NVM Express Workgroup
c/o VTM, Inc.
3855 SW 153rd Drive
Beaverton, OR 97003
USA
info@nvmexpress.org

NVM Express® Technical Proposal for New Feature

Technical Proposal ID	**TP 4129 – KATO Corrections and Clarifications**
Change Date	**2024/06/11**
Builds on Specification	**NVM Express Base Specification 2.0d** **NVM Express TCP Transport Specification 1.0d** **NVM Express RDMA Transport Specification 1.0c**
References Ratified TPs	**TP 4097a Abort Enhancements**
References ECNs	

Technical Proposal Author(s)

Name	**Company**
Fred Knight	NetApp
John Meneghini	RedHat
David Black	Dell Technologies
Costa Sapuntzakis, Randy Jennings	Pure

This Technical Proposal:

- Adds a description of host operation during a communication loss to avoid data corruption that could be caused by interaction of outstanding commands with subsequent commands submitted by that host. This includes:
  - a requirement to determine the controller has stopped processing commands that can change state on the controller before retrying or erroring out that command;
  - requirements on retrying such commands; and
  - examples of incorrect use of retries.
- Adds Command Quiesce Time (CQT) to allow for cleanup time for these outstanding commands on the controller.
- Specifies corrections in the use of the KATO (Keep Alive Time Out) operation. Specifically:
  - requires the controller to detect a KATO if the Keep Alive Timer expires;
  - describes interactions and timings associated with host and target use cases for KATO; and
  - expands and corrects an example implementation for Keep Alive operation on the host, illustrating how the implementation keeps the controller alive.

Revision History

Revision Date	Change Description
8/2/2021	Initial creation
9/27/2021	Add host usage information
9/28/2021	Initial FMDS comment incorporation
10/12/2021	Restructure

1/3/2022	Reorganize content based on feedback from last FMDS meeting
1/11/2022	Add QEMPTY, 2X+n information to expand the section with KATO value setting recommendations; including comments from FMDS meeting.
2022/07/11	Remove guidance around connection loss and KATO from 3.3.2.4 Remove QEMPTY Rewrite Keep Alive section to be more requirements oriented and less implementation description Add KASMG (Keep Alive Support Millisecond Granularity) and CQAT (Command Quiesce after Termination) Add section with guidance around association loss and when host can retransmit
2022/08/08	Remove KASMG Incorporate feedback from July 26, 2022 Keep Alive Feature -> Keep Alive Timer Feature Keep Alive Feature is active -> Keep Alive Timer is active
2022/08/16	Much cleanup from the FMDS call
2022/08/17	Specify worst-case bounds in host recovery procedure in 9.1.b
2022/08/30	Back off of worst-case bounds Add rationale for current bounds
2022/09/06	Keep Alive Timeout value time is now KATT CQAT is now CQT and covers time to initiate cleanup Describe why 2x KATT for Traffic Based Keep Alive
2022/09/23	Move end of 3.3.2.4 to its own section 3.3.2.5 and generalize Add reference to 9.1.TBD in 3.6.2 Add language to NVM Express TCP transport specification on what the host should consider indicating that a controller has seen a connection loss.
2022/09/28	Feedback from FMDS meeting Add figure for 2 * KATT in TBKAS Redo wording in 3.3.2.4 on controller behavior during Disconnect I/O Submission Queue – aborting and completing are not distinct. Abort is a kind of completion Delete sentence in 3.3.2.TBD about controller continuing to process commands. Sentence is repeated in next section.
2022/10/04	Feedback from FMDS meeting – wording changes in Disconnect, TBKAS, and 9.1.TBD Revise figure for TBKAS
2022/10/12	Talk about controllers and communications instead of associations and connections Fix the stricken text to reflect the current draft Apply comments from Samsung TBKAS – simplify description of when a Keep Alive Timeout occurs Revive wording about Connect Command can go through after controller cleans up – that seems to have disappeared
2022/10/17	Apply more comments from Samsung Add wording about when Keep Alive Timeout occurs at host for 3.9.1 and 3.9.2 Fix when Keep Alive Timeout occurs for Keep Alive-Based Keep Alive – old language says that only Keep Alive Command refreshes the timer ½ KATO -> 15 seconds Get rid of accounting for … language when describing how often the host should send a Keep Alive Command or command
2022/10/21	Simplify when a keep alive timeout occurs Give guidance on implementing Keep Alive Command based Keep Alive to controller implementor 15 seconds -> implementation specific time Revert 10/17 change on when Keep Alive Timeout occurs for Keep Alive Command-Based Keep Alive
2022/11/14	Added editorial comments explaining changes Updated navigation pane and fixed font sizes Updated according to some comments

	Refined revised text in 3.9 to show the changes from original text more clearly Added text in 9.1.TBD about when a commands execution status would be unknown (under construction).
2022/11/28	Revise description of when host detect Keep Alive Timeout Finish text in 9.1.TBD that generalizes the discussion to that of missing command completions Define Keep Alive Timeout occurs on controller for Traffic Based Keep Alive in terms of Keep Alive Timeout Interval
2022/12/05	Edits from FMDS meeting Revise host-based detection of Keep Alive Timeout for Traffic-based Keep Alive based on simpler Linux implementation Reword Keep Alive Timeout Interval in Traffic Based Keep Alive Controller to be easier to follow Add Controller Level Reset to conditions which stop command processing Clarify that RDY gets set to 0 only after command processing is stopped and queues are deleted
2022/12/07	Reframe host detection in terms of an example host implementation Remove changes to 3.7.2 – deemed unnecessary and harmful Minor edits from FMDS
2022/12/13	Tune host detection language using feedback from FMDS Replace admin command or I/O command with just command Recommend that host start using new value of KATT immediately after Set Features
2022/12/14 2022/12/19	More language tuning – get rid of pronouns, use submit more instead of send
2023/01/05	Language tuning from FMDS review on 12/20
2023/01/17	Add new line 3.3.2.4 clarifying that the controller shall stop processing all other commands before replying to disconnect command Mention disconnect in 9.1.TBD as a way host can know that commands have completed processing Minor word tuning in traffic based keep alive
2023/01/30	Added section headers and some context for Identify figure and section in TCP spec. Reduced the text of the change in 3.3.2. Added Abort/Cancel immediate to list of situations host knows command processing is done. Modified some of the explanatory comments in 3.9; still working on it
2023/05/08	Rebased on top of ECN117, KATO clarifications, with appropriate revisions Major rewrite of the Host-based algorithm
2023/05/15	Fixed Keep Alive Timer Activation text copied from ECN117 (missed removing red text). For TBKA on the Host, changed the reason for checking for command completions at KATT/4 to the more concrete “prevent the controller from detecting a Keep Alive Timeout” (existing text). “-“ -> “minus” (“-“ was easy to confuse with a dash instead of a minus.) Added comments for issues the FMDS group wanted to return to.
2023/05/19	“If using” -> “For” for CBKA/TBKA controller/host sections (update from ECN117) “to account for delays (e.g.,” when explaining why KATT/2 (update from ECN117) Introduced Keep Alive Send function for the host (that gets activated/deactivated) so we can talk about starting/stopping/expiring “a” Keep Alive Send Timer, and so the function can stay active while a timer is not running. Also, associated Keep Alive Send function with a controller on the host; the host has one for each controller to which it is connected. Fixed antecedent issues in Host algorithm descriptions. Qualify the host looking for a completion on a Keep Alive command with if the Keep Alive command was sent. For TBKA on host, tighten up language in normative text around checking for commands at the rate KATT/4
2023/05/23	Keep Alive Send function -> timer, but enable it instead of activate it (meaning of function was not clear) Host detecting KATO: “may” -> “in this example”

2023/05/31	In section 3.3.2.4 “initiates or detects an NVMe Transport connection loss” -> “terminates an NVMe Transport connection or detects an NVMe Transport connection loss” In section 3.9.TBDcleanup, cleaned up intro (first part is about controller detecting KATO; “cleanup actions” -> “actions”) In section 9.1.TBD: * Limit Cancel command discussion to just canceling specific command, and add note about this not being in Base 2.0c spec. * CSTS.RDY/ CSTS.SHST/Disconnect do not cover fabric commands. * Host needs to wait for a controller to be able to do something, not for it to finish doing something. * Other editorial changes & reference additions.
2023/06/06	* Added TP 4097a Abort Enhancements as a referenced TP * For Host-based Keep alive algorithms, Keep Alive Send Timer text “enables/disables” -> “becomes enabled/disabled” * Section 9.1.TBD host error recovery, “controller to be able to detect/stop processing” -> “ensure the controller has detected/stopped processing”
2023/06/16	* “Keep Alive Send Timer associated with a controller in the host” -> “Keep Alive Send Timer associated with a controller on the host” * “Keep Alive Send Timer in the host” -> “Keep Alive Send Timer” * “Keep Alive Timer in the controller” -> “Keep Alive Timer on the controller” * When discussing how a host knows if processing a command has finished: ** clarified that a host reading CSTS.RDY of ‘0’ indicates a Controller Level Reset is complete () ** “CQE” -> “completion” ** “that command” -> “the command which status is in question” * When discussing when a host will not know if processing a command has finished, put transport-level disconnects last and clarified text. * When describing how to calculate the time the host needs to wait, refer to the new figure for TBKA instead of the section (3.9.2). * Added Description of changes for Base & TCP spec * Added that this document depends on ECN117 in the header. * Revised TP Overview.
2023/06/21	* A few ‘when’ to ‘if replacements. There are still some ‘when’s in the text; they have an element of time in them, so I think they belong. * Revising current pointers to 9.1.TBD to be more dire and insistent. * Adding pointers to 9.1.TBD in these sections: ** 3.3.1.3 Message-based I/O Queue Abort ** 3.6.1 Memory-based Controller Shutdown ** 3.7.2 Controller Level Reset * Now that we have more sections pointing at 9.1.TBD, also pointed at 9.1.TBD instead of a couple of messaging sections when defining CQT in the Identify Controller Data Structure.
2023/07/14	* rebasing on most recent ECN117 version (2023/07/13) * Description of changes -> past tense verbs * fixed section numbering for CBKA/TBKA on controller
2023/08/02	* Added highlighting for 9.1.TBD/3.9.TBDactivate references * Added reference for Set Features figure with KATO field to KATT * brought forward a couple of member review changes from ECN117 that were missed. * minor editorial (spelling correction: ‘Iff -> ‘If’, verb fix: ‘processed’ -> ‘was processed’
2023/08/23	Incorporated comments from ECN117 integration, including some fixes to this TP: * controller KATO algorithms reworded for “shall” (checked; Keep Alive & Set Features do not have “shall” elsewhere for restarting the Keep Alive Timer.)

