SAE INTERNATIONAL

AEROSPACE STANDARD

AS5669 REV. A

Issued 2007-12
Revised 2009-02
Reaffirmed 2014-08

Superseding AS5669

(R) JAUS / SDP Transport Specification

RATIONALE

AS5669A has been reaffirmed to comply with the SAE five-year review policy.

FOREWORD

This document, the JAUS / SDP Transport Specification, delineates the formats and protocols employed for the transport of messages between compliant entities for all supported link-layer protocols and media.

This document may be read in conjunction with AIR5645 (JAUS Transport Considerations), which provides reference and background information pertaining to the transport of JAUS messages between compliant entities, including characteristics of unmanned systems, JAUS and various communication media that are important for the design and implementation of efficient transports. Those interested in the transport of messages across currently supported media should use this Transport Specification [AS5669] document; those interested in the transport of messages across new (currently unsupported) media should refer to the JAUS Transport Considerations document.

This is a living document: as requirements are identified for the support of new transport media, those transports will be specified in sections added in subsequent revisions of this document. Likewise, new functionalities may provoke additional revisions of this document.

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: **http://www.sae.org**

SAE values your input. To provide feedback on this Technical Report, please visit
http://www.sae.org/technical/standards/AS5669A

TABLE OF CONTENTS

1.1 Purpose 4
1.2 Compliance 4
1.3 Document Organization 4
1.4 Field of Application 4
2. REFERENCES 5
2.1 Applicable Documents 5
2.2 Other Applicable References 6
2.3 Acronyms 6
3. CONTEXT 9
3.1 Required Capabilities 9
4. GENERAL TRANSPORT HEADER PAYLOAD WRAPPER 10
5. TRANSPORT STANDARDS: COMMON CHARACTERISTICS 13
5.1 Header Compression Techniques 14
5.1.1 Header Compression 14
5.1.2 Header Compression: An Example 16
5.1.3 Header Compression and Broadcast 27
5.1.4 Additional LSB Techniques 28
5.1.5 Header Compression: Summary 28
5.2 Packing Multiple Application Messages Per Transport Packet 28
5.3 Multi-Part Messaging: Multiple Transport Packets Per Application Message 28
5.4 Maximum Packet Sizes 29
5.5 Broadcast Semantics 29
5.6 Discovery 30
6. JAUS OVER IP 30
6.1 JUDP : JAUS Over UDP 30
6.1.1 Characteristics of JUDP 30
6.1.2 Application Domains 30
6.1.3 Communications Environments 31
6.1.4 Description 31
6.1.5 Implementation of Broadcast Semantics 32
6.1.6 Encapsulation 33
6.1.7 Maximum Packet Size (MPS) 35
6.1.8 Message Prioritization 36
6.1.9 Compression 36
6.1.10 Integrity Mechanisms 36
6.1.11 Configuration 36
6.2 JTCP : JAUS Over TCP 37
6.2.1 Characteristics of JTCP 37
6.2.2 Application Domains 37
6.2.3 Communications Environment 38
6.2.4 Description 38
6.2.5 Implementation of Broadcast Semantics 39
6.2.6 Encapsulation 39
6.2.7 Maximum Packet Size (MPS) 40
6.2.8 Message Prioritization 40
6.2.9 Compression 40
6.2.10 Integrity Mechanisms 41
6.2.11 Configuration 41
6.3 Use of Non-Standard Ports in JUDP & JTCP 41
6.3.1 Messaging to/from Non-Standard Ports 41
6.3.2 Broadcast Propagation 41
6.4 IP Infrastructure 42
7. JSERIAL: JAUS OVER SERIAL 42
7.1 Characteristics of JSerial 42
7.2 Application Domains 42
7.3 Communications Environments 42

7.4 Description 42
7.4.1 JAUS Serial Packet Structure 43
7.4.2 JSerial Transport Header Fields 44
7.4.3 Encoding JSerial Byte Streams 46
7.5 Implementation of Broadcast Semantics 47
7.6 Encapsulation 47
7.7 Maximum Packet Size (MPS) 47
7.7.1 Best Practices Guidance: Actual Packet Sizes on JSerial Links 47
7.8 Message Prioritization 47
7.9 Compression 48
7.10 Integrity Mechanisms 48
7.11 Configuration 48
7.11.1 JSerial Addresses 48
7.11.2 Serial Link 48
7.12 Message Numbering 48
8. DEVELOPING ADDITIONAL TRANSPORT STANDARDS FOR OTHER COMMUNICATIONS MEDIA 48
9. NOTES 48

APPENDIX A 49

FIGURE 1 – JAUS / SDP TRANSPORT CONTEXT 9
FIGURE 2 - GENERAL TRANSPORT HEADER FORMAT 10
FIGURE 3 – TRANSPORT HEADER COMPRESSION - USUAL FIELDS 15
FIGURE 4 - INITIAL ENVIRONMENT AND HEADER COMPRESSION TABLES 17
FIGURE 5 - INITIAL (UNCOMPRESSED) MESSAGE SENT FROM OCU TO ROBOT 17
FIGURE 6 - BEGINNING OF HEADER COMPRESSION ENTRY ON ROBOT (RECEIVER) SIDE 18
FIGURE 7 - ROBOT SENDS RESPONSE MESSAGE AGREEING TO HEADER COMPRESSION 19
FIGURE 8 - OCU RECEIVES MESSAGE AGREEING TO HEADER COMPRESSION FROM ROBOT 20
FIGURE 9 - COMPRESSED MESSAGE SENT FROM OCU TO ROBOT 21
FIGURE 10 - UNCOMPRESS OF OCU→ROBOT MESSAGE 22
FIGURE 11 - HEADER COMPRESSION TABLES FOR ROBOT→OCU AND OCU→ROBOT 23
FIGURE 12 - ROBOT EXPERIENCES A FAULT AND REBOOTS, LOSING HEADER COMPRESSION DATA 24
FIGURE 13 - ROBOT RECEIVES COMPRESSED MESSAGE WITHOUT CORRESPONDING ENTRY 25
FIGURE 14 - ROBOT SENDS ZERO-BYTES-REMEMBERED RESPONSE 26
FIGURE 15 - OCU BEGINS REINITIALIZATION OF HEADER COMPRESSION PROTOCOL 27
FIGURE 16 - MULTIPLE MESSAGES IN A SINGLE UDP PACKET 32
FIGURE 17 - JUDP TRANSPORT PROTOCOL STACK OVER ETHERNET 34
FIGURE 18 - JUDP TRANSPORT PROTOCOL STACK OVER PPP 35
FIGURE 19 - JTCP STREAM FORMAT 38
FIGURE 20 - JTCP TRANSPORT PROTOCOL STACK OVER ETHERNET 39
FIGURE 21 - JTCP TRANSPORT PROTOCOL STACK OVER PPP 40
FIGURE 22 - JSERIAL PACKET STRUCTURE WITH EXPLICIT ADDRESSING 43
FIGURE 23 - JSERIAL PACKET STRUCTURE WITHOUT TRANSPORT EXPLICIT ADDRESSING 44
FIGURE 24 - MULTIPLE MESSAGES IN A SINGLE JSERIAL PACKET 44

1. SCOPE

This SAE Aerospace Standard (AS) specifies a data communications layer for the transport of messages defined by the Joint Architecture for Unmanned Systems (JAUS) or other Software Defined Protocols (SDP). This Transport Specification defines the formats and protocols used for communication between compliant entities for all supported link-layer protocols and media. Although JAUS is the SDP used as the example implemented throughout this document, AS5669 can be used for any SDP that meets the required capabilities.

A Software Defined Protocol is defined as an application data interface for communicating between software elements. The SDP is agnostic of the underlying communications protocol and in fact communicates in much the same manner regardless if the communicating entities are collocated in the same memory space or separated by a satellite link.

1.1 Purpose

The purpose of this document is to facilitate interoperation of systems by standardization of the Transport Layer for JAUS messages or other SDP.

1.2 Compliance

The JAUS / SDP Transport Specification must support compliance assessment. To do so, this specification must be sufficiently precise to enable the “compliant”/”not compliant” distinction to be made at the Transport Layer; it should also enable the determination of this same distinction at the Application Layer.

The JAUS Transport Considerations report [AIR5645] outlines compliance considerations in more detail – we only summarize the resultant Transport Rules here for usage with JAUS and other SDP payloads:

1. Transport shall not modify the content or format of any transported payload.

2. All Transport packets shall explicitly state the version of Transport.

These considerations are supported by the standards defined in this document.

1.3 Document Organization

The Transport Specification defines a family of transports for the conveyance of SDP messaging among entities compliant with the specific SDP such as JAUS. It details message formats, protocols, typical operating environments, and best practices.

The Transport Considerations Report [AIR5645] provides background information and delineates criteria for the design of transports for JAUS messaging across various media and environments. In this respect, the Transport Considerations Report serves as a meta-specification. The Transport Considerations Report explores the impact on JAUS messaging of each of the following:

- Characteristics of communications media.
- Characteristics of JAUS messaging.
- Requirements/constraints of unmanned systems operations.

1.4 Field of Application

The Field of Application for the Transport Specification is data communications between nodes on a computer network.

2. REFERENCES

2.1 Applicable Documents

[AIR5645] "JAUS Transport Considerations", Version 1.0, SAE, 2007.

[X3.28] American National Standards Institute, "Procedures for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links," ANSI X3.28-1976, December 1975.

[RFC1661] Simpson, W., "The Point-to-Point Protocol (PPP)," RFC-1661, Daydreamer, July 1994.

[RFC1662] Simpson, W., "PPP in HDLC-like Framing," RFC-1662, Daydreamer, July 1994.

[RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)," RFC-1994, DayDreamer, August 1996.

[RFC2284] Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)," RFC-2284, Merit Network, Inc, March 1998.

[RFC1055] Romkey, J., "A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP," RFC-1055, June 1988.

[RFC914] Farber, David et al, "A Thinwire Protocol for connecting personal computers to the INTERNET," RFC-914, September 1984.

[RFC768] Postel, J., "User Datagram Protocol," RFC-768, 28 August 1980.

[RFC793] Information Sciences Institute, "Transmission Control Protocol," RFC-793, September 1981.

[RFC794] Cerf, V., "Pre-emption," RFC-794, September 1981.

[RFC813] Clark, David D., "Window and acknowledgment strategy in TCP," RFC-813, July 1982.

[X.25] International Telecommunication Union, "Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DTE)," ITU-T Recommendation X.25, October 1996.

[TECHDEV] National Academy of Sciences, "Technology Development for Army Unmanned Ground Vehicles," National Academies Press, Washington, 2003.

[FLET82] Fletcher, John G., "An Arithmetic Checksum for Serial Transmissions," IEEE Transactions on Communications, January, 1982, IEEE, New York.

[KODIS92] Kodis, John, "Fletcher's Checksum," Dr. Dobb's Journal, May 1992.

[KOOP04] Koopman, Philip and Tridib Chakravarty, "Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks," International Conference on Dependable Systems and Networks (DSN-2004).

[SAXENA90] Saxena, Nirmal R. and Edward J. McCluskey, "Analysis of Checksums, Extended-Precision Checksums and Cyclic Redundancy Checks," IEEE Transactions on Computers, vol. 39 no.7, July 1990, IEEE, New York.

[PEREZ83] Perez, Aram, "Byte-Wise CRC Calculations," IEEE Micro, June 1983, IEEE, New York.

[WILLIA93] Williams, Ross N., "A Painless Guide to CRC Error Detection Algorithms," Rocksoft, Adelaide, Australia, August 1993 [available via ftp: ftp.adelaide.edu.au/pub/rocksoft/crc_v3.txt]

[RAMABA88] Ramabadran, Tenkasi V. and Sunil S. Gaitonde, "A Tutorial on CRC Computations," IEEE Micro, August 1988, IEEE, New York.

[RFC1950] RFC-1950 ZLIB 3.3 Specification. Defines a lossless compressed data format. The format currently uses the DEFLATE compression method.

[RFC1951] RFC-1951 DEFLATE 1.3 Specification Defines a lossless compressed data format that compresses data using a combination of the LZ77 algorithm and Huffman coding, with efficiency comparable to the best currently available general-purpose compression methods.

[RFC1952] RFC-1952 GZIP 4.3 Specification Defines a lossless compressed data format that is compatible with the widely used GZIP utility The format currently uses the DEFLATE compression method.

[RFC3095] Bormann, C. (ed.), "RFC-3095 - Robust Header Compression (RoHC): Framework and Four Profiles," 2001.

[RFC1144] Jacobson, V., "RFC-1144 - Compressing TCP/IP Headers for Low-Speed Serial Links," February 1990.

[SUTT91] Sutterfield, Robert A., "Low-Cost IP Connectivity," Sun User Group, San Jose, CA, 9 December 1991.

[RFC2365] Meyer, D., “RFC-2365 – Administratively Scoped IP Multicast,” University of Oregon, July 1998.

[RFC2608] Guttman, E., et al, “RFC-2608 – Service Location Protocol, Version 2,” Network Working Group, IETF, June 1999.

2.2 Other Applicable References

[COMER96] Comer, Douglas, "Internetworking with TCP/IP,” Volume 1, Prentice-Hall, New York, 1996

[TANEN96] Tanenbaum, Andrew S., "Computer Networks," Third Edition, Prentice-Hall, New York, 1996

[STEVENS93] Stevens, W. Richard, "TCP/IP Illustrated," Volume 1, Addison Wesley, New York, 1993.

[DOUGLA99] Douglass, Bruce Powel, "Custom Embedded Communications Protocols," (Whitepaper), I-Logix, Inc., Andover, MA, 1999.

[CHESH05] Cheshire, Stuart & Steinberg, Daniel, "Zero Configuration Networking: The Definitive Guide," O’Reilly Media, Inc. 2005.

2.3 Acronyms

ACK — Acknowledgment (positive)
Generally, a byte or message sent to acknowledge successful receipt of a packet or message. Specifically, the ASCII control character octet of value 0x06.

ARQ — Automatic Repeat reQuest
A transport protocol is said to be an ARQ protocol if it uses positive and negative acknowledgments and sender timeouts, and further employs packet replacement on request to overcome packet loss or corruption over noisy or unreliable channels.

CRC — Cyclic Redundancy Check
An error check algorithm based on polynomial division over a binary field.

CRC-CCITT — A specific CRC, defined by its polynomial, initial conditions for accumulation, and terminal conditions, as defined by ITU Recommendation X.25 (which was once CCITT recommendation X.25).

DLE	Data Link Escape An ASCII control character, an octet of value 0x10, this character is used in the implementation of data transparent protocols such as those consistent with [X3.28].
EMI	Electromagnetic Interference Interference by electromagnetic signals that can cause reduced data integrity and increased error rates on communication channels.
HMI	Human-Machine Interface See “OCU – Operator Control Unit”
IANA	Internet Assigned Numbers Authority The IANA is responsible for assignment of IP addresses, top level domains and Internet protocol code point allocations (such as port assignments). The IANA assignment data is maintained on the IANA website, http://www.iana.org.
IETF	Internet Engineering Task Force The engineering body responsible for technical specifications, definitions and direction for the continued development of internet technologies.
IGMP	Internet Group Management Protocol A communications protocol used to manage the membership of multicast groups in IP networks.
IP	Internet Protocol The internet layer of the TCP/IP protocol suite stack defines the packet format for message packets to be sent across an internet, and the protocol delivering those packets to their intended destination. The protocol is Internet Protocol (IP); the packet defined is the IP datagram.
IPS	Internet Protocol Suite The protocol suite defining network communications using the TCP/IP family of protocols.
ISO	International Organization for Standardization A non-governmental organization that leverages the activities of the national standards organizations of 146 countries, ISO is the largest developer of technical standards in the world.
JAUS	Joint Architecture for Unmanned Systems An architecture for use in the research, development, design, acquisition and deployment of Unmanned Systems
JTCP	JAUS over TCP The standard for the transmission of JAUS messages over TCP communications links, as defined in Section 6.2 of this document.
JUDP	JAUS over UDP The standard for the transmission of JAUS messages over UDP communications links, as defined in 6.1 of this document.
JSerial	JAUS over Serial The standard for the transmission of JAUS messages over serial communications links, as defined in Section 7 of this document.
MSN	Message Sequence Number A unique identifying “serial number” assigned to packets as transmitted; the MSN is typically used in the detection of missed messages, in providing assurance of correct sequence of delivery, and in the requesting of retries when multiple outstanding messages are supported.

