Technical Descriptions for Cut-Through Forwarding in Bridges

DCN 1-22-0042-05-ICne

Author: Johannes Specht

September 22, 2022

Contents

List of Figures

Part I.

Introduction

1. Purpose

This document is an individual contribution by the author, provided for technical discussion in pre-PAR activities of IEEE 802 (i.e., Nendica). The contents of this document are technical descriptions for the operations of Cut-Through Forwarding (CTF) in bridges. The intent is to provide more technical clarity, and thereby also address the desire expressed by some individuals during the IEEE 802 Plenary Meeting in July 2022 to a certain extent.

2. Relationship to IEEE Standards

This document **IS NOT** an IEEE Standard or an IEEE Standards draft. This allows readers to focus on the technical contents in this document, rather than additional aspects that are important during standards development. For example:

1. The structure of this document does not comply with the structural requirements for such standards. For example, it does not contain mandatory clauses for IEEE Standards [1].

2. Usage of normative keywords has no implied semantics beyond explicit description. For example, usage of the words *shall*, *should* or *may* **DOES NOT** imply requirements or recommendations for conformance of an implementation.

3. This document contains references, but without distinguishing between normative and informative references.

4. This document does not contain suggestions for assigning particular contents to *vehicles* (e.g., IEEE 802 Working Groups, potential amendment projects for existing standards, or potential new standard projects). As a consequence, the clause structure of this document is intended for readability, rather than fitting into the clause structure of a particular Standard (i.e., which would matter for potential amendment projects).

3. Status of this Document

This document is work-in-progress in an early stage. It contains technical and editorial errors, omissions and simplifications. Readers discovering such issues are encouraged for making enhancement proposals, e.g. by sending such proposals to the author by email (johannes.specht.standards@gmail.com).

Part II.

Cut-Through Forwarding in Bridges

4. Generalized Serial Convergence Operations

4.1. Overview

The generalized serial convergence operations are described by a stack of processes that interact via global variables (see 4.4) and service primitive invocations (see 4.2). These processes provide the translation between the Internal Sublayer Service (ISS) and a broad range of lower layers, including (but not limited to) physical layers. Figure 4.1 provides an overview of these processes and their interaction[1]. The processes can

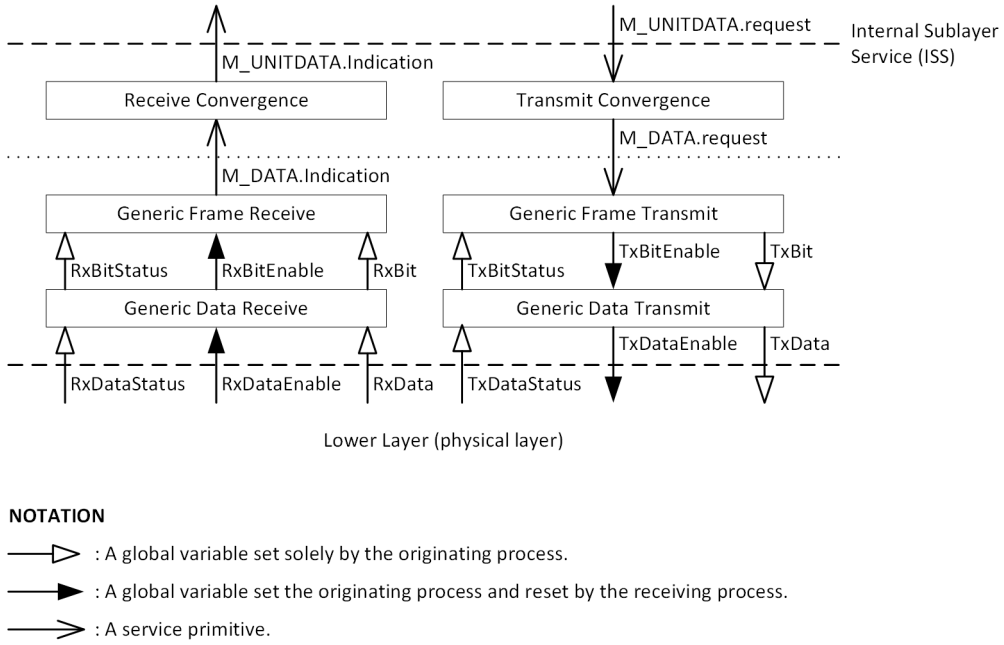

Figure 4.1.: Overview of the generalized serial convergence operations.

be summarized as follows:

[1] This interaction model is inspired by clause 6 and 8.6.9 of IEEE Std 802.1Q[2].

