DOF Protocol Specification

DOF Protocol Specification

Technical Steering Committee

DOF Protocol Specification: DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2008-1
Document Version	7.0.1, 12 January 2018
Build Version	7.0.1.8
Source Information	2018-01-12, "." at commit c4a21c8bf741a32d4fe20776861d03baab74639a on branch "master" of repository core-specifications
External Reference	DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018)

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc.. Contact the committee using the following information.

Contact Information

Post Mail	OpenDOF Project, Inc. Technical Steering Committee 3855 SW 153rd Drive Beaverton, OR 97003
Telephone	1-503-619-4114
Fax	1-503-644-6708
Email	tsc@member.opendof.org [mailto:tsc@member.opendof.org]

Copyright

Patents

Contributors to the OpenDOF Project make a non-enforcement pledge related to any patent required to implement the protocols described in this document. For more information see https://opendof.org/pledge.

License

Table of Contents

List of Figures

List of Tables

Specifications and PDUs

1. About This Document

This document contains information about the DOF Protocol Stack. This is critical information for anyone who needs to work with the protocols themselves, and contains useful background information for those interested in the architecture and inner workings of DOF systems.

This information is highly technical. This document serves as a specification and reference against which someone could implement networked software that interoperated with DOF products without using existing libraries.

1.1. Audience

The primary audience of this document is people implementing libraries that use DOF protocols. Others that are concerned about how DOF protocols work on a network at a very low level can also benefit from this document.

Readers should be familiar with technical protocol documentation. This document is similar to an 'RFC' for DOF protocols, and familiarity with the language used in these types of documents is helpful.

This document is *not* required reading for those who need to use existing DOF libraries, although system designers may benefit from an understanding of this information. In particular, the information related to security and connections can help system designers to better understand how DOF systems work.

1.2. Authority

This document, in its English form, is the authoritative document for DOF protocols. Due to the detailed nature of protocol design, it may be difficult for a reader to implement these protocols correctly based on a translated document. When questions arise, the English version is authoritative.

All implementations that claim to be DOF compliant must satisfy the requirements set forth in this document.

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc., referred to as 'ODP-TSC'. The inside cover of this document contains contact information for the Technical Steering Committee.

The web site for the Technical Steering Committee is https://opendof.org/tsc.

1.3. PDU Definitions

This document deals with the transmission of information on a network. In order to describe this, diagrams show different fields, their network order, and positioning.

All DOF protocol documentation uses a similar diagramming style, discussed in this section. The word 'PDU', which stands for 'Protocol Data Unit', indicates these diagrams.

1.3.1. Context

Each PDU represents information sent on the network in some context. This usually means that there is information sent 'surrounding' the PDU itself. This information provides the 'context' for the PDU.

For example, the context for the DOF Protocol Stack itself is a network transport. This could be UDP/IP, TCP/IP, or any other transport. The transport will require certain headers and trailers, but they are not shown in each DOF Protocol Stack PDU. In this example the transport provide the context of the DOF Protocol Stack PDU.

In a similar way, the DOF Protocol Stack defines the context for the other DOF protocols. The Object Access Protocol, for example, can use the DOF Protocol Stack for its context. Just as the transport headers are not shown for the DOF Protocol Stack, the DOF Protocol Stack headers are not shown for the Object Access Protocol.

Understanding the context requires understanding the relative position of the different layers in the protocol stack. Each PDU defines the ordering of all of its parts, and may indicate that it is itself an instance of some other general PDU format. This means that it is always possible to start at a high-level PDU definition and understand it completely, but not always possible to identify everywhere that a particular PDU may be used (as a part of some other higher-level PDU).

Each protocol specification defines its specific contexts. The context definition includes a description of the underlying stack layer (or the containing layer), and defines the information that must be available from that layer (for reception) or passed to that layer (for transmission).

As an example, the context of the DOF Network Protocol is a transport. The transport has a requirement to provide the addresses of nodes, and so part of the context for the transport is the sender address (for reception) or the target address (for transmission). Any transport that will work with the DOF Protocol Stack must meet the context requirements.

1.3.2. Generalization

Many times a particular PDU will be a specific type of a more general PDU. In these cases, the PDU will indicate that it is an 'instance' of the more general type.

1.3.3. Qualifications

Each PDU will have qualifications placed on its use. The general qualifications include security and transport requirements. The following terms identify these qualifications. Others may be present as well.

1.3.3.1. Transport Qualifications

There are three major transport categories for DOF protocols (see the *DOF Protocol Specification*, Transport Requirements). These are None, Lossy, and Lossless. There are also three addressing types: Unicast, Multicast, and Broadcast. PDUs indicate the ability to use each of these categories as follows:

```
Session: Combinations of None, Lossy (2-node, n-node), or Lossless.
Addressing: Combinations of Unicast, Multicast, or Broadcast.
```

1.3.3.2. Security Qualifications

Each PDU categorizes security qualifications according to the major aspects of security (see the *DOF Security Specification*, Overview). These are Encryption, Data Integrity, Authentication, and Access Control. Encryption prevents viewing of the PDU, Data Integrity ensures that others do not modify the PDU, Authentication indicates knowledge of the nodes involved in the communication, and Access Control restricts those able to use the PDU.

Data Integrity is required according to the DOF Protocol Stack specification for all secure packets. Authentication is required whenever permissions are involved, or when it is important to know something about the relationship between the sender and receiver. Encryption is required if the contents of the packet contain information that must not be visible to potential attackers.

In addition, there is the case of unsecured communication. PDUs that do not require security will list 'Unsecured' as a possible choice. Finally, certain PDUs are required to be Unsecured, and are identified as such.

Each application PDU must specify its security requirements. If Access Control is required, the PDU will indicate the permissions required for the command and for any response. However, it is typical for all of the PDUs of a given protocol to share the same requirements, and so the protocol specification may define its security qualifications in a single section. In this case, the individual PDUs should refer back to this section to avoid confusion.

It is also typical for applications to require the security of the session. A PDU indicates this as 'session security', and is its own security qualification. If the PDU allows the 'None' session type then this implies that Unsecured is allowed, as session type 'None' cannot be secure.

For PDUs that refer to session security or a common security section, one of the following formats is used:

```
Security: See the section 'Security Qualifications'.
```

or

```
Security: Session.
```

The PDU uses the following identifications to specify the security qualifications for a PDU:

```
Unsecured: (description).
Encryption: (description).
Message authentication: (description).

Permission (command): (description).
Permission (response): (description).
```

For each of these, the (description) varies. There are many different combinations of values, as described here.

Unsecured can have the values 'Required', 'Allowed', 'Not Allowed'.

Encryption can have the values 'Required', 'Allowed', 'Not Allowed'.

Message authentication can have the values 'Required', 'Allowed', 'Not Allowed'.

The specific PDU determines the permissions required (for both command and response).

There are combinations that are not valid. The following table defines all allowed combinations if a combination does not appear in the table then it is invalid.

In general, security theory links Encryption and Data Integrity. If one is 'Required' then the other must at least be 'Allowed'. Encryption and Data Integrity are the opposite of Unsecured. If either is 'Required' then unsecured must be 'Not Allowed'. Again, note that since data integrity is a common requirement for all secure packets the PDUs do not list it separately.

Unsecured / Encryption / Message Authentication	**Description**
Required / Not Allowed / Not Allowed	Indicates the PDU must be unsecured.
Allowed / Allowed / Allowed	Indicates there are no security requirements.

Unsecured / Encryption / Message Authentication	**Description**
Not Allowed / Allowed / Allowed	Indicates the PDU must be either encrypted or authenticated. This is the lowest level of security possible for a PDU.
Not Allowed / Required / Allowed	Indicates that encryption is required, and data integrity is optional.
Not Allowed / Allowed / Required	Indicates that encrypted is optional, but authentication is required.
Not Allowed / Required / Required	Indicates both encryption and authentication are required.

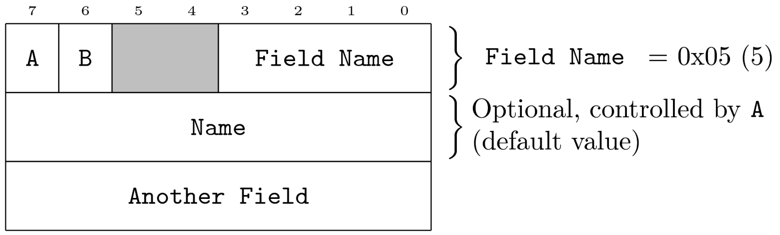
1.3.4. Fields

Each PDU shows a container, represented as a sequence of fields. The table fully describes each field.

The following is an example of a PDU description.

Example PDU$_{\texttt{example-pdu-1}}$

PDU Context



`A` **One bit.** Controls the presence of `Name`.

`B` **One bit.** Field description.

`Field Name` **Type and size description.** Field description.

`Name` **Type and size description. Optional, controlled by** `A`. Field description.

`Another Field` **Type and size description.** Field description.

The container represents the PDU. The diagram shows Fields first to last, left to right and top to bottom. This means that the first fields appear on the wire before the last fields. This is a general principle: things on the top and toward the left in the diagrams appear on the wire before things on the right and bottom. Note that the transport is free to use whatever ordering it must, and memory layout may different from the PDU definition. However, the transfer of a PDU from a program, over an arbitrary transport and into another program must maintain the perceived ordering between the two programs. PDU diagrams are always shown MSB first (most significant byte) and msb first (most significant bit) unless otherwise indicated.

Each field identifies its contents and provides information necessary to understand its use. In the example above, the diagram shows the first six fields of the container. Each field is named, with the exception of reserved fields which are indicated as being grayed out. Field names use `monospace text` similar to that used in program listing. The third field shown is indicated as reserved. Reserved bits have special requirements described in the *DOF Protocol Specification*.

Several different field cases are typical, described in the following sections.

1.3.4.1. Typical Field

A typical field defines its name, its type, and its description. As shown, this field is not optional, and so it must appear in the position indicated by the diagram. The PDU uses `Field Name` throughout the related documentation to refer to the field. The type of the field is a reference to some other defined PDU. Otherwise, the PDU indicates the size of the field and any defined structure.

1.3.4.2. Optional Field

An optional field is similar to the typical case, but is indicated as optional in a note to the right. Each optional field will indicate what controls whether or not it is included as well as the default value that the field takes on when it is absent.

1.3.4.3. Fixed Values

Fields may have required values based on the specific PDU. This is common in the case of a PDU being an instance of another PDU, where the specific instance requires certain field values. In this case, it is not valid for the PDU to contain values other than that specified, if so the PDU is invalid.

The required value is shown to the right of the field as a comment.

1.3.4.4. Bit Fields

It is common for PDUs to leverage a single byte to store multiple fields (bit fields). The packing of multiple fields into a single byte is indicated as shown in the earlier example. This packing of bits may also leave space that can be indicated as reserved.

The PDU indicates reserved bits visually although the PDU does not individual name these fields.

Note that bit fields may specify default values just as other fields. However, single bit fields do not show their hexadecimal representation. In addition, bits without names that have required values may indicate their values in the place of the name.

1.4. Specifications, Notes and Warnings

This is a technical specification document. It is critical to understand how the document indicates specification items. There are also annotations for notes (implementation notes or other information that is worthy of extra notice) and warnings (pitfalls or extremely critical information). In addition, there are implementation notes, which point out optimizations that are not obvious.

Each type of callout ends either when another callout begins or when the text goes back to the normal indentation.

ℹ This is a note. The text is inset and there is a header to indicate the beginning of the text.

💡 This is an implementation note. The text is inset and there is a header to indicate the beginning of the text.

⚠ This is a warning. The text is bold inset and there is a different header to indicate the beginning of the text.

> This is an example$_{\texttt{example-spec-1}}$
>
> This is the text.

Following the summary is descriptive information about the specification item. Each specification item has a unique number assigned.

1.5. Printing This Document

The format of this document is for two-sided printing on US Letter paper.

1.6. Comparing Documents

The easiest way to compare versions of this document is to compare the corresponding source. The source format is AsciiDoctor, which is a markup-style text format. This format can be compared using any text comparison tool, including those that work with source control documents. To facilitate determining the source for each document there is information about the commit, branch, and repository along with the document version information. In addition the following guidelines are followed to make things easy to compare:

1. The input files put each sentence on a separate line, with no hard line-wrapping.

2. To the extent possible, macros and templates are used to determine formatting and structure.

3. Graphics and diagrams also use text formats.

If all you have available is a compiled document (PDF or HTML) then there are tools available to compare them. This section summarizes some of these methods with their pros and cons.

1.6.1. Text Comparison

There are several free tools that will do comparisons of the text of a PDF. One of these is the xdocdiff plugin for WinMerge. The plugin extracts the text from the PDF (including some information like page numbers), and WinMerge then compares the text.

WinMerge is available from http://winmerge.org. The xdocdiff plugin is available from http://freemind.s57.xrea.com/xdocdiffPlugin/en/index.html.

1.6.2. Graphical Comparison

In order to compare formatting, or to see the changes in the context of the document, a graphical compare is required. There are several free tools, but each has limitations.

1. diff-pdf, available from https://github.com/vslavik/diff-pdf, shows an overlay of the matching pages in different colors.

2. DiffPDF, available fromhttp://www.qtrac.eu/diffpdf.html, shows side-by-side red-lined differences of matching pages.

Both of these tools suffer from their page-by-page comparison methods. This means that when context shifts because of additions or removals that they start sensing changes at each page boundary. DiffPDF has the ability to select comparison ranges, allowing the program to synchronize at the beginning of each section, but this requires manual configuration and can be difficult for large documents.

There are also commercial tools that do an excellent job at comparing PDF documents. The first is Adobe Acrobat (http://www.adobe.com/products/acrobat.html), which in later releases has a 'Compare Document' feature. The result of the comparison is an annotated PDF, and hovering the mouse over the change will show the details.

Another commercial tool is PDF Content Comparer (http://www.inetsoftware.de/products/pdf-content-comparer) from i-net Software. This tool does side-by-side comparison, but has an intuitive auto-synchronize that keeps similar content lined up even when additions or deletions have occurred.

2. Overview

This document discusses the protocols and theory behind the Distributed Object Framework, or DOF.

The descriptions of DOF protocols are based on the OSI protocol vocabulary. The target systems (possibly embedded microprocessors) on which these protocols are implemented necessitate some optimizations and other modifications to that standard.

This document discusses the common aspects of the DOF Protocol Stack. It does not cover the details of the various DOF application protocols. Application protocols each have separate specification documents with the exception of the DOF Session Protocol (DSP) which is covered in this document.

There are three main categories of DOF protocols. The first are application protocols. These protocols sit at the top of the protocol stack, and are the protocols that most engineers think of when they think of protocols at all. Examples of standard application protocols are FTP (File Transfer Protocol), HTTP (Hyper-Text Transfer Protocol), Telnet and SMTP (Simple Mail Transfer Protocol).

Five application protocols are critical to DOF systems:

1. DOF Session Protocol (DSP). This protocol allows connected nodes to negotiate options for the protocols that they will use (including which protocols they will use).

2. Object Access Protocol (OAP). This protocol allows a device to present Properties, Events, Methods, and Exceptions to clients.

3. Ticket Request Protocol (TRP). This protocol manages key distribution.

4. Ticket Exchange Protocol (TEP). This protocol manages security and access control.

5. Secure Group Management Protocol (SGMP). This protocol manages secure groups.

Application protocols require the services of other protocols to do their work. These other protocols form the remaining two main categories. The first are the typical stack layers:

1. DOF Presentation Protocol (DPP). This protocol handles encryption and defines which application protocol is being used.

2. DOF Network Protocol (DNP). This protocol allows for encapsulating other DOF Protocols over a variety of transports.

The final main category contains the encryption modes. These are both application protocols and can appear in other stack layers – they define the behavior the DOF presentation protocols when used securely, and they may contain application PDUs.

Of course there are other standard networking protocols that are used in DOF solutions. A primary example is TCP/IP.

2.1. The OSI Model

The OSI model for network protocols looks like this.

Standard OSI Layers.

Each layer in the protocol relates to three different layers: the layer above it in the stack, the layer below it and (virtually) to the corresponding layer in the receiving stack.

While the OSI layering scheme is a powerful abstraction, it is not realistic to implement a fully abstracted stack on an embedded platform, and in fact, many full operating systems combine at least some of the layers. Even so, putting different functionality into 'layers' and defining a protocol between those layers can achieve the goals of modularity and clarity.

With this background, it is useful to identify the specific requirements that the network protocol has and assign them to layers.

2.2. Network Layering

In order to consolidate network features into manageable groups, designers group protocols by the functionality they provide. In many cases, these layers are *logical*, meaning that there is no externally defined API for a particular layer but its functionality represents a "meta-layer" that provides the functionality of many logical layers. This simplifies implementation and reduces required header size. Defined layers should have an associated API.

This document does not describe the standardized layers other than to identify that in DOF protocols several of the layers combined as described.

2.3. Control Authority

The DOF protocols are controlled by the Technical Steering Committee of the OpenDOF Project, Inc. (ODP-TSC).

The ODP-TSC manages the DOF Protocol Stack specification, including all its layer and application protocols. Prior approval is required for all changes to the protocols, or additions to the protocols.

2.4. Summary of Externally Assigned Numbers

DOF protocols use several assigned numbers from different standards bodies. This section summarizes these numbers. Assignments that are associated with a specific transport, like Internet Protocol (IP) are in the specification document associated with that transport.

2.4.1. SMI Private Enterprise Number

The IANA has assigned the decimal number **4561** for DOF protocols by IANA. See the IANA site [http://www.iana.org/assignments/enterprise-numbers] for more information.

2.4.2. ETHERTYPE

IEEE has assigned the hex number **0x8876** is reserved for DOF protocols. See the IEEE site [http://standards.ieee.org/regauth/ethertype/type-pub.html] for more information.

2.4.3. PPP

The IANA has assigned the hex number **0x4027** for DOF protocols. See the IANA site [http://www.iana.org/assignments/ppp-numbers] for more information.

3. Transport Requirements

This document uses the word 'transport' to represent the underlying network that is used to communicate DOF protocols and the software that provides that network. A fundamental goal of DOF specifications is to be transport agnostic, meaning that almost any transport can be used. This section discusses common transport terms and DOF transport requirements. Anyone implementing an DOF transport should be very familiar with the terms and requirements of this section.

The function of a transport is to exchange information between nodes in a network. In doing this it will leverage transport addresses to identify the endpoints of the communication. Further, it will typically exchange information in a logical unit called a datagram.

Throughout this document, the word 'datagram' refers to this unit of information transfer. It is equivalent to a buffer of data with an associated length. Related to a datagram is a PDU, or Protocol Data Unit. This document uses the word 'PDU' to refer not just to the datagram, but also to the definition of the contents of the datagram. The term datagram appears extensively in network terminology, and any confusing uses of the term (those that do *not* refer to a unit of information transfer) will be qualified.

There are several terms related to transports defined in this section. Each of these terms appears in networking literature, and even the standard definitions can be confused. Throughout all DOF documentation the following definitions will be used, even if they are slightly different than accepted industry definitions. Further, many of these terms are important for transport implementations, but not critical to the definition of the DOF protocols. Terms that are used by the DOF protocols are defined as part of the 'context' at the end of this section. The context defines the information passed from the transport to the stack.

The following terms also appear later in this section if more discussion is required:

Broadcast

A type of datagram where all nodes on a network are receivers. Broadcast datagrams are always associated with receiving servers. The term broadcast is also used in the context of a session, and in this case, it means that the datagram is sent to all nodes in the session. However, this second use of the term usually relates to application, rather than transport, behavior.

Client

The node that takes the first action in establishing a session, or the node that sends a datagram to a server. Note that this definition refers to a particular action, not the behavior of an application as a whole. It is common for peer-to-peer systems to have nodes that are both client and server at the same time from an application perspective.

Connection

A 2-node session monitored in such a way that notification will occur if the connection is 'broken' (meaning that datagrams can no longer be transferred). A session converts into a connection through monitoring, whether done by the operation system, library, or application. DOF protocols do not require connections, although they are typically provided (if requested) by most transports. In a layered protocol stack, once a session converts to a connection it appears as a connection by all higher levels of the stack. For example, a TCP transport provides a connection to the lowest DOF stack layer. Once a TCP connection is established (using a TCP server), DOF protocols do not need to do anything in order to maintain the connection. If the connection is broken (due to network failure or other reasons) then the DOF implementation will be notified (typically by the operating system or TCP implementation).

Datagram Transport

A transport type that exchanges datagrams from node to node, preserving the boundaries of each datagram. DOF protocols are based on datagrams, and so it is natural to use a datagram transport.

In-Order

A type of datagram delivery where the order in which datagrams are sent is guaranteed to be the order in which they are received. It is possible to have in-order and lossy, preserving order even if some datagrams are lost. Usually, in-order transports are lossless.

Lossless

A lossless transport is one in which the nodes are guaranteed to receive each datagram sent (unless the transport fails).

Lossy

A lossy transport is one in which the nodes are not guaranteed to receive each datagram sent.

Multicast

A type of datagram that designates a set of nodes as receivers. Multicast datagrams are always associated with receiving servers.

Orderless

A type of datagram delivery where the order in which datagrams are sent is not guaranteed to match the order in which they are received.

Server

Part of the transport on a node that is able to send and receive datagrams, and establish transport sessions and connections. Its ability to send datagrams may require additional state (such as a destination transport address). Not all servers provide all of these capabilities, however. For example, a typical TCP server will not accept a datagram, cannot be used to send a datagram, and will only establish connections. Servers are associated with a specific transport address that is the target address for packets sent to the server and the address used to request sessions/connections from the server.

Session

A relationship between a set of nodes where each node maintains state regarding the other. Sessions can exist at a variety of different levels (transport, security, application, etc.), and so the use of this term may be ambiguous. In this case the use should be clarified by indicating the type of session. For example, a UDP transport provides a server or session to the DOF implementation. State is provided that would indicate the nodes involved in the session, but maintaining any sense of continuity (providing a connection if one is required) is the responsibility of the DOF implementation in this case. In many cases the DOF implementation requires a session for certain behavior, relying on the shared state available to both nodes. Sessions typically refer a set of 2 nodes (the client and the server). However, it is possible to have *n*-node sessions. In this case all of the nodes attempt to maintain state. A lossless session will be able to track shared state exactly, where a lossy session may not be able to synchronize state exactly.

Streaming Transport

A transport type that does not preserve datagram boundaries from node to node. DOF protocols are based on datagrams, and so in the case of streaming transport the specification must include additional information to preserve datagram boundaries.

Unicast

A type of datagram where a single receiver is identified.

There is a difference between lossy/lossless and in-order/orderless. However, this document treats them as the same, and focuses on the lossy/lossless aspect. This means that lossy implies orderless, and lossless implies in-order throughout this document. Further, the property of streaming/datagram is of minor importance to the DOF implementation, and so it is not discussed in detail. The DOF Network Protocol (discussed later) is responsible for converting streaming transports back to datagram. Finally, with regards to state, there are three possibilities: none (no shared state, or state on only one node and not the others), session, and connection. DOF implementations rarely cares about the difference between session/connection, and so the focus will be between none and session.

Excluding the addressing types, these variations combine to form four different relationships: none/lossy, none/lossless, session/lossy, session/lossless. Of these four, the combination of none/lossless is ignored (it is virtually impossible to guarantee no datagram loss without some shared state). This leaves three primary categories of node relationships that are of concern.

Note that at the transport there may be many different combinations that can be considered equivalent to these three. This discussion relates to how DOF implementations view the transport, and what impact the different transport properties have on DOF protocol specifications.

A datagram is received from the transport in one of two ways: on a session (of any type, including through a server), or through a server with no session. In the first case the session alone may be enough to identify the transport addresses used by both the sender of the datagram and the receiver of the datagram (if the session is between two nodes, each with a transport address). Further, in this first case where the session is between 2 nodes only a reference to the session itself is required in order to send a datagram back to the original sender. In the case of no session or a multipoint session, the datagram itself is associated only with the server it was received on and the transport address of the sender. It is possible to send a datagram back to the sender, but doing so requires that the transport be given a server, any session, and the transport address of the destination.

In order to send a datagram on an DOF transport the transport must be given either a session, or a server along with a transport address (and possibly a session).

