



IPDR/Streaming Protocol Specification

Version 2.0

May 12, 2004

© 2004 IPDR, Inc.

Preface

Contacts

For general questions regarding this document and referrals to technical experts for detailed questions, please contact:

IPDR.org Editor-in-Chief:
Steve Cotton
scotton@ipdr.org

Abstract

This document specifies the IPDR Streaming protocol and includes procedures, encodings, and message sets to accomplish this objective.

Change History

Version	Date	Name	Description
Initial Draft	11/11/03	Working Group Team	Initial Draft after harmonization
Review Draft 1	11/15/03	Working Group Team	Consolidation of Several Contributions
Review Draft 2	11/18/03	Working Group Team	Correction of Section Numbering
Review Draft 3	11/22/03	Working Group Team	Addition of Contributions
RD3.1	11/24/03	Working Group Team	Addition of Contributions
RD4	11/25/03	Working Group Team	Addition of Contributions
RD5	12/11/03	Cotton, Givoly, Meyer	Review of RD4, Action Items from 12/10/03 Conference Call
RD6	12/17/03	Weber	Resolution of GW Comments
Ballot Draft	1/2/04	Protocol Working Group	Sent to General Membership for Comment Prior to Steering Committee Approval
2.0	5/4/04	Protocol Group	Resolve Outstanding Issues Prior to Issue

Acknowledgements

Ty Roach, ACE*COMM
Greg Weber, Cisco Systems
Jeff Meyer, Hewlett Packard
Tal Givoly, XACCT Technologies
Steve Cotton, Cotton Management Consulting

Table of Contents

PREFACE	2
Contacts	2
Abstract	2
Change History	2
Acknowledgements	2
1 INTRODUCTION	5
1.1 Specification of Requirements and Notation	5
1.2 Terminology	7
1.3 Problem Statement	9
2 PROTOCOL OVERVIEW	11
2.1 Architecture	11
2.2 Data Types	12
2.2.1 Basic Types	13
2.2.2 Structures	13
2.3 Data Representation	13
2.4 Templates	14
2.4.1 Template Representation	15
2.4.2 Template Transmission and Negotiation	15
2.4.3 Changing Templates	16
2.5 Flow Control	17
2.6 Multiplexed Streams	19
2.7 Reliability	20
2.8 State Machines	21
2.9 Exporter Query Messages	23
2.10 Sessions	23
2.11 Backwards Compatibility	24
2.12 Connection Establishment	24
3 MESSAGE FORMAT	26

3.1	XDR equivalence to “ASCII art”	26
3.2	Common Header	27
4	MESSAGE DETAILS	29
4.1	Connection	29
4.1.1	Connect	29
4.1.2	ConnectResponse	30
4.1.3	Disconnect	31
4.2	Errors	31
4.2.1	Error	31
4.3	Flow Control	31
4.3.1	FlowStart	32
4.3.2	SessionStart	32
4.3.3	FlowStop	32
4.3.4	SessionStop	32
4.4	Template	33
4.4.1	TemplateData	33
4.4.2	ModifyTemplate	33
4.4.3	ModifyTemplateResponse	34
4.4.4	FinishNegotiation	34
4.5	Data	34
4.5.1	Data	34
4.5.2	DataAcknowledge	35
4.6	State Independent	35
4.6.1	GetSessions	35
4.6.2	GetSessionsResponse	36
4.6.3	GetTemplates	36
4.6.4	GetTemplatesResponse	36
4.6.5	KeepAlive	36
5	UNDERLYING TRANSPORT	38
6	SERVICE DISCOVERY	39
6.1	UDP Protocol	39
6.2	Capability Files	39
7	SECURITY	41
8	COMPLETE IPDR/STREAMING IDL DEFINITION	42

1 Introduction

Service Elements are often required to export usage information to mediation and business support systems (BSS) to facilitate accounting. Though there are several existing mechanisms for usage information export, they are becoming inadequate to support the evolving business requirements from service providers.

For example, some of the export mechanisms are legacies of the Telco world. Typically usage information is stored in Service Elements as Log files (e.g., CDR files), and exported to external systems in batches. These are reliable methods; however, they do not meet the real-time and high-performance requirements of today's rapidly evolving data networks.

RADIUS is a widely deployed protocol that may be used for exporting usage information. However, it can only handle a few outstanding requests and has extensibility challenges due to its limited command and attribute address space and complexity of its VSAs. A detailed analysis of limitations of RADIUS can be found in .

DIAMETER is a new AAA protocol that retains the basic RADIUS model, and eliminates several drawbacks in RADIUS. The current DIAMETER protocol and its extensions focus on Internet and wireless network access, and their support of accounting is closely associated with authentication/authorization events. DIAMETER is intended to solve many problems in the AAA area; by doing so, it does not adequately address some critical issues such as efficiency and performance in an accounting protocol.

There are also SNMP based mechanisms that generally require a large amount of processing and bandwidth resources.

Based on more detailed analysis along the lines mentioned above, the need for a reliable, fast, efficient and flexible accounting protocol exists. The IPDR Streaming Protocol is designed to address these critical requirements.

This document defines the Streaming Protocol that enables efficient and reliable delivery of any data, mainly Data Records from Service Elements to any systems, such as mediation systems and BSS/OSS. The protocol is developed to address the critical needs for exporting high volume of Data Records from the Service Element with efficient use of network, storage, and processing resources.

This document specifies the architecture of the protocol and the message format, which must be supported by all Streaming Protocol implementations.

1.1 Specification of Requirements and Notation

In this document, the keywords "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" are to be interpreted as described in BCP 14 . These keywords are not case sensitive in this document.

This document will use initial cap spelling for all keywords which are referenced. For instance the phrase Streaming Protocol is capitalized, to indicate that a concise definition of the meaning of this term is available in the terminology section.

The document will also utilize XDR-based IDL annotation when describing the structure of protocol messages.

1.2 Terminology

Collection System

A system composed of several Collectors with the objective of receiving, reliably, a non-duplicate set of events, perhaps through the use of redundant Collectors and mechanisms to de-duplicate data.

Collector Priority

A Collector is assigned a Priority value. Data Records should be delivered to the Collector in the flow ready state with the highest Priority value (the primary collector) within a Session.

Collector

A Collector is an implementation on the data receiving side of the Streaming Protocol. It is typically part of a Business Support System (BSS) (e.g., Billing, Market Analysis, Fraud detection, etc.), or a mediation system. There could be more than one collector connected to one exporter to improve robustness of the usage information export system.

Data Record

A Data Record is a collection of information gathered by the Service Element for various purposes, e.g., accounting. The structure of a Data Record is defined by a Template, and contains Fields.

Element

The formal declaration of a Field in an IPDR Information Model or Service Definition. Borrowed from the Element term in XML-Schema.

Exporter

An exporter is an implementation on the data producing side of the Streaming Protocol. It is typically integrated with the Service Element's software, enabling it to collect and send out Data Records to an interested consumer system using the protocol defined herein.

Field

A constituent of a Data Record. Formal term is Field instead of attribute or Element.

Identifier or ID

A means of referring to a specific instance of such items as a Field or a Type and distinguish among them. We also have "Field ID" and "Type ID" as a result.

Information Model

A descriptive tool used to capture the set of managed information objects at a

conceptual level, independent of any specific implementations or protocols used to transport the information.

Message

A Message is encoded according to rules specified by the Streaming Protocol and transmitted across the interface between an Exporter and Collector. It contains a common Streaming Protocol header and optionally control or user data payload.

Name

A means to refer to a specific Element and distinguish among them.

Record Sequence Number

An Data Record level sequence number, which is attached to all data messages to facilitate reliable and in-sequence delivery.

Service Definition

An XML Schema representation of an Information Model which conforms to the IPDR.org Service Specification Guidelines .

Service Element

The logical entity in the IPDR Reference Model which senses usage of services by the Service Consumer and exports associated usage information to the IPDR Recorder. Physically, a Service Element can be any component of functionality from the IPDR High Level Model Network and Service Element (NSE) layer. These may include various components traditionally considered infrastructure network elements.

Session

A Session is a logical connection between an Exporter and one or multiple Collectors for the purpose of delivering Data Records. Multiple sessions may be maintained concurrently in an Exporter or Collector; they are distinguished by Session IDs.

Streaming Protocol

The Streaming Protocol maybe referred as Streaming, or the Protocol in this document. The Streaming Protocol is used at the interface(s) between an Exporter and one or multiple Collectors for the purpose of delivering Data Records.

Template

Specification of the layout of Fields within a Data Record. A Template defines the structure of any types of Data Record, and specifies the data Type, meaning, and location of the Fields in the record.

Type

A constraint on the value and format of a field, e.g. dateTime

1.3 Problem Statement

Industry members, as well as the IPDR.org members themselves, perceive the main issue addressed by IPDR.org as the standardization of a billing-grade protocol to replace the variety of protocols currently used to export usage data from network and service elements. Every day that goes by without a streaming protocol of such nature in the arsenal of protocols turns people away from IPDR.org to solve their needs.

Target applications for this Streaming protocol would be Service Elements such as:

- Soft switches / VoIP Gateways
- Web/Proxy servers
- Application servers
- Firewalls
- Content Delivery Networks (CDN)
- Streaming media servers
- Game servers
- Passive/active Probes
- Edge access devices (routers, CSU/DSU, CMs, etc).
- Location-based wireless services

What would it take to make such a protocol gain adoption rapidly?

