iSCSI implementer's Guide Update

Mallikarjun Chadalapaka

Hewlett-Packard Company

Agenda

- Goals
- Response Fence
- FastMultiTaskAbort
- Discussion/Questions

Goals

- Clarify the iSCSI specification (RFC 3720) where ambiguous
- Offer required implementation guidance where RFC 3720 is silent
- Fix defects where necessary
- Enhance RFC 3720 where there is a compelling case
- Complement RFC 3720, not replace it
- Update RFCs 3720, 3722, 3723, and 4173

Response Ordering

- Non-issue for single-connection iSCSI sessions
- When SCSI responses travel on multiple iSCSI connections back to a SCSI initiator, SCSI→SCSI response ordering is not guaranteed.
- Works fine for 99% of SCSI responses where SCSI doesn't care about response ordering
- Is an issue in the following cases identified so far:
  - Multi-task abort scenarios (Abort Task Set, Clear Task Set etc.)
  - Auto Contingent Allegiance (ACA) scenarios
- In short: whenever one SCSI response signals action taken on multiple SCSI tasks **and** those tasks have differing connection allegiances..

But….

- Response ordering semantics are very specific to SCSI.
  - Would like to avoid (limit, to begin with) SCSI-specific code in iSCSI layer
  - Don't want to update iSCSI RFCs every time SCSI implies a new implied response ordering need on a new opcode.

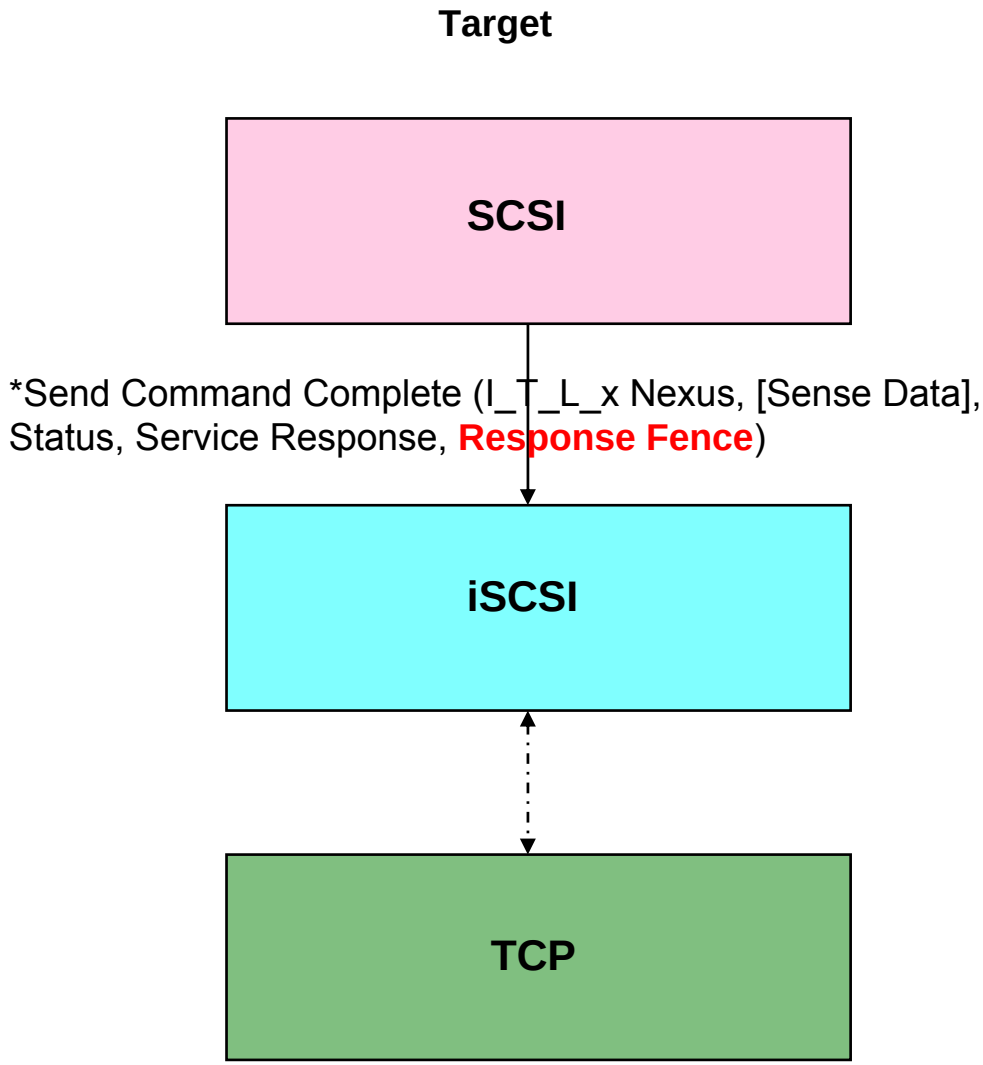
- Proposal:
  1. Abstract away the response ordering need into a new argument, Response Fence, from SCSI→iSCSI transport service call.
  2. Guarantee response ordering at iSCSI level whenever Response Fence is set.
  3. Grandfather the two known SCSI ordering dependencies (Task management & ACA) – i.e. always assume Response Fence for the two.