1. A Receive Convergence process (4.7) that translates each invocation of the M_DATA.-indication service primitive (4.2.1) into a corresponding invocation of the M_UNITDATA.-indication service primitive (4.2.2).

2. A Generic Frame Receive process (4.6) that generates M_DATA.indication invocations for bit sequences originating from the Generic Data Receive process of at least LEN_MIN (4.3.5) bits.

3. A Generic Data Receive process (4.5) that translates a lower layer-dependent[2] serial data stream into delineated homogeneous bit sequences of variable length, each typically representing a frame.

4. A Transmit Convergence process (4.10) that translates each invocation of the M_UNITDATA.request service primitive into a corresponding invocation of the M_DATA.request service primitive.

5. A Generic Frame Transmit process (4.9) that translates M_DATA.request invocations into bit sequences for the Generic Data Transmit process.

6. A Generic Data Transmit process (4.8) that translates bit sequences from the Generic Frame Transmit process into a lower layer-dependent serial data stream.

The generalized serial convergence operations are inspired by the concepts described in slides by Roger Marks [3, slide 15], but follow a different modeling approach with more formalized description of these functions and incorporate some of the following concepts, as suggested by the author of this document during the Nendica meetings on and after August 18, 2022. The differences can be summarized as follows:

- Alignment with the state machine diagram conventions in Annex E of IEEE Std 802.1Q[2].
- Support for serial data streams from lower layers with arbitrary data word length[3].
- Explicit modeling of atomic ISS service primitive invocations.

By keeping ISS service primitive invocations atomic, the approach in this document is intended to provide a higher level of compatibility with existing IEEE 802.1 Stds, similar to the modeling approach via frame look-ahead of service primitive invocations/prescient functions[4, slides 7ff.].

[2] Such a lower layer may be an entity on the physical layer (PHY), but the generalized receive operations are not limited to this.

[3] This generalization is intended to allow a wide range of lower layers. In addition, the support for word sizes (e.g., 8 bits, 32 bits or 64 bits) may be close to realities found in hardware implementation. It is subject to discussion whether this and other generalizations over [3] introduced by the author are considered to be helpful.

4.2. Service Primitives

4.2.1. M_DATA.indication and M_DATA.request

The M_DATA.indication service primitive passes the contents of a frame from the Generic Frame Receive process to the Receive Convergence process. The M_DATA.-request service primitive passes the contents of a frame from the Transmit Convergence process to the Generic Frame Transmit process. This parameter signatures of the service primitives are as follows[4]:

M_DATA.indication(DA, SA, SDU, FCS)

M_DATA.request(DA, SA, SDU, FCS)

The parameters are defined as follows:

4.2.1.1. DA

An array of zero to LEN_ADDR (4.3.3) bits, containing the destination address of a frame.

4.2.1.2. SA

An array of zero to LEN_ADDR (4.3.3) bits, containing the source address of a frame.

4.2.1.3. SDU

An array of zero or more bits, containing a service data unit of a frame. The number of bits after complete reception of a frame is an integer multiple LEN_OCT (4.3.2).

4.2.1.4. FCS

An array of zero to LEN_FCS (4.3.4) bits, containing the frame check sequence of a frame.