- It would have to be a final version of a protocol ratified by IPDR.org
- It would have to have a production-grade implementation (not merely reference implementations)
- The production-grade implementation would have to be embeddable in a variety of network/service element platforms with relative ease (low risk and fast implementation time).
- It should be simple enough so people could implement a compliant version on their own
- Endorsed by 1-2 leading network equipment manufacturers
- The protocol must address a set of requirements (informally specified below):
 - Reliable data delivery with intrinsic fault-tolerance, enables auditing and duplicate elimination
 - Flexible - supports arbitrary concurrent export of data structures
 - Efficient implementation possible:
 - o Network link efficiently utilized (overhead/payload and compact payload)
 - o Memory footprint efficient (buffers could be small, copies could be prevented)
 - o Avoids unnecessary copies and conversions in network/service element
 - Scalable:
 - o Can handle on the order of 30,000 records/second/processor (assuming 1Ghz processor).
 - o Can operate across a disruptive WAN links

- Securable to avoid tampering and eavesdropping
- Streaming real-time – IPDR already has a document oriented IPDR/File Transfer protocol, what’s missing is a Streaming Protocol.

2 Protocol Overview

2.1 Architecture

The IPDR/Streaming Protocol is an application running over a reliable transport layer protocol. The transport layer protocol is responsible for delivering IPDR/Streaming messages between Exporters and Collectors. The transport layer must support the following capabilities:

1. Reliable, in-sequence message delivery.
2. Connection oriented.

The transport layer may support:

1. Authentication.
2. Bundling of multiple messages into a single datagram.

Possible transport layer protocols may be TCP or SCTP . TCP supports the minimal requirements for IPDR/Streaming, but lacks some desirable capabilities that are available in SCTP, these include:

1. Session level authentication.
2. Message based data delivery (as opposed to stream based).
3. Fast connection failure detection.

Another possible candidate transport protocol is BEEP (RFC3080) . BEEP sits between low level transport protocols such as TCP or SCTP and higher level application protocols such as HTTP.

Reliable delivery of Data Records is achieved through both the transport layer level and the IPDR/Streaming Protocol level. The transport layer acknowledgements are used to ensure reliable delivery of data packets and detection of lost exporters, the IPDR/Streaming Protocol acknowledges IPDR/Streaming Messages after they have been processed and the accounting information has been placed in persistent storage.

Being a reliable protocol for delivering Data Records, traffic flowing from an Exporter to a Collector is mostly Data Records. There are also bi-directional control message exchanges, though they only comprise a small portion of the traffic.

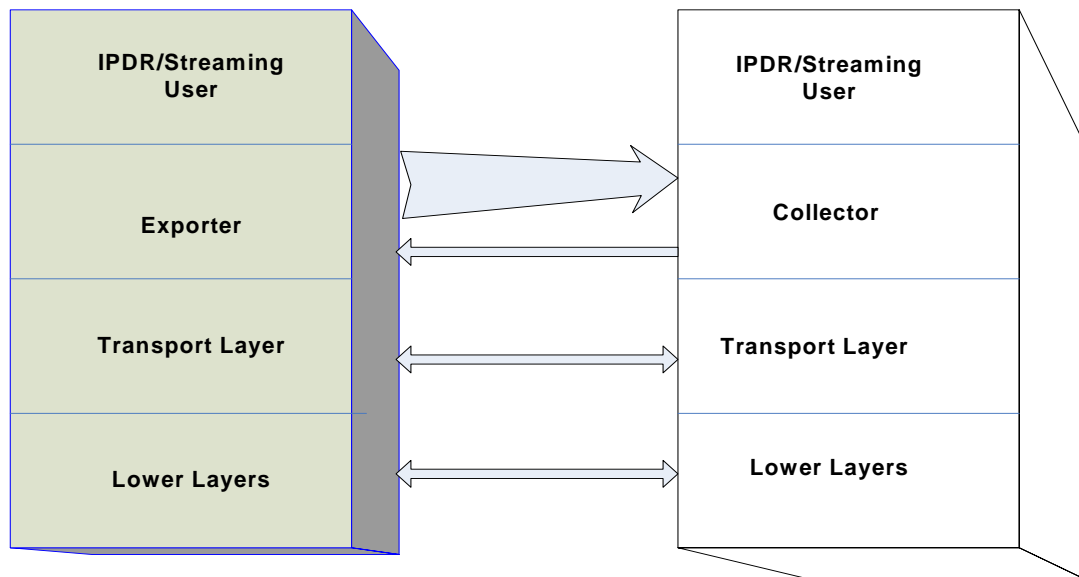
Streams of Data Records from a given Exporter have a large amount of internal consistency. That is, the accounting events all represent instances of a small set of accounting record types. IPDR/Streaming, like the IPDR/XDR format utilizes the concept of Templates in order to eliminate the transmission of redundant information such as field identifiers and typing information on a per accounting record basis.

IPDR/Streaming also incorporates IPDR/Service Definitions, based on XML-Schema, by reference. IPDR/Service Definitions describe in detail the properties of the various accounting records and their fields. IPDR/Streaming Templates, like IPDR/XDR Templates identify the URI of the schema which describes a given accounting record structure and identifies each field using its qualified name, as defined in the referenced XML-Schema or in subordinate imported schemas.

An IPDR/Streaming Exporter may be associated with multiple Collectors. For any given stream of accounting records (termed a “session”), a single active Collector will be targeted with those Data Records. However in the event of a detected failure, if an alternate Collector is available, that stream will be redirected to the alternate Collector.

If an IPDR/Streaming Exporter is managing multiple Sessions these may designate different Collectors as their primary consumer.

The figure below summarizes the relationship:



2.2 Data Types

A Session may exist between an Exporter and one or more Collectors. The Exporter is responsible for configuring network addresses of all Collectors belonging to the Session.

The means of configuration is outside the scope of this document. A Collector Priority is assigned to each Collector. The Collector Priority reflects the Exporter's preference regarding which Collector should receive Data Records. The assignment of the Collector Priority should consider factors such as geographical distance, communication cost, and Collector loading, etc. It is also possible for several Collectors to have the same priority. In this case, the Exporter could randomly choose one of them as the primary Collector to deliver Data Records.

2.2.1 Basic Types

The following tables illustrate the Type IDs for the base types used in the protocol. The type names are taken directly from XML-Schema Datatypes, when available, or have been added as part of the IPDR namespace.

Type ¹	Type ID
int	0x00000021
unsignedInt	0x00000022
long	0x00000023
unsignedLong	0x00000024
float	0x00000025
double	0x00000026
base64Binary	0x00000027
hexBinary	0x00000027
string	0x00000028
boolean	0x00000029
byte	0x0000002a
unsignedByte	0x0000002b
short	0x0000002c
unsignedShort	0x0000002d

The following are the Type IDs for the derived types used in the protocol:

Type	Type ID
dateTime	0x00000122
ipdr:dateTimeMsec	0x00000224
ipdr:ipV4Addr	0x00000323
ipdr:ipV6Addr	0x00000427
ipdr:UUID	0x00000527
ipdr:dateTimeUsec	0x00000627

2.2.2 Structures

Support for structures shall be a negotiable feature. The IPDR/XDR encoding format for IPDR 3.5 does however describe a model of binary encoding of structures which may be applied to IPDR/Streaming.

2.3 Data Representation

All data must use an encoding of big-endien.

¹ See the Business Solution Requirements for definitions of types.

IDL shall be used to describe data and message formats as well as the use of XDR with the following exceptions:

1. Indefinite length (IPDR/XDR docs [at least])
2. No 32-bit alignment padding

2.4 Templates

The IPDR/Streaming Protocol enables efficient delivery of Data Records. This is achieved by negotiating a set of Templates for a Session before actual Data Records are delivered. A Template defines the structure of a Data message payload by describing the data type, meaning, and location of the fields in the payload. By agreeing on Templates, Collectors understand how to process Data messages received from an Exporter. As a result, an Exporter only needs to deliver actual data attributes (Fields) without attaching any descriptors of the data; this reduces the volume of information sent over communication links.

A Template is an ordered list of Field Identifiers. A Field Identifier is the specification of a Field in the Template. It specifies an accounting item that a Service Element may collect and export. Each Field specifies the Type of the Field.

A Template references an IPDR Service Definition document, where a more complete definition of the Template is included.

The Template is optionally negotiated upon setup of the communication between the Exporter and the Collector. This allows the Exporter to avoid sending Fields that the Collector is not interested in.

The IPDR/Streaming Protocol supports usage of several Templates concurrently (for different types of records). Fields contained in a Template could be enabled or disabled. An enabled Field implies that the outgoing DR data record will contain the data item specified by the key. A disabled Field implies that the outgoing record will omit the specified data item. The enabling/disabling mechanism further reduces bandwidth requirement; it could also reduce processing in Service Elements, as only needed data items are produced.

In an IPDR/Streaming Protocol Session, all the Collectors and the Exporter must use the same set of Templates and associated negotiated enable/disable status. The Templates' configuration and connectivity to an end application must be the same in all Collectors. The Collector must publish the relevant Templates to all Collectors in a Session through user configuration, before it starts to send data according to the Templates.

The complete set of templates residing in an Exporter must bear a configuration ID that identifies the Template set. Each Data Record is delivered with the Template ID and the Configuration ID, so that the correct Template can be referenced. A Collector, when receiving a record with an older Configuration ID, may handle the record gracefully by keeping some Template history.

The transport layer should ensure that a Collector would not get messages with future Configuration IDs.