Response Fence Model

If Response Fence is set:

• Response with Response Fence MUST chronologically be delivered after all the "preceding" responses on the I_T_L nexus to the initiator.

• Response with Response Fence MUST chronologically be delivered prior to all the "following" responses on the I_T_L nexus.

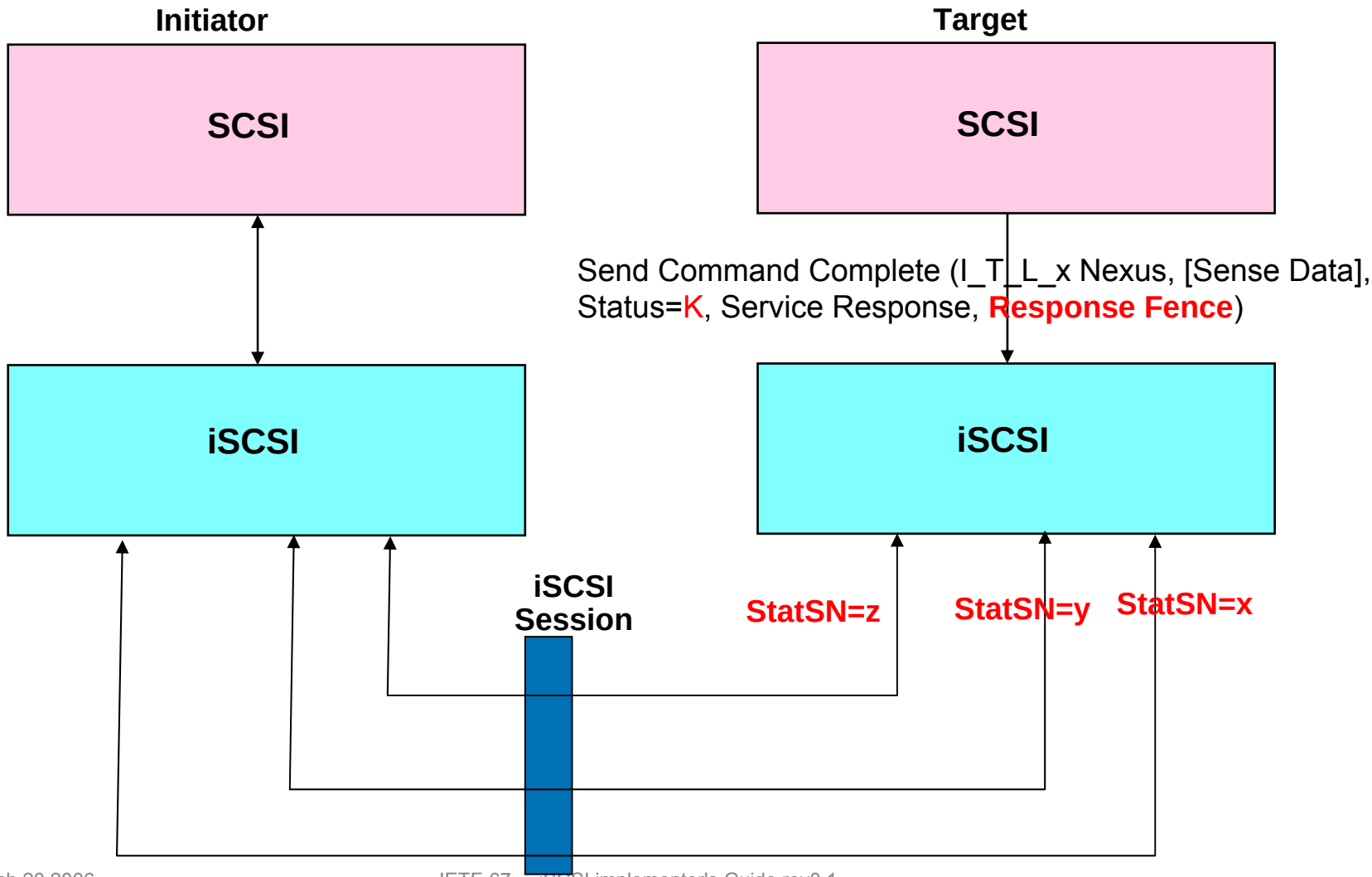


*Alternatively, this could be:

