Tape Error Recovery in Queued Environments

Objectives

- Detection of and recovery from errors without ULP timeout
- Don’t add protocol overhead for normal case
    - Limit delays on sending next queued command
    - OK to add overhead when things go wrong
- Implementable with existing protocol chips
- Compatible with existing devices
- Maintain command ordering
- Minimize data retention requirements in target
    - Timely confirmation that exchange is complete
- Desirable to have Host-initiated rather than target-initiated recovery

Alternatives Evaluated - Command Ordering

- Recommend Dropped/Deferred:
    - FCP_CONFIRM on FCP_CMD - before next command can be sent
    - Keep info until X_ID reused, poll to confirm delivery of CMD
    - Continuously increasing sequence count + flush
    - Utilize parameter field of FC header
    - Continuously increasing X_ID
    - Continuously increasing SEQ_ID
- Recommended:
    - "I/O identifier"/Command sequence number in FCP_CMND resvd field.

CROSSROADS

FCP_CONFIRM on FCP_CMD - before next command can be sent

- Pros: Works, Synchronous interlock, simple to understand, implement
    - Maintains command ordering
    - Compatible with existing devices
- Cons: Potentially breaks HW, breaks SW, Sync mode lowers performance
    - New IU at this point in command may break hardware assists
    - Adds protocol overhead in normal case
    - Interlock restricts rate of sending commands, especially through large fabrics
        - precisely the cases we want more commands in queue for performance
- Recommendation: Defer

Keep info until X_ID reused, poll to confirm delivery of CMD

- Pros: simple, works
  - Can be used to maintain command ordering
  - Compatible with existing devices
  - Implementable with existing protocol chips
- Cons: Memory/resource hog, inhibits performance, impairs the normal flow
  - Adds protocol overhead in normal case
  - Interlock restricts rate of sending commands, especially through large fabrics
    - precisely the cases we want more commands in queue for performance
  - Does not allow release of resources in a timely manner after completion of exchange
- Recommendation: Defer

Continuously Increasing Sequence Count + no_traffic Flush

- Pros: Detects lost frames immediately
  - Maintains command ordering
  - Immediate error detection in in-order fabric
    - Nightmarish in out-of-order
  - No protocol overhead except in no-traffic case
  - No interlocks to delay next command
  - Minimizes data retention requirements in target, without requiring FCP_CONF
- Cons:
  - Will not work with existing protocol chips
  - Need mechanism (ELS?) to identify missing frame to sequence initiator - IDs not known.
  - Target equal participant in recovery
- Recommendation: Defer

Utilize parameter field of FC header

- Pros: Simple, effective
  - Maintains command ordering
  - Does not add overhead for normal cases
  - Non-interlocked
  - Could be made compatible with existing targets
  - May work with existing protocol chips
  - In conjunction with FCP_CONF, could minimize target data retention requirements
- Cons: Could break HW, repugnant
  - Wrong layer - embedding FCP behavior in FC-2
  - An FC-2 fix for an FC-4 problem
- Recommendation: Drop

CROSSROADS

Continuously Increasing X_ID

- Pros: No change to standard
  - Choice of OX_IDs is outside the standard
  - Doesn’t add protocol overhead
  - Non-interlocked
  - Compatible with existing devices
  - In conjunction with FCP_CONFIRM, minimizes target data retention
- Cons:
  - Can’t use with existing protocol chips
  - Difficult to detect lost commands
  - Difficult to maintain which OX_ID should be next, particularly at level that would need to generate them
  - FC-2 fix for FC-4 problem
- Recommendation: Drop

Continuously increasing SEQ_ID

- Pros: No change to standard
  - Choice of SEQ_IDs is outside the standard
  - Doesn’t add protocol overhead
  - Non-interlocked
  - Compatible with existing devices
  - In conjunction with FCP_CONFIRM, minimizes target data retention
- Cons:
  - Can’t use with existing protocol chips
  - Difficult to detect lost commands
  - Difficult to maintain which SEQ_ID should be next, particularly at level that would have to generate them
  - FC-2 fix for FC-4 problem
- Recommendation: Drop

"I/O identifier"/Command sequence number in FCP_CMND resvd field.

- Pros: Meets Objectives
  - Detects and facilitates recovery from lost/misordered commands
  - Implementable with existing chips
    - Implemented in FCP driver
  - Compatible with existing targets
  - Allows sending multiple commands without interlock
  - No additional frames unless an error condition or no traffic
  - Use with FCP_CONF to allow timely release of resources by target
  - Initiator-driven recovery
- Cons:
  - Need to ensure existing chips don’t check for reserved fields = 0.
- Recommendation: This should be adopted

Implementation Details

- Enabled by PRLI
  - Byte 3 bit 8 to employ Command Sequence Numbering
- Embed Command Sequence Number in Byte 0 of FCP_CNTL field of FCP_CMND
  - Continuously increasing on an I-T-L nexus
  - Target can detect out-of-order command & respond to it
  - Initiator not to reuse sequence number until delivery confirmed
- Target presents Response Code for lost commands
  - May elect to wait R_A_TOV for out-of-order command if FLOGI allowed OOO delivery
  - RSP_CODE 06 to mean Command Received Out Of Order; all OOO commands aborted
  - Initiator to abort lost command(s) with ABTS and reissue with correct sequence number, new OX_ID; Clear Queue resets sequence number to 0.
- Delivery confirmation via
  - FCP_XFER_RDY, FCP_DATA, or FCP_RSP, OR Acceptance of next command OR REC.