Automotive Digital Cockpit

Comprehensive Project Report

Author: Abhishek Chauhan

Introduction

Auto-generated report for project located at: /home/l-100791/ADC

Project Files & Analysis

backend/.env

PORT=5000 DB_URL=mongodb://localhost:27017/vehicle_health API_KEY=123456. PORT=5 DB_URL=mongODb://127.0.1.1:5000/Vehicle_Health. Port=5000DB_URL="http://www.mogodb.com/vitals/vehicles/health" API_key="123456" PORT="5000" DB_url="http/www/mogoda.com" DB URL=http:// www.mogsoda.org/viles/vehicular-health. Pport=5000Database URL=" http://www/www.mgodb

backend/CMakeLists.txt

Cockpit. cmake_minimum_required(VERSION 3.10) project(AutomotiveDigitalCock Pit) set(CMAKE_CXX_STANDARD 17) include_directories(include) add_executable(cockpit main.cpp)

backend/Dockerfile

Cake is a tool for building a Cockpit in the Linux operating system. It can be used to build and test Linux applications. It is based on the popular C# language.

backend/include/api.hpp

Ifndef API_HPP define API-HPP. ifndef define API_Papers. defineAPI-Papers, API-Parses. defineapi-parses,API-Portages.

Module/Top comments:

ifndef API_HPP
define API_HPP

backend/main.cpp

include <iostream> include "include/api.hpp" include "includes/api" include 'include/include/hpp' include 'includes/includes/hpps'

Module/Top comments:

include <iostream>
include "include/api.hpp"

Functions / Methods:

- int main() {

backend/model/interface.cpp

include <iostream> include iostREAM. include iostream. include include include <ioboom.com> include ioboom, iobom.com, and iiboom.

Module/Top comments:

include <iostream>

Functions / Methods:

- void connectFrontend() {

backend/server.js

Dependencies: 1. --- 1. Dependencies: 2. --- 2. Dependence: 3. --- 3. Dependency: 4. --- 4.

Module/Top comments:

--- 1. Dependencies ---

backend/src/api.cpp

include <iostream> include "api.hpp" include "API.hPP" include "api" include "api" includes "api", "api," "api".hpp.

Module/Top comments:

include <iostream>
include "api.hpp"

Functions / Methods:

- void initAPI() {

backend/src/telemetry_handler.cpp

include <iostream> include <fstream> include include <string> include </fstream>. include </string>. include <stream] include </stream>

Module/Top comments:

include <iostream>
include <fstream>
include <string>

Functions / Methods:

- while (std::getline(file, line)) {

- void readTelemetryData() {

frontend/.gitignore

CNN.com will feature iReporter photos in a weekly Travel Snapshots gallery. Please submit your best shots of the U.S. for next week. Visit CNN.com/Travel next Wednesday for a new gallery of snapshots.

Module/Top comments:

Logs

frontend/README.md

CNN.com will feature iReporter photos in a weekly Travel Snapshots gallery. Please submit your best shots of New York for next week. Visit CNN.com/Travel next Wednesday for a new gallery of snapshots.

Module/Top comments:

React + Vite

frontend/api.js

It's a good practice to have this in one place. Define the base URL of your backend API. It's also a good idea

to have it in a single place.

Module/Top comments:

Define the base URL of your backend API.
It's a good practice to have this in one place.

Functions / Methods:

- if (!response.ok) {

frontend/eslint.config.js

import js from '@eslint/js' import globals from 'globals' import reactRefresh from 'eslints/react-refresh' import { defineConfig, globalIgnores, { sourceType: ' module' }); export default defineConfig. defineConfig([globalIgnore(['dist']), { files: ['**/*.{js,jsx}'], extends: [JS.configs.recommended, JS. configs['recommended-latest'], ReactRefresh.Configs.vite,. reactHooks.configS.vites, reactH hooks.vits, }); defineConfig: defineConfig

frontend/src/components/UserList.js

const UserList = () => { const [users, setUsers] = useState([]), [error, setError] =UseState(null), [isLoading, setIsLoading] = UseState(true) }); const fetchedUsers = await getUsers(); setUsers(fetchedUsers), setError(null); catch (err) { // Call the function when the component mounts. return <div style={{ color: 'red' }}>Error: {error} ; }

export default UserList; export default userList. return (UserList, user, key, user.id, key.name, users.map)

Functions / Methods:

- if (isLoading) {

- if (error) {

frontend/src/server/server.js

React component fetching data from backend API. React component is a React component that fetches data from the backend API of a React app.

Module/Top comments:

Example React component fetching data from backend API

Functions / Methods:

- function DataFetcher() {

- function DataFetcher() {

frontend/src/socket.js

const SOCKET_URL = 'http://localhost:3000'; // Create the socket connection export const socket = io(SOCKET _URL); export const sendMessage = (message) => { socket.emit('message', message); }); export constSendMediaUpdate = (state) => socket, state; export const SendClimateUpdate = socket, data, data;

frontend/vite.config.js

vite.dev/config/ export default defineConfig. import { defineConfig } from 'vite' import react from '@vitejs/ plugin-react'

reports/project_report.txt

This report was generated automatically by scanning the codebase at: /home/l-100791/ADC. It summarizes file metrics (lines of code, blank lines, comment lines, functions, classes), detects common dependency manifests, and collates designed diagrams.

/home/l-100791/ADC/frontend/node_modules/react-dom/cjs/ React-dom.development.js [JavaScript] LOC=29923 funcs=900 classes=0 /home/./l/100792/ADc/frontended/node-entry.js (JavaScript) LOC=27273 funcs =349 classes=123 /home./l://www.google.com/search?q=react-icons/pi/index.d.ts [TypeScript]

Design Diagrams

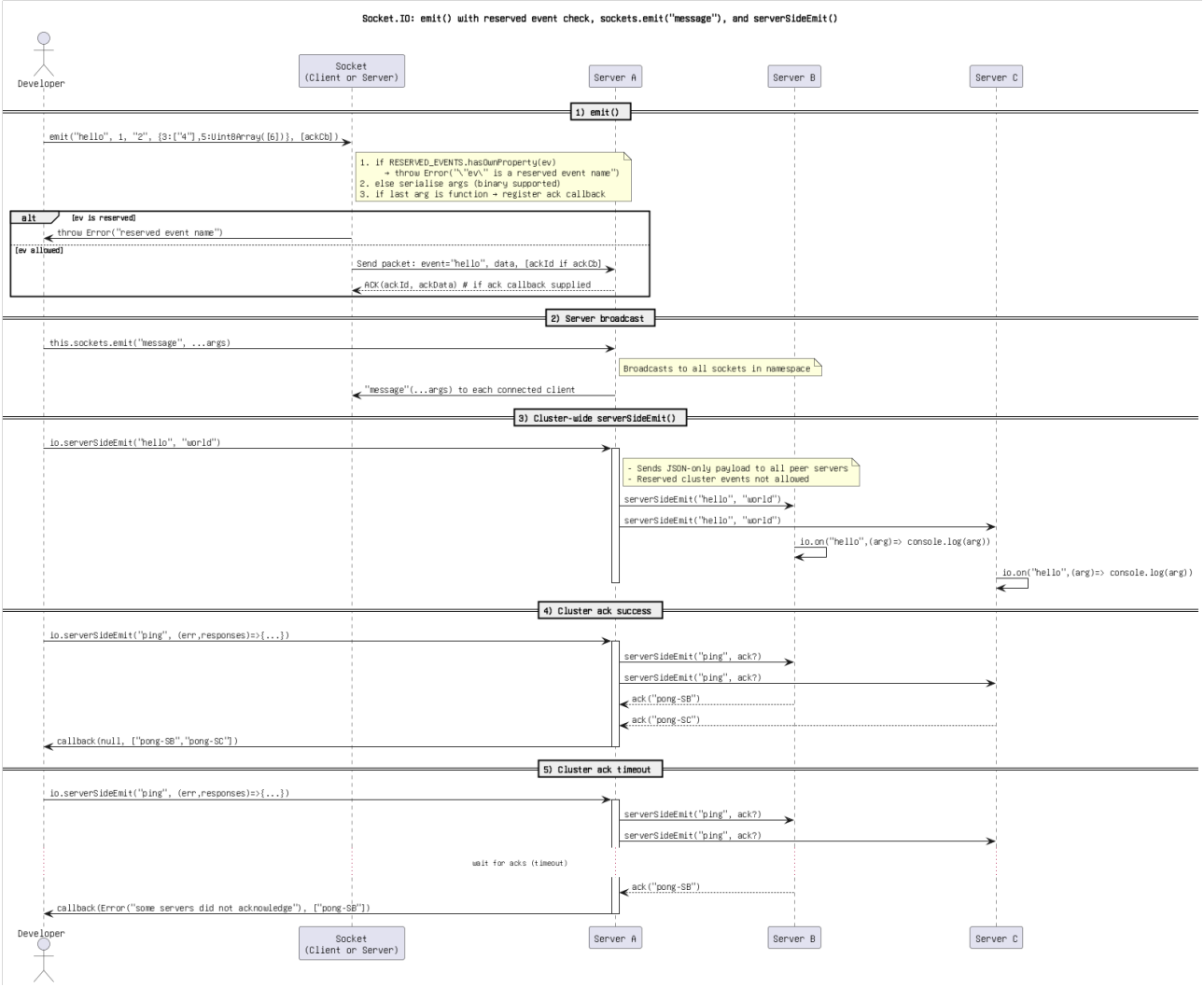

Figure: 1. @startuml title Socket.IO_ emit() with reserved event check, sockets.emit(_message_), and serverSideEmit() actor Developer as Dev participant _Socket_n(Client or Server)_ as Sock participant _Server A_ as SA participant _.png