MTU	Maximum Transfer Unit The Maximum Transfer Unit (MTU) is a term for the size of the largest datagram that can be passed by a layer of a communications protocol.
NAK	Acknowledgment (negative) Generally, a byte or message sent to indicate unsuccessful receipt or non-receipt or a packet or message. Specifically, the ASCII control character octet of value 0x15.
OCU	Operator Control Unit A device by means of which a human operator may control an Unmanned System.
OSI	Open Systems Interconnect A data communications model developed by ISO to assure communications interoperability across disparate systems.
PDU	Protocol Data Unit An application message (for the purposes of this specification, this will be assumed to be a JAUS message) being propagated down the protocol stack on the sender side, or up the protocol stack on the receiver side.
PPP	Point-to-Point Protocol An encapsulation protocol for sending IP datagrams across serial communications links, PPP also incorporates strong blockchecks, control protocols and a high degree of configurability. [RFC1661]
RA	Reference Architecture (JAUS) The Reference Architecture is the technical specification used to implement unmanned systems in compliance with the Joint Architecture for Unmanned Systems (JAUS).
RF	Radio Frequency Electromagnetic energy or signaling based thereon whose frequency is normally associated with radio wave propagation.
SLIP	Serial Line Internet Protocol An encapsulation protocol for sending IP datagrams across serial communications links. [RFC1055]
SDP	Software Defined Protocol A communications protocol that is defined by implementation of it's lexicons in software. It is assumed to be an addressed entity, message based protocol
TCP	Transmission Control Protocol A reliable, connection-oriented message delivery protocol defined by [RFC793] and related IETF documents.
UMS	Unmanned Systems.
UAV	An unmanned aerial vehicle; may be teleoperated or autonomous.
UGV	An unmanned ground vehicle; may be teleoperated or autonomous.
USV	An unmanned surface-of-water vehicle; may be teleoperated or autonomous.
UUV	An unmanned undersea vehicle; may be teleoperated or autonomous.
UDP	User Datagram Protocol An unreliable best-effort connectionless message delivery protocol defined by [RFC768] and related IETF documents.

3. CONTEXT

The context for the JAUS / SDP Transport Specification is network data communications. The JAUS / SDP Transport Layer provides the interface between the SDP Application Layer and the underlying communications media.

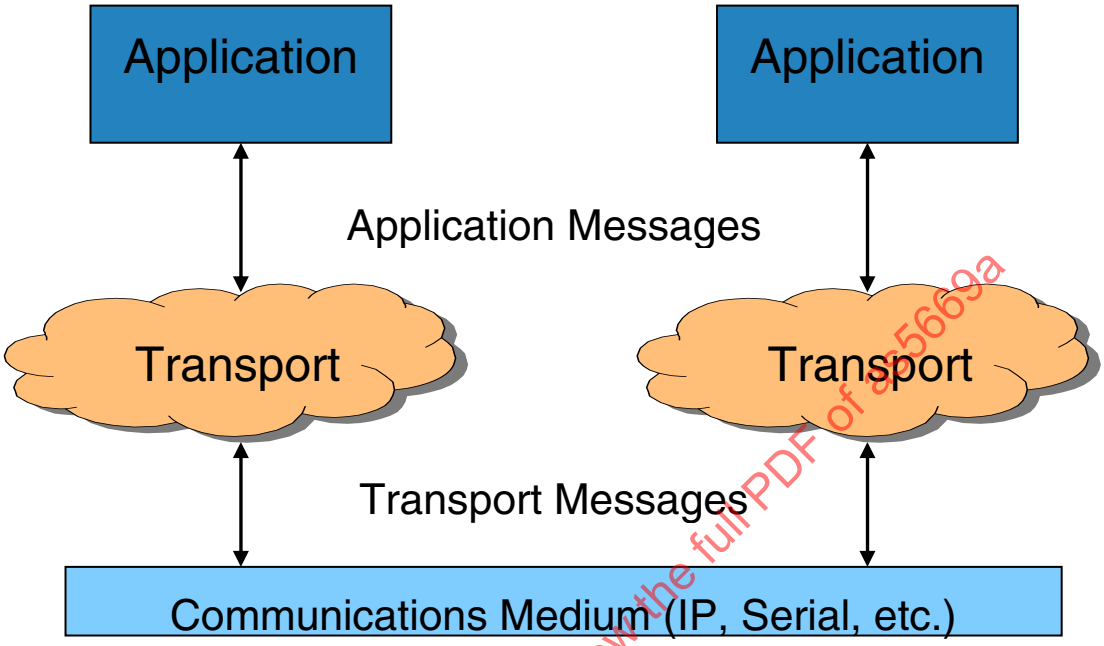


FIGURE 1 – JAUS / SDP TRANSPORT CONTEXT

3.1 Required Capabilities

In order to fulfill its role, the SDP Transport Specification must support the following capabilities:

- Packet transmission, packet reception and received packet handling by message priority.
- Corruptions checks on packet data, with transport-specific NAK and retry mechanisms where required.
- Broadcast semantics for communicating with all entities on the wire conform to the particular SDP. Broadcast semantics need only be supported in transport implementations that contain no retry semantics.
- Data-reduction and data-compression for reducing required bandwidth in reduced bandwidth environments.[1]
- A process to discover the addresses of other AS5669 compliant entities on the wire using the same SDP.

[1] Please refer to 5.1 of this document, as well as the Data Compression section of AIR5645. Data reduction and compression necessary for operation within severe transport bandwidth constraints may be performed on a message-by-message basis by the specific transport implementation (i.e., basic generic compression techniques); compression specific to the nature of the application data stream is best performed at the application level (i.e., JPEG compression of image data, etc.).

4. GENERAL TRANSPORT HEADER PAYLOAD WRAPPER

The specification of the General Transport Header is detailed below. This header serves as a wrapper around every application-layer payload that assists in the routing, verification and delivery of the message to the desired entity. This header and related field descriptions are derived from the JAUS Reference Architecture, Version 3.3, Part 2, Section 3 and the previous revision of this document. The basic format of the message header is shown in Figure 2. Details for formats and semantics of the various fields that comprise this header follow.

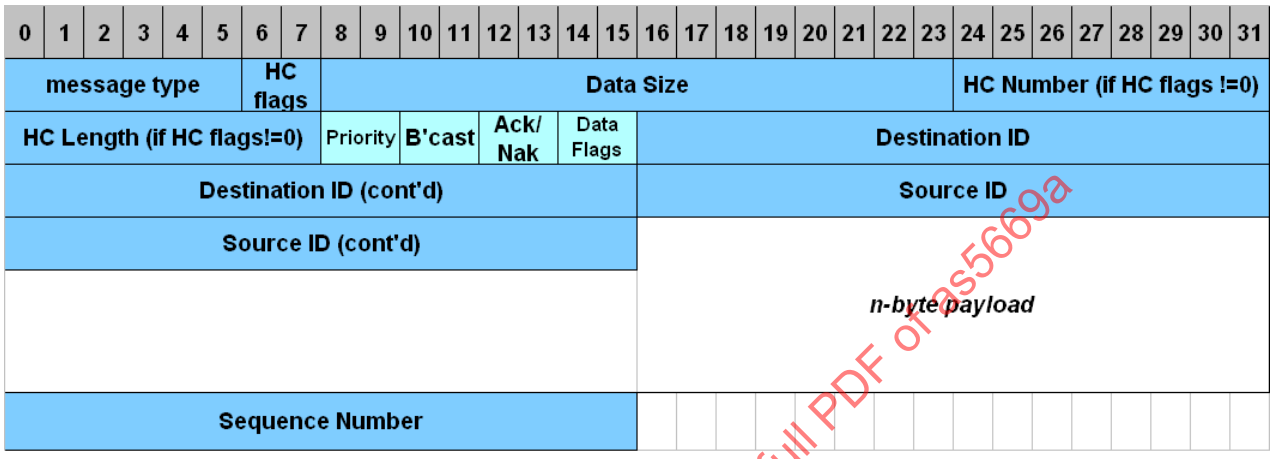


FIGURE 2 - GENERAL TRANSPORT HEADER FORMAT

The contents of the fields required for proper transport in this common transport header follow certain guidelines. Valid ranges, proper combinations for bit mapped fields and other field details are supplied below.

Message Type

The type of message that is sent through the transport layer. The message type refers to the tranport type of message NOT the message content of the payload. The message type allows for multiple Software Defined Protocols to use a common transport mechanism. This is currently set to zero but will be enumerated in future revisions of AS5669

- Initial value = 0
- Values 1 through 32 are reserved for future use.

Header Compression (HC) Flags

The Header Compression Flags field ("*HC Flags*") indicates the type of message for the header compression protocol. While Header Compression is discussed in detail in Section 5, the meaning of the Flags value is summarized here:

0. Zero: No header compression is used in this message.

1. One: The sender of this message is requesting that the receiver engage in header compression.

2. Two: The meaning of a message with the "*HC Flags*" field set to 2 depends upon the contents of the "*HC Length*" field.

3. Three: The sender is sending a message containing compressed data. The "*HC Header Number*" and "*HC Length*" fields are present in this message.

If the HC Flags field is zero, the 'HC Number' and 'HC Length' fields shall be removed from the General Transport Header.

Data Size

This is the size of the entire message including the General Transport Header. It has a minimum value of 14 decimal and a maximum value of 65535. Note, however, that each supported medium may further restrict this maximum value as described in the Maximum Packet Size sections. When Header Compression is used, the Data Size is the size of the compressed message. The packet length shall be presented formatted in Little-Endian format.

Header Compression Header Number Field

The Header Compression Header Number Field ("*HC Header Num*") is used to enumerate the headers under compression – see Section 5. This field supports the full range of unsigned values (0…255).

Header Compression Length Field

The Header Compression Length Field ("*HC Length*") has semantics defined by the value of the Header Compression Flags – see Section 5. This field supports the full range of unsigned values (0…255).

Priority

Although the handling of Priority is implementation dependent, multiple priority levels are assigned.

- The Priority field shown in Figure 2 is used to set the message priority. A two bit field supports the following four values:
    - Low Priority Message (Decimal value 0)
    - Standard Priority Message (Decimal value 1)
    - High Priority Message (Decimal value 2)
    - Safety Critical Priority Message (Decimal value 3)
- *Safety critical messages shall have priority over all other messages*.

Broadcast Flags

The Broadcast Flags are used to mark messages that are intended for multiple destinations. At present, three values are supported.

- No Broadcast (Decimal value 0): The message is intended for a single destination.
- Local Broadcast (Decimal value 1): The message is intended for the local destinations only.
- Global Broadcast (Decimal value 2): The message is intended for all available destinations.

Note that the concept of a local destination is domain-dependent, and therefore user-definable and beyond the scope of this Specification. For example, locally scoped broadcasts may be limited to the sender's subnet for IP-based networks, or all computing nodes sharing the same backplane.

Additional information on broadcast symantics is available in 5.5, 6.1.5, 6.2.5, and 7.5

ACK/NAK Flags

Ack/Nak bits shown in Figure 2 are used to set the message Acknowledge/Negative Acknowledge (ACK/NAK) behavior. ACK/NAK bit usage is defined as follows:

- Message Originator (0 or 1)
  - ACK/NAK = 0 No response required
  - ACK/NAK = 1 Response required
- Message Responder (2 or 3)
  - ACK/NAK = 2 Message negative acknowledge
  - ACK/NAK = 3 Message acknowledged OK

- The response message shall consist of the original message header, with destination and source IDs swapped. The message data shall not be returned and the data size shall be reset to the size of the header only. The returned sequence number shall match that of the original message.

- If a message is received with an ACK bit set, and if the receiver of that message does not know where to find that message's destination address, then Transport at the receiver will reply with a NAK message using the destination address.

- If the original sender receives a NAK message, or fails to receive an ACK in a timely fashion, the sender shall retransmit the original message. The number of transmission attempts and the time interval between attempts shall be set to a value representative of the priority, authority, urgency and expected latency within the system performance requirements.

Note that acknowledgement represents only successful delivery of a message to the destination. The receiver may still choose to ignore or reject the message, depending on the specific application.

Data Flags

Messages larger than Maximum Packet Size defined for each medium are considered large data sets. The data flag bit field is used to parse large data sets into smaller messages while enabling re-assembly by the receiver.

Bit Value	Description
00	Only data packet in single-packet stream
01	First data packet in multi-packet stream
10	Normal (middle) data packet
11	Last data packet in stream

The sender of a multi-packet data stream may require acknowledgement or not as best suits the implementation. ACK/NAK rules will be applied in either case. The following rules shall be enforced when sending multi-packet messages:

- The first message in the stream shall set the Data Flag Bit field = 1 decimal.
- The last message in the stream shall set the Data Flag Bit field = 3 decimal.
- The Sequence Number field shall identify each message. Each successive message of a multi-packet data stream shall increment the Sequence Number by 1.
- Each message between the first and last message shall set the Data Flag Bit field = 2 decimal. The packet shall be retransmitted when ACK was required, but NAK was returned. Retransmission will contain the same sequence number and payload.

Destination ID

This 32 bit value represents the globally unique identifier of the target or destination of the payload. Its value is undefined by this standard. The Destination ID shall be presented formatted in Little-Endian format.

Source ID

This 32 bit value represents the globally unique identifier of the owner or source of the payload. Its value is undefined by this standard. The Source ID shall be presented formatted in Little-Endian format.

N-Byte Payload

The payload is the content of the message. It may or may not contain an application header as defined by the SDP.

Sequence Number

This 16 bit value increments for every message from a given source id. It starts at 0 with the first message and increments to 65535 and then wraps back to zero. Retransmitted packets because of a perceived Nak are transmitted with the same sequence number as the original transmission. The Sequence Number shall be presented in Little-Endian format.

5. TRANSPORT STANDARDS: COMMON CHARACTERISTICS

The Transport standards defined in this document share some characteristics. A single discussion of those characteristics is called out in this section, rather than duplicating this work for each existing standard. The following sections are:

- Section 5.1, *Header Compression Techniques.*
- Section 5.2, *Packing Multiple Application Messages Per Transport Packet.*
- Section 5.3, *Multi-Part Messaging: Multiple Transport Packets Per Application Message.*
- Section 5.4, *Maximum Packet Sizes.*
- Section 5.5, *Broadcast Semantics.*
- Section 5.6, Discovery.

5.1 Header Compression Techniques

The desire to minimize bandwidth utilization (particularly in wireless communications) is a strong factor in the design of SDP Transport. The JAUS Transport Considerations document [AIR5645] discusses bandwidth considerations in detail with respect to unmanned systems. Here, it is sufficient to note that bandwidth-reduction techniques are highly desirable – to put it succinctly, “You can never have too much bandwidth.”