2.4.1 Template Representation

Templates reference IPDR Service Definition as a formal, externally defined specification of the Template structure. The following is the IDL fragment that defines the formal specification of the TemplateBlock and FieldDescriptor used to describe the structure of a single Template:

```
struct FieldDescriptor{
int      typeId;      /* ID of field type */
int      fieldId;     /* unqualified field code that can be used
/* as alternative accessor handles to fields
UTF8String  fieldName; /* Note that field names are to be qualified
/* with the Namespace name, as an example:
/* http://www.ipdr.org/namespace:movieId
/* The namespace must match one of those
/* targeted by the schema or schema imports
};

struct TemplateBlock{
short    templateId; /* ID of template - context within configId
/* Provides numeric identifier to
/* IPDR service specification for context of
/* session/config
UTF8String  schemaName; /* Reference to IPDR service specification
UTF8String  typeName; /* Reference to IPDR service specification
FieldDescriptor fields<>; /* Fields in this template
};
```

Notice that field schemaName and typeName form the external reference to the IPDR Service Definition whereas the Template would later be referenced only with templateId – a numeric identifier assigned by the Exporter to the Template.

Fields within a template are identified by fieldName. fieldName must be qualified within the namespace. Each Field also defines the Data Type of the Field by specifying typeId. The fieldId can be optionally set by the Exporter to uniquely allow access to (set/get of Fields) without using the textual fieldName. The templateId and fieldId are not guaranteed to maintain the same value between different Sessions and different connections unless externally declared otherwise by the Exporter.

2.4.2 Template Transmission and Negotiation

In the simplest form, the Exporter declares the Templates it employs and communicates this information to one or more Collectors. Optionally, Templates may be negotiated. The negotiation is an advanced and advantageous feature compared to most of the alternative protocols for usage exchange. This allows the Collector and Exporter to negotiate a set of Fields within Templates to be sent. Rather than requiring the Exporter to export all Fields, whether the Collector is interested in receiving them or not, this allows the Collector(s) to indicate to the Exporter which Fields they are interested in and this allows the Exporter to determine whether or not to support the requirements of the Collector.

Ultimately, the Exporter determines which Fields to send. A simple Exporter may indicate that Template negotiation is not a supported capability. Another Exporter may

support negotiation, but may based on local configuration chose to send Fields in the initial proposed Template even if the Collector did not indicate an interest.

IPDR/Streaming Protocol has a feature of capability negotiation. The capability to negotiate Templates is a capability that may exist or be absent from a compliant implementation. A Collector and an Exporter must be able to communicate with a party that is different than its own capabilities. For example, a Collector that supports Template negotiation should be able to interoperate with both an Exporter that does and doesn't support Template negotiation. It can determine which type of Exporter it is connected to through the capability negotiation that precedes the potential Template negotiation.

Templates are negotiable between an Exporter and a Collector. A Collector may propose changes to the Templates received from an Exporter (e.g., enabling some keys and disabling others), or it can acknowledge the Templates as is. In the case that a Template or a Field is not recognized by the Collector (e.g., they might be added to the Exporter after the Collector configuration has completed), the Collector may choose to disable each unknown Field or unknown Templates in order to avoid unnecessary traffic. A Template is disabled when all the keys are disabled It is the Exporter's responsibility to decide what would be the final set of Templates used by a Session
The Collector must not disable Fields in the Template if they are mandatory based on the IPDR Service Definition unless the Collector is disabling all the Fields in a Template to effectively disable transmission of that Data Record type.

Templates are negotiated only with the highest priority Collector that first establishes connectivity with the Exporter within a Session. All other Collectors negotiations attempts will not affect the Templates exported by the Exporter.

2.4.3 Changing Templates

Over time the set of information available from an Exporter may change.

In this case it may be desirable for the Exporter to modify any existing Sessions in order to begin transmitting Data Records which incorporate these changes.

IPDR/Streaming offers a few mechanisms to accomplish this change:

1. Introduce a new TemplateData message into the existing Session which describes the modified Data Record(s). Subsequent Data messages on the stream may then refer to this new templateId.
2. End the current Session by sending a SessionStop message and subsequently issue a new SessionStart message indicating the beginning of a new Session. The new set of Templates may then be announced prior to sending any Data Records. The Exporter in this case should indicate a boundary in the Data Record stream by specifying a new documentId in the SessionStart message and reset the sequence number to 0.

3. If renegotiation of templates is desired, the Exporter should send a SessionStop followed by a Disconnect. Upon reestablishment of the connection with the Collector, Template Negotiation may be used to allow the Collector to identify the set of Fields of interest to the Collector. The Exporter should (as in option 2) specify a new documentId for this session and reset the sequence number to 0.

In the case of multiple Collectors available for the same Session, the exporter may utilize techniques 2 or 3 to inform the Collectors with lower Priority of the change.

This specification does not define the means by which an Exporter is configured to utilize a new set of Templates, nor does it define the means by which an Exporter determines which of the options cited above will be applied..

2.5 Flow Control

Flow control mechanisms are employed to ensure that data is sent from an Exporter to a Collector only if it is ready to receive data. Flow control mechanisms are likewise used to indicate to the Collector whether an Exporter is sending to it data as the primary Collector or it is a redundant/backup Collector for some other Collector that is currently Primary. The Flow control also provides information on the data sequence numbers and document Id so that the Collectors can collectively guarantee that no Data Records are lost.

Four messages (FlowStart, FlowStop, SessionStart and SessionStop) are employed to support flow control.

FlowStart and FlowStop are sent by the Collector to indicate whether it is ready or not ready to receive data, in the case of FlowStop it provides information to the Exporter about why the Collector is no longer willing to receive Data Records via the reasonCode and reasonInfo fields.

```
struct FlowStart {
struct IPDRStreamingHeader header;
};

struct FlowStop {
struct IPDRStreamingHeader header;
short reasonCode;          /* values of 0-255 are reserved for standard */
                           /* reason codes. Values > 255 may be used for */
                           /* vendor specific codes. */
                           /* 0 = normal process termination */
                           /* 1 = termination due to process error */
                           /*
UTF8String reasonInfo;
};
```

SessionStart and SessionStop messages are sent by the Exporter to designate the associated Collector the active/inactive Collector and to provide information static within the Session. For instance, information that could be used to allow de-duplication by the Collector for arbitrarily long timeframes as well as identification of Data Records within IPDR documents are provided, as specified below:

```
struct SessionStart {
    struct IPDRStreamingHeader header;
    int exporterBootTime; /* boot time of exporter */
    long firstRecordSequenceNumber; /* first sequence number to be expected */
    long droppedRecordCount /* number of records dropped in gap situations */
    boolean primary; /* indication that the collector is the primary */
    int ackTimeInterval; /* the maximum time between acks from collector */
    int ackSequenceInterval; /* the maximum number of unacknowledged records */
    char documentId[16]; /* the UUID associated with the info being sent */
                        /* in this session */
};

struct SessionStop {
    struct IPDRStreamingHeader header;
    short reasonCode; /* values of 0-255 are reserved for standard */
                    /* reason codes. Values > 255 may be used for */
                    /* vendor specific codes. */
                    /* 0 = end of data for session */
                    /* 1 = handing off to higher priority coll */
                    /* 2 = exporter terminating */
                    /* 3 = congestion detected */
    UTF8String reasonInfo;
};
```

Data Records have 64-bit counters. Together with a UUID that represents the document, it is possible to uniquely distinguish records from one another. Additional measures can be used such as `exporterBootTime`. `firstRecordSequenceNumber` could be used by the Collector to ensure that it correctly expects the right sequence ordering sent by the Exporter.

Also defined per session are parameters that determine the maximum time and number of records permitted by the Exporter between acknowledgements are sent from the Collector. The Collector must acknowledge records within this timeframe.

In addition to explicit flow control messages mentioned above. KeepAlive messages can be periodically sent between either communicating party to the other to ensure the connection is still available. These KeepAlive messages are sent on both primary connections and standby/backup connections to ensure that backup links are also operational in case of a fail-over due to failure of link or active/primary Collector.

After Templates have been negotiated or set, if negotiation is supported, and both Session and Flow have been started, Data messages are sent from the Exporter to the active Collector. This is usually the highest priority Collector that is connected and operational. Each Data message contains a Data Sequence Number (DSN). The primary Collector must accept the data as long as it is in-sequence. Out-of-sequence Data messages should be discarded.

Upon reception of the message with initial DSN (equal to that of the `firstRecordSequenceNumber` set in the SessionStart message), the Collector must accept all in-sequence DATA messages. The DSN must be incremented by 1 for each new DATA message originated from the Exporter.

A Collector must acknowledge the reception and correct processing of Data messages by intermittently sending DataAcknowledge messages when the window of outstanding data

records is closing or acknowledgement timers fire. The DataAcknowledge must contain the DSN of the last processed in-sequence Data message.

The Exporter is responsible for delivering all the records. In the case of a redundant Collectors configuration, there could be a scenario when one Collector does not receive all the records but another redundant Collector for the same collection system receives the rest of the records. For example, Collector #1 could receive records 3042-3095 and then 3123-..., with Collector #2 receiving records 3096-3122. It is the Exporter's responsibility to deliver all the records, in-sequence, but not necessarily to the same Collector.

The Collection System eventually receives all the Data Records, possibly through more than one Collector. The Exporter must convey all the records it received to the collection system. This may result in duplicate records in the collection system. In this case, the DSN must be used to remove duplicates. To aid the process of duplicate removal, whenever a Data Record is re-sent (sent for any time past the first time an attempt to send it has been attempted) to another Collector, its 'D' bit must be set to suggest that this Data Record might be a duplicate.

