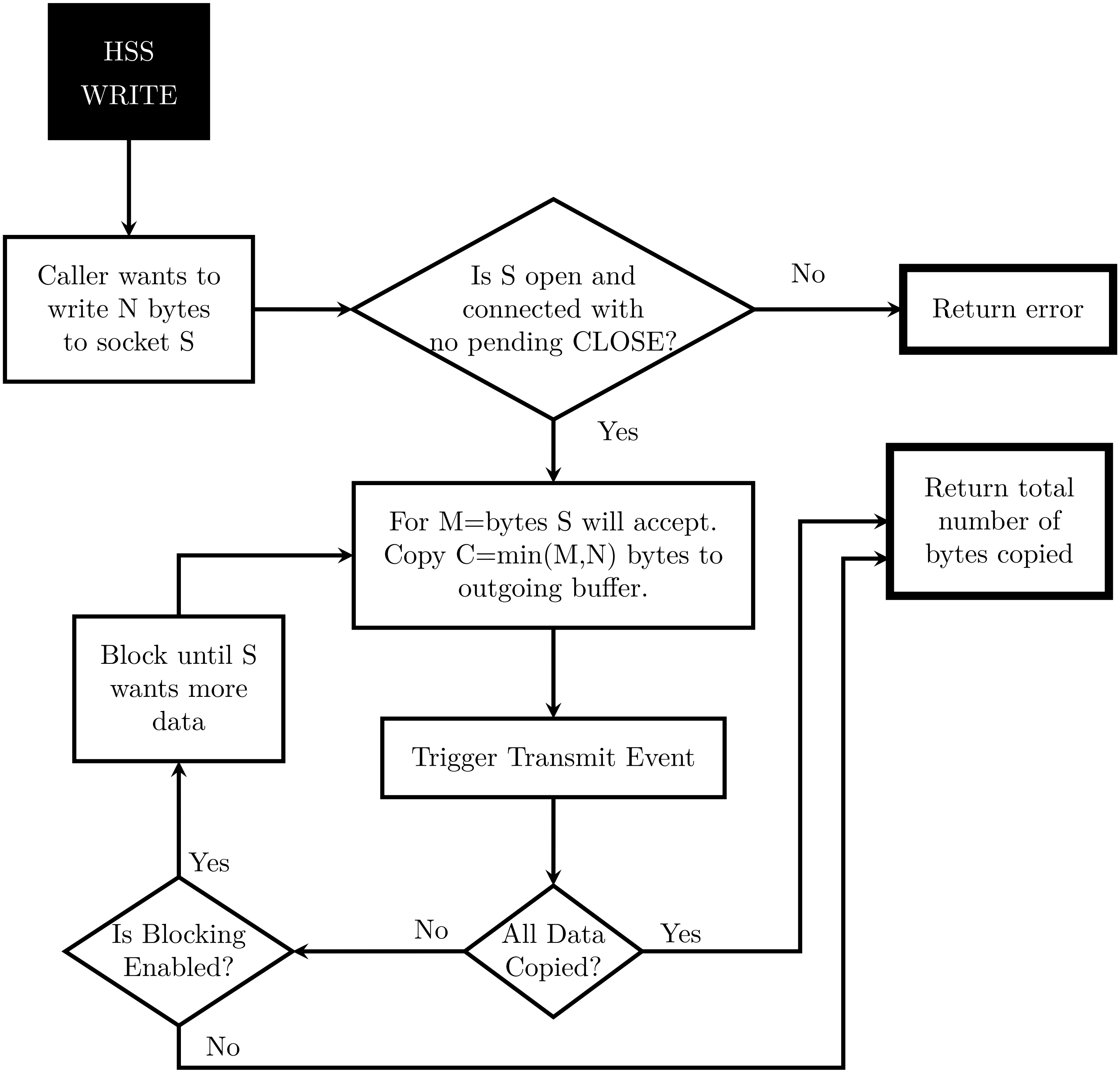
Figure 5: Reading Transmitted Data.




#### 4.1.4 Writing Data

The figure below illustrates how either the host or device should its handle socket write calls.

Figure 6: Writing Data to HSS.




#### 4.1.5 Command Flow

Commands involve controlling sockets and sending ACKs for commands.

Open and Connect require confirmation from the host before the device can change its behavior. Disconnect and Close take effect regardless of acknowledgment, subsequent writes on a closed channel will return an error.

Figure 7: Command Flow.