This realization leads directly to the introduction of data compression techniques in the JAUS / SDP Transport Standards.

Since these compression techniques are being deployed on the individual transport headers and their encapsulated payloads, the compression techniques and formats will be very similar from one transport to the next. The general format for header compression is discussed in 5.1.1. This is followed by a detailed example in 5.1.2. The combination of header compression with broadcast semantics is examined in 5.1.3. Additional techniques for improving the performance of the header compression algorithm are discussed in 5.1.4. Lastly, a brief summary wraps up the topic in 5.1.5.

5.1.1 Header Compression

[RFC3095] discusses a number of generic techniques available to implement header compression. Considering the format of the transport header, the technique that will provide the best and the easiest compression is LSB (Least Significant Bit/Byte) encoding.

Briefly, this technique is opportunistic and works as follows:

- A message sender indicates a readiness to engage in header compression on a particular message header, giving the following information: a number by which this header should be referenced, and a number of bytes at the beginning of this header over which compression will occur.
- A message receiver responds to this indication with a message agreeing to do header compression on that particular message header.
- All subsequent messages flowing from sender to receiver using that particular header are header-compressed by employing the LSB technique: Instead of sending the complete header, simply send:
  - The header number.
  - The number of bytes at the beginning of the header of the current message that are the same as those in the agreed-to compressed header.
  - The rest of the bytes in the header of the current message.

There are two aspects of this technique that must be understood. First, no entity may engage in header compression over a communications link without the express permission of both ends of that communication. Second, within an active system, this technique is deployed for a particular ordered tuple of {sender, receiver, message header}. Each such unique, ordered tuple requires independent negotiation of header compression.

This last point demonstrates that there will be some initial overhead involved in setting up header-compressed communications, as each ordered tuple of {sender, receiver, message header} is negotiated.

This desire to deploy a uniform header compression technique over the Application Layer data leads to Transport Layer message formats that are common across communications media, although the particulars of field sizes, etc. will vary depending on the characteristics of each. In general, a transport message will follow this format:

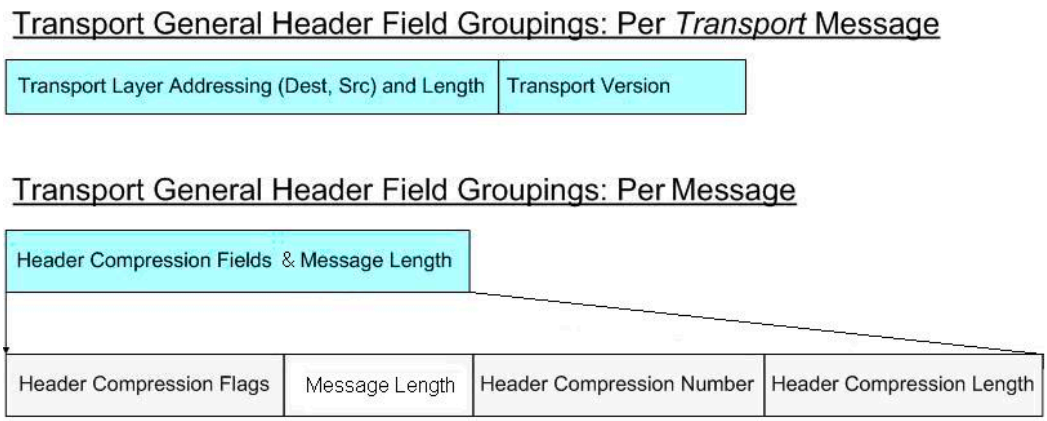


FIGURE 3 – TRANSPORT HEADER COMPRESSION - USUAL FIELDS

A JAUS / SDP Transport Standard will typically include the types of header fields found in Figure 3. In discussing a few details of the characteristics of these fields below, the value “AAAA” will be used for the entity that initiates the header compression, and the value “BBBB” will be used for the entity that agrees (or not) to the compression; this will help keep the roles straight in the discussion. Now, a few details:

- 5669 Transports will always include a version number for the transport itself, to provide a means for backward compatibility/determinability within the standard.
- The Header Compression Fields will typically be comprised of the following sub-fields:
  - Header Compression Flags: These flags must be sufficient to distinguish the following cases:
    - Sender (AAAA) does not support header compression (“0” value in the flags).
    - Sender (AAAA) is requesting that the Receiver (BBBB) engage in header compression on this message (“1” value in the flags, with a non-zero length value).
    - Sender (BBBB) is accepting header compression on this message (“2” value in the flags, with a non-zero length value).
      - Note that this is a response message in the header compression protocol initiated by AAAA, and so therefore the “Sender” here is BBBB, responding to AAAA’s request.
      - Note also that if BBBB does not wish to engage in header compression, it has merely to not respond to AAAA’s request – that is, there is no explicit message for the rejection of a header-compression-request message.
    - Sender (AAAA) is sending a header-compressed message (“3” value in the flags).
    - There are two additional cases, provided in order to accommodate recovery mechanisms:
      - Sender (AAAA) is requesting that the Receiver (BBBB) forget a particular header compression (“1” value in the flags, with a zero length value). [Note: This scenario is not often encountered, but is presented here in order to demonstrate that either side of an established header compression may terminate the compression.]

    - Sender (BBBB) has no record of this header compression, and is requesting the header compression protocol restart from the beginning (sender supplies the “2” value in the flags, with a zero length value). [Note: This situation may occur if a asset has agreed to header compression with some remote entity, and then reboots, losing its previous header compression table – that is, this message provides a recovery mechanism for the header compression protocol.]

- Header Compression Number: The number that the Sender and Receiver are agreeing to use to refer to this message header.
    - If the Sender does not support header compression, the value of this field is always zero.
- Header Compression Length: Depending upon the value in the Header Compression Flags, this value is one of the following:
    - Header Compression Flags = 0: If the Sender (AAAA) does not support header compression, the value of the Header Compression Length field is always zero.
    - Header Compression Flags = 1: If the Sender (AAAA) is requesting that the Receiver (BBBB) engage in header compression on this message, the value of the Header Compression Length field is the number of bytes that the Sender is requesting the Receiver remember – that is, the maximum number of bytes of the header that the Sender will compress.
        - Note that while this technique is termed “header compression”, there is no reason why this compression cannot extend deeper into the message body – as long as the byte values are the same from one message to the next, “header compression” will yield benefits. For JAUS messaging, particularly with messages that include presence vectors, this is somewhat likely.
    - Header Compression Flags = 2: If the Sender (BBBB) is accepting header compression on this message (a header compression initiated by the Receiver (AAAA) of this message – this is a response message in the header compression protocol), the value of the field is the number of bytes of the Receiver’s header that the Sender is promising to remember.
        - Again, note that while the initiator of a header compression may have requested that the other party remember any number of bytes, the other party is under no obligation to accept that value. Header compression may be rejected altogether (by never agreeing to do it in the first place), or may be modified by agreeing to remember a smaller number of bytes than the original request.
    - Header Compression Flags = 2: If the Sender (BBBB) has no record of this header compression, this message is sent with the value of the Header Compression Length field set to zero, indicating a request to the Receiver (AAAA) to restart the compression protocol.
    - Header Compression Flags = 3: If the Sender (AAAA) is sending a header-compressed message, the value of the field is the number of bytes from the original header that should be placed at the start of the current message in order to form the uncompressed message.

The example given in 5.1.2 should clarify these details.

5.1.2 Header Compression: An Example

The header compression technique to be employed over the 5669 Transport is best understood by means of an example. This document will examine the case of an operator control unit (OCU) and a robot agreeing to a header compression, both from OCU to robot, and from robot to OCU. The robot will then experience a fault, reboot, and re-establish header compression.

In each of the sections below, the figures show not only the message formats, but also the tables that are built up internal to the OCU and the robot that maintain the header compression data.

5.1.2.1 Initial State: No Compression

The initial state of the system is to start with uncompressed headers, as shown in Figure 4.

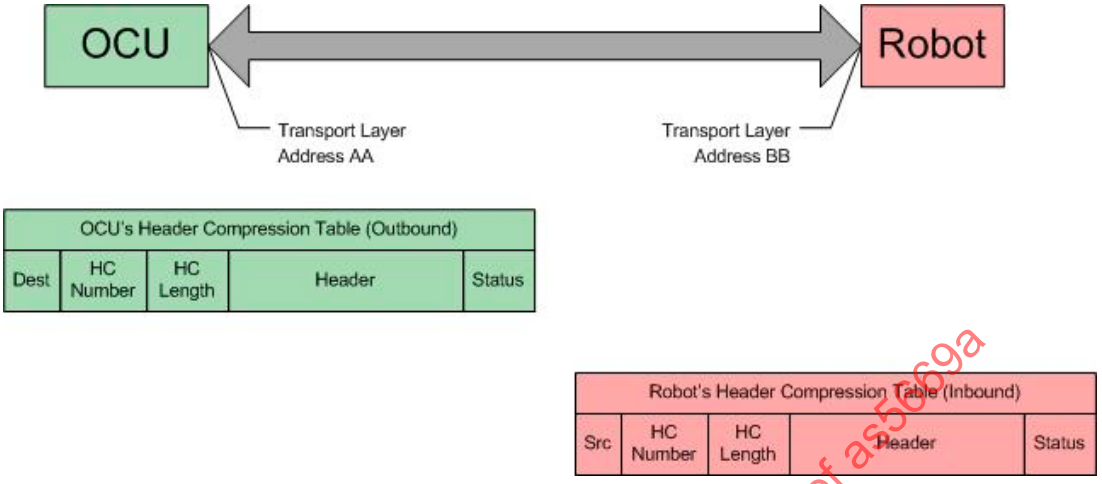


FIGURE 4 - INITIAL ENVIRONMENT AND HEADER COMPRESSION TABLES

Note that in the initial state there are no entries in any of the header compression tables. Note also that there are different tables for Inbound and Outbound header compressions.

5.1.2.2 Send of Initial Message Requesting Header Compression: OCU to Robot

In order to establish header compression for a particular message that is sent by the OCU to the robot, the message must first be sent once in its uncompressed form, along with the information required to know how to refer to that message when it is in its compressed form in later transmissions. This initial transmission is shown in Figure 5.

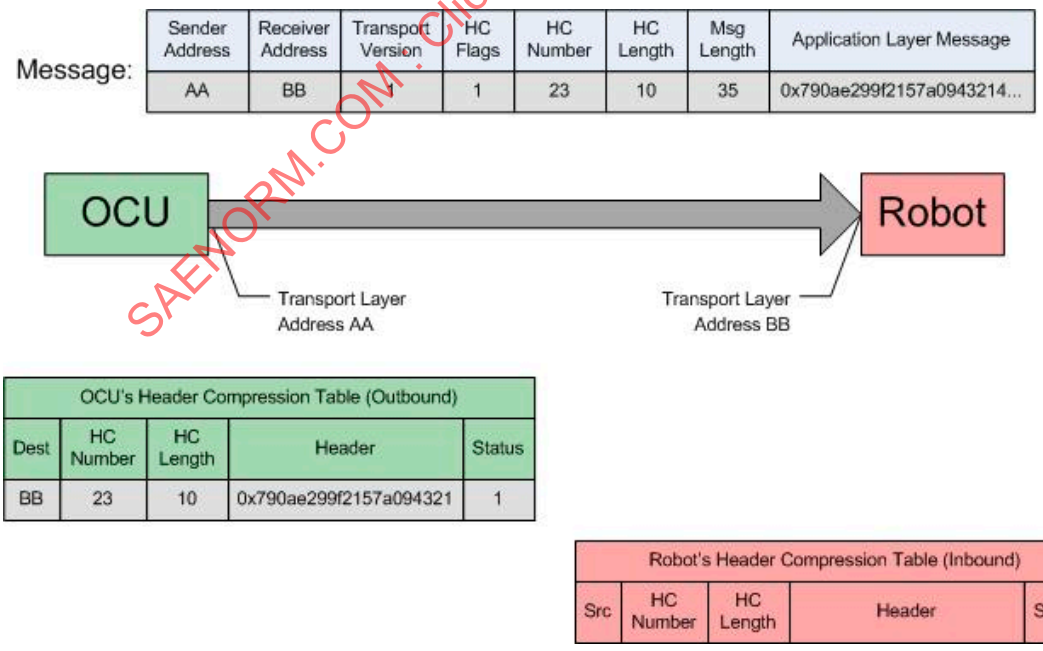


FIGURE 5 - INITIAL (UNCOMPRESSED) MESSAGE SENT FROM OCU TO ROBOT

Notice that the sender creates an entry in the “Outbound” table with a value of 1 for the status field, indicating that this message has been sent out, but the OCU has not got a response agreeing to header compression.

5.1.2.3 Receive of Initial Message Requesting Header Compression: OCU to Robot

When the robot receives this message, it will pass the actual application message up to the Application Layer for processing. At the Transport Layer, it notices that the “HC Flags” field has a value of 1, indicating that the sender is willing to engage in header compression on this message. The robot may then decide upon one of two courses of action: If header compression is not desired, then the transport layer may simply ignore the header compression request; if header compression is desired, then the Transport Layer begins by storing away the necessary header compression information. This is shown in Figure 6.

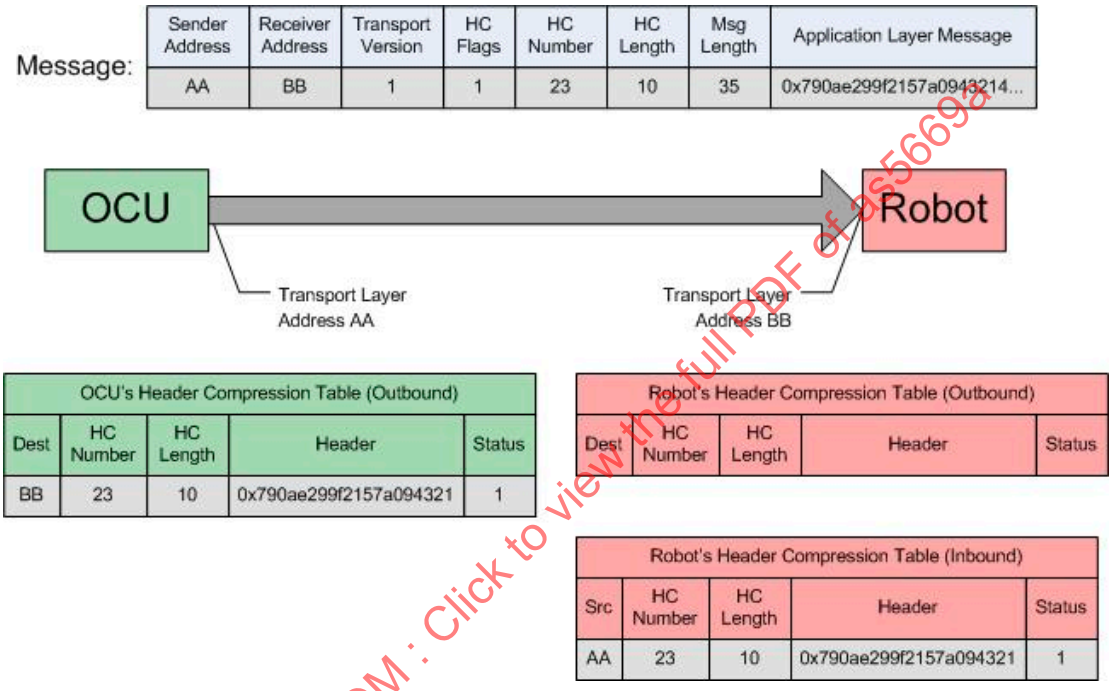


FIGURE 6 - BEGINNING OF HEADER COMPRESSION ENTRY ON ROBOT (RECEIVER) SIDE

Notice that the header compression data is stored in the “Inbound” table on the robot (Receiver), which is distinct from the “Outbound” table. Remember that this header compression technique establishes these compressed headers for the ordered tuple of {sender, receiver, message}.