In a true distributed peer-to-peer system, almost every node in the system is both a server and a client. This means that most nodes have the ability to initiate or respond to a session. It is important to understand the typical implementation guidelines for these types of nodes, and to define the expected behavior of a server and client. This is one purpose of this section of this document.

DOF architecture imposes several requirements on clients and servers. These requirements are independent of transport, but do depend on the specifics of the session or connection provided. The following assumptions are made by the architecture. Failure to meet these assumptions will result in application failures, although depending on system design these may be expected and acceptable.

DOF implementations assume:

1. It is possible to send a response to the sender of a datagram (the client). Without this capability no responses could be received by the client. This is a very typical feature of all transports, and so is a safe assumption. DOF implementations further assume that for datagrams received by servers, a response may originate from a different server (transport address) than the server that received the datagram. This assumption is more complicated, and is discussed further later in this section.

2. If the server allows lossless sessions to be created, then that server can be identified by the transport address of a lossy response datagram. This assumption forms a link between the transport address of the lossless server and the transport address of the source of the lossy response datagram. This assumption only indicates that such a server *may* exist; it does not mandate that it *must* exist.

3. If the server allows lossless connections to be created, then that server can be identified by the transport address of a lossy command (non-response) datagram. This is the same assumption as above but in the opposite direction. This assumption also only makes sense in a peer to peer sense, because it presumes a server on the (original) client.

If possible, these assumptions should be satisfied by all transport implementations. The following sections go into further detail on the ways in which DOF implementations use the transport. These assumptions form the basis for how DOF implementations remain transport independent at the application layer.

3.1. Typical Session and Server Types

The discussion above indicates several properties that are associated with transports, clients, and servers. Out of the many different combinations of properties, there are three types that are so common that they are assumed to exist. These three types were introduced above.

3.1.1. Session/Lossless

This typical combination assumes that a server was used to create the session, and that from that point the session exchanges datagrams that are lossless and in-order (assuming that the two are related). Note that it is common for this combination to also provide a streaming session, but that DOF protocols do not differentiate streaming/datagram for this combination: it may be either. It is typical that this type of session will include only 2 nodes, but it is possible to have *n*-node sessions that are lossless.

DOF protocols make the following assumptions about these types of sessions and the servers that establish them:

1. The server type is session, although it may be connection (connection is never required). The transport will indicate the creation of the session to the DOF implementation, and it will create associated state. The DOF implementation will manage this state until (in the case of a connection) the transport indicates that the connection is terminated or that fact is determined by higher protocol layers.

2. All communication through the server is associated with a session. No datagram will be received (by the DOF implementation) that is not associated with a previously created session.

3. The session and datagram contains enough information for a response or new command to be sent to the sender of a datagram. In the typical case of a 2-node session then only the session is required.

4. Only the unicast address type is allowed for both nodes in a 2-node session. In *n*-node sessions the other address types (multicast/broadcast) are allowed, although broadcast is limited to the nodes in the session. The session or connection established uniquely identifies a single application on each node.

5. Since a session or connection must be created before a datagram is exchanged, the transport must provide a notification method for session creation. Further, the application must explicitly accept the new session in order to correctly track the state that it must maintain. Once created, the session can exchange a datagram. A received datagram is always provided in the context of a session. The session may be streaming, in which case the DOF Network Protocol will provide framing to preserve the datagram boundaries.

Lossless sessions are almost always associated with streaming, and so this category is often called 'streaming' servers. However, it is actually the session and lossless aspects of the sessions, along with the number of nodes in the session, that is the most critical for DOF operation.

3.1.2. None/Lossy

This typical combination assumes that no session is created. This means that one node maintains state about the other (the client), and that the other node utilizes a server that receives datagrams. The lack of shared state almost always indicates a lossy relationship.

Note that the lack of shared state (even though one node is likely maintaining state) means that no session exists. This lack of session means that the server treats each individual datagram as unrelated (although datagram relationships may be reintroduced due to different protocol layers).

DOF implementations make the following assumptions about these relationships:

1. The server does not create a session, it only accepts datagrams.

2. All three addressing types (unicast, multicast, and broadcast) are possible with these servers.

3. Since no session is created, the transport merely provides a notification method for the arrival of a datagram. Each datagram has identical framing to when it was sent.

4. Each datagram that arrives is tagged by the transport with enough information that a corresponding datagram can be sent back to the sender by using a server and transport address.

5. Each datagram that arrives is tagged by the transport with the sender’s address, allowing other clients using the same transport to send a datagram to the sender (this is the definition of 'discovering' a transport address).

Because of their association with a datagram, this category is often called 'datagram' servers. However, it is actually the lack of session and lossy nature of the server that is the most critical for DOF operation (along with the capability of multicast and broadcast).

3.1.3. Session/Lossy

This typical combination assumes that a session is created, but that datagrams may be lost. All nodes maintain state about the session. It is very typical for an implementation to create this combination from the none/lossy combination just described through the use of different protocol layer functions. Note that the client likely already maintains state, and so only the server needs to be told to keep track of the particular client. Note that some sort of session identification is likely required as well, provided by different protocol layers on top of the transport. This identification relates different datagrams to each other, as well as to the session information.

DOF implementations make the following assumptions about these relationships:

1. The server itself may not create a session, and may only accepts datagrams. The DOF implementation creates the session if required and not provided.

2. Any session created by the server is automatically destroyed when no longer used. The DOF implementation does not manage the session in this case.

3. All three addressing types (unicast, multicast, and broadcast) are possible with these sessions. The session established uniquely identifies a single application on each node.

4. The transport provides a notification for the creation of a new session (if handled by the transport). Otherwise the session is managed by the DOF implementation.

3.2. General Transport Properties

The general function of the transport is to provide a datagram to the DOF implementation and accept a datagram from the implementation that will be sent on the network. Each datagram is either in the context of a node (to be sent to a server), a session, or a server (to be sent to a node).

Servers and the sessions that they create are generally bundled into applications, with each application running on a node being associated with a unique set of servers (and associated transport addresses). The assumptions introduced earlier in this section are meant to allow discovery of the relationships between the servers associated with a single application.

Relating to this association, the overall assumption is that there is a relationship between the type of server and the associated transport addresses. From the perspective of a node, knowing something about the datagram properties discussed above allows for mapping from one server type to another. As DOF implementations deal fundamentally with two types of servers (lossy, and those that create lossless sessions), this means that there are generally two servers that must be related.

Each transport identifies the format and definition of its transport addresses. This information is opaque to the DOF implementation. Further, each transport has an associated Maximum Transmission Unit, or MTU. This represents the maximum datagram size that can be transmitted using the transport. In general there is a relationship between datagram size and network efficiency. This relationship is not exposed to the DOF implementation, other than a recognition that as datagram size approaches the MTU that efficiency likely drops.

DOF implementations require that the following properties are associated with each datagram, and they must be provided by the transport implementation:

1. The source transport address. For 2-node sessions, this is associated with the session itself. For servers and *n*-node sessions it is likely associated directly with the datagram.
2. The datagram data and size.
3. The source transport address' type. This will always correspond to a unicast address.
4. The target transport address' type. This is related to the server in most cases, otherwise with the session itself.
5. The properties of the server/session that received the datagram:
6. Lossy/Lossless (corresponding to Orderless/In-order).
7. Streaming/Datagram.
8. None/2-node Session/*n*-node Session.

3.2.1. Impact of Security on Datagram Properties

As mentioned earlier, it is often difficult for an application or transport implementation to accurately determine the target transport addressing that was used on any particular datagram. Even in the case of sessions that should be point to point, the possibility of packet injection means that the 'real' source of a datagram is suspect. In the case of lossy servers it may not be possible to determine if the client really used transport unicast, multicast, or broadcast.

This can cause problems for DOF implementations, since application behavior relies on the specific property values associated with the datagram.

In general, this means that in the absence of security that the client and server must do their best to determine the actual datagram properties, but that they must realize that the information cannot be fully trusted.

However, once a security layer has been added then the properties must be trusted. This means that the source and destination application addressing (*not transport addressing*) is verified, that the datagram data and size is verified, and that the transport address type (unicast, multicast, or broadcast) is verified. When the security protocol cannot validate any of these properties then it must modify the datagram properties to represent what is *possible*, rather than trust the transport-determined property value.

For example, if a session (including the DOF protocol layers) can guarantee that a particular datagram arrived from a particular sender and can only be received by a single node, then it can modify the property to indicate unicast addressing. If, however, the security protocol cannot guarantee a single receiver, then it must modify the property to indicate multicast or broadcast as appropriate. Applications that base behavior on these datagram properties for a secure datagram must use the property values as modified by the security protocol.

> **Secure datagram transport properties must represent only secure information.** `dof-2008-1-spec-1`
>
> This allows applications to determine the level of trust for all information that they receive, even from the transport layer. Other information that is untrusted may be presented, but should not indicate that it is secure (unless the transport knows it is secure). For example, transport addressing is typically not known to be secure, and so should not be trusted unless the transport indicates that they are secure.

3.3. Lossless Requirements

As discussed above, lossless servers are always associated with a session or connection. The server must indicate to the implementation that a new session has been created, and the implementation must explicitly accept the new session.

The following requirements apply to each lossless session accepted by the implementation.

> **Sessions must be monitored and must guard against improper clients.** `dof-2008-1-spec-2`
>
> A session with a non-communicative or ill-defined client must not be allowed to exist for too long. There are no explicit timeouts defined in this case, and so the implementation is free to impose whatever requirements it will. The implementation can use the features of the DOF Presentation Protocol to ensure that a session remains valid.

Sessions are always initiated by clients. The client must determine the appropriate target transport address for the server. For lossless sessions, the client must maintain the session, and is primarily responsible for ending the session. If the client determines that the session is no longer required then it should use the transport to close the session in a way that the server session is also closed. This should involve correctly closing the session at each layer of the protocol stack, beginning with the application layer and ending with the transport layer.

However, it is possible for the server to terminate a session as well. In this case the client must remove the state associated with the session at each layer of the stack.

3.3.1. Traffic Symmetry

The DPS frequently runs over lossless transports like TCP/IP. These transports are typically optimized for lossless bi-directional data, and can have performance issues when used to transmit uni-directional, bursty datagrams. Unfortunately, many small-footprint stacks maximize this performance problem by limiting their implementations. To minimize problems it is best that each datagram be acknowledged before the next can be sent, or at least that the datagrams flowing in each direction are roughly equal.

There are two common optimizations specific to TCP/IP that can cause performance problems in these situations. The first is the Nagle Algorithm, which can slow the sending of data assuming that it is desirable to combine many small datagrams into fewer larger datagrams. The second is the introduction of a delay on the ACK assuming that it is better to piggyback the ACK on a data datagram than to send it with no data. The combination of these two optimizations can lead to poor performance of the DPS on TCP/IP.

There are typically options to disable the Nagle Algorithm (TCPNODELAY). Because this affects outgoing traffic there is little more that can be done to optimize performance other than disabling it.

The second optimization, that of delaying an ACK until data exists to "piggyback" it on, is typically not configurable. The answer to this performance issue is to ensure that the protocol in question is symmetric - meaning that each datagram sent in one direction elicits a response datagram going the other direction.

Unfortunately, a typical use case is to establish a session, establish state, and then wait for responses. In this case, little or no traffic moves in one direction and many periodic datagrams move in the other direction. This is the worst situation for both the Nagle Algorithm and the ACK-delay problem. The Nagle Algorithm delays the commands, combining them into larger datagrams. The ACK messages delay because no datagrams are going in the other direction. This is standard problem with asymmetric traffic in TCP/IP.

The DPS provides the commands (through the Ping and Heartbeat commands) to allow for symmetry in the application protocols. This means that if an application protocol knows that it is in a non-symmetric situation and wants to even out the datagrams that it can use the DPP common commands. This behavior should depend on the transport, as it knows whether symmetry is beneficial.

3.4. Lossy Requirements

As discussed above, lossy can refer to both sessions (between nodes) and servers (between a node and a server). In the case of a server there is no indication of communication except for the arrival of a datagram. Any session creation is the responsibility of the DOF implementation. As discussed earlier, the client transport implementation should be able to map between these inbound datagram responses and an associated lossless server transport address.

Datagram responses should be mappable to an associated lossless server transport address. `dof-2008-1-spec-3`

A general architecture principle discussed earlier is that a datagram response can be used to determine a corresponding lossless server transport address (if one exists). This means that the transport information from a datagram response should contain enough information to identify the transport address associated with a lossless server (on the responding server).

Lossy commands should be mappable to an associated lossless server transport address. `dof-2008-1-spec-4`

An outbound lossy command should allow any corresponding lossless server transport address to be determined by the recipient. This allows any command to be used to identify the transport address of the client's lossless server.

3.5. Multicast Requirements

Servers may have configured transport address for transport multicast communications. In most cases, this transport address used by the DOF implementation should be standardized for the transport. This allows greater compatibility between systems. If a set of applications (a system) wants to operate in isolation then they may use a different multicast addressing, although this additional addressing should be registered with IANA (for IP) or other applicable standard body.

Even in the case of systems using different multicast addressing, the servers should still use the standardized transport addressing in addition to their system addressing.

✓ **Multicast servers should use registered, consistent transport addresses.** **`dof-2008-1-spec-5`**

This requirement helps with interoperability of devices.

3.6. Address Discovery

In order for a client to communicate with a server it must know the appropriate transport address. This information is typically obtained in one of three ways:

1. Static configuration. The client can be told explicitly where to connect. This method is always possible, but is usually inflexible and generally not recommended. If static configuration must be used (for example, in the case of WAN system where discovery is not possible), then naming services (like DNS) should be used. The use of a fixed transport address is strongly discouraged.

2. Discovery. Addresses can be discovered based on multicast discovery. This is more flexible than static configuration, but requires multicast to be enabled on the network and can only discover local servers. Discovery does offer the potential for minimal configuration.

3. Dynamic configuration. Many transports support dynamic configuration. For example, DHCP can be used on IP-based networks to configure DOF properties. Like discovery, dynamic configuration offers the potential for minimal configuration. The use of standard mechanisms to configure DOF properties (such as additional fields in a DHCP request) should be registered with the applicable standard body.

3.7. Transport Specifications

Each transport that can be used with DOF systems must have a specification document. The specification must outline how the transport requirements detailed here are satisfied. It must also define any standardized addressing and other properties.

The transport specification must indicate the following:

1. The format and meaning of its transport addressing.

2. The types of transport addressing that are supported (unicast, multicast, broadcast).

3. The types of servers and sessions that are possible (lossless, lossy, 2-node, *n*-node), with any associated MTU information.

4. The mapping that is possible between different server types based on transport addresses.

5. The different types of configuration that are possible, including the provisions for any dynamic configuration and how DOF properties can leverage dynamic configuration.

This information is required for determining the transport context information.

Up to this point the discussion of different transport address types has not been combined with the different types of transport configurations. Keep in mind the three types of transport combinations discussed: none/lossy, session/lossy, session/lossless. These can be combined with three different address types: unicast, multicast, and broadcast. However, only the 'none' type can utilize the different address types, with sessions always being associated with unicast addressing.

Each datagram used by DOF implementations can therefore be categorized by its requirements in three areas:

1. Whether a session is required, and the number of nodes that can be in the session.

2. Which addressing is allowable (unicast, multicast, or broadcast).

3. Whether the session must be lossless (implying in-order), or if lossy is allowed (implying orderless).

This can be summarized as the 'Session' requirements and 'Addressing' requirements. Session is one of: None, Lossy (2-node, *n*-node), or Lossless (2-node, *n*-node). Addressing is one of (Unicast, Multicast, Broadcast).

Each DOF PDU will indicate the session and addressing requirements that it has. In the case of a session the source of the session will be identified.

4. DOF Protocol Stack (DPS)

The DOF Protocol Stack (DPS) is the foundation for all DOF protocols. These protocols operate on a variety of platforms, including small devices with limited resources. Even though the small devices require optimization, the protocols follow the OSI model of layer separation as much as possible.

The DOF Protocol Stack also operates on a number of different transports. To the extent possible, the DPS and other DOF Protocols do not require specific information about the transport being used. For example, the protocols do not expose details of transport-layer logical addressing at higher levels. In the same way, implementations of the protocols maintain this same transport-agnostic view. Later sections of this document note specific transport requirements.

The lowest layer of the DPS is the DOF Network Protocol (DNP). This layer provides the minimum requirements in order for the DPS to be identified (including versioning), and provides the datagram length information on streaming transports. The DNP also implements some transport layer functionality, including a sense of addressing that can augment what the transport provides.

The session and presentation layers are combined in the DOF Protocol Stack as the DOF Presentation Protocol (DPP). DPP handles encryption and replay-attack prevention (functions of the presentation layer) through encryption modes of operation. It does not handle retries or network-layer acknowledgements. DPP does provide the foundation for commands and responses, including operation identification and loop detection.

Finally, there is the DOF Application Layer. Other references discuss specific DOF application protocols, although some discussion common to all application protocols appears here.

4.1. General Principles

The following sections discuss general principles that govern the DPS and its associated application protocols. These topics relate generally to both DNP, DPP, and application layers, although the specifics may apply more to one than the other.

These principles are also not dependent on the version or layer used. This means that they apply to the stack as a whole, and that any application protocol can depend on the behavior.

4.1.1. Reserved Bits

Throughout the DPS and DOF application protocols, there are references to 'Reserved' bits. The following discussion applies to all of these fields.

This protocol specification defines the meaning of every bit that passes over the wire. In many cases, it does so by defining a field and then defining the range of values that can appear in that field. In other cases, it defines a 'bit field,' and in doing so leaves some bits as available. These bits may require a constant value, or the specification may indicate that they are **Reserved**.

In the absence of other requirements, the handling of these remaining bits would be ambiguous, which is not a good idea in protocol design. However, the possible future use of these bits is restricted because existing implementations will not understand any newly defined behavior.

Any future use must then:

1. Not change the 'format' of the PDU, as existing implementations would not understand how to read the PDU (or would read it incorrectly).

2. Not affect the meaning of the command or response in a way that the receiver must understand.
3. Only introduce optional behavior that does not change the format of the command or response.

Note that any changes that do not meet these requirements necessitate changing the protocol revision.

Handling these bits requires correct management. The following rule defines correct behavior.

> **Senders must set all *RESERVED* bits to zero.** `dof-2008-1-spec-6`
>
> The zero value ensures a known state (which will be the default) for any new behavior.

> **Receivers must ignore all *RESERVED* bits.** `dof-2008-1-spec-7`
>
> This ensures that old implementations will ignore behavior defined in the future, although they will accept the PDU independent of the value.

4.1.2. Timeouts

A general requirement of the DPS is that non-communicating or ill-behaved sessions should be detected and terminated/closed as quickly as possible. Ill-behaved sessions are particularly dangerous (from a security perspective) during stack establishment. A goal is to force negotiation as quickly as possible, without being so restrictive that slow network links cause problems.

> In order to force negotiation quickly each layer in the protocol stack may define timeout behaviors. These timeouts are serial, not parallel. This means that a single timer for each session can enforce all timeouts. The behavior of the stack when a timeout occurs is uniform: the implementation terminates the session. Reaching each milestone in stack negotiation resets the timeout based on the requirements of the new stack layer or layer set.

This general rule applies throughout negotiation and through the authentication phase. The single timer use does not continue into the application protocols, but may still be used (based on the transport requirements) to determine communication failures. These types of timeouts are dependent on the implementation.

4.1.3. Protocol Discovery

Different nodes in the same network may support different versions of DPS layers and different application protocol versions. A primary goal of DOF protocols is interoperability, and so it is important to be able to discover these different nodes and communicate with them if possible.

At the lowest layer (transport), there should be the ability to communicate with all nodes. This usually requires either multicast or broadcast. Specific system implementations may utilize non-standard addressing in order to localize communication. Independent of specific system requirements, DOF nodes should still listen on the standard addresses and ports defined for the transport in order to facilitate discovery by general nodes.

The ability to speak multiple versions of a protocol is generally associated with a gateway. However, it is possible that some nodes (clients or servers) may not function as a gateway, but may speak different protocol versions.

If these nodes simultaneously attempted to use all of the versions that they can, worst-case traffic would result. This is because they may be sending protocol versions that no other node understands, and so the traffic is wasted.

The same is true of a gateway. If gateway nodes simultaneously used all of their versions then the same worst-case traffic would result. It only makes sense for a gateway (or a multi-protocol server or client) to speak versions that others on the network are able to understand.

DOF protocols provide a method for discovering protocol versions in an optimized way. As a requirement, the nodes must be capable of using a multicast or broadcast transport. In general, the difference between 'node discovery' and protocol discovery is that in protocol discovery the goal is *not* to identify each node, but rather each protocol and protocol version. Whether there are 10 nodes speaking a version or 1,000 nodes is not important (that number can be determined later by using node discovery). This allows for some specific optimizations for protocol discovery.

As an example of the problem of multiple versions, consider the following. A company sells an DOF-based sensor product. The product implements specific versions of the DNP, DPP, and applications. The sensor only responds when queried. Years pass, and the customer has removed the original product that used the sensor, but has not removed the sensor. The customer installs a new product, which speaks different DNP and DPP versions.

How can the new product speak to the old sensor? Without aid (some sort of gateway) it cannot, unless the new product also speaks the older protocol versions. Assuming that it does not speak the older protocol version, then a gateway (a special node that speaks multiple protocol versions) is required. How can such a gateway discover which DNP and DPP versions are available on a network?

One solution, discussed above, would be to cycle through all possible versions, discovering nodes that use each combination. This is expensive in terms of network traffic. Another solution would be to have each node advertise itself on the network periodically, but this is also expensive in terms of network traffic that would be mostly redundant.

In order to solve this problem, DOF specifications define version zero of each protocol (and protocol layer) as a query protocol. Each node is required to support version zero, although it does not encapsulate normal PDUs. The version zero protocol for each layer follows a pattern that allows version discovery.

This means that all nodes will support at least two versions of each stack layer: version zero (for version discovery) and some other version (for PDU encapsulation). Implementations drop unrecognized versions.

Implements must drop PDUs that use unrecognized versions on lossy transports. `dof-2008-1-spec-8`

Unrecognized PDUs may arrive on lossy transports. This means that dropping PDUs using unrecognized versions on a lossy transport is safe. There is no indication on the receiving or sending node when this happens.

Note that by supporting the query version (version 0) a node is only required to respond to queries. If a given node has no reason to discover other version information then it never needs to initiate a query.

4.1.4. Protocol Negotiation

Different nodes in the same network may simultaneously be using different versions of DOF protocols, including the DNP, DPP and applications. This means that when two nodes communicate, they need to determine which versions they will use.

If the nodes are using a lossy session or server then the sender can either assert a version or use version discovery (unicast) to determine the versions spoken by the target. If the sender asserts a version that the receiver does not understand then the receiver silently drops the datagram.

When two nodes establish a lossless session, they have the benefit of both bi-directional non-broadcast communication and shared state. This makes the negotiation of protocol versions possible. It is also possible that a lossless session spans local networks where protocol discovery (which uses a multicast or unicast datagram) is not possible. This means that protocol negotiation is required for the two nodes to communicate.

There are two phases to protocol negotiation of the DPS on a lossless session. First, the nodes negotiate DNP and DPP (simultaneously). Following that negotiation (and assuming successful negotiation) a specific application protocol, the DOF Session Protocol (DSP), is used to negotiate the application layer protocols and their options that will be used on the session. This negotiation includes the specific authentication and key distribution protocols that are required.

> **Two-node lossless transport sessions immediately negotiate protocols.** `dof-2008-1-spec-9`
>
> Protocol version negotiation takes place immediately after the transport-level lossless session is established and before any application PDUs are exchanged. It begins with the client sending a datagram.

Negotiation of the protocol version begins with the client. The client identifies the DNP and DPP that it prefers (possibly the only ones it understands) by sending a two byte datagram (one byte DNP, one byte DPP) using only the protocol version bytes and omitting flag bytes, meaning that the flag bits must be clear.

> This definition of negotiation assumes that lossless 2-node sessions use a mechanism that provides in-order guaranteed delivery. It also assumes that nodes may send any number of bytes that the other side of the session receives in a timely fashion.

If the server can accept those protocol versions (even if it does not prefer them because they are older and less capable), it echoes a two-byte datagram using the same protocol versions, or alternatively it just begins using those protocol versions by sending a datagram that uses them. Both the client and server know that they are speaking the same versions, and the negotiation of application protocols begins (described further in the section on Application Protocols). The server knows the protocol versions because it echoed the datagram (acceptance), the client knows because it received a datagram with the same protocol versions that it sent.

