



Switch Abstraction Interface (SAI) Specification for Link Layer Retry

v1.0

October 28, 2026

Copyright © 2026 Ultra Ethernet Consortium.

The Ultra Ethernet Consortium™ (UEC) is a consensus-based standards organization operating under the Linux Foundation. Its members collaborate openly to define and promote high-performance Ethernet technologies for modern computing environments. This specification ("Specification") contains Approved Deliverables of UEC as such term is defined in the UEC Charter.

This document is made available under the terms of the Creative Commons Attribution-NoDerivatives 4.0 International License (CC BY-ND 4.0). As stated in the license, you may copy and redistribute the document, provided that attribution is given to the Ultra Ethernet Consortium. However, if you create a derivative work from the material, you may not distribute that work. A copy of the license is available at <https://creativecommons.org/licenses/by-nd/4.0/>.

This Specification is provided on an "AS IS" basis. UEC and The Linux Foundation (the "Parties") make no warranties of any kind, either express or implied, including without limitation warranties of merchantability, fitness for a particular purpose or noninfringement of any third party intellectual property rights, or compliance with applicable laws. Use of the information in this Specification is at your own risk. The entire risk as to the results and performance of this Specification is assumed by the user. In no event will the Parties (as defined above) be liable to any other party for lost profits or any form of indirect, special, incidental, or consequential damages of any character from any causes of action of any kind with respect to this Specification or its governing document, whether based on breach of contract, tort (including negligence), or otherwise, and whether or not the recipient of this Specification has been advised of the possibility of such damage.

This Specification may include references to technologies for which patent rights might be claimed. UEC does not make any representation regarding the existence or status of any such rights. Implementers of this Specification are solely responsible for obtaining any necessary licenses or permissions from the appropriate rights holders.

UEC reserves the right to adopt any changes or alterations to this Specification as it deems necessary or appropriate.

Ultra Ethernet™, Ultra Ethernet Consortium™, and the Ultra Ethernet Consortium logo are the unregistered trademarks of Ultra Ethernet Consortium in the United States and other countries. All rights reserved. Other brands, names or logos contained herein are the property of their respective owners.

For more information about the Ultra Ethernet Consortium, visit: <https://ultraethernet.org>.

Contents

1	Introduction	7
1.1	Overview	7
1.2	Handling Errors	7
1.3	SAI Specification	7
1.3.1	SAI Logical Pipeline	8
1.3.2	Port LLR Profile	8
1.3.3	SAI Port Updates	9
1.3.4	New Port Error Status	10
1.3.5	New LLR Port Statistics	10
1.3.6	Capability Query	11
1.4	Normative References	12

Tables

Table 1-1 - Attributes of the new SAI PORT LLR PROFILE object type	8
Table 1-2 - Frame action types.....	9
Table 1-3 - New Attributes of the existing SAI PORT object type	9
Table 1-4 - sai_port_llr_tx_status_t	10
Table 1-5 - sai_port_llr_rx_status_t	10
Table 1-6 - Port Error Status	10
Table 1-7 - Port LLR Statistics	10

Figures

Figure 1-1 - Overview of SAI attributes for LLR.	8
---	---

Acknowledgments

This specification is developed under the Process, Procedures, and Guidelines of the Ultra Ethernet Consortium (UEC). The organization would like to thank all that are involved. A special acknowledgment goes out to the following individuals for their leadership, authoring, and technical contributions to this specification.

Steering Committee at Time of Publication

Role	Name	Affiliation
Chair	J Metz	AMD
Vice Chair	Abdul Kabbani	Microsoft
TAC Chair	Tom Emmons	Arista
TAC Vice Chair	Brad Matthews	Marvell

Working Group Leadership at Time of Publication

Working Group	Name	Affiliation
Physical Layer WG	Jannik Hammel Nielsen	Huawei
	Adee Ran	Cisco
Link Layer WG	Bob Alverson	HPE
	Yan Zhuang	Huawei
Transport Layer WG	Torsten Hoeffler	Microsoft
	Karen Schramm	Broadcom
Software Layer WG	Rip Sohan	Broadcom
	Jianxin Xiong	Intel
Management WG	Ravindranath Kanakarajan	Marvell
	Hemal Shah	Broadcom
Compliance & Test WG	Dave Estes	Viavi Solutions
	Cliff Tavares	Keysight
Performance & Debug WG	Rajalaxmi Angadi	Intel
	Nathan Hanford	LLNL
Storage WG	Venkat Goli	Broadcom
	Claudio DeSanti	Dell

