INTERNATIONAL STANDARD

ISO/IEC 4396-9

First edition
2023-12

Telecommunications and information exchange between systems — Recursive inter-network architecture —

Part 9: Error and flow control protocol

Télécommunications et échange d'information entre systèmes — Architecture récursive inter-réseaux —

Partie 9: Protocole de contrôle d'erreurs et de flux

Reference number
ISO/IEC 4396-9:2023(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6 *Telecommunications and information exchange between systems*.

A list of all parts in the ISO/IEC 4396 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document describes the Error and Flow Control Protocol (EFCP) specification. EFCP is the data transfer protocol of the Recursive InterNetwork Architecture (RINA).[1],[2],[3] It supports a permanent connection with all types of data transfer services.

RINA is a new network architecture based on the idea that networking is inter-process communication (IPC) and only IPC. RINA imposes the strict separation of mechanisms and policies as one of the main architectural features. EFCP assumes that other RINA components are in place and supplies other mechanisms such as addressing and flow allocation, which are outside of its scope.

EFCP[4] is based on the concept of timer-based reliable management of connections.[5],[6] In this way, EFCP operates with a minimum exchange of packets to manage connections and to keep protocol machines involved in a connection synchronised. EFCP uses direct control messages to preserve data from being lost, mis-sequenced or duplicated.

Telecommunications and information exchange between systems — Recursive inter-network architecture —

Part 9:
Error and flow control protocol

1 Scope

This document provides the Error and Flow Control Protocol (EFCP) specification. EFCP provides an inter-process communication (IPC) service to an application process, which can be a (N+1)-IPC process (IPCP), with the requested Quality of Service (QoS). One or more service data units (SDUs) are passed on the (N)-port-id to the (N)-DIF (distributed IPC facility) to be sent to the destination application process. Protocol data units (PDUs) transferred by the (N)-DIF are delivered to the (N)-port-id for the using Application Process. This document describes the placement of EFCP within RINA, the components EFCP consists of, and the mechanisms and policies that are involved in EFCP's work, and the timers and control mechanisms required to manage the connection.

EFCP comprises two logical components, the data transfer procedures (DTP), providing tightly bound mechanisms and the data transfer control procedures (DTCP), which provides loosely bound mechanisms.

This document provides:

— the service definition;

— an overview of EFCP;

— a description of the placement of EFCP within recursive internetwork architecture (RINA);

— the common elements of data transfer protocol (DTP) and data transfer control protocol (DTCP);

— DTP structure and functions;

— DTCP structure and functions;

— an informative list of all policies in EFCP.en

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 4396-1, *Telecommunications and information exchange between systems — Recursive Inter-Network Architecture — Part 1: RINA Reference Model*

ISO/IEC 4396-7, *Telecommunications and information exchange between systems — Recursive Inter-Network Architecture — Part 7: RINA Flow Allocator*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 4396-1, ISO/IEC 4396-7 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
data transfer state vector
DT-SV
vector that provides shared state information for the connection maintained by the data transfer protocol (DTP) and the data transfer control protocol (DTCP)

3.2
maximum PDU lifetime
MPL
maximum time a protocol data unit (PDU) sent by a member of a distributed inter process communication (IPC) facility (DIF) may exist in the DIF

3.3
data transfer application entity
DTAE
error and flow control protocol (EFCP)-protocol machine (PM) task managing the shared state with its peer

4 Overview of the EFCP

EFCP assumes the fundamental idea that reliability can be achieved only through the bounding of three times. These times are the maximum PDU lifetime (MPL), the maximum time for the delay in acknowledging a PDU (A-time), and the maximum time to exhaust retransmitting a PDU (R-time). If these times are bounded by the quantities MPL A (maximum delay to send an acknowledgement) and R (maximum time before giving up retransmitting a PDU), then reliability can be guaranteed. Using these bounds, two important times are set up to reliably manage the data transfer, the sender and receiver inactivity times. The time Δt is calculated as the sum of Maximum PDU Lifetime, MPL; Maximum Time to before Sending an Ack, A; and the Maximum Time to Exhaust Retries, R.

The upper bound for the receiver's inactivity timer is 2 Δt and 3 Δt for the sender's inactivity time.

Every flow instantiated within a DIF triggers the creation of an instance of EFCP and its associated state vector.[3] For security purposes, all EFCP connections have flow control, although retransmission control may not be required. The EFCP is able to provide other services, such as reliable transmission with QoS.

DTP uses a single PDU that carries addresses, a connection-id, a sequence number, and various flags to signal congestion, and other conditions that are relevant to the operation of the protocol.

While there are four PDU types defined for DTCP, these are supported by only three PDU formats: Selective Ack/Nack/Flow, Ack/Flow, and Rendezvous/Flow. Each of these carries addresses, a connection-id, a sequence number, and retransmission control and/or flow control information, etc. The opcodes indicate which fields in the PDU are valid. Not only is Ack and Flow Control information sent to update the sender, but the receiver's left window edge is sent with Credit as a check on the state. All connections are flow controlled to provide a countermeasure against a denial-of-service attack. While DTCP controls the flow of DTP PDUs, DTCP never has cause to inspect their contents. In essence, DTP writes to the state vector and DTCP reads the DTP state from it.

EFCP is based on concepts first described by Watson in,[5] and adapted as part of the RINA as the mechanism to send and receive variable-length blocks of information.

EFCP separates mechanism and policy. There is a default action associated with each policy. The policy can take additional action beyond the default or replace the default action. A policy may only reference the state variables defined by this document and the parameters specified by the management objects to instrument EFCP. Policies may define both local and persistent variables. Local variables are assumed to be new instantiations on each call, while persistent variables retain their value between calls. The latter allows a policy to maintain state, if necessary. An optional parameter list may be present. The parameters used should be variables internal to the policy. The informative Annex A provides a summary of the EFCP policies.

5 Role of EFCP within the IPC process

5.1 General

The EFCP is responsible for carrying out the data transfer and data transfer control tasks. In order to do so, each EFCP instance exchanges protocol data units (PDUs) with an EFCP instance in another inter process communication (IPC) process (IPCP) (although this other instance can also be located in the same IPCP in a degenerate case).

The IPCP provides IPC services to one or more application processes (which in turn may be a higher-layer IPCP). Each instantiation of an IPC data-transfer-service is called a flow and is identified by a port-id. Application processes can write or read service data units (SDUs) from the port, thus communicating with one or more remote application process instances. When an application process writes an SDU to a port, first the delimiting module associated to the port may fragment the SDU or concatenate it with other SDUs (or SDU fragments), producing one or more user data fields (UDFs). UDFs are delivered to the EFCP instance associated to the port, which will add a header to the UDF [the protocol control information (PCI)], producing a PDU that is delivered to multiplexing task (MT). The MT multiplexes PDUs from different EFCP instances into one or more N-1 ports provided by one or more underlying IPCPs.

In the “read path”, the MT will deliver to the EFCP module the PDUs addressed to the IPCP. EFCP then makes available each PDU to the relevant EFCP instance, which will process the PCI, remove it and deliver the UDF to the delimiting module to obtain the original SDUs. SDUs will be buffered until they are read by the application process owning the port associated to the EFCP instance.

The lifecycle of EFCP instances is managed by a layer management component of the IPCP called flow allocator (FA). When an application process requests the creation of a flow to the IPCP, the FA is the component that deals with the request. Part of the responsibilities of the FA are to create the EFCP instance (with the appropriate mix of policies) and bind it to the port providing the flow.

The FA monitors the EFCP connection and, at some point, it can decide that it is time to create a new EFCP instance, bind it to the flow and remove the old one (for example, to prevent sequence number rollover to happen). This capability is enabled by the fact that ports and EFCP instances are decoupled: only ports are exposed to application processes outside of the layer. At any moment in time there can only be one active EFCP instance bound to a specific port.