Task Management Function Executed

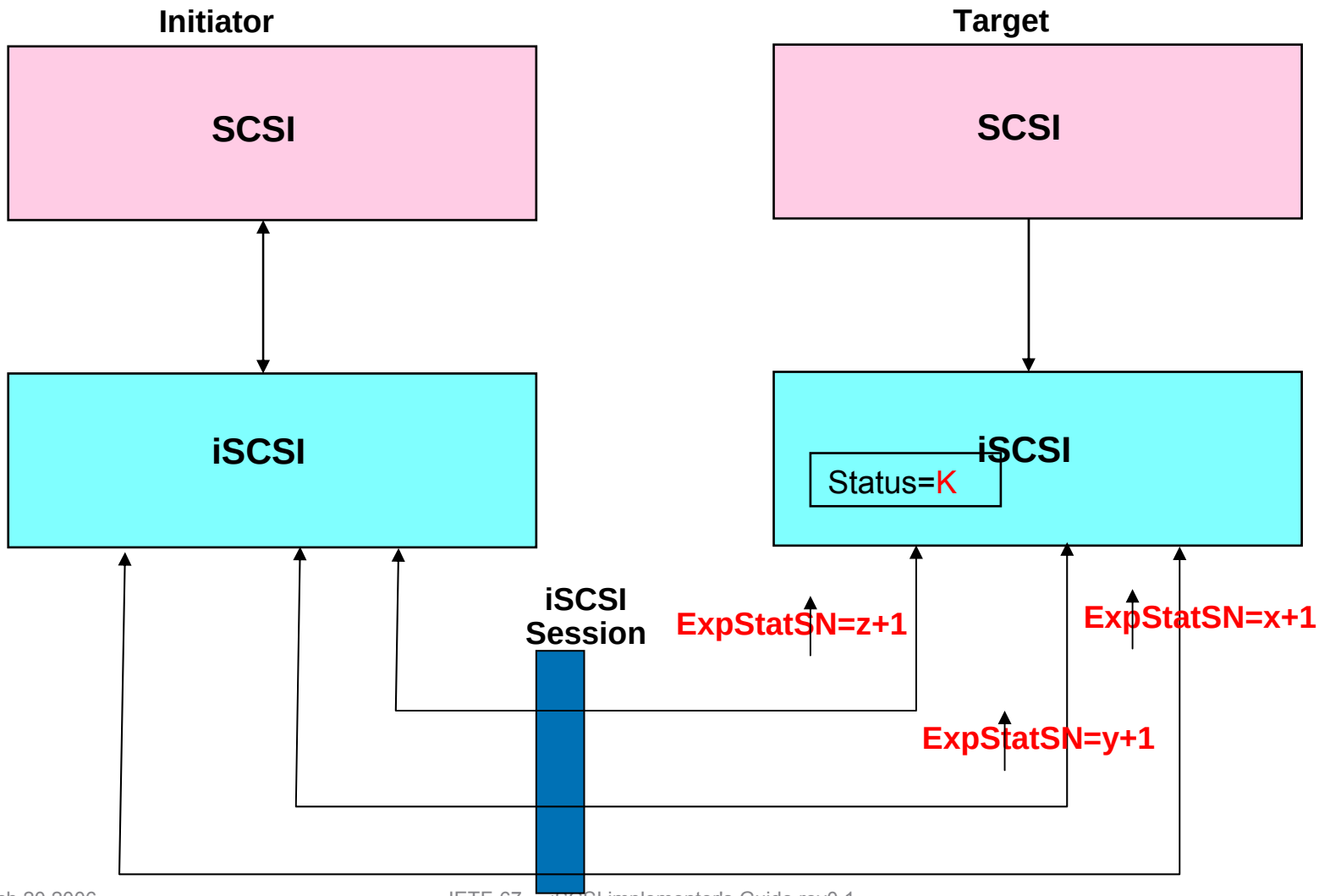
Target iSCSI action: Step.1

On Seeing Response Fence, note the last sent StatSN on each connection in the session. Fence is established.