4.2.2. M_UNITDATA.indication and M_UNITDATA.request

As specified in IEEE Std 802.1AC[6, 11.1], with the parameter signatures summarized as follows:

[4]The parameters in this version of this document limit to those introduced in Roger Marks' GSCF slides [3]. Future versions may introduce more flexibility (e.g., for IEEE Std 802.11 [5, 9.2]).

```
M_UNITDATA.indication(
    destination_address,
    source_address,
    mac_service_data_unit,
    priority,
    drop_eligible,
    frame_check_sequence,
    service_access_point_identifier,
    connection_identifier
)

M_UNITDATA.request(
    destination_address,
    source_address,
    mac_service_data_unit,
    priority, drop_eligible,
    frame_check_sequence,
    service_access_point_identifier,
    connection_identifier
)
```

4.2.3. Atomic Invocation Models

4.2.3.1. Bit-Accurate Modeling

All invocations of service primitives in this document are atomic. That is, each invocation is non-dividable (see also 7.2 of IEEE Std 802.1AC[6]). Service primitive invocations are modeled more explicitly in this document, allowing for accurate description of operations within a Bridge, while retaining atomicity. This explicit model comprises the following:

1. A service primitive provides two attributes[5], *'start* and *'end*. These attributes are used in subsequent descriptions to indicate the start and the end of the indication, respectively.

2. The parameters of a service primitive are explicitly modeled as bit arrays.

3. The values of parameters during invocations of a service primitive are passed according to a call-by-reference scheme.

In a series of sequential *processing stages* (e.g., the processes introduced in 4.1 or a sub-process of the forwarding process in 6), this model allows later processing stages to access contents in service primitive parameters that are incrementally added by an earlier processing stage.

[5] The concept of *attributes* is inspired by the *Very High Speed Integrated Circuits Hardware Description Language*, VHDL[7], which provides predefined attributes (e.g., *'transaction*) that allow modeling over multiple VHDL simulation cycles at the same instant of simulated time.

4.2.3.2. Parameter-Accurate Modeling

At higher levels processing stages, service primitives of frames and processing of these frames themselves is modeled at parameter level accuracy. The purpose of this model is to

1. provide means for compact description of temporal control (4.2.3.3) in and across processing stages,
2. enable re-use of existing transformation rules from IEEE 802.1 Stds by reference, and
3. avoid low level details that would not provide any value to the clarity and unambiguity of descriptions.

The parameter-accurate operates at the resolution of symbolic and/or numeric parameters instead of bit arrays (4.2.3.1). A parameter is said to be *complete* once the *minimal information* is available to *unambiguously* determine the parameter's value. The minimal information may be

1. a sequence of bits found in a frame,
2. the result of composition and/or computation across bits located at various locations in a frame,
3. based on out-of-band information, or
4. any combination of the aforesaid.

As an example, the vlan_identifier parameter of EM_UNITDATA.indication (5) invocations can be derived from a subset of underlying bits of the associated SDU parameter of M_DATA.indication invocations (4.2.1) that are located in a VLAN Tag according to the specification of the Support for the EISS defined in IEEE Std 802.1Q [2, 6.9], or originate from out-of-band information such a default value, potentially set by management, in absence of a VLAN Tag.

Most of the data transformations from bit-contents of frames and potential out-of-band information is already unambiguously specified in the relevant IEEE 802.1 Standards. This document omits description of already specified transformations and instead just refers to the relevant transformations existing IEEE 802.1 Standards.

4.2.3.3. Temporal Control

Parameter-accurate modelling allows formulating temporal control in processing stages. A processing stage (4.2.3.1) may *stall* further processing of a frame, including (but not limited to) passing this frame to a subsequent processing stage, until one or more parameters are complete (4.2.3.2). Most processing stalls are implicit due to the data dependencies already specified in IEEE 802.1 Standards (e.g., Ingress Filtering as part of the forwarding process in IEEE Std 802.1Q[2, 8.6.2] depends on the availability of

a frame's VID, which therefore implictly requires completion of the vlan_identifier parameter of EM_UNITDATA.indication invocations), however, explicit modeling of processing stalls may be expressed by formulations in natural language.

Example formulations:

1. *"Processing **stalls** on the **vlan_identifier** parameter."*
2. *"Further execution in a MAC bridge is **stalled** on the **destination address** of a frame prior to the forwarding database lookup of the destination ports."*

4.3. Global Constants

4.3.1. PREAMBLE

A lower layer-dependent array of zero[6] or more bits, containing the expected preamble of each frame.

4.3.2. LEN_OCT

The integer number eight (8), indicating the number of bits per octet.