When the application process using the flow [or the operating system (OS) in its behalf, for example if the application process has crashed] decides that it no longer needs it, it invokes the deallocate flow primitive. This operation causes the FA to destroy the EFCP instance and release the port that was associated to the flow.

5.2 Description of the Data Transfer Protocol (DTP) Task

A DTP Task is always created when a flow is allocated. DTP performs all mechanisms associated with the Data Transfer PDU, e.g. sequencing, invokes SDU Protections and Delimiting to do fragmentation/reassembly.

While the DTCP-State can be discarded during long periods of no traffic, the DTP-State Vector cannot. The DTP-State is only discarded after an explicit release by the AP or by the System (if the AP crashes), i.e. the port-ids are released. The DIF shall have procedures to ensure that when traffic resumes the CEP-id is associated with the correct port-id.

The indirection introduced by the distinction between the port-id and CEP-id avoids the requirement of a separate connection for integrity mechanisms (encryption). To prevent replay, insertion, or deletion, a sequence number is introduced but it can't be allowed to wrap. When using encryption in a DIF, as a flow sequence number nears its limit, a new connection is allocated with different CEP-ids and bound to the port-id. This will be monitored by the flow allocator instance (FAI) that created the flow. The FAI will create the new connections and create the bindings with the port-id. There will be a period when PDUs are accepted from both connections. After 2*Δt, the earlier connection will be allowed to expire. For flows without encryption or flows that do not send sufficient data for sequence numbers to roll over, the port-id and the CEP-id may be the same.

Depending on the policies in force on the flow, the Delimiting will provide fragmentation/reassembly functions (breaking SDUs into more than one PDU) and concatenation/separation functions (combining multiple SDUs into one PDU). DTP creates PDUs and processes them upon arrival and imposes sequencing. DTCP provides either retransmission or flow control or both. If there is flow control and the flow control window is closed, then PDUs are put on the closed window queue. There is a policy to monitor the length of this queue, and sending PDUs comes under the control of the DTCP. A Transmission Control Policy may override and queue PDUs to slow the data rate for other reasons. If flow control is not present or the window is open, then the PDUs are posted to the Multiplexing Task. If retransmission control is being used, a copy of the PDU is placed on the retransmission queue and a timer started.

When PDUs arrive for DTP, SDU Protection determines if they can be processed, and if so, they are ordered. If Retransmission Control is in use, i.e. DTCP is present and a Data Transfer PDU is received, if the Sequence Number of the PDU is less or equal than the last sequence number acknowledged, then this PDU is a duplicate and is discarded. Otherwise, the PDU is put on the PDU Reassembly Queue and Delimiting is invoked to create SDUs and deliver them to the using application process.

Since EFCP may allow gaps in the data stream, the concept of the left window edge exists regardless of whether DTCP is present. With DTCP present, it is assumed that all PDUs with a sequence number smaller than or equal to the value of the receiver left window edge (RcvLeftWindowEdge) have been acknowledged. This means that there will be no retransmissions of PDUs with sequence numbers less than or equal to the RcvLeftWindowEdge. All gaps have been resolved one way or another. When DTCP is not present, this implies that all PDUs with sequence numbers smaller than or equal to the value of the RcvLeftWindowEdge have been processed for delivery to the process above.

If out of order delivery is allowed, then it shall be in terms of SDUs. As indicated by the previous discussion of the use of Left Window Edge in DTP, this makes the most sense when there is one SDU per PDU. If not, the PDUs have to be effectively ordered to reassemble them. Randomly choosing initial sequence numbers for sequences of PDUs that constitute an SDU is not recommended since this can adversely affect the transmission rate.

This has implications for whether or not order is imposed on the PDUs. Regardless of whether there is an ordering, a PDU identifier of some sort is required. Most implementations assign these sequentially. Random assignment is not recommended for the following reasons:

a) much less of the sequence number space can be used within an MPL;

b) the assumption shall be made that an SDU is no larger than the Maximum PDU size.

If b) does not hold, then SDUs shall be reassembled and the order of the PDUs shall be known at least within a sequence. If sequence numbers are assigned randomly and the number of PDUs sent within an MPL is a sizable fraction total sequence number space of PDUs, the chances of the same sequence number occurring with an MPL is fairly high. It can be necessary to limit transmission to 25 % or even less of the sequence number space, whereas with sequential assignment the full sequence number space can be used.

Transmission Control provides the means to control the sending or not sending of PDUs beyond what is indicated by the feedback from the apposite DTCP instance. For example, there can be conditions under which the destination indicates PDUs may be sent, but other conditions, e.g. congestion, that indicate that PDUs should not be sent or fewer sent or sent at a slower rate. Transmission Control also responds to the detection of Lost Control PDUs.

5.3 DTP user service API definition

5.3.1 Read API or read immediate invoked

Parameters: Port-id, pointer to a buffer, array of buffers, or list of buffers, a length in bytes.

This event occurs when the user of the flow does a Read or Read Immediate. Several options affect the Read, such as Partial Delivery and MaxSDUSize. A Read will read available SDUs, but not those with the A-timer running on them, i.e. there are gaps that may be filled. Read Immediate overrides the constraints and reads as many SDUs as requested. Partial Delivery indicates that SDUs can be delivered into more than one buffer or with multiple Reads. Incomplete Delivery indicates that SDUs with gaps can be delivered, i.e. missing PDUs. In this latter case, the API should indicate that a gap in the SDU is being delivered (usually by indicating an error with the Read). This error should differ from a read of a zero-length SDU.

5.3.2 Read/Read immediate action

```
If Read Then

    Copy as many SDUs as requested in the available buffers taking into account whether
    partial delivery is allowed.

    Updating RcvLeftWindowEdge and RcvRightWindowEdge has already occurred.

Fi

If Read Immediate Then

    Copy as many SDUs as requested from the list of available SDUs taking into account
    whether partial delivery is allowed. If that does not fulfil the request, go to the
    PDUReassemblyQueue and invoke Delimiting on the PDUs there to fulfil the request.
    This will require deactivating A-Timers and updating the RcvLeftWindowEdge and
    RcvRightWindowEdge and sending the appropriate Ack/Flow Control PDU.

Fi
```

5.3.3 Write action

Parameters: Port-id, pointer to a buffer, array of buffers, or list of buffers, a length in bytes.

This API call delivers SDUs to the delimiting module, which delimits SDUs into some number of User-data fields for the DTP task. Some OSs may assign semantics to reading a zero-length SDU. This implies that the API shall allow a zero-length SDU and not “optimize” it out of existence.

5.4 Description of the DTCP task

5.4.1 General

DTCP is instantiated when the DTP is created. The DT-SV can be discarded during long periods of no traffic. However, the state associated with the FAI for this data transfer application entity (DTAE)-instance shall be explicitly released.

DTCP processing performs the more complex policies for flow control, transmission control and re-transmission control. DTCP controls whether DTP PDUs can be posted to the RMT. It may do this based on feedback information from the receiver or based on its own estimators. PDUs placed in its queues for flow control and retransmission become DTCP’s responsibility to send at the appropriate time.

DTCP PM requires a degree of synchronization with its apposite to avoid deadlocks and to ensure progress. DTCP uses the sequence numbers in the DTP PDUs and bounding three timers to ensure synchronization. Data Transfer and the feedback mechanisms are used to establish this shared state. The three timers shall be bounded and are:

— MPL;

— maximum time to ack, A;

— the time to attempt the maximum number of retries to deliver a PDU,R.