Specification Development

Role	Name	Affiliation
Author	Rupa Budhia	Marvell
Author	Ravindranath Kanakarajan	Marvell
Contributor	Jai Kumar	Broadcom
Editor	Paul Congdon	Linux Foundation
Program Manager	Rob Craig	Linux Foundation

1 Introduction

When packets are dropped in a network due to uncorrectable FEC errors or CRC errors, protocols traditionally rely on end-to-end retransmission of lost packets. Delays caused by these end-to-end retransmissions negatively impact the performance of latency-sensitive workloads. Link Layer Retry (LLR) is a hop-by-hop mechanism that provides the capability for local retransmission of frames lost due to link errors, thereby avoiding the impact on application performance.

1.1 Overview

LLR uses sequence numbers and periodic acknowledgements to track Ethernet frames. At the LLR sender (LLR Tx), each transmitted frame is assigned a sequence number, which is encoded in the Ethernet frame's preamble. The frame is also placed in a Replay Buffer for possible retransmission if the link partner does not receive it. The sequence number is incremented by 1 for each subsequent frame.

The link partner that receives the packets (LLR Rx), periodically sends acknowledgements (LLR_ACKs) for frames that were successfully received. LLR_ACK carries the sequence number of the most recent successfully received frame. Ultra Ethernet LLR [2] sends the acknowledgements in Control Ordered Sets (CtlOS) inserted into the data stream.

When an LLR_ACK is received by the sender, all frames up to the sequence number in the received LLR_ACK are removed from the replay buffer, freeing up space.

1.2 Handling Errors

LLR uses a Go-Back-N scheme to handle missing frames. If LLR Rx receives a frame with an out of order sequence number or a CRC error, it sends a negative acknowledgement (LLR_NACK) back to the sender with the sequence number of the most recent successfully received frame. LLR Rx then discards all subsequent received frames until a frame with the expected sequence number is received.

At the sender, upon receipt of an LLR_NACK, frames in the replay buffer are retransmitted, starting with the missing frame identified from the LLR_NACK. To handle lost LLR_NACKs, a replay timer is maintained at LLR Tx, and replay is initiated when the replay timer expires. Replay can be performed multiple times if no further LLR_ACKs are received.

1.3 SAI Specification

LLR is a new feature in SAI. This SAI spec addresses LLR as defined in Chapter 5.1 of UE Specification v1.1. To support LLR, new attributes and statistics are added to the SAI Port Object. Additionally, a new object called a Port LLR Profile is introduced to model the LLR parameters of a port.

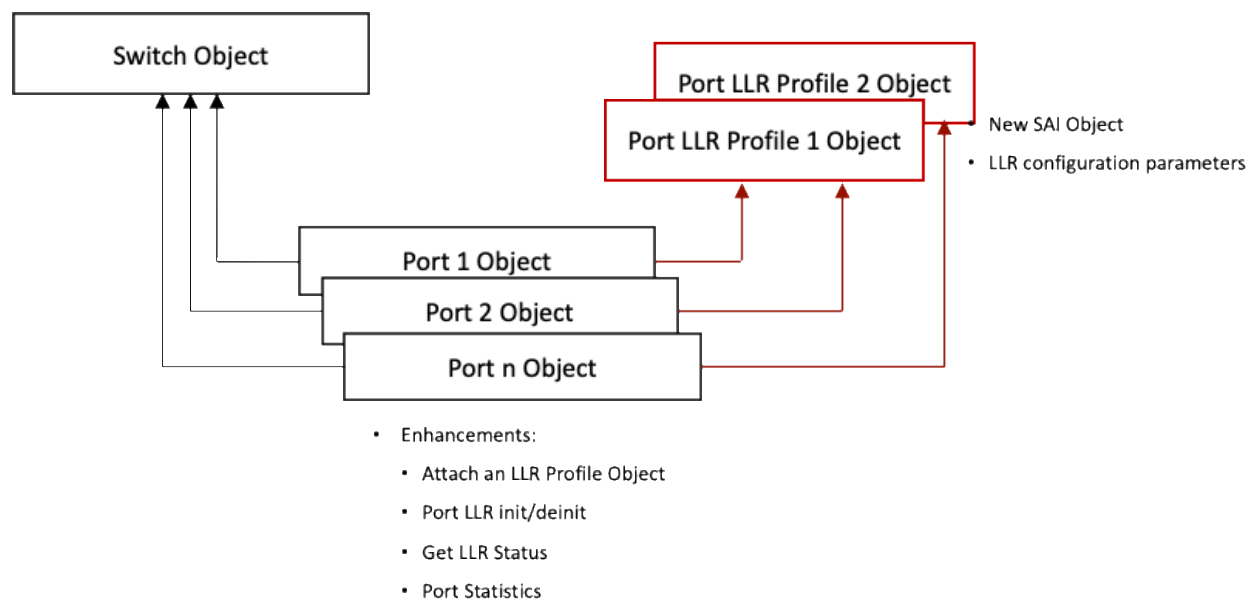


Figure 1-1 - Overview of SAI attributes for LLR.