	* a couple more host submits instead of sends * qualify CBKA & TBKA use on controller with support for Keep Alive Timer feature.
2023/08/29	From FMDS, receiving Keep Alive & Set Features restarts timer only when Keep Alive Timer is active.
2023/09/29	* From ECN117 integration, 3.9 Keep Alive intro: “a watchdog timer intended to detect” -> drop “intended” * add pronunciation hint for KATT * Adding a way to tell a command is completed for message based transport, resuming communication with a controller. * clarified a Keep Alive Send Timer exist on the host for each controller to which it is connected. * cleaned up text around enabling/disabling Keep Alive Send Timer.
2023/10/20	* Added pointer to outstanding command handling to RDMA Binding specification. * Added note about processing of command is different than background processing triggered by a command.
2023/10/26	* Removed redundant “time” following “CQT”. * From list of events that mark that there is no more processing happening on a command, reworded to get rid of “in question”. * Reworded text about lack of ordering of command execution.
2023/11/14	* Handle comments from David Black * Revise host sections, primarily to provide more explanation of what the host needs to do and how the example does it, clearer compare/contrast of CBKA & TBKA example algorithms, active voice steps in algorithm, plus minor grammatical changes. * Remove redundant “the controller shall” from controller TBKA algorithm. * Revised clarifying text on how long it takes a controller using TBKA to detect a keep alive. (also, controller receive -> controller receive and fetch) * Explain what “quiesce” means when used in CQT. * Clarify text explaining when the host can tell a command has finished processing (outstanding command, text around host controlling dependent commands). * Clarify text explaining determining communication lost. * Revised text about CQT in recovery section, and calculated -> specified.
2023/11/21	* Added references to this TP in the “Description of Changes” sections. * Added yellow highlighting to a figure reference * Trying past tense for referencing the value to which Keep Alive Send Timer was set when the Keep Alive Send Timer expires. * Fixed more “last” -> “most recent”. * TP4097 was not ratified after NVMe Base 2.0 spec. * Communication loss ‘on’ -> ‘with’ a queue. * time indicated in the ‘CQT’ -> ‘CQT field’ * A couple of minor punctuation/capitalization fixes.
2023/11/29	* Updated summary to identify corruption we are trying to prevent and added rationale for the other changes. * Revised TBKA diagram to refer to most recently fetched command. Made it Word shapes so others can edit the diagram. Note, diagram word placement looks wrong when you hide footers and spaces between pages. * For detection of communication loss in 9.1.TBD, added “either” before “with controller” and “with queue”. * Moved Description of Changes for TCP/RDMA doc to top of doc.
2023/12/22	Initial draft text for Dell Member Review comments – section 9.1.TBD on “Handling Outstanding Commands during a Communication Loss” split into two sections in order to add material on how to handle outstanding commands during a communication loss

2024/01/12	For section 9.1.TBD: * Added back reference to 3.4.1 about command ordering, but reduced context text around it. * Moved host command dependency/retry text to an annex. * Revised it to focus on deciding to retry. * Added examples of what can go wrong when retrying. * Updated copyright notices to 2024.
2024/01/19	Comments from FMDS: * Added definition for idempotent commands * Disconnect: controller does not process -> ensures no processing * In the new annex section, * When talking generally about a missing response, used “missing response” or “response not received” instead of “lost response”. A specific example still refers to a “lost response”. * Reworded example “This shows how” sentence to describe what is required first and then state the type of command being retried. * Added third controller to 2 host example * Made some misc. ordering & editorial changes.
2024/01/22	* Removed stray words on title page (same line as NVMe graphic) * fixed Figure label & text referring to it (Detecting Timeout Takes up to 2 * KATT) Comments from Mike Allison & Judy Brock: * Removed section references in description of changes * Identified hosts more correctly (specifically or broadly, depending on context) * Got rid of a couple more ~~Keep Alive~~ Command Based Keep Alive instances. * Made some small word, punctuation & rearranging changes.
2024/01/29	Comments from Mike Allison & Judy Brock, refined by FMDS meeting: * copy over comments about integration issue (duplicate ‘the’) and concern about delayed command can cause corruption without interactions with other commands (internal memory buffers) * Refined pointer to the new error handling section (9.1.TBD); ‘could’ -> ‘is possible’, and move reference to a parenthetical comment in the corruption threat sentence * Revised some Set Features text (going back to not qualifying timer modification by new KATO value being lower, providing more context in host example) * Rearranged bullet order in host examples. * Labeled periods in TBKA diagram * Defined quiesce parenthetically instead of in a trailing sentence * Removed ‘from host’ in transport specific command completion ‘events’ * Removed trailing command completion ‘events’ sentence and moved background operations warning to own paragraph * Explained where ‘two delays’ came from. * Fixed TCP Transport Error Handling section number * Added new changes (definition of idempotent, ensures no processing) to Description of Changes. * Removed editorial fixes from TCP spec and filed https://bugzilla.nvmexpress.org/show_bug.cgi?id=283
2024/01/30	* Moved reference to 3.9.5 when host detects a Keep Alive Timeout from example text to normative text (at beginning of Host CBKA/TBKA sections). * Added categorizing of commands to section 9.1.TBD and added “shall” instructions for the host based on them. * Added stronger “this is able to cause corruption” language. * Changed “implementation specific” -> “higher level software” for handling non-retriable requests.

	* Revised pictures in annex to be more clear about host determining controller has stopped processing commands Moved to base 2.0d * Rectified some typos in text and updated section/figure numbers. * ECN117 was integrated, updated TBD section numbers, removed coloring.
2024/02/05	* Added requirement to not retry or error out command until host ensures processing has stopped to Technical Proposal summary * Updated Description of changes * “modify state on the controller” -> “modify state in the subsystem” * Minor editorial and figure numbering updates
2024/02/06	* Move most of idempotent definition content and command retry categories to a new error handling section. * Reworked pointer to 9.1.TBD to state requirement on host * “submitted by a host” -> “submitted by a host on another controller” * “no processing” -> “no further processing” * clarify command being referenced in a few places.
2024/02/13	* Extensive rework of section 9.* material to cleanly separate concepts: -- loss detection, end of command processing, and command retry. * Add some section 3.4 content to support FMDS discussion (see comments) * Refer all mention of host reaction to communication loss to section 9. * Edits and comments (e.g., editing instructions) from FMDS discussion.
2024/02/16	Correct date in previous entry, accept all changes, plus: * Editorial and terminology changes from FMDS discussion * Moved text on when a command is not outstanding to new 3.4.TBD section * Extensive editing of Annex B examples to line up with 2/13 changes * Added text to strongly discourage using commands such as Write and Set Features for multi-host coordination and synchronization.
2024/02/20	Editorial changes from FMDS discussion, primarily to section 9.TBD material.
2024/02/27	Editorial changes from FMDS discussion, primarily to Annex B material.
2024/03/05	Editorial cleanup. Add a sentence to Annex B to point out that Compare and Write is not idempotent (even though its components are). * Revised technical proposal summary to include text about retries * Revised description of changes for structural changes relating to outstanding commands and retries. * Minor edits on retry example figures * Add a PCIe example for loss of communication
2024/03/06	Editorial changes from Mike Allison and the document editor. Pick up additional keep alive text from Base Spec that is (or is not) being modified by this TP.
2024/03/12	Additional editorial changes.
2024/03/14	Clean version for second member review
2024/06/11	Integrated

Markup Conventions:

Black:	Unchanged (however, hot links are removed)
~~Red Strikethrough~~:	Deleted
Blue:	New
Blue Highlighted:	TBD values, anchors, and links to be inserted in new text.
<Green Bracketed>:	Notes to editor
Orange:	Changes made by a referenced TP
Purple:	Moved (to this location)
~~Purple~~:	Moved (from this location)

Description for Changes Document for NVM Express Base Specification 2.0d

- **Handling Outstanding Commands during a Communication Loss**
  - **Feature Enhancements:**
    - Added **Command Quiesce Time (CQT)**, which the controller reports as the time it takes to stop processing commands after the controller detects a communication loss with the host.
  - **Required Changes:**
    - Described, in a new section, error recovery if a communication loss happens and there are outstanding commands, specifically:
      - how a host detects loss of communication with a controller;
      - how a host determines no further controller processing of outstanding commands is possible; and
      - restrictions for using retries of a command for recovery.
  - **References:**
    - TP 4129 KATO Corrections and Clarifications
- **Keep Alive Corrections**
  - **Feature Enhancements:**
    - Expanded the host examples of CBKA and TBKA, focusing them on sending Keep Alive commands to keep the controller alive, covering more situations.
  - **Required Changes:**
    - Clarified that the concepts of Keep Alive Timer and Keep Alive Timer Interval apply to the controller, not directly to the host.
    - For TBKA, changed the interval the host checks to send a Keep Alive from KATT/2 to KATT/4.
  - **Backwards Incompatible:**
    - If a Keep Alive Timer expires, the controller is required to detect a Keep Alive Timeout.
  - **References:**
    - TP 4129 KATO Corrections and Clarifications

1 Introduction

1.5 Definitions

1.5.TBD idempotent command

A command that produces the same end state in the NVM subsystem and returns the same results if that command is resubmitted one or more times with no intervening commands. Refer to section 9.TBD.3.1.