> **Negotiation of version is in decreasing order of desire.** `dof-2008-1-spec-10`
>
> Each node must have a list of the versions of DNP and DPP that it can speak. It orders these by most desirable to least. Negotiation always begins with the most desirable, and progresses to the least desirable.

> **Accept the first valid negotiated versions.** `dof-2008-1-spec-11`
>
> If a node receives a combination of versions of DNP and DPP that it *can* speak (meaning they are understandable), it must accept it even if it does not *prefer* to speak those versions (possibly because they are less optimal).

If the server cannot understand the protocol versions requested, it responds with a two byte datagram using the protocol versions that it prefers (potentially the only ones that it understands), and the roles of client and server are reversed (for the purposes of further negotiation). This means that the old 'client' node is in the role of responding with either an acceptance, beginning to use those versions (by sending a datagram), or another 'rejection' in which case the roles are reversed yet again.

Upon receiving a two byte datagram during negotiation with protocol versions that are not understood, the response is always a two byte datagram with the next preferred protocol versions that are understood or a full datagram that uses a set of versions. This proceeds until either side has no untried protocol versions or a bad datagram (non-negotiation and using versions that are not understood), at which point the session is terminated.

It is common for devices to implement a single set of network and presentation protocol versions. It is also common for devices to initiate sessions with more powerful nodes. This means that in the typical case a client (the node sending the first negotiation bytes in the common case) speaks a single set of protocol versions. The same situation can apply to a server.

In this case where the sender speaks only a single set of protocol versions, then the first datagram sent may include the flag byte, control fields and encapsulated application data. This is 'asserting' protocol versions. This kind of assertion is always a valid response during negotiation, and the client may immediately use it.

> The presence of an application PDU determines the difference between 'asserting' and negotiating on a session. All negotiation PDUs will lack an application PDU. All asserted PDUs would contain an application PDU.

During negotiation, immediately terminate 2-node session on receipt of unknown versions. `dof-2008-1-spec-12`

> During negotiation of a 2-node session, if the server or the client receives a non-negotiation datagram that includes protocol versions that it does not understand, the receiver must terminate the session.

If the server speaks only a single set of protocols, it may preemptively send a full datagram with a flag byte, control fields, and application data when the transport session is established. The client would treat this as the end of negotiation. In the case that the client cannot speak those versions, the client terminates the session. If the client had also asserted and the versions did not match then the client and server would both terminate the session. This is correct behavior because an assertion is only possible if the node understands only a single set of versions. Any mismatch of asserted versions means communication is not possible.

> During negotiation, when a node reaches the last sets of versions it understands, it is optimal to assert those versions. For example, a node that speaks A_1, B_1 and A_1, B_2 may send A_1, B_2 as a negotiation datagram. Assuming that the other side continues negotiation (implying that those versions are not understood), then the node can continue to A_1, B_1. However, in this case if the other node tries to continue negotiation there are no more versions to try and the node will terminate the session. Asserting A_1, B_1 at that point saves the node from continuing to negotiate when it is pointless.

DNP/DPP version negotiation must complete within 10 seconds. `dof-2008-1-spec-13`

> Measured from when the client establishes the transport session, if the stack cannot converge to an agreed set of network and presentation protocols within **ten (10) seconds** then the DPS session must terminate. Convergence is the time at which the transport client sends its first non-negotiation datagram.

4.1.5. Transport Addresses

Throughout the DPS documents, several references to transport addresses exist. The use of this term 'transport address' means a unique identification of the source of some communication on a specific transport. It also identifies an individual target on a transport.

Different transports may use similar logical addresses. There can also be multiple interfaces speaking the same protocol with identical addresses. The stack should consider all these cases as *different* transport addresses.

There are transports that lack a well-defined sense of addresses, or where implementations make it necessary to add additional data in order to distinguish addresses. To resolve this problem the DPS adds its own sense of a logical address, called an DNP (DOF Networking Protocol) address. It is important to understand its relationship between DNP addresses and transport addresses.

> **Correctly distinguish transport addresses.** **`dof-2008-1-spec-14`**
>
> Implementations must track the source of communication such that it can send responses back to the sender. This source (and response destination) is called the "transport address," even though it will likely need to contain additional information beyond the actual transport logical address. Implementations must permit DNP addresses, even if the transport itself provides unique addresses.

In the case of sessions, the notion of 'transport address' really relates to the session and not just the transport. For example, it is possible to have both a UDP datagram session and a TCP streaming session that involves the same 'transport address.' In this case, the 'transport' is also part of the 'transport address,' even though in both cases the underlying transport is IP.

This distinction is critical because many stack issues relate to the session, and not directly to the 'transport address' when used in the more limited sense.

4.1.6. Loopback Prevention

A common issue with broadcast and multicast transports is receiving datagrams that the node sent, either because of operating system problems or because of network echoes. In order to prevent each layer of the DOF Protocol Stack from needing to address this issue the transport layer of the DPS must ignore these datagrams.

The transport (referring to the layer directly below the DNP) is uniquely able to remove these echoes because only it is aware of the specific meaning of transport addresses. It also knows (or can know) the different types of sessions and servers that are in use, and whether it is possible for loopback to occur.

In addition, if a transport is not able to determine on its own whether loopback will occur, there is a specific DNP version (127, described later) which a transport implementation can use to determine if loopback is occurring.

> **DPS transports must correctly reject loopback datagrams sent from the same application.** **`dof-2008-1-spec-15`**
>
> As described, all DPS transport implementations must correctly reject datagrams that originate from the same application. Application protocols may rely on the fact that they will not receive loopback datagrams.

Stated another way, all DPS transport implementations must reject inbound multicast or broadcast datagrams that sent by the receiving application. Pay particular attention to the wording – datagrams from the same *application* must be rejected, not the same *node*.

In general, this means that implementations must compare the source information (for example, host, and port for UDP) for inbound datagrams with the source information for each session for the receiving application. If the source information matches, then the receiver drops datagram. If such a comparison is not possible or may not result in precisely correct results then the implementation may use DNP version 127 to send a datagram and watch for loopback.

As a further example, two applications running on the same node may be receiving multicast datagrams. Both applications hear datagrams sent from the first application, but only the first application drops them. The same is true for the second application and its datagrams.

4.1.7. Invalid PDU Handling

Receiving any invalid PDU signals either a corrupt communication channel, loss of state, or a possible attack on the protocol. Invalid PDUs include things like invalid op-codes, malformed flags or other fields, PDU under runs and over runs during parsing, or other structural problems. Invalid PDUs do not include cases where the structure and references are correct, but the operation fails to execute for other reasons.

Throughout the entire DOF Protocol Stack and all related application protocols, the requirements for handling invalid PDUs are identical. The primary consideration is maintaining correct session state (both transport and DPS). If session state cannot be ensured, then the session (whether transport or DPS) must be closed.

For session 'none' state management is not a concern, because there is no DPS session state by definition. However, there is still transport state to manage. For example, streaming protocols like TCP may not be able to identify PDU boundaries without state. If this state is confused by an invalid PDU then the transport session state is lost.

In most cases, datagram PDUs can be dropped without loss of state, while streaming PDUs cannot be dropped without loss of state. Streaming sessions usually have an in-order, guaranteed delivery feature that is leveraged to minimize state transferred in each PDU (for example, packet sequence numbers for security). In these cases, dropping PDUs will affect the state and must therefore result in the session being closed.

> ✓ **Correctly handle invalid PDUs, dropping sessions when correct state cannot be ensured.** `dof-2008-1-spec-16`
>
> As described, all DPS transport implementations must correctly reject datagrams that originate from the same application. Application protocols may rely on the fact that they will not receive loopback datagrams.

4.2. Operations

Operations are a general feature of the DOF Presentation Protocol (DPP). A complete understanding of operations requires understanding the principle (described here), the syntax (described in the appropriate DPP version section), and usage that is presented in each of the application protocol descriptions which use operations. This section introduces operations because they form the basis for the requirements and terminology used throughout the following sections.

Operations are part of the DPP because they form a common platform for all application protocols to build on. They are also the foundation of providing equivalent behavior across different transports. Since a general goal of the DPS is to hide this kind of transport information from the application, it makes sense to define operations in the DPS.

Operations also form the foundation for command/response handling in DOF systems, and so it is useful to share the specifics between all of the different application protocols.

Finally, operations provide the basis for detecting and managing loops and duplicate commands. This is a key requirement for DOF mesh networking, which allows network topologies that include loops.

The protocols use the word 'operation' in many different ways. The most general is any command or command/response pair used by an application protocol. Note that in this case what the application terms an 'operation' may or may not be an operation to the presentation layer. This is the case if the operation does not include an operation identifier.

There are two primary ways that DOF protocols uses operations:

1. FLOODED: In this case, a node sends a single operation over multiple links, with the goal of complete coverage of some set of nodes (including all nodes). To prevent infinite looping, the presentation layer *must* maintain state and uniquely and globally identify each flooded operation. The reason is that the same operation may arrive again (from a different source) due to loops in the network topology. It is the responsibility of the presentation layer to handle this situation, and state is required to accomplish this.

2. DIRECTED: In this case, an application determines the propagation of the operation based on application state. Each node directs the operation forward to another node in such a way that loops will never occur.

Because state is required at the DPP layer in the FLOODED case, each DPP version must be able to identify whether an operation is FLOODED or DIRECTED.

✓ **Correctly maintain the DIRECTED and FLOODED state of operations.**

`dof-2008-1-spec-17`

A node may convert a DIRECTED operation to a FLOODED operation, but not the reverse. Once an operation is FLOODED then a node must never convert it back to a DIRECTED operation. Each DPP version defines the methods used to guarantee this behavior.

Operation identification is a critical requirement of the DPP, and the method used is to assign each an operation identifier. The *DOF Common Types* document defines the specific format of a general operation identifier, which applies across all protocol versions. There are always two pieces of data determined by the creator/owner of an operation: the SID (Source ID) and the Operation Count (OPCNT). While the general format of the OPID is common, each specific DPP version defines the methods used to represent an OPID in a PDU. DPP versions may define various compression techniques in order to save space in the PDU, all of which must be transparent to the DPS.

In addition, while the definition of operation identifiers is that they are globally unique, not all operation identifiers are global. Some commands, particularly those that are flooded through a network, require the global use of the same operation identifier. There is a cost to doing this: compression of the operation identifier may not be possible because some compression methods may be dependent on the source node information. Some directed operations might benefit from having their operation identifier change at each node in order to allow better optimization. This is generally possible for DIRECTED operations – it is never possible for FLOODED operations.

The presentation layer passes the operation identifiers, but the application assists in their management. Not all operations require an operation identifier, and some operations may have their operation identifiers mapped by a node to allow size optimizations. The application must communicate these requirements to the presentation layer so that appropriate information is included in the presentation headers. Received operations, the current operation state, the passage of time, and input from the application manage the lifecycle of the operation.

The SID is the primary source of uniqueness for the operation identifier, with each source assigning a unique count for the operations that it creates (the OPCNT). The SID is associated with an application. In cases where the implementation knows that a given node only provides a single application then it may associate the SID with a node. However, if the node can run multiple applications that do not share state, then each must use different SID.

Operation identifiers form the basis for directing responses back to the command. The DPP identifies responses and they use the same operation identifier as the corresponding command. Both commands and responses are operations.

Each identified operation must use a unique operation identifier. `dof-2008-1-spec-18`

It is a serious error to have two active and different identified operations with the same operation identifier. If this occurs then the behavior is undefined.

Responses use the correct operation identifier. `dof-2008-1-spec-19`

Responses always use the operation identifier of the corresponding command, if present. Note, however, that the specific PDU format used may differ from command to response. Nodes that map operation identifiers between arrival and forwarding must appropriately map the operation identifier of the response back to the original.

4.2.1. Rules for SID Use

The SID is the primary unique part of an operation identifier. This means that they are typically large (to achieve the goal of global uniqueness). The SID used on a secure connection requires the requesting of permission to use the SID as discussed below.

A typical case for a limited-resource node is:

1. It runs on a small platform.

2. It knows the details of its hardware.

3. No other application is running on the same hardware.

This is an ideal case for determining a SID because the SID can be a function of any unique information associated with the node.

However, there are situations where one or more of the following are true:

1. The client or server is not unique to the node (for example, a client running on an operating system).

2. The application *cannot* determine the specifics of the hardware, and so a unique SID is difficult to determine.

3. A node is acting as a bridge, and may need to represent multiple nodes.

In these cases, the application must choose the SID very carefully. In general, each instance of a running application (over time) should use a unique SID. The SID may reflect both some node property and also a sequence number or other unique factor to make each program instance unique. The goal is to prevent reuse of operation identifiers, even accidental reuse.

✓ **A single transport address (including DNP logical addressing) must be associated with a single SID in each Security Domain.** `dof-2008-1-spec-20`

This requirement allows caching of SIDs based on the sending transport address. This means that a SID must uniquely identify the *logical sender*. This means that two *logically separate* applications must use different SIDs, even if they run on the same node. It also means that applications that share transport addresses must use the same SID and guarantee that their operation identifiers never collide, or the applications may use an DNP logical address. Note that the requirement applies to each individual Security Domain, where unsecured traffic is treated as if it were in an "Unsecured Security Domain".

This discussion results in the following points for a client or server that cannot determine that it is unique on a node:

1. It must determine a SID and use it consistently and universally.
2. It must choose the SID such that it will not cause operation identifiers to alias one to another.
3. It may use a transport-based operation identifier with additional unique information.

SID comparison (and operation identifier comparison) is independent of the specific SID format used in a particular PDU.

✓ **Handle different OPID representations during comparison and forwarding.** `dof-2008-1-spec-21`

The common OPID format defines OPID comparison, independent of the received PDU format.

4.2.2. SID Security

Using a SID on secure sessions requires appropriate permissions. These permissions are associated with a particular DPP version, and so the particular permission formats are dependent on the DPP version used.

Clients and servers must negotiate any required SID permissions as part of the initial permission requests for the session, since without the permissions neither can send additional secure operations (including a request for additional permissions).

4.2.3. Operation Graphs

The statements above about DIRECTED and FLOODED operations reference state required for a DIRECTED operation, and state maintained for FLOODED operations. This section discusses these state requirements.

In order to manage a FLOODED operation, each node must maintain information about the operation in order to recognize duplicates and make decisions about how to handle the duplicates. Recognizing the duplicates requires a common identifier, which is the operation identifier. Using the operation identifier means that it must not change as it propagates through the network – a signature requirement for FLOODED operations. Another key requirement is that state for the operation must persist throughout the network in order to prevent loops and ringing.

The ultimate requirement for FLOODED operations is to create a DAG (directed acyclic graph) for each FLOODED operation. The root of the DAG is the original operation source, and it includes all other appropriate nodes as children. This ordered structure means that any node in the DAG has a sense of its parent, called the 'source.' Operations can outlive connections, and so the DAG for an operation may change over time.

> Note that the rules allow that the structure of the DAG for a given operation may change during the lifespan of the operation. This can occur because of either new connectivity or lost connectivity.

The DAG resulting from a FLOODED operation is the basis for establishing new operation state. To understand this, examine two common cases: responses and reactions.

The DOF DPP layer defines response operations. The definition allows for specific behavior for the response, as follows:

1. Responses always follow the DAG of the operation to which they respond. This definition means that responses never have to deal with loops.
2. Special responses, called 'final' responses, finalize an operation, resulting in the removal of operation state.

Responses are immune from many issues regarding state management, always relying on the established state of a single 'command' operation.

> **Unless specifically allowed by a PDU, transport unicast is required for all responses.** **`dof-2008-1-spec-22`**
>
> Responses traverse the DAG of the operation from child to parent back to the original source. Sending a response to node that is not the source is highly unusual, and only allowed in cases where the PDU explicitly defines the behavior.

Contrast the behavior of a response with the class of operations called 'reactions.' A reaction is an operation that results from a causing operation, but that is not a response. There are two primary cases for a reaction: a pseudo-response and a general reaction. Pseudo-responses follow the rules for a response (traversing a single DAG to a source) and do not require any special discussion. General reactions are more complicated, because they are not restricted to a single DAG.

Note that there is an assumption in discussing a reaction that it has available to it the DAG of some (possibly multiple) causing operations. Each of these operations has an associated DAG. However, reacting to multiple DAGs by way of combining them in general does *not* create a DAG. This means that while each operation itself has a DAG, reacting simultaneously using multiple DAGs must follow the rules of a FLOODED operation. To understand this, imagine a network in which each node establishes a causing operation at the same time. A node, anywhere in the same network, that sends a reacting operation will need that operation to go to every node in the network – and must in the process handle the presence of loops and prevent ringing. This is exactly the definition of a FLOODED operation.

Finally, note that reactionary operations may themselves establish a DAG. However, they must do this based on their own state and rules. Other operations may use this DAG, and in doing so, they are not FLOODED. This is in fact the definition of a DIRECTED operation. This implies that all DIRECTED operations must identify the method used to establish the DAG that they will use.

4.2.4. Operation Lifecycle

The DPP manages operations based on datagrams sent on the network. These datagrams transfer state from one node to another, and based on the operation both the sender and the receiver must manage the state.

The management of operation state is a critical security issue. Receivers must validate a datagram, from a security perspective, before the receiver modifies any operation state based on the datagram. The *DOF Security Specification* discusses this at greater length.

Programs create and manage operations, and so operation identification and 'program' identification are related. There are two identifying marks for a program in an DOF network:

1. The transport addresses that it uses to create (send) operations.

2. The source identifier (SID) that it uses.

This means that an operation source equates to a transport address. The operation itself relates to a SID, although maybe not the SID of the sender (in the case of an operation forwarded through another node).

The state associated with an operation is similar to the state of a session: it has a beginning, it is maintained, it can be cancelled (closed), and it allows state to be shared between endpoints. A key difference between sessions and operations is that operations distribute throughout a network (typically a mesh network). This section discusses common aspects of operations and the lifecycle controls that are included in the DOF specifications.

To emphasize this last point, the state associated with an operation distributes equally among all participating nodes. There is no way to use a single operation to maintain *different* state between *each* receiver and the sender. Shared state that is limited to a single sender and receiver requires its own operations.

For any operation, each node is in one of the following states:

1. Initial: The node first identifies the operation and creates state.

2. Retry: The node updates the state of an existing operation.

3. Lost: An operation is lost when the source of the operation is no longer available while the operation remains active. There is a related flag called Proxy Lost, which means that a node is aware of someone else’s source being lost while its source remains valid.

4. Timeout: The operation has timed out, meaning that it reaches the end of its defined lifecycle.

5. Cancelled: The operation has ended based on direct action of the source.

The receipt of a datagram causes most changes to operation state. The exception is the Timeout state, and sometimes the Lost state. There is a special case possible where an operation arrives (Initial state), but has already timed out (Timeout state). There are several rules for handling this situation:

1. Receivers must pass application operations (those without operation identifiers) to the application. The application must be able to determine that the operation has timed out. Since there is no operation identifier, the receiver does not create presentation layer state.

2. Receivers must pass identified operations to the application. The application must be able to determine that the operation has timed out. Even though an operation identifier is included, the receiver should use the fact that it has timed out to prevent creating presentation layer state. Note that the node will

drop any responses to these operations, as there is no presentation layer state to associate the response with the operation.

3. As a further special case, receivers must ignore FLOODED operations that have timed out. This means that both an operation identifier and duration are required for FLOODED operations.

> ✓ **Correctly handle immediately timed-out operations.** `dof-2008-1-spec-23`
>
> The behavior in this case must be the same as receiving and processing the operation followed by the operation timing out. However, it is permissible for the implementation to optimize the handling of these two phases. Receivers assume in this case that the source of the operation has already timed out, and so do not send responses based on DPP state. Receivers ignore FLOODED operations that are timed-out.

The following sections discuss the DPP requirements for operation management. Application protocols will have their own requirements.

4.2.4.1. Operation State: Initial

The initial state is entered whenever a new operation identifier is referenced that has not been seen before. It usually happens because of the receipt of a new application PDU, with the operation information extracted from the DPP.

> ℹ It is possible that a retry on one node looks like a new operation on another due to datagram loss. The definition of the response behavior of the initial PDU and a retry PDU are identical for this reason.

In order for the operation to enter the Initial state, it must be authentic and represent a valid operation in the context of the sender and receiver. For example, this may mean that the sender has permission to perform the operation. The receiver must utilize the application to make this determination.

4.2.4.2. Operation State: Retry

Retries occur whenever a new, valid PDU references an existing operation identifier. Datagram loss may cause problems distinguishing between the Initial and Retry states, and so applications avoid defining different behavior for each.

> ℹ There is a difference between DPS retry and transport retries (or retransmit). On lossy transports, it is beneficial to send datagrams multiple times in order to improve the changes of reception. This is **not** the same as an DPP retry, because in this case the PDU remains unchanged. It is a transport requirement to ignore these retransmissions.

Operation retries serve to distribute new operation state, including extending the lifecycle of the operation. Receivers must exercise care not to make changes to existing operation state until the application validates the operation.

4.2.4.3. Operation State: Lost

Nodes enter this state for particular operation when the source of an active operation is no longer available. This document discusses the specific handling of this state later on, but in general results in a search for a new source. The lack of a new source causes the operation to enter the Timeout state.

The commands used to search for a new source are dependent on the specific DPP version; however the logic is global and described later.

Related to the Lost state is a flag called Proxy Lost, which indicates that a node is aware of another node that is in the Lost state while its own source remains valid.

4.2.4.4. Operation State: Timeout

Each operation defines its duration. Exceeding the duration causes the operation to enter the Timeout state. This removes all state for the operation, but does not cause sending any datagrams. Note that operations will timeout uniformly throughout the network (with some possible differences because of clock skew).

4.2.4.5. Operation State: Cancelled

The source of an operation may cancel it. The specifics on operation cancellation are dependent on the specific DPP version in concert with the application. However, in general sources cancel operations by establishing no duration for the operation.

4.2.5. Operation Mutability

Not all state associated with an operation is invariant, but some is. For example, the operation identifier uniquely identifies the operation and is invariant. However, a source can rename the operation (view this contradiction as a cancellation followed by the establishment of a new operation). The duration, on the other hand, changes with each retry of the operation.

The situation of operation state gets more complicated when the application protocols are involved. It is important that protocol implementations be aware of what state can change and what state cannot change.

The purpose of this section is to define the common situations of mutable operation state, specify the common rules, and point out that the application must identify which state can change and which state cannot.

There are two cases when the state of an operation can change. First, when the source of the operation changes. Some operation state relates directly to the source, and so when the source changes then the state associated with the source must change. Second, state changes when the source retries the operation. A retry of the operation may change any state other than invariant state.

This leads to the following classifications for operation state:

1. Invariant: This state must not change for the duration of the operation. Any attempt to change this state will result in undefined behavior.

2. Mutable: This type of state changes over the lifetime of the operation through either a re-source or a retry. This is the default type for all state.

3. Source: This type of state has no meaning apart from the source of the operation, and so if the source changes then this state must be invalidated. Source state is mutable.

For DPP, only the operation identifier is invariant. Specific DPP versions may identify additional state, and should classify the type of state they introduce.

✓ **Treat all operation state as Mutable unless specified otherwise.** **`dof-2008-1-spec-24`**

This means that a retry of the operation may change all operation state not identified as Invariant.

✓ **Invalidate operation state identified as Source when the operation source changes.** `dof-2008-1-spec-25`

The DPP must notify application protocol implementations of source changes so that they may correctly invalidate Source state.

4.2.6. Operation Consolidation

In many application protocols, it is desirable to consolidate multiple operations into a single operation that has the same effect. This can result in both bandwidth and CPU benefits. However, the nature of the DOF Protocol Stack makes such consolidation difficult unless certain rules are followed precisely.

Each application protocol must define the specific rules for consolidating its own operations. This section highlights the difficulties that all application protocols must avoid. If an application protocol makes no mention of consolidation, then assume that consolidation is not possible.

Key to understanding consolidation is the description of DIRECTED and FLOODED operations presented above. A not so obvious effect of FLOODED operations is that the source node does not know, in general, if a receiving node 'accepts' the sender as the authoritative source of the operation.