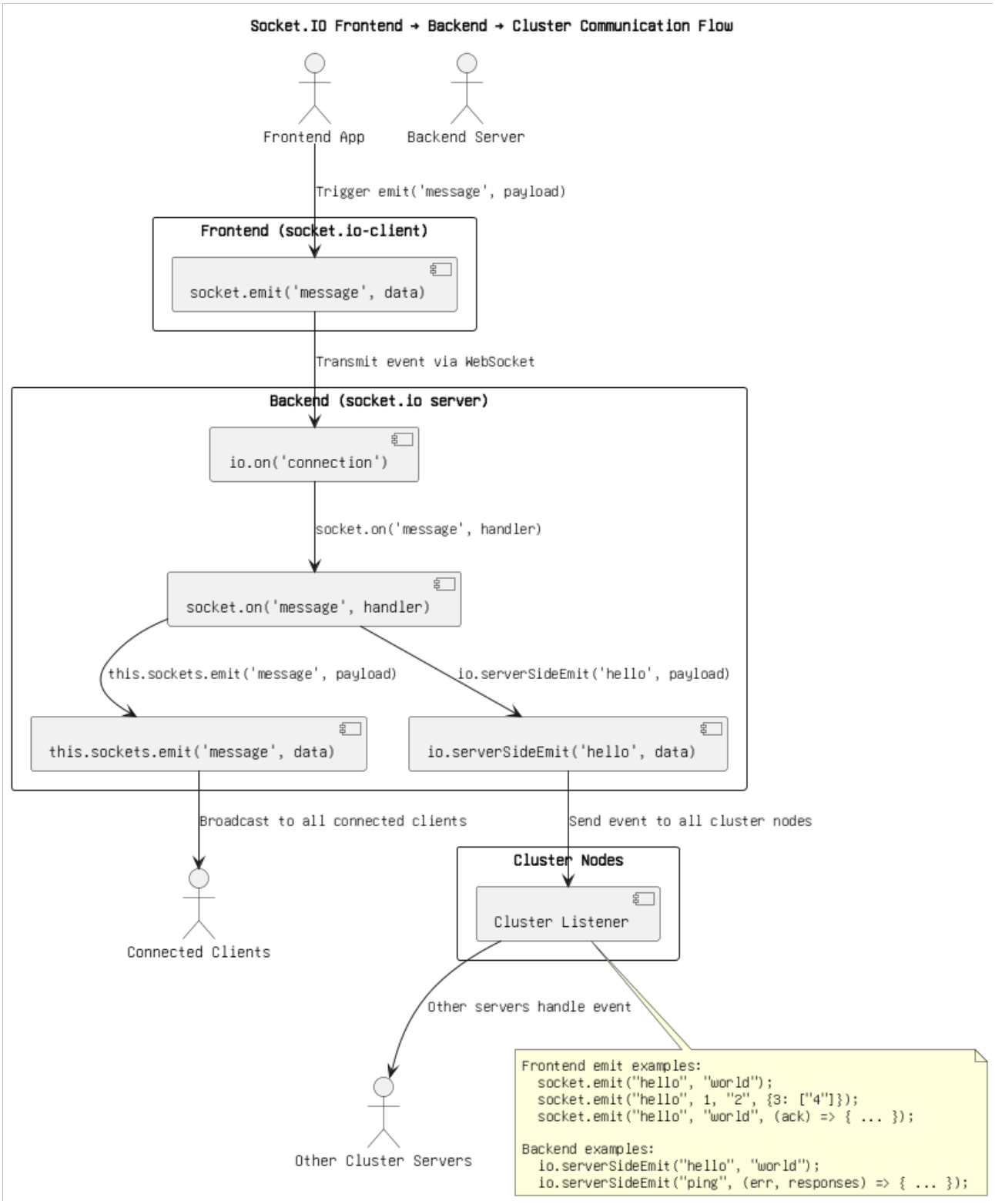

Figure: 2. Cluster Communication Flow (Copy).png

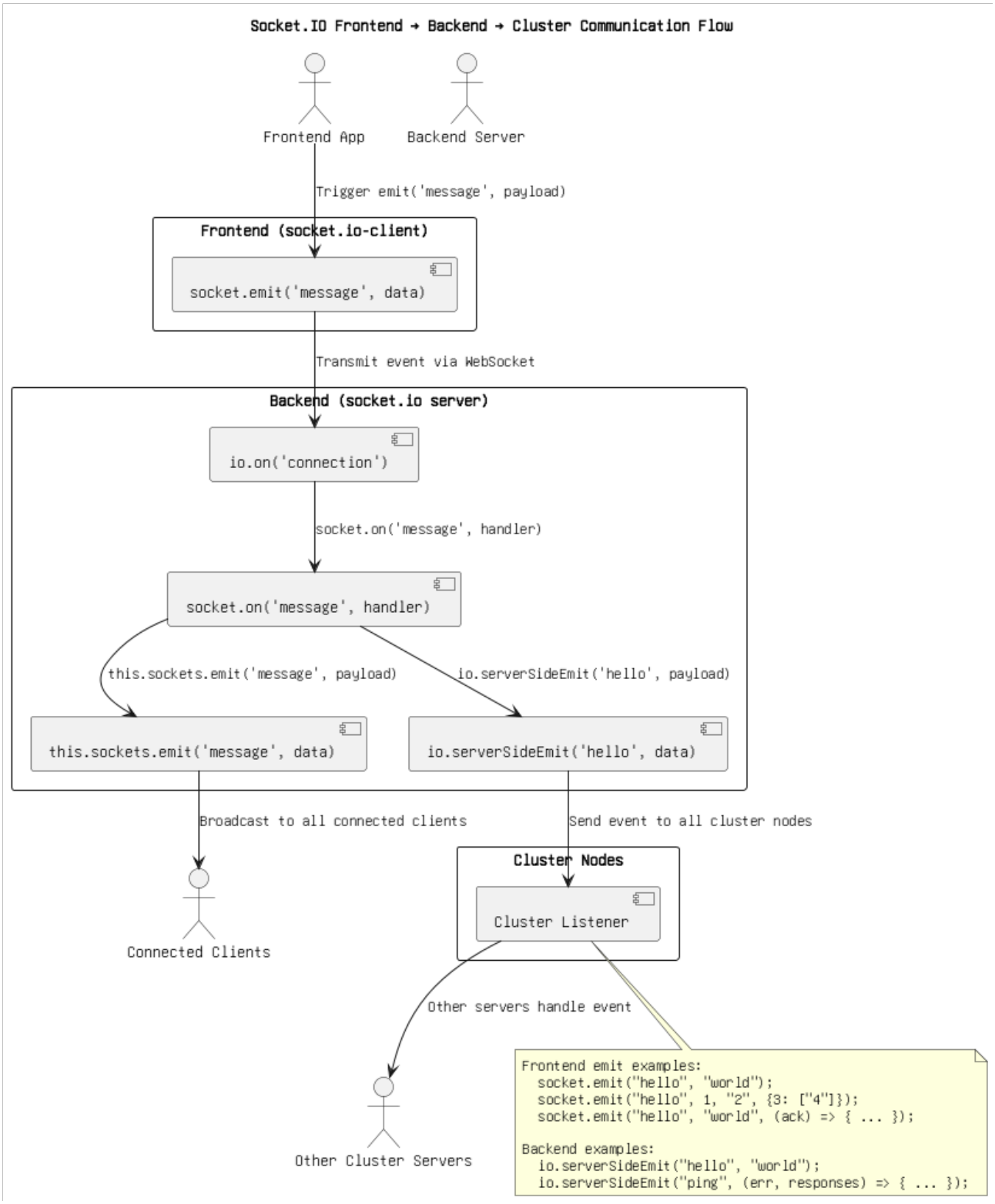

Figure: 3. Cluster Communication Flow.png

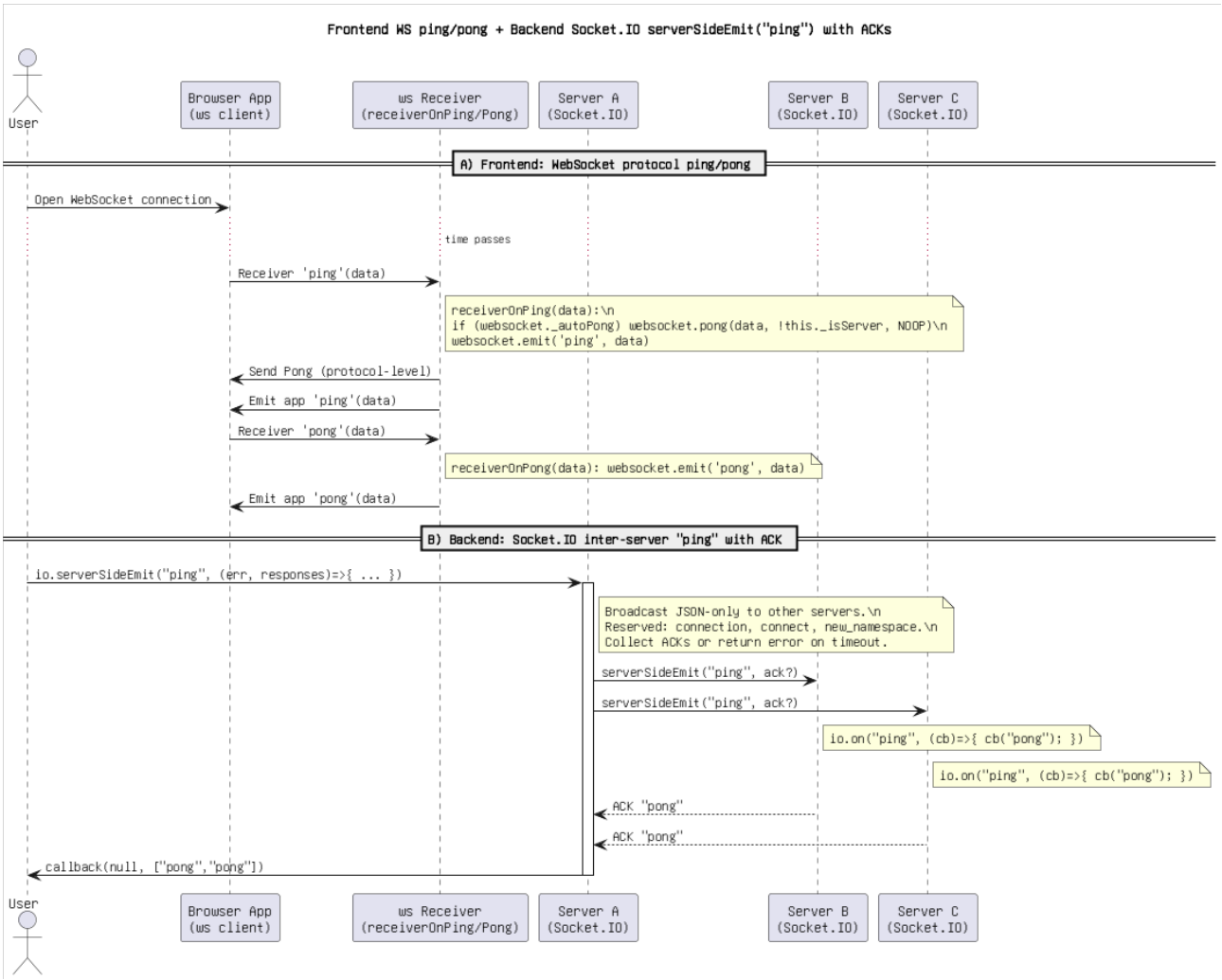

Figure: 4. Frontend WS ping_pong + Backend Socket.IO serverSideEmit(_ping_) with ACKs(1).png

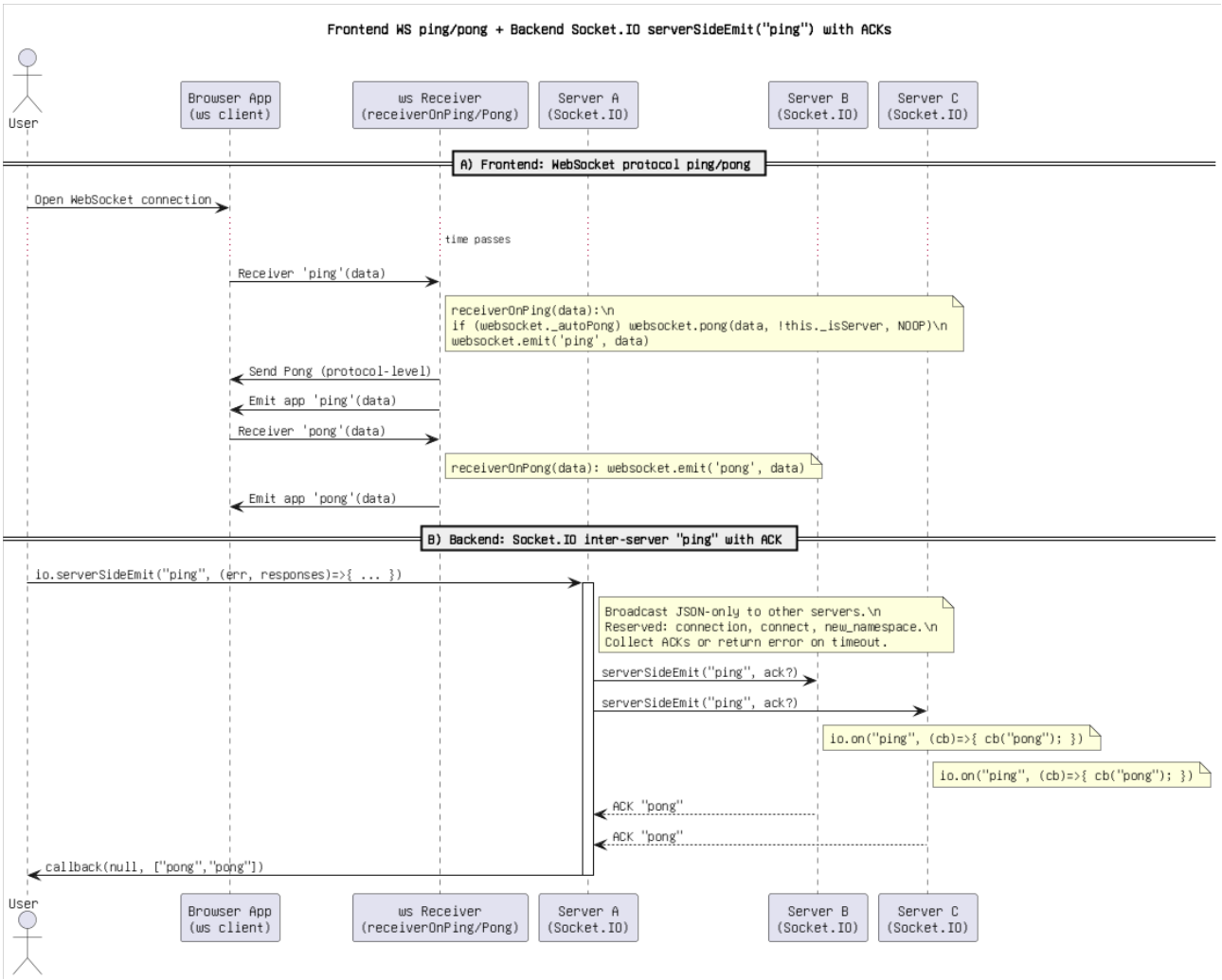

Figure: 5. Frontend WS ping_pong + Backend Socket.IO serverSideEmit(_ping_) with ACKs(2).png