3 NVM Express Architecture

3.3 NVM Queue Models

3.3.1 Memory-based Transport Queue Model

3.3.1.3 Queue Abort

To abort a large number of commands, the host may use:

- the Cancel command (refer to section 7.TBD); or
- delete and recreate the I/O Submission Queue (refer to section 3.7.3).

Specifically, to abort all commands that are submitted to an I/O Submission Queue, host software should:

- issue a Cancel command to that queue with the Cancel Action set to Multiple Command Cancel and the NSID field set to FFFFFFFFh; or
- issue a Delete I/O Submission Queue command for that queue. After that submission queue has been successfully deleted, indicating that all commands have been completed or aborted, then host software should recreate the queue by submitting a Create I/O Submission Queue command. Host software may then re-submit commands to the associated I/O Submission Queue.

If the host is no longer able to communicate with the controller before that host receives either:

- completions for all outstanding commands submitted on that I/O Submission Queue (refer to section 3.4.TBD); or
- a successful completion for the Delete I/O Submission Queue command for that I/O Submission Queue,

then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

3.3.2 Message-based Transport Queue Model

3.3.2.4 I/O Queue Deletion

NVMe over Fabrics deletes an individual I/O Queue and may delete the associated NVMe Transport connection as a result of:

- the exchange of a Disconnect command and response (refer to section 6.4) between a host and controller; or
- the detection and processing of a transport error on an NVMe Transport connection.

…

A Disconnect command is the last I/O Submission Queue entry processed by the controller for an I/O Queue. Controller processing of the Disconnect command completes or aborts all commands on the I/O Queue on which the Disconnect command was received. The controller determines whether to complete or abort each of those commands. Until the controller ~~detects an NVMe Transport connection loss or~~ sends a successful completion for a Disconnect command, outstanding commands may continue being processed by the controller. The controller ensures that there is no further processing of any command sent on that I/O Queue after posting the completion queue entry for the Disconnect command as described in section 6.4.

The response to the Disconnect command is the last I/O Completion Queue entry processed by the host for an I/O Queue. To avoid command aborts, the host should wait for all outstanding commands on an I/O Queue to complete before sending the Disconnect command.

If the controller terminates an NVMe Transport connection or detects an NVMe Transport connection loss, then the controller shall stop processing all commands received on the I/O Queues associated with that NVMe Transport connection within the time reported in the CQT field (refer to Figure 276), if non-zero. ~~Until the controller detects an NVMe Transport connection loss or sends a successful completion for a Disconnect command, outstanding commands may continue being processed by the controller.~~

If the host terminates an NVMe Transport connection or detects an NVMe Transport connection loss before the responses are received for all outstanding commands submitted to the associated I/O Queue (refer to section 3.4.TBD), then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller. ~~then there is no further information available to the host about the state of those commands (e.g., each individual outstanding command may have been completed or aborted by the controller).~~

~~If an NVMe Transport connection is lost as a result of an NVMe Transport error, then before performing recovery actions related to commands sent on I/O queues associated with that NVMe Transport connection, the host should wait for at least the longer of:~~

- ~~the NVMe Keep Alive timeout; or~~
- ~~the underlying fabric transport timeout, if any.~~

3.4 Command Processing ~~Architecture Submission and Completion Mechanism~~

3.4.TBD Outstanding Commands

A command is outstanding if:

- the host has submitted that command to the controller;
- the host has not received a completion for that command; and
- as described in this section:
  - the host has not performed an action that causes that command to no longer be outstanding; and
  - the host has not otherwise determined that that command is no longer outstanding.

A submitted command is no longer outstanding after the host:

- receives a completion for that command;
- receives a successful completion with Immediate Abort Not Performed bit cleared to '0' in Dword 0 of the completion queue entry for an Abort command specifying that outstanding command (refer to section 5.1);
- receives a successful completion with the Commands Aborted field set to 1h for a Cancel command with an Action Code of Single Command Cancel and specifying that outstanding command (refer to section 7.TBD); <7.TBD referenced from TP4097a>
- reads a CSTS.RDY bit value that indicates a controller is not able to process commands except for Fabrics commands (i.e., a value of '0'), if that outstanding command is not a Fabrics command (refer to section 3.7.2);
- reads a CSTS.SHST field value that indicates that the controller shutdown is complete (i.e., a value of 10b), if that outstanding command is not a Fabrics command (refer to section 3.6.1 for memory-based transports and section 3.6.2 for message-based transports);
- if using a memory-based transport, receives a successful completion for a Delete I/O Submission Queue command if that outstanding command was sent on the deleted I/O Submission Queue (refer to section 3.3.1.3);

- if using a message-based transport:
  - receives a successful completion for a Disconnect command if that outstanding command was sent on the same I/O queue as the Disconnect command (refer to section 3.3.2.4); or
  - restores communication to the same controller after losing communication to that controller (refer to section 3.9.5).

If an outstanding command ceases to be outstanding for one of these reasons, then further controller processing of that command is no longer possible.

…

3.6 Shutdown Processing

3.6.1 Memory-based Transport Controller Shutdown

…

It is an implementation choice whether the host aborts all outstanding commands to the Admin Queue prior to the controller shutdown. The only commands that should be outstanding to the Admin Queue when the controller reports shutdown processing complete are Asynchronous Event Request commands**.**

If the host is no longer able to communicate with the controller before that host receives either:

- completions for all outstanding commands submitted to that controller (refer to section 3.4.TBD); or
- a CSTS.SHST field value that indicates that the controller shutdown is complete,

then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

3.6.2 Message-based Transport Controller Shutdown

…

After the host initiates a controller shutdown, the host may either disconnect at the NVMe Transport level or the host may choose to poll CSTS.SHST to determine when the controller shutdown is complete (i.e., the controller should not initiate a disconnect at the NVMe Transport level). It is an implementation choice whether the host aborts all outstanding commands prior to initiating the shutdown.

If the host is no longer able to communicate with the controller before that host receives either:

- completions for all outstanding commands submitted to that controller (refer to section 3.4.TBD); or
- a CSTS.SHST field value that indicates that the controller shutdown is complete,

then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

The CC.EN field is not used to shutdown the controller (i.e., it is used for Controller Reset).

…

3.7 Resets

3.7.2 Controller Level Reset

…

Note that all Controller Level Reset cases except a Controller Reset result in the controller immediately losing communication with the host. In all these cases, the controller is unable to indicate any aborts or update any completion queue entries.

If the host is no longer able to communicate with the controller before that host receives either:

- completions for all outstanding commands submitted to that controller (refer to section 3.4.TBD); or
- a CSTS.RDY bit value cleared to '0',

then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

3.9 Keep Alive

The Keep Alive capability uses the Keep Alive Timer on a controller as a watchdog timer ~~intended~~ to detect communication failures (e.g., transport failure, host failure, or controller failure) between a host and a controller. If the Keep Alive Timer feature is supported (i.e., the KAS field is set to a non-zero value (refer to Figure 276)), the controller shall support the Keep Alive command.

The term Keep Alive Timeout Time (KATT) refers to the time indicated by the value of the Keep Alive Timeout field on a controller (refer to Figure 342).

The NVMe Transport binding specification for the associated NVMe Transport defines:

…

3.9.2 Keep Alive Timer Activation

The Keep Alive Timer is active if:

- CC.EN is set to '1';
- CSTS.RDY is set to '1';
- CC.SHN is cleared to '00b';
- CSTS.SHST is cleared to '00b'; and
- The Keep Alive Timer feature is enabled as a result of the KATO field being set to a non-zero value (refer to section 3.9.1).

Otherwise, the Keep Alive Timer is inactive, and a Keep Alive Timeout as described in sections 3.9.3.1 and 3.9.4.1 shall not occur on the controller. Activating an inactive Keep Alive Timer (e.g., a Set Features command successfully ~~sets~~ setting the Keep Alive Timeout value to a non-zero value from a value of 0h, or the host enabling a controller that supports NVMe over Fabrics where the Connect command specified a non-zero Keep Alive Timeout value (refer to Figure 381)) shall initialize the Keep Alive Timer to the Keep Alive Timeout value.