4.3.3. LEN_ADDR

An integer denoting the length of the DA and SA parameters of M_DATA.indication parameters, in bits. For example,

$$\text{LEN\_ADDR} = 48 \tag{4.1}$$

indicates an EUI-48 addresses.

4.3.4. LEN_FCS

An integer denoting the length of frame check sequence and the length FCS parameter of M_DATA.indication parameter, respectively, in bits. For example,

$$\text{LEN\_FCS} = 32 \tag{4.2}$$

indicates a four octet frame check sequence.

4.3.5. LEN_MIN

A lower layer-dependent integer, denoting the minimum length of a frame, in bits. Invocation of the The M_DATA.indication service primitive starts once the Generic

[6] Including length zero permits to support lower layers that do not expose a preamble to the Generic Data Receive process.

Frame Receive process received the first LEN_MIN bits of a frame. Values for LEN_MIN with

$$\text{LEN\_MIN} \geq \text{PREAMBLE.length} + \text{LEN\_FCS} \tag{4.3}$$

are valid.

4.3.6. LEN_MAX

A lower layer-dependent integer, denoting the maximum length of a frame, in bits. Invocation of the The M_DATA.indication service primitive ends at latest once the Generic Frame Receive process received at most LEN_MAX bits of a frame. Values for LEN_MIN with

$$\text{LEN\_MAX} \geq \text{PREAMBLE.length} + 2\text{LEN\_ADDR} + \text{LEN\_FCS} \tag{4.4}$$

are valid.

4.3.7. LEN_DATA

A lower layer-dependent integer, denoting the width of the RxData variable, in bits.

4.4. Global Variables

4.4.1. RxBitEnable

A Boolean variable, set by the Generic Data Receive process and reset by the Generic Frame Receive process, which indicates an update of the RxBit variable, RxBitStatus variable, or both.

4.4.2. RxBit

A bit variable used to pass a single bit value to the Generic Frame Receive process.

4.4.3. RxBitStatus

An enumeration variable used to pass the receive status from the Generic Data Receive process to the Generic Frame Receive process. The valid enumeration literals are as follows:

RECEIVING Indicates that the Generic Data Receive process received data from lower layers in a serial stream without knowledge of the remaining length of the overall data stream.

TRAILER Indicates that the Generic Data Receive process received data from lower layers in a serial stream with the knowledge that LEN_FCS or less bits follow.

Algorithm 4.1 Definition of data type `low_data_t`.

```
typedef struct {
  Boolean start;
  Boolean end;
  bit[]   value;
} low_data_t;
```

4.4.4. RxDataEnable

A Boolean variable, set by a lower layer and reset by the Generic Data Receive process, which indicates an update of the RxData variable, RxDataStatus variable, or both.

4.4.5. RxData

A variable of composite data type *low_data_t*, used for serially passing data words of frames from a lower layer to the Generic Data Receive process. Type low_data_t is defined in Listing 4.1. The semantics of the constituent parameters is as follows:

start Indicates whether the data word is the first word of a frame (TRUE) or not (FALSE).

end Indicates whether the data word is the last word of a frame (TRUE) or not (FALSE).

value A lower layer-dependent non-empty array of up to LEN_DATA (4.3.7) bits, containing a data word of a frame. An array length less than LEN_DATA bits is only valid if eof is TRUE.

4.4.6. RxDataStatus

An enumeration variable used to pass the receive status from lower layers to the Generic Data Receive process. The valid enumeration literals are as follows:

RECEIVING Indicates that data stream reception from lower layers is active.

IDLE Indicates that data stream reception from lower layers is not active.

4.4.7. TxBitEnable

A Boolean variable, set by the Generic Frame Transmit process and reset by the Generic Data Transmit process, which indicates an update of the TxBit variable.

4.4.8. TxBit

A bit variable used to pass a single bit value to the Generic Data Transmit process.

4.4.9. TxBitStatus

An enumeration variable that establishes a back pressure mechanism from the Generic Data Transmit process to the Generic Frame Transmit process. The valid enumeration literals are as follows:

READY Indicates that the Generic Data Transmit process can accept one or more bit(s) from the Generic Frame Transmit process.