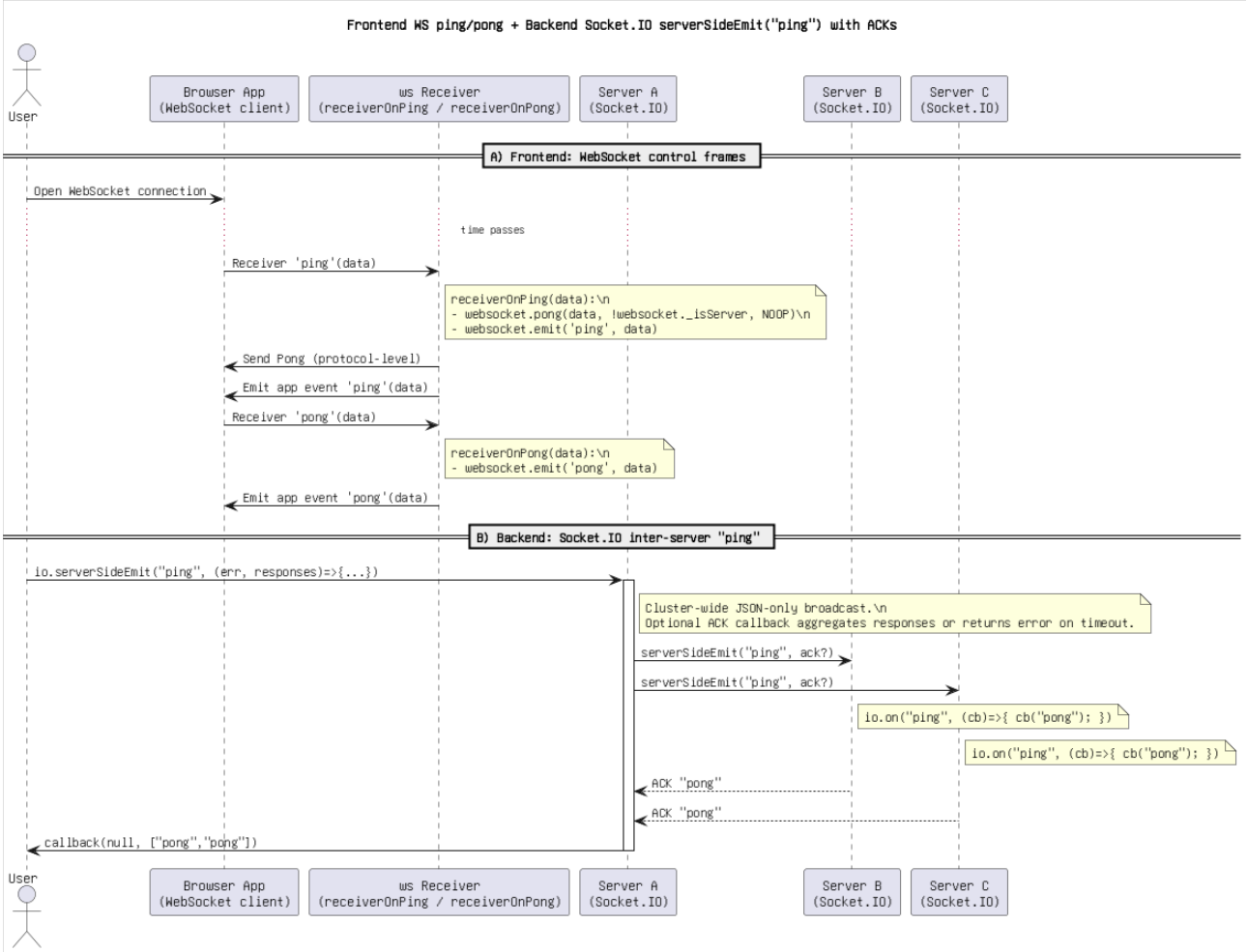

Figure: 6. Frontend WS ping_pong + Backend Socket.IO serverSideEmit(_ping_) with ACKs.png

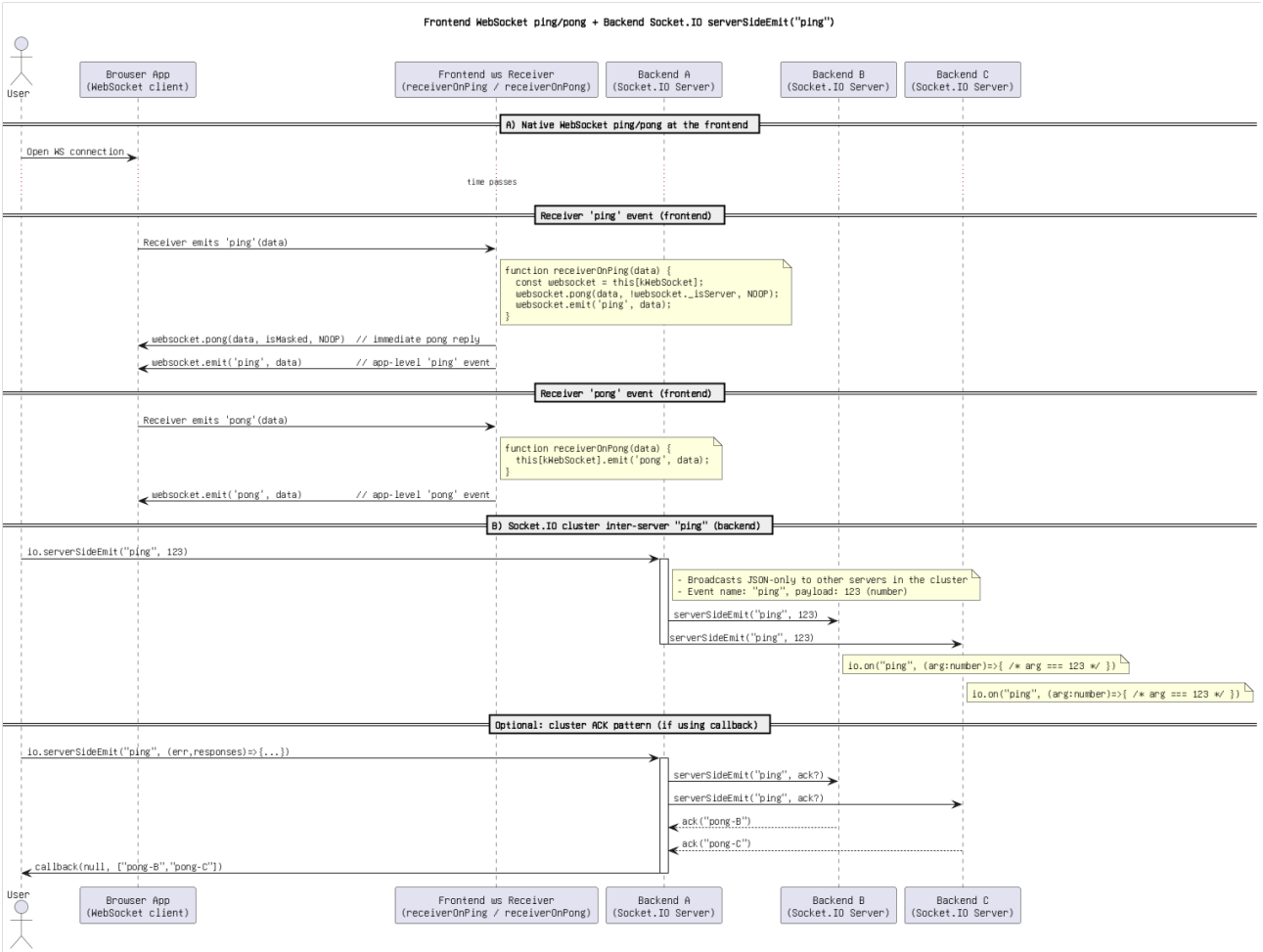

Figure: 7. Frontend WebSocket ping_pong + Backend Socket.IO serverSideEmit(_ping_).png

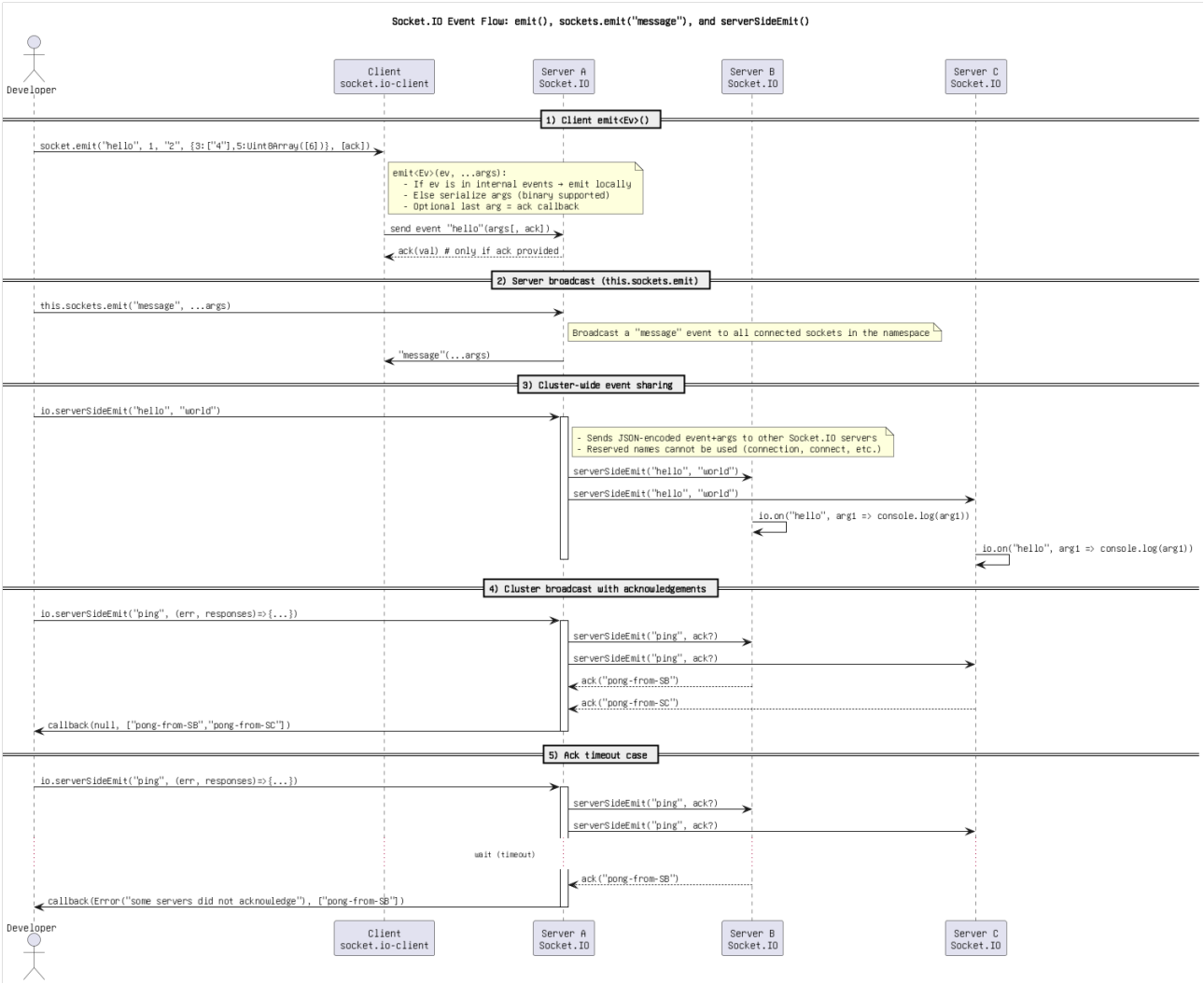

Figure: 8. Socket.IO Event Flow_ emit(), sockets.emit(_message_), and serverSideEmit().png