5.4.2 Retransmission control

Retransmission control requires sequencing in the associated DTP instance. Based on the sequence numbers seen by the receiver, an ack PDU is sent to the sender. The Ack/Nack class of PDUs supports what has been traditionally called positive (Ack) and negative (Nack) acknowledgements.

There are two modes in which the retransmission mechanism can be used: positive ack, or negative ack. This mechanism applies to all PDUs generated for this flow. The sender of Data Transfer PDUs will maintain a retransmission queue, i.e. each sender maintains such a queue. When a Data Transfer PDU is sent by DTP, a copy is placed on this retransmission queue and a timer is started. The timer has a value TR (~RTT + A + ε, where RTT is round trip time and ε is the local processing time and allows for variation in time the Ack arrives, generally an estimate of standard deviation), and is generally a function of round-trip time. Calculation of the timer value is carried out by the RTT estimator policy. If the timer expires before an AckPDU is received, all PDUs on the retransmission queue with sequence numbers less than or equal to the sequence number of the PDU associated with this time are re-sent.

If an ack is received, then the sender will delete all PDUs from the retransmission queue with sequence Numbers less than or equal to the contents of the Ack/Nack field and discontinue any retransmission timers associated with these PDUs. If a Nack is received, the sender will retransmit all PDUs on the retransmission queue with sequence numbers greater than or equal to the contents of the Ack/Nack field.

If a selective ack is received, the PDUs designated by the ranges of sequence numbers indicated by the Ack/Nack List are deleted from the Retransmission Queue and any timers associated with these PDUs are discontinued.

If a selective nack is received, the PDUs with sequence numbers from those ranges are retransmitted. Since it is not possible to know whether a retransmitted PDU arrives, a negative ack can never cause PDUs to be removed from the retransmission queue.

To ack or nack a single PDU, the entry in the selective ack/Nack List should have starting and ending ranges of the same Sequence Number.

5.4.3 Flow control

Flow control policies may utilize both a rate-based strategy or the classic sliding window mechanism. Both may be used with or without retransmission control. With either approach, the receiver of PDUs provides an indication to the sender when it can send. With the window mechanism, the sender is given the Left Window Edge and Credit from which the RightWindowEdge can be calculated. The sender can send PDUs with sequence numbers less than the RightWindowEdge; whereas with the rate mechanism the sender is told at what rate it may emit PDUs. Flow control is always expressed as the number of PDUs or as the number of PDUs per unit time, or more precisely, how many PDUS may be sent in a unit of time.

Each approach has its advantages depending on the traffic characteristics of the connection. By having two independent methods, a policy may actually use both, allowing one to dominate the other, except under extreme conditions when the other may exert an effect. For example, one can allow rate-based flow control to dominate by keeping the right window edge sufficiently high that the receiving transport protocol machine is delivering data at about the same rate. However, the credit level is maintained at something close to the maximum buffer allocation for this connection. If the protocol machine should get behind and begin to exceed its maximum buffer allocation, the credit-based flow control would dominate and temporarily stop the flow on the connection, until the buffer pool was restored to an appropriate level.

A mechanism required by flow control shall be considered to deal with the case of a zero-length window and/or a zero rate: the rendezvous mechanism. When the sender's state indicates that there is data to send, and a zero window exists, the sender wants to be informed when the window gets open. This is achieved through a special type of control PDU, the rendezvous PDU, which is sent by the sender to the receiver. If there is retransmission control, another condition that has to be met for the rendezvous PDU to be sent is that all data already transmitted have been Acked.

The rendezvous PDU is protected against duplication because it uses a sequence number from the Control PDU sequence number space. Furthermore, it is a reliable PDU, because it is retransmitted until acknowledged, thus protected against loss. The acknowledgement can be either a normal flow control PDU opening the window or a ControlAck PDU to inform the sender that the receiver will keep the window closed.

When the sender receives either one, it will stop sending the rendezvous PDU. If the ack is lost, the Rendezvous PDU will be retransmitted and the receiver will know that it has to retransmit the ack.

Once the receiver opens the window, it sends a reliable Ack/FC PDU. The acknowledgment of this reliable Ack/FC PDU consists of a DT PDU with the data run flag (DRF) set, signalling the start of a new data run. This reliable Ack PDU is retransmitted until the receiver receives the DT PDU.

In case that at some point the exchange of reliable PDUs is interrupted, with all of them being lost, eventually the maximum number of retransmissions will be achieved. At this point it will be for the inactivity timers to restart the communication.

6 Common elements of the DTP and DTCP tasks

6.1 Abstract and encoding rules

EFCP abstract syntax defines the semantics and ordering of the fields of the different EFCP PDUs. This syntax is defined primarily in terms of constants such as AddrLength or Port-id-length. These are defined at "compile-time" for the DIF or class of DIFs, providing a concrete syntax for a specific EFCP variant. In most cases, the length of the fields has very little effect on the design of the protocol beyond the modulus of the arithmetic. Hence, there is no reason to define separate protocols for every field length combination. The specification can be fully defined by utilizing the abstract syntax and defining a set of concrete encoding rules, i.e. specific syntaxes for different operational environments. Choosing lengths that are not a multiple of 8 bits will likely have a performance penalty.

6.2 DIF-wide data

6.2.1 General

All IPCPs working within a particular DIF have to agree on certain things.

6.2.2 Constants

QoSIdLength: Unsigned Integer – The length of a QoS-id field in bits.

PortIdLength: Unsigned Integer – The length of a Port-id field in bits.

CEPIdLength: Unsigned Integer – The length of a CEP-id field in bits.

SequenceNumberLength: Unsigned Integer – The length of a SequenceNumber field in bits.

AddressLength: Unsigned Integer – The length of an Address field in bits.

LengthLength: Unsigned Integer – The length of a Length field in bits.

DIF.Integrity: Boolean – This is True if PDUs in this DIF have data corruption detection, time-to-live, and/or encryption. Since headers are encrypted, not just user data, if any flow uses encryption, all flows within the same DIF shall do so and the same encryption algorithm shall be used for every PDU. The flow that a particular PDU belongs to cannot be determined until it has been decrypted.

MPL: Unsigned integer – Maximum PDU Lifetime – Upper bound on in-flight PDUs.

6.2.3 Data structures

QoS-id: Unsigned(QoSIdLength)

CEP-id: Unsigned(CEPIdLength)

Port-id: Unsigned(PortIdLength)

Address: Unsigned(AddressLength)

Length: Unsigned(LengthLength)

SequenceNumber: Unsigned(SequenceNumberLength)Sequence numbers on a flow go from 0 to 2^n-1. They identify PDUs.

Connection-id: Struct

  QoS-id: QoS-id

  Destination-CEP-id: CEP-id

  Source-CEP-id: CEP-id

6.3 Data Transfer Application Entity (DTAE) constants

Max-SDU-Size: Unsigned. The maximum size of an SDU admitted by the DIF.

Max-PDU-Size: Unsigned. The maximum size of PDUs used within the DIF.

PCI-Length: Unsigned. The length of the EFCP header (PCI).

6.4 PDU format

The following fields are considered to be the Common PCI or just PCI. This PCI is present in all PDUs with different PDU-Type. Meaning of all these fields is the same for every PDU-Type.

User-Data is formatted according to PDU-Type.

Version: 8 Bits

Destination-Address: AddressLength

Source-Address: AddressLength

Connection-id: Struct

  QoS-id: QoSIdLength

  Destination-CEP-id: CEPIdLength

  Source-CEP-id: CEPIdLength

PDU-Type: 8 bits

Flags: 8 bits

PDU-Length: LengthLength

Sequence-Number: SequenceNumberLength

User-Data: Sequence

Version – An identifier indicating the version of the protocol. (seems prudent)

Destination-Address – A synonym for the Application-Process-Name designating an IPC-Process with scope limited to the DIF and a binding to the Destination Application Process.