When the amount of unacknowledged Data Records reaches a threshold or the time since the last DataAcknowledge message exceeds that set during the session initiation, all unacknowledged Data Records should be transmitted to an alternate Collector with the 'D' bit set in the Data message(s); if alternate Collectors are not available, the Exporter may in response to system limitations chose to drop some accounting records. This loss would be indicated by a gap in sequence numbers.

2.6 Multiplexed Streams

It is sometimes desirable to support multiple different sets of Templates for different applications. For instance, one set of Templates may relate to Accounting/Billing data as well as associated Audit information while another set of Templates may be used for Fraud application or Traffic Engineering.

Sessions define the set of Templates supported by an Exporter. The basic Exporter supports at least one Session that defines the set of Templates it exports.

Although it is desirable to support multiple sets of Templates in some instances, it is not generally so. Therefore, the capability to support multiple Sessions is a capability that may be optionally supported and is negotiated as part of the capabilities negotiation stage of the protocol.

Whether within a single Session or within multiple Sessions, information related to multiple Templates is also multiplexed over the same connection. The protocol message header indicates which Session that message relates to, to support the multiplexing of multiple Sessions over the same connection. See the protocol message header, below:

```
struct IPDRStreamingHeader {
    char version;      /* version of protocol, for this version, set to 2      */
    char messageId;   /* the ID of this message (see MessageIds)                  */
    char sessionId;   /* the ID of this session or 0 if not session-specific      */
}
```

```
    char messageFlags; /* reserved/unused and set to 0 */
    int  messageLen;  /* length in bytes of message including header */
};
```

Here sessionId is the dynamic/transient identifier associated with a given Session that distinguishes any of the messages for one Session from the other emanating from the same Exporter.

Each Data Message indicates in its header the templateId to which it relates, as can be seen below:

```
struct Data {
    struct IPDRStreamingHeader header;
    short templateId; /* a template id relative to the session defined
                      /* in the header. The fields in this template were
                      /* reported at the beginning of the session via
                      /* TemplateData messages
    char  configId; /* (see above)
    char  flags; /* currently just duplicate flag
    long  sequenceNum; /* session relative sequence number of this record
    opaque dataRecord<>; /* XDR encoded fields based defined by templateId
};
```

The ability to support multiple Sessions is indicated by setting the MULTISESSION bit in the capabilities bit-mask set within the Connect message. If both communicating parties support MULTISESSION then multiple sessions may exist. An Exporter may indicate support for MULTISESSION but still export only one Session.

2.7 Reliability

Two primary means are employed to reduce the likelihood of data loss:

- Any number of Collectors can be connected to an Exporter
- DataRecords are acknowledged by Collectors when received. The Collector should make every attempt to make the data recoverable upon Collector failure before acknowledging data records. As an example, a Collector may write the records received to a redundant persistent storage array, flush and sync the disk to assure survivability in case of Collector failure. This provides a rather high degree of confidence that the data can be recovered. The indication of acknowledgement from a Collector can be assumed by the Exporter to mean that it no longer has responsibility for these Data Records and may remove them from its transient buffer.

For purposes of improved reliability and robustness, redundant Collectors configuration may be employed. Deployment of redundant collectors is a deployment choice by the operator which will be based on the business impact of lost Data Records.

Additional features such as load balancing may be implemented in a multi-Collector environment. The process of configuring Exporter is carried out using the NE's configuration system and is outside the scope of this document.

An Exporter should deliver Data Records to its perceived operating Collector with the highest priority; if this Collector is deemed unreachable, the Exporter must deliver the Data Records to the next highest priority Collector that is perceived to be operating

Data Record delivery should revert to the higher priority Collector when it is perceived to be operating again.

The protocol does not specify how an Exporter should redirect Data Records to other Collectors, which is considered an implementation issue. But all the supporting mechanisms are provided by the protocol to work in a multiple-Collector environment (e.g., the template negotiation process, and configuration procedures, etc.). The transport layer (together with some other means) is responsible for monitoring Collector's responsiveness and notifying protocol for any failures. The Exporter may choose to transition to an alternate Collector.

When a Session is initially established from an exporter to a collector, a unique documentId is assigned by the exporter in the SessionStart message.

If a Session is subsequently reestablished with a Collector as part of recovery, and it represents the continuation of some prior stream of data, the Exporter should indicate this by using the same documentId as was used in the previous Session. In this recovery situation the seqNum should be in sequence with the previous stream. This may include sequence numbers already used, if some records which were not acknowledged on the old stream are being retransmitted.

An Exporter can indicate logical boundaries in the stream (e.g. indicating a new logical document of records), by closing and then reestablishing a Session with the Collector using a new unique documentId.

The sequence numbering of the data records should begin at 0 each time a new documentId is assigned to a Session.

2.8 State Machines

The following diagram (figure 1) illustrates the basic state machine employed by both an IPDR Exporter and Collector. Independent Exporter and Collector state machine instances exist on each end of the connection. And after moving to the ReadyToReceive state a unique state machine exists for each Session.

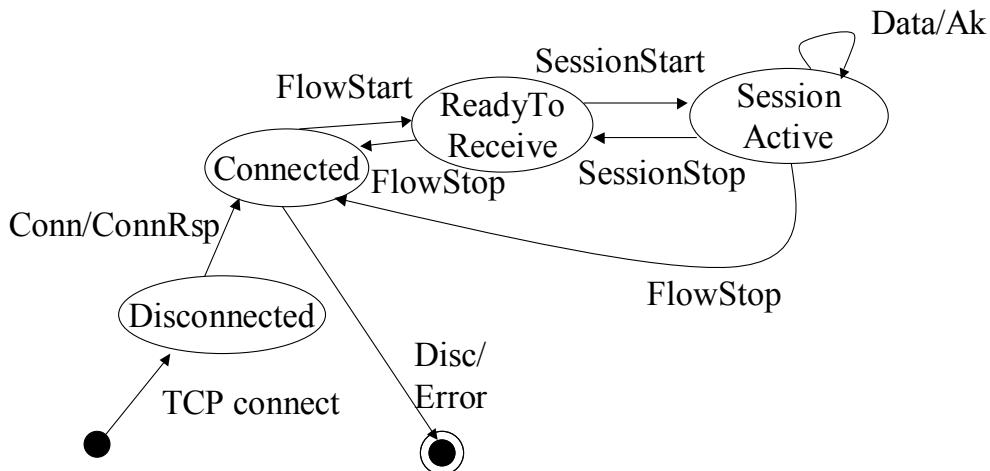


Figure 1: IPDR/Streaming Simple State Diagram

Either the collector or the exporter may initiate connections; the receiving party of a Connect request is responsible for sending the corresponding Connect Response. The initiator of the lower level reliable transport connection must send the Connect.

Section 2.5 details the messages and the state transitions for the above diagram. Figure 2 illustrates the state machine when the optional template negotiation mechanism is employed.

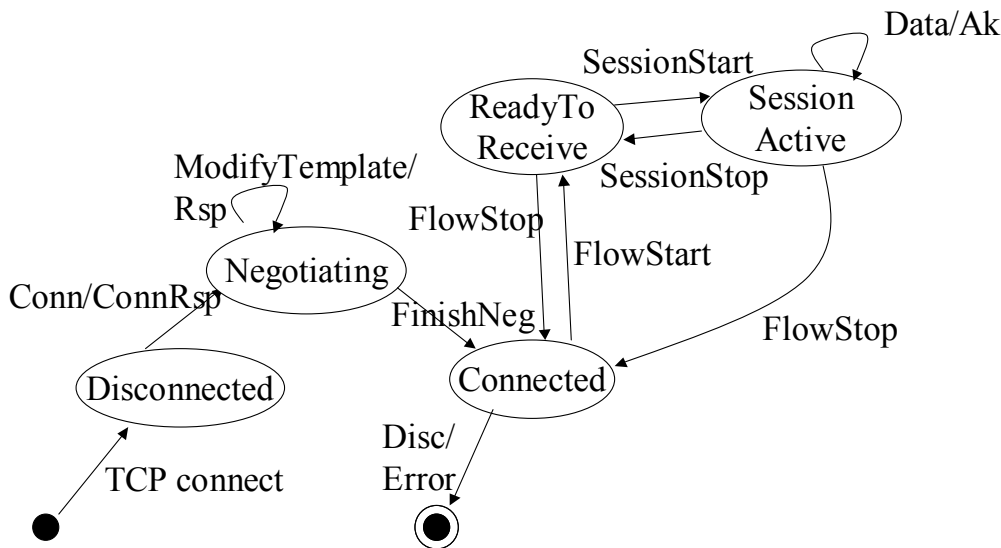


Figure 2: IPDR/Streaming Negotiation State Machine

Support for template negotiation is identified at connection time based on the capabilities flags sent on the ConnReq and ConnRsp. If both sides indicate support for template negotiation, then the “Negotiating” state is entered as opposed to the “Connected” state. While in the negotiating state, the collector may send one or more ModifyTemplate requests to request a change in the

accounting information for a particular session. For example it may only want to receive a subset of the fields available from the exporter in order to reduce throughput requirements.

The collector issues a FinishNeg message to conclude the negotiation phase and enter the Connected state. From that point, the state machine behavior is identical to the simpler “no negotiation” machine illustrated in Figure 1.

2.9 Exporter Query Messages

A collector may query an exporter’s status by sending query messages after it has established a connection with the exporter. The exporter must respond with response messages. The collector must initiate all the query messages.

Two query messages are defined:

- GetSess – returns information about all the sessions that are available from this exporter.
- GetTemplates – returns information about the templates used on a particular session.