1.3.1 SAI Logical Pipeline

LLR does not need any changes in the SAI logical pipeline.

1.3.2 Port LLR Profile

Port LLR Profile contains LLR attributes associated with a port. These attributes are listed in the table below.

Table 1-1 - Attributes of the new SAI PORT LLR PROFILE object type

SAI_PORT_LLRLR_PROFILE_ATTR	Type	Flags	Default	Description
OUTSTANDING_FRAMES_MAX	sai_uint32_t	MANDATORY_ON_CREATE CREATE_ONLY	NA	Limit on the number of unacknowledged frames. LLR Spec reference: outstanding_seq_max
OUTSTANDING_BYTES_MAX	sai_uint32_t	MANDATORY_ON_CREATE CREATE_ONLY	NA	Limit on the number of unacknowledged bytes. LLR Spec reference: outstanding_data_max
REPLAY_TIMER_MAX	sai_uint32_t	CREATE_AND_SET	0	A configuration parameter that sets the maximum timer value. When this timer expires, a replay is initiated. The value is specified in nanoseconds. LLR Spec reference: replay_timer_max
REPLAY_COUNT_MAX	sai_uint8_t	CREATE_AND_SET	1	Limit on the number of replays if no further LLR_ACKs are received. Once the limit is reached, the port enters the 'FLUSH' state. LLR Spec reference: replay_ct_max
PCS_LOST_TIMEOUT	sai_uint32_t	CREATE_AND_SET	0	If the port PCS status is FALSE for this duration, the port enters the 'FLUSH' state. The value is specified in nanoseconds. LLR Spec reference: pcs_lost_status_timer_max
DATA_AGE_TIMEOUT	sai_uint32_t	CREATE_AND_SET	0	Limit of the time a frame can reside in the replay buffer. The value is specified in nanoseconds.

SAI_PORT_LLRLR_PROFILE_ATTR	Type	Flags	Default	Description
				LLR Spec reference: data_age_timer_max
INIT_LLRLR_FRAME_ACTION	typedef sai_llr_frame_action_t (Refer Table 1-2 - Frame action types)	CREATE_AND_SET	BEST_EFFORT	The action to be taken on frames received from the higher layers when the LLR mechanism enters the "INIT" state. LLR Spec reference: llr_init_behavior
FLUSH_LLRLR_FRAME_ACTION	typedef sai_llr_frame_action_t (Refer Table 1-2 - Frame action types)	CREATE_AND_SET	BEST_EFFORT	The action to be taken on frames received from the higher layers when the LLR mechanism enters the "FLUSH" state. LLR Spec reference: llr_flush_behavior
RE_INIT_ON_FLUSH	bool	CREATE_AND_SET	false	Enable the LLR mechanism to automatically re-initialize the port LLR from the "FLUSH" state instead of manual intervention to re-activate LLR. LLR Spec reference: re_init_on_flush
CTLOS_TARGET_SPACING	sai_uint16_t	CREATE_AND_SET	2048	Desired spacing between CtIOS messages in bytes between transmission of successive LLR_ACK / LLR_NACK CtIOS. LLR Spec reference: ctlos_target_spacing

Table 1-2 - Frame action types

typedef sai_llr_frame_action	Description
SAI_LLRLR_FRAME_ACTION_DISCARD	Frames are dropped and SAI_PORT_STAT_IF_OUT_DISCARDS increments.
SAI_LLRLR_FRAME_ACTION_BLOCK	Blocks transmission requests from higher layers to the MAC.
SAI_LLRLR_FRAME_ACTION_BEST_EFFORT	Frames are treated as regular packets i.e., frames are not placed in the replay buffer and not assigned a sequence number.

1.3.3 SAI Port Updates

New attributes are added to the existing SAI Port object to attach a LLR Profile to the Port, enable LLR, and read LLR status.