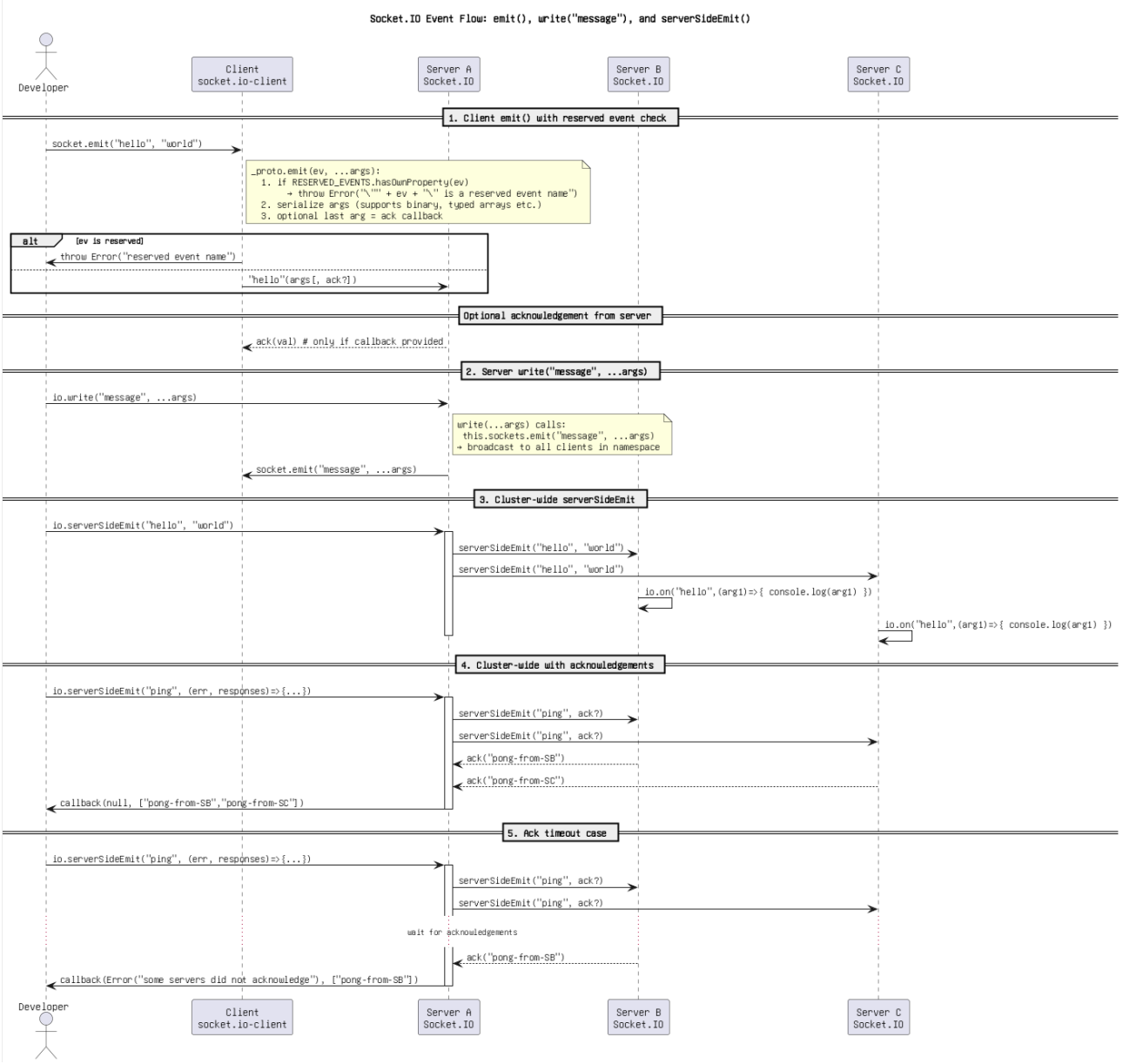

Figure: 9. Socket.IO Event Flow_ emit(), write(_message_), and serverSideEmit().drawio.png

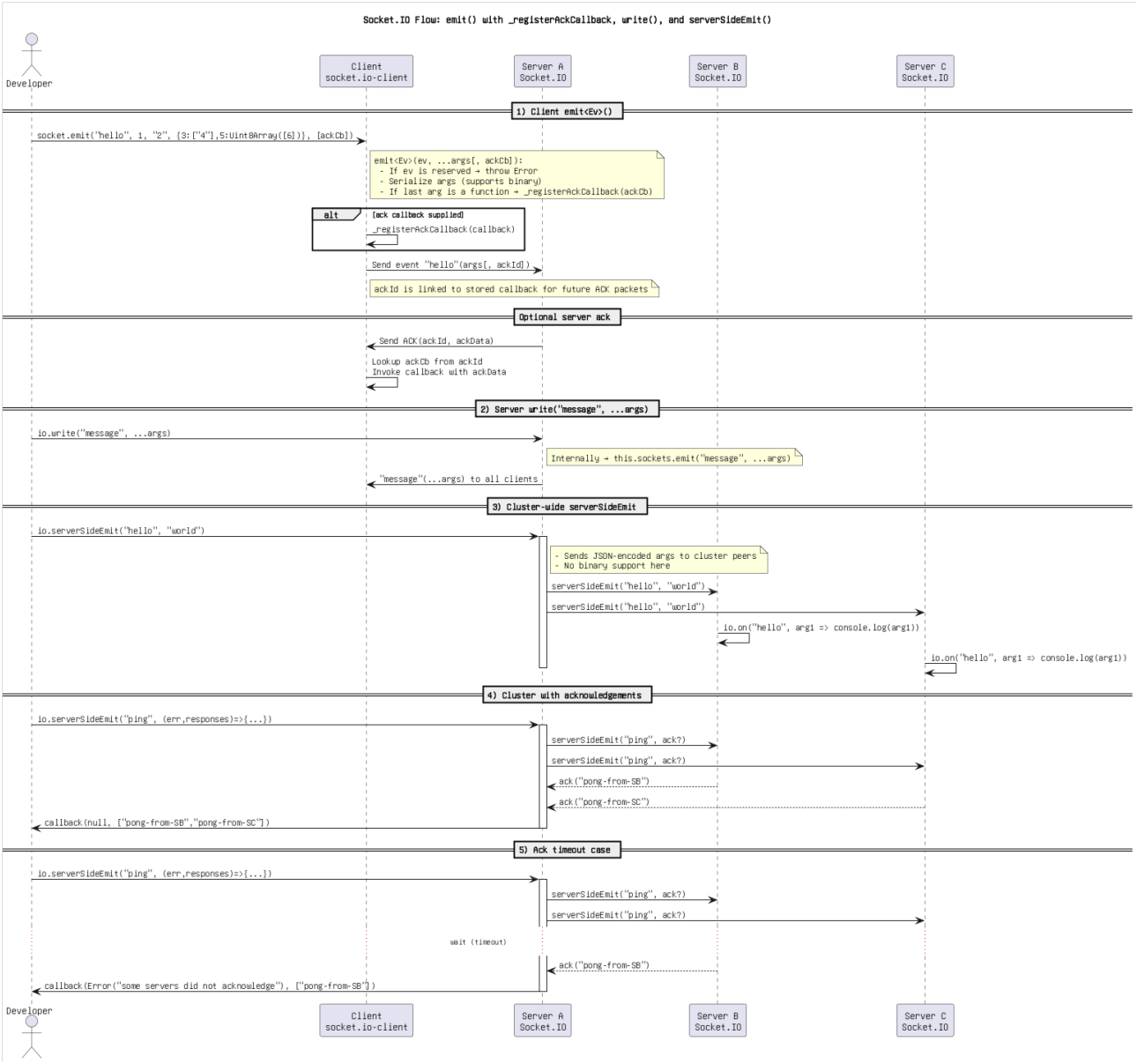

Figure: 10. Socket.IO Flow_ emit() with _registerAckCallback, write(), and serverSideEmit().png

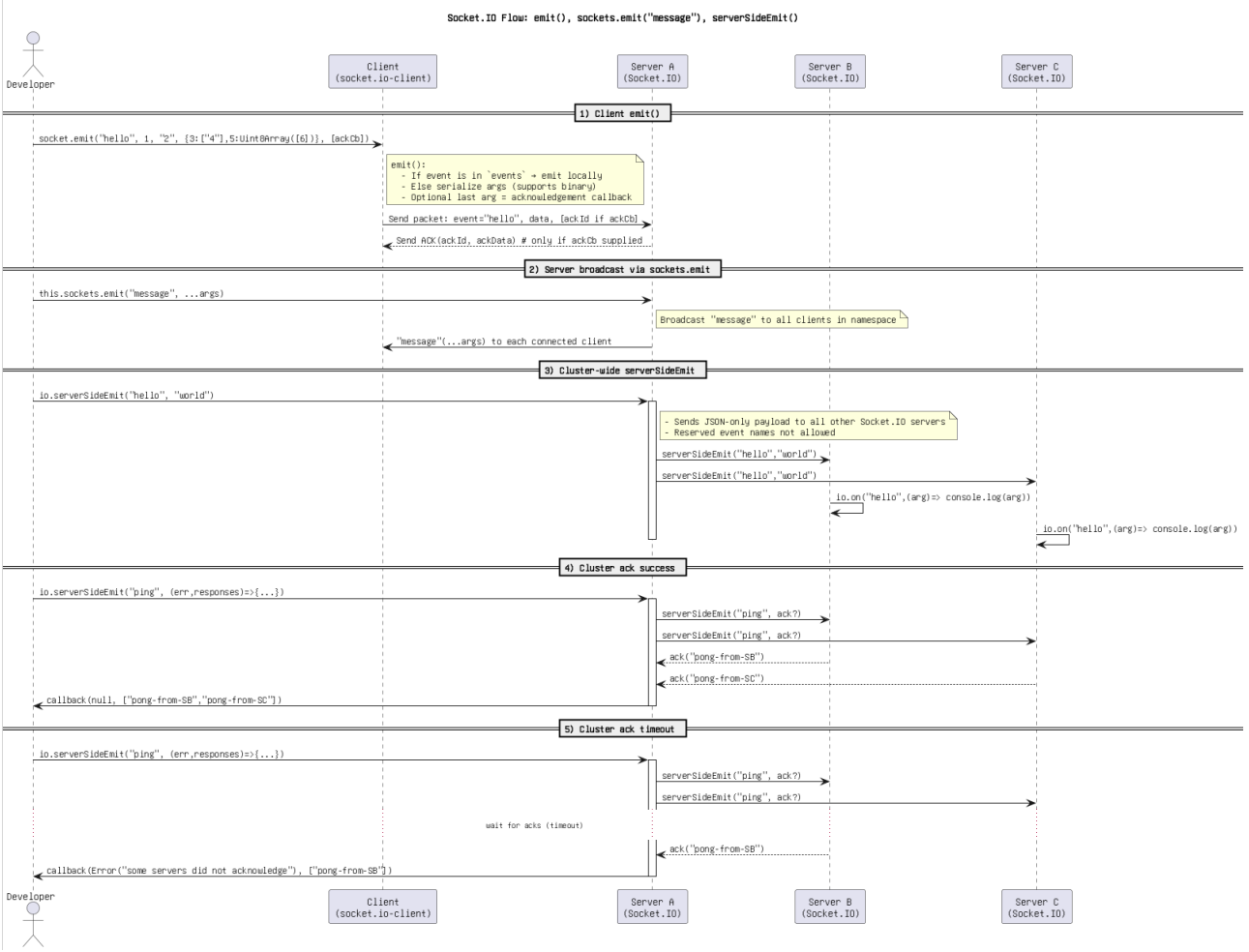

Figure: 11. Socket.IO Flow_ emit(), sockets.emit(_message_), serverSideEmit().png