2.10 Sessions

IPDR/Streaming supports the ability to have multiple sessions for communication of different types of accounting records to different Collectors. For example, performance related information to one Collector and billing-related information to another. At times, the same Collector may serve as the Collector for multiple types of information. Therefore, the ability to have multiple sessions on the same link is desirable.

An exporter may deliver Data Records to different collection systems by establishing different sessions.

Each session may consist of several collectors in a redundant configuration. The session ID embedded in all the IPDR/Streaming messages determines which session a given IPDR/Streaming message is associated with.

Once a collector’s relationship with an exporter is in the connected state, the collector indicates its willingness to participate in sessions by issuing a FlowStart message for each target session.

An exporter indicates the status of a given Session with a collector via the SessionStart and SessionStop messages.

Each session has its set of templates (these may be the same templates, but the Fields could be enabled or disabled differently). The available sessions are configured in the exporter, each with a different session name with associated Session IDs. The means of this configuration is outside the scope of this specification.

A collector may take part in different sessions. When configuring a collector, it needs to know the sessions in which it participates. The exporter can issue a GET SESS message to receive a list of available sessions provided by an exporter. The configuration of sessions that a collector may participate is outside the scope of this specification.

2.11 Backwards Compatibility

IPDR/Streaming builds upon two existing specifications, namely IPDR's XDR file format and RFC3423, Common Reliable Accounting for Network Elements (CRANE). As part of this evolution, some level of compatibility with previous versions is a goal.

IPDR/Streaming uses the same basic version negotiation mechanism as the original CRANE and borrows much of its message set from CRANE. IPDR/Streaming messages are distinguished from CRANE messages by advancing the version field in the message headers to 2.

Beginning with IPDR version 3.5 IPDR/XDR and IPDR/XML formats are able to record the contents of any IPDR/Streaming stream as documents.

In the case of IPDR/XML, this is supported by further expansion of the XML-Schema subset which may be used in creating IPDR Service definitions, and the generation of XML document instances which are valid according to the Schema.

In the case of IPDR/XDR, there are complementary extensions which align it with the encoding model of IPDR/Streaming messages. Advancing the version number in the IPDR/XDR header from 3 to 4 indicates IPDR/XDR documents capable of preserving any IPDR/Streaming flow of accounting records.

The reader may notice some discrepancies between IPDR/XDR's template and data encodings and those of IPDR/Streaming. These are considered artifacts of the streaming versus documented oriented design center. IPDR/XDR and IPDR/Streaming are aligned in terms of their information content.

2.12 Connection Establishment

The IPDR/Streaming protocol state machine allows for connections to be established by either the Exporter or Collector.

An implementation of an Exporter may choose to only enable connection establishment in one direction. The Exporter may only initiate connections and not accept inbound connections, or it may only accept inbound connections and not attempt to create connections itself.

To ensure interoperability a Collector should support both directions of connection establishment.

It may prove useful in some deployments to choose a model for connection establishment that is effectively unidirectional; e.g. all Collectors establish connections or accept connections. For instance security policies or device constraints may dictate a unidirectional approach.

In general, support for both directions of connection may reduce the delay in reestablishing communication when either a collector or exporter is restarting. Initiating a connection upon startup provides the quickest means to identify that an Exporter or Collector is available. Alternatively polling strategies may be employed, where connection reestablishment attempts are made during some configurable interval. Means of configuring such intervals are outside the scope of this document.

3 Message Format

IPDR/Streaming describes its message format using an augmented form of RFC1832, External Data Representation (XDR).

Two augmentations of XDR used by IPDR are as follows:

1. Support for indefinite length specification. This allows for stream based encoding of information without knowing or calculating the entire length of a message or document in advance. The value of -1 in a length field indicates that, based on template information, a decoder should be able to determine where a message completes.
2. No 32-bit alignment padding. Beginning in IPDR 3.5, both IPDR/XDR and IPDR/Streaming remove the padding constraint specified by XDR. This allows for specification to the byte level of structures. This augmentation is described in section 3.20 of RFC1832, "Areas for Future Enhancement".

By using these two extensions the protocol specification can move away from the use of "ASCII art" common in most IETF protocol specifications and use a more concise and formal C style syntax for describing protocol message formats.

Note it would be possible to create automated tools to generate ASCII art from the XDR IDL specification language, the converse would be significantly more difficult.

3.1 XDR equivalence to "ASCII art"

The use of XDR specification of protocol message structure is a departure from the majority of IETF protocols operating in the accounting domain. Rather the typical mode of specification is based on the use of "ASCII art" diagrams which illustrate a series of bytes and the position various fields occupy in that byte table.

XDR Augmentation number 2 cited above, allows the same information to be conveyed in the more concise and machine consumable format.

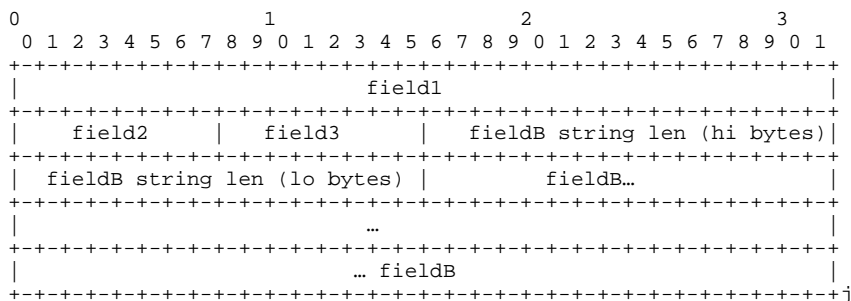
Note that one might also consider a "pure" XML mechanism for this based on additional Schema constraints and mapping policies. However, existing tools and specifications supporting XDR's byte level encoding make the use of XDR IDL convenient for our current problem.

By way of example, the basic mapping from an XDR style specification to its equivalent ASCII art model is illustrated below:

```
struct ExampleA {
    int field1;
    char field2;
    char field3;
}

struct ExampleB {
    struct ExampleA fieldA;
    UTF8String fieldB;
}
```

Example B layout:



The ExampleB structure has at its beginning an embedded ExampleA structure. The ASCII art shows the ExampleA structure elements in their appropriate sizes and position.

UTF8String's have a 32-bit length field according to XDR encoding rules. Since we are augmenting the XDR encoding by doing byte level packing, the string length is not aligned to 32-bit word boundaries. The fieldB string is variable length, so is shown spanning multiple bytes of indeterminate length. Note that fieldB's contents also begin at the packed byte boundary, no 32-bit boundary alignment is performed.

If 32-bit alignment is desired, it can be achieved by populating structures with a reserved char, or several to pad out the location of a desired aligned field.

3.2 Common Header

All IPDR/Streaming Messages begin with the following 8 byte header:

```

struct IPDRStreamingHeader {
char version;          /* version of protocol, for this version, set to 2          */
char messageId;       /* the ID of this message (see MessageIds)                       */
char sessionId;       /* the ID of this session or 0 if not session-specific          */
char messageFlags;    /* reserved/unused and set to 0                                  */
int messageLen;       /* length in bytes of message including header                   */
};

```

It is followed by a variable length message payload. Some of the messages do not have any additional payload part. All IPDR/Streaming messages are XDR encoded in Network Byte order.

Details of each field are described below:

- version - indicates the supported IPDR/Streaming protocol implementation. This field must be set to 2.

-
- msgId - identifies the type of the message. The message IDs defined by IPDR/Streaming 3.5 are defined as follows. See Table 1. in section 4 for additional details of each message.
 - sessionId - identifies the session with which the message is associated. The session ID is ignored in the case of basic connection related messages. Connection related messages are: CONNECT, CONNECT_RESPONSE, DISCONNECT, KEEP_ALIVE, FINISH_NEGOTIATION, GET_SESSIONS, GET_SESSIONS_RESPONSE and ERROR. For implementations which do not support multiple sessions, the sessionId 0 should always be used
 - msgFlags - used to identify options associated with the message. Currently no flags are defined.
 - msgLen - the total length of the IPDR/Streaming message in octets including the header.

The message set supported by a basic valid implementation is dependent on the capabilities identified in the connection request.

4 Message Details

Table 1 summarizes the message set defined for IPDR/Streaming v3.5. The following sections will provide details on the structure of each message, using the augmented XDR encoding described in section 3.

Table 1

Category	ID	Message	Direction	Comments
Connection	0x05	Connect	Either	
	0x06	ConnectResponse	Either	Opposite of Connect
	0x07	Disconnect	Either	
Errors	0x23	Error	Either	In lieu of response, underlying connection cleared
Flow Control	0x01	FlowStart	C→E	Framed in sessions
	0x08	SessionStart	E→C	
	0x03	FlowStop	C→E	
	0x09	SessionStop	E→C	
Template	0x10	TemplateData	E→C	
	0x1a	ModifyTemplate	C→E	Optional
	0x1b	ModifyTemplateResponse	E→C	Optional
	0x1c	FinishNegotiation	C→E	Optional
Data	0x20	Data	E→C	
	0x21	DataAcknowledge	C→E	
State Independent	0x14	GetSessions	C→E	
	0x15	GetSessionsResponse	E→C	
	0x16	GetTemplates	C→E	
	0x17	GetTemplatesResponse	E→C	
	0x40	KeepAlive	Either	May affect fail-over

4.1 Connection

Connection related messages deal with moving the state machine for an IPDR/Streaming Exporter and Collector from the Disconnected State to either the Connected or negotiating state. For more details on the state machine, see the Figures and discussion in section 2.8.