For example, the following situations can arise:

1. The receiver does not accept the operation because of security issues (some security issues result in *silently* dropping the operation).

2. The receiver has a 'better' source for the operation. The loop prevention logic of the presentation layer may result in the use of a different source.

This lack of knowledge makes it difficult for a node to consolidate operations, because it does not know (again, in general) whether it will be the source of the consolidated operation. If it turns out that the sender is not the source, then the operation that has been 'merged' into the consolidated operation may remain unknown to the receiver (assuming that consolidation results in *not* forwarding the operation). In turn, this may result in the receiver's behavior being different from if it had used the sender as the source, making it impossible for the consolidating node to maintain the behavior of the merged operation.

In order to assure that consolidation is valid, a node must only consolidate operations that meet specific requirements. These requirements ensure that the receivers of the consolidated operation accept to the sender as the source of the operation (if they would accept the operation from any node).

In general, this means that consolidated operations must be compatible from a security perspective and either:

1. Be DIRECTED operations. A node that is sending a DIRECTED operation is following the DAG of some other operation (discussed above in the Operation Graphs section). DIRECTED operations are always 'owned' by the sender. This ensures that the receiver will treat the sender as the source. Senders can always consolidate operations that received from other nodes into these outbound DIRECTED operations (based on the application requirements).

2. Be FLOODED operations that created and managed by the sender. In this case the operation is not directed, but the sender 'owns' the operation. This results in the node being at the root of the DAG associated with the operation, and so that node is free to consolidate other operations into that operation (whether they are DIRECTED or FLOODED). However, the node must *not* create a new, FLOODED operation just in order to consolidate operations – the application on the node must manage the

FLOODED operation. To do otherwise may result in ringing and self-sustaining operations, assuming more than a single node tries to consolidate a single set of operations.

3. Be operations received from the same source. Nodes may consolidate these operations is discussed in the specific DPP version documentation because they are already linked to one another by being from the same source.

The first two rules actually consolidate to a single rule: nodes may consolidate operations that use the consolidator’s SID. This also guarantees that a single application (which uses a single SID by definition) is *always* able to consolidate its own operations.

The third rule indicates that nodes may consolidate operations from the same source as defined by each DPP version.

4.2.7. Priority of State Modifications

It is important that all implementations handle operation state changes in the same way. Typically, this is not an issue; however, the edge cases need clarification. These situations are difficult based on security considerations.

Nodes apply the following sequence for all operation state changes. Note that these steps are logical, and may be implemented in many equivalent ways. For each received operation:

- As an optimization, the receiver evaluates the packet before validation to determine whether it will drop the packet. Receivers do this only if the can make *no modifications to system state*. Receivers drop packets for many reasons, and this optimization allows bypassing the decryption/validation process in these cases.
- The receiver authenticates and validates the packet. Unsecured packets are assumed authenticated, secured packets must pass authentication and permission checks. Failures at this step cause the receiver to drop the packet as specified.
- The receiver updates the DPP state for the operation. This includes checks for retries, and any state managed by individual DPP versions. These checks may result in the receiver dropping the packet.
- The receiver updates the operation state. This includes duration modification (including cancelling of operations)
- The receiver passes the operation to the application for further processing.

> **Apply operation state changes in the defined order.** `dof-2008-1-spec-26`
>
> Receivers handle all modifications to operation state as defined in the stack specification.

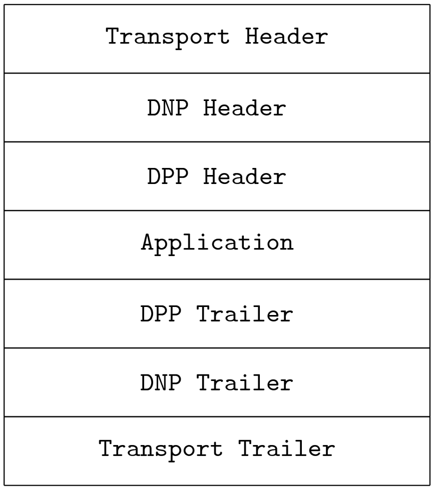
4.3. General Stack Sequence

The DOF Protocol Stack is composed of three general layers: the network layer, the presentation layer, and the application layer. A given PDU always begins with the network layer. Once the network layer has completed reading then the next byte begins the presentation layer. Once the presentation layer has completed reading then the next byte begins the application layer. The application layer completes the PDU. The layering is then reversed and associated trailers (if present) are read.

This strict layer does not mean that other control layers cannot be added to the stack. However, when they are added they must be understood (logically) as being part of the layer above them. For example, a transport layer could be added above the network layer. In doing this, however, a particular version of network layer would need to be created, and that layer would understand how to read the (new) transport layer. Once the new transport layer is read then the next byte must be the presentation layer.

Further, each layer except the application layer has an (optional) associated trailer. Each layer header defines a context for the next layer. The DOF Protocol Stack defines the standard context information for the stack itself, and only information specified in the context should pass from one layer to the next.

DOF Protocol Stack `dof-2008-1-pdu-1`



Transport Header
DNP Header
DPP Header
Application
DPP Trailer
DNP Trailer
Transport Trailer

`Transport Header`	**Variable-length, optional.** This is defined by the transport.
`DNP Header`	**Instance of** General DNP Header`dof-2008-1-pdu-2`.
`DPP Header`	**Instance of** General DPP Header`dof-2008-1-pdu-7`.
`Application`	**Instance of** General Application Header`dof-2008-1-pdu-13`.
`DPP Trailer`	**Instance of** General DPP Trailer`dof-2008-1-pdu-8`.
`DNP Trailer`	**Instance of** General DNP Trailer`dof-2008-1-pdu-3`.
`Transport Trailer`	**Variable-length, optional.** This is defined by the transport.

DNP version fixed for a single 2-node session. `dof-2008-1-spec-27`

The negotiated DNP version is not allowed to change during a single 2-node session. Any attempt to do so must result in the session being closed.

DPP version fixed for a single 2-node session. `dof-2008-1-spec-28`

The negotiated DPP version is not allowed to change during a single 2-node session. Any attempt to do so must result in the session being closed.

DNP version query (version 0) must be supported on all lossy servers. **`dof-2008-1-spec-29`**

All servers that support lossy transports must support DNP version 0 for datagram version query as described. This means that they must correctly respond to queries. They do not need to issue queries. DNP version 0 is not allowed on lossless transports.

DPP version query (version 0) must be supported on all lossy servers. **`dof-2008-1-spec-30`**

All servers that support lossy transports must support DPP version 0 for datagram version query as described. This means that they must correctly respond to queries. They do not need to issue queries. DPP version 0 is not allowed on lossless transports.

APPID version query (DSP) must be supported on all lossy servers. **`dof-2008-1-spec-31`**

All nodes that support lossy transports must support DSP for datagram version query as described. This means that they must correctly respond to queries. They do not need to issue queries.

APPID must be present in all lossless datagram after negotiation. **`dof-2008-1-spec-32`**

This means that each lossless datagram (excluding negotiation) must contain an DNP Header, DPP Header, and APP PDU.

4.3.1. Optimized Datagram Reading

The following discussion applies to reading DPS datagrams on streaming sessions. Datagram transports provide entire datagrams to the reader, and so there is usually not a question of how much data to read. However, on streaming sessions it is typical to obtain data as it arrives without regard to the datagram boundaries that were sent.

The issue is that issuing a read for more data than the datagram contains may cause an indefinite block (at least until another datagram arrives), and reading too few bytes, while safe, is not optimal for many implementations.

The ideal situation is to read the datagram in 2 read operations. The first read obtains enough of the datagram to completely determine the remaining length, and the second read obtains the rest of the datagram. In the worst case this may not be possible, and 3 read operations are required. This occurs when the datagram length changes (getting larger), and not enough data is obtained in the first read to determine the length.

In order to optimize behavior, it is useful to know the smallest datagram size possible, as it will always be possible to read that length. Referring to the DPS.1: DOF Protocol Stack definition; there are always at least three layers present. The DNP and DPP have a mandatory header byte, and the Application has a header that is at least one byte. This means that there are at least three bytes in any datagram. However, in the case of streaming sessions there must be length information present in the DNP. This means that the minimum datagram size is at least four bytes, with at least two bytes required *after* the DNP layer.

Also note that on streaming sessions that the DNP version (which determines the length) is negotiated (or asserted) immediately, and so the particular DNP version in use will be known and will not change.

On streaming transports, the DNPv1header has a minimum size of three bytes (version, flag, one byte length) and a typical maximum size of five bytes (version, flag, three byte length). Additional bytes after the length can be included with the rest of the PDU for this discussion. The question is whether it is always safe to read the additional two bytes. This is guaranteed to be true because the DOF Protocol Stack must always include an application protocol, and the required headers are at least two bytes long.

> Once the DNP version is known, then it can be asked what its own minimum header size is. It is always safe to read that amount *plus two bytes*, as the DPS requires that at least two bytes follow the DNP layer. As long as the size of the datagram can be determined by reading this combined minimum length then the datagram can be read in two read requests.

4.3.2. MTU

Each transport that supports the DPS defines an MTU or Maximum Transmission Unit. The MTU defines the maximum PDU length, based on the transport, and is independent of the capabilities of the protocol itself.

Independent of the transport, the maximum MTU for any transport is 2^{24}-1. This is a massive number, and typical environments will never need to handle datagrams that large.

> **The maximum DPS MTU is 2^{24}-1.** `dof-2008-1-spec-33`
>
> Stack layers will never need to work with a datagram that is larger than this limit. Typical datagrams will be much, much smaller. Imposing a limit can help in API design and data type choice.

Note that no current DPS version defines fragmentation, so there is currently no way to send an DPS datagram that is longer than the MTU on that transport. The solution to this problem is to use a streaming session and stream the datagram over that connection. If streaming sessions are not supported then the information can be fragmented at the application layer, with an interface allowing access to the different fragments of data or a streaming transport (below DPS) can be provided.

Transports may also define a 'minimum transport unit', or the minimum datagram size that can be sent. It is the responsibility of the DNP to provide methods for padding the datagrams so that any minimums are met.

4.4. Sessions and Servers

The basis of all DOF communication is a datagram, which was presented and defined in the previous section. Datagrams arrive and are sent using either servers or sessions. Servers do not maintain state, sessions do maintain state.

Connections, which are a special form of session, not only maintain state but monitor the session and can notify the application if the session stops operating.

In the definition of DPS context, each DOF operation indicates the session requirements that it has. Based on the operation and session, different state must be maintained (both in the operation and the session).

4.5. Transport

The specific requirements and definitions for the transport layer are covered in an earlier section. This section defines the context information required for PDUs sent **to** the transport.

The following information is required as context for transmitting a PDU:

1. SESSION: This information refers to the information associated with the datagram by the transport, server, or session that it arrived on. It encompasses the associated transport information, as well as the characteristics of the datagram/transport (such as streaming/datagram, lossless/lossy, etc.). These parameters were defined earlier in the section on Transports.

2. BUFFER: The data and associated length of the DNP PDU.

4.6. DOF Network Protocol

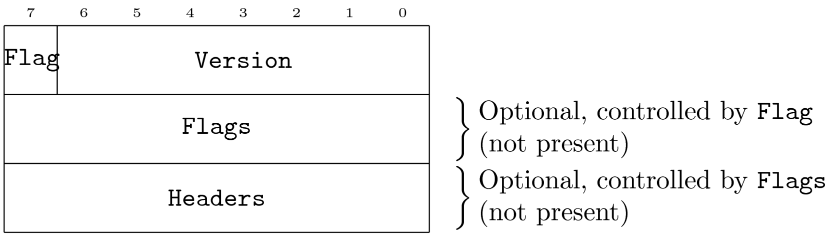
There is a common header format for all DPS Network Protocols. All DNP datagrams begin with a single byte header. This header defines the specific DNP version, and indicates whether flags bytes are present.

The specific format of the flag bytes and control fields (including their length and the individual meaning of each byte) is defined by each specific DNP. In general, if the protocol is not understood then the flags and fields are not understood.

As described above, the specific DNP in use on a lossless session must be negotiated. In this case, the negotiation must be done with *no flag bytes and no control fields*. Any attempt to use flags terminates negotiation and asserts a specific version.

This is the general specification for all versions of DNP.

General DNP Header **`dof-2008-1-pdu-2`**



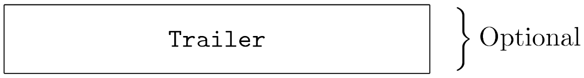
`Flag` **One bit.** Controls the presence of the `Flags` field, which is present if set.

`Version` **Seven bits.** Specifies the DNP version.

`Flags` **Variable length. Optional, controlled by `Flag`.** Contains options defined by the version.

`Headers` **Variable length. Optional, controlled by `Flags`.** Header fields required by the `Version` or controlled by `Flags`.

General DNP Trailer `dof-2008-1-pdu-3`



`Trailer` **Variable length. Optional, as needed.** Trailing data as defined by the `Version`.

Each specific version of the DNP should be available for use on all transports. This only means that DNP versions should be specified such that they can be used on many different transports (lossy and lossless, for example). Violations of this rule should only be allowed in order to gain a very specific benefit and should be extremely rare. This does not mean that all of the same control fields are required on all transports, but rather that control fields that are not applicable to all transports should be controlled by flag bits.

4.6.1. Context

The DNP layer sits on top of the Transport layer. As discussed in the Transport section, each transport provides sessions and servers, provides meaning for transport addresses, and deals with the different datagram types (unicast, multicast, broadcast).

The DNP itself is related to a SESSION. The SESSION includes information necessary to send and receive datagrams on a particular transport. Further, based on the transport, different SESSIONS may be related.

For each received PDU the DNP requires the following context information from the Transport:

1. SESSION: This is associated with the DNP. It includes the information discussed in the Transport section, specifically:

2. Any addressing and server required to send responses.

3. To the extent possible, the method used to receive the PDU (unicast, multicast, broadcast).

4. The type of session (streaming/datagram).

5. BUFFER: The received data and associated length. In the case of streaming sessions the received data and length may not represent an entire PDU (or may represent more than a single PDU). In this case, it is the responsibility of the DNP to continue obtaining BUFFER until a full DNP PDU has been read.

For each PDU that the DNP is asked to transmit, the context given must contain the following *in addition* to the information required of the transport context:

1. SESSION: This includes transport information including any target transport address if required.

2. BUFFER: The DPP PDU data and associated length. Any transport-defined minimum datagram lengths must be satisfied.

4.6.2. DOF Network Protocol Versions

There are potentially many different versions of this protocol. The specific version is always determined by the first header byte as shown above.

For a current list of registered versions, please contact the ODP-TSC or go to https://opendof.org/registry-dps-protocol.

DNP versions are registered with ODP-TC before use in products. **`dof-2008-1-spec-34`**

All DNP versions must be registered with the ODP-TSC before use in products.

The following describes behavior and requirements that apply to all versions of the DOF Network Protocol.

Reserved DNP versions that cannot be used. **`dof-2008-1-spec-35`**

The following DNP versions are reserved and are never valid. All of these versions collide with previous DOF releases that are not currently supported: 0x20 (version 3.1 and 3.2), 0x21 (version 3.1 and 3.2), 0x30 (version 3.1 and 3.2), 0x31 (version 3.1 and 3.2).

4.6.3. Flags

Each DNP version defines a 'default' value for its flags. The specific default values can be found in the documentation for the version. This default flag value must be independent of transport. After any negotiation datagrams, if the flags are not present then the default flag value must be assumed.

The flag bit of DNP must be set on all streaming datagrams except during negotiation. **`dof-2008-1-spec-36`**

This allows a single byte to determine whether negotiation is complete. Assuming in-order guaranteed delivery (the definition of stream transports) the first byte can be read. If the flag bit is clear then the datagram consists of two bytes (negotiation). If the flag bit is set then the datagram is not a negotiation datagram and more than a single byte must be read to determine the actual length.

4.6.4. DNP Required Functionality

DNP must provide for determining the size of DPP datagrams. It must also provide for any additional logical addressing in addition to that required by the transport.

4.6.5. DNP Logical Addressing

DNP logical addresses are an extension of transport logical addressing. Each datagram received can contain a specific source and destination DNP logical address. This information is encapsulated along with the transport address information to form the 'logical address' for DOF communications. The permitted values for DNP addresses include the special value NULL and integer values from zero (0) to 2^{22}-1 (4194303).

The specification of each DNP version will present the methods used to handle these requirements.

Default DNP address for lossless server-side session. **`dof-2008-1-spec-37`**

The server side of each lossless session must initially be associated with DNP address NULL. This allows negotiation to occur based on that target address. Each DNP version must use DNP address NULL for all traffic that does not explicitly include a different DNP address.

The server side of each lossless session must initially be associated with DNP address NULL. This allows negotiation to occur based on that target address. Each DNP version must use DNP address NULL for all traffic that does not explicitly include a different DNP address.

DNP addresses are associated with the transport address. This means that if the transport address changes for any reason that the associated DNP logical address is no longer valid. For example, a command that is received that includes a source DNP address but requires a multicast response would need to drop the command's DNP address in the response packet (because it is being sent to a different transport address).

4.7. DNP Discovery and Loopback

The following sections define the versions of DNP for version discovery and loopback detection.

4.7.1. DOF Network Protocol - Version 0 (DNPv0)

As described above, there are problems attempting to discover all of the different versions of the different DPS layers that are in use in a network. In order to solve this problem, all nodes are required to understand DNP version 0 in addition to at least one other version.

There are two PDUs defined for version 0. The first is a query, and it corresponds to a single byte header. There can be no flag bytes. The second is a response, which is sent some time after the query, and which corresponds to a list of DNP versions supported by the sending node. The format of this PDU is the header byte followed by a number of bytes each representing a version. There must be at least one additional version other than version 0 supported by a node, and so the PDU length (either the query consisting of all zero (0) bytes or the response that must contain at least one non-zero number) differentiates between a query and its response.

Each query is associated with a SESSION that includes a lossy server. This means, for example, that a node supporting multiple multicast addresses must track queries (as described below) for each SESSION.

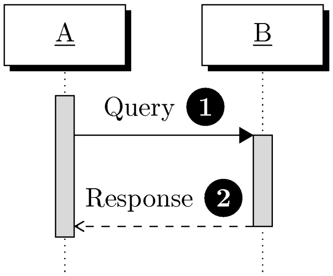
There are transports that require minimum datagram lengths. In order to support query on these transports, certain trailers are defined on both the query and response.

First, the query PDU may contain as many trailing zero (0) bytes as necessary to meet datagram length requirements. Query datagrams are then defined as datagrams consisting of all zero bytes.

Second, the response PDU may contain as many trailing zero (0) bytes as necessary to meet datagram length requirements. Response datagrams are then defined as a zero (0) byte, followed by some number of non-zero bytes, followed optionally by some number of zero (0) bytes. Note that there is a problem in determining the correct response to a request where there are multiple servers on a single transport address (differentiated by DNP port). As there is no way to identify which server should respond, the (single) response should aggregate all servers.

Responses to unicast queries are sent unicast.

Figure 4.1. DNP Unicast Query



1. Query is sent from A to B.

2. Response is sent from B to A immediately.

In the example above, node Query wants to determine the DNP versions supported by node A. It sends a unicast DNPv0 Query PDU to A. Since the PDU is unicast, node A immediately responds without any use of timers or delay.

Responses to multicast queries must be sent within three (3) minutes of a query, but the exact timing should be determined randomly and controlled by other network traffic as described below. Responses to unicast queries should be sent quickly.

For multicast queries, between the time of the query and the time of the response three events can prevent sending a version in the query response (that consists of a set of versions):

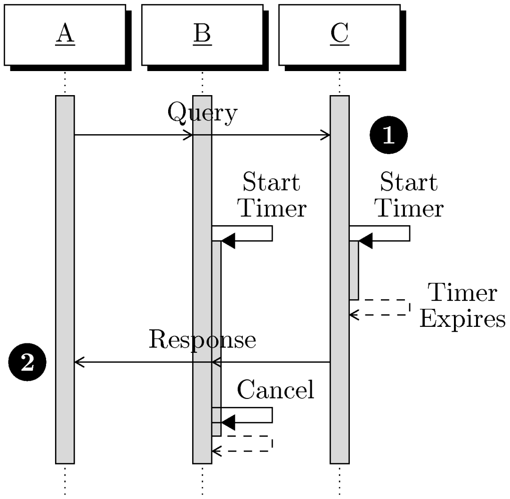
1. The node with a pending response sends a multicast PDU using the version on the same SESSION as the query.

2. A multicast PDU is received by the node with a pending response using the version on the same SESSION as the query.

3. A multicast query response is received from another node with a version listed on the same SESSION as the query.

The goal of this logic is to only send information that cannot be determined by watching network traffic. The specific goal of version detection is *not* to discover nodes on the network, but rather to just discover versions of protocols that are in use.

Figure 4.2. DNP Multicast Query



1. Query is sent from A to all nodes (multicast).

   - B and C both start randomized timers.

   - Timer on node C expires.

2. Response is sent from C to all nodes (multicast).

   - Node A receives response.

   - Node B cancels timer, preventing response.

In the example above, a node Query initiates version discovery by multicasting an DNPv0 Query PDU. This PDU is heard by nodes A and B. Both of these nodes pick random times within the next three minutes and start timers for when they will respond. The timer on B expires, and so B sends a multicast DNPv0 Query Response. This PDU contains the same version as A, and so in node A cancels its timer without sending a response.

If each version in the pending query response is covered by one of these cases then the query response is not sent. Otherwise at least the versions that have not been seen are sent.

For example, assume that a node N speaks version A and B of DNP. It receives a multicast PDU that is an DNP version 0 query. It determines (randomly) that it will send its response in 2 minutes. The response will be a PDU consisting of a zero byte (header), followed by A and B.

After some time and before the two minutes pass a PDU is multicast from node N using DNP version A. Sometime later, but still before the two minutes pass, a multicast PDU is received by node N using DNP version B. In this case, the pending response PDU is cancelled.

Assume however that the second PDU is not received. The 2 minutes pass. In this case a response containing at least B is sent.

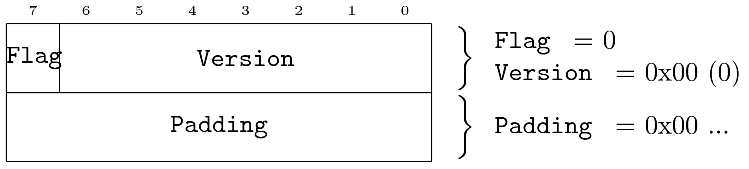
This protocol exists only to accomplish version discovery on a network. In particular, it cannot be used to encapsulate any other stack layers.

4.7.1.1. Query

Request all DNP versions on a network.
Session: None
Addressing: Unicast, Multicast, Unicast

⇄ **DNPv0 Query** `dof-2008-1-pdu-4`

Instance of General DNP Header`dof-2008-1-pdu-2`



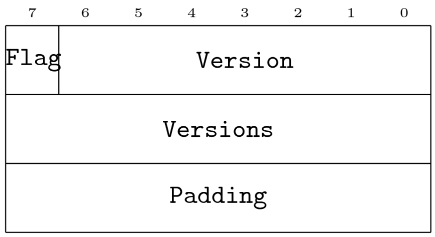
`Flag` **One bit.**

`Version` **Seven bits.**

`Padding` **Optional, variable length.** The padding consists of as many zero bytes as are necessary to meet minimum transport size requirements.

DNPv0 Query `dof-2008-1-pdu-5`

Instance of General DNP Header`dof-2008-1-pdu-2`



Flag = 0
Version = 0x00 (0)

Padding = 0x00 ...

`Flag` **One bit.**

`Version` **Seven bits.**

`Versions` **Variable length.** This list contains only non-zero bytes. Each byte must be a valid version, meaning that the most significant bit must be zero (0). At least one version must be present.

`Padding` **Optional, variable length.** The padding consists of as many zero bytes as are necessary to meet minimum transport size requirements.

DNPv0 Version List must be sorted in increasing order. `dof-2008-1-spec-38`

Sorting the list makes it easier to remove matching responses and find matching versions.

This version can only be used as described above in discovery on lossy transports.

In order to optimize traffic, responses to multicast queries are sent using multicast, not unicast.

4.7.2. DOF Network Protocol - Version 127 (DNPv127)

This section describes version one-hundred twenty seven (127) of the DOF Network Protocol. The protocol begins with a general one-byte header as described above. This byte contains the version (127).