5.1.2.4 Receiver Replies Accepting Header Compression: Robot to OCU

Once the receiver has established an entry in its “Inbound” header compression table for the message, it sends a response message agreeing to header compression for this message. This is shown in Figure 7.

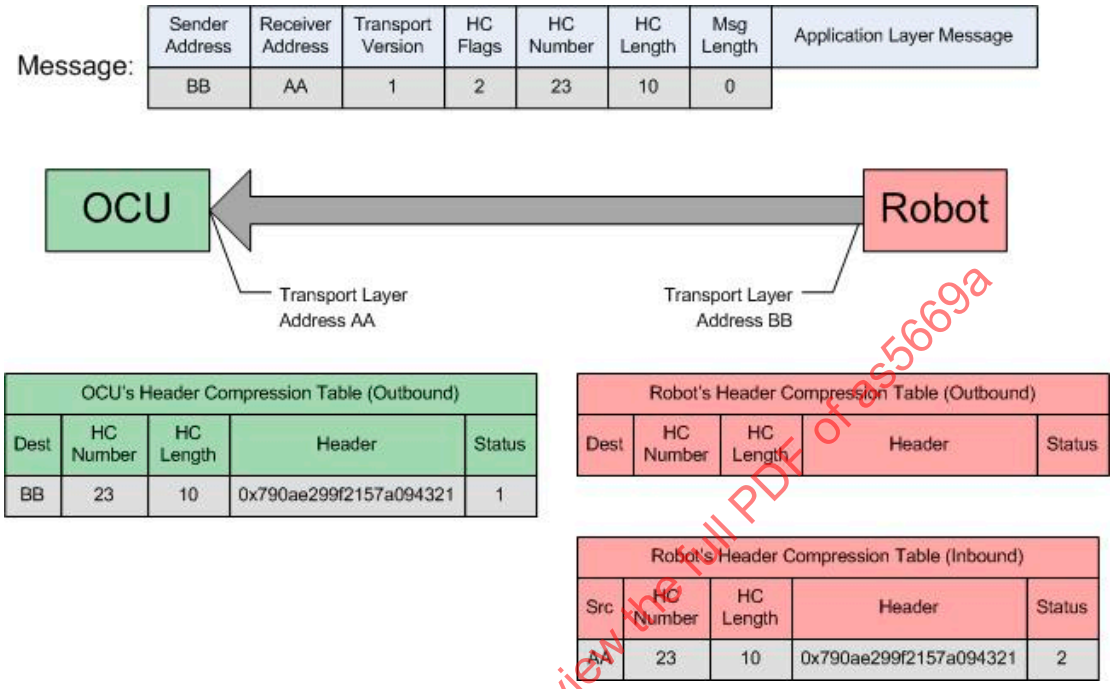


FIGURE 7 - ROBOT SENDS RESPONSE MESSAGE AGREEING TO HEADER COMPRESSION

Notice that the “Status” value in the Robot’s “Inbound” table has changed value from 1 to 2, indicating that an agreement response has been sent. Note also that the message sent back to the originator has a zero-length Application Layer message, making it in effect a transport-specific message, sent only to establish the header compression. This is part of the up-front cost that is paid to establish compression.

5.1.2.5 Sender Receives Reply Accepting Header Compression: Robot to OCU

When the OCU receives this message, it modifies its “Outbound” table as indicated in Figure 8.

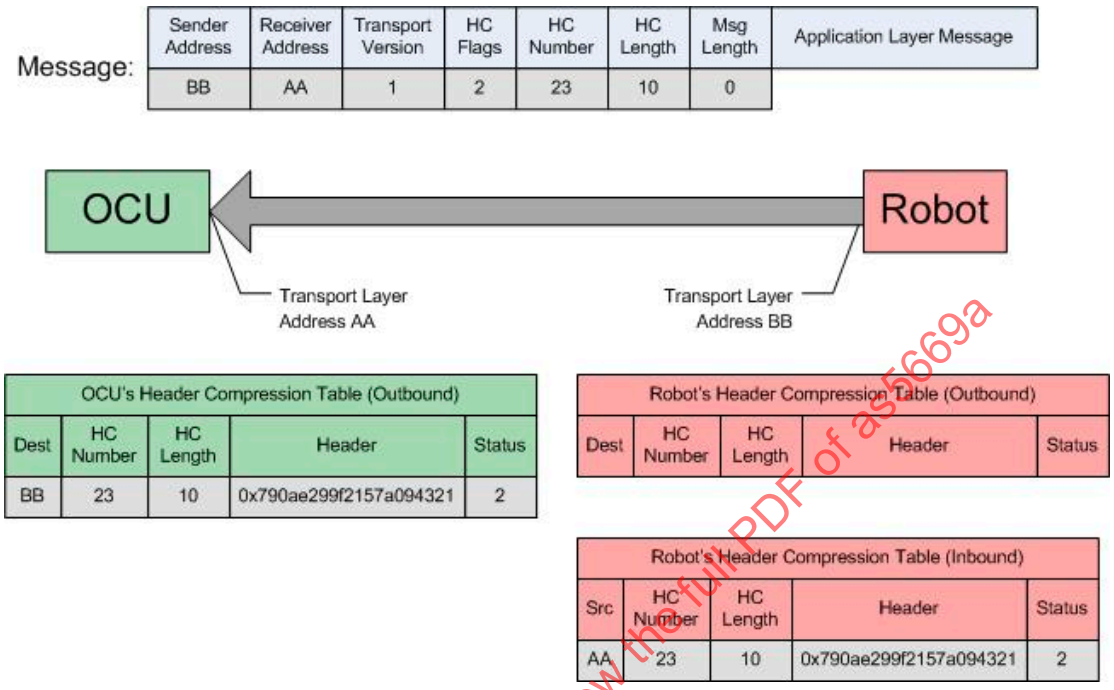


FIGURE 8 - OCU RECEIVES MESSAGE AGREEING TO HEADER COMPRESSION FROM ROBOT

Notice that the “Status” in the OCU’s “Outbound” table has its value changed from 1 to 2, indicating the receipt of the agreement message. If the agreement message had indicated a value in the “HC Length” field that was less than that stored in the table, that entry in the table would likewise be updated.

5.1.2.6 Sending a Compressed Message: OCU to Robot

Now that header compression has been agreed to for this particular message flowing from OCU to robot, the next time the OCU wishes to send a message that begins with that same series of bytes (or at least a sufficiently large portion of those same bytes), header compression may be employed, as shown in Figure 9.

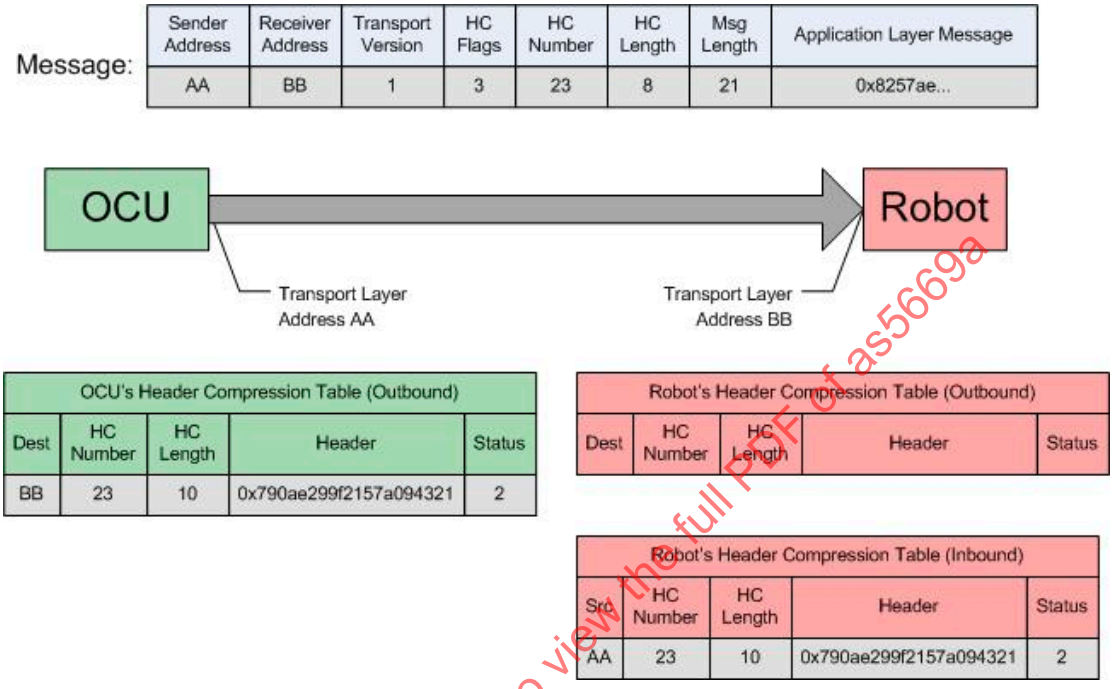


FIGURE 9 - COMPRESSED MESSAGE SENT FROM OCU TO ROBOT

Notice that the value in the “HC Length” field indicates the number of bytes that are being omitted from the beginning of this message via the LSB header compression scheme.

5.1.2.7 Uncompressing a Compressed Message: OCU to Robot

When the robot receives this message, it looks at the “HC Flags” field and realizes that this is a message that requires decompression.

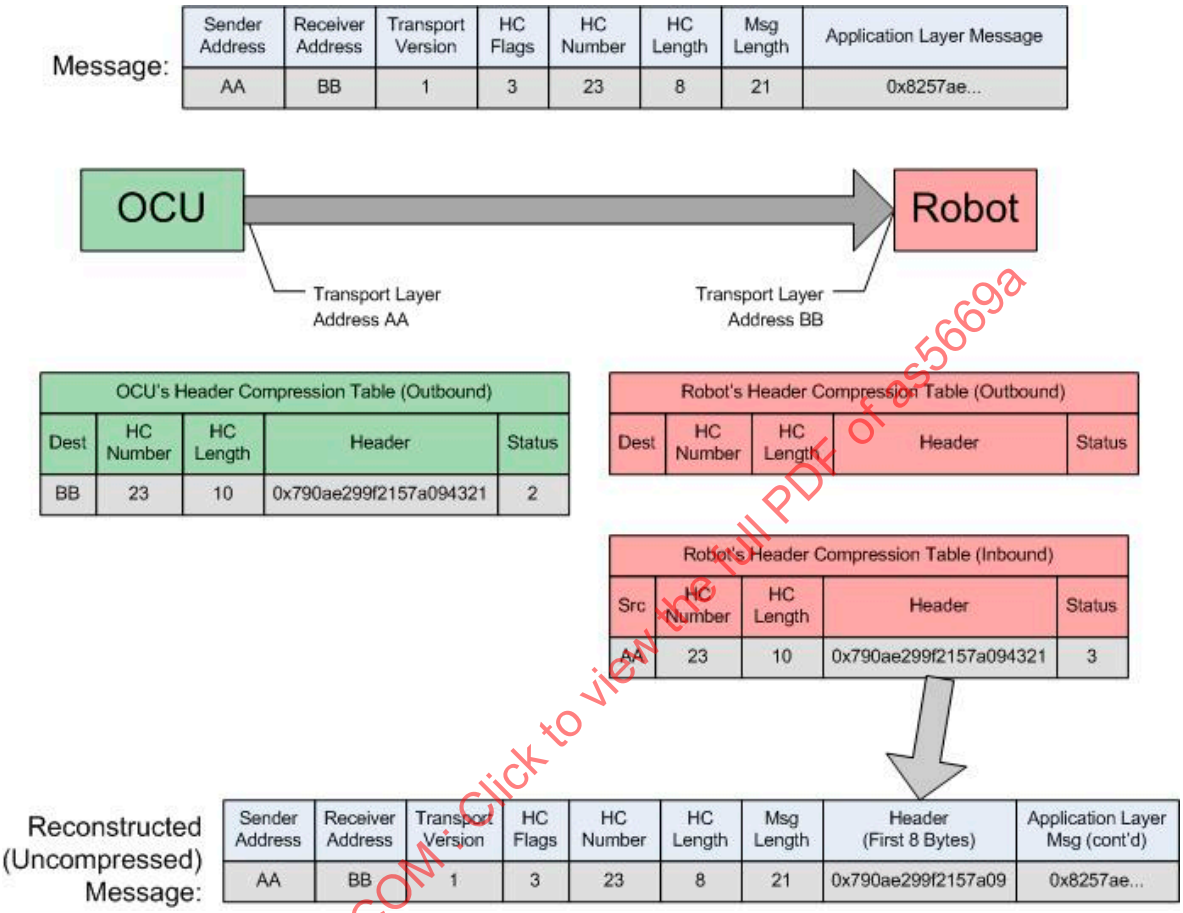
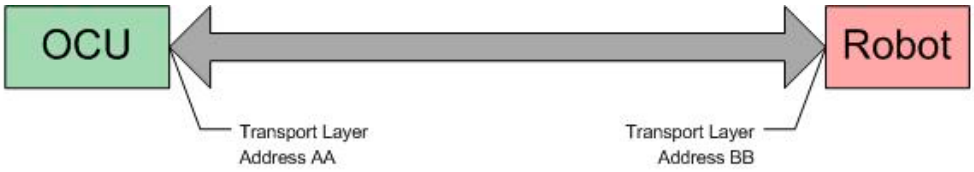


FIGURE 10 - UNCOMPRESS OF OCU→ROBOT MESSAGE

Figure 10 shows how the original message is reconstructed from the compressed message, by inserting the first 8 bytes of the previously stored header in front of the remaining (different) bytes of the balance of the message. Notice also that the status in that line of the Robot’s Header Compression Table (Inbound) is changed from 2 to 3, indicating that a compressed message has now been received for that header compression.

5.1.2.8 Symmetric Establishment of Header Compression: Robot to OCU

At the same time that the OCU is establishing a compressed header with the robot, the robot may establish its own compressed headers with the OCU. The resultant tables are shown in Figure 11.



OCU's Header Compression Table (Outbound)

Dest	HC Number	HC Length	Header	Status
BB	23	10	0x790ae299f2157a094321	2

Robot's Header Compression Table (Outbound)

Dest	HC Number	HC Length	Header	Status
AA	25	5	0x2397a8e4f0	2

OCU's Header Compression Table (Inbound)

Src	HC Number	HC Length	Header	Status
BB	25	5	0x2397a8e4f0	3

Robot's Header Compression Table (Inbound)

Src	HC Number	HC Length	Header	Status
AA	23	10	0x790ae299f2157a094321	3

FIGURE 11 - HEADER COMPRESSION TABLES FOR ROBOT→OCU AND OCU→ROBOT

Notice the following characteristics of the header compression tables:

- The “Status” entries in the “Outbound” tables have the value 2, indicating that a message has been received from the corresponding destination address agreeing to header compression on the given entry.
- The “Status” entries in the “Inbound” table have the value 3, indicating that a header-compressed message has been received.
- The values in the OCU’s “Inbound” and “Outbound” tables are distinct, and have nothing to do with each other. Remember that header compression is done uniquely for each ordered tuple of {sender, receiver, message header}.