BUSY Indicates that the Generic Data Transmit process cannot accept bits from the Generic Frame Transmit process.

4.4.10. TxDataEnable

A Boolean variable, set by the Generic Data Transmit process a lower layer and reset by the lower layer, which indicates an update of the TxData variable.

4.4.11. TxData

A variable of composite datatype low_data_t (4.1), used for serially passing data words of frames from the Generic Data Transmit process to a lower layer.

4.4.12. TxDataStatus

An enumeration variable that establishes a back pressure mechanism from the lower layer to the Generic Data Transmit process. The valid enumeration literals are as follows:

READY Indicates that a lower layer can accept one or more bit(s) from the Generic Data Transmit process.

BUSY Indicates that a lower layer cannot accept bits from the Generic Data Transmit process.

4.5. Generic Data Receive

The Generic Data Receive process translates a lower layer-dependent[7] serial data stream into a uniform bit stream. In addition, it realizes the following functions:

- Determine the position in the serial data stream of a frame at which the frame check sequence begins (delay line modeling).
- Truncate excess bits to satisfy the frame length requirements implied by the parameter definition of the M_DATA.indication primitive (4.2.1).

[7] Such a lower layer may be an entity on the physical layer (PHY), but the generalized receive operations are not limited to this.

4.6. Generic Frame Receive

4.6.1. Description

The Generic Frame Receive process transforms a serial bit streams of frames from the Generic Data Receive process into invocations of the M_DATA.indication primitive.

4.6.2. State Machine Diagram

The operation of the Generic Frame Receive process is specified by the state machine diagram in Figure 4.2 , using the variables and functions defined in subsequent subclauses.

4.6.3. Variables

4.6.3.1. cnt

An integer counter variable, used to count the number of bits in the current parameter of the frame.

4.6.3.2. len

An integer variable holding the actual length of a frame under reception, in bits.

4.6.3.3. status

An enumeration variable holding the current status of the Generic Frame Receive process. The valid enumeration literals are as follows:

Ok Indicates that no error has been discovered prior or during frame reception.

FrameTooLong Indicates that a frame under reception exceeded LEN_MAX bits.

FCSInvalid Indicates inconsistency between the FCS parameter an the remaining parameters of a frame under reception.

4.6.4. Functions

4.6.4.1. append(parameter,bit)

The append function appends a given bit at the end of a particular parameter of an M_DATA.indication service primitive.

4.6.4.2. FCSValid(FCS)

The FCSValid function determines if the FCS parameter consistent with the remaining parameters of the M_DATA.indication service primitive (TRUE) or not (FALSE).