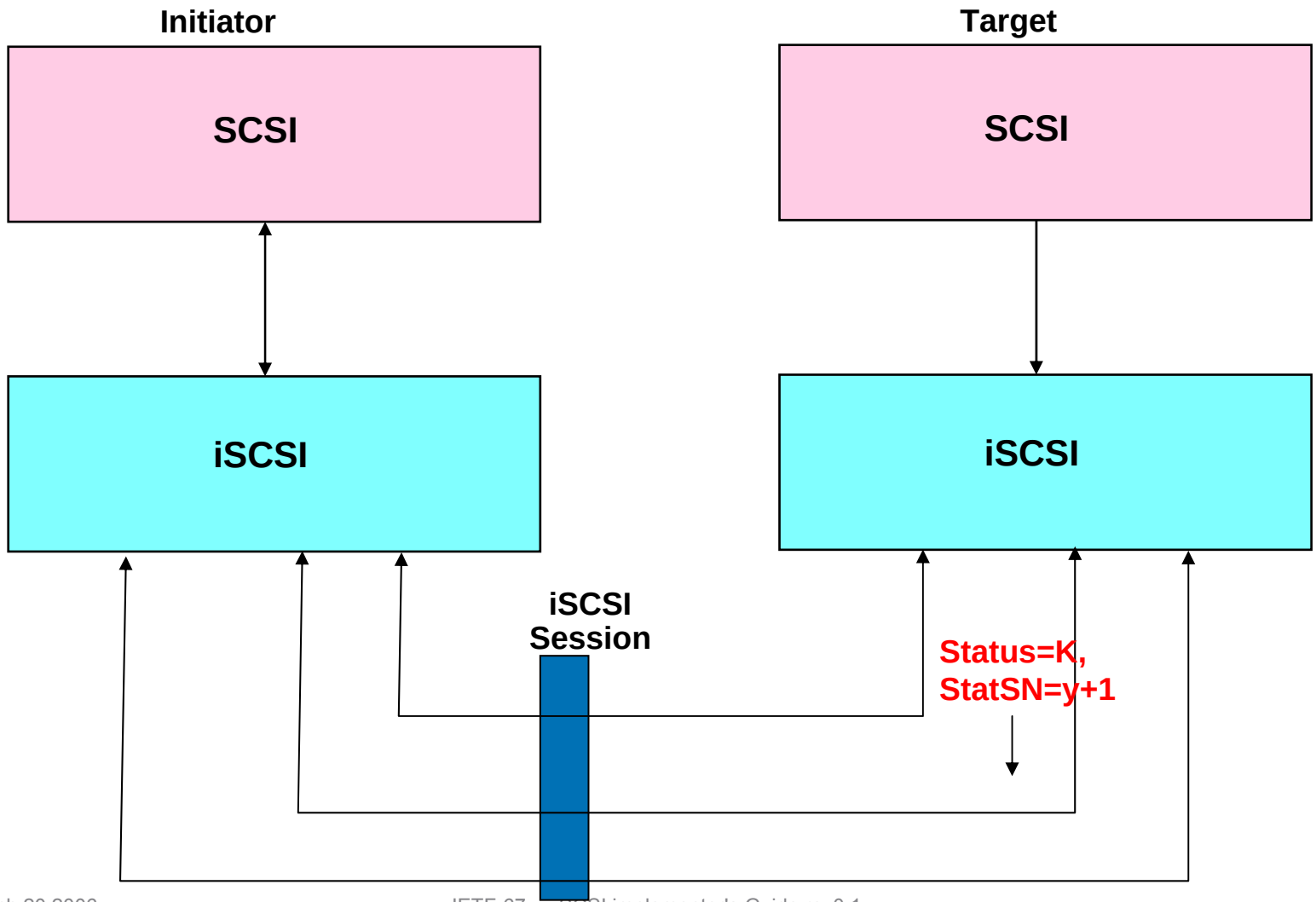
Target iSCSI action: Step.2

Wait for the StatSN ack on each connection.