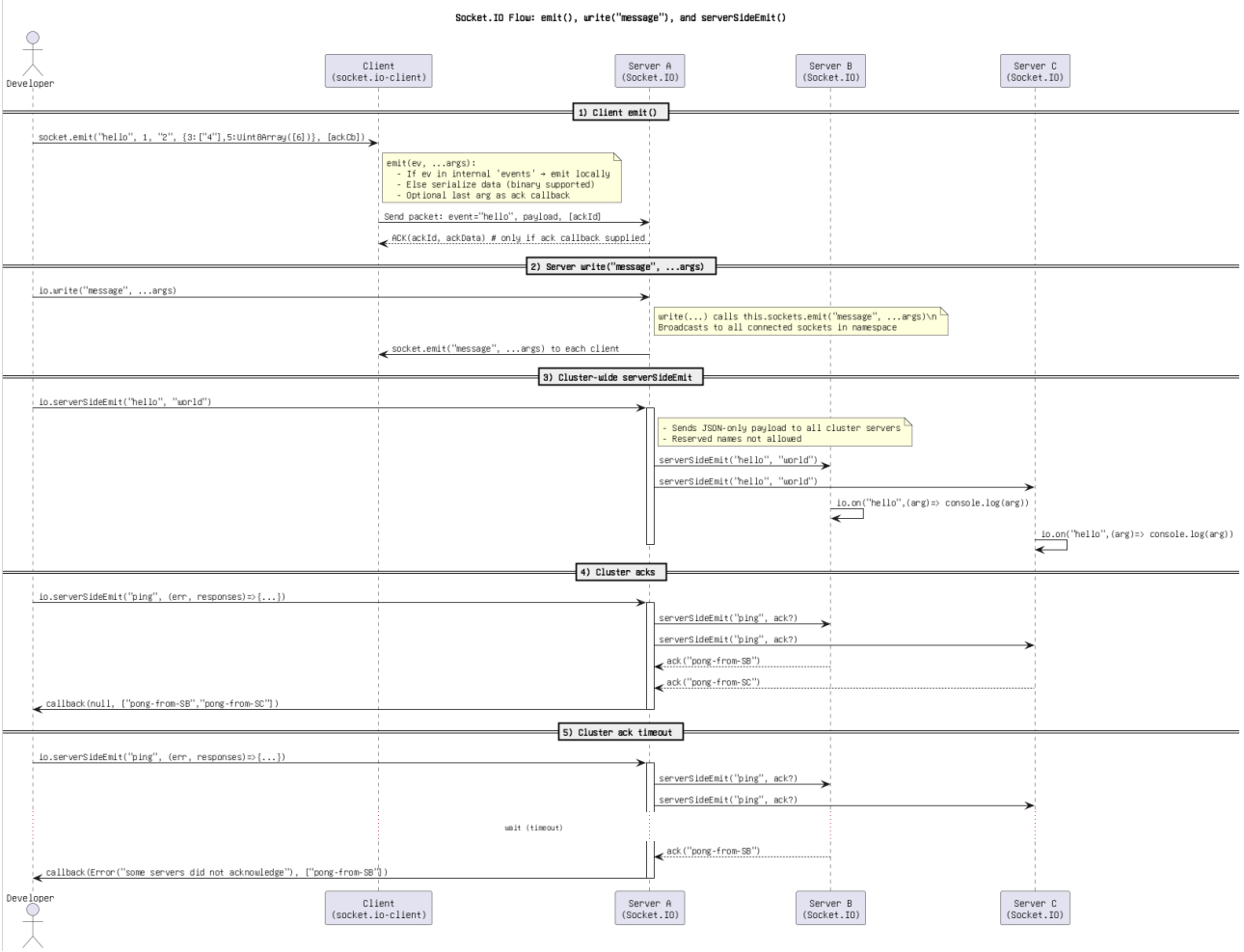

Figure: 12. Socket.IO Flow_ emit(), write(_message_), and serverSideEmit().png

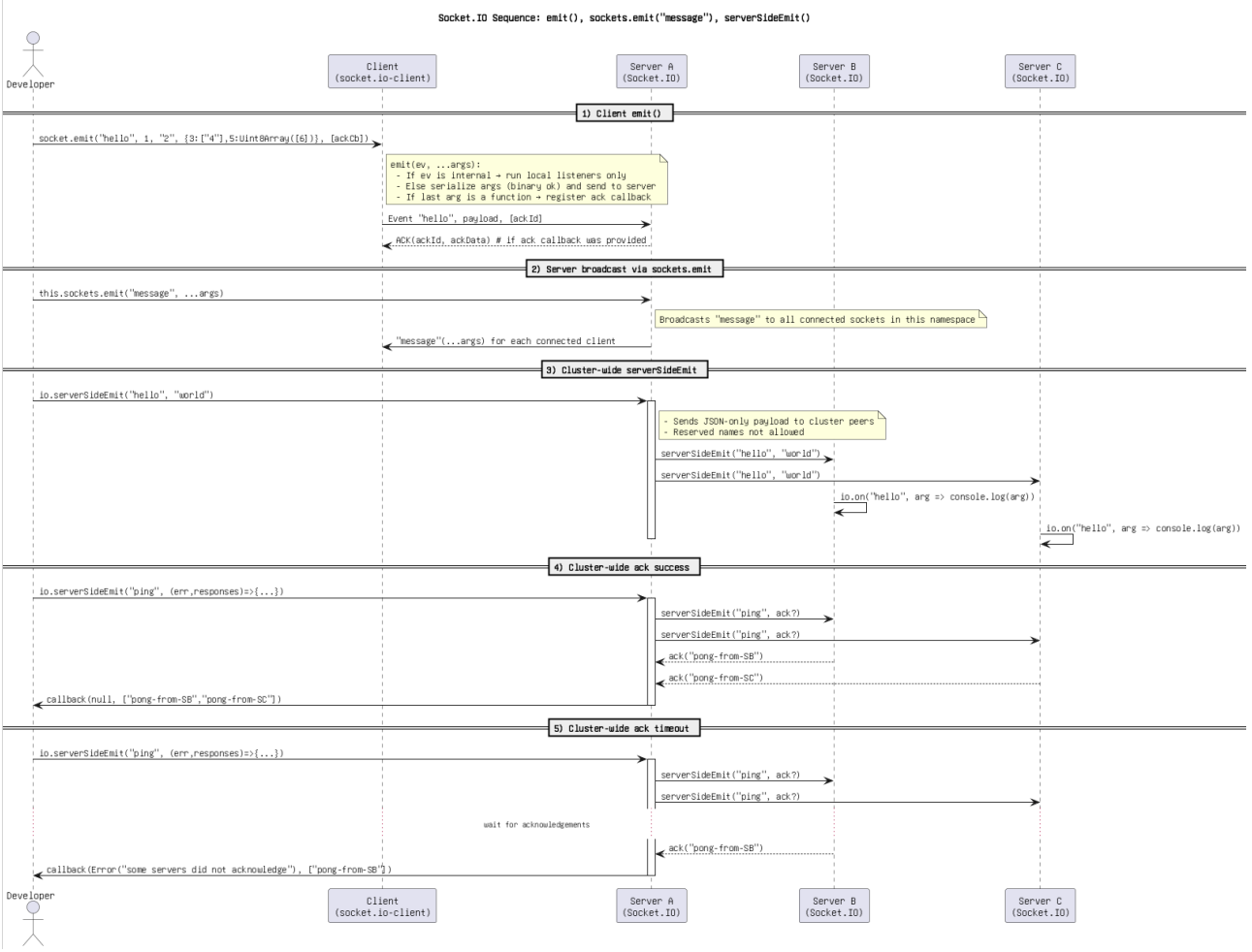

Figure: 13. Socket.IO Sequence_ emit(), sockets.emit(_message_), serverSideEmit().png

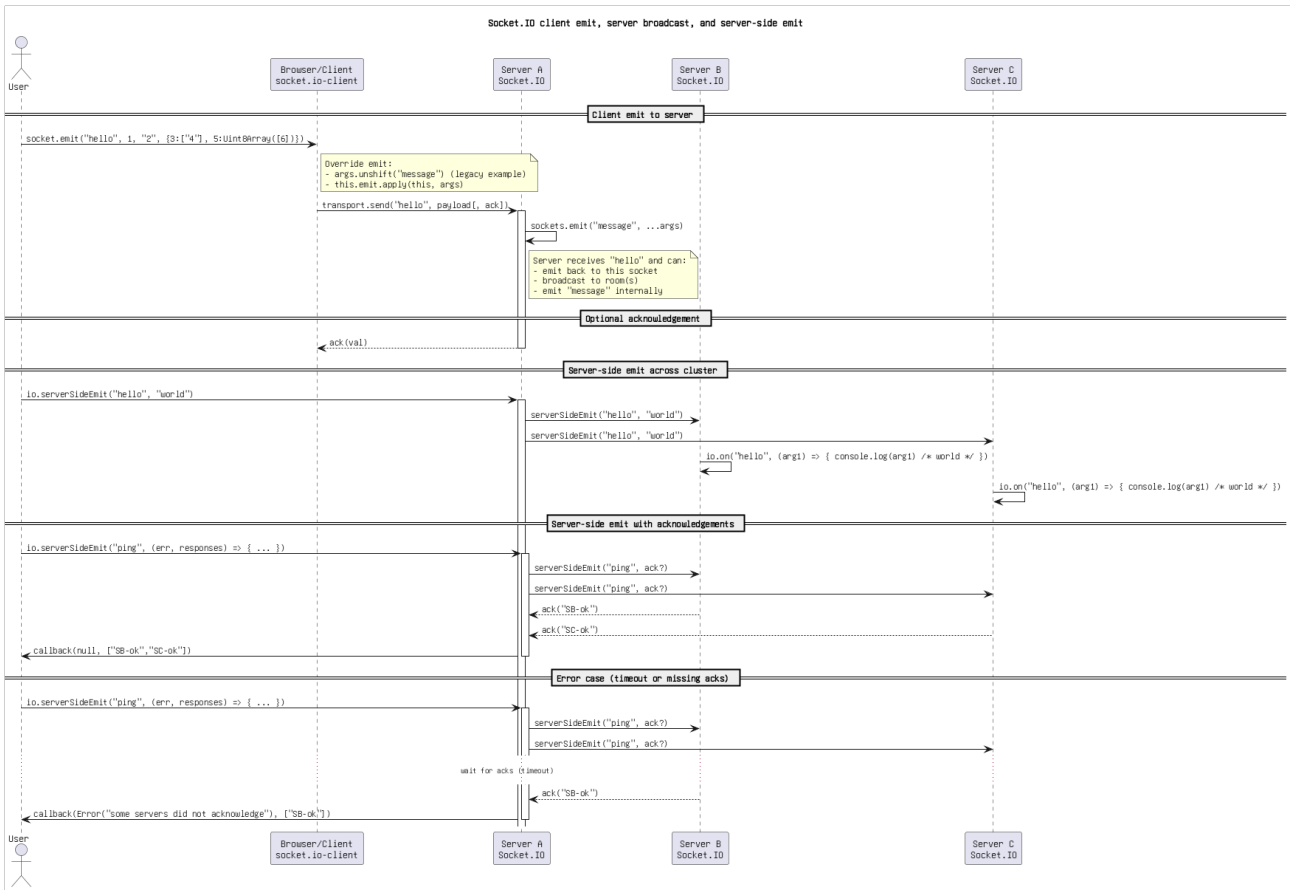

Figure: 14. Socket.IO client emit, server broadcast, and server-side emit.png

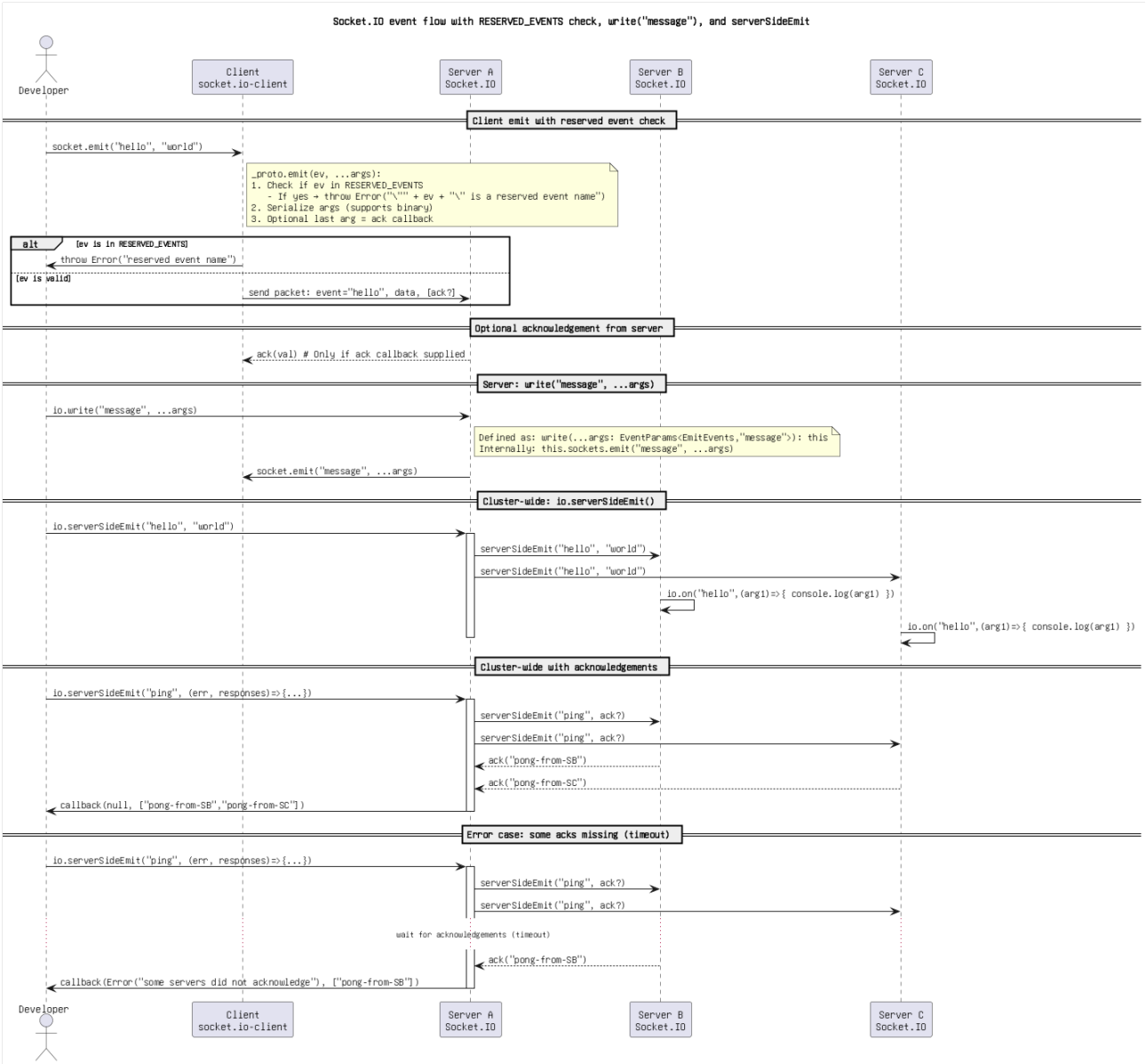

Figure: 15. Socket.IO event flow with RESERVED_EVENTS check, write(_message_), and serverSideEmit.drawio.png