Source-Address – A synonym for the Application-Process-Name designating the IPC-Process with scope limited to the DIF and a binding to the Destination Application Process.

Connection-id – A three-part identifier unambiguous within scope of two communicating IPC-Processes used to distinguish connections between them. A Connection-id consists of the following:

QoS-id – A DIF-assigned identifier only known within the DIF that stands for a particular QoS hypercube.

Destination-CEP-id – An identifier unambiguous within the system in which the destination-IPC-Process resides that identifies the binding between the IPC-Process and an Application-Entity-Instance of the Destination-Application-Process.

Source-CEP-id – An identifier unambiguous within the system in which the source-IPC-Process resides that identifies the binding between the IPC-Process and an Application-Entity-Instance of the Source-Application-Process.

PDU-Type – This field identifies the type of PDU. The encoding is as follows:

X'80' = Data Transfer PDU

X'Cx' = DTCP PDU

X'40' = Management PDUs

Flags – This field indicates conditions that affect the handling of the PDU. Flags should only indicate conditions that can change from one PDU to the next. Conditions that are invariant over the life of the connection should be established during allocation or by the action of management. Interpretation of flags depends on PDU-Type.

PDU-Length – The contents of this field is total length of the PDU in bytes.

Sequence-Number – The contents of this field is the sequence number of the PDU.

7 Detailed description of the DTP task

7.1 General

This specification is entirely logical; it is intended to indicate what needs to be done. It is indicative, not prescriptive of an implementation design, and to aid clarity, omits some details that can be reconstructed from the requirements of the message format, such as setting flags appropriately.

7.2 DTP State Vector — DTP-SV

This is the data associated with a single active flow.

DTPState: Struct

/* Constants */

Max-Flow-SDU-Size: Unsigned

Max-Flow-PDU-Size: Unsigned

SequenceNumberRollOverThreshold: Unsigned(SequenceNumberLength)

/* Variables: General */

DTCPpresent: Boolean

SetDRF: Boolean /* set when the next PDU sent should have DRF set */

PartialDelivery: Boolean /* Partial Delivery of SDUs is Allowed */

IncompleteDelivery: Boolean /* Delivery of Incomplete SDUs is Allowed */

RcvLeftWindowEdge: Unsigned(SequenceNumberLength)

MaxSequenceNumberRcvd: Unsigned(SequenceNumberLength)

SenderLeftWindowEdge: Unsigned (SequenceNumberLength)

NextSequenceNumberToSend: Unsigned(SequenceNumberLength)

LastSequenceNumberSent: Unsigned(SequenceNumberLength)

PDUReassemblyQueue: List of {PDUs, SeqNum}

User_DataQueue: List of {User-Data} /* User-data fields created by Delimiting */

Max-Flow-SDU-Size – The maximum size of SDUs allowed in the flow. Its value shall be less or equal to those defined for the DIF.

Max-Flow-PDU-Size – These parameters can set smaller values of the DTAE constants for a given flow. These shall have values less than or equal to those defined for the DIF.

SequenceNumberRollOverThreshold – This can/should be in the Connection. It is the value at which the EFCP notifies the FAI that a new connection will soon be needed to avoid sequence number rollover to support data integrity.

SequenceNumberLength – The length of sequence numbers needs to be roughly 2^n greater than times the quantity of the sum of MPL, A and R time T, where MPL is maximum PDU lifetime, R is maximum time for Retries, A is the maximum time before an Ack is sent, and T is the data rate in the units sequence numbers are incremented.

MaxSequenceNumberRcvd – The largest sequence number of a PDU received. There may be gaps between this sequence number and the RcvLeftWindowEdge.

NextSequenceNumberToSend – The sequence number of the next sequence number to be assigned to a PDU being sent on this connection.

Partial Delivery – This Boolean indicates that SDUs do not have to be delivered all at once, but can be delivered incrementally.

IncompleteDelivery – This Boolean indicates that SDUs with missing fragments can be delivered.

RcvLeftWindowEdge – This variable contains the sequence number of the last in-order received Data Transfer PDU. (In most common case it will be equal to MaxSeqNumRcvd).

SetDRF – This Boolean indicates that the next PDU sent should have the DRF set.

LastSequenceNumberSent – The sequence number of last sent PDU.

7.3 Data transfer PDU

The Data Transfer PDU consists of Common PCI and User-Data field. Fields that have special meaning within a Data Transfer PDU are described below.

PDU-Type – Data Transfer PDUs have PDU-Type equal to X'80'

Flags–A PDU with multiple SDUs shall have an integral number of complete SDUs.

'80'X – indicates Data Run Flag (DRF).

'01'X – ECN (Explicit Congestion Notification).

The Data Run Flag indicates that all PDUs sent previously on this connection have been acknowledged.

The ECN Flag indicates that this PDU was relayed by an IPCP experiencing congestion.

UserData – This field contains one or more octets that are uninterpreted by the EFCP. The Delimiting function creates this field containing SDU-Fragments and/or one or more complete-SDUs up to the (MaxPDUSize-PCISize). The Delimiting of the User-Data Field is defined by the Delimiting Policy.

7.4 Common procedures

Common Procedure SetDRFinPDU(override);

Boolean override;

```
If SetDRF or override Then
   SetDRFinPDU := True;
   SetDRF:= False;
Else
   SetDRFinPDU := False;
Fi
```

7.5 API events

7.5.1 General

Events:

User calls

Write: SDU delivered from N-port-id

Arriving PDUs

Receive: PDU delivered from RMT

Timeouts

Receiver Inactivity Timer timeout

Sender Inactivity Timer timeout

A-Timer timeout

7.5.2 Event — SDU delivered from (N)-port-id

Stop the SenderInactivityTimer, if it is running; there is traffic to send. When a Write API is invoked, Delimiting will create one or more User-Data fields from the Write. EFCP then encapsulates these user-data fields in PDUs for sending.

If Flow Control is present, there are two kinds: sliding window and rate-based. If sliding window is in use then, we check to see how many PDUs can be sent. A TransmissionControlPolicy is provided to allow DTP to not send even if the window is open, for example, because the ECN bit has been set on incoming PDUs. If there are more PDUs to send than allowed by the sliding window or the TransmissionControlPolicy, the PDUs are put on the ClosedWindowQueue and a flag is set indicating the window is closed. A policy is provided for handling the Closed Window Queue getting too long.

With rate-based flow control, the sender is allowed to send a burst of X PDUs in a given time period. If all the PDUs that can be sent in this time period have been sent, the remaining PDUs are placed on the ClosedWindowQueue to be sent in the next time period. Policies are provided to allow rate-based to send fewer PDUs in the allotted time, and to allow rate-based flow control to exceed the rate (hopefully for a short time).

A Policy is also provided to reconcile conflicts if window-based and rate-based are being used at the same time. Once it is known how many PDUs can be posted, then they are posted to the Multiplexing Task and to the Retransmission Queue with the retransmission timers set. Finally, the SenderInactivityTimer is set in case these can be the last PDUs sent for a while.