The connection and connectionRsp messages provide for both the acknowledgement of both peers that they are capable of participating in the exchange of records via IPDR/Streaming as well as a means of both peers indicating any optional protocol capabilities they support. To use any optional capability, both exporter and collector must indicate their support of that capability.

4.1.1 Connect

```
struct Connect {
    struct IPDRStreamingHeader header;
```

```
int collectorAddress; /* IPv4 IP address of the collector */
short collectorPort; /* The transport protocol port number of the collector */
int capabilities; /* an array of capability bits for capability negotiation*/
int keepAliveInterval; /* the maximum delay between some indication from remote */
};
```

The creator of the underlying reliable transport connection must send the connection message. Either the exporter or the collector may initiate the underlying transport connection.

The Connect message has the msgId set to 0x5 in the header.

The capabilities flags indicate supported options for this connection. Specifically the following capabilities are optional:

- **STRUCTURES** – if set, this flag indicates that the accounting records being sent may be structured as supported in IPDR 3.5, vs. being in first normal form, as specified in IPDR 3.1.1. Structured data includes repeating primitive type fields or substructures within a single record. Common scenarios where this might arise are in conference calls or other multiparty accounting that may group all parties in a single record. Note that typical Relational Databases require data in first normal form, so there are benefits to observing this model. However, the need to represent existing industry standard structures and still carry them in IPDR has driven the support for STRUCTURES.
- **MULTISESSION** – if set indicates that the party allows support for multiple sessions of accounting records. Simpler equipment and implementations may chose to only support a single stream.
- **TEMPLATE_NEGOTIATION** – if set allows the collector to request a different set of accounting records or fields than that advertised initially by the exporter. Simpler exporters may choose not to support negotiation. In this case this capability should be off, and completing the connection dialog puts the two parties in the Connected state. Negotiating exporters and collectors enter a template negotiation state after exchanging the connection dialog when this flag is set.

The KeepAlive interval indicates the maximum amount of idle time on a connection before a KeepAlive message should be set to assure the underlying transport connection is still available. This value is expressed in seconds.

4.1.2 ConnectResponse

```
struct ConnectResponse {
    struct IPDRStreamingHeader header;
    int capabilities; /* an array of capability bits for capability negotiation*/
    int keepAliveInterval; /* the maximum delay between some indication from remote */
};
```

The recipient of a connect message may send a ConnectResponse. The response indicates the desired keep alive interval of the responder as well as its capabilities.

The meaning of capabilities and keepAlive are the same as defined in Connect.

Note that the responder should only indicate those capabilities which were previously sent by the connecting party and which it is willing to support. Identifying capabilities which were not proposed by the connector has no effect, those capabilities are not available for the current connection.

4.1.3 Disconnect

```
struct Disconnect {
    struct IPDRStreamingHeader header;
};
```

Either party in the course of graceful termination of a connection may send the Disconnect message. It is not acknowledged and should be followed by sending party disconnecting the underlying transport, and the receiving party, issuing the corresponding close of connection on their side.

The disconnect has a msgId of 0x07 in the header.

4.2 Errors

Either the exporter or the collector may issue an Error message in the event where either a communication error has been detected or other failures impacting the connection. The error message must be followed by the sender initiating a disconnect of the underlying transport. The recipient of an Error will also issue the corresponding close of connection on their side.

4.2.1 Error

```
struct Error {
    struct IPDRStreamingHeader header;
    int timeStamp;           /* time of error (in seconds from epoch) */
    short errorCode;        /* values of 0-255 are reserved for standard */
                          /* reason codes. Values > 255 may be used for */
                          /* vendor specific codes. */
                          /* 0 = keepalive expired */
                          /* 1 = message invalid for capabilities */
                          /* 2 = message invalid for state */
                          /* 3 = message decode error */
                          /* 4 = process terminating */
    UTF8String description;
};
```

4.3 Flow Control

Appropriate flow control is necessary for managing the reliable delivery of accounting records and ensuring the collector has recorded them.

Flow control deals with the acknowledging and throttling of exported streams of records, as well as enabling redundant collectors to perform rapid failover with duplicate detection.

Bear in mind that IPDR/Streaming already assumes an underlying reliable transport. However the protocols at this level leave ambiguous details about the disposition of data in transit at the time of a failure. A simple application level acknowledgement scheme, based on windows of configurable sizes, allows for this necessary exchange between the exporter and collector, while imposing minimal overhead.

4.3.1 FlowStart

```
struct FlowStart {
    struct IPDRStreamingHeader header;
};
```

FlowStart messages may only be sent by the collector to indicate its willingness to participate in a session for a particular stream of accounting records.

4.3.2 SessionStart

```
struct SessionStart {
    struct IPDRStreamingHeader header;
    int exporterBootTime; /* boot time of exporter */
    long firstRecordSequenceNumber; /* first sequence number to be expected */
    long droppedRecordCount /* number of records dropped in gap situations */
    boolean primary; /* indication that the collector is the primary */
    int ackTimeInterval; /* the maximum time between acks from collector */
    int ackSequenceInterval; /* the maximum number of unacknowledged records */
    char documentId[16]; /* the UUID associated with the info being sent */
    /* in this session */
};
```

SessionStart messages are only sent by the exporter to indicate that an accounting stream is now actively flowing to a collector. This distinguishes between the situations where no accounting records are available and when accounting records are being sent to an alternate collector.

4.3.3 FlowStop

```
struct FlowStop {
    struct IPDRStreamingHeader header;
    short reasonCode; /* values of 0-255 are reserved for standard */
    /* reason codes. Values > 255 may be used for */
    /* vendor specific codes. */
    /* 0 = normal process termination */
    /* 1 = termination due to process error */
    UTF8String reasonInfo;
};
```

FlowStop messages may only be sent by the collector to indicate that it is no longer able to participate in a particular session. The reasonInfo field contains a string which the Collector may use to provide additional details. The Exporter may choose to log this information for operational support purposes.

4.3.4 SessionStop

```
struct SessionStop {
    struct IPDRStreamingHeader header;
    short reasonCode; /* values of 0-255 are reserved for standard */
    /* reason codes. Values > 255 may be used for */
    /* vendor specific codes. */
    /* 0 = end of data for session */
    /* 1 = handing off to higher priority coll */
    /* 2 = exporter terminating */
    /* 3 = congestion detected */
    UTF8String reasonInfo;
};
```

SessionStop messages are only sent by the exporter to indicate that a collector is no longer the active recipient of a stream of accounting records.

4.4 Template

Templates form the foundation for efficient exchange of large volumes of similar accounting records. By summarizing the different accounting record fields, their types and order in a template, the individual encoded accounting records can be efficiently packed according to augmented XDR.

IPDR/Streaming borrows the same self describing data model as used by IPDR/XDR and IPDR/File formats. Specifically templates point to XML-Schemas which are compliant with the IPDR Service Definition Guidelines. The templates also identify the field's name and type code. By putting the name and type code in the template, along with the schema, a consumer need not have access to the schema directly in order to decode Data messages, because the types all have well defined encoding policies.

```

    struct FieldDescriptor{
int         typeId;          /* ID of field type */
int         fieldId;        /* unqualified field code that can be used */
UTF8String  fieldName;     /* as alternative accessor handles to fields */
/* Note that field names are to be qualified */
/* with the Namespace name, as an example: */
/* http://www.ipdr.org/namespace:movieID */
/* The namespace must match one of those */
/* targeted by the schema or schema imports */
};
    struct TemplateBlock{
short       templateId;    /* ID of template - context within configId */
/* Provides numeric identifier to */
/* IPDR service specification for context of */
/* session/config */
UTF8String  schemaName;   /* Reference to IPDR service specification */
UTF8String  typeName;     /* Reference to IPDR service specification */
FieldDescriptor fields<>; /* Fields in this template */
};

```

4.4.1 TemplateData

```

struct TemplateData {
struct IPDRStreamingHeader header;
char configId;
/* Identifies context of template definitions.*/
/* Changes in template should use a different */
/* configId (0 if unused) */
char flags;
/* Unused and reserved */
TemplateBlock templates<>; /* Definitions of templates supported */
};

```

The exporter immediately following the establishment of a session sends a TemplateData message. The Template block identifies all the templates that will be used over this session. Sending Data messages that identify template Ids not carried in the TemplateData messages is invalid, and should result in an Error message being sent with decode error as the cause.

4.4.2 ModifyTemplate

```

struct ModifyTemplate {
struct IPDRStreamingHeader header;
char configId;
/* Identifies context of template definitions.*/
/* Changes in template should use a different */
/* configId (0 if unused) */
char flags;
/* Unused and reserved */
TemplateBlock changeTemplates<>; /* Definitions of templates */
};

```

If the template negotiation capability was identified by both parties during connection establishment, the Collector may issue zero or more ModifyTemplate requests in order to alter the set of accounting records and their fields which will be transferred.

The exporter is not obligated to recognize any of the proposed changes, but indicates on a ModifyTemplateResponse the set of templates for that session after applying any approved changes in the ModifyTemplate message.

4.4.3 ModifyTemplateResponse

```
struct ModifyTemplateResponse {
  struct IPDRStreamingHeader header;
  char configId; /* Identifies context of template definitions. */
                /* Changes in template should use a different */
                /* configId (0 if unused) */
  char flags; /* Unused and reserved */
  TemplateBlock resultTemplates<>; /* Definitions of templates - final results */
};
```

Upon receiving a ModifyTemplate message, the exporter is not obligated to recognize any of the proposed changes. It indicates on a ModifyTemplateResponse the set of templates for that session after applying any approved changes in the ModifyTemplate message.