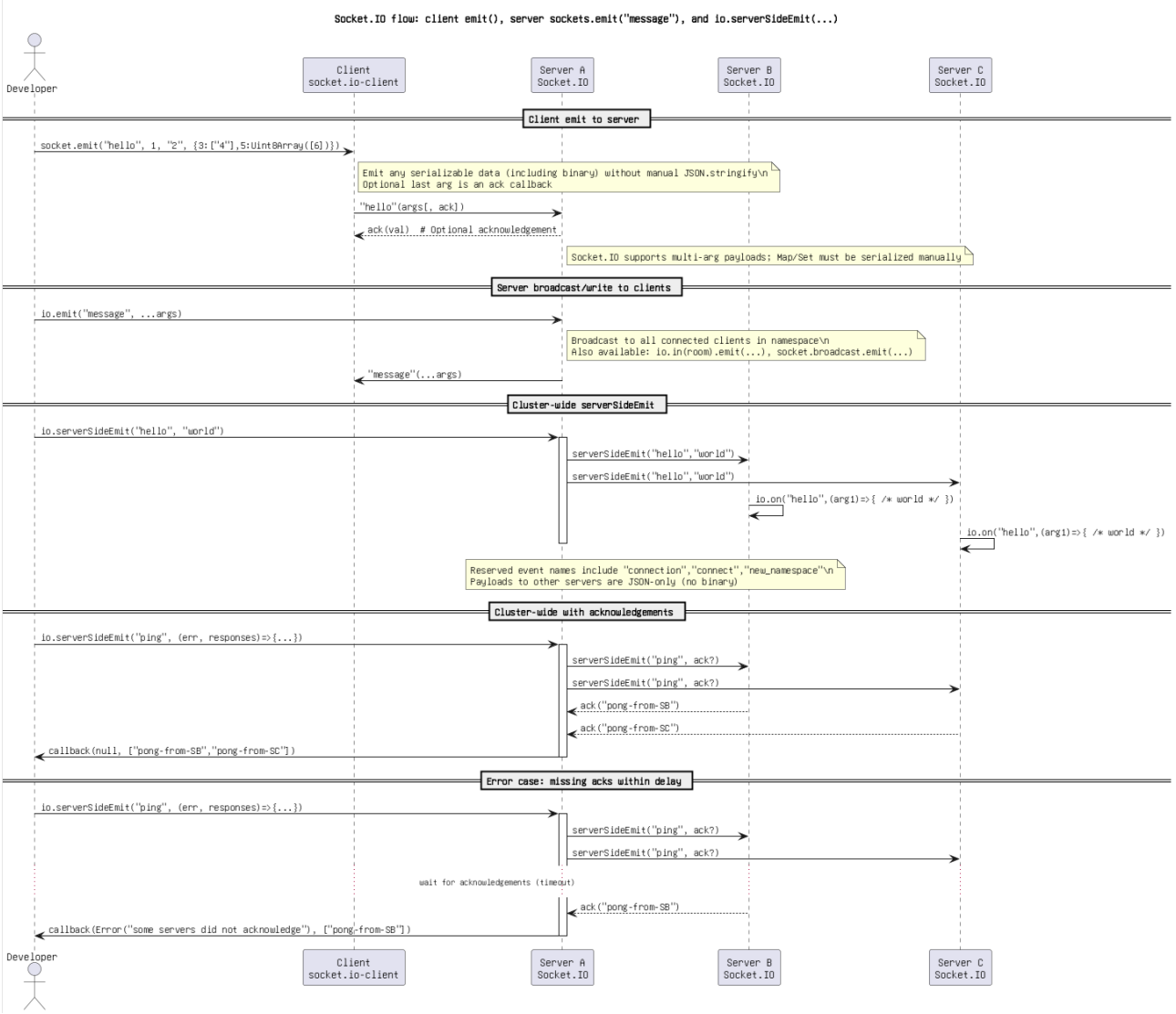

Figure: 16. Socket.IO flow_ client emit(), server sockets.emit(_message_), and io.serverSideEmit(...).drawio.png

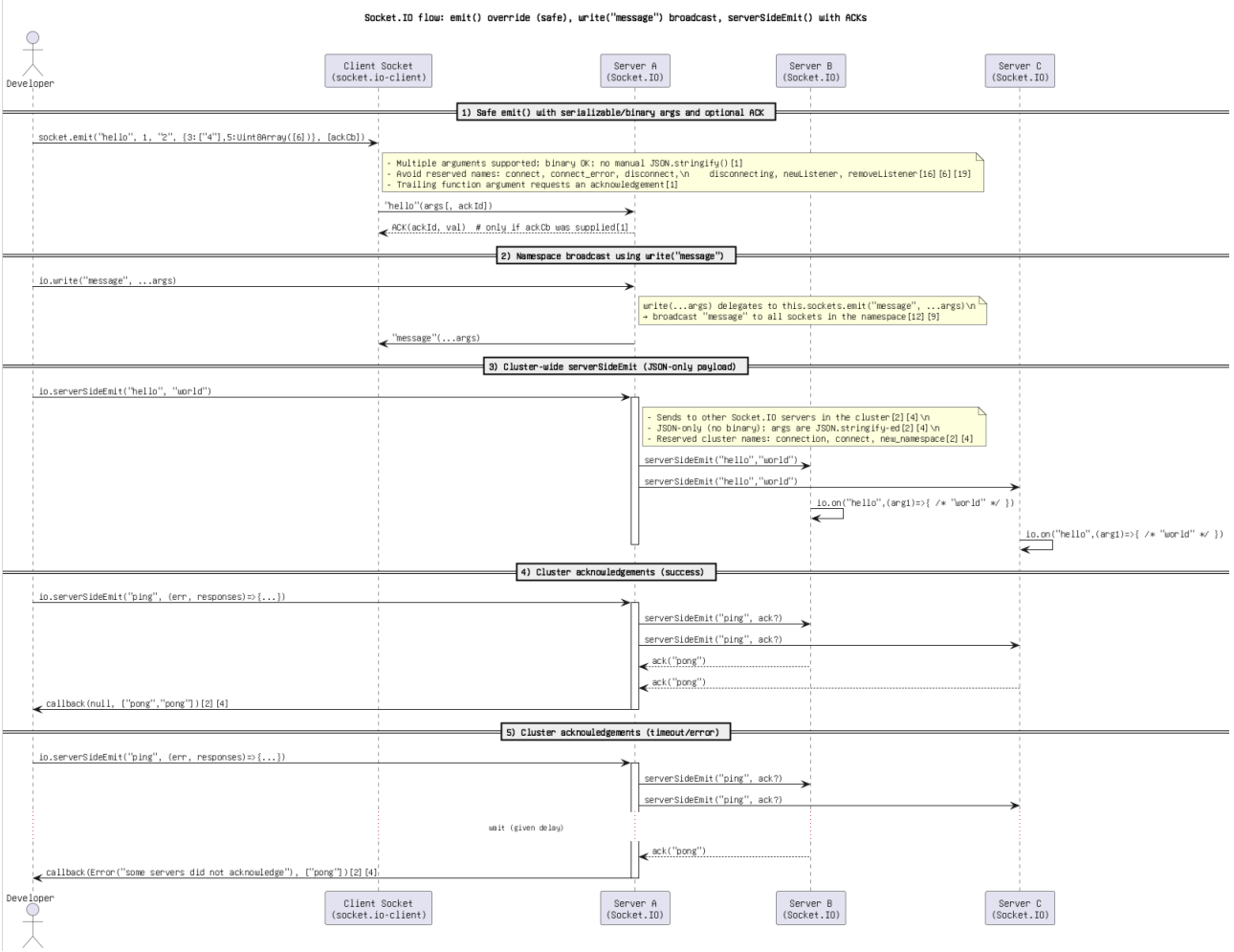

Figure: 17. Socket.IO flow_ emit() override (safe), write(_message_) broadcast, serverSideEmit() with ACKs.png

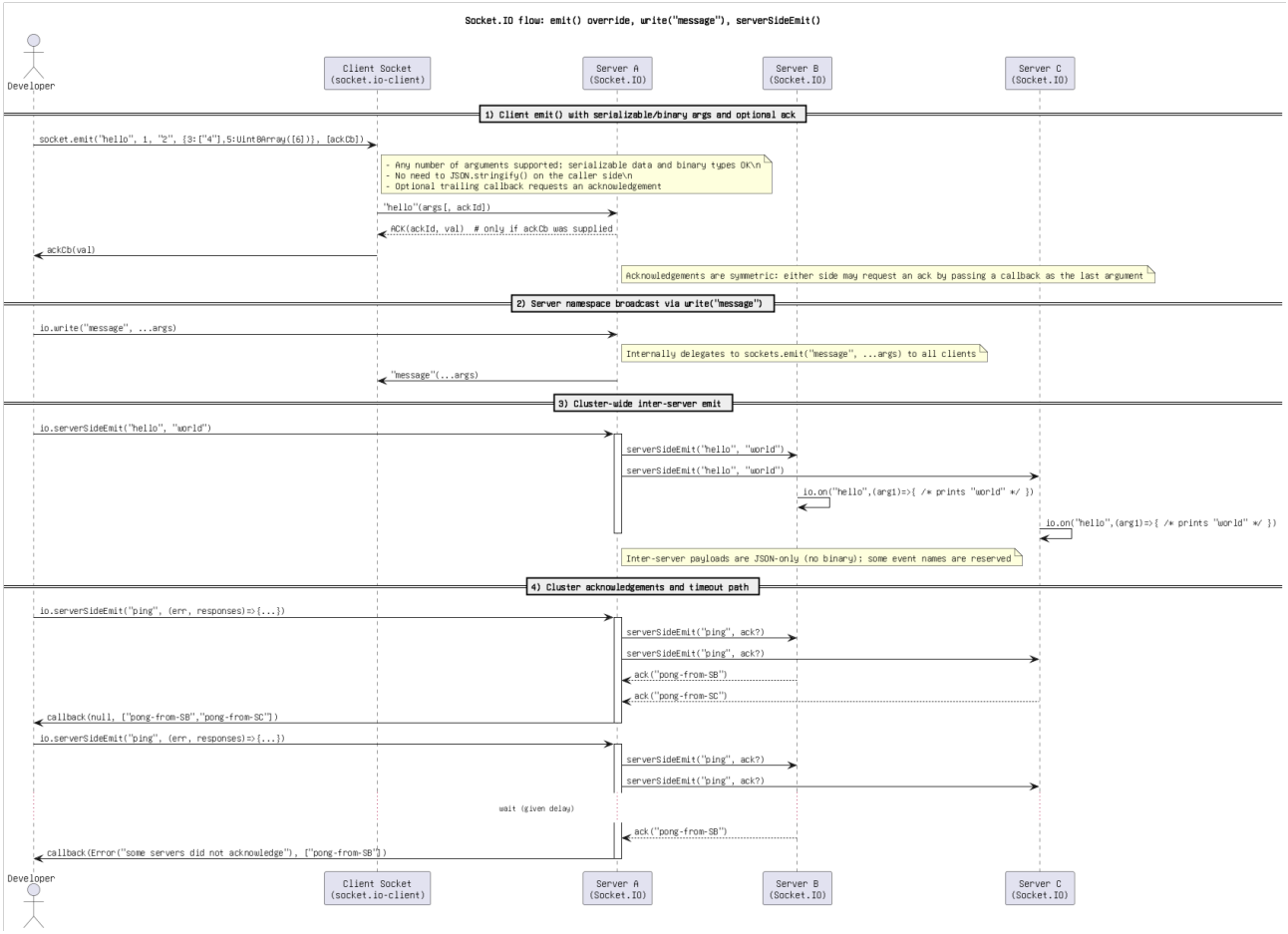

Figure: 18. Socket.IO flow_ emit() override, write(_message_), serverSideEmit().png