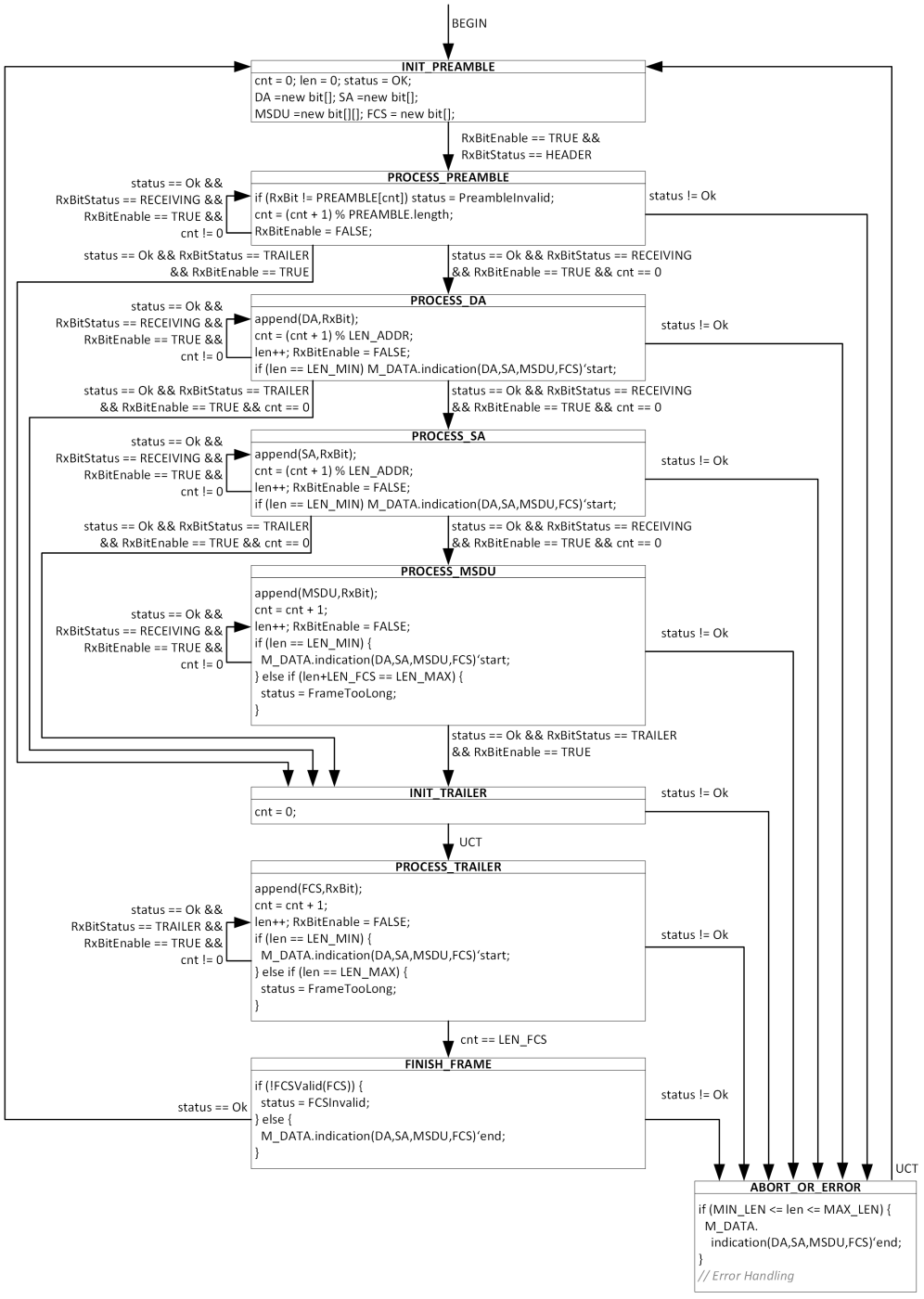

Figure 4.2.: State Machine Diagram of the Generic Frame Receive Process.

4.7. Receive Convergence

The Receive Convergence Process implements the translation of M_DATA.indication invocations to M_UNITDATA.indication invocations. The supported translations are lower layer-dependent and include, but are not limited to, those specified in clause 13 of IEEE Std 802.1AC[6].

Each M_DATA.indication invocation results in an associated M_UNITDATA.-indication invocation. During the translation, the M_UNITDATA.indication parameters are extracted from the M_DATA.indication parameters according to the rules defined for the underlying lower layer.

4.8. Generic Data Transmit

PLACEHOLDER, for descriptions symmetrical to 4.5.

4.9. Generic Frame Transmit

PLACEHOLDER, for descriptions symmetrical to 4.6.

4.10. Transmit Convergence

PLACEHOLDER, for descriptions symmetrical to 4.7.

5. Translation between Internal Sublayer Service (ISS) and Enhanced Internal Sublayer Service (EISS)

Data translation from service primitive invocations of the ISS and service primitive invocations of the EISS follows the associated rules specified in IEEE Std 802.1Q [2, 6.9]. The temporal behavior (4.2.3.3) between ISS and EISS service primitive invocations is as follows:

1. For EM_UNITDATA.indication invocations, EM_UNITDATA.indication'start and EM_UNITDATA.indication'end follow instantaneously after M_UNITDATA.-indication'start and M_UNITDATA.indication'end, respectively.

2. For EM_UNITDATA.request invocations, M_UNITDATA.request'start and M_UNITDATA.-request'end follow instantaneously after EM_UNITDATA.indication'start and EM_UNITDATA.indication'end, respectively.