…

3.9.3 Command Based Keep Alive

For Command Based Keep Alive, the Keep Alive command is sent periodically from the host to the controller ~~over~~ on the Admin Queue. The completion of the Keep Alive command indicates that the host and controller are able to communicate. For message-based transports, the Keep Alive Timeout is the maximum time an association remains established without processing a Keep Alive command.

3.9.3.1 Command Based Keep Alive on the Controller

The controller is using Command Based Keep Alive if the controller supports the Keep Alive Timer feature and the TBKAS bit is cleared to '0' in the ~~CTRATT~~ Controller Attributes field in the Identify Controller data structure (refer to Figure 276).

For Command Based Keep Alive:

- The controller shall start t~~T~~he Keep Alive Timer ~~in the controller starts when~~if the Keep Alive Timer becomes active (refer to section 3.9.2).
- The controller shall restart t~~T~~he Keep Alive Timer ~~in the controller restarts~~ if the Keep Alive Timer is active, and:
  - a Keep Alive command completes successfully; or
  - a Set Features command specifying the Keep Alive Timer feature and a non-zero KATO field (refer to section 5.27.1.12) completes successfully.
- The controller shall expire t~~T~~he Keep Alive Timer ~~in the controller expires~~ if:
  - the Keep Alive Timer is active on the controller; and
  - ~~time equal to the Keep Alive Timeout value~~KATT has elapsed since the Keep Alive Timer was most recently started or restarted.

If the Keep Alive Timer ~~i~~on the controller expires then the controller ~~may~~ shall consider a Keep Alive Timeout to have occurred. Upon the occurrence of a Keep Alive Timeout, the controller shall perform the cleanup actions described in section 3.9.5.

3.9.3.2 Command Based Keep Alive on the Host

The host may use Command Based Keep Alive regardless of the Keep Alive mode used by the controller. To prevent the controller from detecting a Keep Alive Timeout during the use of Command Based Keep Alive on the host, the host should send Keep Alive commands at ~~half of the time equal to the Keep Alive Timeout value~~KATT/2 to account for delays (e.g., transport round-trip times, transport delays, command processing times, and the Keep Alive Timer granularity) while the Keep Alive Timer is active on the controller. If the host receives a successful completion to a Set Features command for the Keep Alive feature, then the host should adjust the time at which the host sends the next Keep Alive command because the controller restarts the Keep Alive Timer.

If a host detects a Keep Alive Timeout and has outstanding commands for which that host has not received completions (refer to section 3.4.TBD), then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

For an example host implementation of Command Based Keep Alive, the host maintains a Keep Alive Send Timer for each controller to which the host is connected. The host uses the Keep Alive Send Timer to track when the host sends a Keep Alive command to the corresponding controller. The host does not know when the controller fetches the Keep Alive command. Conservatively, the host assumes the controller fetches the Keep Alive command immediately upon the host sending the Keep Alive command. The host tracks this time as the last expired timestamp of the Keep Alive Send Timer, for use when starting or restarting the Keep Alive Send Timer. While the host does restart the Keep Alive Send Timer after a successful Set Features command for the KATO feature, the host does not change the last expired timestamp when sending that Set Features command because at that point in time the host does not know the results of that command.

This example host implementation of Command Based Keep Alive behaves as follows:

- The host enables the Keep Alive Send Timer if the host requests activation of the Keep Alive Timer on the controller (e.g., the host enables the controller (refer to section 3.9.2)). The host records the current time as the last expired timestamp when the Keep Alive Send Timer becomes enabled.
- The host disables the Keep Alive Send Timer if the host requests deactivation of the Keep Alive Timer on the controller and the host receives a successful response from the controller (e.g., the host disables the controller (refer to section 3.9.2)).
- The host starts a Keep Alive Send Timer if:
    - the Keep Alive Send Timer becomes enabled; or
    - a successful Keep Alive command completion is processed by the host.
- The host restarts a Keep Alive Send Timer if:
    - the Keep Alive Send Timer is enabled;
    - a Keep Alive command is not outstanding; and,
    - a successful Set Features command completion is processed by the host where the command specified the Keep Alive Timer feature (i.e., Feature Identifier 0Fh) and a non-zero KATO field.
- If a Keep Alive Send Timer starts or restarts, the host sets the Keep Alive Send Timer to:
    - KATT/2 minus the time elapsed since the last expired timestamp; or
    - zero, if the time elapsed since the last expired timestamp is greater than KATT/2.
- The host stops a Keep Alive Send Timer if the Keep Alive Send Timer becomes disabled.
- If a Keep Alive Send Timer expires, (i.e., the Keep Alive Send Timer is still enabled and time has elapsed equal to the value to which the Keep Alive Send Timer was set since the host most recently started or restarted the Keep Alive Send Timer), then the host records the current time as the last expired timestamp and sends a Keep Alive command.
- The host detects a Keep Alive Timeout if the host sends a Keep Alive command and does not receive a completion for the Keep Alive command before KATT elapses from when the Keep Alive command was sent.

~~For Command Based Keep Alive:~~

- ~~The Keep Alive Timer in the host starts or restarts if:~~
    - ~~a Set Features command with Feature Identifier 0Fh and a non-zero Keep Alive Timeout (KATO) field was submitted to the Admin submission queue; or~~
    - ~~a Keep Alive command was submitted to the Admin submission queue.~~
- ~~The Keep Alive Timer in the host expires if:~~
    - ~~time equal to the Keep Alive Timeout value has elapsed since the Keep Alive Timer was most recently started or restarted; and~~
    - ~~a completion of a Keep Alive command has not been received since the Keep Alive Timer was most recently started or restarted.~~

~~If the Keep Alive Timer in the host expires, then the host may consider a Keep Alive Timeout to have occurred~~. ~~If a host detects a Keep Alive Timeout, the host should perform the actions described in section 3.9.5.~~

3.9.4 Traffic Based Keep Alive

Traffic Based Keep Alive allows the host and controller to avoid a Keep Alive Timeout in the presence of Admin or I/O command processing without sending Keep Alive commands.

3.9.4.1 Traffic Based Keep Alive on the Controller

If the controller supports the Keep Alive Timer feature, then~~Controller~~ support for Traffic Based Keep Alive is indicated by the TBKAS bit in the Controller Attributes field in the Identify Controller data structure (refer to Figure 276). If the Controller does not support Traffic Based Keep Alive (i.e., the TBKAS bit is cleared to ‘0’), then the operation of the Keep Alive Timer feature is described in section 3.9.3.

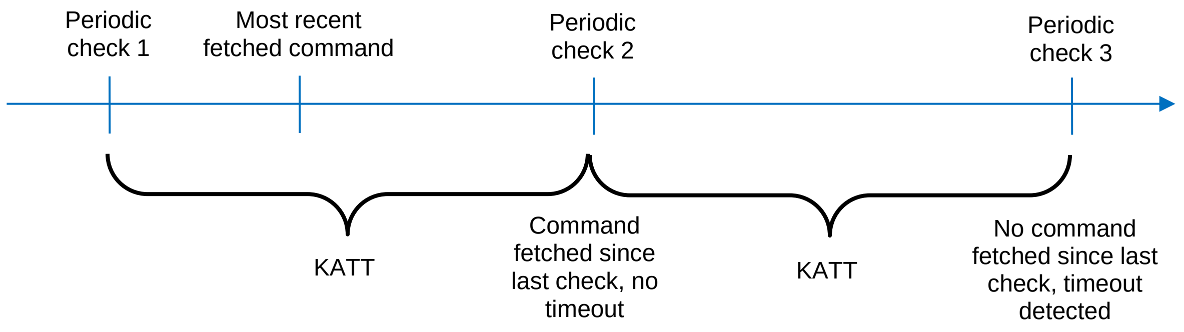
For Traffic Based Keep Alive:

- The controller shall start a~~A~~ Keep Alive Timeout Interval ~~on the controller starts when~~if the Keep Alive Timer becomes active (refer to section 3.9.2).
- The controller ~~may~~shall consider a Keep Alive Timeout to have occurred if:
  - the Keep Alive Timer is active;
  - ~~time equal to the Keep Alive Timeout value~~ KATT has elapsed since the start of the most recent Keep Alive Timeout Interval; and
  - no Admin command or I/O command was fetched by the controller in the Keep Alive Timeout Interval.
- The controller shall end a~~A~~ Keep Alive Timeout Interval ~~on the controller ends~~ and:
  - not start a new Keep Alive Timeout Interval ~~does not start~~ if:
    - a Keep Alive Timeout occurs; or
    - the Keep Alive Timer becomes inactive (refer to section 3.9.2).
  - start a new Keep Alive Timeout Interval ~~starts~~ if the Keep Alive Timer is active, and:
    - a Keep Alive command completes successfully;
    - a Set Features command specifying the Keep Alive Timer feature and a non-zero KATO field (refer to section 5.27.1.12) completes successfully; or
    - ~~time equal to the Keep Alive Timeout value~~KATT has elapsed since the start of the Keep Alive Timeout Interval and a Keep Alive Timeout did not occur in the Keep Alive Timeout Interval (e.g., an Admin command or an I/O command was fetched by the controller during the Keep Alive Timeout Interval).

Upon the occurrence of a Keep Alive Timeout, the controller shall perform the cleanup actions described in section 3.9.5.

A controller using Traffic Based Keep Alive may require up to 2 * KATT after the controller fetches the most recent command to detect a Keep Alive Timeout as shown in Figure TBDto.