DNPv127 must not include flags. `dof-2008-1-spec-39`

There are no flags for this DNP version, and so the high bit must be clear.

This version is somewhat special in that it is never really 'supported' by the DPS. It is defined, but its use is reserved for the DPS transports, and not the DPS itself. Because of this, version 127 is *not* advertised using DNPv0.

DNPv127 must not be advertised. `dof-2008-1-spec-40`

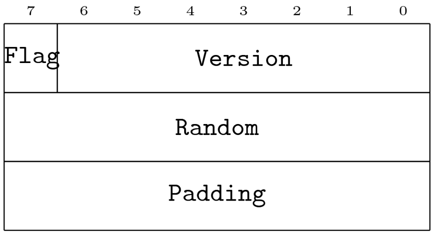
This version is for loopback detection only. It is not "supported" in the sense of encapsulating other layers, and so it is not advertised.

4.7.2.1. Check Loop

Send random data on the network that can be used to detect loopback.
Session: None
Addressing: Unicast, Multicast, Unicast

DNPv127 Header `dof-2008-1-pdu-6`

Instance of General DNP Header`dof-2008-1-pdu-2`



`Flag` $= 0$
`Version` $= 0\text{x}7\text{F}\ (127)$

`Flag` **One bit.**

`Version` **Seven bits.**

`Random` **Four bytes.** The sender must include these four bytes of random data.

`Padding` **Optional, variable length.** The sender may include additional random data, including any necessary to meet minimum transport size.

DNPv127 must only be used on lossy transports. `dof-2008-1-spec-41`

This DNP version must only be used on lossy transports, use on lossless transports should result in the session being closed or terminated.

This protocol version exists to aid transports in determining transport-level 'loopback'. Loopback is possible on some transports (based on both the transport and the addressing), and means that a datagram sent on the transport 'loops back' and is received by the sending node.

The DPS requires that the transport remove these 'echoed' datagrams, but it is not always possible for a transport to know when loopback will occur. In this case the transport can use this protocol version. To do this the transport determines a random payload and uses this protocol version to send the datagram. It then determines if this datagram (with matching random data) is received. If it is, then the sending address is flagged as 'ignored'.

Note that because this protocol version is not encrypted that there is a possible denial-of-service attack against a node. The attacker would listen for the use of this protocol version and then forge a datagram from some other node to the listening (sending) node. This would trick the node into ignoring an otherwise valid sender.

The danger of this attack is minimal. A node with the ability to carry out such an attack (to forge datagrams) would be able to easily interfere with communication in other ways.

The payload of this datagram is not specified apart from being random and at least four bytes long. Receivers must ignore the datagram (except, of course, for determining the presence of loopback). The datagram must not exceed the transport MTU, and must satisfy any minimum datagram size associated with the transport. It can accomplish the latter requirement by adding more random data to the datagram.

4.8. DOF Presentation Protocol

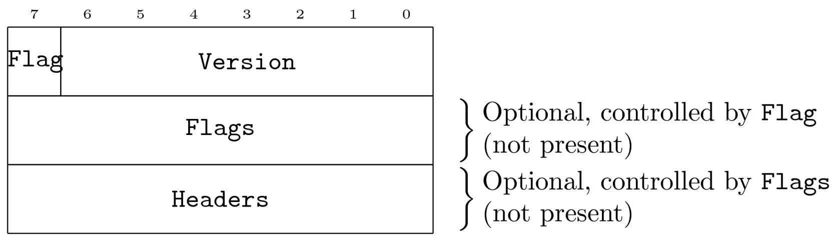
There is a common header format for all DPS Presentation Protocols. All DPPs begin with a single byte header. This header defines the specific DPP version, and also indicates whether or not flag bytes and control fields are present.

The specific format of the flag bytes (including their length and the individual meaning of each byte) is defined by each specific DPP. In general, if you do not know the protocol then you cannot understand the flags.

> As described above, the specific DPP version in use on a lossless session must be negotiated. In this case, the negotiation must be done with *no flag bytes and no control fields*. Any attempt to use flags terminates negotiation and asserts a specific version.

This is the general specification for all versions of DPP.

General DPP Header `dof-2008-1-pdu-7`



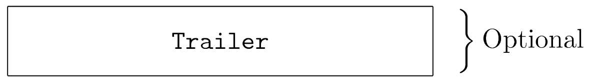
`Flag` **One bit.** Controls the presence of the `Flags` field, which is present if set.

`Version` **Seven bits.** Specifies the DPP version.

`Flags` **Variable length. Optional, controlled by `Flag`.** Contains options defined by the version.

`Headers` **Variable length. Optional, controlled by `Flags`.** Header fields required by the `Version` or controlled by `Flags`.

General DPP Trailer `dof-2008-1-pdu-8`



`Trailer` **Variable length. Optional, as needed.** Trailing data as defined by the `Version`.

In general, gateway nodes should understand all DPPs. Device nodes may understand a single DPP. This may limit communication between device nodes except in the presence of a gateway.

Each specific DPP should be available for use on all transports. Violations of this rule should only be allowed in order to gain a very specific benefit and should be extremely rare. This does not mean that all of the same flags and control fields are required on all transports, but rather that control fields that are not applicable to all transports should be controlled by flag bits.

4.8.1. Context

The DPP layer sits on top of the DNP layer.

For each received PDU the DPP requires the following context information from the DNP, in addition to the context provided by the Transport:

SADDR

The DNP address associated with the sender (the DNP can add a layer of logical address to the transport). This address may be used with a different server (with a multicast or broadcast address type) to send a datagram to the sending node.

BUFFER

The received data and associated (DPP) length. For each PDU that the DPP is asked to transmit, the context given must contain the following *in addition* to the information required of the DNP and transport context:

COMMAND/RESPONSE

Whether the PDU represents a command or a response.

ENCRYPT,
AUTHENTICATE

Determined by the security requirements of the PDU.

BUFFER

The DPP PDU data and associated length.

In addition, for each COMMAND the following must be specified:

DIRECTED/FLOODED

Whether the PDU is directed or will potentially loop back to the sender. FLOODED requires that OPID and a non-zero DURATION be specified.

SINGLERESPONSE/
MULTIRESPONSE

Indicates that a single response will complete the operation and can remove the associated DPP state, or for MULTIRESPONSE that multiple responses may be received. Note that since DPP has no application knowledge, it is important that the application correctly set this indicator. DPP will remove the state on the command (precluding further responses) if SINGLERESPONSE is set, even if the application allows multiple responses.

OPID/NOOPID

If present, represents the Operation Identifier associated with the PDU. For NOOPID, specifies that no operation identifier is required (although operation identifiers may always be present).

RETRY

Indicates that the command is a retry. This requires that OPID be specified.

DURATION

Indicates the duration of the operation. Each DPP may have different limits on the durations allowed, and so the DPP should return the actual duration that will be understood by receivers of the datagram.

Finally, for each RESPONSE the following must be specified:

FINAL/INTERMEDIATE

Whether the response is FINAL or INTERMEDIATE. INTERMEDIATE responses do not complete an operation, even if SINGLERESPONSE is specified on the command. This only has effect on SINGLERESPONSE commands, and is ignored for MULTIRESPONSE.

4.8.2. DOF Presentation Protocol Versions

There are potentially many different versions of this protocol. The specific version is always determined by the first header byte as shown above.

> For a current list of registered versions, please contact the ODP-TSC or go to https://opendof.org/registry-dps-protocol.

> **DPP versions are registered with ODP-TSC before use in products.** **`dof-2008-1-spec-42`**
>
> All DPP versions must be registered with the ODP-TSC before use in products.

4.8.3. Flags

Each DPP version defines a 'default' value for its flags. The specific default values can be found in the documentation for the version. During the negotiation phase no flags are sent, and the meaning of the default flag value is also ignored. After negotiation, if the flag byte is not present then the default value must be assumed.

4.8.4. Encryption and Message Authentication

One of the major purposes of DPP is to protect data through encryption and message authentication. Note that DPP does *not* provide authentication or key distribution, which are provided by specific application protocols. It is the job, however, of DPP to actually encrypt and decrypt the datagram.

Each DPP version defines how it provides security, although specific security modes of operation are leveraged. DPP is limited in this role to the information provided to it in the context of the datagram, either from layers above (like encryption keys), information that it transmits (fields) and information from the network protocol.

DPP is told that a particular datagram must be secure, and whether encryption or message authentication is required. It also allows the application protocols to determine whether a particular received datagram was secure, and how (authenticated, encrypted). The specifics of how this information is passed on the wire are dependent on the DPP version, and the method by which an application protocol obtains the information is dependent on the DPP implementation. DPP will likely leverage other protocols (security modes of operation) to actually encrypt and authenticate the information, and pass the security-related information on the wire.

However, all secure datagrams (whether authenticated or encrypted) must ensure that no DPP or Application data is modified on the wire.

> **Secure datagrams (authenticated or encrypted) must be rejected if DPP or Application data is modified in transit.** **`dof-2008-1-spec-43`**
>
> This requirement begins with the DPP header (including the version and flag bit) and extends to the end of the DPP trailer. This requirement does not require encryption of this information, only validation. How this is accomplished depends on the Security Mode.

Each security mode of operation must define the security fields that are required for each PDU. In order to abstract the security headers that are dependent on different security modes of operation, the following placeholders are defined.

Encryption always begins either in or immediately after the DPP.9: Security Mode of Operation Header Fields. This ensures that all application data is encrypted if required. Encryption always ends at the beginning of DPP.11: Security Mode of Operation Trailer Fields. Finally, the DPP.11: Security Mode of Operation Trailer Fields are always at the end of DPP.10: General DPP Trailer.

This definition means that the process of encrypting and decrypting datagrams can be entirely defined by the following information:

1. The DPP BUFFER (beginning with the DPP header byte, to the last DPP trailer byte).

2. The offset in the BUFFER of DPP.9: Security Mode of Operation Header Fields.

Security Mode of Operation Header and Trailers are placed correctly, defining encryption boundaries. `dof-2008-1-spec-44`

This specification allows the mode of operation to easily determine header and trailer placement, and to encrypt and decrypt datagrams as necessary.

Security Mode of Operation Header `dof-2008-1-pdu-9`

Fields

`Fields` **Variable.** The format of the data is controlle dby the mode of operation, which is always known in the context.

Security Mode of Operation Trailer `dof-2008-1-pdu-10`

Fields

`Fields` **Variable.** The format of the data is controlle dby the mode of operation, which is always known in the context.

4.8.5. Common DPP Capabilities

Each version of DPP (exception version 0) must provide the common functionality described in this section. Application protocols may rely on the availability of this information independent of the actual DPP version being used. Version 0 has no common behaviors and no context.

Most of the required functionality deals with managing operations and operation lifecycle, discussed above.

Rather than encapsulate different PDUs in the DPP header space, the application identifier 0x7FFF is reserved for DPP. This does not mean that the implementation needs to be identical for each DPP version, only that the application layer can be leveraged by DPP. This includes the ability to use security.

✓ **DPP Common Behaviors must be accepted as soon as DPP is negotiated.**

`dof-2008-1-spec-45`

This ensures that the common behaviors are available as soon as DPP is.

The following behaviors are required. Specific DPP versions may define additional capabilities, and they would be described in the specific version specification document.

It is important to note that the specifications and behaviors of these behaviors are defined here in a common way. The individual versions of DPP only have control over the specific formatting of the commands and responses that they provide.

4.8.5.1. Command/Response

DPP must allow each PDU to be classified as a command or a response. This is a major categorization, and application protocols must specify whether a PDU is classified as a command or a response. The command/response classification is independent of whether the PDU uses command identification (operation identifiers).

4.8.5.2. Command Identification and Lifecycle

DPP must allow commands to be uniquely identified, and must allow the relationship of command and response to be maintained if they are identified. The method for command/response identification is standardized and all DPP versions must support the same format. DPP must allow for unidentified commands, although unidentified commands have no DPP lifecycle. They may have application-defined lifecycle.

DPP must allow the lifecycle of an identified operation to be tracked. This includes the stages described above in the Lifecycle section. Note that lifecycle tracking requires command identification. Unidentified operations have no lifecycle and immediately enter the DPP Cancelled state. Any lifecycle information present in a PDU must be ignored and should be cleared for PDUs that do not include identification. However, the information must be available to the application.

In particular, each DPP version must allow for operations that are complete on response. This is a special case of an identified command that uses lifecycle, but is automatically completed when a final response is processed. DPP responses must allow for both intermediate and final responses. An intermediate response does not complete an operation (even if marked complete on response). A final response will complete an operation that is complete on response.

4.8.5.3. SID Security

DPP versions that support security must define and negotiate the permissions necessary for using a given SID on the wire, including the behaviors of IAM and ACTAS.

4.8.5.4. Loop Prevention

DPP must handle the case of flooded operations. Flooded operations may 'loop', and this can cause ringing to occur if not managed. It is DPP that defines how to prevent loops. There is overhead for this determination, and so it is enabled by specifying FLOODED in the context.

It should be possible for a receiver to determine whether an operation is DIRECTED or FLOODED.

4.8.5.5. Source Tracking

It is the responsibility of DPP to manage the *source* of each identified operation. Throughout the network, tracking an operation along its sources will lead directly (with no looping) to the originator of the operation.

Another view of operating sources is that the source 'pointers' form a tree for each operation, with the root being the originator of the operation.

The transport SESSION and DPP SADDR are used to define the source.

4.8.5.6. Operation Management

Operation management is closely related to operation lifecycle. This section covers the common DPP capabilities that are related to operation management.

When a program starts it must determine the SID to use. The determination of the SID is discussed earlier. It is possible that a node will know (when it starts) that it was previously using a different SID. In this case, it is appropriate to cancel all operations associated with the prior SID. For example, a program may use a persistent number (like a sequence number) to make its SID unique. When it starts up it can read the last sequence (corresponding to the last SID), cancel it, determine the new SID, and store it. The ability to cancel all operations that use a given SID is a common DPP capability (Cancel All).

When a node shuts down both the transport address and the SID become invalid. When a node leaves a transport then the transport address becomes invalid. Both of these cases impact the operations that have been created or forwarded by the node. In this case the common capabilities of Cancel All and Node Down can be used. These commands are always used by the source node, and they do not flood and are not forwarded. The effects of these operations do have effects that are propagated, although not using the same commands.

Nodes that receive the Cancel All command must immediately cancel any operations where the SID of the operation matches the SID passed in the Cancel All command. These cancels must be handled through the application protocol, which may cause additional cascading cancels (using different commands) on the network. Note that this rule applies to all operations that use the SID, independent of the transport addresses.

Nodes that receive the Node Down command do the same as Cancel All, and in addition cause all operations that share the source transport address to enter the Operation:Lost state.

Whenever a SID becomes invalid using either of these methods, it should not be used again for some time so that the all operations throughout the network can be cancelled. This prevents a type of operation identifier aliasing which can occur if a SID is reused quickly.

> The management of SIDs is a complex topic. In general, nodes will either pick a completely random SID (which can be large on the network), or will base the SID on some physical property that the node possesses (like a MAC address). However, when multiple programs run on the same node then they must synchronize their SID usage so that no SID collisions occur. This can be handled by using an incrementing number in association with the SID based on a physical property. If the SID is known to be unique then there are no chances for SID collisions.

> **Nodes should wait at least 16 seconds after a Cancel All or Node Down before reusing a SID.** `dof-2008-1-spec-46`
>
> Nodes that send the Node Down or Cancel All command should not use the referenced SID for at least sixteen (16) seconds. This requirement prevents race conditions that can cause operation aliasing. If a new operation is created it may match a previously used operation identifier that is in

the process of being cancelled. The delay ensures that the system is stable before new operations are created. This does not apply if a new unique SID is chosen and used, as aliasing is not possible with a new, unique SID.

Use and respond to the Node Down and Cancel All commands. `dof-2008-1-spec-47`

Nodes should listen for and process the Node Down and Cancel All commands. Nodes should also send the Node Down command before leaving a transport (shutting down), and should also send it (and delay as discussed above) if there is a chance that there are orphaned operations in the network. This would be a typical situation if a node rebooted quickly, for example.

During the lifespan of an operation, it is possible for nodes to lose communication capabilities with the source of an operation. Note that in a typical case a node will explicitly cancel operations before it leaves a network, and so the source lost behavior is not typical. Also note that operation timeout is distinct from loosing communication with the source. Loss of communication is determined by the implementation, usually through some defined connection logic.

When communication with the source is lost it is *not* appropriate for the node to assume that the operation is cancelled, because there may be alternate paths to the original source that can continue to maintain the operation state. In general, the DOF specification does not require that these alternate sources be tracked, and this means that when a source is lost that a search for a new source should take place. All nodes should make use of the Source Lost command, although non-routing nodes are not required to unless they track operation sources. Nodes that forward operations to other nodes (routing nodes) must use Source Lost as appropriate.

The goal of this search is either that the operation is cancelled because no suitable source can be found, or the operation is re-established with a new source (or maybe the same, but newly validated source). This search is done using a flood technique, but with different logic than normal operation flooding. The reason for this modified flooding to avoid the overhead of creating operations along with the fact that loops can be prevented through the state of the existing operations.

Note that there are two different types of operations: local and global. Local operations are likely understood by a single other node, while global operations are likely to be known by many nodes. In the case of searching for a source, the desire is not to do an exhaustive search of all nodes in the network, only those that are reachable by the node that lost the source and could become the source.

For example, it makes no sense to query the entire network about an operation that is known only to a single other node (the source of a local operation).

There are two commands involved in the search: Source Lost and Source Found.

The process begins with a node realizing that the source of an existing operation has been lost. This can occur because either the communication between the node and the source fails (requiring that a session exists) or because the source of the operation sends a Source Lost command. A node with a lost source (for either of these two reasons) does two things:

- Sets the state of the operation to Lost and shortens its current timeout to 16 seconds if it is currently greater. Operations in this state will accept any compatible source as the new source for the operation (independent of the current operation state).

- If the operation was global, forwards a Source Lost request to all compatible sessions, including the operation identifier. A compatible session is one that would be acceptable as a new source, without fundamental changes to the operation. This usually means that the security information is the same.

- If the operation was local, forwards a Source Lost request on any session that could reach the SID of the lost operation in a single hop. If this is not known, then it may use the same sessions as in the global case, setting the Sequence of the Source Lost command to 255, limiting the propagation to a single hop.

When a node receives a Source Lost request from another node *that is not its source for the operation* it:

- Checks to see if the operation is known. If it is not, the request is ignored and not forwarded.
- Checks if the operation is flagged as 'Proxy Lost' already. If it is, the request is ignored and not forwarded.
- Otherwise, it validates the operation. If the node is *positive* that the operation still exists then it follows the process for Source Found. This would normally only be true if the node itself sourced the operation.
- Otherwise, it flags the operation as 'Proxy Lost' and forwards a Source Lost to the source of the operation. The 'Proxy Lost' flag is in effect for 16 seconds and cleared through the process of Source Found. Operations with this flag set ignore Source Found commands except when sent from their source.

If the source remains lost, meaning that no suitable source can be identified in the Lost state, then the operation will timeout after a maximum of 16 seconds. If, on the other hand, a suitable source is found then it will begin the Source Found process.

The Source Found command begins the process of undoing the effects of Source Lost. Note that it is not possible for intermediate nodes to use an application-layer retry of the operation to 're-source', although the original sender may use a retry in place of Source Found. Note that Source Found is sent to all compatible sessions, but it is not a 'flooded' operation as it does not follow the DPP rules for flooded operations.

In the case of Source Found, the receiving nodes follow substantially the same rules as for a retry of the specified operation, modifying their state similar to if they had received a retry.

The receiving node:

- Checks to see if the operation is known. If it is not, the request is ignored and not forwarded.
- Checks to see if the operation is in the 'Source Lost' state or has the 'Proxy Lost' flag set. If not, then the request is ignored and not forwarded.
- If the operation is in the 'Source Lost' state, the sender is accepted as the source and other DPP state is updated. In this case the application is told of the new source. This has the effect of clearing the 'Source Lost' state. The operation is then forwarded to all compatible sessions.
- If the operation has the 'Proxy Lost' flag and the sender is not the source of the operation then the operation is ignored and not forwarded. Otherwise, the sender is the source and the 'Proxy Lost' flag is cleared and the operation forwarded to all compatible sessions.

Note that receiving a retry of an operation has the same effect as a Source Found, although in the case of a retry the application is provided more information about the operation (the application data is missing in the Source Found case).

✓ **Use and respond to the source lost and source found requests. `dof-2008-1-spec-48`**

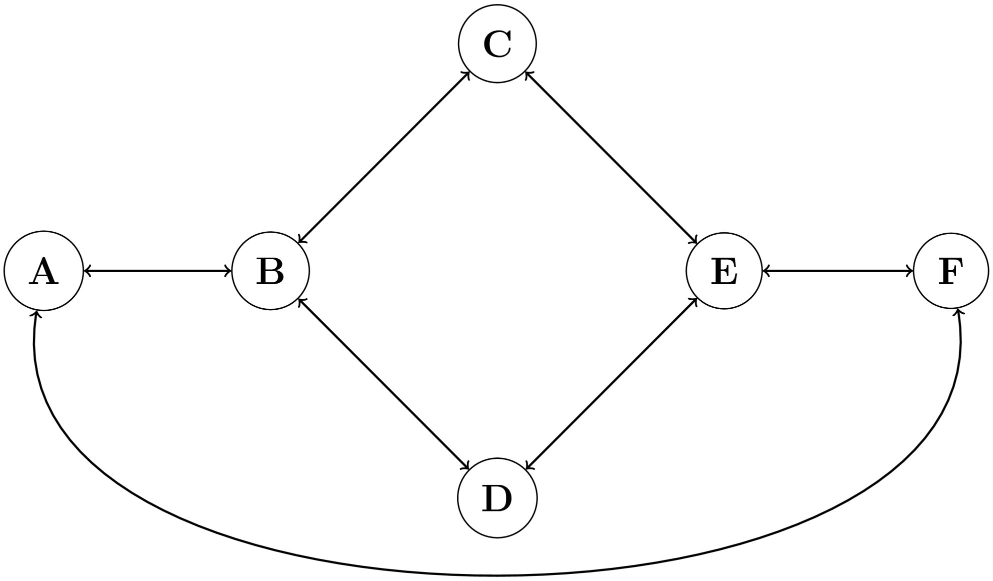
Nodes must listen for and process the Source Lost request. Nodes must also send and process the Source Found command as discussed.

Use operation retry for all operations that include source state. `dof-2008-1-spec-49`

Source Found may only be used for operations that do not contain Source State. Operations that include Source State must use an operation retry in response to Source Lost.

Because of the complex behavior of Source Lost and Source Found a detailed example is now provided. We will assume the following set of sessions. Keep in mind that whether these sessions are connections, groups or connectionless isn't important – only the fact that they are compatible. We are going to focus on a single operation which is started at node A.

Figure 4.3. DPP Source Lost and Found Example Network



This network has 6 nodes and 7 sessions. We will assume that the global operation begins at node A with a delay of 600 seconds. We can draw the current state of this network as follows (this information is shown in a single table, but is actually distributed through the network):

Node	Source	Original Source?	Normal (S), Proxy Lost (PL), Lost (L)	Remaining
A	-	Y	N	600
B	A	N	N	600
C	B	N	N	600
D	B	N	N	600
E	F	N	N	600
F	A	N	N	600

This table represents the stable state after the operation has propagated throughout the network.

Now we will simulate the effect of the B to D session going down without notice (meaning without a Node Down PDU) after 30 seconds. In this case node D sees that the parent (B) of the operation is not available. This will initiate the Source Lost handling. The Source Lost PDU will contain the operation identifier, and will flood through the network. Note that because the network is fully connected that all nodes will either enter the 'Lost' state or be flagged as Proxy Lost based on whether the sender is their source (except A). The state of the system after the flooding of the Source Lost would be:

Node	Source	Original Source?	Normal (S), Proxy Lost (PL), Lost (L)	Remaining
A	-	Y	N	570
B	A	N	PL	570
C	B	N	PL	570
D	?	N	L	16
E	F	N	PL	570
F	A	N	PL	570

Node A will potentially receive two Node Lost PDUs: one from F and another from B. Each will trigger a Source Found process. This Source Found will reset the state of the table as shown:

Node	Source	Original Source?	Normal (S), Proxy Lost (PL), Lost (L)	Remaining
A	-	Y	N	570
B	A	N	N	570
C	B	N	N	570
D	E	N	N	570
E	F	N	N	570
F	A	N	N	570

Note that the parent of node D has changed. Depending on the specifics of the protocol version other parents may change as well.