Table 1-3 - New Attributes of the existing SAI PORT object type

SAI_PORT_ATTR	Type	Flags	Default	Description
LLR_MODE_LOCAL	bool	CREATE_AND_SET	FALSE	Enable the port to start receiving LLR INIT messages. LLR Spec reference: llr_mode_local
LLR_MODE_REMOTE	bool	CREATE_AND_SET	FALSE	Enable the port to start sending LLR INIT messages. LLR Spec reference: llr_mode_remote
LLR_PROFILE	@objects SAI_OBJECT_TYPE_PORT_LLRLR_PROFILE	CREATE_AND_SET	SAI_NULL_OBJECT_ID	The LLR profile to configure the port's LLR mechanism. Mandatory when LLR_RX_ENABLE and/or LLR_TX_ENABLE is TRUE
LLR_TX_STATUS	typedef sai_port_llr_tx_status_t (Refer Table 1-4 - sai_port_llr_tx_status_t)	READ_ONLY	NA	Port LLR TX state
LLR_RX_STATUS	typedef sai_port_llr_rx_status_t	READ_ONLY	NA	Port's LLR RX state

SAI_PORT_ATTR	Type	Flags	Default	Description
	(Refer Table 1-5 - sai_port_llr_rx_status_t)			

Table 1-4 - sai_port_llr_tx_status_t

typedef sai_port_llr_tx_status_t	Description
SAI_PORT_LL_RX_STATUS_OFF	LLR_TX_ENABLE is FALSE
SAI_PORT_LL_RX_STATUS_INIT	LLR Tx is enabled and sending periodic INIT messages. Waiting for INIT_ECHO to be received from link partner
SAI_PORT_LL_RX_STATUS_ADVANCE	LLR Tx in normal operation, and no replay is in progress
SAI_PORT_LL_RX_STATUS_REPLAY	LLR Tx Replay is in progress
SAI_PORT_LL_RX_STATUS_FLUSH	LLR mechanism is in flush state

Table 1-5 - sai_port_llr_rx_status_t

typedef sai_port_llr_rx_status_t	Description
SAI_PORT_LL_RX_STATUS_OFF	LLR_RX_ENABLE is FALSE
SAI_PORT_LL_RX_STATUS_SEND_ACKS	LLR receive is in normal operation and sending ACKs. No missing/bad frames have been received.
SAI_PORT_LL_RX_STATUS_SEND_NACK	LLR detected a missing frame or received a bad frame, and is in the process of sending the NACK
SAI_PORT_LL_RX_STATUS_NACK_SENT	LLR sent a NACK and is waiting for the expected frame

1.3.4 New Port Error Status

When the SAI implementation detects a port state change, it uses `sai_port_error_status_t` to convey the operational status. Each enum in `sai_port_error_status_t` is a bit to encode an operational state. This enum is extended to encode the LLR state of the port.

Table 1-6 - Port Error Status

typedef sai_port_error_status_t	Description
SAI_PORT_ERROR_STATUS_LL_RX_FLUSH	Denote the port LLR TX mechanism entering/exiting flush state

1.3.5 New LLR Port Statistics

The following new port statistics needed for LLR.

Table 1-7 - Port LLR Statistics

SAI_PORT_STAT	Description
LLR_TX_INIT_CTL_OS	Number of LLR_INIT control ordered sets transmitted
LLR_TX_INIT_ECHO_CTL_OS	Number of LLR_INIT_ECHO control ordered sets transmitted
LLR_TX_ACK_CTL_OS	Number of LLR_ACK control ordered sets transmitted
LLR_TX_NACK_CTL_OS	Number of LLR_NACK control ordered sets transmitted
LLR_TX_DISCARD	Number of LLR-eligible frames discarded by the LLR TX when <ul style="list-style-type: none"> the TX state machine is in the INIT state, and the <code>llr_init_behavior</code> is set to DISCARD, or the TX state machine is in the FLUSH state, and the <code>llr_flush_behavior</code> is set to DISCARD.
LLR_TX_OK	Number of LLR-eligible frames transmitted with a good FCS
LLR_TX_POISONED	Number of LLR-eligible frames transmitted with a poisoned FCS
LLR_TX_REPLAY	Number of times that LLR Tx completed a replay operation