6. Bridge Relay Operation

PLACEHOLDER, for describing the differences of the Bridge Relay operation as presented earlier by the author [8, p.52ff.][9, p.10f.].

7. Management Parameters

7.1. Overview

The management parameters for CTF fall into three categories:

1. Control Parameters (7.2)
2. Timing Parameters (7.3)
3. Error Counters (7.4)

The control parameters allow to (i) determine whether CTF is supported on a per Port and per Port per Traffic Class resolution, and if CTF is supported, to (ii) enable and disable CTF on these resolutions. These parameters are available in reception Ports and transmission Ports. For a pair of bridge ports, frames can only be subject to the CTF operation if CTF is supported and enabled on both Ports.

The timing parameters expose the delays experienced by frames passing from a particular reception Port to another transmission Port. These parameters are primarily intended for automated network and traffic configuration, for example, by a Centralized Network Controller (CNC) using the associated mechanisms from IEEE Std 802.1Q [2, clause 46].

The error counters expose information on frames that were subject to the CTF operation in a bridge, even though such frames have consistency errors (i.e., a frame check sequence inconsistent with the remaining contents of that frame) during reception by this bridge. These counters are primarily intended for manual diagnostic purposes to support identifying erroneous links or stations, for example, by a human network administrator.

7.2. Control Parameters

7.2.1. CTFTransmissionSupported

A Boolean read-only parameter that indicates whether CTF on transmission is supported (TRUE) or not (FALSE). There is one CTFTransmissionSupported parameter for each traffic class of each transmission Port.

7.2.2. CTFTransmissionEnable

A Boolean parameter to enable (TRUE) and disable (FALSE) CTF on transmission. There is one CTFTransmissionEnable parameter for each traffic class of each transmission Port. The default value of the CTFTransmissionEnable parameter is FALSE for

all traffic classes of all transmission Ports. It is an error if a CTFTransmissionEnable is set to TRUE if the associated CTF Transmission Supported parameter is FALSE.

7.2.3. CTFReceptionSupported

A Boolean read-only parameter that indicates whether CTF on reception is supported (TRUE) or not (FALSE). There is one CTFReceptionSupported parameter for each reception Port.

7.2.4. CTFReceptionEnable

A Boolean parameter to enable (TRUE) and disable (FALSE) CTF on reception. There is one CTFReceptionEnable parameter for each reception Port. The default value of the CTFReceptionEnable parameter is FALSE for all reception Ports.It is an error if a CTFReceptionEnable is set to TRUE if the associated CTFReceptionSupported parameter is FALSE.

7.3. Timing Parameters

7.3.1. CTFDelayMin and CTFDelayMax

A pair of unsigned integer read-only parameters, in units of nanoseconds, describing the delay range for frames that are subject to the CTF operation and encounter zero delay for transmission selection [2, 8.6.8]. This occurs when the queue for the frame's traffic class is empty, the frame's traffic class has permission to transmit, and the egress Port is idle (not transmitting). There is one pair of CTFDelayMin and CTFDelayMax parameters per reception Port per transmission Port traffic class pair.

7.4. Error Counters

7.4.1. CTFReceptionDiscoveredErrors

An integer counter, counting the number of received frames with discovered consistency errors. There is one CTFReceptionDiscoveredErrors parameter for each reception Port. A frame with discovered consistency errors has been identified as such by a bridge on the upstream path from which the frame originates and marked by that an implementation-dependent marking mechanism. The value of the counter always increases by one

1. if
    a) the upstream bridge that applied the marking,
    b) all bridges on the path of that bridge to the reception Port associated with the CTFReceptionDiscoveredErrors counter and

c) the receiving bridge of which the reception Port is a part of are different instances of the same bridge implementation, and

2. the underlying marking mechanism is identical for all these instances if multiple marking mechanisms are supported by these instances.

If either of the conditions in items 1 through 2 is unsatisfied, CTFReceptionUndiscoveredErrors may be increased instead of CTFReceptionDiscoveredErrors[1].