Figure TBDto: Detecting Timeout Takes up to 2 * KATT



Periodic check 1
Most recent fetched command
Periodic check 2
Periodic check 3
KATT
Command fetched since last check, no timeout
KATT
No command fetched since last check, timeout detected

Figure TBDto shows that periodic check 3, not periodic check 2, detects the Keep Alive Timeout. Therefore, the time between fetching the most recent command and the check that detects the timeout (i.e., periodic check 3 in Figure TBDto) is up to 2 * KATT.

3.9.4.2 Traffic Based Keep Alive on the Host

The host ~~may~~ is able to use Traffic Based Keep Alive only if the controller is also using Traffic Based Keep Alive. The host should not use Traffic Based Keep Alive if the controller is not using Traffic Based Keep Alive because a controller that uses Command Based Keep Alive detects a Keep Alive Timeout based on the absence of Keep Alive commands, not the absence of all commands.

~~The host should check for a command completion queue entry for any Admin commands and I/O commands at half of the time equal to the Keep Alive Timeout value to account for delays (e.g., transport roundtrip times, transport delays, command processing times, and the Keep Alive Timer granularity).~~ To

~~prevent the controller from detecting a Keep Alive Timeout during the use of Traffic Based Keep Alive on the host, if no Admin command and no I/O command is sent to the controller during half of the Keep Alive Timeout Interval, the host should send a Keep Alive command.~~Traffic Based Keep Alive on the host is the same as Command Based Keep Alive on the host (refer to section 3.9.3.2), with two exceptions:

- The host is not required to submit a Keep Alive command if the host submitted an Admin command or I/O command and the host processed the completion for that command since the most recent time the host checked whether sending a Keep Alive command was necessary.
- To prevent the controller from detecting a Keep Alive Timeout during the use of Traffic Based Keep Alive on the host, the host should check for sending a Keep Alive command at a rate of KATT/4, instead of sending a Keep Alive command at a rate of KATT/2, while the Keep Alive Timer is active on the controller.

Like Command Based Keep Alive, if the host receives a successful completion to a Set Features command for the Keep Alive feature, then the host should adjust the time at which the host checks for sending the next Keep Alive command because the controller restarted the Keep Alive Timer.

If a host detects a Keep Alive Timeout and has outstanding commands for which that host has not received completions (refer to section 3.4.TBD), then it is strongly recommended that the host take the steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

For an example host implementation of Traffic Based Keep Alive, the host maintains a Keep Alive Send Timer for each controller to which that host is connected. The host uses the Keep Alive Send Timer to track when the host checks for sending a Keep Alive command to the corresponding controller. The host does not know when the controller fetches the Keep Alive command. Conservatively, the host assumes the controller fetches the Keep Alive command immediately upon the host sending the Keep Alive command. Whether or not the host sends a Keep Alive command after the Keep Alive Send Timer expires, the host tracks this time as the last expired timestamp of the Keep Alive Send Timer, for use when starting or restarting the Keep Alive Send Timer. While the host does restart the Keep Alive Send Timer after a successful Set Features command for the KATO feature, the host does not change the last expired timestamp when sending that Set Features command because at that point in time the host does not know the results of that command.

This example host implementation of Traffic Based Keep Alive behaves as follows:

- The host enables the Keep Alive Send Timer if the host requests activation of the Keep Alive Timer on the controller (e.g., the host enables the controller (refer to section 3.9.2)). The host records the current time as the last expired timestamp when the Keep Alive Send Timer becomes enabled.
- The host disables the Keep Alive Send Timer if the host requests deactivation of the Keep Alive Timer on the controller and the host receives a successful response from the controller (e.g., the host disables the controller (refer to section 3.9.2)).
- The host starts a Keep Alive Send Timer if:
  - the Keep Alive Send Timer becomes enabled; or
  - a successful Keep Alive command completion is processed by the host.
- The host restarts a Keep Alive Send Timer if:
  - the Keep Alive Send Timer is enabled;
  - a Keep Alive command is not outstanding; and
  - a successful Set Features command completion is processed by the host where the command specified the Keep Alive Timer feature (i.e., Feature Identifier 0Fh) and a non-zero KATO field.
- If a Keep Alive Send Timer starts or restarts, the host sets the Keep Alive Send Timer to:
  - KATT/4 minus the time elapsed since the last expired timestamp; or
  - zero, if the time elapsed since the last expired timestamp is greater than KATT/4.
- The host stops a Keep Alive Send Timer if the Keep Alive Send Timer becomes disabled.
- If a Keep Alive Send Timer expires, (i.e., the Keep Alive Send Timer is still enabled and time has elapsed equal to the value to which the Keep Alive Send Timer was set since the host most

recently started or restarted the Keep Alive Send Timer), then the host records the current time as the last expired timestamp and the host either:
  - starts a new Keep Alive Send Timer, if at least one Admin command or I/O command was submitted to the controller and the completion of that command was processed by the host since the last Keep Alive Send Timer started (not restarted); or
  - sends a Keep Alive command.
- The host detects a Keep Alive Timeout if the host sends a Keep Alive command and does not receive a completion for that Keep Alive command before KATT elapses from when the Keep Alive command was submitted.

~~For Traffic Based Keep Alive:~~

- ~~The Keep Alive Timer in the host starts or restarts if:~~
  - ~~a Set Features command with Feature Identifier 0Fh and a non-zero Keep Alive Timeout (KATO) field was submitted to the Admin submission queue;~~
  - ~~a Keep Alive command was submitted to the Admin submission queue.; or~~
  - ~~time equal to the Keep Alive Timeout value has elapsed since the Keep Alive Timer was most recently started or restarted, and a completion was received for an Admin command or an I/O command since the Keep Alive Timer was most recently started or restarted.~~
- ~~The Keep Alive Timer in the host expires if:~~
  - ~~time equal to the Keep Alive Timeout value has elapsed since the Keep Alive Timer was most recently started or restarted; and~~
  - ~~the host did not receive a completion of any Admin command or any I/O command since the Keep Alive Timer was most recently started or restarted.~~

~~If the Keep Alive Timer in the host expires, the host may consider a Keep Alive Timeout to have occurred. If a host detects a Keep Alive Timeout, the host should perform the actions described in section 3.9.5.~~

3.9.5 Keep Alive Timeout Cleanup

If a controller detects a Keep Alive Timeout ~~is detected:~~
~~a) by the controller~~, then the controller shall perform the following actions within the time specified by the CQT field (refer to Figure 276):

- record an Error Information Log Entry with the status code Keep Alive Timeout Expired;
- stop processing commands;
- set the Controller Fatal Status (CSTS.CFS) bit to ‘1’; and
- for message-based ~~NVMe T~~transports:
  - terminate the NVMe Transport connections for this association; and
  - break the host to controller association~~;~~.

~~and~~

~~b) by the host, the host assumes all outstanding commands are not completed and re-issues commands as appropriate.~~

For message-based ~~NVMe T~~transports, after completing these steps, a controller may accept a Connect command (refer to section 6.3) for the Admin Queue from the same or another host in order to form a new association.

If a host detects a Keep Alive Timeout and has outstanding commands for which that host has not received completions (refer to section 3.4.TBD), then it is strongly recommended that the host takes steps described in section 9.TBD to avoid possible data corruption caused by interaction between outstanding commands and subsequent commands submitted by that host to another controller.

5 Admin Command Set

5.17 Identify command

5.17.2 Identify Data Structures

5.17.2.1 Identify Controller Data Structure (CNS 01h)

…

Figure 276: Identify – Identify Controller Data Structure, I/O Command Set Independent

Bytes	I/O[1]	Admin[1]	Disc[1]	Description
…				
Admin Command Set Attributes & Optional Controller Capabilities				
…				
385				Reserved
387:386	M	M	M	**Command Quiesce Time (CQT):** This field indicates the expected worst-case time in 1 millisecond units for the controller to quiesce all outstanding commands (i.e., the controller shall satisfy all Immediate Abort requirements for those commands (refer to section 5.1.TBD)) <5.1.TBD referenced from TP4097a> after a Keep Alive Timeout (refer to section 3.9) or other communication loss (refer to section 9.TBD). If this field is cleared to 0h, then a command quiesce time is not reported. If the controller does not require any time to quiesce, the controller should set this field to 1h (i.e., 1 millisecond).
511:388 ~~385~~				Reserved

6 Fabric Command Set

6.4 Disconnect Command and Response

…

The completion queue entry for the Disconnect command shall be the last entry submitted to the I/O Queue Completion queue by the controller (i.e., no completion queue entries shall be submitted to the I/O Queue Completion Queue after the completion queue entry for the Disconnect command). The controller shall ~~not perform~~ ensure that no further command processing is performed for any command on an I/O queue after sending the completion queue entry for the Disconnect command.

…

9 Error Reporting and Recovery

9.TBD Communication Loss Handling

If the host loses communication with a controller, then the host is unable to receive a completion (CQE) for any outstanding command that has been submitted to that controller (refer to section 3.4.TBD). If the host is able to use another controller to access the same NVM subsystem or re-establish communication with the original controller, then it is strongly recommended that any host use of that controller to recover from communication loss follow the procedures and requirements in this section in order to avoid possible corruption of user data and unintended changes to NVM subsystem state.