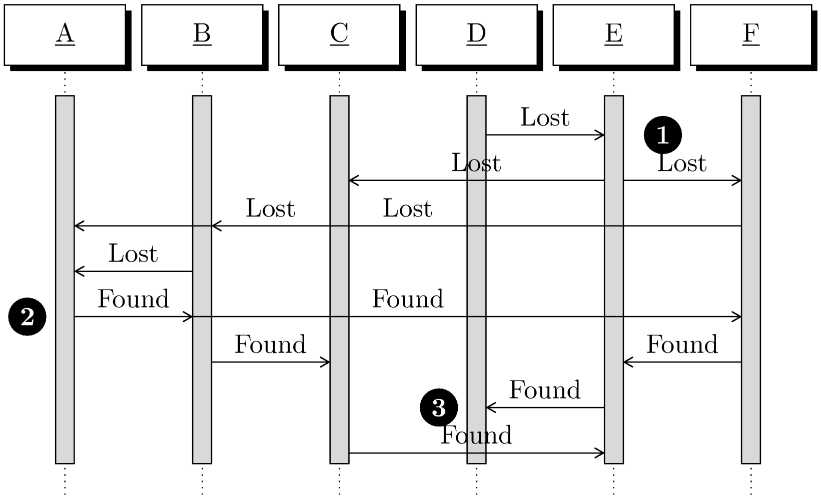
Now we are going to simulate the effect of the A to B session and the E to F session going down at the same time. The Source Lost will propagate, although this time the network is no longer fully connected and so the scope is different. The result looks like this:

Node	Source	Original Source?	Normal (S), Proxy Lost (PL), Lost (L)	Remaining
A	-	Y	N	570
B	?	N	L	16
C	?	N	L	16
D	?	N	L	16
E	?	N	L	16
F	A	N	N	570

Since no authoritative source was found, no Source Found PDUs were created. In this case the operation will time out after the 16 second delay on nodes B, C, D, and E.

The following sequence diagram shows the resulting datagrams on the wire in both the first and second examples:

Figure 4.4. DPP Lost and Found Example 1



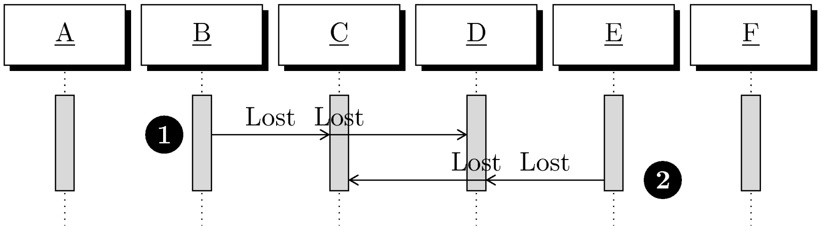
1. D determines that the connection with B is lost and initiates Source Lost.

2. A receives Source Lost and as the original parent begins Source Found.

3. D receives Source Found from E, who becomes the new parent.

This example is a hypothetical example of the sequence of PDUs. As shown there is no Lost from F to A. This is because the Found in step 7 arrived before it would have sent the Lost.

Figure 4.5. DPP Lost and Found Example 2



1. B enters Source Lost and notifies C and D.

   - Not shown that C and D will send Source Lost to E.

2. E enters Source Lost and notifies C and D.

Note that in this example node A and F receive no messages because their sessions to the other nodes were broken. The loss is localized to the isolated section of the network.

Certain nodes will consolidate operations and only forward a single representative operation instead of multiple operations. This determination cannot be made by the DPP itself because it requires application knowledge. However, since operation consolidation involves the DPP it must be able to support it.

This kind of behavior creates a relationship (which must be managed by the application) between two or more input operations and a single output operation.

An issue that arises in this case is how to handle a cancellation or timeout of a single 'input' operation that uses the operation identifier that was forwarded and represents the consolidated operation. This problem is solved with the Operation Rename request.

✓ **Correctly handle and forward Operation Rename request from the operation parent.** `dof-2008-1-spec-50`

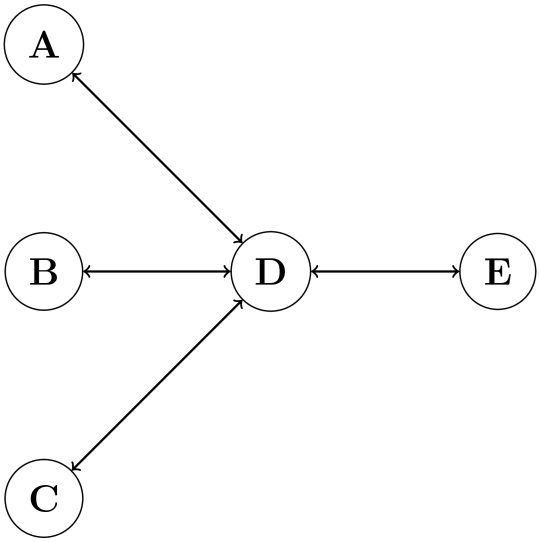
When Operation Rename is received from the parent of an operation then it must be handled and forwarded as appropriate.

The alternate solution would be to use a cancel and then begin a new operation; however this solution causes more traffic and may cause instability if used frequently. The single Operation Rename accomplishes both tasks in a single request. When a node receives this operation it verifies that the referenced operation identifier (in the payload) exists on the node and that the request is coming from the parent and on a compatible session. If it does not then the request is silently discarded. If the operation does exist, then:

- The DPP state associated with the referenced operation is replaced with the state of the Operation Rename.
- The application associated with the operation is notified of the rename.
- The PDU is forwarded. The application should guide this forwarding, but if not then it must be flooded.

The following example shows how this operation could be used. Consider five nodes, A, B, C, D, and E.

Figure 4.6. DPP Operation Rename Example Network

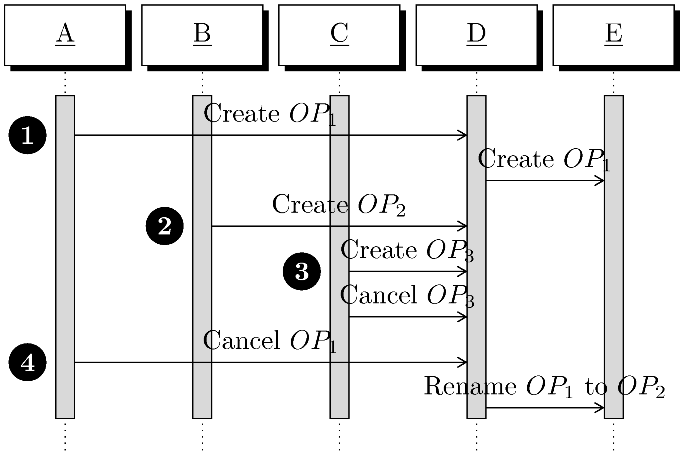


There are sessions established as shown in the diagram. Node A initially creates an operation OP_1 on D, which D then forwards to E. At some later time node B establishes OP_2 on D. The application knows that the effects of OP1 and OP2 are the same, and so this second operation is not forwarded to node E. The same occurs later between node C regarding OP_3.

At this point each of the nodes A, B, C, and E has a single operation (although not all the same). Node D is aware of all three operations. Assume that OP_3 is cancelled. In this case node D needs to handle this, but node E is not concerned with OP_3.

Now consider the case where OP_1 is cancelled (or otherwise terminates). The effects of the operation need to remain in order to satisfy OP_2. Forwarding OP_2 to node E is an option, but that may have other side effects that are not desired. In this case node D can use Operation Rename on OP_1, renaming it to OP_2. The sequence looks like this:

Figure 4.7. DPP Rename Example



1. A creates OP_1 and forwards to D, which forwards it to E.
   - Note that the operation creation between A, B, C and D are not shown for simplicity.
2. B creates OP_2 and forwards to D, which does nothing as the operation has the same effect as OP_1.
3. C creates and then cancels OP_3, for which D does nothing.
4. A cancels OP_1. D then renames OP_1 to OP_2 to preserve the effect on node E.

Note that the Rename in step 7 is very similar to a cancel to that node. Assuming that node E had forwarded OP_1 to other nodes, it must propagate the Rename in order to complete the re-association.

4.8.5.7. Ping and Ping Response

There are situations in which a node desires to know that another node is still operating. This is accomplished with a Ping command and a Ping response.

✓ **Respond correctly and within 10 seconds to Ping PDUs.** `dof-2008-1-spec-51`

When a transport unicast Ping is received a response must be sent. Multicast and broadcast pings should be silently ignored unless it is equivalent to transport unicast as determined by the security mode of operation. A response (meaning any traffic) should be received within ten (10) seconds of the ping. Secure pings must result in secure responses.

The Ping command requests that the receiver send traffic to the sender. It is permissible for the node to send *any* directed (not multicast/broadcast) traffic, not just a Ping response. However, if there is no other traffic to send then the Ping response can be sent.

Multicast and broadcast commands should be ignored. However, it is difficult on some transports to determine with precision whether a command is received multicast or broadcast. In these cases it is allowable to send a response.

4.8.5.8. Heartbeat

A heartbeat command has no response. It is useful when a transport requires that a PDU be sent, but no behavior is desired or required. All DPP fields are ignored for the heartbeat.

4.8.5.9. Keep Alive

There can be issues on some streaming transports like TCP/IP with determining when a session is broken. This is typically a problem where the connectivity is lost inside the network (for example, between routers). If there is no traffic on the session then the fact that the session no longer exists may not be known to the application for a long time.

If the stack cares about this case, it can be desirable to pass information over the session. The ping or heartbeat PDUs described above may be used in this case. The application may choose whether a response is desired (Ping) or whether just sending a datagram will do (Heartbeat).

Whether a particular implementation includes the use of 'keep-alive' datagrams is up to the implementation. They are not required in order for the protocol stack to function.

4.9. DPP Discovery

The following DPP protocol versions are used for DPP discovery.

4.9.1. DOF Presentation Protocol - Version 0 (DPPv0)

As described earlier, there are problems attempting to discover all of the different versions of the different DPS layers that are in use in a network. In order to solve this problem, all nodes are required to understand version 0 in addition to at least one other version.

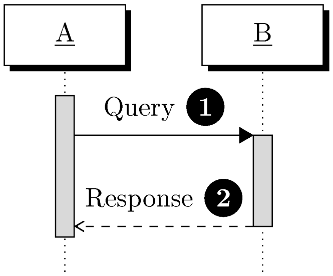
The use of DPPv0 is described here. It may only be used with a non-zero version for DNP. In other words, once a particular DNP version is known then DPP versions can be queried.

There are two PDUs defined for version 0. The first is a query, and it corresponds to a single byte header. There can be no flags. The second is a response, which is sent some time after the query, and which corresponds to a list of DPP versions supported by the sending node. The format of this PDU is the header byte followed by a number of bytes each representing a version. There must be at least one additional version other than version 0 supported by a node, and so the PDU length (either the one-byte query or the multi-byte response) differentiates between a query and its response.

Responses to multicast queries must be sent within three (3) minutes of a query, but the exact timing should be determined randomly and controlled by other network traffic as described below. Responses to unicast queries should be sent quickly.

Responses to unicast queries are sent unicast.

Figure 4.8. DPP Unicast Query



1. Query is sent from A to B.

2. Response is sent from B to A immediately.

In the example above, node 2 wants to determine the DPP versions supported by node R2. It sends a unicast DPPv0 Query PDU to R2. Since the PDU is unicast, node R2 immediately responds without any use of timers or delay.

For multicast queries, between the time of the query and the time of the response there are three events that can prevent sending a version in the query response (that consists of a set of versions):

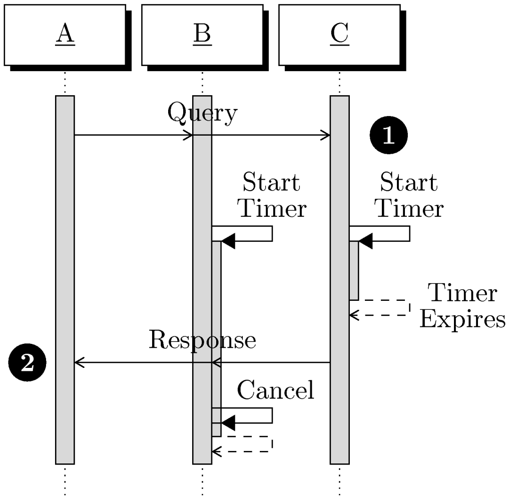
- The node with a pending response sends a multicast PDU using the version on the same SESSION as the query.

- A multicast PDU is received by the node with a pending response using the version on the same SESSION as the query.

- A multicast query response is received from another node with a version listed on the same SESSION as the query.

The goal of this logic is to only send information that cannot be determined by watching network traffic. The specific goal of version detection is *not* to discover nodes on the network, but rather to just discover versions of protocols that are in use.

Figure 4.9. DPP Multicast Query



1. Query is sent from A to all nodes (multicast).

   - B and C both start randomized timers.

   - Timer on node C expires.

2. Response is sent from C to all nodes (multicast).

   - Node A receives response.

   - Node B cancels timer, preventing response.

In the example above, a node Q initiates version discovery by multicasting an DPPv0 Query PDU. This PDU is heard by nodes R2 and R3. Both of these nodes pick random times within the next three minutes and start timers for when they will respond. The timer on R3 expires, and so R3 sends a multicast DPPv0 Query Response. This PDU contains the same version as R2, and so node R2 cancels its timer without sending a response.

If each version in the pending query response is covered by one of these cases then the query response is not sent. Otherwise at least the versions that have not been seen are sent.

For example, assume that a node N speaks version A and B of DPP. It receives a PDU that is an DPP version 0 query. It determines (randomly) that it will send its response in 2 minutes. The response will be a PDU consisting of a zero byte (header), followed by A and B (along with whatever DNP bytes are required).

After some time and before the two minutes pass a PDU is multicast from node N using DPP version A. Sometime later, but still before the two minutes pass, a multicast PDU is received by node N using DPP version B. In this case, the pending response PDU is cancelled.

Assume however that the second PDU is not received. The 2 minutes pass. In this case a response containing at least B is sent.

This protocol exists only to accomplish version discovery on a network. In particular, it cannot be used to encapsulate any other stack layers.

4.9.1.1. Query

Request all DPP versions on a network.
Session: None
Addressing: Unicast, Multicast, Unicast

DPPv0 Query `dof-2008-1-pdu-11`

Instance of General DPP Header`dof-2008-1-pdu-7`

7	6	5	4	3	2	1	0
Flag	Version						

`Flag` $= 0$
`Version` $= 0\text{x}00\ (0)$

`Flag` **One bit.**

`Version` **Seven bits.**

DPPv0 Query Response `dof-2008-1-pdu-12`

Instance of General DPP Header`dof-2008-1-pdu-7`

7	6	5	4	3	2	1	0
Flag	Version						
Versions							

`Flag` $= 0$
`Version` $= 0\text{x}00\ (0)$

`Flag` **One bit.**

`Version` **Seven bits.**

`Versions` **Variable length.** This list contains only non-zero bytes. Each byte must be a valid version, meaning that the most significant bit must be zero (0). At least one version must be present.

DPPv0 Version List must be sorted in increasing order. `dof-2008-1-spec-52`

Sorting the list makes it easier to remove matching responses and find matching versions.

This version can only be used as described above in discovery on lossy transports.

A query is represented as a single byte PDU with just the header; a response is formed by appending a list of version bytes after the header byte.

In order to optimize traffic, responses to multicast queries are sent using multicast, not unicast.

4.10. DOF Application Protocol

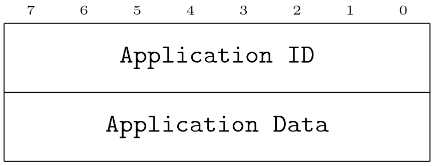
The goal of the DPS is to pass application protocols from one point to another. Each of these application protocols must be identified as it passes through the network. This identification is through an assigned Application Protocol Identifier (APPID).

APPIDs are assigned both for specific protocols, and also for different versions of the same protocol. Most application protocols are not described in this document. In that documentation will be the assigned APPID for the protocol.

The method of passing, or encapsulating, the APPID and associated application data is defined here and is standardized for the DPS.

The following shows the general format for APPIDs:

General Application Header `dof-2008-1-pdu-13`



`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Application Data` **Optional, variable length.** Contents are defined by the application.

The most common Application Identifiers are assigned in the range of 0-127, allowing them to be compressed to a single byte. Control protocols and protocols used mainly during setup should be assigned using numbers greater than 0x7F.

For a current list of registered versions, please contact the ODP-TSC or go to https://opendof.org/registry-dps-protocol.

Application versions are registered with ODP-TSC before use. `dof-2008-1-spec-53`

Application Identifiers must be registered with the ODP-TSC before use in products.

Each APPID is independent of the DNP and DPP used. This means that the same APPID can be used with many different DNP and DPP.

The range of APPIDs from 0x6000 to 0x6FFF is reserved for security modes of operation.

Security modes of operation are assigned APPIDs from 0x6000 to 0x6FFF. `dof-2008-1-spec-54`

This range is critical because security modes of operation may appear unsecured on secure sessions. Only this range of APPIDs is allowed this capability.

> **The use of any invalid APPID closes a lossless 2-node session.** `dof-2008-1-spec-55`
>
> DSP is used over all sessions to negotiate or discover application protocols. The use of any APPID (other than DSP) on a session before negotiation is complete is invalid. Note that DPP uses an internal APPID for itself, which is exempt from this requirement. Once established, the use of any non-negotiated or invalid APPID must result in the DPS session being closed.

As a special case, it is always allowed to pass an empty **Application Data** to an application. This is a 'no-op', or a way of passing information to the application without requesting that it perform any specific action. For example, an DPP cancel may not include a specific application PDU, but the application must still be notified of the lifecycle change for the given operation.

> **Empty application PDUs are always allowed and defined as a no-op.** `dof-2008-1-spec-56`
>
> Other stack communication may occur on these datagrams, for example to notify the application of lifecycle changes.

4.10.1. Peer-to-Peer Protocols and Client/Server Protocols

The DPS can be used for both peer-to-peer and client/server protocols. Each application protocol must specify whether it is client/server or peer-to-peer.

The following characteristics apply to peer-to-peer protocols:

- They can be negotiated by either (or both) nodes during DSP. If negotiated by either node, the protocol may be used equally by both nodes.
- They do not distinguish between the client and the server. It is permissible for the application to specify some behavior that is specific to the client or server, but any behavioral differences should be minor.

The following characteristics apply to client/server protocols:

- They must be negotiated by each node independently.
- The behavior and requirements of a node are dictated by its role – if both nodes negotiate the same protocol then they act in both roles individually and simultaneously.
- It is not necessary that the application 'client' be the same as the transport 'client'. In fact the transport 'client' may be the application 'server' depending on which node negotiates the protocol.

4.10.2. Context

The APP layer sits on top of the DPP layer.

For each received PDU the APP requires the following context information from the DPP, in addition to the context provided by the DPP, DNP, and Transport:

BUFFER — The received APP data and associated length.

COMMAND/ RESPONSE — Whether the PDU is an DPP command or response.

OPID — The operation identifier of the command or response.

DIRECTED/ FLOODED — Whether the operation is directed or flooded. This may not be known exactly depending on the specific DPP version, although if it is not known then FLOODED should be assumed.

LIFECYCLE — One of INITIAL, RETRY, NEWSOURCE, TIMEOUT, or CANCELLED.

SECURE — One of UNSECURE, AUTHENTICATED, or ENCRYPTED.

4.11. APP Version Discovery

As described above, there are problems attempting to discover all of the different versions of the different DPS layers that are in use in a network.

In the case of DNP and DPP version discovery the DOF Protocol Stack could not be fully utilized during the discovery process. APP version discovery is different, as it can use the full support of the lower layers (including framing, operations, etc.).

Because of this, APP version discovery is incorporated into the DOF Session Protocol (DSP), discussed later in this document.

4.12. DPS Intermediate Node Behavior

Intermediate nodes are defined as nodes which can forward operations to other nodes. They fall into two categories:

- Transparent intermediate nodes or bridges.
- Normal intermediate nodes that include application behavior.

A bridge is something that does not modify the application PDU in any way, and maps both the DPS versions and the transport in reversible and identical ways. For example, a node that bridges UDP to DLC can do so without modifying the application PDU. It can also map transport behavior (unicast, broadcast, multicast) to equivalent (or superset) behaviors on the other transport. The behavior of these nodes is fully defined by the requirements described in this paragraph – they do not need to deal with the full requirements of this section.

Another simple case is a bridge going between transports with different capabilities – like streaming vs. datagram. These nodes may not understand the operation behaviors of the application PDUs, but they may need to understand differences in the behavior of the DPS over different transports. For example, they may need to track operations so that appropriate 'Source Lost' behavior can be implemented. Obviously, as the number of situations that a node must deal with increases then the more required behaviors it must implement. Anyone writing an intermediate node should fully understand the requirements of this section and also the requirements of the application PDUs that it will work with.

Note, however, that a primary goal of the DPS is that fully (or very) capable intermediate nodes can be written that are agnostic to many application behaviors. Intermediate nodes must provide the common DPP behaviors and manage operation state.

4.13. DPS Sessions

The preceding sections have dealt with the specifics of how the DOF Protocol Stack uses layered protocols to transfer information from place to place. The description of sessions was introduced earlier in this document. With the understanding of the DPS discussed in this section, the following descriptions of DPS session establishment can be introduced.

Keep in mind the difference between a transport session and an DPS session. It is possible for a single transport session to encapsulate multiple DPS sessions, even if the transport session is lossless. DPS sessions also include another layer of logical addressing. However, they do share terminology with transport sessions.

In order to support the creation of sessions from sessions, the idea of a session server is introduced here. The definition of a transport server comes from the 'Transport' section of this document. Servers are not typically associated with sessions (other than they create them), although the DOF specification allows for server-based sessions using a security state identifier (SSID). Similar to this ability is that of using an existing session to create a new session, by treating one of the session endpoints (whether it was the original client or server doesn’t matter) to create a new session.

This is done by having the new session’s client (whether or not it was the transport client) pick a new DNP port (unique to the transport session) and using this port as the source of a packet to the existing session’s DNP address. This packet is understood by the receiver as belonging to a new session of type 'None', and that it should take the role of the server for this communication.

The new session begins at the DSP/Security Negotiation phase (described below), allowing for security and protocol negotiation to occur again on the new session.

The original transport session client maintains ownership of the transport session, which should remain established as long as there are any established sessions that use it. Once all such DPS sessions are closed the transport session may be closed.

4.13.1. Summary of Negotiation Phases

This section discusses the following negotiation phases, which are included here for reference in the descriptions of the session lifecycles.

DNP/DPP Negotiation

Described in this document, this negotiation only occurs on the establishment of a lossless transport session. This means that it does *not* get repeated when a lossless DPS session is created from an existing lossless session.

DSP Negotiation

Described in this document, this allows for negotiation of application protocols. It occurs for each lossless DPS session.

Opened

This point is defined as the transition between negotiating or discovering application protocols and using them. An unsecured DPS session moves from Opened to Established immediately because there is no Security Negotiation phase.

Security Negotiation

This phase uses application protocols (either discovered on a lossy session or negotiated on a lossless session) to establish a secure session, including shared security state.

Established

This is the transition point for a session at which it becomes established. It follows the negotiation phases discussed above. Secure sessions are secure once established. Note that there are many different ways in which an established session can be meaningless. For example, a session that fails to negotiate permissions for required behavior would be unusable. Sessions in this kind of dead-end situation should be closed with an error indication to the application.

Session Closure

This is the transition point from established to closed. A session that is never established may be terminated.

> **Sessions must authenticate (if required) and be established within 30 seconds.**
>
> **`dof-2008-1-spec-57`**
>
> This timer begins as soon as the DSP session is opened.

The use of these negotiated protocols, including periodic use of the authentication protocol as required, continues until the session is closed.