4.4.4 FinishNegotiation

```
struct FinishNegotiation {
  struct IPDRStreamingHeader header;
};
```

The collector sends a FinishNegotiation to indicate that it is done with the modifications and is ready to begin session based delivery of accounting records.

4.5 Data

The Data message is the vehicle for all accounting records moved from the exporter towards a collector. The Data message uses the templating model described earlier to reduce the amount of redundant information passed in each message. Specifically, by specifying in advance the fields and their types and their order, the encoded data record can pack all the values together according to the augmented XDR encoding.

A sequence number does data acknowledgement on a configurable window of outstanding Data messages determined by the exporter. Because the underlying transport is reliable, this window may be set very large (e.g. hundreds or thousands of records) and does not grow and shrink over the session.

The exporter also specifies a minimal ack interval, to ensure timely acknowledgements when accounting record traffic volumes are low.

4.5.1 Data

```
struct Data {
  struct IPDRStreamingHeader header;
  short templateId; /* a template id relative to the session defined */
};
```

```

                                /* in the header. The fields in this template were */
                                /* reported at the beginning of the session via */
                                /* TemplateData messages */
                                /* (see above) */
char  configId;                /* currently just duplicate flag */
char  flags;                  /* session relative sequence number of this record */
long  sequenceNum;           /* XDR encoded fields based defined by templateId */
opaque dataRecord<>;
};

```

The exporter sends Data messages to a collector in the context of a session. The session id is carried in the message header.

The template id in the Data message refers to a template identifier previously sent by the exporter on a TemplateData message. This describes the fields, their order and type and is used to define the augmented XDR encoding applied to construct the binary dataRecord.

The sequence number is a relative to a session. It increases by one for each accounting record sent. The number begins at 0 when a new Session “document” is initiated. See StartSession.

The flags field contains one defined flag, duplicate, which if set indicates that this record was previously sent to another collector which failed to acknowledge its receipt.

4.5.2 DataAcknowledge

```

struct DataAcknowledge {
    struct IPDRStreamingHeader  header;
    char  configId;             /* (see above) */
    long  sequenceNum;         /* session relative sequence number of last record */
                                /* received in a series. */
};

```

4.6 State Independent

The following messages are available at any point in the state machine for IPDR/Streaming.

The two query messages, GetSessions and GetTemplates, are always initiated by a collector to identify the available streams of information emanating from an exporter.

The KeepAlive message is sent in low traffic periods to ensure that both parties are still in communication. Note that some reliable transports such as TCP may have large periods of time before signaling connection loss, hence an application KeepAlive allows for independent and low overhead monitoring of the connection between and exporter and a collector. SCTP’s more configurable timeout and failover mechanism may lessen the need for KeepAlive, but this conclusion has not yet been reached.

4.6.1 GetSessions

```

struct GetSessions {
    struct IPDRStreamingHeader  header;
    short requestId;           /* numeric ID of request for sessions */
};

```

The GetSessions message is sent by a collector to identify the streams of accounting records available on each stream provided by this exporter.

Note that based on authorization mechanisms not defined in this specification, an exporter may report different available sessions to different collectors.

Note that the SessionId setting is ignored by the exporter for this message.

4.6.2 GetSessionsResponse

```
struct GetSessionsResponse {
    struct IPDRStreamingHeader header;
    short requestId; /* allows match up of response to request */
    UTF8String vendorId; /* string to identify vendor that created */
    /* templates. "IPDR.org" if these are */
    /* IPDR defined templates. */
    struct SessionBlock sessionBlocks <>; /* description of supported sessions */
};
```

An exporter must respond to a GetSession message with the list of sessions available to this collector. The session lists may then in turn be used by a collector to issue GetTemplates messages for specific sessions of interest.

Note that the SessionId setting is ignored by the collector for this message.

4.6.3 GetTemplates

```
struct GetTemplates {
    struct IPDRStreamingHeader header;
    short requestId; /* numeric ID of request for templates */
};
```

A collector may identify the available templates for a given session context by issuing the GetTemplates message. Responses are linked to requests by setting the reqId.

4.6.4 GetTemplatesResponse

```
struct GetTemplatesResponse {
    struct IPDRStreamingHeader header;
    short requestId; /* allows match up of response to request */
    TemplateBlock currentTemplates <>; /* supported active templates */
};
```

An exporter must respond with a GetTemplatesResponse for any valid session which the collector is allowed to consume. If a sessionId is not available, an Error message should be sent and the connection terminated.

4.6.5 KeepAlive

```
struct KeepAlive {
    struct IPDRStreamingHeader header;
};
```

In situations where there is low traffic, it is useful to have application controlled knowledge about the availability of the connection peer.

The KeepAlive must be sent by either peer when no traffic has been sent by that party within the time requested on the Connect or ConnectResponse message.

An implementation may send KeepAlives at any time.

5 Underlying Transport

TCP may be used as a transport layer for IPDR/Streaming. If TCP is used an implementation must follow these rules:

- Either the Exporter or Collector may initiate TCP over a specific TCP port.
- The initiator of a connection should be responsible for reestablishing a connection in case of a failure.
- Messages are written as a stream of bytes into a TCP connection, the size of a message is specified by the Message Length field in the message header.

It is anticipated that in the future, other transports may be used to carry IPDR/Streaming messages. Any such future mechanisms will have their own usage specifications.

As discussed in Section 2.12 “Connection Establishment”, implementations may chose to only request or receive connections. Collectors should support both connection initiation and reception to ensure interoperability. Deployment considerations may influence the choice of connection models.

6 Service Discovery

6.1 UDP Protocol

Since the Streaming protocol may evolve in the future and it may run over different transport layers, a transport neutral version negotiation mechanism running over UDP is defined. An Exporter or a Collector should implement version discovery and negotiation as defined herein. Either party may inquire about the Streaming protocol version and transport layer support by sending a UDP packet on an agreed UDP port. If the receiving party implements version discovery and negotiation, it must respond to this request with a UDP packet carrying the protocol version, the transport type and the port number used for the specific transport. .

The inquiring party (either Collector or Exporter) sends the following message to query the other party's protocol support.

```
struct VersionRequest {
    int requesterAddress; /* IPv4 address of the version request initiator */
    int requesterBootTime; /* boot time of the version request initiator */
    char [4] msg; /* must be 'CRAN' for version 1 and 'IPDR' for version 2 */
}
```

The receiving party will respond with the following information about the protocols that it supports.

```
struct VersionResponse {
    ProtocolInfo defaultProtocol;
    ProtocolInfo [] additionalProtocols;
}
```

6.2 Capability Files

In addition to the UDP based mechanism described above, IPDR's existing transport mechanisms utilize an XML based capability file to define the supported protocols of an IPDR Transmitter (an Exporter in the case of IPDR/Streaming). The basic mechanism is described in the associated IPDR document .

This section defines the extensions to the IPDR Capability file structure to describe an IPDR/Streaming capable IPDR Transmitter.

The capability mechanism allows IPDR consumers to determine the formats and protocols supported by an IPDR Transmitter. IPDR Capability files are intended to be extended to address other IPDR transfer mechanisms in a uniform, extensible manner.

The "supportedProtocolItem" XML element is the basic unit for defining a means of communicating with an IPDR Transmitter (an Exporter). IPDR/Streaming represents one means

of transmission and as such the information for IPDR/Streaming is contained in one of these elements.

The following example shows the structure for XML capability files describing the IPDR/Streaming protocol. It mirrors the information content of the UDP based “versioning”, described previously. The items shown in **bold** indicate specific details related to IPDR/Streaming. The non-bold items are directly taken from the existing Capabilities specification.

```
<CapabilityRsp>
  <supportedProtocolList>
    <supportedProtocolItem version="2"
      protocolMapping="Streaming"
      encoding="XDR">
      <primitiveList>
        <primitiveItem>Push</primitiveItem>
      </primitiveList>
      <extension>
        <exporterAddress>192.168.1.22</exporterAddress>
        <transportType>TCP</transportType>
        <portNumber>666</portNumber>
      </extension>
    </supportedProtocolItem>
  </supportedProtocolList>
</CapabilityRsp>
```

The information content in the UDP model “versioning” is:

- a version number – the version shall be 2 for IPDR/Streaming.
- an exporter address – a fully qualified domain name or IP address.
- a transport type – for now this is the string “TCP”. SCTP or BEEP or other protocols may be mapped later.
- a port number – the integer port number where the exporter may be located.

This is sufficient for a collector to determine its ability to communicate with an exporter. Additional information can be determined using the messages defined in IPDR/Streaming itself.

An exporter may thus advertise a URI which locates this capabilities file and from the information contained within a collector can determine its options for communicating with that system.

7 Security

The IPDR/Streaming protocol can be viewed as an application running over a reliable transport layer, such as TCP or SCTP. The protocol is end-to-end in the sense that the messages are communicated between exporters and collectors identified by the host address and the transport protocol port number. Before any sessions can be initiated, a set of collectors' addresses should be provisioned on the exporter. Similarly, a collector maintains a list of exporters' addresses with which it communicates. The provisioning is typically carried out securely using a network management system; in this way, the end-points can be authenticated and authorized. As this scheme is static, without additional security protections the protocol is vulnerable to attacks such as address spoofing.

IPDR/Streaming does not offer strong security facilities; therefore, it cannot ensure confidentiality and integrity or non repudiation of its messages. It is strongly recommended that administrators evaluate their deployment configurations and implement appropriate security policies. For example, if the IPDR/Streaming is deployed over a local area network or a dedicated connection that ensures security, no additional security services or procedures may be required; however, if exporters and collectors are connected through the Internet, lower layer security services should be used.