Host recovery from communication loss with a controller consists of three functional components:

- Host determination that communication with a controller has been lost is described in section 9.TBD.1.
- Host determination that no further processing of outstanding commands is possible on that controller is described in section 9.TBD.2.
- Host retry, if any, of outstanding commands after communication loss is described in section 9.TBD.3.

These functional components interact with each other. Host detection of communication loss is necessary before the host is able to determine when no further controller processing of outstanding commands is possible. Host retries of outstanding commands that modify user data or NVM subsystem state are able to corrupt user data or make unintended changes to NVM subsystem state unless the host determines that no further controller processing of the original commands is possible as described in section 9.TBD.2.

9.TBD.1 Host Communication Loss with a Controller

A host determines that communication has been lost with a controller if:

- the host detects a Keep Alive Timeout (refer to section 3.9);
- for message-based transports, the host or the controller terminates the NVMe Transport connection on which the command was sent (refer to section 3.3.2.4); or
- the host detects a transport connection loss using methods outside the scope of this specification (e.g., the transport notifies the host of a loss of communication either with the controller to which the command was submitted or with the queue on which the command was sent).

A controller may detect a loss of communication at a different time (e.g., later) than the host detects that loss of communication. As explained in section 9.TBD.2, additional time may be required for the controller to stop processing commands after the controller detects a loss of communication.

9.TBD.2 End of Controller Processing of Outstanding Commands

This section describes how a host determines that no further controller processing of an outstanding command is possible after a loss of communication happens. At the time when a host detects a communication loss with a controller, the outstanding commands, if any, are commands for which the host is unable to receive a CQE as a result of the communication loss (refer to section 3.4.TBD).

Some commands (e.g., Sanitize) initiate background operations. These background operations are able to continue after a host loss of communication with the controller that started the background operation. After such a loss of communication, additional measures (e.g., commands submitted to a different controller) are necessary for the host to track progress and completion of such a background operation.

A host that is unable to communicate with a controller should perform the following steps in order to determine that no further controller processing of outstanding commands is able to occur:

1. For message-based transports, terminate the association and the associated transport connections. This step is skipped for memory-based transports.
2. Wait for sufficient time to ensure that the controller has detected a loss of communication using at least one of the following:
    a. If the controller uses Command Based Keep Alive (refer to section 3.9.3.1), wait at least until 2 * KATT (refer to section 3.9) from the time the host submitted the most recent Keep Alive Command to the controller;
    b. If the controller uses Traffic Based Keep Alive (refer to section 3.9.4.1), wait at least until 3 * KATT from the time the host submitted the most recent command to the controller; or
    c. Receive a transport-specific notification for determining that the controller has terminated an NVMe Transport connection or detected a loss of communication (e.g., a fabric notification or a PCIe surprise link down error notification for a PCIe link that directly connects a host to an NVM subsystem (e.g., an SSD)).

3. Wait for additional sufficient time to ensure that the controller has stopped processing commands using one of the following:
   a. If the CQT field (refer to Figure 276) is non-zero, wait for the amount of time indicated in the CQT field to elapse; or
   b. If the CQT field is cleared to 0h, wait for an implementation specific amount of time (e.g., 10 seconds). The host should allow this value to be administratively configured.

The specification of the times to wait to ensure that the controller has detected a Keep Alive Timeout described in this section (i.e., 2 * KATT and 3 * KATT) assumes that the transport delays any command by at most one KATT. Once the last command is fetched by the controller, the controller is required to detect a Keep Alive Timeout after at most a further 1 * KATT for Command Based Keep Alive and at most 2 * KATT for Traffic Based Keep Alive (refer to Figure TBDto). The sum of the two delays (i.e., the transport delay and the delay to detect the Keep Alive timeout) is 2 * KATT for Command Based Keep Alive and 3 * KATT for Traffic Based Keep Alive.

9.TBD.3 Command Retry After Communication Loss

If the host loses communication with a controller, then the host is unable to receive a completion (CQE) for any outstanding command (refer to section 3.4.TBD) that has been submitted to that controller. If the host is able to use another controller to access the same NVM subsystem or re-establish communication with the original controller, the host may be able to use that controller to recover from the communication loss by retrying outstanding commands. It is strongly recommended that any host retry of any outstanding commands after communication loss follow the procedures and requirements in this section in order to avoid possible corruption of user data and unintended changes to NVM subsystem state (e.g., Reservation state (refer to section 8.19)).

For command retry purposes, every outstanding command falls into one of three command retry categories, Unrestricted Retry, Delayed Retry, or State-Dependent Retry, based on whether the command is idempotent (refer to section 9.TBD.3.1), and whether the command modifies user data or NVM subsystem state. Section 9.TBD.3.2 defines these categories and describes requirements and restrictions on retrying outstanding commands in each category.

9.TBD.3.1 Idempotent Commands

Controller processing of an idempotent command produces the same end state on the NVM subsystem and returns the same results if that command is resubmitted one or more times with no intervening commands. All commands tend to modify some ancillary state on the controller (e.g., tracking statistics); these ancillary changes to state do not prevent a command from being considered idempotent. The results of the command include the status code (excluding transient status codes or error conditions, e.g., due to a loss of communication), any data returned to the host and any NVM subsystem changes to user data or state (e.g., reservation state, feature contents).

For example, a read command addressed to a specific location (e.g., LBA) in a namespace is an idempotent command. The read command addressed to a valid location in a namespace returns the same data with a successful completion status code if that command is submitted repeatedly. Similarly, a write command addressed to a valid location in a namespace writes the same data to that location if submitted repeatedly. This command is also an idempotent command.

On the other hand, a Namespace Management command (refer to section 5.23) that creates a namespace is not idempotent (i.e., is a non-idempotent command), as repeating the Namespace Management command creates additional namespaces with different namespace identifiers. Similarly, a Reservation Register command that unregisters a host (refer to section 7.4) is also not idempotent because repeating the command attempts to unregister a host that is no longer registered and returns an error status code.

9.TBD.3.2 Command Retry Categories and Requirements

For command retry purposes, an outstanding command is in one of three categories:

- **Unrestricted Retry**: The outstanding command is an idempotent command (refer to section 9.TBD.3.1) that is able to be retried without restrictions because that outstanding command has no effect on user data or NVM subsystem state (e.g., an NVM Command Set Read command). Refer to section 9.TBD.3.2.1.
- **Delayed Retry**: The outstanding command is an idempotent command for which any retry and/or reporting of the result of that retry is required to be delayed until no further controller processing is possible of that outstanding command because the outstanding command modifies user data or NVM subsystem state (e.g., an NVM Command Set Write command, except as described in section 9.TBD.3.2.2). Refer to section 9.TBD.3.2.2.
- **State-Dependent Retry**: The outstanding command is not an idempotent command (e.g., a Namespace Management command that creates a namespace) or is an idempotent command that is able to affect behavior of other hosts. The procedures for recovery from such an outstanding command depend on the extent, if any, to which that outstanding command has been processed by the controller. Refer to section 9.TBD.3.2.3.

Sections 9.TBD.3.2.1, 9.TBD.3.2.2, and 9.TBD.3.2.3 define each command retry category and describe host requirements and restrictions that prevent retry of outstanding commands from corrupting user data or making unintended changes to NVM subsystem state.

9.TBD.3.2.1 Unrestricted Retry Commands

An outstanding command is an Unrestricted Retry command if that command:
a) is an idempotent command (refer to section 9.TBD.3.1); and
b) does not change more than ancillary state in the NVM subsystem (e.g., statistics such as the value of the Data Units Read field in the SMART / Health Information log page (refer to Figure 208)).

For an Unrestricted Retry command:
- the Controller Capability Change (CCC) bit;
- the Namespace Inventory Change (NIC) bit;
- the Namespace Capability Change (NCC) bit; and
- the Logical Block Content Change (LBCC) bit

are all cleared to ‘0’ in the Commands Supported and Effects data structure (refer to Figure 212) in the Commands Supported and Effects log page (refer to section 5.16.1.6). If any of these four bits is set to ‘1’, then that command is not an Unrestricted Retry command.

A host may retry any outstanding command that is an Unrestricted Retry command immediately after communication loss without determining whether further controller processing of that outstanding command is possible.

For recovery purposes, a host may treat any outstanding command that is an Unrestricted Retry command as if that command were a Delayed Retry command or a State-Dependent Retry command.

9.TBD.3.2.2 Delayed Retry Commands

An outstanding command is a Delayed Retry command if that command:
a) is an idempotent command (refer to section 9.TBD.3.1); and
b) changes user data or NVM subsystem state (e.g., Read Recovery Level (refer to section 8.17) or Reservation state (refer to section 8.19)),

unless the changes to user data or NVM subsystem state are able to affect the behavior of any other host (e.g., refer to the example in Annex B.TBD.4). As explained further in Annex B.TBD.4, use of individual Delayed Retry commands (e.g., an NVM Command Set Write command) that are not part of a fused operation to affect the behavior of other hosts is strongly discouraged.

A host should treat an outstanding command that is a Delayed Retry command as having command ordering requirements with respect to other commands where those command ordering requirements are enforced by higher-level software (refer to section 3.4.1). Hence, for any such command, a host should not report completion or error status, including errors caused by communication loss, to higher-level software (e.g., an associated application, filesystem or database) until the host has determined that no further controller processing of that command and retries, if any, of that command is possible. If a host violates this recommendation, corruption of user data and unintended changes to NVM subsystem state are possible; refer to Annex B.TBD.1 for an example where user data is corrupted.

