

Standard Operating Procedure

Document Number: BRP-SYS-001

Standard Operating Procedure (SOP): Bounded Reliable Protocol (BRP) for Safe Message Transfer

Revision: 1.0

Last Updated: [DATE]

Contents

1. Introduction
 - 1.1 Purpose
 - 1.2 System Overview
 - 1.3 Regulatory Compliance
2. System Specifications
 - 2.1 Message Flags and Variables
 - 2.2 Component Modules and States
3. Operational Protocols
 - 3.1 Initialization and Startup
 - 3.2 Request and Acknowledgment Logic
 - 3.3 Message Reading and Writing Sequences
4. Emergency Operations
 - 4.1 Timer-Based Retries
 - 4.2 Error Indication and Recovery
5. Maintenance Requirements
 - 5.1 Flag and State Reset
 - 5.2 Safety Assurance Validation
6. Quality Assurance
 - 6.1 Data Safety Properties
 - 6.2 Control Path Consistency Checks

- 7. Security Protocols
 - 7.1 Loss Prevention
 - 7.2 Toggle and Message Consistency
 - 8. Environmental Considerations
 - 8.1 Message Loss Conditions
 - 8.2 Concurrency and Resource Load Handling
 - 9. Training Requirements
 - 9.1 Protocol State Awareness
 - 9.2 Debugging Failed States
 - 10. Document Control
 - 10.1 Revision History
 - 10.2 Authorization
 - 11. Process Flows and State Transitions
 - 11.1 Main Transfer Path
 - 11.2 Retransmission Timer Workflow
 - 11.3 Indication and Confirmation Transition Logic
-

1. Introduction

1.1 Purpose

- Define reliable transfer of bounded messages using retries and confirmations.
- Prevent loss, duplication, or misordering of messages.

1.2 System Overview

- The protocol relies on structured message transfer using request, write, and confirm sequences.
- Includes a SAFE flag guaranteeing correctness during transitions.
- Timers and retry modules ensure recovery from faults.

1.3 Regulatory Compliance

- Aligned with safety-critical protocol modeling (e.g., AG s.SAFE invariant).
- Suitable for formally verified systems where message loss is unacceptable.

2. System Specifications

2.1 Message Flags and Variables

- Message (`Msg`) has:
 - `first` : Marks the beginning of a message.
 - `last` : Marks the end of a message.
 - `toggle` : Used to distinguish successive messages.
- Protocol states track indicators, requests, confirmations, and acknowledgment.

2.2 Component Modules and States

- Major modules include:
 - `write_req` , `write_K` , `write_L` , `write_ind` , `write_ind_error`
 - `read_req` , `read_conf` , `read_ind` , `read_ind_error` , `read_K` , `read_L_MANY` , `read_L_ONE`
 - Loss modules: `lose_msg` , `lose_ack`
 - Timers: `rtimer` , `stimerTRUE_QUIT` , `stimerTRUE_RETRY` , `stimer2`
 - Control states include WR (write req), SF (send file), WA (wait ack), SC (send conf), WT2 (wait retry)
-

3. Operational Protocols

3.1 Initialization and Startup

- All flags and modules initialized to default safe states.
- State machine variables `SAFE`, `K_full`, `L`, `afile`, etc. set to `FALSE`.

3.2 Request and Acknowledgment Logic

- `write_req` triggers only if request is not full.
- `read_req` transitions state if a request is pending and system is in `WR`.
- Confirmation from `write_conf` concludes the cycle based on remaining file and toggles.

3.3 Message Reading and Writing Sequences

- `write_K` writes the message `K` structure (first/last/toggle).
- `read_K` reads `K` and stores in local message holder.
- `write_L` issues link acknowledgment (`L = TRUE`).
- `read_L_ONE/MANY` transitions file and toggle state depending on `ONE/MANY` flags.

4. Emergency Operations

4.1 Timer-Based Retries

- If message confirmation is not received, retry via `stimerTRUE_RETRY`.
- If timed out without retry, escalate to `stimer2` to reset sequence.

4.2 Error Indication and Recovery

- `write_ind_error` activates if reply fails (`NOK`) and sets error flags.
 - Re-enables retry logic and resets indicator variables.
-

5. Maintenance Requirements

5.1 Flag and State Reset

- After each message sequence, reset:
 - `K_full` , `L` , `ind_full` , `conf_full`
 - `SAFE` , `afile` , `aerror` , `abusy`

5.2 Safety Assurance Validation

- At key transition points (e.g., `write_conf` , `write_ind`), assert `SAFE = TRUE` .
 - Check logical conditions for file correctness and toggle consistency.
-

6. Quality Assurance

6.1 Data Safety Properties

- At all stages, confirm that data movement does not violate `SAFE` invariants.
- Validate message sequence integrity via toggle control.

6.2 Control Path Consistency Checks

- Confirm no illegal state transitions occur.
 - Automata constraints are respected in sequential module behavior.
-

7. Security Protocols

7.1 Loss Prevention

- Modules `lose_msg` and `lose_ack` are bounded and controlled.
- Retry logic ensures message loss is detected and recovered.

7.2 Toggle and Message Consistency

- Message toggle and first/last bits must alternate correctly.
- Prevent reuse of toggle bits without confirmation.

8. Environmental Considerations

8.1 Message Loss Conditions

- Simulates message and acknowledgment loss using `lose_msg`, `lose_ack`.
- Recovers using timers and redundancy.

8.2 Concurrency and Resource Load Handling

- Timer overlaps and concurrent transitions are guarded using enable flags.
 - No overlapping updates to critical state allowed.
-

9. Training Requirements

9.1 Protocol State Awareness

- Operators must be trained in reading protocol state machine transitions.
- Must understand `spc` and `rpc` control state semantics.

9.2 Debugging Failed States

- Familiarity with identifying stuck processes and stalled transitions.
- Use message and timer flags for root cause analysis.

10. Document Control

10.1 Revision History

- Rev 1.0 – Initial SOP based on BRP formal model specification (`brp.txt`)

10.2 Authorization

- Approved by: Formal Protocol Systems Architect
- Reviewed by: Real-Time Safety Assurance Committee

11. Process Flows and State Transitions

11.1 Main Transfer Path

- `write_req` → `read_req` → `write_K` → `read_K` → `write_ind` → `write_L` → `read_L` → `write_conf`

11.2 Retransmission Timer Workflow

- `stimerTRUE_RETRY` triggers retry into SF (send file).
- `stimerTRUE_QUIT` ends path to SC (send conf).
- `stimer2` reverts system to WR (write req).

11.3 Indication and Confirmation Transition Logic

- `read_K` sets local message state.
- `write_ind` sets flags based on message boundaries (FIRST, INCOMPLETE, LAST).
- `write_conf` updates protocol status and clears busy state.