To achieve strong security, lower layer security services are strongly recommended. The lower layer security services are transparent to IPDR/Streaming. Security mechanisms may be provided at the IP layer using IPsec , or it may be implemented for transport layer using TLS . The provisioning of the lower layer security services is out of the scope of this document.

8 Complete IPDR/Streaming IDL Definition

The following listing shows the **normative** IDL specification for constructing IPDR/Streaming messages.

```

/*****
 ipdr_streaming.idl - IPDR/Streaming Protocol XDR IDL definition

This XDR IDL is used to describe all messages and structures used by
IPDR/Streaming Protocol.

The IDL is organized in the following logical sub-sections:

- Header structure definition - common to all transport protocol messages.
- Enumeration of Message IDs.
- Structure of messages used in transport protocol.
- Data structures and enumerations used within messages.
- Messages and structures used in UDP version discovery and negotiation.

Note:
Comments within the text are not normative but rather only used to provide
minimal documentation. The complete description is available within the
IPDR/Streaming Protocol specification main body.
*****/

/*****
Header structure definition:
*****/

struct IPDRStreamingHeader {
    char version;          /* version of protocol, for this version, set to 2          */
    char messageId;       /* the ID of this message (see MessageIds)                    */
    char sessionId;       /* the ID of this session or 0 if not session-specific         */
    char messageFlags;    /* reserved/unused and set to 0                               */
    int messageLen;       /* length in bytes of message including header                */
};

/*****
Enumeration of Message IDs:
*****/

enum MessageIds {
    CONNECT = 0x05,
    CONNECT_RESPONSE = 0x06,
    DISCONNECT = 0x07,
    FLOW_START = 0x01,
    FLOW_STOP = 0x03,
    SESSION_START = 0x08,
    SESSION_STOP = 0x09,
    KEEP_ALIVE = 0x40,
    TEMPLATE_DATA = 0x10,
    MODIFY_TEMPLATE = 0x1a,
    MODIFY_TEMPLATE_RESPONSE = 0x1b,
    FINISH_NEGOTIATION = 0x1c,
    GET_SESSIONS = 0x14,
    GET_SESSIONS_RESPONSE = 0x15,
    GET_TEMPLATES = 0x16,
    GET_TEMPLATES_RESPONSE = 0x17,
    DATA = 0x20,
    DATA_ACK = 0x21,
    ERROR = 0x23
};

/*****
Structure of messages used in transport protocol:
*****/

```

```
struct Connect {
    struct IPDRStreamingHeader header;
    int collectorAddress; /* IPv4 IP address of the collector */
    short collectorPort; /* The transport protocol port number of the collector */
    int capabilities; /* an array of capability bits for capability negotiation*/
    int keepAliveInterval; /* the maximum delay between some indication from remote */
};

struct ConnectResponse {
    struct IPDRStreamingHeader header;
    int capabilities; /* an array of capability bits for capability negotiation*/
    int keepAliveInterval; /* the maximum delay between some indication from remote */
};

struct Disconnect {
    struct IPDRStreamingHeader header;
};

struct Error {
    struct IPDRStreamingHeader header;
    int timeStamp; /* time of error (in seconds from epoch) */
    short errorCode; /* values of 0-255 are reserved for standard */
                    /* reason codes. Values > 255 may be used for */
                    /* vendor specific codes. */
                    /* 0 = keepalive expired */
                    /* 1 = message invalid for capabilities */
                    /* 2 = message invalid for state */
                    /* 3 = message decode error */
                    /* 4 = process terminating */
    UTF8String description;
};

struct FlowStart {
    struct IPDRStreamingHeader header;
};

struct SessionStart {
    struct IPDRStreamingHeader header;
    int exporterBootTime; /* boot time of exporter */
    long firstRecordSequenceNumber; /* first sequence number to be expected */
    long droppedRecordCount /* number of records dropped in gap situations */
    boolean primary; /* indication that the collector is the primary */
    int ackTimeInterval; /* the maximum time between acks from collector */
    int ackSequenceInterval; /* the maximum number of unacknowledged records */
    char documentId[16]; /* the UUID associated with the info being sent */
                    /* in this session */
};

struct FlowStop {
    struct IPDRStreamingHeader header;
    short reasonCode; /* values of 0-255 are reserved for standard */
                    /* reason codes. Values > 255 may be used for */
                    /* vendor specific codes. */
                    /* 0 = normal process termination */
                    /* 1 = termination due to process error */
    UTF8String reasonInfo;
};

struct SessionStop {
    struct IPDRStreamingHeader header;
    short reasonCode; /* values of 0-255 are reserved for standard */
                    /* reason codes. Values > 255 may be used for */
                    /* vendor specific codes. */
                    /* 0 = end of data for session */
                    /* 1 = handing off to higher priority coll */
};
```

```

                                        /* 2 = exporter terminating */
                                        /* 3 = congestion detected */

    UTF8String reasonInfo;
};

struct TemplateData {
    struct IPDRStreamingHeader header;
    char configId; /* Identifies context of template definitions.*/
                  /* Changes in template should use a different */
                  /* configId (0 if unused) */
    char flags; /* Unused and reserved */
    TemplateBlock templates<>; /* Definitions of templates supported */
};

struct ModifyTemplate {
    struct IPDRStreamingHeader header;
    char configId; /* Identifies context of template definitions.*/
                  /* Changes in template should use a different */
                  /* configId (0 if unused) */
    char flags; /* Unused and reserved */
    TemplateBlock changeTemplates<>; /* Definitions of templates */
};

struct ModifyTemplateResponse {
    struct IPDRStreamingHeader header;
    char configId; /* Identifies context of template definitions.*/
                  /* Changes in template should use a different */
                  /* configId (0 if unused) */
    char flags; /* Unused and reserved */
    TemplateBlock resultTemplates<>; /* Definitions of templates - final results */
};

struct FinishNegotiation {
    struct IPDRStreamingHeader header;
};

struct Data {
    struct IPDRStreamingHeader header;
    short templateId; /* a template id relative to the session defined */
                    /* in the header. The fields in this template were */
                    /* reported at the beginning of the session via */
                    /* TemplateData messages */
    char configId; /* (see above) */
    char flags; /* currently just duplicate flag */
    long sequenceNum; /* session relative sequence number of this record */
    opaque dataRecord<>; /* XDR encoded fields based defined by templateId */
};

struct DataAcknowledge {
    struct IPDRStreamingHeader header;
    char configId; /* (see above) */
    long sequenceNum; /* session relative sequence number of last record */
                    /* received in a series. */
};

struct GetSessions {
    struct IPDRStreamingHeader header;
    short requestId; /* numeric ID of request for sessions */
};

struct GetSessionsResponse {
    struct IPDRStreamingHeader header;
    short requestId; /* allows match up of response to request */
    UTF8String vendorId; /* string to identify vendor that created */
                        /* templates. "IPDR.org" if these are */
};

```

```

                                /* IPDR defined templates.          */
struct SessionBlock sessionBlocks <> /* description of supported sessions */
};

struct GetTemplates {
    struct IPDRStreamingHeader header;
    short requestID; /* numeric ID of request for templates */
};

struct GetTemplatesResponse {
    struct IPDRStreamingHeader header;
    short requestID; /* allows match up of response to request */
    TemplateBlock currentTemplates <> /* supported active templates */
};

struct KeepAlive {
    struct IPDRStreamingHeader header;
};

/*****
Data structures and enumerations used within messages:
*****/

struct FieldDescriptor{
    int typeId; /* ID of field type */
    int fieldId; /* unqualified field code that can be used
                /* as alternative accessor handles to fields
                UTF8String fieldName; /* Note that field names are to be qualified
                /* with the Namespace name, as an example:
                /* http://www.ipdr.org/namespace:movieID
                /* The namespace must match one of those
                /* targeted by the schema or schema imports
};

struct TemplateBlock{
    short templateId; /* ID of template - context within configID
                    /* Provides numeric identifier to
                    /* IPDR service specification for context of
                    /* session/config
    UTF8String schemaName; /* Reference to IPDR service specification
    UTF8String typeName; /* Reference to IPDR service specification
    FieldDescriptor fields<>; /* Fields in this template
};

struct SessionBlock {
    char sessionId;
    char reserved;
    UTF8String sessionName; /* Optional names and description for the session */
    UTF8String sessionDescription;
    int ackTimeInterval; /* the maximum time between acks for this session */
    int ackSequenceInterval; /* the maximum number of unacknowledged data items */
};

enum Capabilities {
    STRUCTURES = 0x01,
    MULTISESSION = 0x02,
    TEMPLATE_NEGOTIATION = 0x04
}

/*****
Messages and structures used in UDP version discovery and negotiation:
*****/

struct VersionRequest {
    int requesterAddress; /* IPv4 address of the version request initiator */
    int requesterBootTime; /* boot time of the version request initiator */
    char [4] msg; /* must be 'CRAN' for version 1 and 'IPDR' for version 2 */
}

```

```
struct VersionResponse {
    ProtocolInfo defaultProtocol;
    ProtocolInfo [] additionalProtocols;
}

struct ProtocolInfo {
    int transportType; /* TransportTypeId of the transport protocol */
    int protocolVersion; /* IPDR Streaming Protocol version supported over transport */
    short portNumber; /* Transport protocol port used */
    short reserved; /* Reserved/unused */
}

enum TransportTypeIds {
    TCP = 1,
    SCTP = 2
}
```