

---

# **Efficient Synchronization of Linux Memory Regions over a Network: A Comparative Study and Implementation (Notes)**

A user-friendly approach to application-agnostic state synchronization

Felicitas Pojtinger (Stuttgart Media University)

2023-08-04

## Contents

- Abstract: A comparative analysis and implementation of various methods for synchronizing Linux memory options over a network
- Introduction
  - Examining Linux's memory management and relevant APIs
  - Use cases for memory region synchronization
- Option 1: Handling page faults in userspace with `userfaultfd`
  - Introduction to `userfaultfd`
  - Implementing `userfaultfd` handlers and registration in Go
  - Transferring sockets between processes
  - Examples of handler and registration interfaces (byte slice, file, S3 object)
  - Performance assessment of this approach
- Option 2: Utilizing `mmap` for change notifications
  - Concept: `mmap` a memory region with `MMAP_SHARED` to track changes in a file
  - Method 1 for detecting file changes: `inotify`
  - Limitations: `mmap` does not generate `WRITE` events
- Option 3: Hash-based change detection
  - Comparing hashes of local and remote `mmaped` regions
  - Evaluation of hashing algorithms
  - Introduction to delta synchronization (e.g., `rsync`)
  - Custom protocol for delta synchronization
  - Multiplexing synchronization streams
  - The function of `msync`
  - Performance assessment of this approach
- Option 4: Detecting changes with a custom filesystem implementation
  - Intercepting writes to the `mmaped` region using a custom filesystem
  - Exploring methods for creating a new, custom Linux filesystem
    - \* In the kernel
    - \* `NBD`
    - \* `CUSE`
    - \* `BUSE`
    - \* `FUSE`
    - \* Upcoming options (`ublk`, etc.)
  - Detailed analysis of the `NBD` protocol (client & server)
  - Implementing the client and server in Go based on the protocol

- Server backend interface and example implementations
  - Asynchronous writeback protocol and caching mechanism
  - Performance assessment of this approach
- Summary:
  - Comparing options in terms of ease of implementation, CPU load, and network traffic
  - Identifying the optimal solution for specific use cases: data change frequency, kernel/OS compatibility, etc.