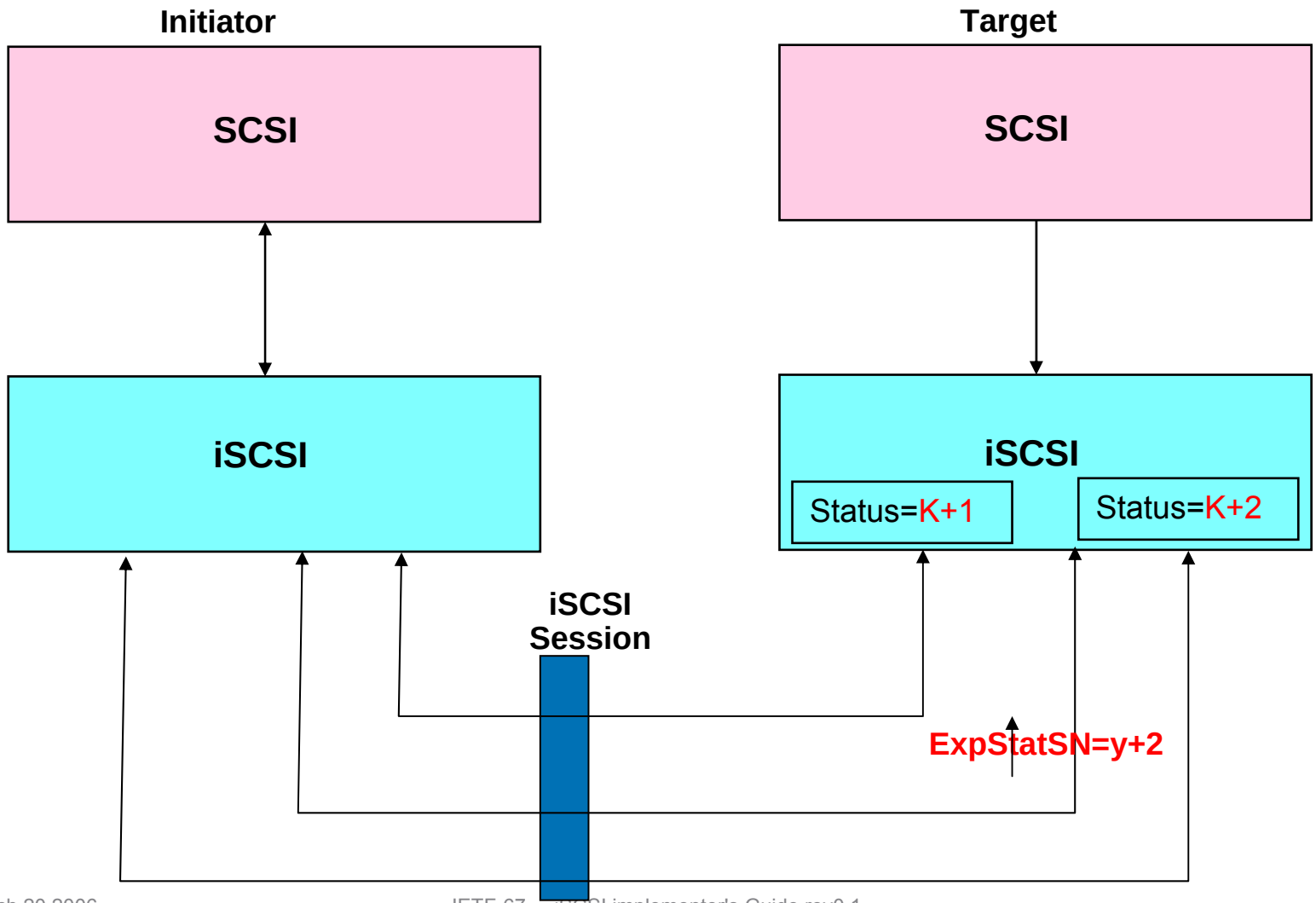
Target iSCSI action: Step.3

Send Status=K on the appropriate connection.