5.1.2.9 Header Compression Forgotten: Robot Reboots

Once header compression has been established, it is possible that one of the parties to the communication may experience a fault and need to be restarted. This may cause the loss of the header compression data, as shown in Figure 12.

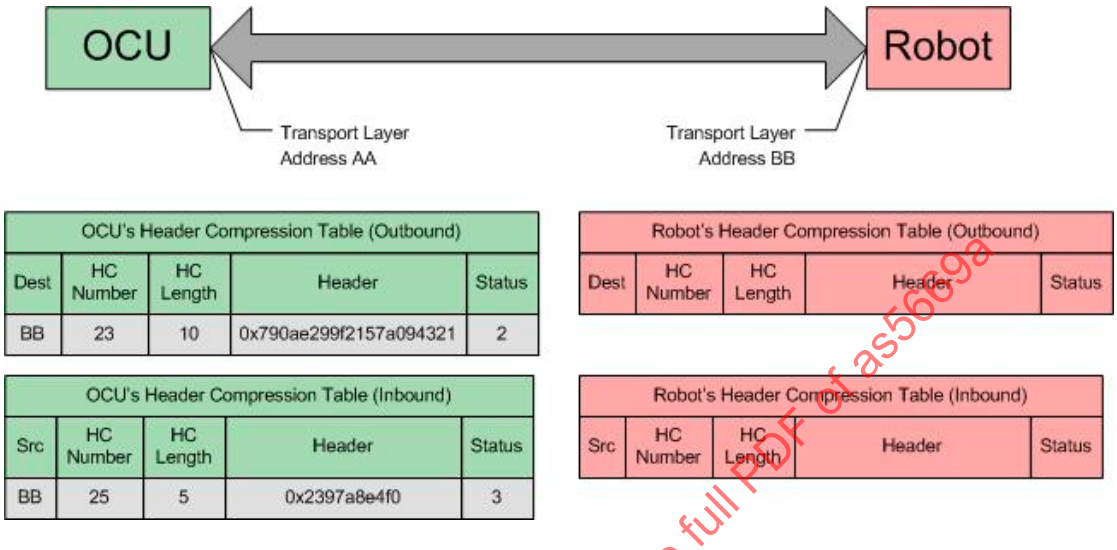


FIGURE 12 - ROBOT EXPERIENCES A FAULT AND REBOOTS, LOSING HEADER COMPRESSION DATA

Notice that the header compression data in both the “Inbound” and “Outbound” tables on the Robot is lost.

5.1.2.10 Reestablishing Header Compression: Robot Receives Compressed Message

When the robot reboots, it has no knowledge that previously established header compression data exists in the memory of other entities in the system (such as the OCU). These other entities are likewise unaware of the robot's loss of header compression data, and so entities like the OCU are still sending the compressed messages, as shown in Figure 13.

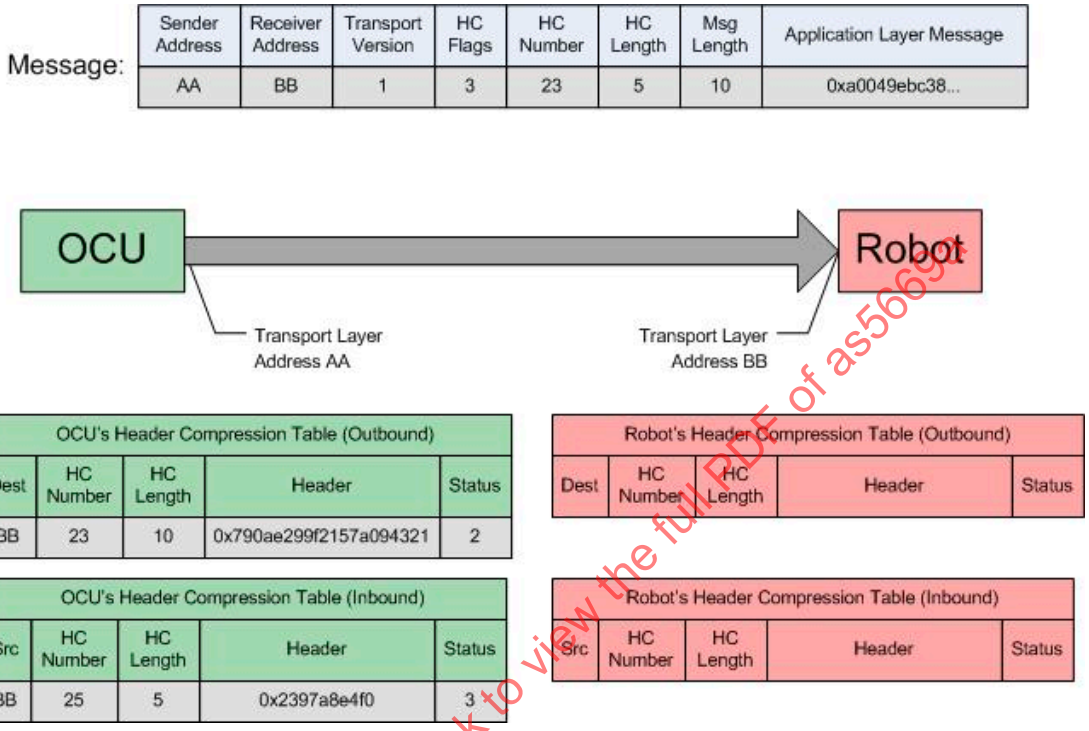


FIGURE 13 - ROBOT RECEIVES COMPRESSED MESSAGE WITHOUT CORRESPONDING ENTRY

When the robot receives this message and attempts to look it up in its "Inbound" table, it finds no record, and so cannot uncompress the message.

5.1.2.11 Reestablishing Header Compression: Robot Sends a “Zero Bytes Remembered” Response

When the robot receives a compressed message for which it cannot find a corresponding entry in its “Inbound” table, it responds by sending back a message indicating its lack of knowledge regarding the compressed header. This is done by placing the value 2 in the “HC Flags” field and the value 0 in the “HC Length” field – in effect telling the sender that the receiver is currently remembering zero bytes of the header in question. This is shown in Figure 14.

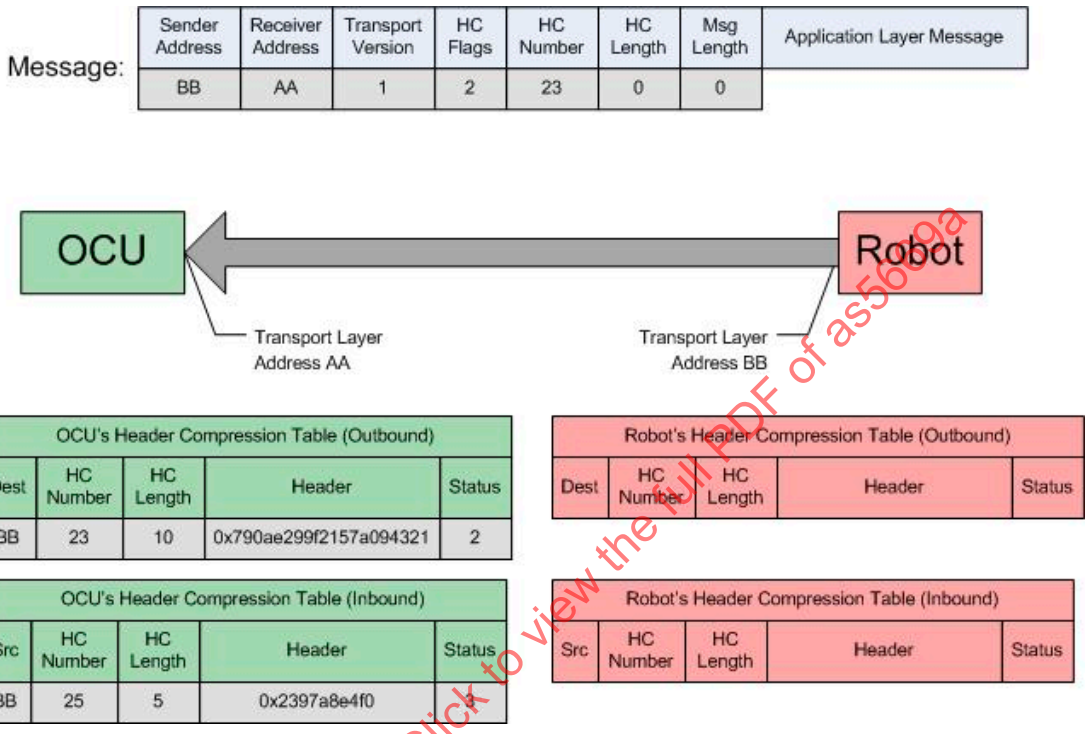


FIGURE 14 - ROBOT SENDS ZERO-BYTES-REMEMBERED RESPONSE

Notice that no entries are made in the Robot’s header compression tables at this time. And again, we notice that the Application Message has a zero length, making this a transport-specific message.

5.1.2.12 Reestablishing Header Compression: OCU Receives Zero-Bytes-Remembered Message

When the OCU receives this message, it notes the “HC Flags” and “HC Length” values, and so determines that the Robot knows nothing of this particular header. So, the OCU updates its header compression tables to indicate this, and reinitiates the header compression protocol. This is shown in Figure 15.

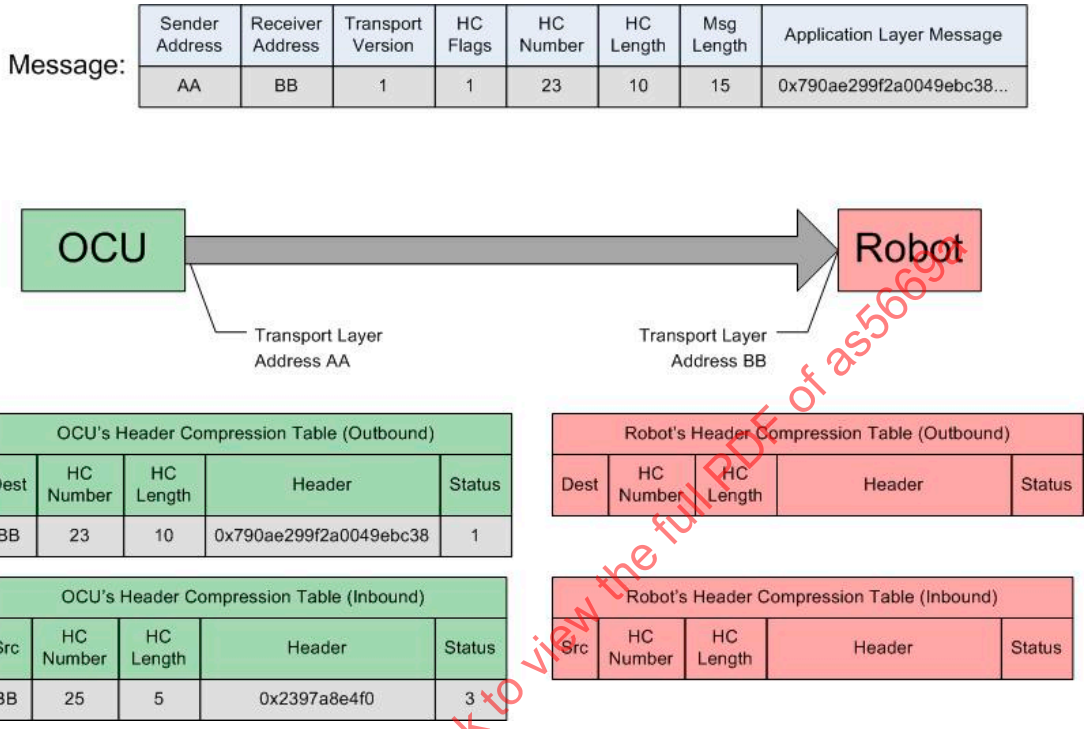


FIGURE 15 - OCU BEGINS REINITIALIZATION OF HEADER COMPRESSION PROTOCOL

Notice that the “Status” field in the record in the OCU’s “Outbound” header compression table now has the value 1, indicating that the protocol has returned to its initial step.

5.1.2.13 Reestablishing Header Compression: Updating the OCU’s Header Compression Table

After the Robot reboots, the header compression tables in the OCU contain invalid data. The above sections have demonstrated how the OCU’s “Outbound” table is updated. The “Inbound” table must likewise be updated.

However, this is relatively easy as the updating of the “Inbound” table happens as a natural consequence of the header compression protocol. When the Robot sends its initial (post-reboot) messages, they will contain the initial “HC Flags” value of 1, which will then be stored in the “Inbound” table, overwriting the previous, outdated entries.

5.1.3 Header Compression and Broadcast

Messages whose destination is a broadcast identifier can never be compressed. The reason that broadcasts cannot be compressed is that the compression protocol requires that the receiver of compression request respond with an agreement message before compression can take place, and there is no entity within a system that can so respond on behalf of a broadcast destination.

5.1.4 Additional LSB Techniques

The LSB technique as defined in [RFC3095] can also be specified using an additional “mask” on the header bytes. This mask allows only selected pieces of the header to be used in the compression. Supplying such a mask essentially allows selecting only some of the header fields for compression in order to achieve better compression ratios (see [RFC3095] for details). For instance, if the more invariant data lies deeper in the header (following data that is more variable), then the use of a mask may provide better compression results.

At this point in time, the trade-off between the complexity of allowing masks for header compression versus the additional compression gained by employing them is deemed insufficiently rewarding to justify the expense.

5.1.5 Header Compression: Summary

The LSB header compression technique is demonstrated in the above example. However this example is done without reference to the particulars of a specific transport medium, or defined packet formats, field lengths and placement, etc. In the actual JAUS / SDP Transport Standards (given in 6.1, 6.2 and Section 7), each header compression field demonstrated in the example is defined and fully specified to create the necessary functionality.

5.2 Packing Multiple Application Messages Per Transport Packet

The domain of (uncompressed) Application messaging currently comprises messages of any size (such as, for example, large image data, map data, etc. – albeit these larger messages are broken down into 4K pieces – see 5.4). However, a typical JAUS message generally falls into the 20 to 40-byte size range.

If only a single message were packed into an individual transport packet, this would represent a significant inefficiency in use of typical transport media. No matter what the specific transport medium is (Ethernet, serial, etc.), there is non-trivial overhead involved in not only the packaging of the packet format, addressing, etc., but also in more low-level functionalities such as establishing exclusive control of the transport medium in order to send, checksums on the overall packet, etc.

Therefore, the JAUS / SDP Transport Standards support the packing of multiple application messages into a single transport packet. This enables the development of more efficient Transport Layers for transport media in which the standard packet size is significantly larger than the typical size of a message.

The specific implementation at the Transport Layer must decide the criteria by which it will determine when a transport packet is “full enough” to send – sending packets as soon as a message is available will result in one-message-per-transport-packet and very inefficient bandwidth usage; conversely, waiting to fill an entire transport packet before sending it may result in unacceptable latencies for the application messages early in the packet. This trade-off must be made by the responsible Engineer. General guidance would include such engineering techniques as the calculation of a latency budget, and latency mitigation requirements such as (for example), “Send emergency-priority messages immediately in their own packet.”, which may provide for a more generous budget in the general case.

All of the standards currently defined in Sections 6 and 7 of this document conform to this constraint. Therefore, all current JAUS / SDP Transport Standards support the packing of multiple application messages into a single transport layer message.

5.3 Multi-Part Messaging: Multiple Transport Packets Per Application Message

While the typical message is relatively short, there are messages (such as those that transmit image and/or map data) that may be rather lengthy. Support for multi-part messaging is provided by the Data Flags and Sequence Number as defined in the General Transport Header.