7.4.2. CTFReceptionUndiscoveredErrors

An integer counter, counting the number of received frames with undiscovered consistency errors. There is one CTFReceptionUndiscoveredErrors parameter for each reception Port. This counter is increased by one if a frame with consistency errors is received at the associated reception Port and CTFReceptionDiscoveredErrors is not increased.

[1] It is assumed that there is a variety of options for implementing a frame marking mechanism. For example, by using physical layer symbols [10, l.121 - l.126] or special frame check sequences [8, p.54, 2.2.][11, p.17]. The current description in this document permits any marking mechanism, but the associated error counters are only consistent in networks with homogeneous implementation instances, and may be inconsistent in heterogenoues networks. However, term (CTFReceptionDiscoveredErrors + CTFReceptionUndiscoveredErrors) on a reception Port should be identical in several heterogeneous networks. A human network administrator may be able to localize erroneous links or stations solely by considering this term along multiple reception Ports across a network instead of its constituents.

Part III.

Cut-Through Forwarding in Bridged Networks

480 PLACEHOLDER, for contents on using CTF in networks [8, p.46 – p.49].

Part IV.

Appendices

A. Interaction of the Generalized Serial Convergence Operations with existing Lower Layers

PLACEHOLDER, for describing the relationship Generalized Serial Convergence (4) lower layer interface and existing lower layers.

Bibliography

[1] IEEE Standards Association, *2021 IEEE SA Standards Style Manual*. [Online]. Available: https://mentor.ieee.org/myproject/Public/mytools/draft/styleman.pdf

[2] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018.

[3] Roger Marks (EthAirNet Associates), *Generic Serial Convergence Function (GSCF)*, 2022. [Online]. Available: https://mentor.ieee.org/802.1/dcn/22/1-22-0040-02-ICne-generic-serial-convergence-function-gscf.pdf

[4] Johannes Specht (Self; Analog Devices, Inc.; Mitsubishi Electric Corporation; Phoenix Contact GmbH & Co. KG; PROFIBUS Nutzerorganisation e.V.; Siemens AG; Texas Instruments, Inc.), *CTF - Considerations on Modelling, Compatibility and Locations*. [Online]. Available: https://mentor.ieee.org/802.1/dcn/22/1-22-0021-04-ICne-ctf-considerations-on-modelling-compatibility-and-locations.pdf

[5] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[6] "IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Service Definition," *IEEE Std 802.1AC-2016 (Revision of IEEE Std 802.1AC-2012)*, pp. 1–52, 2017.

[7] "IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Service Definition," *IEEE Std 802.1AC-2016 (Revision of IEEE Std 802.1AC-2012)*, pp. 1–52, 2017.

[8] Johannes Specht, Jordon Woods, Paul Congdon, Lily Lv, Henning Kaltheuner, Genio Kronauer and Alon Regev, *IEEE 802 Tutorial: Cut-Through Forwarding (CTF) among Ethernet networks – DCN 1-21-0037-00-ICne*, 2021. [Online]. Available: https://mentor.ieee.org/802.1/dcn/21/1-21-0037-00-ICne-ieee-802-tutorial-cut-through-forwarding-ctf-among-ethernet-networks.pdf

[9] Johannes Specht (Self; Analog Devices, Inc.; Mitsubishi Electric Corporation; Phoenix Contact GmbH & Co. KG; PROFIBUS Nutzerorganisation e.V.; Siemens AG; Texas Instruments, Inc.), *Cut-Through Forwarding (CTF): Towards an IEEE 802.1 Standard*, 2021. [Online]. Available: https://www.ieee802.org/1/files/public/docs2021/new-specht-ctf-802-1-1121-v01.pdf

[10] Astrit Ademaj (TTTech) and Guenter Steindl (Siemens), *Cut-Through – IEC/IEEE 60802 – V1.1*, 2019. [Online]. Available: https://www.ieee802.org/1/files/public/docs2019/60802-Ademaj-et-al-CutThrough-0919-v11.pdf

[11] Peter Jones (Cisco), *802.3 NEA CTF: CTF concerns*, 2022. [Online]. Available: https://www.ieee802.org/3/ad_hoc/ngrates/public/calls/22_0427/jones_nea_01_220427.pdf