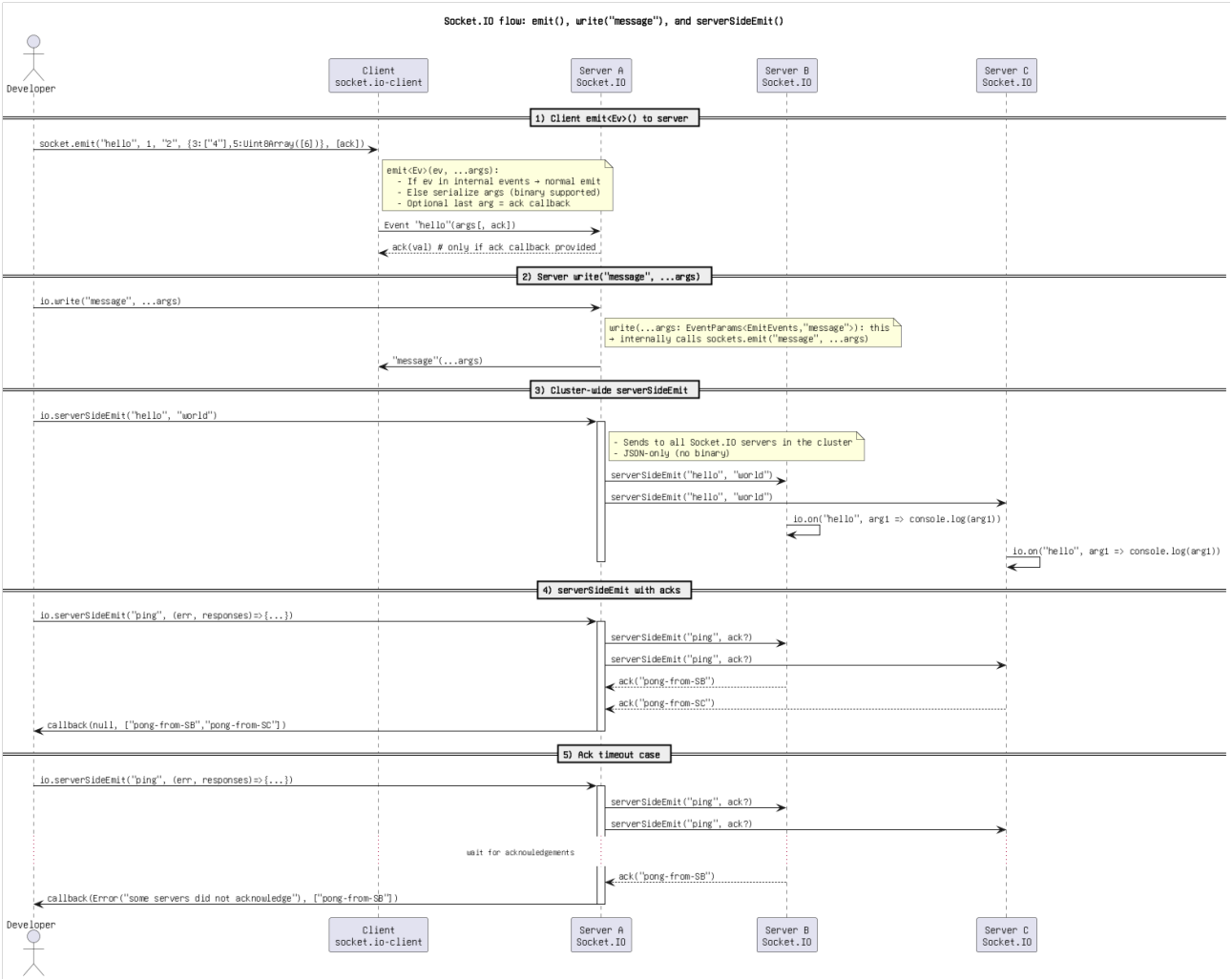

Figure: 19. Socket.IO flow_ emit(), write(_message_), and serverSideEmit().png

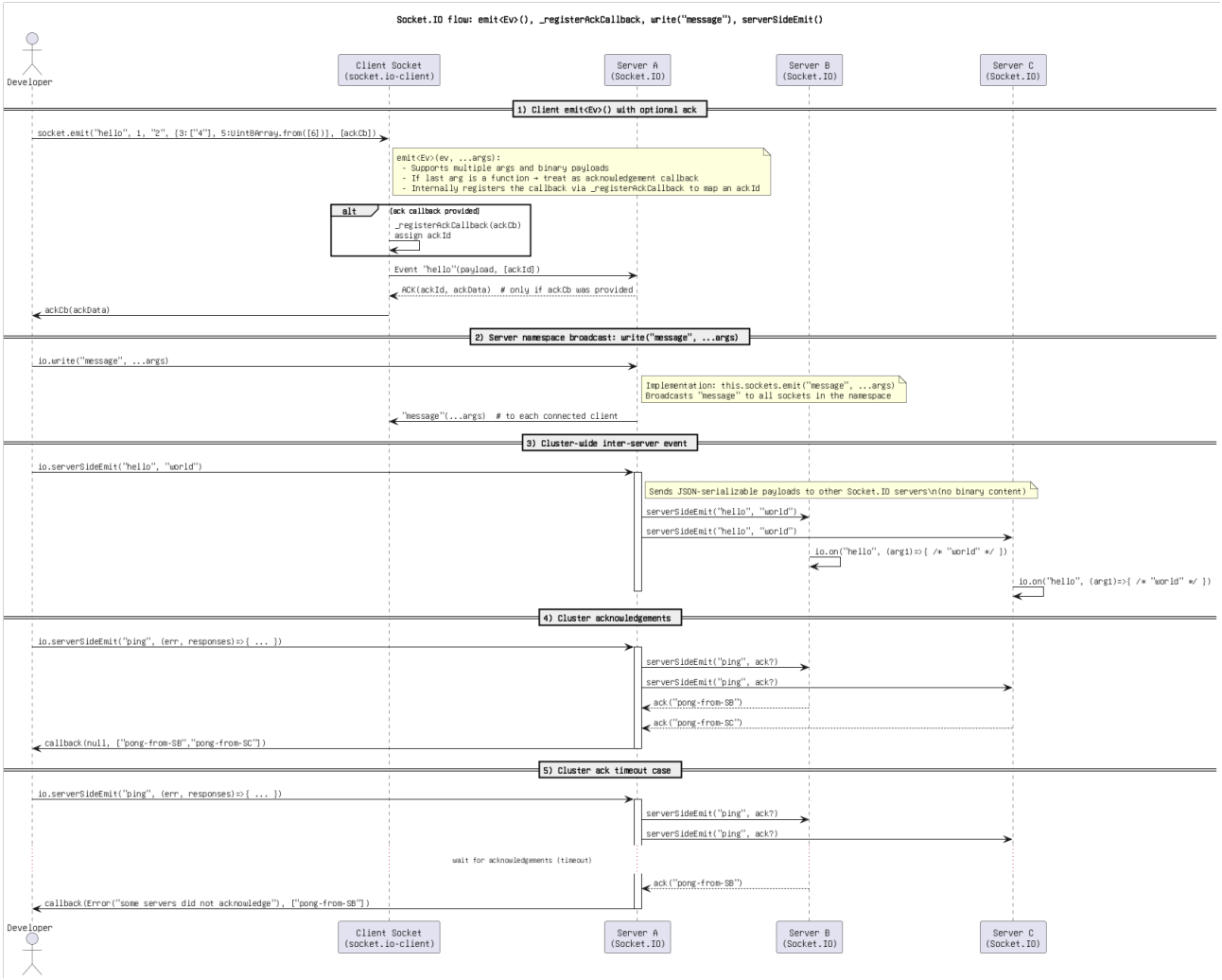

Figure: 20. Socket.IO flow_ emit_Ev_(), _registerAckCallback, write(_message_), serverSideEmit(.png

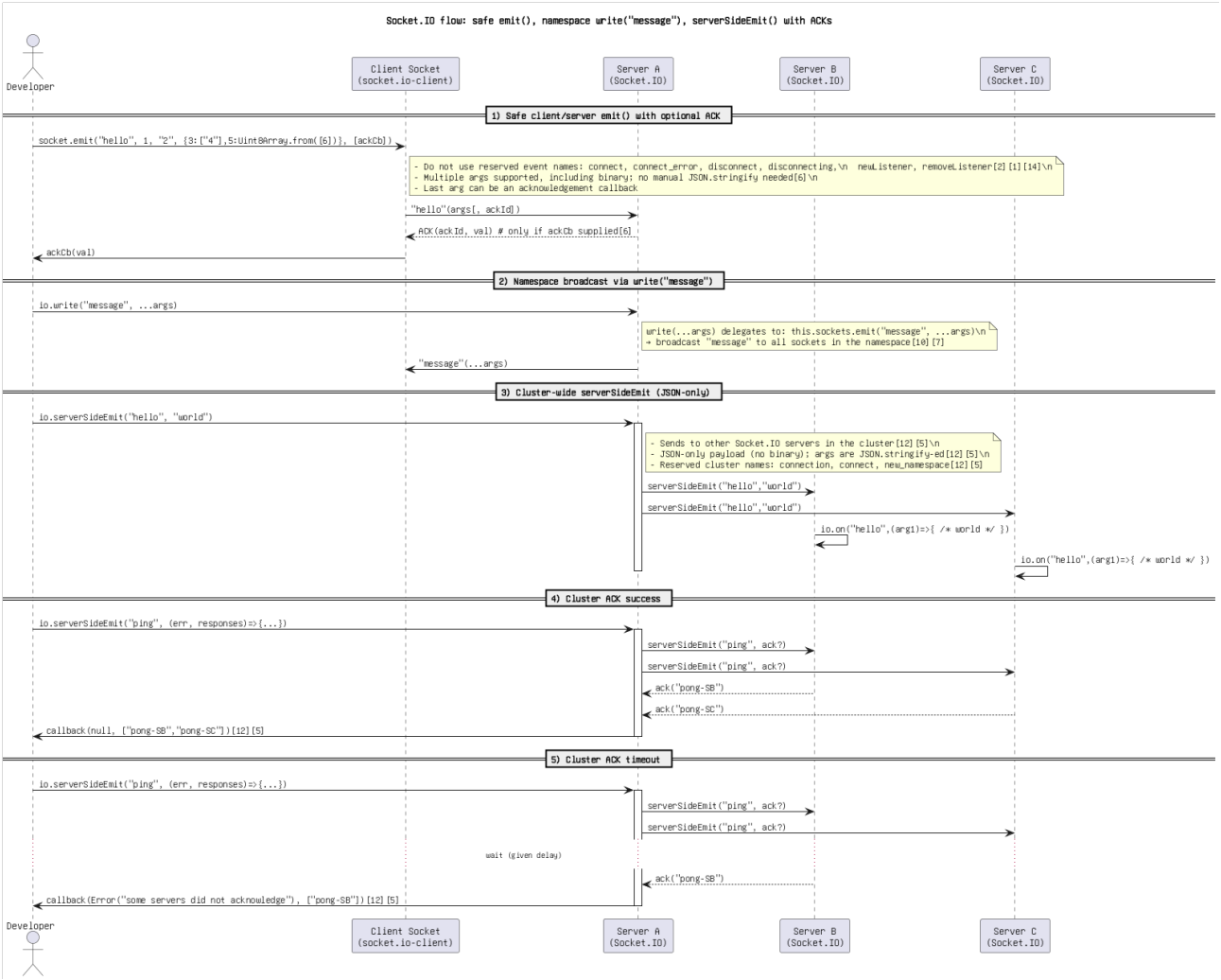

Figure: 21. Socket.IO flow_ safe emit(), namespace write(_message_), serverSideEmit() with ACKs(1).png

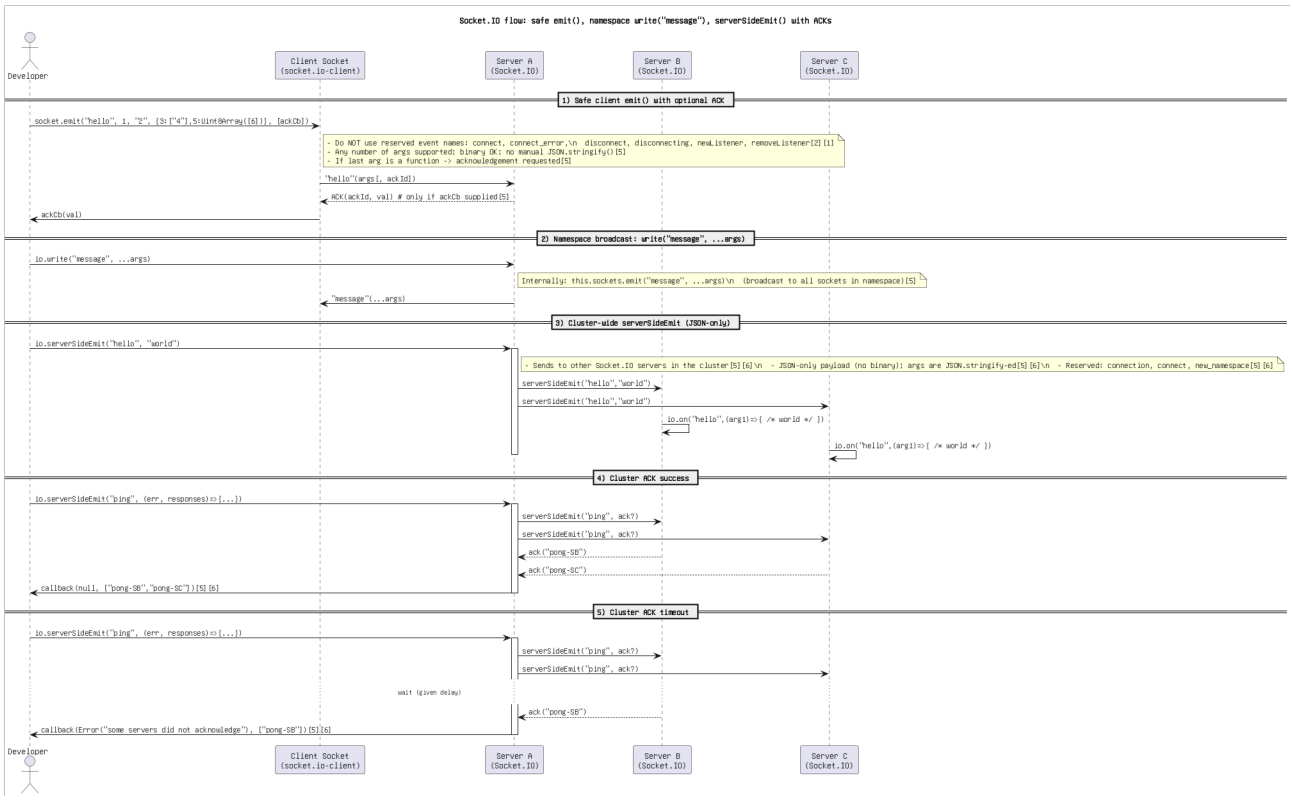

Figure: 22. Socket.IO flow_ safe emit(), namespace write(_message_), serverSideEmit() with ACKs.png