SAI_PORT_STAT	Description
LLR_RX_INIT_CTL_OS	Number of LLR_INIT control ordered sets received
LLR_RX_INIT_ECHO_CTL_OS	Number of LLR_INIT_ECHO control ordered sets received
LLR_RX_ACK_CTL_OS	Number of LLR_ACK control ordered sets received
LLR_RX_NACK_CTL_OS	Number of LLR_NACK control ordered sets received
LLR_RX_ACK_NACK_SEQ_ERROR	Number of LLR_ACK/LLR_NACK sequence number errors. The counter is incremented each time an LLR_ACK or LLR_NACK is received with an unexpected sequence number
LLR_RX_OK	Number of LLR-eligible frames received with a good FCS
LLR_RX_POISONED	Number of LLR-eligible frames received with a poisoned FCS
LLR_RX_BAD	Number of LLR-eligible frames received with a bad FCS
LLR_RX_EXPECTED_SEQ_GOOD	Number of LLR-eligible frames received with a good FCS and with the expected sequence number
LLR_RX_EXPECTED_SEQ_POISONED	Number of LLR-eligible frames received with a poisoned FCS and with the expected sequence number
LLR_RX_EXPECTED_SEQ_BAD	Number of LLR-eligible frames received with a bad FCS and with the expected sequence number
LLR_RX_MISSING_SEQ	Number of LLR-eligible frames received with a sequence number that indicated a missing LLR-eligible frame in the sequence, irrespective of FCS status
LLR_RX_DUPLICATE_SEQ	Number of LLR-eligible frames received with a duplicate sequence number, irrespective of FCS status
LLR_RX_REPLAY	Number of times that LLR Rx detected the start of a replay

1.3.6 Capability Query

1.3.6.1 Feature Capability

No new APIs or attributes are needed for Feature Capability Query. The existing attribute capability query API can be used to identify LLR capability.

```
sai_attr_capability_t attr_cap[1];
sai_query_attribute_capability(switch_id, SAI_OBJECT_TYPE_PORT,
SAI_PORT_ATTR_LLR_PROFILE, attr_cap);

if (attr_cap.set_implemented) {
    /* Enable LLR on supported platforms */
    sai_set_port_attribute_fn(sai_port_id, switch_id, 1, sai_attr_list);
} else {
    return -1; // LLR is not supported
}

sai_query_attribute_capability(switch_id, SAI_OBJECT_TYPE_PORT,
SAI_PORT_ATTR_LLR_MODE_LOCAL, attr_cap);
if (attr_cap.set_implemented) {
    /* Enable LLR Locally on supported platforms */
    sai_set_port_attribute_fn(sai_port_id, switch_id, 1, sai_attr_list);
}

sai_query_attribute_capability(switch_id, SAI_OBJECT_TYPE_PORT,
SAI_PORT_ATTR_LLR_MODE_REMOTE, attr_cap);
if (attr_cap.set_implemented) {
    /* Enable LLR remote on supported platforms */
    sai_set_port_attribute_fn(sai_port_id, switch_id, 1, sai_attr_list);
}
```

1.3.6.2 Statistics Capability Query

The existing statistics capability query can be used to identify supported port LLR statistics.

```
sai_stat_capability_list_t sai_port_stats_capability;
sai_port_stats_capability.count = SAI_PORT_STAT_END;
sai_stat_capability_t stats[SAI_PORT_STAT_END] = {};
sai_port_stats_capability.list = stats;
```

```

sai_query_stats_capability(switch_id, SAI_OBJECT_TYPE_PORT,
sai_port_stats_capability);

/* Fetching the supported LLR port statistics */

sai_port_stat_t supported_stat_ids[SAI_PORT_STAT_END] = {0};

for (int i = 0; i < sai_port_stats_capability.count; i++) {
    switch (sai_port_stats_capability.list[i].stat_enum) {
        case SAI_PORT_STAT_LL_R_TX_INIT_CTL_OS:
        case SAI_PORT_STAT_LL_R_TX_INIT_ECHO_CTL_OS:
            :
        case SAI_PORT_STAT_LL_R_RX_REPLAY:
            if (sai_port_stats_capability.list[i].stat_modes & SAI_STAT_MODE_READ) {
                supported_stat_ids[j++] = sai_port_stats_capability.list[i].stat_enum;
            }
            break;
        default:
            break;
    }
}
sai_get_port_attribute_fn(sai_port_id, switch_id, j, supported_stat_ids, counters);

```

1.4 Normative References

- [1] <https://github.com/opencomputeproject/SAI/pull/2225>
- [2] Chapter 5.1 of <https://ultraethernet.org/wp-content/uploads/sites/20/2025/06/UE-Specification-6.11.25.pdf>