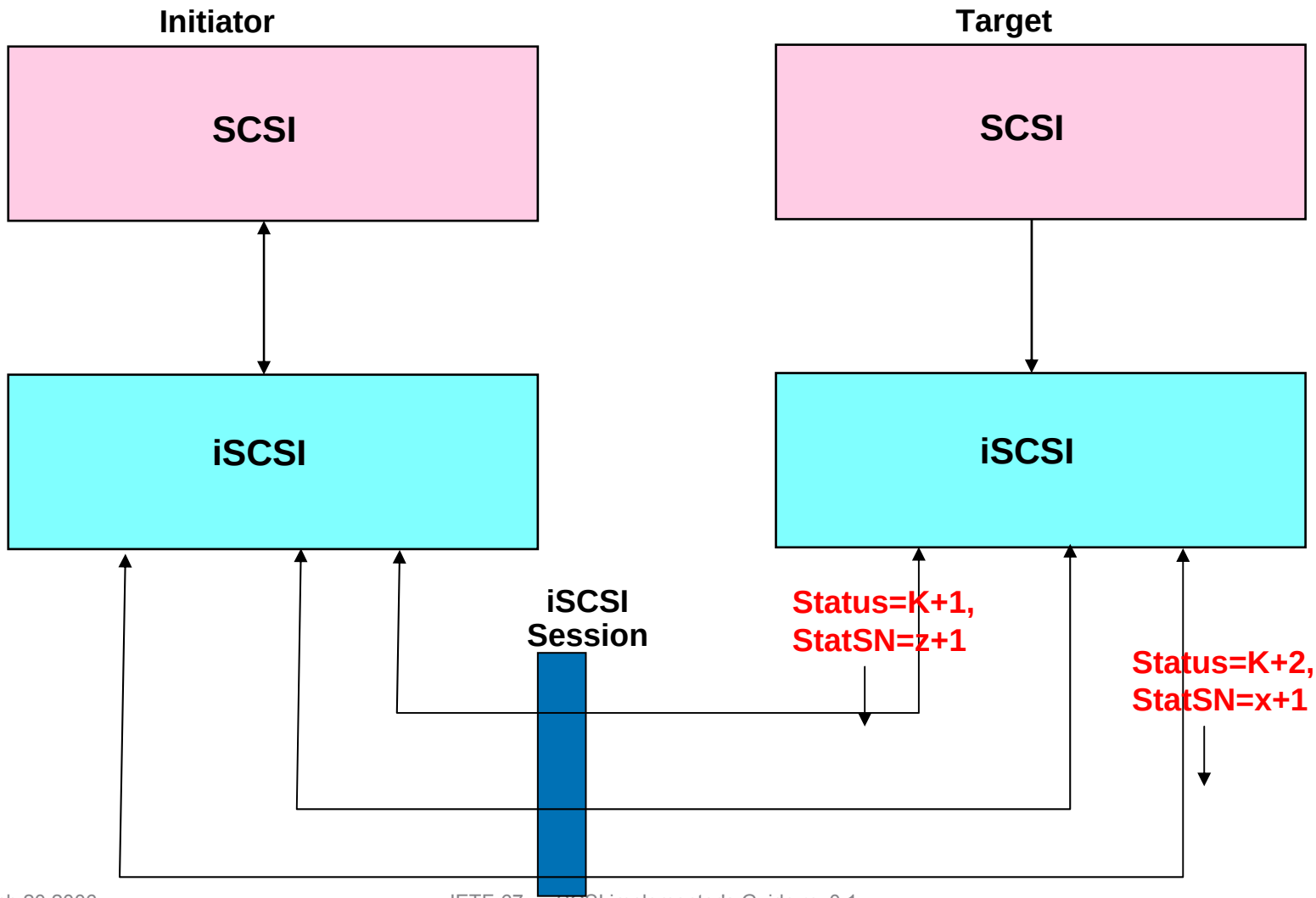
Target iSCSI action: Step.4

Wait for StatSN ack on Status=K. Hold back all the new responses.



Target iSCSI action: Step.5

Release all waiting statuses as usual. Fence is lifted.



Task Management: Problem statement

- Multi-task aborting with RFC 3720 semantics *could* take a long time….
  - CmdSN wait on third-party sessions
  - TTT wait on issuing & third-party sessions
  - Third-party StatSN acknowledgment waits
- If a task is hung on any initiator, TTT waits forever, the abort never completes…..
- Wanted: a new efficient Task management model that does:
  - Eliminate waits wherever possible
  - Fully backward compatible with RFC 3720 implementations
  - Should work identically for traditional iSCSI & RDMA-supported iSCSI
  - No new SCSI-level requirements (e.g. TAS=1)

Proposal Outline

1. Eliminate CmdSN/StatSN wait - all cross-nexus wait summarily.
   - Would still be SAM-2-compliant.

   - ~~CmdSN wait on third-party sessions~~
   - ~~TTT wait on issuing & third-party sessions~~
   - ~~Third-party StatSN acknowledgment waits~~

2. Define a new iSCSI key FastMultiTaskAbort.

3. When FastMultiTaskAbort=1….
   1. Delink TTT invalidation from TMF completion. Allow TTT invalidation to complete in a lazy fashion.
   2. Introduce a new Async event for target iSCSI layer to rely on to free up the TTTs (and the data buffers) – after the TMF is finished.
   3. Allow an “escape hatch” for implementation-dependent timeout for TTT reclaim (also must drop the connection)

4. When FastMultiTaskAbort = 0…..
   1. Follow RFC 3720 semantics (i.e. TMF finishes only after TTTs are invalidated)

Why not simply require lazy behavior always?

Why not propose “lazy invalidation of active TTTs on the issuing session” as the default behavior (i.e. without the new key)?

Such a mandate differs from the RFC 3720 behavior significantly.