This procedure can be done one PDU at a time, but it means checking whether Flow Control and Retransmission Control are in use for every PDU.

```
DTPState State;

/* Iterate over the User-Data fields and generate PDUs. */
/* PDU sequence numbers start at State.NextSequenceToSend */
List GeneratedPDUs := Empty
PDU CurrentPDU := Null
Stop any SenderInactivityTimers running, there is data to send.

For each User-Data in {User-Data} /* for all User-data fields produced by delimiting */
      Create complete PDUs:
      Assign static fields (addresses, CEP-ids, etc.)
      PDU.DRF := SetDRFinPDU(false);
      Assigning PDU.SeqNum := State.NexSeqNumToSend;
      State.NextSeqNumToSend := * + 1;
      Invoke SDU Protection
      Add PDU to GeneratedPDUs
 Rof

/* State.NextSequenceToSend has been incremented by total number of
PDUs on Generated PDUs */

/* Iterate over generated PDUs and decide if we can send them. */
If DTCPpresent Then
      List PostablePDUs := Empty
      If FlowControl Then
      For each PDU in GeneratedPDUs
            If Window-based Flow Control Then
                  If PDU.SequenceNumber <= State.SndRightWindowEdge Then
                        /* The Window is Open. */
                        Policy TransmissionControl with Default:
                        If DTCPState.ECN == False then:
                              Add PDU to PostablePDUs as Window Allows,
                              closing it if necessary
                              And Set the ClosedWindow flag appropriately.
                        Fi
                        End Default;
                  Else
                        /* The Window is Closed */
                        ClosedWindow:=True;
                        If Rendezvous-at-the-sender==False then:
                              Rendezvous-at-the-sender:=True;
                              Start Rendezvous Timer;
                              Send Rendezvous PDU;
                        Fi
                        If ClosedWindowQueueLength <
                  MaxClosedWindowQueueLength-1 Then
                              Put the PDU on the ClosedWindowQueue
                        Else
                              Policy SndFlowControlOverrun with Default:
```



```
                              Put the PDU on the ClosedWindowQueue.
                              Block further Write API calls on this port-id
                              End Default;
                        Fi
                  Fi
            Fi
            If Rate-based Flow Control Then
                  If SendingRate == 0 And Rendezvous-at-the-sender==False
            Then
                        Rendezvous-at-the-sender:=True;
                        Start Rendezvous Timer;
                        Send Rendezvous PDU;
                  Fi
                  If PDUsSentinTimeUnit < SendingRate Then
                        Policy NoRate-SlowDown with Default:
                  If DTCPState.ECN = False then
                        Add PDUs to PostablePDUs
                        PDUsSentinTimeUnit := * +1
                  Fi
                  End Default;
            Else
                  /* Exceeding Sending Rate */
                  Policy NoOverrideDefaultPeak with Default:
                  RateFulfilled := True;
                  If ClosedWindowQueueLength <
                  MaxClosedWindowQueueLength-1 Then
                        Put the PDU on the ClosedWindowQueue
                  Fi
                  End Default;
            Fi
       Fi
       If (Window-based AND Rate-based) AND (WindowClosed XOR
       RateFulfilled) Then
            ReconcileFlowConflict with Default:
            End Default;
       Fi
   ROF
   Else
       Add all PDUs from GeneratedPDUs to PostablePDUs
   Fi

   /* Iterate over PostablePDUs and give them to the RMT */
   If RetransmissionControl Then
      For each PDU in PostablePDUs
            /* Put a copy of each PDU in the RetransmissionQ */
            Timer := start RetransmissionTimer for this PDU
            add {PDU,Timer} to DTCPState.RetransmissionQ
            PostPDUtoRMT( dest-addr, QoS-id, PDU )
      ROF
   Else /* No Retransmission Control, but Flow Control */
      Post all PostablePDUs to RMT
   Fi
Else /* DTCP not Present */
   Post all PDUs GeneratedPDUs to the RMT
Fi
Start new SenderInactivityTimer;
```

7.5.3 Event — PDU delivered from RMT

Sending Transfer PDUs is straightforward. Receiving them is less so. There are five basic cases to cover and sub-cases within them:

a) **RcvInactivityTimer == 0 AND DRF is set in the PDU**: This is either the first PDU on the flow, or starting a new session. When received, if DTCP is present, it indicates all previously sent DataTransferPDUs have been successfully acknowledged. There should not be any PDU on ReassemblyQueue.. If DTCP is not present, the DRF signifies the end of a set of PDUs, delimiting should be invoked.**RcvInactivityTimer == 0 /* AND DRF is not set */:** This is missequenced PDU. Policy may choose to put this PDU on reassemblyQ and wait for PDU with DRF to arrive.

b) **PDU SequenceNumber <=RcvLeftWindowEdge OR PDU.SequenceNumber = MaxSequenceNumberRcvd**. This is a true duplicate. There are no gaps here and/or everything has been acked. The PDU is discarded. If DTCP is present, then send an Ack with current left and right window edge to just ensure sender is on the same page. No further action if DTCP is not present.

c) **RcvLeftWindowEdge < PDU.SequenceNumber < MaxSequenceNumberRcvd**. This is in the area where the PDU can be a duplicate or contribute to filling a gap. If it is a duplicate, then it is handled the same as b). Otherwise, the PDU is put in the ReassemblyQ in Sequence Number Order. If DTCP is present, SVUpdate(PDU.SequenceNumber) is invoked to update the window edges, and delimiting is invoked to create as many SDUs as possible. This PDU may have filled a gap and moved the RcvLeftWindowEdge to the right. If DTCP is not present, then there are two choices: If data is being delivered immediately, then Delimiting is invoked. Otherwise, we wait for the A-Timer to expire.

d) **PDU.SequenceNumber equals MaxSequenceNumberRcvd+1.** The PDU is in order! MaxSequenceNumberRcvd is incremented. If DTCP is present Then SVUpdate(PDU. SequenceNumber is invoked and Delimiting is invoked. If DTCP is not present there are two choices: If there is immediate delivery, then Delimiting is called to create SDUs. immediate is just an A-Timer value of zeroand the rcvLeftWindowEdge is incremented accordingly. Otherwise, the A-Timer is started. There is some time to wait to see if others arrive. When it goes off, the rcvLeftWindowEdge is increased.

e) **PDU.SequenceNumber > MaxSequenceNumberRcvd+1.** This means there is a new gap. If DTCP is present, SVUpdate(PDU.SequenceNumber) is invoked, followed by Delimiting being invoked. If DTCP is not present there are two choices: If there is immediate delivery, then delimiting is called to create SDUs. immediate is just an A-Timer value of zero. Otherwise, the A-Timer is started. There is some time to wait to see if others arrive.