5.4 Maximum Packet Sizes

Messages may flow across a variety of transports on their way to their destination – standard Ethernet, wireless IP-based radios (like 802.11b/g), long-range serial radios, etc. Each of these communications media generally has a parameter known as the MTU, or Maximum Transfer Unit (see definition in 2.3). Generally, it is desirable to keep the size of one's packets less than or equal to the MTU size, as this allows for the most efficient communications, particularly in lossy communications environments.

When working in a heterogeneous communications environment, it is good practice to restrict the size of an individual packet to be at most the smallest of the various MTU values for the various communications media. For example, consider a system that is working with both:

- Ethernet: Typical MTU: 1500 bytes.
- 802.11b Wireless: Typical MTU: 2346 bytes (also known as the "fragmentation threshold").

Even those assets whose communications interfaces are on the 802.11b part of the network would do well to restrict their packets to the smaller MTU: 1500 bytes – avoiding the fragmentation on the Ethernet portion of the network.

There is another subtlety here: Consider UDP packets, which according to the UDP standard may be up to (2^{16}-1) bytes long. Typically no one uses such long UDP packets on current communications media, as they would be broken up into MTU-sized pieces further down the IP protocol stack anyway. However, if the JAUS / SDP Transport Standard were to accept the (2^{16}-1) bytes value as the constraint on packet sizes across a UDP-based transport, then all engineers wishing to design a compliant and interoperable system using the UDP transport would have to develop code/hardware that could handle such large packets, even though surely almost no one would actually send them.

Likewise, serial communications media could employ equally large packets. However in practice, most packets would not be nearly so long, as the potential for bit errors to invalidate the entire message increases with the size of the message, making such large packets risky in a noisy communications environment. Again, all engineers wishing to design a compliant and interoperable system using such a serial transport would have to develop code/hardware that could handle the receiving of such large packets, even though (again) almost no one would actually send them.

This problem, if not dealt with in the JAUS / SDP Transport Standards, would cause engineers who develop these systems to devote time and resources to this compliance/interoperability issue. They would be developing functionality that would (in practice) almost never be exercised. This is inefficient.

Therefore, each JAUS / SDP Transport Standard defines a Maximum Packet Size (MPS). This value is defined for each transport standard to aid the development of efficient, compliant and interoperable systems.

5.5 Broadcast Semantics

The semantics of the Application messaging requires support for broadcast semantics. Of concern to transport is the requirement to support such semantics for the following two cases:[2]

- Broadcast to all available destinations (global broadcast)
- Broadcast to local destinations only (local broadcast)

Note that the ACK/NAK protocol is not supported for broadcast transmissions – broadcast semantics and guaranteed message receipt are incompatible.

[2] See Section 4 for examples of broadcast scoping.

5.6 Discovery

The discovery of an entity on the wire by another entity on the wire using the same SDP such as JAUS is accomplished by a valid transmission and reception of a valid message. The fact that a discernible source ID sent a message to a discernible destination ID allows transport to identify a mapping of the location of various Source ID against physical transport locations.

This approach supports the use of heartbeat messages, creation of specific application layer identification messages as well as monitoring only applications.

For this discovery process to work properly the following rules apply:

- The physical transport location that sends a message must also receive one from the same source ID. For example, an implementation that sends a UDP broadcast message on port 10001 must also accept incoming messages on that port.
- Multiple source IDs may be assigned to the same physical transport location

6. JAUS OVER IP

IP-based networks are the most common standard for modern communications. The JAUS / SDP Transport Specification supports the transmission of messages over IP-based networks. The "JAUS Over UDP" (or JUDP) standard supports unreliable, best-effort communications, while the "JAUS Over TCP" (or JTCP) standard supports reliable, unbounded-latency communications. JAUS is used as the example protocol implemented within these transports but can really be any SDP.

The standard described in this section applies to both IPv4 and IPv6. Future versions of this standard may allow these formats, etc. to diverge once greater experience in IPv6 networks indicates the best avenues to exploit IPv6 functionality.

6.1 JUDP : JAUS Over UDP

The JAUS / SDP Transport Specification supports the transmission of messages over IP-based networks primarily via UDP packets. UDP provides the low-latency-but-potentially-lossy characteristics that are required for most communication in unmanned systems environments. Hence it is the primary standard for communication in IP networks.

In the context of JAUS, this standard is known as the JUDP standard.

6.1.1 Characteristics of JUDP

JUDP provides non-reliable, best-effort message transport of JAUS message traffic between entities. The JUDP transport shall not preclude the use of other protocols or messaging schemes via UDP between the same nodes.

JUDP is a datagram-based transport built atop UDP/IP, and intended for implementation as a wrapper around UDP/IP transport providing additional non-native services.

6.1.2 Application Domains

JUDP may be used in any application domain possessing adequate bandwidth to maintain acceptable system performance. It is unlikely, for example, that JUDP will be applicable to the Unmanned Undersea Vehicle (UUV) application domain.[3] Applicability must be determined by the responsible Systems Engineer.

[3] For definition and description of application domains and their communication characteristics, please refer to [AIR5645].

6.1.3 Communications Environments

The JUDP standard supports at a minimum the following two environments: an Ethernet environment (supporting UDP/IP over Ethernet), and a serial communications environment (supporting UDP/IP encapsulation via PPP for communications via a serial link of sufficient speed[4]).

6.1.3.1 Ethernet

The JUDP standard supports UDP/IP communications over Ethernet.

6.1.3.2 Encapsulation for Serial Links

The JUDP standard supports UDP/IP communications over serial links via PPP encapsulation of IP datagrams.

6.1.4 Description

The following subsections provide a description of JUDP packet structure, and of the protocol build-up of JUDP packets for communication across Ethernets and over (sufficiently speedy) serial links via PPP. All JUDP messaging shall follow the packet format and semantics described in this section.

6.1.4.1 JAUS UDP/IP Packet Structure

Each message packet is structured as one or more payloads encapsulated in a JUDP wrapper. The JUDP packet structure may be outlined as follows:

- The Per-Transport-Packet Header: Native UDP already provides source and destination addressing as well as an overall length of the packet. Therefore, the JUDP per-packet header need only identify the transport version.
- The Per-Message Header: The General Transport Header given in Section 4 is added to each message in the packet.

Remember that we may pack one or more messages into a single UDP packet – as is shown in Figure 16. The packet therefore contains only a single version byte, but each message within the packet includes the General Transport Header.

[4] Please refer to [AIR5645] for a discussion of the relevant characteristics of serial, PPP, Ethernet, etc. in order to determine what “sufficient speed” is for the system under consideration.

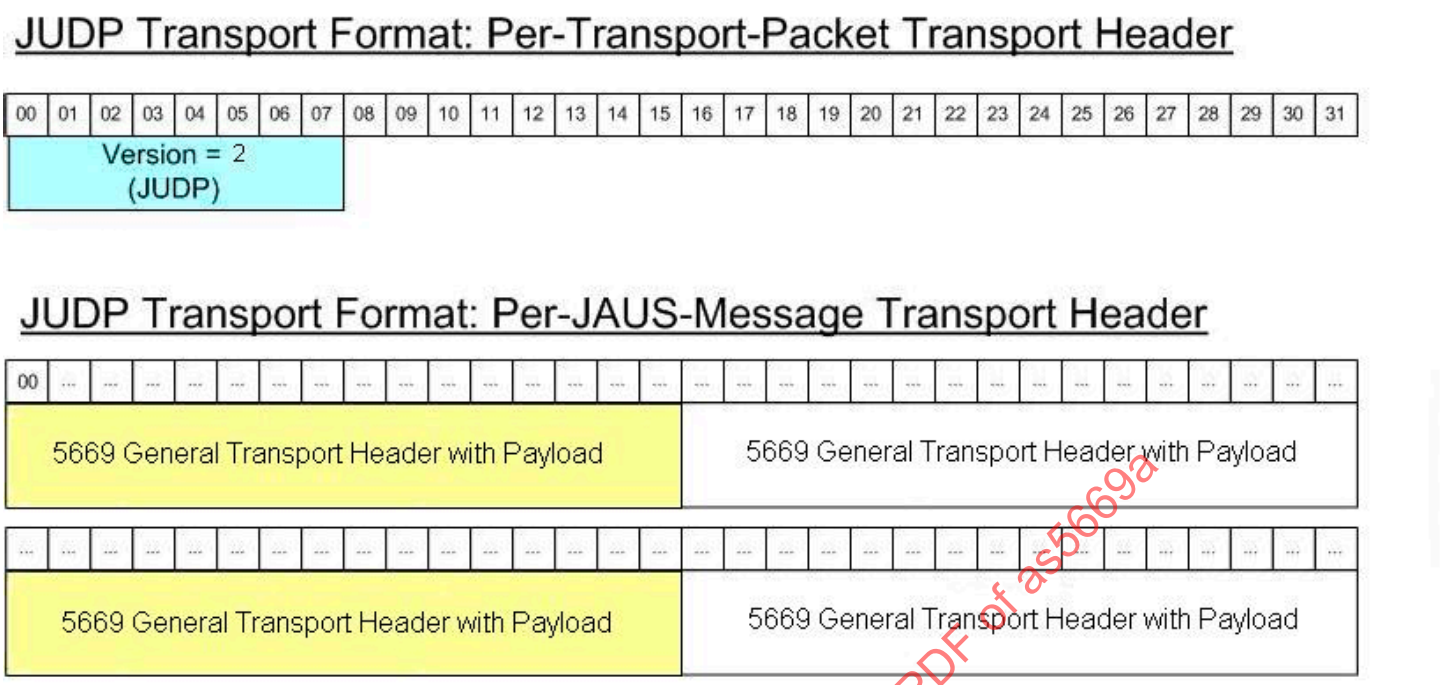


FIGURE 16 - MULTIPLE MESSAGES IN A SINGLE UDP PACKET

6.1.4.2 JUDP Transport Header Field: Version

The JUDP transport version is encoded in the first byte of the packet. This allows any receiver of the message to determine compatibility. Any receiver of a JUDP message packet shall determine compatibility based on the Transport Version, and shall ignore and discard ("silently discard") any packet whose version (hence structure) is unknown to or unsupported by the receiver.

In the initial version of the JUDP Transport, the JUDP Transport Version was 0x01. This document increments the JUDP Transport Version to 0x02

As a note: The value 'J' (decimal 74) is reserved for the Version field. Early, experimental versions of JAUS transport used a transport header that began with this value in the first byte of the UDP packet payload. While it is somewhat unlikely that this would ever lead to confusion within the AS5669 standard itself – as this standard is unlikely to go through 73 revisions – this value is nevertheless reserved/documented here in order to indicate that these legacy/historical implementations may be encountered "in the wild" in the early days of adoption of this standard.[5]

6.1.5 Implementation of Broadcast Semantics

The System Designer may choose to implement the broadcast by any appropriate means that preserves the semantics of it – that is, any means of communication that achieves the "best effort to distribute the message to all intended parties" functionality. In the world of IP-based networking, this is typically done one of two ways: multicast, or "true broadcast".

A "true broadcast" implementation might call for a local broadcast message to be sent, for example, to an address like 192.168.128.255 – a classic "Class C" private subnet broadcast (on an IPv4 subnet 192.168.128.0/24). A multicast implementation might call for a broadcast message to be sent to, for example, an address like 239.255.0.1 – a typical multicast address. Broadcasts can be scoped for local or global destinations using different multicast groups.

Either choice is valid. It is therefore advisable to design implementations that are capable of handling both based on easily modifiable configuration parameters.

[5] Documentation regarding this early packet format may be found in the archives of the AS-4 (JAUS) Experimentation Task Group, at the JAUS Working Group website, http://www.jauswg.org.

6.1.5.1 Best Practices Guidance: The Most Common Implementation of Broadcast Semantics

The implementation of broadcast semantics in the JUDP standard is typically done via multicast messaging, which allows network administration to control/restrict the flow of message traffic more easily – at least more easily than would "true broadcast" addressing.

The actual assignment of multicast addresses is no more within the scope of this standard than is the assignment of regular IP addresses – both must be handled by the Network Administrator/Designer for the system in question. A well-designed system implementation will leave the multicast address as a configuration parameter that may be easily modified to suit the fielded environment.

One typical assignment (and the one used by the JAUS Experimentation Task Group (ETG) in its Interoperability Experiments) is to use a multicast address from the "statically assigned link-local"[6] range: 224.0.0.0/24. The multicast address the ETG uses is usually 224.1.0.1. However, this address falls into a range specifically reserved by IANA, and is therefore not suitable for real-world deployment.

A multicast address from the "administratively scoped IPv4 multicast space"[7], in the IPv4 Local Scope (such as 239.255.0.1) is a more appropriate assignment. [RFC2365] also provides guidance on mapping the selected multicast address to the IPv6 address space.

Additionally, if unmanned assets deploy other protocols which also use multicast, the System Designer may wish to use the same multicast address for all protocols. Using the same multicast address would both reduce network bandwidth devoted to the dissemination of data regarding multicast groups (IGMP), and more easily enable multicast routing to handle routing all packets from multiple multicasting protocols within the same network boundaries.

6.1.6 Encapsulation

The JUDP transport must support at least two means of delivery: encapsulation of the JUDP packet within an Ethernet packet for transmission across an Ethernet transport, and encapsulation within a PPP packet for transmission across serial links.

These two encapsulations are illustrated and discussed in the following two subsections.

[6] As described by [RFC2365], Section 6.3.
[7] As described by [RFC2365], Section 6.1.

6.1.6.1 Ethernet

The following figure illustrates the progressive encapsulation of a message within a JUDP packet for transport via UDP/IP across an Ethernet[8].

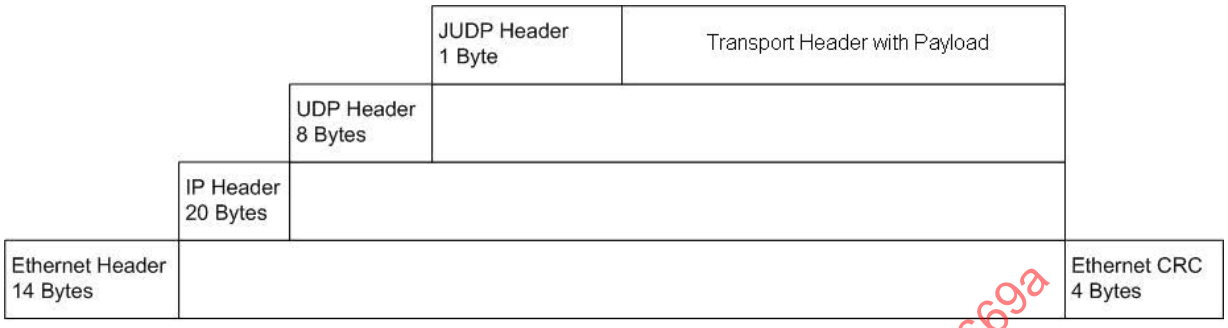


FIGURE 17 - JUDP TRANSPORT PROTOCOL STACK OVER ETHERNET

IEEE 802.2 / 802.3 encapsulation[9] is similar, differing primarily in the addition of the 802.2 LLC and 802.2 SNAP. The differences between IP encapsulations within Ethernet and IEEE 802.2 / 802.3 frames are described succinctly in [STEVENS93].