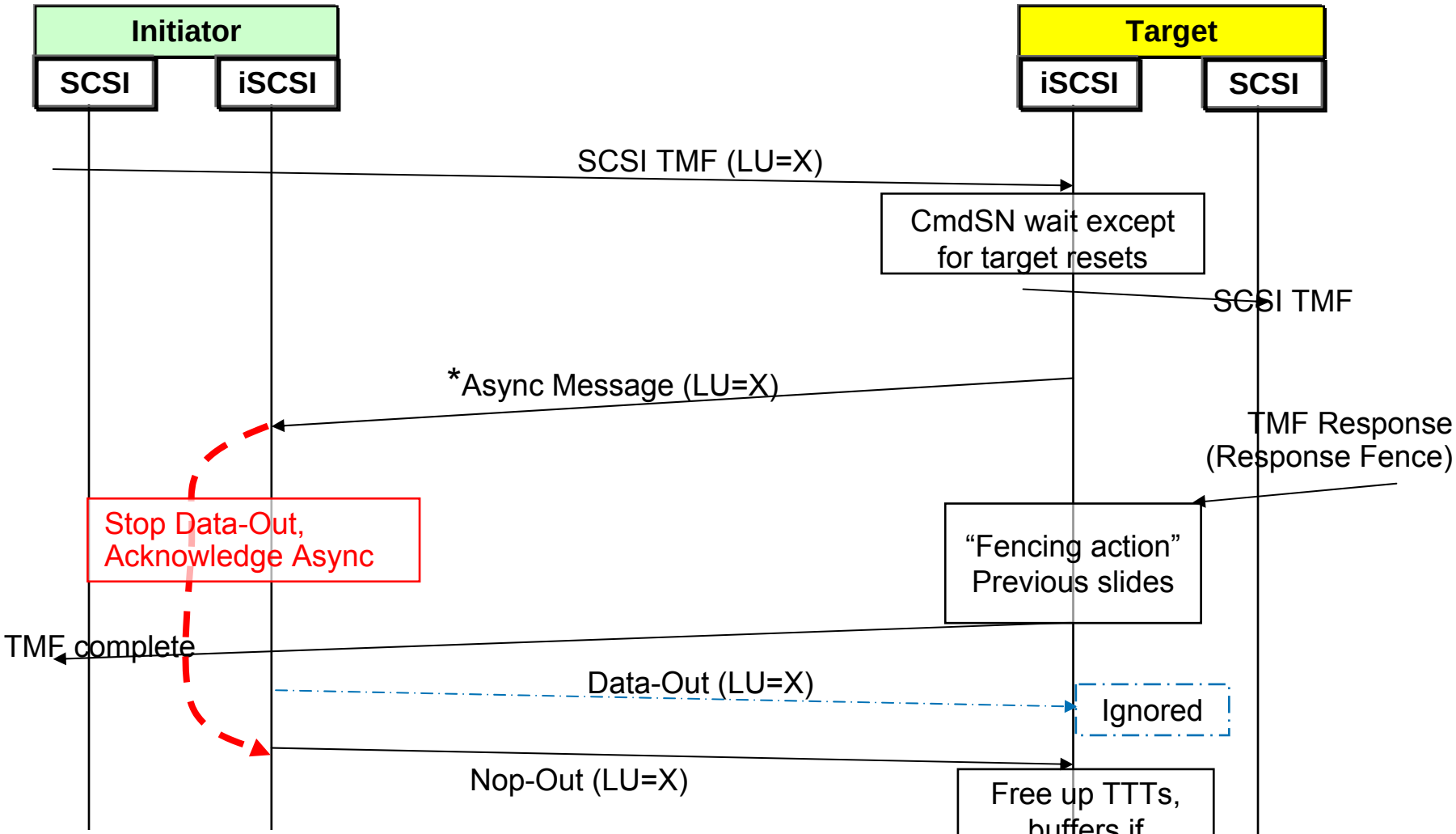
1. RFC 3720 assumes TTTs and their associated buffers do not exist beyond task termination.
2. RFC 3720 implies immediate re-usability of outstanding TTT-LUN pairs to other tasks even on the same session-connection once the task that owns a TTT is terminated.
3. RFC 3720 requires all Data-Out PDUs with an invalid ITT-TTT combination (which it would be if ITT isn’t valid anymore) to be rejected.
4. RFC 3720 does not envision a lazy reclaim of TTTs, so does not have an architected mechanism for a lazy reclaim of TTTs.
5. RFC 3720 requires that initiator should respond to active TTTs if the task is alive (lazy reclaim requires the initiator to stop sending data, but signal the target that it is done with using the TTT).

- In short, a default lazy reclaim cannot meet “fully backward compatible with RFC 3720” goal….

Current draft approach

- Two task management approaches
  - “Clarified Semantics”
  - “Updated Semantics”
- Clarified Semantics
  - RFC 3720 behavior extended to LU Reset & Target Resets.
  - Minus all cross-nexus waits (CmdSN & StatSN).
- Updated Semantics
  - Same as Clarified Semantics, and…
  - Allow FastMultiTaskAbort key negotiation
  - Drop the TTT invalidation wait when FastMultiTaskAbort=1

New Multi-task Abort Timeline



*Async Message step happens once for each LUN+session+connection combination if that combination has an allegiant affected task from the TMF action.

Questions?

Motion

Seek WG consensus on:

- Response fencing
- FastMultiTaskAbort