The following is an example pseudo-code to show the main steps of the processing of a PDU delivered from the MT.

```
If State = Null Then
   /* The policy takes charge of the PDU and shall drop or queue it */
PolicyUnknownFlow(PDU) Then
      Discard the PDU
End Default;
   Return
Fi
Stop the RcvrInactivityTimer;

If DTCPpresent then
      If Window-based control then:
            If PDU.SequenceNumber > RcvRightWindowEdge then
                  Policy RcvFlowControlOverrun with default:
                        Discard PDU
                        Send Ack/FlowControl PDU
                        Start RcvrInactivityTimer
                        Return
                  End default
            Fi
      Else If Rate-Based control then:
            If PDUsRcvdinTimeUnit + 1 > RcvRate then;
                  Policy RcvFlowControlOverrun with default:
                        Discard PDU
                        Send Ack/FlowControl PDU
                        Start RcvrInactivityTimer
                        Return
                  End default
            Fi
      Fi
Fi

If RcvInactivityTimer = 0 AND PDU.DRF = True Then /* Case 1a) DRF is set. Either first PDU
or new run */

   If DTCPpresent Then
      If ReassemblyQ not empty then
```

```
            Report an error to the upper flow (delimiting does not work properly)
      Fi
   Else
     Put the PDU on the ReassemblyQ (its at the head)
     If DTCPpresent Then /* update rexmsn control */
       Invoke SVUpdate(PDU.SequenceNumber)
  Else
       RcvLeftWindowEdge := PDU.SequenceNumber
     Fi;
     Invoke Delimting;
Else If RcvInactivityTimer = 0
       <policyName> with default:
            /* This policy allows the implementation to store misequenced DT-PDUs and used
it upon reception of DT-PDU with DRF set. */
            Drop this PDU
       End <policyName>
Else /* Not the start of a run */

If PDU.SequenceNumber <= RcvLeftWindowEdge OR PDU.SequenceNumber ==
    MaxSequenceNumberRcvd Then /* Case 2) A Real Duplicate. */
Discard PDU and increment counter of dropped duplicates PDU
Send an Ack/Flow Control PDU with current window values
  Return
Fi /* Not a true Duplicate. (Might be a duplicate amongst the gaps) */
If RcvLeftWindowEdge < PDU.SequenceNumber <MaxSequenceNumberRcvdThen
    If a Duplicate among the gaps Then /* Case 3) Duplicates among gaps. */
    Discard PDU and increment counter of dropped duplicates
    Send an Ack/Flow Control PDU with current window values
         Return
    Else /* Case 3) This goes in a gap */
      Put at least the User-Data of the PDU with its Sequence Number on
         ReassemblyQ in Sequence Number order
      SVUpdate(MaxSequenceNumberRcvd) /* Update Left Edge, etc. */\
      /* No A-Timer necessary, already running */
      Start RcvrInactivityTimer
    Fi
    Invoke Delimiting
    Return
    Fi
Fi
/* Case 4) This is in order */
If PDU.SequenceNumber = MaxSequenceNumberRcvd+1 Then
    Put PDU on ReassemblyQ
    Increment MaxSequenceNumberRcvd;
    If DTCPpresent Then
    SVUpdate(MaxSequenceNumberRcvd) /* Update Left Edge, etc. */\
    Else
    RcvLeftWindowEdge := MaxSequenceNumberRcvd;
        Start A-Timer
    Fi
  Invoke Delimiting /* Create as many whole SDUs as possible */
Else /* Case 5) it is out of order */
 If PDU.SequenceNumber > MaxSequenceNumberRcvd+1 Then
   MaxSequenceNumberRcvd := PDU.SequenceNumber
   Put PDU on ReassemblyQ
   If DTCPpresent Then
   SVUpdate(MaxSequenceNumberRcvd) /* Update Left Edge, etc. */
   Else
      Start A-Timer
   Fi

   Invoke Delimiting
  Fi
  Start RcvrInactivityTimer(PDU.SequenceNumber) /* Backstop timer */
Fi

/* If we are encrypting, we can't let PDU sequence numbers roll over */
If DIF.Integrity and PDU.SeqNum > SequenceNumberRollOverThreshhold Then
   /* Security requires a new flow */
   RequestFAICreateNewConnection( PDU.FlowID )
 Fi
```

```
Fi
```

7.5.4 Event — Receiver inactivity timer timeout

When Set

This timer is set to detect long periods of inactivity on the flow. This Timer detects that there has been no traffic for a long time. It should be set upon completion of the processing of the last PDU received and should generally be set to $2\Delta t$. This is the ultimate upper bound on out of order messages used when Retransmission Control is in use.

Action Upon Expiration

When this timer expires, it indicates that there is a gap greater than allowed on this connection and retransmissions have not been successful. The receiver's state can be safely reinitialised, i.e. state variables can be initialised to the default values and PDUs in the different queues discarded.

```
Policy RcvrInactivityTimer with Default:

Next data PDU is expected to have the DRF set.

Reset receive variables (rcvLWE, rcvRWE, discard PDUs in the Sequencing/Reassembly queues,
etc).

End Default;
```


7.5.5 Event — Sender inactivity timer timeout

When Set

This timer is set to detect long periods of inactivity on the flow. This Timer detects that there has been no traffic for a long time or that retransmissions have been exhausted. It should generally be set to 3 times the sum of MPL, A, and R. This is the ultimate upper bound on out of order messages used when Retransmission Control is in use.

Action Upon Expiration

When this timer expires, it indicates that there is a gap greater than allowed on this connection and retransmissions have not been successful. SenderInactivityTimerPolicy determines what to do.

Policy SenderInactivityTimer with Default:

```
   SetDRF := True
   Policy InitialSequenceNumber with Default:
       NextSeqNumToSend := Random;
   End Default;
   If PDURetransmissionQueue OR ClosedWindowQueue NOT empty Then
    Discard any PDUs on the PDURetransmissionQueue and ClosedWindowQueue;
    Report error to upper flow;
   EndIF
   Reset sender variables (sndLWE, sndRWE, ).
End Default;
```

7.5.6 Event — A-Timer timeout

This timer bounds A for incoming DataTransfer PDUs [reference to Delta-t]. When DTCP is present, this timer bounds the time before an Ack is sent. When this timer expires, and there are allowable gaps up to the SequenceNumber that initiated this Timer, they are acked as well. If it is not running and a DataTransfer PDU arrives, the timer is started.

This timer also has a similar function when DTCP is not present. In those cases, when DataTransferPDUs arrive out of order, it represents the time the PDUs are held to see if the gaps are filled. When the timer expires and if any allowable gaps exist, the PDUs up to the PDU that started this timer are passed to delimiting. A Read-Immediate will read these PDUs and overrides the A-Timer.

When Set

This timer is started when a DataTransfer PDU arrives and it is not running.

Action Upon Expiration

```
When this timerfires:
If DTCPpresent Then
Policy SendingAck with Default:
      SVUpdate(MaxSequenceNumberRcvd) /* Update Left Edge, etc. */
      Invoke Delimiting /* to create SDUs to the user */
      Reset the RcvrInactivityTimer
End Default;
Else /* No DTCP */
      Update RcvLeftWindowEdge
      Invoke Delimiting /* to create SDUs to the user */
      Reset the RcvrInactivityTimer
FI
```

8 Detailed description of the DTCP task

8.1 General

This specification is entirely logical; it is intended to indicate what needs to be done. It is indicative, not prescriptive of an implementation design, and to aid clarity, omits some details that can be reconstructed from the requirements of the message format, such as setting flags appropriately.

8.2 Types of control PDUs

Control PDUs share the Common PCI. The format of the rest of the Control PDU depends on specific PDU-Type. All the possible Control PDU sub-type combinations are described in Clause 9.

9 Detailed description of the DTCP task

9.1 General

This specification is entirely logical; it is intended to indicate what needs to be done. It is indicative, not prescriptive of an implementation design, and to aid clarity, omits some details that can be reconstructed from the requirements of the message format, such as setting flags appropriately.

9.2 Types of control PDUs

Control PDUs share the Common PCI. The format of the rest of the Control PDU depends on specific PDU-Type. All the possible Control PDU sub-type combinations are described in this subclause.

PDU-Type – Control PDUs have PDU-Type equal to X'Cz' where x is formatted as follows:

B'xxx1' indicates Ack information present

B'xx1x' indicates Nack information present

B'x1xx' indicates Flow Control information present (always set)

B'1xxx' indicates Selective Ack/Nack

9.3 DTCP state vector

DTCPStateVector: Struct

Window-based: Boolean

Rate-based: Boolean

RetransmissionPresent: Boolean

ATime: Unsigned Time in ms before sending Acknowledgements. /* ATime = 0 means immediate delivery. */

SndRightWindowEdge: Unsigned(SequenceNumberLength)

RcvRightWindowEdgeSent: Unsigned(SequenceNumberLength)

RTT: RTTLen

RetransmissionQueue: List of {PDU, Timer}

ClosedWindowQueue: List of {PDU}

ClosedWindowQueueLength: Integer

MaxClosedWindowQueueLen: Integer

ClosedWindow: Boolean

RateFulfilled: Boolean

ECN: Boolean

ATime – If Retransmission Control is present, this variable indicates whether Acks are sent immediately (ATime = 0) or after the ATimer expires,

RTT – Round-Trip Time – Amount of time in ms for the PDU to get to the destination and back.