This results in:

- An overhead of 46 bytes per packet – 14+20+8+4 (with the “14”, “20” and “8” values coming from the corresponding headers, and the “4” being the ending CRC on the packet).
- An overhead of 1 byte per JUDP packet to accommodate the version number field that is only listed once at the beginning of the packet.
- An overhead of 14 bytes per message for the General Transport Header.
- A maximum of 61 bytes total overhead per payload message.

This overhead is not prohibitive given the relatively high available bandwidth typical of Ethernet implementations. Using header compression techniques and multiple-messages-per-UDP-packet will significantly improve bandwidth utilization.

[8] As described by [RFC894].
[9] Described in [RFC1042].

6.1.6.2 PPP

The use of IP encapsulation techniques for transmission of IP datagrams over serial links is inefficient of bandwidth. For example, transmitting a message using UDP/IP transport via PPP encapsulation over a serial link results in the following protocol buildup:

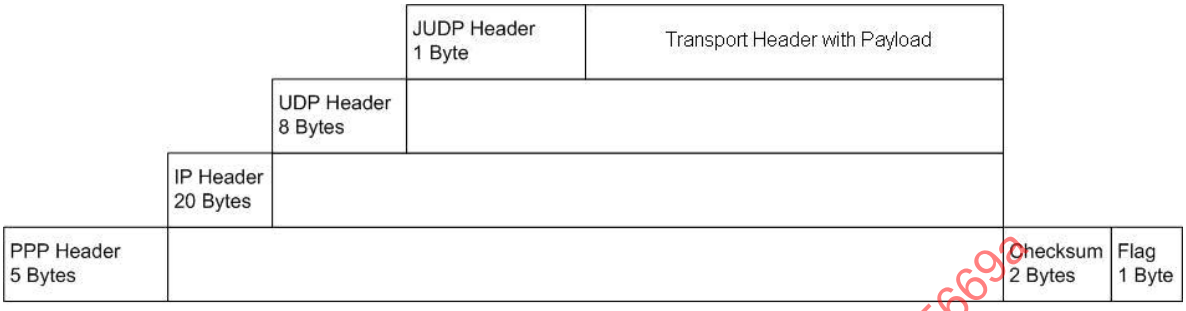


FIGURE 18 - JUDP TRANSPORT PROTOCOL STACK OVER PPP

Transmission of a message using UDP/IP via PPP encapsulation results in:

- An overhead of 36 bytes per packet – 5+20+8+2+1 (with the “5”, “20”, and “8” values coming from the corresponding headers, and the “2” and “1” being the checksum and flag at the end of the packet).
- An overhead of 1 byte per JUDP packet to accommodate the version number field that is only listed once at the beginning of the packet.
- An overhead of 14 bytes per message from the General Transport Header.
- A maximum of 51 bytes total overhead per message.

If a 9600 b/s link were to be used, use of this protocol stack would add approximately 34 msec transmission time to each packet sent.

Due to the increased overhead, the use of JUDP via PPP encapsulation for serial links is not recommended for use over low-bandwidth serial links. The use of other, lower-overhead transports, such as JSerial[10], is suggested when low-bandwidth serial links are employed.

6.1.7 Maximum Packet Size (MPS)

As previously discussed in 5.4, each transport standard must define a Maximum Packet Size for that standard. For JUDP, the MPS shall be defined on the uncompressed application data format to be 4101 bytes – including the JUDP header, General Transport Header, and uncompressed message payload.

Note that for most current IP networks[11], the MTU size will be less than the MPS size and so it is the MTU size that should be used when constructing outgoing packets.[12] However on the receiving end, the MPS size does yield a reasonable “maximum buffer size” for incoming JUDP messages.

[10] See Section 7.
[11] As of April 2007.
[12] See 5.4 for more detail on selection of the relevant MTU size, for heterogeneous networks.

6.1.8 Message Prioritization

To support the mechanism of message priority, JUDP implementations shall provide a priority-based message transmission scheme that assures that messages queued for sending will be sent in accordance with their assigned priority. Message priority shall be defined by the message priority field in the General Transport Header.

6.1.9 Compression

The compression scheme that shall be used in JUDP is that outlined in 5.1.

6.1.10 Integrity Mechanisms

JUDP implementations shall use the UDP checksum option. The UDP checksum spans the UDP datagram, as well as critical data in the enclosing IP datagram[13]; this provides detection of corruption. JUDP implementations rely on UDP checksum processing to assure that only non-corrupt data is delivered.[14]

UDP provides no notification of receipt of a datagram failing the checksum; UDP silently discards the corrupt datagram[15]. As a result, receipt of a failed datagram is non-detectable to the JUDP transport, and no action is taken. Since corrupt datagram receipt is indistinguishable from missed messages, recovery follows the same path as it would in handling a lost message.

6.1.11 Configuration

Commonly changed transport configuration parameters must be accessible for modification. A minimum set of accessible parameters, their default values, and accessibility requirements are described in the following subsections. Transport configuration parameters shall be accessible for re-assignment without rebuilding, reinstalling, or otherwise modifying the software operating the host node(s) involved. It is acceptable to require re-boot, re-start, or power-cycle of the node(s) for transport configuration changes to take effect.

6.1.11.1 UDP Port Number

The IANA has assigned port 3794 for UDP and TCP for use with JAUS messages. This port, whose port name is “jaus” (lower case), shall be used as the “primary contact port” for JUDP message transport.

The UDP Port Number is not expected to change; it is not required to be accessible for modification.

6.1.11.2 UDP Checksum

Use of the UDP Checksum is mandatory for JUDP; checksum use is not optional. The use of the checksum is therefore not regarded as a configurable transport parameter.

6.1.11.3 IP Addresses

IP addresses of all nodes shall be accessible for modification.

[13] [COMER95], section 12.4
[14] This assumes that the network frame provides a strong packet blockcheck. The most common network environments foreseen are Ethernet and PPP over serial links. Ethernet frames are protected by a 32 bit CRC, PPP packets by a 16 bit CRC. Thus, the UDP checksum is sufficient given encapsulation within the stronger frame/packet checks.
[15] [STEVENS93], section 11.3

6.1.11.4 IP Subnet Masks

Subnet masks shall be accessible for modification.

6.1.11.5 Broadcast/Multicast Addresses

The broadcast/multicast address used to implement broadcast semantics should be accessible for modification.

The default multicast address is 239.255.0.1 from the administratively scoped IPv4 multicast address space.

6.1.11.6 Multicast TTL Values

If using a multicast address to implement broadcast semantics, the TTL (time-to-live) parameter should also be accessible for modification.

The default multicast TTL value is 16.

6.2 JTCP : JAUS Over TCP

Applications may also require support for the transmission of messages without loss (albeit with consequently unbounded latency) in IP networks. The AS5669 Transport Specification supports the lossless transmission of messages over IP networks via TCP connections.

In the context of JAUS, this standard is known as the JTCP standard.

6.2.1 Characteristics of JTCP

JTCP provides reliable transport of message traffic between entities, but with unbounded latency. JTCP is also strictly a point-to-point transport that does not support broadcast semantics. The JTCP transport shall not preclude the use of other protocols or messaging schemes via TCP between the same nodes.

JTCP is a stream-based transport built atop TCP/IP, and intended for implementation as a wrapper around TCP/IP transport providing additional non-native services.

6.2.2 Application Domains

JTCP may be used in any application domain possessing adequate bandwidth to maintain acceptable system performance. It is unlikely, for example, that JTCP will be applicable to the Unmanned Undersea Vehicle (UUV) application domain.[16] Applicability must be determined by the responsible Systems Engineer.

Due to the overhead involved in establishing and maintaining a TCP connection, it is important for the System Designer to understand the appropriate usage of this transport. The crucial characteristic of this transport that must be observed is the unbounded latency aspect of the TCP stream. This makes the TCP transport an inappropriate transport for things like Emergency-priority messages – even though the transport is reliable, it is far more important in this case that it be quick – employing an Application Layer acknowledgment/resend mechanism if necessary.

Hence, the domain of JTCP is typically large messages that are loss-intolerant. For example, a large amount of map data might be an appropriate subject for a JTCP message – the latency constraints are not likely to be severe, and map data is something that most likely cannot tolerate loss.

[16] For definition and description of application domains and their communication characteristics, please refer to [AIR5645].

All Application Layer protocols must specify the messaging characteristic they desire:

- Unreliable, best effort, low latency: Use JUDP.
- Reliable, unbounded latency: Use JTCP.

Currently, all specified JAUS protocols fall into the first category, and use application layer acknowledgments to handle retry semantics when necessary.

6.2.3 Communications Environment

The JTCP standard supports at a minimum the same environments as are supported by JUDP – see 6.1.3.

6.2.4 Description

The following subsections provide a description of the structure of the JTCP stream, and of the protocol build-up of JTCP data for communication across Ethernet and over (sufficiently speedy) serial links via PPP. All JTCP messaging shall follow the packet format and semantics described in this section.

6.2.4.1 JAUS TCP/IP Stream Structure

The JTCP stream structure is outlined as follows:

- Transport Header: Native TCP already provides source and destination addressing. Therefore, the JTCP per-connection (stream) header need only specify the transport version.
- Per-Message Header: The General Transport Header given in Section 4 is added to each message in the stream.

Therefore, the structure of a generic JTCP stream is shown in Figure 19.

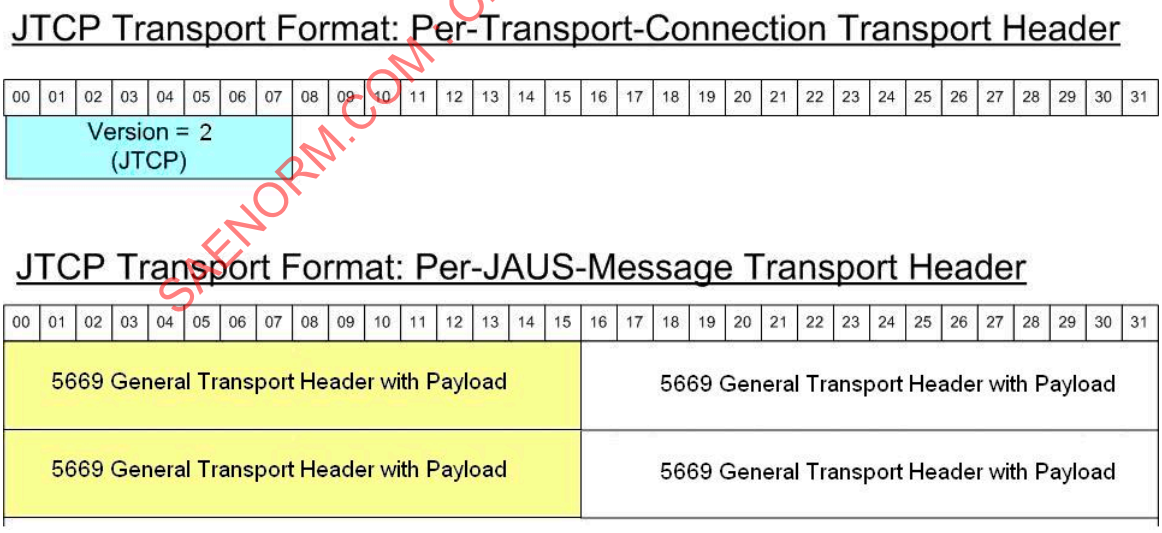


FIGURE 19 - JTCP STREAM FORMAT

6.2.4.2 JTCP Transport Header Fields: Version

The JTCP transport version is encoded in the first byte of the stream. This allows any receiver of the stream to determine compatibility. Any receiver of a JTCP message stream shall determine compatibility based on the Transport Version, and shall ignore and terminate any stream whose version (hence structure) is unknown to or unsupported by the receiver.

For this version of the JTCP Transport, the JTCP Transport Version shall be 0x02.

6.2.5 Implementation of Broadcast Semantics

JTCP does not support broadcast semantics – it is a point-to-point transport only.

6.2.6 Encapsulation

The JTCP transport must support at least the same two means of delivery as JUDP (see 6.1.7).

6.2.6.1 Ethernet

The following figure illustrates the progressive encapsulation of messages with the individual packets that comprise a JTCP stream for transport for TCP/IP across an Ethernet[17]

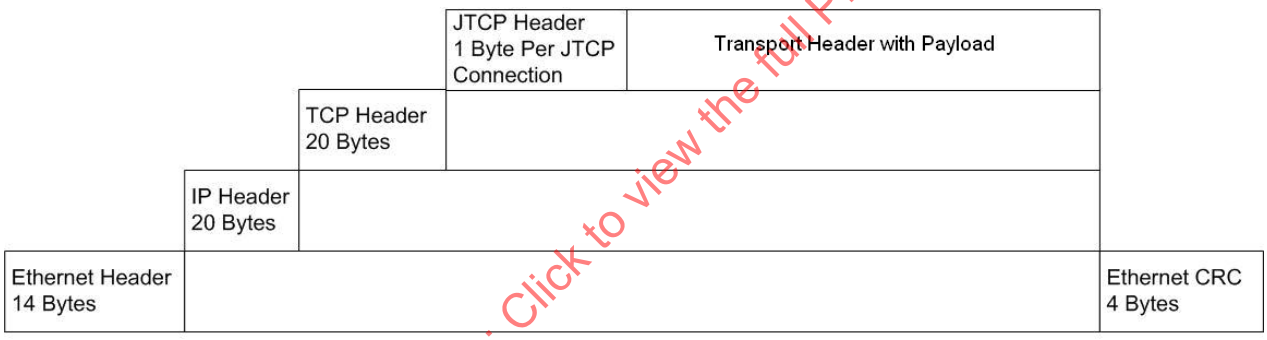


FIGURE 20 - JTCP TRANSPORT PROTOCOL STACK OVER ETHERNET

IEEE 802.2 / 802.3 encapsulation is similar, differing primarily in the addition of the 802.2 LLC and 802.2 SNAP. The differences between IP encapsulations within Ethernet and IEEE 802.2 / 802.3 frames are described succinctly in [STEVENTS93].

This results in:

- An overhead of 58 bytes per packet of the connection – 14+20+20+4 (with the “14”, “20”, and “20” values coming from the corresponding headers, and the “4” being the ending CRC on the packet).
- An overhead of 1 byte per connection for the JTCP version number that is only listed once at the beginning of the TCP stream.
- An overhead of 14 bytes per message from the General Transport Header.

This overhead is not prohibitive given the relatively high available bandwidth typical of Ethernet implementations.

[17] As described in RFC793.

6.2.6.2 PPP

The use of IP encapsulation techniques for transmission of TCP streams over serial links is inefficient of bandwidth. For example, transmitting a message using TCP/IP transport via PPP encapsulation over a serial link results in the following protocol buildup:

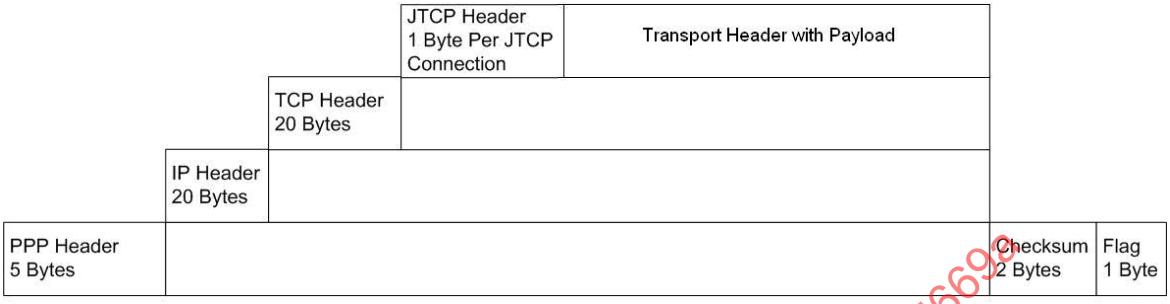


FIGURE 21 - JTCP TRANSPORT PROTOCOL STACK OVER PPP

Transmission of a message using TCP/IP via PPP encapsulation results in:

- An overhead of 48 bytes per packet of the connection – 5+20+20+2+1 (with the “5”, “20”, and “20” values coming from the corresponding headers, and the “2” and the “1” being the checksum and the flag at the end of the packet).
- An overhead of 1 byte per connection for the JTCP version number that is only listed once at the beginning of the TCP stream.
- An overhead of 14 bytes per message from the General Transport Header.