A host is able to comply with this “should not report” recommendation by delaying submission of a retry of any outstanding command that is a Delayed Retry command until no further controller processing is possible of the original outstanding command and any previously submitted retries of that command. This avoids host delays in reporting completion of any command upon receiving the CQE for that command.

A host may treat any outstanding command that is a Delayed Retry command as if that command were a State-Dependent Retry command.

A host that does not adhere to the recommendations in this section for handling outstanding commands that are Delayed Retry commands risks causing corruption of user data. It is strongly recommended that host NVMe implementations adhere to these recommendations to avoid data corruption.

9.TBD.3.2.3 State-Dependent Retry Commands

An outstanding command is a State-Dependent Retry command if the command changes user data or NVM subsystem state, and the command:

a) is not an idempotent command (refer to section 9.TBD.3.1); or
b) changes user data or NVM subsystem state in a way that is able to affect the behavior of other hosts.

A host should not retry an outstanding command that is a State-Dependent Retry command without first determining that command retry is the appropriate recovery action. This is because retrying such a command may have different results than the original command, duplicate the results of the original command, or affect the behavior of other hosts in a different manner than the original command. In general, determination of the appropriate recovery action is only able to be performed by higher-level software (e.g., an associated application, filesystem or database) that is able to determine the extent, if any, to which the outstanding command has been processed and enforce ordering requirements among commands (refer to section 3.4.1).

A host should treat an outstanding command that is a State-Dependent Retry command as having command ordering requirements enforced by higher-level software with respect to other commands (refer to section 3.4.1). Hence, for any such command, a host should not report completion or error status, including errors caused by communication loss, to higher-level software (e.g., an associated application, a filesystem or database), until the host has determined that no further controller processing of the outstanding command and retries, if any, of that command is possible. If a host violates this recommendation, corruption of user data or unintended changes to NVM subsystem state are possible; refer to Annex B.TBD.2 for an example where unintended changes occur to NVM subsystem state.

A host that does not adhere to the recommendations in this section for handling outstanding commands that are State-Dependent Retry commands risks causing corruption of user data and unintended changes to NVM subsystem state. It is strongly recommended that host NVMe implementations adhere to these recommendations to avoid these outcomes.

Annex B. Host Considerations (Informative)

B.TBD Examples of Incorrect Command Retry Handling After Communication Loss

Section 9.TBD.3 describes requirements for host retry of outstanding commands after communication loss. In this situation, the response of a command is unknown and hence the host has no information about the extent, if any, to which the controller has processed that command. Many commands are not safe to unconditionally retry if they have been processed in part or completely. This annex describes examples of problematic situations caused by retrying an outstanding command without regard to the consequences of that retry.

B.TBD.1 Write after Write

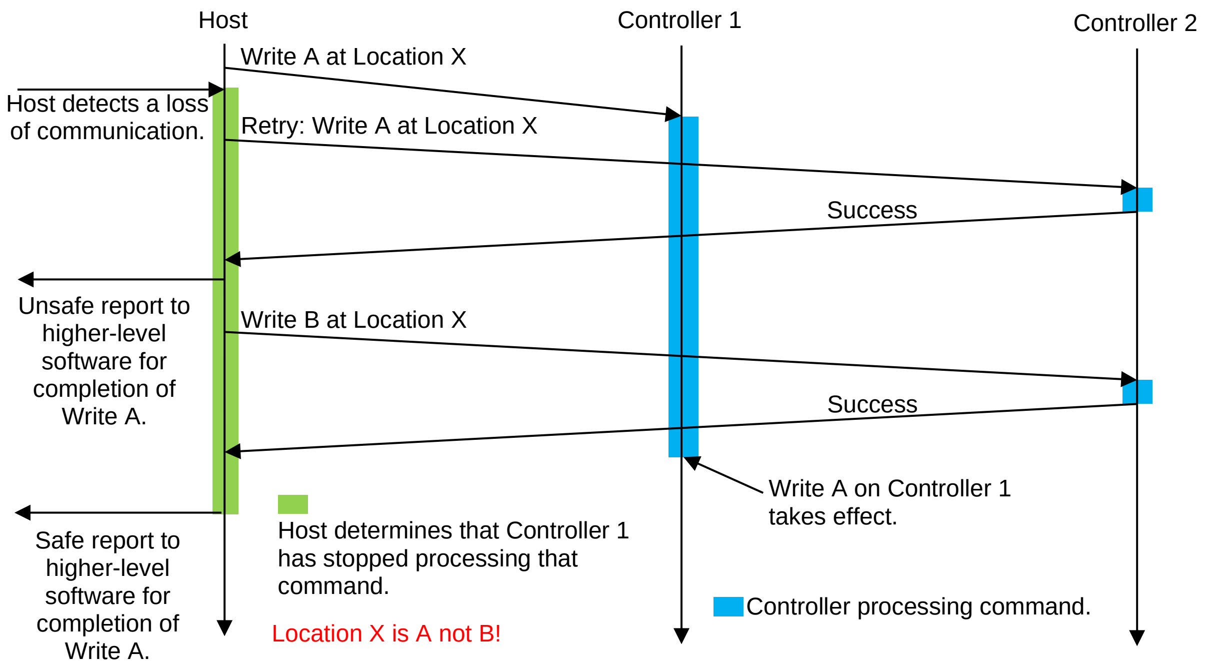
In the example shown in Figure TBDx1, the host loses communication with Controller 1 and does not receive a response from Controller 1 for an idempotent command that changes user data at location X to A (e.g., an NVM Command Set Write command). The following events occur:

- The host retries that command on Controller 2 (Retry: Write A at Location X), and the retry succeeds quickly.
- The completion of that retry leads to the host subsequently submitting a command that changes the user data at the same location to B (Write B at Location X).
- During this time, Controller 1 has been processing the original outstanding command (Write A at Location X), and that command's change of user data at location X to A finally takes effect after the user data at location X has already been changed to B.

The final outcome is that the user data at location X is A, which is incorrect and an example of data corruption.

For an idempotent command that changes user data or NVM subsystem state, this example shows why the host should not report the results of that command, including any retry of that command, to higher-level software until the host is able to determine that no further controller processing of that command and any retry of that command is possible (refer to section 9.TBD.2).

Figure TBDx1: Write after Write

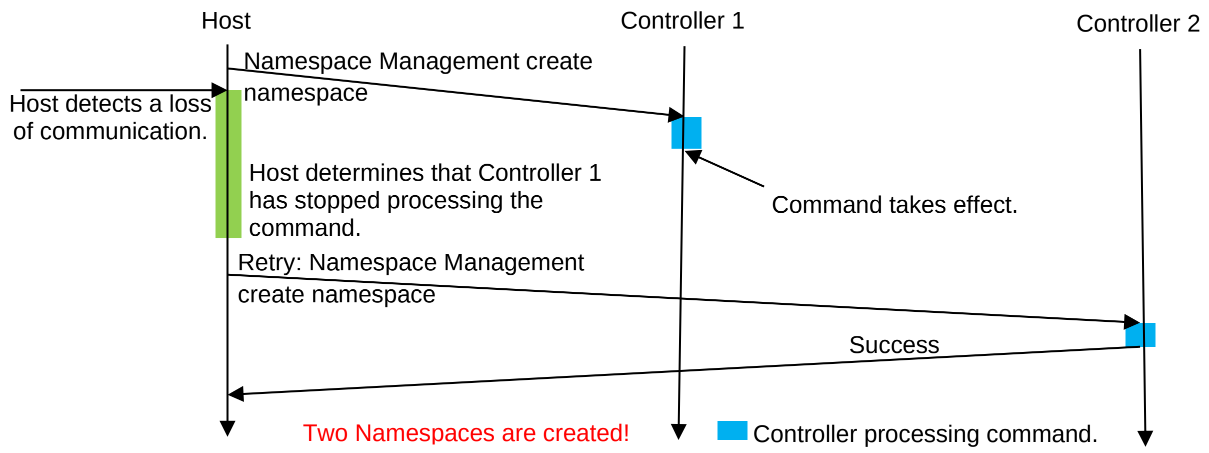

Host
Controller 1
Controller 2
Write A at Location X
Host detects a loss of communication.
Retry: Write A at Location X
Success
Unsafe report to higher-level software for completion of Write A.
Write B at Location X
Success
Write A on Controller 1 takes effect.
Host determines that Controller 1 has stopped processing that command.
Safe report to higher-level software for completion of Write A.
Location X is A not B!
Controller processing command.

B.TBD.2 Non-Idempotent Command

In the example shown in Figure TBDx2, the host loses communication with Controller 1 and does not receive a response from Controller 1 for a Namespace Management command that creates a namespace (refer to section 5.23). The host ensures that no further controller processing of that command is possible (refer to section 9.TBD.2), and then retries that command on Controller 2, which creates a second namespace.

This example shows why higher-level software (e.g., an associated application, filesystem or database) should take steps to determine that a retry of a non-idempotent command does not cause unintended changes to NVM subsystem state (e.g., number of namespaces).