Retransmission Control

RetransmissionPresent – Indicates whether the Retransmission control is present.

SndRightWindowEdge – The highest sequence number that the remote application is currently willing to accept on this connection.

RetransmissionQueue – The queue of PDUs that have been handed off to the RMT but not yet acknowledged.

NextSenderControlSequenceNumber – This state variable will contain the Sequence Number to be assigned to a Control PDU sent on this connection.

LastRcvControlSequenceNumber – This state variable contains the sequence number of the next expected Transfer PDU received on this connection.

SndLeftWindowEdge – This is the last Sequence Number acked by the Receiver of DT-PDUs. That is the opposite side's RcvLeftWindowEdge. It indicates the Sequence Number of DT-PDU that the Receiver won't request to be retransmitted. The smallest Sequene Number on RetransmissionQ is at least one bigger than SndLeftWindowEdge.

DataReXmitMax – The maximum number of retransmissions of PDUs without a positive acknowledgement that will be tried before declaring an error.

Nota Bene: DTCP PDU Retransmission: Note that there is no retransmission queue for DTCP Control PDUs, when a lost control PDU is detected a new one is generated with the current information.

Flow Control State

Window-Based – Window-based Flow Control is activated.

Rate-Based – Rate-based Flow Control is activated.

ClosedWindowQueue – The queue of PDUs ready to be sent once the window opens.

ClosedWindowQueueLength – The number of PDUs queued to send because the flow control window is shut.

MaxClosedWindowQueueLen – The maximum number of PDUs that can be queued by the flow control window being shut.

ClosedWindow – This Boolean indicates whether or not the flow control window is closed.

RateFulfilled – This Boolean indicates that with rate-based flow control all the PDUs that can be sent during this time period have been sent.

TimeUnit – This field contains the unit of time in milliseconds over which the rate is computed.

SndRightWindowEdge – The largest Sequence Number of a PDU that may be sent without the receiver discarding it.

SendingRate – This variable contains the number of PDUs that may be sent in one Time Unit. The rate is defined such that the sender may send the specified number of PDUs in that unit of time. Thus, the rate will not necessarily generate a steady flow, but may exhibit a bursty pattern.

PDUsSentinTimeUnit – This variable contains the number of PDUs sent in this Time Unit. When PDUsSentinTimeUnit equals SndrRate, the sender shall wait for the beginning of a new time unit before additional PDUs may be sent.

SendingTimeUnit – The time period used to measure the SendingRate (measured in milliseconds).

SendBytesFree – The number of bytes that this flow can assume it has available for Writes.

SendBytesPercentFree – The percent of bytes that are free for Writes.

SendBytesThreshold – The number of free bytes at which flow control blocks the user from doing any more Writes.

SendBytesPercentThreshold – The percent of free bytes at which flow control blocks the user from doing any more Writes.

SendBuffersFree – The number of buffers of MaxSDU size that this flow can assume it has available for Writes.

SendBuffersPercentFree – The percent of buffers of MaxSDU size that are free for Writes.

SendBuffersThreshold – The number of free buffers at which flow control blocks the user from doing any more Writes.

SendBufferPercentThreshold – The percent of free buffers at which flow control blocks the user from doing any more Writes.

RcvRightWindowEdge – The absolute value of the credit on this flow. It is the highest sequence number of Data Transfer PDU that the receiver is willing to receive without discarding it.

RcvRate – This variable contains the current rate that the receiver has told the sender that it may send PDUs at.

PDUsRcvdinTimeUnit – This variable contains the number of PDUs received in this Time Unit. When PDUsRcvdinTimeUnit equals RcvRate, the receiver is allowed to discard any PDUs received until a new time unit begins.

RcvBytesFree – The number of bytes that this flow can assume it has available for incoming PDUs on this connection.

RcvBytesPercentFree – The percent of bytes that are free for incoming PDUs on this connection.

RcvBytesThreshold – The number of free bytes at which flow control does not move the Right Window Edge.

RcvBytesPercentThreshold – The percent of free bytes at which flow control does not move the Right Window Edge.

RcvBuffersFree – The number of buffers of MaxPDU size that this flow can assume it has available for incoming PDUs

RcvBuffersPercentFree – The percent of buffers of MaxPDU size that are free.

RcvBuffersThreshold – The number of free buffers at which flow control does not advance the Right Window Edge.

RcvBufferPercentThreshold – The percent of free buffers at which flow control does not advance the Right Window Edge.

ECN – This Boolean indicates whether there is congestion that would forbid sender from sending DataTransferPDUs. This variable is checked in default behavior of TransmissionControl policy.

Rendezvous-at-the-sender – This Boolean indicates whether there is a zero-length window and a Rendezvous PDU has been sent.

Rendezvous-at-the receiver – This Boolean indicates whether a Rendezvous PDU was received. The next DT-PDU is expected to have a DRF bit set to true.

9.4 Syntax of PDUs

9.4.1 General

All fields used in ControlPDUs are named from its sender point of view. In other parts of this document the term Sender can refer to “Sender of DataTransferPDUs”.

9.4.2 Ack/Flow control PDUs

PDU TYPE = X’C5’ Ack and Flow Control

PDU TYPE = X’CE’ Nack and Flow Control

PDU TYPE = X’CD” Sack and Flow Control

ControlSequenceNumber: Integer(ControlSequenceNumberLength)

Ack/NackSequenceNumber: Integer(SequenceNumberLength)

Credit: Integer(CreditLength)

RcvRate: RateLen

TimeUnit: TimeLen

SndLeftWindowEdge:SeqenceNumberLength

SndRate:RateLen

These PDUs are used for simple Retransmission and Flow Control updates. All PDU types with the Ack/ Nack Present bit set will contain Ack/NackSequenceNumber. All Control PDUs contain Flow Control fields.

ControlSequenceNumber – This is the sequence number of the DTCP PDU. It is taken from a different sequence number space than the data. The DTCP PDUs are not normally acked except to probe the apposite protocol machine or to detect lost DTCP PDUs. This field is present in all DTCP PDUs.

Ack/NackSequenceNumber – This field contains the base SequenceNumber being Acked or Nacked. If an Ack, the sender is to assume that no PDUs with SequenceNumbers less than or equal to this one will be requested for retransmission. In the most common use-case (in-order, no gaps) this will be the Sequence number of the last received DT-PDU, i.e. RcvLeftWindowEdge. If a Nack, the sender is to retransmit the PDU with this SequenceNumber and all PDUs with a Sequence Number greater up to LastSequenceNumberSent in its retransmission queue and restart their retransmission timeouts.

Credit – This field contains the credit value that is added to Ack field to be SndRightWindowEdge.

RcvRate – If Flow Control is in use, this field contains the value of the rate that the receiver is giving to the sender. The Sender is allowed to send this many PDUs in the given Time Unit.

TimeUnit – If Flow Control is in use, this field contains the unit of Time in milliseconds over which the rate is computed.

9.4.3 Selective Ack PDU

PDU TYPE = X'CD' Selective Ack and Flow Control

ControlSequenceNumber: Integer(ControlSequenceNumberLength)

RcvRate: RateLen

TimeUnit: TimeLen]

SndLeftWindowEdge:SeqenceNumberLength

Credit (CreditLength)

SndRate:RateLen

Ack List Length: Integer(8)

Ack List: Sequence(StartingSequenceNumber Integer(SequenceNumberLength),

EndingSequenceNumber Integer(SequenceNumberLength))

These PDU types are used to selective Ack PDUs. Generally, this is the form used, the Selective Ack operation with Flow Control is provided for completeness.