4.13.2. The 'None' Session

As discussed in the Transport section, a server may receive datagrams for which no session exists. In fact, all DPS sessions start in this state, even though the transport session exists. In a similar way, each new DPS session acts like it is in this state. However, there are timing restrictions on certain transport sessions that mean that the DPS sessions cannot remain in the 'None' state forever – they must actually establish a session.

Commands that do not require a session always execute in the 'None' session. They are never secure (as security requires a session to maintain state). Another way to refer to this state is 'stateless lossy' or 'stateless datagram'.

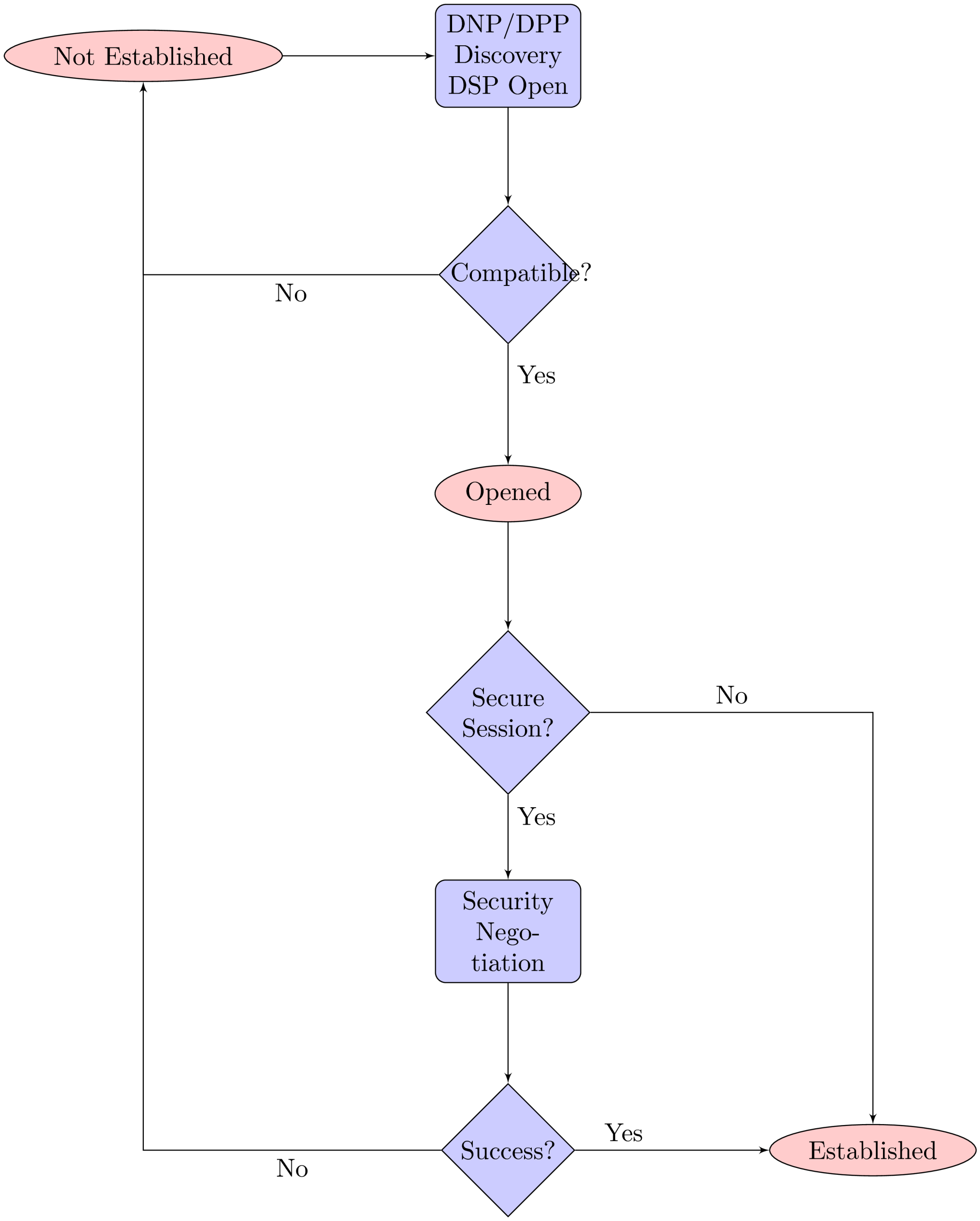
The 'None' session is characterized by the following characteristics:

- There is no negotiation, either of the DPS layer protocols or of application protocols. Discovery may be used.
- Flag bytes are either present, or use defaults. There is no session to maintain state.
- There is no shared state between endpoints, so each PDU must be self describing.
- There is no security.

4.13.3. Lossy 2-Node Sessions

There are two methods that can be used to create a lossy session: DSP Open, and Security Negotiation. Both of these methods follow the same general process:

Figure 4.10. Lossy Session Lifecycle



DSP Open is discussed earlier in this section, and creates unsecured DPS sessions between a lossy client and server. An DNP address is used on the server to identify the session. Note that an unsecure lossy session can be transitioned to a secure lossy session by using an authentication protocol. In this case it is the DPS addresses that define the session rather than a security state identifier.

Sessions can also be created through security negotiation. Authentication protocols that allow this behavior describe how it is accomplished. In this case the same server address is used, and the session is identified by using a security state identifier (SSID).

In the case of sessions opened through security negotiation there is a question of which application protocols may be used. These sessions do not negotiate, but rather use discovery to determine which protocols may be used. In general, the same rules as for negotiated sessions apply, meaning that incompatible protocols may not be used.

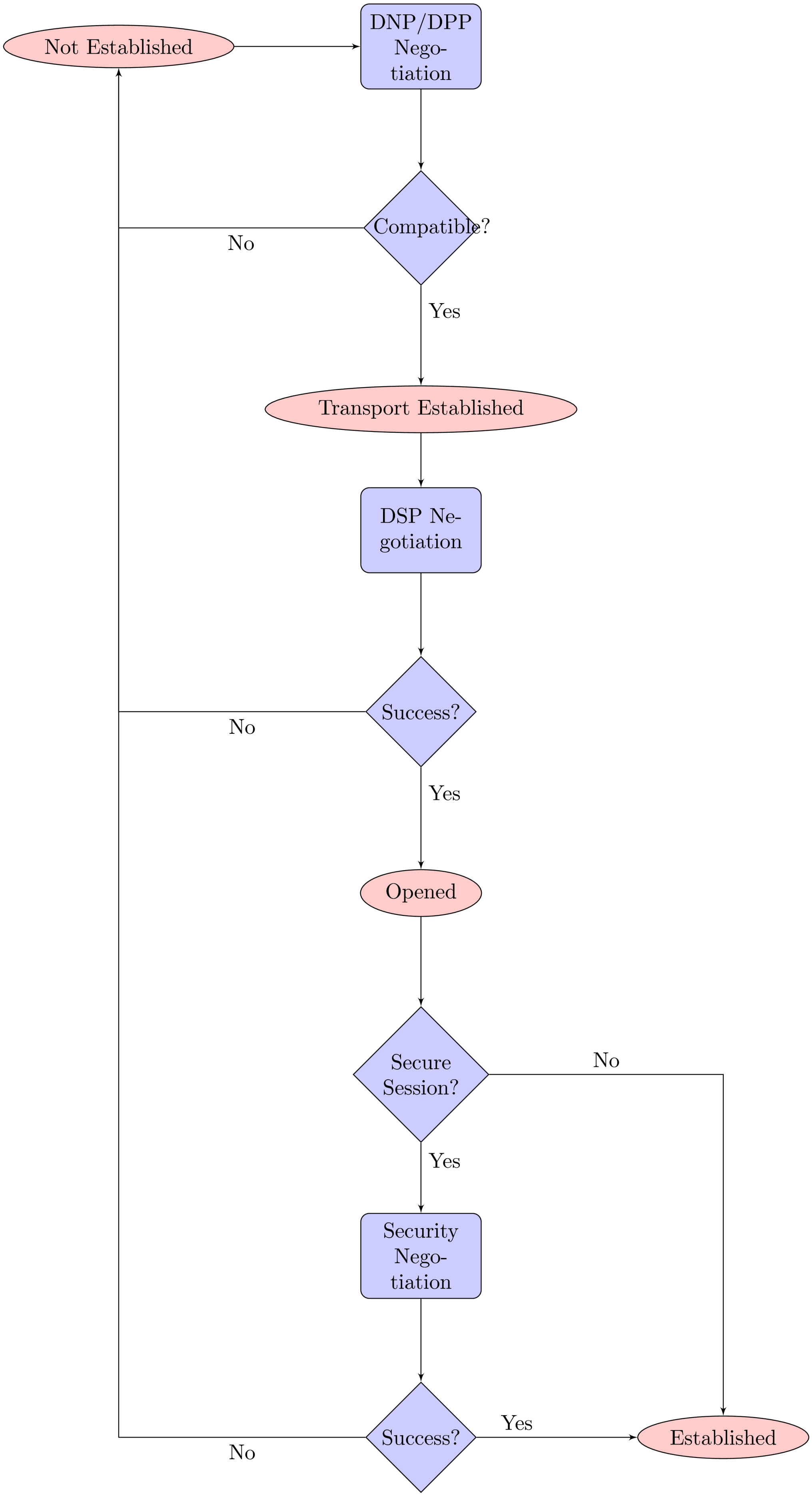
In all cases, these sessions are controlled by the client. Peer-to-peer protocols may not be used on the session until either the client sends an application datagram using that protocol or acknowledges to the server that the protocol is acceptable using an DSP Query Response. At that point the particular version of the protocol is determined and the server may respond or generate its own datagrams. The client my initiate a client/server datagram at any time, and its use should remain consistent. If the server desires to use a protocol then it should use the DSP Query to determine what is appropriate. This request may be made directly to the client node of the session. Note that the server already knows the application versions that it advertised in the DSP Open response, and by getting an DSP Query response knows the versions allowed by the client. There exists a potential race condition in the case where the client desires to use an old version of an application protocol. In this case it could respond to the server's Query including multiple versions, allowing the server to begin speaking with a version of its choice, and the client simultaneously begins using a different version. This is resolved through the intent of the client. If the client desires to use a particular application (including the version) then it should only include that version in its Query response. If it does not care, and in fact is not going to initiate the use of an application protocol, then it can allow the server to choose by returning multiple versions.

Note that it is possible to have lossy 2-node sessions that require security, or allow both, or only allow unsecure. The particular state is determined by the response to the DSP Open that begins the session. See the DSP Open command for more information.

Lifecycle management of lossy 2-node sessions is handled through the DSP or security negotiation protocol, depending on how the session was created.

4.13.4. Lossless 2-Node Sessions

When lossless sessions are involved, independent of the transport used, full negotiation is required for the initial transport session. The following diagram shows the states that a lossless session passes through before application data can be exchanged. Note that there are two places that sessions can originate, with only the first session after transport session establishment requiring DNP/DPP negotiation:


Not Established
DNP/DPP Negotiation
Compatible?
No
Yes
Transport Established
DSP Negotiation
Success?
No
Yes
Opened
Secure Session?
No
Yes
Security Negotiation
Success?
Yes
No
Established

The different protocols used in the phases shown are discussed in this document or in the different security documents.

Lifecycle management of lossless 2-node sessions is handled through the DSP.

4.13.5. N-Node Sessions

Multiple-node sessions are always created between servers and always assume lossy transports. The sense of an 'established' n-node session is also somewhat different than a 2-node session, as it is possible to have an established 1-node session (no other members).

N-node sessions are always established using a group-management security protocol. They may utilize version discovery, but they do not use any negotiation except for security negotiation (managed by the group-management security protocol).

Since n-node sessions always involve servers communicating with other servers, they require the use of a security state identifier (SSID). Further, since n-node sessions rely on transport capabilities for broadcast or multicast, the use of DNP ports is not possible.

Lifecycle management for N-node sessions is handled differently for multicast vs. unicast. Multicast N-node sessions have no lifecycle management for individual nodes, although the session itself has a lifecycle based on their being any members or not.

Unicast N-node sessions are a hybrid of lossy 2-node session characteristics and group characteristics. It is beneficial in these sessions to better manage node state (closer to lossy 2-node session capabilities). However, the protocols that manage these sessions do not have membership tracking commands. It is typical for lossy 2-node sessions to use traffic received in order to verify the activity of the nodes involved. The DPP Ping and Heartbeat are examples of commands that can be used. Upon joining a unicast N-node session, a node should unicast a Heartbeat command to the hub. This command may be lost (since these are lossy sessions), and so the hub must not rely on its reception. If the hub desires to know the state of a node then a Ping should be sent.

When nodes leave a unicast N-node session they should send an DSP Close/Terminate. The response to this command may be used as an indication that the packet was received. Note that while the session state associated with a node may be cleared using this technique, the security state must not be cleared. To clear the security state could allow replay attacks. Note that normal N-node security modes automatically clear unneeded state over time.

✓ **Nodes joining N-node unicast sessions should confirm by sending an DPP Heartbeat to the hub.** `dof-2008-1-spec-58`

The hub may also use an DPP Ping to verify the node has joined. The session should be considered established only after such a confirmation has been received, which could be any encrypted traffic or the response to an DPP Ping.

✓ **Nodes leaving N-node unicast sessions should send an DSP Close/Terminate.** `dof-2008-1-spec-59`

This may be repeated until a response is received, but should not continue forever. Security state must not be affected by this command, although session state may be cleared.

5. DOF Session Protocol (DSP)

The DOF Session Protocol (DSP) is an DOF application protocol. It is unique in that it is not negotiated, and is required to open an DPS session. DSP is always used on sessions that include only 2 nodes, and can be used in all cases for APP version discovery. DSP negotiation is required on lossless sessions, and the Close/Terminate PDU is also used on lossy sessions.

DSP is used to negotiate which application-level protocols may be used on a session, to configure any options for those application-level protocols, and (optionally) to terminate and close sessions. This protocol is based on PPP's Link Control Protocol in theory, although major changes have been made to the packet structure.

The DOF Protocol Stack section describes the general lifecycle of a connection. DSP is an integral part of this lifecycle, particularly during the beginning of the session.

Part of the negotiation involves requiring an authentication protocol or not. The signal as to whether to require authentication or not may come from the application.

There is no required negotiated closure of a session. As a convenience, a Close/Terminate command may be sent before closing a session, but the DPS does not require it.

5.1. Application Format

DSP uses an application identifier of 0x00 (0). The value that defines DSP is fixed; although the specific behavior of DSP can be controlled by the negotiated DOF Presentation Protocol. This means that negotiation of the DOF Presentation Protocol (discussed earlier) also negotiates the specific behavior and version of DSP that will be used on that link.

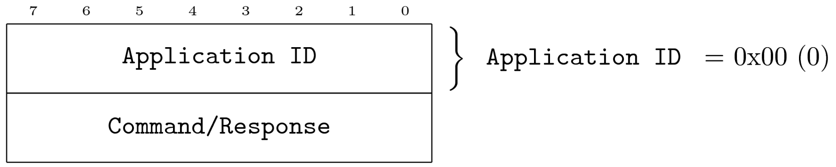
This version of DSP is associated with all DOF Presentation Protocols as of the time of publication.

DSP is a configuration protocol and should not necessarily be using a short APPID. However, APPID 0 is special. APPID 0 is reserved in DPS for application version query in DNP and DPP. DSP is similar, but has additional functionality.

When used in the DPS, the following format is used.

DSP Application Format `dof-2008-1-pdu-14`

Instance of General Application Header`dof-2008-1-pdu-13`.



`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Command/Response` **Instance of** DSP Command/Response Format`dof-2008-1-pdu-15`.

Each application PDU uses the following format.

⇄ **DSP Command/Response Format** `dof-2008-1-pdu-15`

`Opcode`
`Data`

`Opcode` **One byte.** The value indicates the particular operation.

`Data` **Optional, variable length.** Each `Opcode` will define the contents of this field.

5.2. DSP State Machine

The following state machine controls the DOF Session Protocol when used on lossless sessions. Note that there are many similarities between this state machine and the state machine of LCP as described in RFC 1661. The following assumptions are made for DSP that simplify the state machine, most of which relate to the fact that DSP is built on a connected transport.

- The PPP "Up" and "Open" events are tied together. "Up" would refer to the session being opened; "Open" would refer to the administrator running the server process (accepting sessions).
- There is no dedicated serial connection, making it impossible to have crossovers and resulting in the collapse of Closed, Stopped, Closing, and Stopping states.
- Once negotiated, configuration cannot be changed. Attempting to do so closes the session.
- There are no retries, as the lower layer is assumed to be error-free and guaranteed delivery.
- There are no defined timeouts, other than an overall DSP timeout defined later.
- The negotiation is serialized, and so each side goes through the same state transitions but in a different order.

The following table identifies the Events that drive the state machine:

Table 5.1. DSP Event Descriptions

Event	Description
Open	A session has been accepted at a lower layer.
Close	The session is requested to close either by software or an operator.
Down	The lower layer has closed the connection.
RCR+	An acceptable Configuration Request has been received.
RCR-	An unacceptable Configuration Request has been received.
RCA	A Configuration Acknowledge has been received.

Event	Description
RCN	Either a Configuration Negative Acknowledge or Configuration Reject has been received.
RTR	A Close/Terminate command has been received.
RTA	A Close/Terminate response has been received.
RUC	An unknown Opcode has been received.

The following table identifies the Actions that are performed by the state machine:

Table 5.2. DSP Action Descriptions

Action	Description
scr	Send Configuration Request if possible, otherwise send a Close/Terminate command.
str	Send a Close/Terminate command.
sta	Send a Close/Terminate response.
tlu	The layer is up, meaning that application protocols can communicate.
tld	This layer is down, meaning that application protocols may no longer communicate.
sca	Send Configuration Acknowledge.
scn	Send either a Configuration Negative Acknowledge or a Configuration Reject.

This is the state transition table. States are indicated by the column, with the event response on the row. Note that all transitions into state 0 (zero) from another state indicate that the session must be closed at the lower layer and the state machine can terminate. The same is true for the Close and Down actions while in state 0 (zero) – which is the initial state for the server. The implementation should attempt to make sure that any outstanding data (for example, a Close/Terminate response) is sent before the session is closed. This could be done using a timeout or any other method.

DSP is negotiated in both directions, but serialized. Negotiation is first done by the client, and assuming success the server then negotiates. This is indicated in the state machine by having separate initial states for client and server.

Table 5.3. DSP State Transitions

Event	0 (Initial)	4 Term-Sent	6 Request	8 Respond	9 Opened
Open (Client)	scr/6	-	-	-	-
Open (Server)	/8	-	-	-	-
Close	-	-	str/0	str/0	tld,str/4
Down	-	/0	/0	/0	tld/0
RCR+ (Client)	-	-	/0	sca,tlu/9	tld,str/4
RCR+ (Server)	-	-	/0	sca,scr/6	tld,str/4
RCR-	-	-	/0	scn/8	tld,str/4
RCA (Client)	-	-	/8	/0	-
RCA (Server)	-	-	tlu/9	/0	-

Event	0 (Initial)	4 Term-Sent	6 Request	8 Respond	9 Opened
RCN	-	-	scr/6	/0	-
RTR	-	sta/0	sta/0	sta/0	tld,sta/0
RTA	-	/0	/0	/0	/0
RUC	-	-	str/4	str/4	tld,str/4

These state transitions are only slightly related to RFC 1661, although much of the theory remains the same.

5.2.1. Timeouts

The DSP state machine initializes when the DPS completes DNP and DPP negotiation. DSP must not allow non-responsive sessions to indefinitely keep themselves open. This same requirement applies even when negotiation is not required (as in the case of lossy 2-node sessions).

DPS 2-node sessions must open within 30 seconds. `dof-2008-1-spec-60`

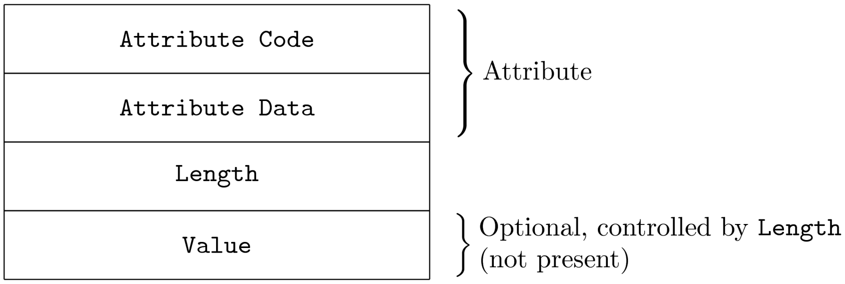
This period begins at the end of negotiation or when the DSP Open is received. It ends when the DPS session is open. Failure to open must result in the DPS session being terminated.

5.3. Configuration Options

The configuration option lists used in the Configuration Request and Configuration Negative Acknowledge PDUs follow the same format. Each Option list is formed from a list of Attribute-Value Pairs, each of which has an attribute (formed from a code and data) and a value (which may be empty, and includes a length).

Attributes are dealt with by DSP. Each attribute is formed by combining an **Attribute Code** with **Attribute Data**. Associated with each attribute is a value, formed from the **Value Length** and **Value Data**.

Attribute/Value Pair `dof-2008-1-pdu-16`



`Attribute Code` **One byte.** This field identifies the attribute. It must be registered with the ODP-TSC.

`Attribute Data` **Two bytes, MSB.** This field completes the definition of an attribute in combination with the code.

`Length` **One byte.** This field determines the length of the `Value`. It may be zero, in which case the `Value` field is not present.

`Value`	**Variable length. Optional, controlled by** `Length`. Contains the value for the associated attribute.

Attribute codes may be either single- or multiple-valued. A single-value attribute code means that the attribute code may appear a single time in a Configuration Request. A multiple-value attribute code means that the attribute code may appear multiple times in a Configuration Request. Note that it is the **Attribute Code** that is single-value or multiple-value. An attribute (code and data) may always appear a single time in any Configuration Request.

> An 'attribute' is always the combination of an **Attribute Code** and an **Attribute Data**.

Attributes are further differentiated by their value. Each attribute defines the format and restrictions placed on its associated data. Each data format unambiguously defines a set of behaviors associated with the negotiated attribute, which is understood by both the requestor and the acknowledger.

> For a current list of registered codes, please contact the ODP-TSC or go to https://opendof.org/registry-dsp-attribute.

> **DSP Attribute Codes are registered with ODP-TSC before use.** **`dof-2008-1-spec-61`**
>
> DSP Attribute Codes must be registered with the ODP-TSC before use in products.

The use of a specific attribute code has benefit during negotiation: if a particular attribute is not understood by the receiver, then the response will contain all of the appropriate choices for attribute data associated with the code. While this response can grow over time, it allows optimizing the next request based on the acceptable values.

This means that each specific attribute code should be used with related attribute data. For example, families of protocols or protocols that are used in a certain way (as will be seen with authentication protocol) are appropriate uses of the attribute code. However, there are only 255 possible attribute codes. Even so, the handling of each attribute code is uniform in the way it is negotiated.

5.4. Option Consistency

It is not a requirement that DSP itself understand the options that it negotiations – it is more of a negotiation framework. However, it is required that at the end of negotiation that the options are consistent (discussed later on).

This usually means that the application (or at least the application protocol implementations) must be involved in the negotiation process in some way, and in particular should approve the final accepted options.

5.5. APP Version Discovery

As described above in the section on the DOF Protocol Stack, there are problems attempting to discover all of the different versions of the different DPS layers that are in use in a network.

In the case of DNP and DPP version discovery the DOF Protocol Stack could not be fully utilized during the discovery process. APP version discovery is different, as it can use the full support of the lower layers (including framing, operations, etc.).

Because of this, APP version discovery is incorporated into this protocol.

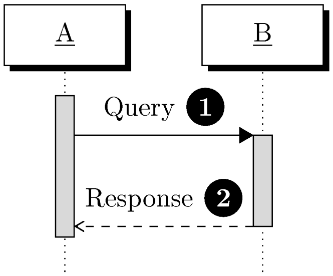
There are two PDUs defined for APPID version query over lossless transports. The first is a query, and second is a response, which is sent some time after the query, and which corresponds to a list of application identifiers supported by the sending node.

> **Unadvertised APPIDs should not be discoverable.** `dof-2008-1-spec-62`
>
> APPID zero (0) and 0x7FFF are not advertised. In the case of DNP and DPP there is no version 0 defined apart from the version query protocol itself. The application layer is different because there is an APPID zero (0) defined, DSP. APPID 0x7FFF is defined for common DPP behaviors and is not advertised.