If a 9600 b/s link were to be used, use of this protocol stack would add approximately 53 ms transmission time to each packet sent – and that is not counting the TCP packets in the handshake used to initially establish the connection.

Due to the increased overhead, the use of JTCP via PPP encapsulation for serial links is not recommended for use over low-bandwidth serial links. The use of other, lower-overhead transports, such as JSerial[18], is suggested when low-bandwidth serial links are employed.

6.2.7 Maximum Packet Size (MPS)

For JTCP, the relevant parameter is actually the maximum number of bytes that may be pumped down any single JTCP stream. Since TCP connections are generally written to handle an inbound connection providing an unknown, potentially large number of bytes, and since TCP connections are explicitly closed, there is no need to be restrictive here.

6.2.8 Message Prioritization

The usage of message priority data in the JTCP transport is the same as it is in the JUDP transport – see 6.1.8.

6.2.9 Compression

The compression scheme that shall be used in JTCP is that outlined in 5.1.

[18] See Section 7.

6.2.10 Integrity Mechanisms

JTCP implementations already have a checksum built-in to the TCP standard. This provides detection of corruption. JTCP implementations rely on TCP checksum processing to assure that only non-corrupt data is delivered.[19]

TCP will automatically handle packets that fail the checksum by discarding them and initiating a resend. This will obviously add to the latency of the JTCP stream, but no other effect is seen.

6.2.11 Configuration

Commonly changed transport configuration parameters must be accessible for modification. A minimum set of accessible parameters, their default values, and accessibility requirements are described in the following subsections. Transport configuration parameters shall be accessible for re-assignment without rebuilding, reinstalling, or otherwise modifying the software operating the host node(s) involved. It is acceptable to require re-boot, re-start, or power-cycle of the node(s) for transport configuration changes to take effect.

6.2.11.1 TCP Port Number

All entities that support services/protocols whose messaging requires the kind of reliable delivery that TCP ensures support TCP connections on the same port(s) as their UDP connections.

6.2.11.2 IP Addresses

IP addresses of all nodes shall be accessible for modification.

6.2.11.3 IP Subnet Masks

Subnet masks shall be accessible for modification.

6.3 Use of Non-Standard Ports in JUDP and JTCP

IP infrastructure supports the use of multiple ports – the port assigned to JAUS by IANA is 3794. Having paid the price of the overhead entailed in an IP-based network, provisions should be made to “get the good out of it” – that is, to take advantage of the efficiencies and functionalities that IP-based networks provide.

Therefore, in order to support efficient message routing for IP-based networks, the JUDP and JTCP transports do not restrict all JAUS processes to the use of the 3794 port only. Instead, an entity within a JAUS system that constitutes the end of a communications link (for IP-based networks, this would be an IP address) must support at least one JAUS process, and that process must be on the 3794 port. The entity may, however, support additional processes on additional ports. Once the configuration of an IP-based JAUS system is known (either a priori or through a discovery protocol), packets can be routed to the correct port numbers.

6.3.1 Messaging to/from Non-Standard Ports

All entities shall expect to receive incoming packets on the same port from which they send outgoing packets.

6.3.2 Broadcast Propagation

Discovery protocols usually make use of some kind of broadcast semantics in order to provoke responses from the things they are seeking out. Without a priori knowledge of non-standard port numbers, a discovery protocol’s only option is to use the standard 3794 port.

[19] This assumes that the network frame provides a strong packet blockcheck. The most common network environments foreseen are Ethernet and PPP over serial links. Ethernet frames are protected by a 32 bit CRC, PPP packets by a 16 bit CRC. Thus, the TCP checksum is sufficient given encapsulation within the stronger frame/packet checks.

Therefore, all broadcast packets (whether implemented via multicast or “true broadcast”) shall be destined for the 3794 port; it is the responsibility of the processor that resides on the end of the communications link to ensure that any broadcast packets are correctly routed both internal to itself and (if the processor is a multi-homed host) for any entities that sit behind the interface being broadcast to (IP-forwarding and other routing techniques).

See Appendix A.2 for examples of techniques for using non-standard JAUS ports.

6.4 IP Infrastructure

Most IP networks contain infrastructure intended to facilitate the efficient operation of that network. Where such infrastructure exists, entities should “do the right thing”.

For example, if IP addresses are handed out via a DHCP server on the network, then entities should acquire their IP addresses from that DHCP server. If a network hosts an mDNS/DNS-SD (ZeroConf[20]) or SLP[21] server, then entities should register with that server.

The exact infrastructure on any particular IP network is of course a design decision to be taken by the responsible System Engineer.

7. JSERIAL: JAUS OVER SERIAL

The AS5669 Transport Specification supports the transmission (both lossless and lossy) of JAUS messages over serial links via the specified serial protocol.

In the context of JAUS, this serial transport protocol standard is known as the JSerial standard.

7.1 Characteristics of JSerial

The JSerial standard specifies transport of JAUS message traffic between software entities on distinct nodes. The standard specifies both:

- Non-reliable best-effort transport.
- Reliable transport with unbounded latency (and no broadcast).

7.2 Application Domains

The JSerial transport is primarily intended for AS5669 transport support of bandwidth-constrained serial communications links.

7.3 Communications Environments

The JSerial transport supports both full-duplex serial links and half-duplex serial links, where the design of the overall system indicates that the half-duplex link will truly only need to support unidirectional message flow.

7.4 Description

JSerial is a packet-based transport. The JSerial packet includes message and header error checks. The transport assures that messages delivered are non-corrupt. JSerial does provide transport-level ack/nak flags to allow recovery attempts by retry for corrupt received packets and detection of missed packets. JSerial does not assure in-order delivery of received messages. JSerial is a data-transparent transport supporting variable data length packets. All JSerial messaging shall follow the packet format and semantics described in this section.

[20] See [CHESH05], which is also known as Apple’s Bonjour protocol.
[21] See [RFC2608].

7.4.1 JAUS Serial Packet Structure

Each message packet is structured as one or more messages encapsulated in a JSerial wrapper, with header and body of the serial message delimited by the appropriate digraphs. The JSerial packet structure may be outlined as follows:

- The Per-Transport-Packet Transport Header:
  - Version: The JSerial per-packet header must specify the JSerial transport version.
  - Retry Flags: These flags are used to specify the desired retry semantics.
  - Packet Length: The overall length of the data that follows between the DLE-STX and DLE-ETX, not including any of the packet digraphs or checksums, and without taking DLE-insertion into account. (This is also known as the message payload.)
  - Addressing: If your serial link is a point-to-point link or is otherwise providing its own transport-layer addressing (inherent or otherwise), then those fields are left out of the Transport Header. If no such addressing is provided, then source and destination address fields must be included.
- The Per-Message Header: The General Transport Header wraps each message payload.
- The Packet Boundary Markers: These are the <DLE-SOH>, <DLE-STX> and <DLE-ETX> digraphs that are included in the byte stream to demarcate where various parts of the packet are.
- The Packet Checksums: These are the checksums immediately following the <DLE-STX> and the <DLE-ETX>. The first checksum is intended to ensure that the header information is correct – in particular, that the value for the length of the data contained in the body of the packet is correct.

Therefore, the structure of a generic JSerial packet is shown in Figure 22 and Figure 23, below.

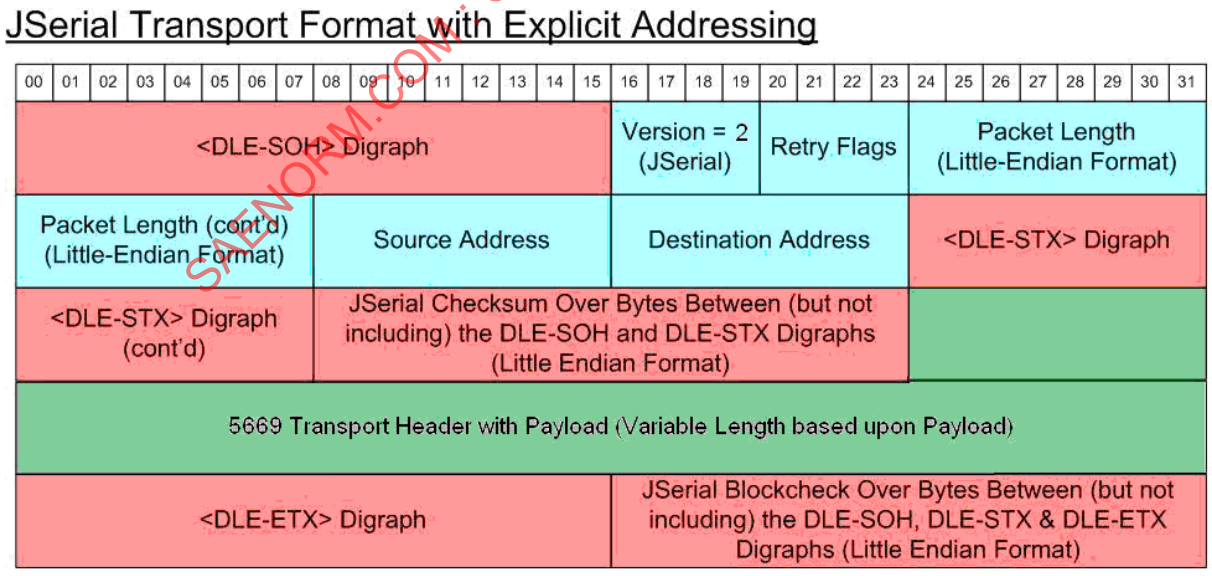


FIGURE 22 - JSERIAL PACKET STRUCTURE WITH EXPLICIT ADDRESSING

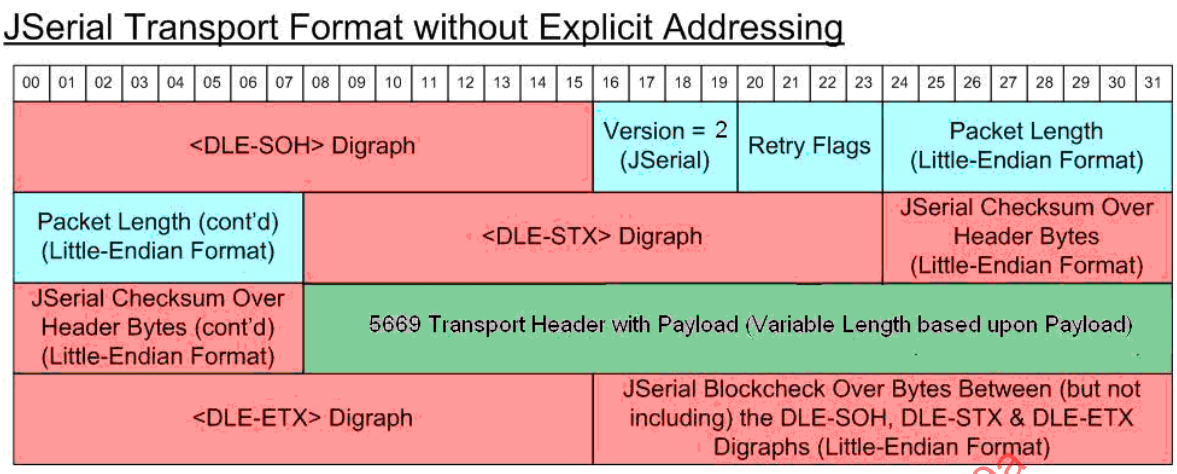


FIGURE 23 - JSERIAL PACKET STRUCTURE WITHOUT TRANSPORT EXPLICIT ADDRESSING

The "Payload" portion comprises a single General Transport Header (see Section 4) and message.

Remember that we may pack multiple messages into a single serial packet – as is shown in Figure 24.

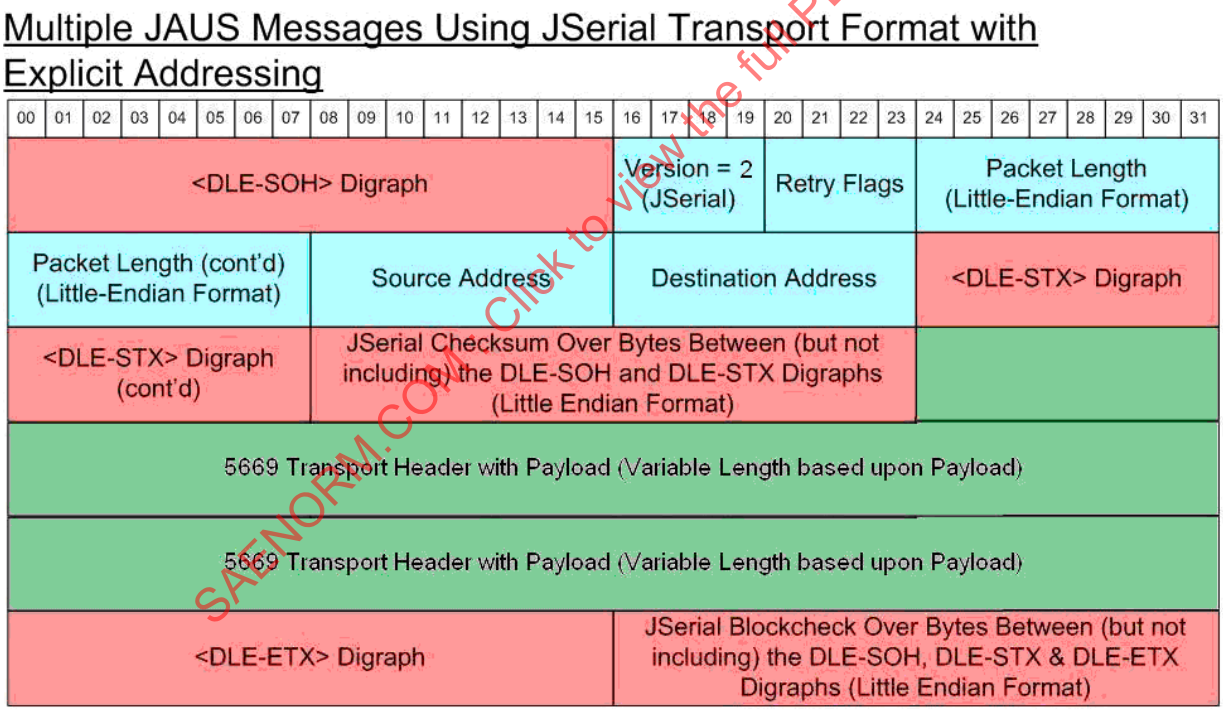


FIGURE 24 - MULTIPLE MESSAGES IN A SINGLE JSERIAL PACKET

The header compression and Message Length fields have the same semantics and functionality as they do in the JUDP standard. The main difference between the JSerial and JUDP packet formats is the inclusion of the Retry Flags field in the per-packet header. This field is defined in 7.4.2.2.

7.4.2 JSerial Transport Header Fields

The fields of the JSerial header are defined below.