ControlSequenceNumber – This is the sequence number of the DTCP PDU. It is taken from a different sequence number space than the data. The DTCP PDUs are not normally acked except to probe the apposite protocol machine or to detect lost DTCP PDUs. This field is present in all DTCP PDUs.

RcvLeftWindowEdge – This specifies the RcvLeftWindowEdge or the lower bound for all of the sequence numbers in the Ack/Nack List. This may be a new Left Window Edge or it may just be confirming the previous Left Window Edge.

Credit – This field contains the credit value that is added to Ack field to be SndRightWindowEdge.

SndRate – This parameter confirms the Rate the Sender has on its inbound flow. (This is sent just to confirm that the two ends are coordinated.)

Ack List Length – This field specifies the number of Sequence Number pairs in the Ack List. A value of zero indicates that there is no list present.

Ack List – This field consists of pairs of Sequence Numbers. The first designating the beginning of a range of Sequence Numbers and the second the end of the range of Sequence Numbers to be acked inclusively.

9.4.4 Control Ack PDU

PDU TYPE = X'C0'

ControlSeqenceNumber: Integer(ControlSequenceNumberLength)

LastControlSequenceNumberRcvd: Integer(ControlSequenceNumberLength)

SndLeftWindowEdge:Integer(SeqenceNumberLength)

Credit: Integer(CreditLength).

RcvRate:RateLen

This PDU type is used to recover synchronization when side believes the other may have lost a DTCP PDU. Strictly speaking this PDU is not required. After 2MPL, the Protocol Machine can simply set the DRF flag and start sending data. This PDU Type may be used to shorten that time period.

LastControlSequenceNumberRcvd – This field contains the sequence number of the last Control PDU received.

RcvLeftWindowEdge – This specifies the RcvLeftWindowEdge or the lower bound for all of the sequence numbers in the Ack/Nack List. This may be a new Left Window Edge or it may just be confirming the previous Left Window Edge.

RcvRightWindowEdge – The sender of this controPDU RcvRightWindowEdge.

SndLeftWindowEdge – The sender of this ControlPDU SndLeftWindowEdge. That is, the sender of the DataTransferPDUs being acked/Nacked will compare this value with its RcvLeftWindowEdge. (This is sent just to confirm that the two ends are coordinated.)

Credit – This field contains the credit value that is added to Ack field to be SndRightWindowEdge.

9.4.5 Rendezvous PDU

PDU TYPE = X'CF'

```
ControlSequenceNumber: Integer(ControlSequenceNumberLength)
LastControlSequenceNumberRcvd: Integer(ControlSequenceNumberLength)

RcvLeftWindowEdge(SequenceNumberLength)
RcvRightWindowEdge(SequenceNumberLength)
SndLeftWindowEdge:SequenceNumberLength
SndRightWindowEdge:SequenceNumberLength
```

RcvRate:RateLen

This PDU type is used by the Rendezvous mechanism to notify the receiver that the sender cannot transmit more DT-PDUs because of a zero-length window or a zero rate. The lifetime of this PDU will be set to Δt, and when sent, the Sender Inactivity Timer will be reset.

LastControlSequenceNumberRcvd – This field contains the sequence number of the last Control PDU received.

RcvLeftWindowEdge – This specifies the RcvLeftWindowEdge of the sender of the Rendezvous PDU. It is sent so that the receiver can check consistency with its window values.

RcvRightWindowEdge – TheRcvRightWindowEdge of the sender of the Rendezvous PDU. It is sent so that the receiver can check consistency with its window values.

SndLeftWindowEdge – TheSndLeftWindowEdge of the sender of the Rendezvous PDU. The receiver of the Rendezvous PDU will compare this value with its RcvLeftWindowEdge.

SndRightWindowEdge – The SndRightWindowEdge of the sender of the Rendezvous PDU. The receiver of the Rendezvous PDU will compare this value with its RcvRightWindowEdge, to check consistency.

9.5 Detailed specification of common functions

Procedure Name:CommonRcvControl
Function
This procedure handles common processing of control PDUs.

Parameters
An incoming control PDU.

Description of Algorithm

This procedure first checks for duplicate acks and credit and logs the incident and discards the control PDU. If it detects a lost control PDU, i.e. this is not the next control sequence number expected, it invokes the LostControlPDU Policy.

```
CommonRcvControl(PDU)
Begin
If PDU.SequenceNumber ≤ LastControlSequenceNumberRcvd Then
   DupControlPDU := * + 1 Fi
   Discard PDU
    Return False;
Else
    If PDU.SequenceNumber > LastControlSequenceNumberRcvd+1 Then /* out of order */
    Policy LostControlPDU with Default:
    Send empty TransferPDU
         LastControlPDURcvd := PDU.ControlSequenceNumber
         End Default;
    Else /* in order */
         LastControlPDURcvd := PDU.ControlSequenceNumber
     Fi
 Fi
Return True;
End CommonRcvControl
```


9.6 Event processing

9.6.1 Event — SVUPdate(PDU.SequenceNumber) occurred

When invoked

This event is invoked by DTP when there is a change in the state vector that is of interest to the DTCP, potentially on every PDU received. This procedure may be invoked when updating NextSequenceNumberToSend.

There are two default Ack policies: immediate and timed. The immediate default sends an ack as soon as a Transfer PDU is received. The Timed default sends an Ack only when the A-Timer fires. The value of the A-Timer will have to be slightly less than A to ensure that an ack is sent within A.

This is here to emphasize the separation of DTP and DTCP. In a hardware implementation, it would be simple enough for this to be a signal that causes the appropriate action. In software, such implementations are generally less than efficient; hence some or all of this action may be implemented A

Action upon invocation

```
      Update RcvLeftWindowEdge
      If Window-based Then
      Policy RcvrFlowControl with Default:
            /* This policy has to be the one to impose the condition to
            set WindowClosed to True */
            If RcvPercentBuffersFree > RcvPercentBuffersThreshold Then
                  Adjust Credit to Maintain QoS Parameters;
              Else /* Buffers are getting low */
                  Adjust Credit to keep Right Window Edge unchanged.
              Fi
End Default;
Fi
If Retransmission present Then
           Policy RcvrAck with Default:
            If ATime = 0 Then
                Send an Ack/FlowControl PDU;
                Stop any A-Timers associated with this PDU and earlier ones.
           Else
                Set A-timer for this PDU
           Fi
           End Default;
   Fi
   If no Retransmission Control Then
           Policy ReceivingFlowControl with Default:
           If Window-based Then
                Send FlowControl PDU
                /* Already updated the window and not sending Ack */
           Fi
           End Default:
   Fi
```

9.6.2 Event — Ack/Nack flow control PDUs arrived

Function

This section describes the action to be taken for the simple ack/nack/flow control types of DTCP PDUs. This section handles these PDU types: X'C5' Ack and Flow Control, X'C6' Nack and Flow Control, and X'C4' Flow Control only.

Conditions for generating this PDU

The conditions for sending it are the subject of RcvrAck policy.

Action on receipt of this PDU

When the PM receives a PDU of one of these types, it determines whether it is well-formed and if not the PDU is discarded; otherwise:

```
   If CommonRcvControl(PDU) = False then
      Return;
   Fi
    Retrieve the Time of this Ack
    RTT := RTTEstimator.

    If this is an Ack Then
   Policy SenderAck with default:
              Delete all PDUs from the RetransmissionQ with
   SequenceNumber less than or equal to the value of Ack/Nack field and
   discontinue any Retransmission Timers associated with them.
        Update SndLeftWindowEdge accordingly
   End Default;
     If RetransmissionQ = empty then
           setDRF := True
```