Responses to multicast queries must be sent within three (3) minutes of a query, but the exact timing should be determined randomly and controlled by other network traffic as described below. Responses to unicast queries should be sent quickly.

Responses to unicast queries are sent unicast.

Figure 5.1. DSP Unicast Query



1. Query is sent from A to B.

2. Response is sent from B to A immediately.

In the example above, node Query wants to determine the APP versions supported by node A. It sends a unicast DSP Query PDU to A. Since the PDU is unicast, node A immediately responds without any use of timers or delay.

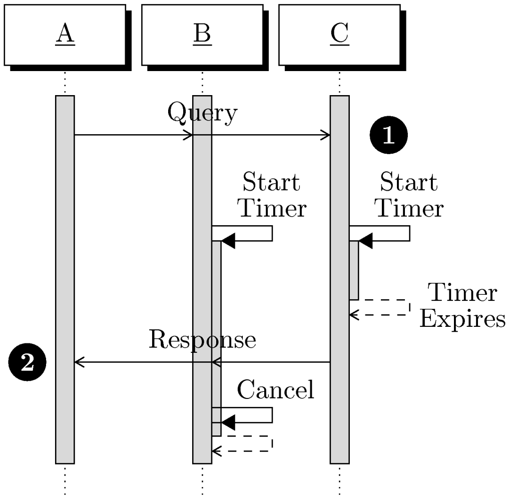
For multicast queries, between the time of the query and the time of the response three events can prevent sending a version in the query response (that consists of a set of versions):

- The node with a pending response sends a multicast PDU using the version on the same SESSION as the query.
- A multicast PDU is received by the node with a pending response using the version on the same SESSION as the query.
- A multicast query response is received from another node with a version listed on the same SESSION as the query.

Note that in these cases the SESSION includes the set of nodes that can communicate using multicast. The state and timers must be kept for each SESSION independently. For example, a node that uses multicast on multiple addresses would require state for each.

The goal of this logic is to only send information that cannot be determined by watching network traffic. The specific goal of version detection is *not* to discover nodes on the network, but rather to just discover versions of protocols that are in use.

Figure 5.2. DSP Multicast Query



1. Query is sent from A to all nodes (multicast).

   - B and C both start randomized timers.

   - Timer on node C expires.

2. Response is sent from C to all nodes (multicast).

   - Node A receives response.

   - Node B cancels timer, preventing response.

In the example above, a node Query initiates version discovery by multicasting an DSP Query PDU. This PDU is heard by nodes A and B. Both of these nodes pick random times within the next three minutes and start timers for when they will respond. The timer on B expires, and so B sends a multicast DSP Query Response. This PDU contains the same version as A, and so node A cancels its timer without sending a response.

If each version in the pending query response is covered by one of these cases then the query response is not sent. Otherwise at least the versions that have not been seen are sent.

For example, assume that a node N speaks version A and B of APP. It receives a multicast PDU that is an DSP Query. It determines (randomly) that it will send its response in 2 minutes. The response will be a PDU consisting of A and B.

After some time and before the two minutes pass a PDU is multicast from node N using APP version A. Sometime later, but still before the two minutes pass, a multicast PDU is received by node N using APP version B. In this case, the pending response PDU is cancelled.

Assume however that the second PDU is not received. The 2 minutes pass. In this case a response containing at least B is sent.

5.6. DSP Command and Response Definitions

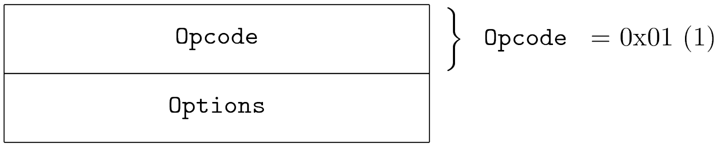
The following sections cover each command and response in the DOF Session Protocol. Note that DSP begins with the client sending a Configuration Request to the server. DSP then negotiates client options, followed by server options.

5.7. Configuration Request

Request a particular set of options on a session.
Session: Lossless
Addressing: Unicast
Unsecured: Required
Encryption: Not allowed
Message authentication: Not allowed
Permission (command): None
Permission (response): None

⇄ Configuration Request `dof-2008-1-pdu-17`

COMMAND/DIRECTED
NOOPID
Instance of DSP Command/Response Format`dof-2008-1-pdu-15`

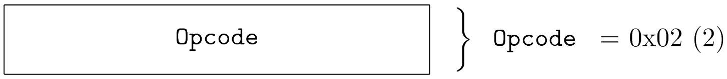


`Opcode` **One byte.**

`Options` **Array of** Attribute/Value Pair`dof-2008-1-pdu-16`.This field list contains the requested options.

⇄ Configuration Acknowledge `dof-2008-1-pdu-18`

RESPONSE
Instance of DSP Command/Response Format`dof-2008-1-pdu-15`



`Opcode` **One byte.**

Configuration Negative Acknowledge `dof-2008-1-pdu-19`

RESPONSE
Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$

`Opcode`	`Opcode` $= 0\text{x}03\ (3)$
`Options`	

`Opcode` **One byte.**

`Options` **Array of** Attribute/Value Pair$_{\texttt{dof-2008-1-pdu-16}}$.This field list contains the requested options.

Configuration Reject `dof-2008-1-pdu-20`

RESPONSE
Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$

`Opcode`	`Opcode` $= 0\text{x}04\ (4)$
`Attributes`	

`Opcode` **One byte.**

`Attributes` **Array of** Attribute Data List$_{\texttt{dof-2008-1-pdu-21}}$.This field list contains the rejected attributes.

Attribute Data List `dof-2008-1-pdu-21`

`Attribute Code`	
`Count`	
`Attribute Data`	Optional, controlled by `Count` (not present)

`Attribute Code` **One byte.** This field identifies the attribute. It must be registered with the ODP-TSC.

`Count` **Two bytes, MSB.** This field determines the length of the `Attribute Data`. It may be zero, in which case the `Attribute Data` field is not present.

`Attribute Data` **Two bytes, MSB.** This list contains Attribute Data values that are understood and are acceptable with the associated Attribute Code.

5.7.1. Small Implementation Hints

This command and the responses must be supported. A critical aspect for backward (and forward) compatibility is the ability to determine whether or not a requested configuration is acceptable. Incorrectly acknowledging a configuration without understanding it is a serious (and potentially fatal) error. Shortcuts should not be taken when validating a request or responding to a request.

It is possible that a small implementation has a single, hard-coded outbound Configuration Request command, and if it is rejected then the session will fail.

It is critical that an application correctly respond with acceptable configuration options so that the other node can make adjustments.

5.7.2. Flows

Configuration Request() → Configuration Acknowledge(): Success

Configuration Request() → Configuration Negative Acknowledge(): Retry requested

Configuration Request() → Configuration Reject(): Retry requested

5.7.3. Details

Once a transport-level lossless session is established between two nodes, each side must issue a Configuration Request (with the client going first, and the server continuing once the client negotiation is complete). This is done after the DOF Protocol Stack negotiation is completed. The configuration options field is filled with the current desired configuration of the sender.

Upon reception of a Configuration Request, an appropriate reply must be transmitted. The specific reply will be based on whether the request is not understood (Configuration Reject), not acceptable (Configuration Negative Acknowledge) or accepted (Configuration Acknowledge). A single Configuration Acknowledge is allowed from the server and client. Multiple Configuration Reject and Configuration Negative Acknowledge responses are allowed.

For each request and negative acknowledge, the **Options** field is variable in length, and contains the list of attribute/value pairs that the sender desires to negotiate or respond with. The list of attributes and values are responded to as a group with a single response, although the negotiation continues until a timeout or an acceptable request is made.

A client (meaning the node that requests the session) must send a Configuration Request as soon as possible. A server must wait until it has acknowledged the client configuration before negotiating its own options. In this case the server is not required to negotiate any options, in which case an empty request is sent. This may be done, for example, if the clients request includes everything that the server desires. The server may request additional options. However, the server must not change any options requested by the client, and the client must accept any options requested by the server that the client negotiated.

If every attribute received in a request is recognizable and all values are acceptable, then the implementation must transmit a Configuration Acknowledge. This indicates that the sending node is prepared to begin using the protocols and protocol options specified.

✓ **DSP must result in a consistent set of negotiated options.** `dof-2008-1-spec-63`

This requirement is met by having the client negotiate first. Once the server has acknowledged the client options they become fixed. The server may then negotiate additional, non-conflicting options. Any attempt by the server to negotiate a conflicting option, or a rejection by the client of an option that is already fixed, must result in the session being closed. This applies in particular to peer-to-peer options, client/server options may be negotiated in each direction differently. DSP uses the application to determine the specific requirements of consistency.

✓ **DSP must result in an unambiguous set of options.** `dof-2008-1-spec-64`

Ambiguity in this case refers to the application understanding of the final set of options. For example, if a session that requires security fails to negotiate options that allow for key distribution, then the means for obtaining security are ambiguous. The same would be true if options contradict each other. This is possible even if the options themselves (compared against each other) are consistent. For example, negotiating multiple authentication protocols for a secure session would be ambiguous, as the two options contradict each other (each option says to use a particular protocol to authenticate, but only a single authentication protocol can be used).

If every instance of the received attributes is recognizable, but some values are not acceptable, then the implementation must transmit either a Configuration Negative Acknowledge or (optionally) a Configuration Reject. In the case of a negative acknowledge, the **Options** field is filled with information that is based on the unacceptable attributes from the request. All attributes with values that are acceptable are filtered out of the Configuration Negative Acknowledge, but otherwise the attributes from the request must not be reordered.

For each requested attribute/value pair in the request, the negative acknowledge either removes it (if acceptable), or replaces it with a list of attributes that enumerate acceptable values for the attribute. It is possible that the attribute data format can encapsulate these options in a single block, which is acceptable as long as the negotiated meaning is unambiguous (in other words, both the sender and receiver understand the single set of options that are negotiated). The response list associated with a given attribute may contain a single entry, or several.

Finally, after the requested attributes have been responded to as described, the sender may include additional attribute/value pairs at the end of the negative response. These attributes act as 'hints' to the other node to include these options in its next request. Again, these lists may include multiple choices for a given attribute, although the requestor must pick a single acceptable attribute of each to include in its next request. If a suitable choice for each 'hinted' attribute is not appropriate for the sender then it may attempt to continue negotiation without including the attribute. However, if no other changes are made to the request then it is impossible for the response to be accepted. It is also possible that the different attributes listed are not allowed in the same request. In this case the requestor may choose the appropriate options that do not conflict.

For example, a server that requires authentication may understand different options using different attributes. A single negative acknowledge may list several options. However, the request is not allowed to request multiple authentication protocols in the same request and so the requestor must choose one of the suggested options.

If some of the received attributes in a request are not recognizable or are not acceptable for negotiation (as configured by the application), then the implementation must transmit a Configuration Reject. The response includes a list of each **Attribute Code** field that is not acceptable, along with a list of the associated **Attribute Data** fields that are acceptable to the sender of the reject.

Optionally, and similarly to the Configuration Negative Acknowledge PDU, the Configuration Reject PDU may contain attributes that were not present in the corresponding request. In this case the reject sender is suggesting to the other node that additional options (using those codes) should be included in the next request.

For example, a server may desire a session to be secure. There are two possible responses to a request that does not include the appropriate options:

- Use Configuration Negative Acknowledge. In this case the server must provide specific options (including data). This is a very specific way of suggesting that an option be included.
- Use Configuration Reject. In this case the server need only include the attribute, without needing to specify the attribute data. This gives the client more flexibility in how to form the next request.

Reception of a valid Configuration Reject indicates that a new Configuration Request should be sent that should only include the **Attribute Code** and **Attribute Data** that have not been rejected.

Reception of a valid Configuration Negative Acknowledge indicates that a new Configuration Request should be sent with the **Options** modified to include data that was included in the negative acknowledge.

5.7.4. Retry Behavior

This command is used only on lossless sessions, and retries are not allowed.

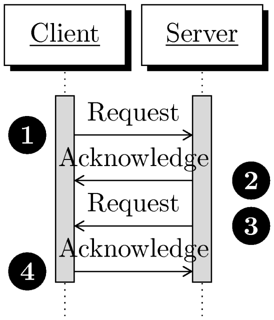
5.7.5. Routing Rules

DSP must not be routed.

5.7.6. Sequence Diagrams

This first sequence shows the easiest negotiation possible.

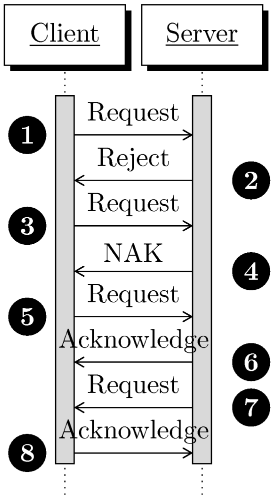
Figure 5.3. Basic DSP Negotiation



1. Request is sent from the client.
2. Request is evaluated and accepted, acknowledge sent from the server.
3. Request is sent from the server.
4. Request is evaluated and accepted, acknowledge sent from the client.

In this sequence the client requests a configuration. The server acknowledges, and then begins its request, which is acknowledged.

Figure 5.4. Basic DSP Negotiation



1. Request is sent from the client.
2. Request is evaluated and either not understood or unacceptable, reject sent from the server.
3. A modified request is sent from the client.
4. Request is evaluated and understood, but is unacceptable, negative acknowledge sent from the server.
5. A further modified request is sent from the client.
6. Request is evaluated and accepted, acknowledge sent from the server.
7. Request is sent from the server.
8. Request is evaluated and accepted, acknowledge sent from the client.

In this more complicated example only a single negotiation is shown. In step 1 the client issues a configuration request for authentication protocol 1, application protocol 2 with options a and b, and application protocol 4.

In step 2 the server rejects authentication protocol 1 and application protocol 4. The client drops its request for authentication because it doesn’t know any alternate authentication protocols. It also must decide a fallback for protocol 4, and it chooses APPID 3.

The changes to its request can be seen in the request in step 3. The request for application protocol 2 has not changed, but application protocol 3 has replaced the request for 4 and the authentication protocol request is gone.

The server now issues a negative acknowledge in step 4. The response contains a list of all options that would be acceptable. In this case the requested options a and b are not allowed. Again, the client must determine an appropriate fallback. In this case the choices allowed are presented.

In step 5 the client makes its third request. It has chosen application protocol 2 with options a and c (which was a choice in the negative acknowledge).

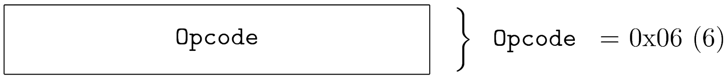
In step 6 the server acknowledges the choices. The two protocols with the selected options will be used.

5.8. Open

Open a stateful session on a lossy transport.
Session: Lossless
Addressing: Unicast
Unsecured: Required
Encryption: Not allowed
Message authentication: Not allowed
Permission (command): None
Permission (response): None

Open (command) `dof-2008-1-pdu-22`

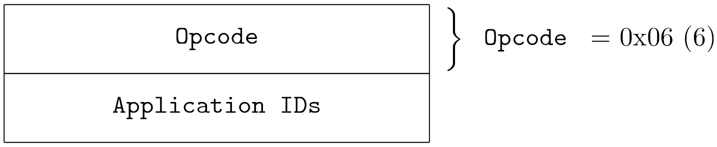
COMMAND
DURATION optional
DIRECTED
OPID
Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$



`Opcode` **One byte.**

Open (response) `dof-2008-1-pdu-23`

RESPONSE
Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$



`Opcode` **One byte.**

`Application IDs` **Array of** Compressed Unsigned 16-Bit$_{\texttt{dof-2009-1-pdu-1}}$.

Secure Open (response) `dof-2008-1-pdu-24`

RESPONSE

Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$

`Opcode`
`Application IDs`

`Opcode` $= 0\mathrm{x}07\ (7)$

`Opcode`	**One byte.**
`Application IDs`	**Array of** Compressed Unsigned 16-Bit$_{\texttt{dof-2009-1-pdu-1}}$.

Open response (both cases) Version List must be sorted in increasing order.

`dof-2008-1-spec-65`

Sorting the list makes it easier to remove matching responses and find matching versions.

This command is used to request the creation of a new session on a lossy transport and also to discover the set of protocols that are acceptable outside of a session. These two cases are distinguished by the duration of the operation as discussed below. In all cases the command is sent unicast and must include an DPP operation identifier.

The first, and easiest, case is when the duration is zero. In this case no session is established, but a response is sent with an **APPID List** that contains the application protocols that are acceptable by the node outside of a session, or an empty list if there is a failure. Note that this response is different from that of Query, because a Query outside of a session responds with *all* protocols understood, whether inside or outside a session.

Note that a non-error response should be sent when the duration is zero even if sessions are not normally supported – in other words even if a failure would have been returned if the duration were greater than zero. This allows the use of this packet to discover application protocols that can be spoken without having an active session.

The second case applies when the duration is greater than zero. In this case, receivers of this command should begin the establishment of a session with the sender. Each session request is associated with the request operation identifier, which is used to uniquely identify each request and prevent aliasing and retransmission problems. The duration of the operation controls how long the initial handshake of creating the session can take. The handshake follows this procedure:

- The client sends an Open, with an operation identifier and duration. This Open establishes the DNP/DPP versions that will be used on the session. It also begins the timeout period for the DPS session to be open, limiting the duration available in this command.

- The server assigns a logical address for the session. This must use the same transport address and a new DNP port. The Open response is sent, unicast, and using the same operation identifier. If the server cannot create the session, then a response is sent unicast from the original target address (no new DNP port is used), and with an empty **APPID List**. The empty **APPID List** indicates the failure to the client. The **APPID List** indicates the application protocols that are acceptable by the node on the new session.

- The client sends a datagram to the server using the new logical address. The receipt of this datagram on the server completes the opening of the session. The choice of this packet should be guided by the security discussion (below) which places certain requirements on the first application protocol command that is sent.

Until the session is opened the client may use DPP to cancel the request. This terminates the session.

When opened, the session may need to proceed through security negotiation. Secure sessions are not established until this is completed. Determining whether the server requires security depends on the **Opcode** used in the response. Only in the case of a server that requires a secure session sending a successful response is the value 0x07 (7) used for **Opcode**, and only in the response PDU.

If the server allows, but does not require, security the same problem of determining whether the session will be secure exists and must be determined by the client. The client declares its security desire through the choice of the PDU that it uses to open (and possibly establish) the session (the last step described above). There are three cases:

- The client sends and DNP, DPP, DPP Common, or DSP packet. These serve to open the session but do not determine the security desire, delaying the determination until a later packet.
- The client sends a packet from an authentication protocol (including a no-op). This indicates that the session will be secure. Authentication protocols are identified as part of their protocol registration, and can be found with their registration on https://opendof.org/registry-dps-protocol.
- The client sends a packet from a non-authentication protocol (including a no-op). This indicates that the session will be unsecured.

The packet received indicates to the server whether the session is established (non-secure) or opened and enters security negotiation (secure). Note that a session that enters security negotiation is not established until security negotiation successfully completes.

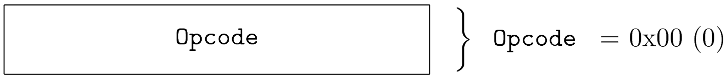
Once the session is established it may be maintained using periodic datagram exchanges (DPP Heartbeat, DPP Ping). The session is closed either through timeout, or the Close/Terminate command can be used.

5.9. Query

Request all Application IDs on a network, or those acceptable on a session.
Session: Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Unsecured: Allowed
Encryption: Allowed
Message authentication: Allowed
Permission (command): None
Permission (response): None

Query (command) `dof-2008-1-pdu-25`

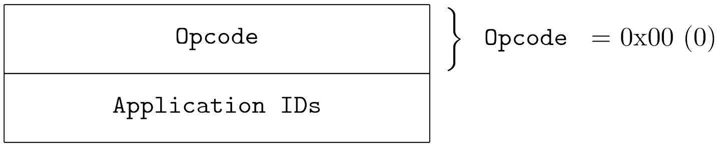
COMMAND
DIRECTED
NOOPID
Instance of DSP Command/Response Format`dof-2008-1-pdu-15`



`Opcode` **One byte.**

Query (response) **`dof-2008-1-pdu-26`**

RESPONSE
Instance of DSP Command/Response Format`dof-2008-1-pdu-15`


Opcode
Application IDs

Opcode $= 0x00\ (0)$

`Opcode` **One byte.**

`Application IDs` **Array of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.

Query response Version List must be sorted in increasing order. **`dof-2008-1-spec-66`**

Sorting the list makes it easier to remove matching responses and find matching versions.

In order to optimize traffic, responses to multicast queries are sent using multicast, not unicast.

Queries made in the 'none' session always result in a response that includes all protocol versions understood by the sending node. Multicast Queries further follow the rules described earlier for APP version discovery.

Queries made in a session result in response that includes only those protocol versions that are acceptable at the time of the request. For example, after creation of a lossy 2-point session using the Open command the acceptable protocols may change. This can occur if the session must be secured. Once secured, using the Query command on the session will return the protocols that are acceptable on the secured session.

This behavior is also true (secured and unsecured) when this command is used by the server in a lossy 2-point session. In this case the client will respond with the acceptable protocols on that session. This command may be used in secure n-point sessions. In this case the handling is as in the unsecure none-session, but scoped to only nodes in the secure session.

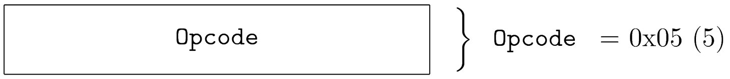
5.10. Close/Terminate

Request that a session be closed or terminated.
Session: Lossy (2-node), Lossless
Addressing: Unicast
Unsecured: Allowed
Encryption: Allowed
Message authentication: Allowed
Permission (command): None
Permission (response): None

Close/Terminate (command) **`dof-2008-1-pdu-27`**

COMMAND

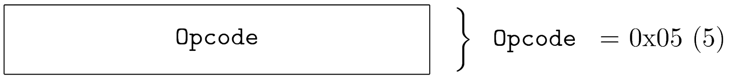
DIRECTED
NOOPID
Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$



`Opcode` **One byte.**

⇄ **Close/Terminate (response)** **`dof-2008-1-pdu-28`**

RESPONSE
Instance of DSP Command/Response Format$_{\texttt{dof-2008-1-pdu-15}}$.



`Opcode` **One byte.**

5.10.1. Small Implementation Hints

This command is completely optional.

5.10.2. Flows

Close/Terminate() $\rightarrow$ response(): Success

5.10.3. Details

DSP includes a Close/Terminate command in order to provide a mechanism for the orderly closing of an established session or the termination of an open session. The security used on the command and response must match that of the established secure session that it is used on.

An implementation wishing to close a session should transmit a Close/Terminate command, and the receiver should respond with a Close/Terminate response.

Upon reception or sending of a Close/Terminate response the session should be closed. This logic allows for previously sent packets to be 'flushed' before the session is terminated. It is typical that the side wishing to close the session has recently sent packets indicating, for example, some error condition. If those packets were sent, followed by a Close/Terminate command, and then *immediately* the session were closed, there would be a chance that none of the packets will actually be sent before the session is closed. In this case the other side of the session is left without the error information that would inform it of the reason that the session was closed.

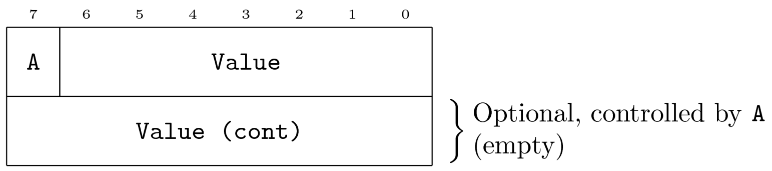
Waiting for a Close/Terminate response allows for more certainty that all previously sent packets have actually been received by the other side of the session.

The implementation should not wait indefinitely for the Close/Terminate response, but there is not a specified timeout.

6. External PDU Reference

6.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018)

Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`A` **One bit.** Controls the size of `Value`.

`Value` **Either seven or fifteen bits, MSB, controlled by A.** If the `A` field is zero then the `Value` is contained in a single byte. If the `A` field is one then the `Value` is passed in fifteen bits with the least significant bits passed in a second byte.