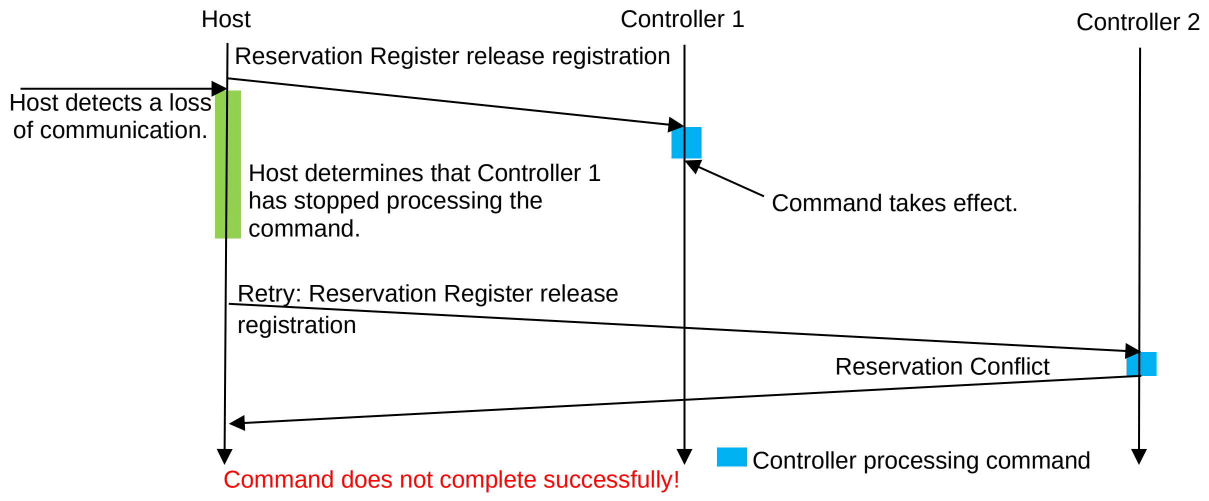
Figure TBDx2: Non-Idempotent Command


Host
Controller 1
Controller 2
Namespace Management create namespace
Host detects a loss of communication.
Host determines that Controller 1 has stopped processing the command.
Command takes effect.
Retry: Namespace Management create namespace
Success
Two Namespaces are created!
Controller processing command.

B.TBD.3 Retried Command Does Not Succeed

In the example shown in Figure TBDx3, the host loses communication with Controller 1 and does not receive a response to a Reservation Register command that unregisters the host (refer to section 7.4). The host ensures that no further controller processing of that command is possible (refer to section 9.TBD.2), and then retries that command on Controller 2. As a result of the original command unregistering the host, the host is no longer a registrant, and for that reason, the controller returns a status code of Reservation Conflict (refer to section 8.19.4).

Figure TBDx3: Retried Command Does Not Succeed


Host
Controller 1
Controller 2
Reservation Register release registration
Host detects a loss of communication.
Host determines that Controller 1 has stopped processing the command.
Command takes effect.
Retry: Reservation Register release registration
Reservation Conflict
Controller processing command
Command does not complete successfully!

This example shows why an error status code is able to be returned if a non-idempotent command is retried after the original command has been processed. An analogous example is possible for the Compare and Write fused operation (refer to the Fused Operation section of the NVM Command Set Specification) because that fused operation is not idempotent.

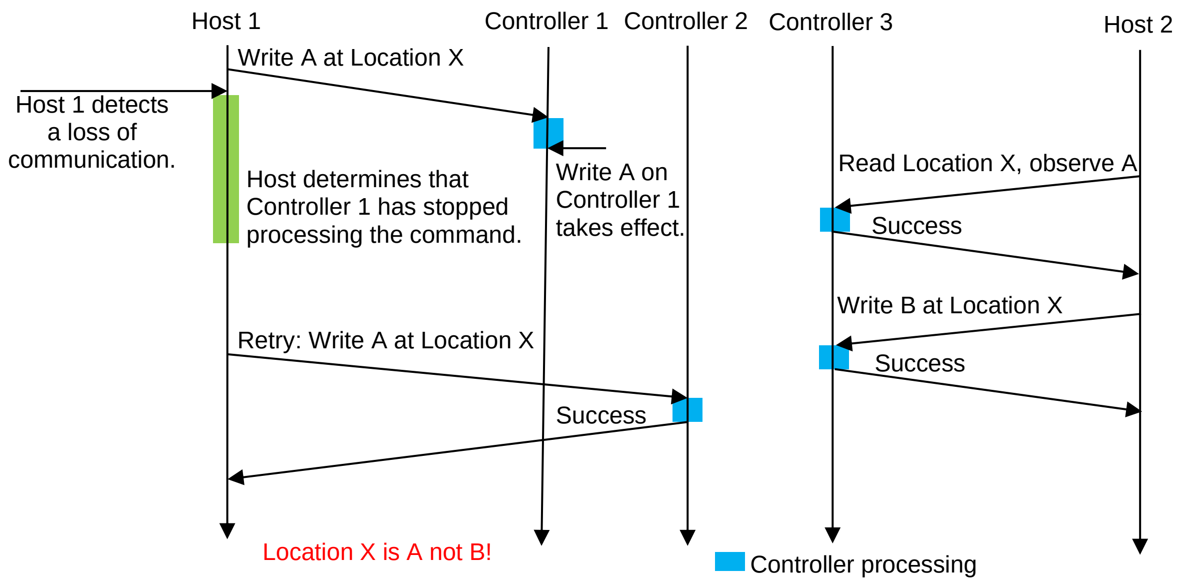
B.TBD.4 Retried Command Affects Another Host

In the example shown in Figure TBDx4, two hosts use Location X in a namespace for coordination. Writing the value A to Location X indicates that step A in a processing sequence has been completed, and writing the value B indicates that step B in the processing sequence has been completed, where higher-level software requires that step B follow step A.

Host 1 indicates completion of step A by writing the value A to Location X, but loss of communication prevents Host 1 from receiving the completion of that command. Host 2 observes that step A is complete, quickly performs step B, and indicates completion of step B by writing the value B to Location X.

In the absence of receiving a completion for the original command, Host 1 retries writing the value A to Location X, overwriting the completion of step B reported by Host 2. This example shows that retry of commands that are able to affect the behavior of other hosts is problematic. In this example, higher-level software needs a mechanism to indicate that the writes to Location X are not safe to retry after a delay.

Figure TBDx4: Retried Command Affects Another Host


Host 1
Controller 1
Controller 2
Controller 3
Host 2
Write A at Location X
Host 1 detects a loss of communication.
Host determines that Controller 1 has stopped processing the command.
Write A on Controller 1 takes effect.
Read Location X, observe A
Success
Write B at Location X
Success
Retry: Write A at Location X
Success
Location X is A not B!
Controller processing

This sort of higher-level software usage of ordinary NVMe commands (e.g., NVM Command Set Write commands) for coordination and synchronization among multiple hosts is strongly discouraged because retry of these commands after communication loss is problematic. Higher-level software should instead use mechanisms intended for coordination among multiple hosts. Two examples of such mechanisms are:

- Reservations (refer to section 8.19); and
- Compare and Write fused operations (refer to the Fused Operation section of the NVM Command Set Specification).

In addition, command retries that modify NVM subsystem state (e.g., a Set Features command that modifies a feature that has any scope that is visible to other hosts as described in Figure 317 <NOTE: NVM Express Base Specification NEXT_2023_11_08 has a scope column (Figure 409) not present in 2.0d>) is able to affect the behavior of other hosts. Use of commands that modify NVM subsystem state for coordination and synchronization among multiple hosts is likewise strongly discouraged.

Description for Changes Document for NVM Express TCP Transport Specification 1.0c

3 Transport Binding

3.5 Error Handling Model

3.5.1 Transport Error Handling

…

In response to a H2CTermReq PDU, the controller shall terminate the connection. If the controller does not terminate the connection within 30 seconds, the host may terminate the connection. The maximum H2CTermReq PDU data size shall not exceed 128 bytes. The controller shall ignore a H2CTermReq PDU with a PDU length (PLEN) that exceeds 152 bytes (24-byte PDU Header plus 128-byte PDU data) and shall terminate the connection immediately. Additionally, the controller shall ignore a H2CTermReq PDU with PLEN less than 24 bytes and shall terminate the connection immediately. If the host is unable to send a H2CTermReq PDU, then the host shall reset the TCP connection.

The host and controller may detect connection loss at different times (refer to the Communication Loss Handling section in the NVM Express Base Specification). The host should not assume the controller has detected connection loss if there is a loss of a TCP connection detected by the host (e.g., due to host-side failures or an intermediary such as a firewall sending a TCP RST). The host should treat the reception of a C2HTermReq PDU as an indication that the controller has initiated clean-up of the NVMe Transport Connection.

…

Description for Changes Document for NVM Express RDMA Transport Specification 1.0b

3 Transport Binding

3.5 Error Handling Model

3.5.1 RDMA Transport Errors

Errors detected by the RDMA Transport may result in the termination of any command capsule, response capsule, or data transfer operations and may result in the tear down of the RDMA QP(s). The RDMA Transport may detect errors that are not directly associated with a capsule or data transfer operation (e.g., tear down of the RDMA QP due to connection loss, data corruption, or protection error). In the case of a RDMA QP tear down, the RDMA Transport is responsible for terminating the RDMA QP, freeing up any NVMe Transport resources, and then informing the NVMe layer about the termination and the associated cause.

The host and controller may detect connection loss at different times (refer to the Communication Loss Handling section in the NVM Express Base Specification). The host should not assume the controller has detected connection loss if the RDMA Transport has terminated the RDMA QP (e.g., due to host-side failures).