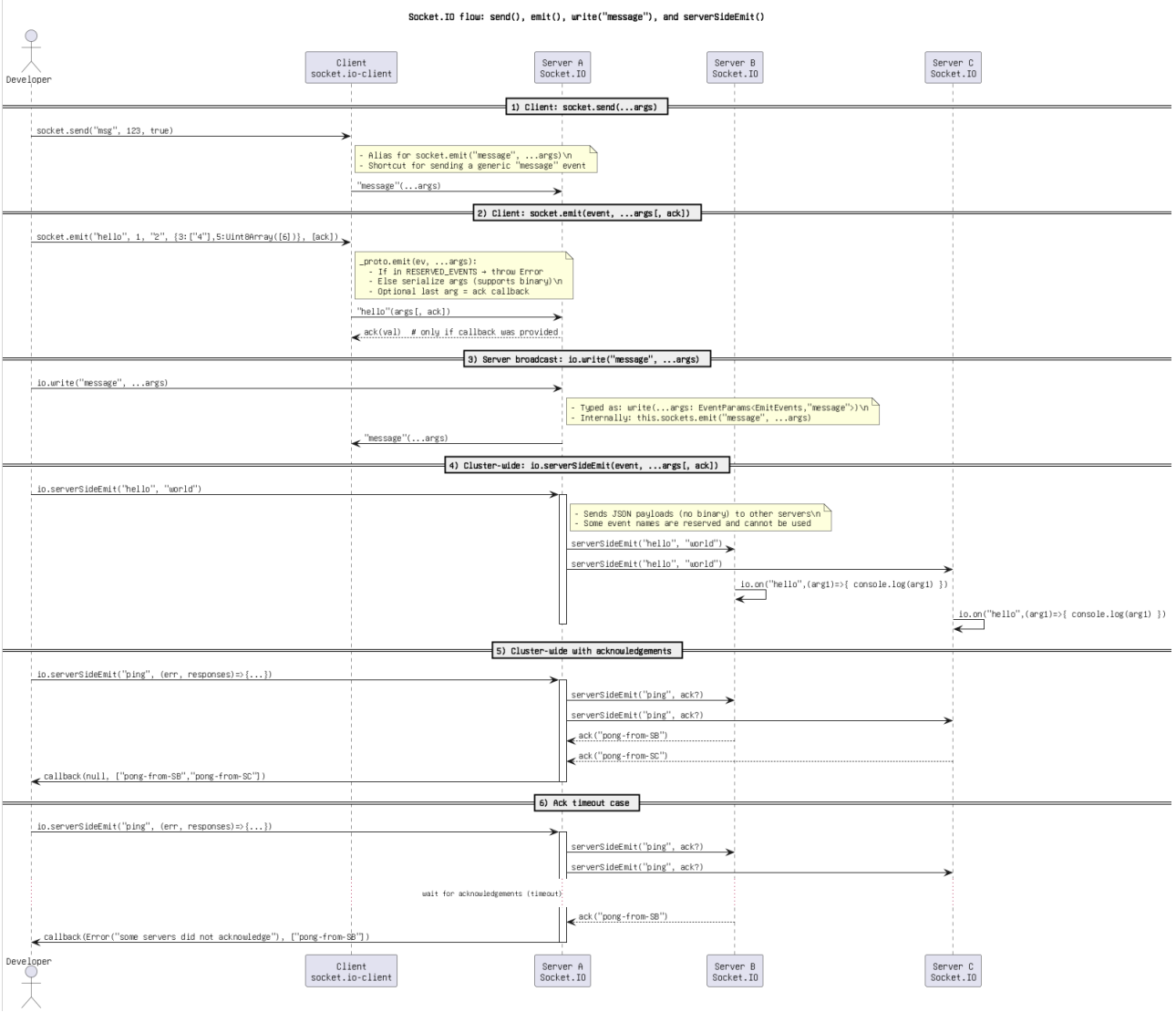

Figure: 23. Socket.IO flow_ send(), emit(), write(_message_), and serverSideEmit().png

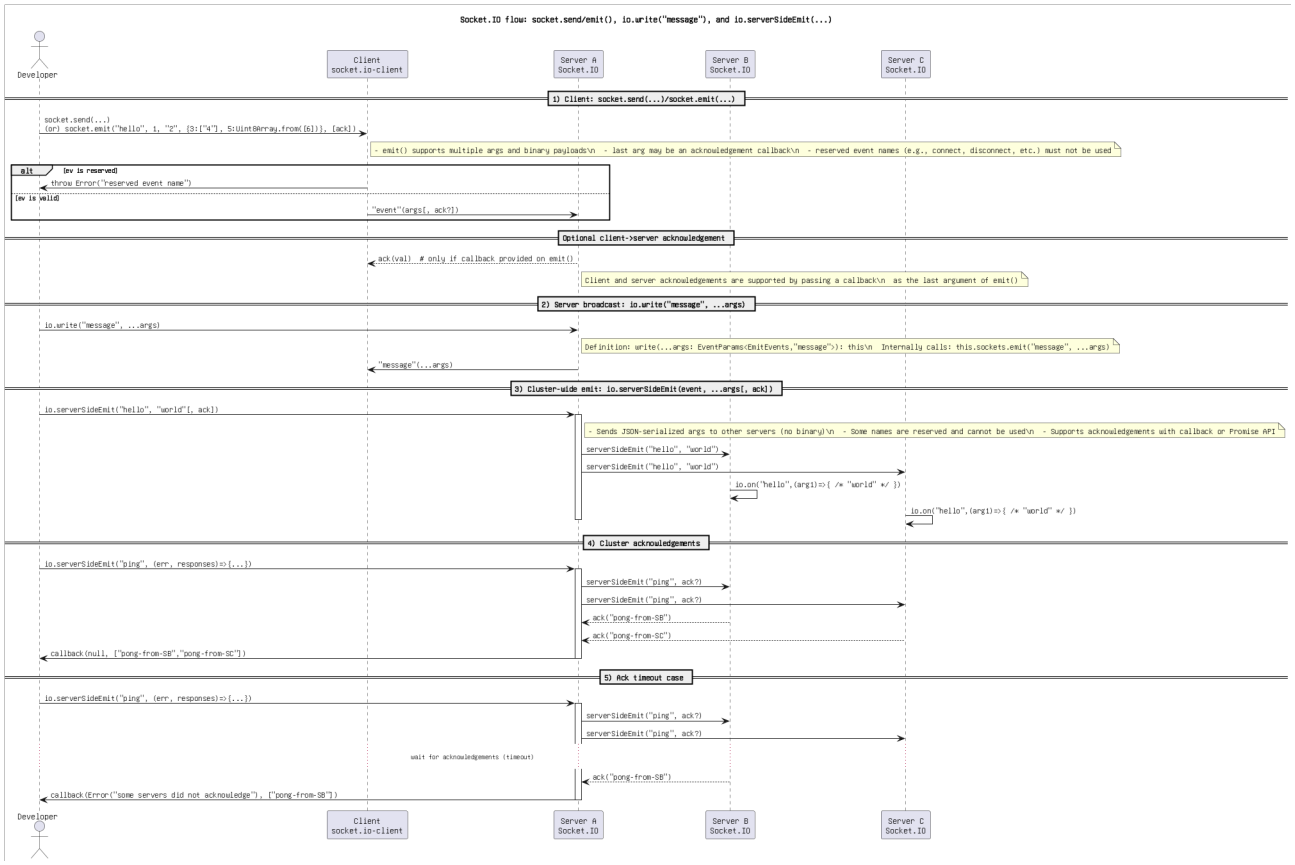

Figure: 24. Socket.IO flow_ socket.send_emit(), io.write(_message_), and io.serverSideEmit(...).png

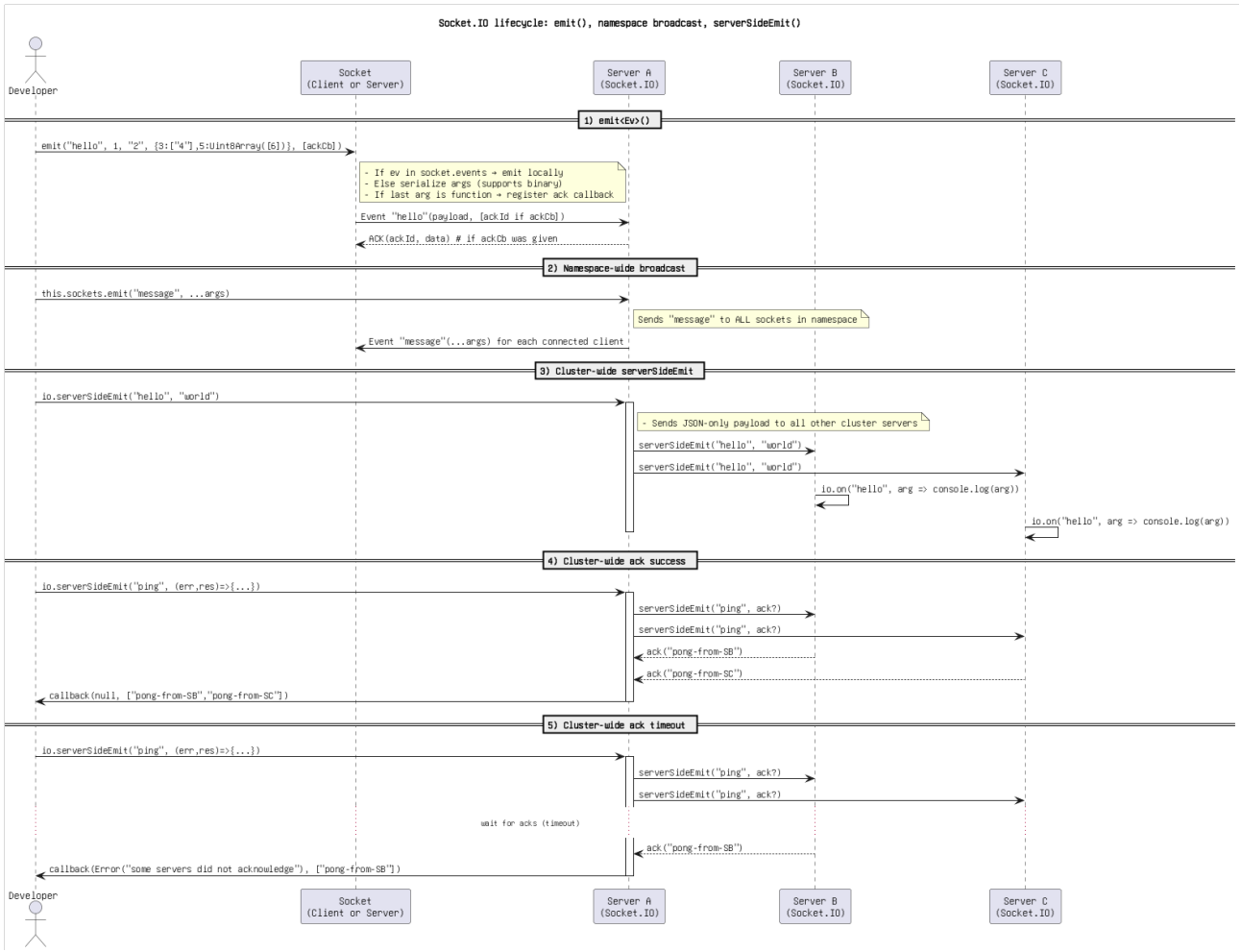

Figure: 25. Socket.IO lifecycle_ emit(), namespace broadcast, serverSideEmit().png

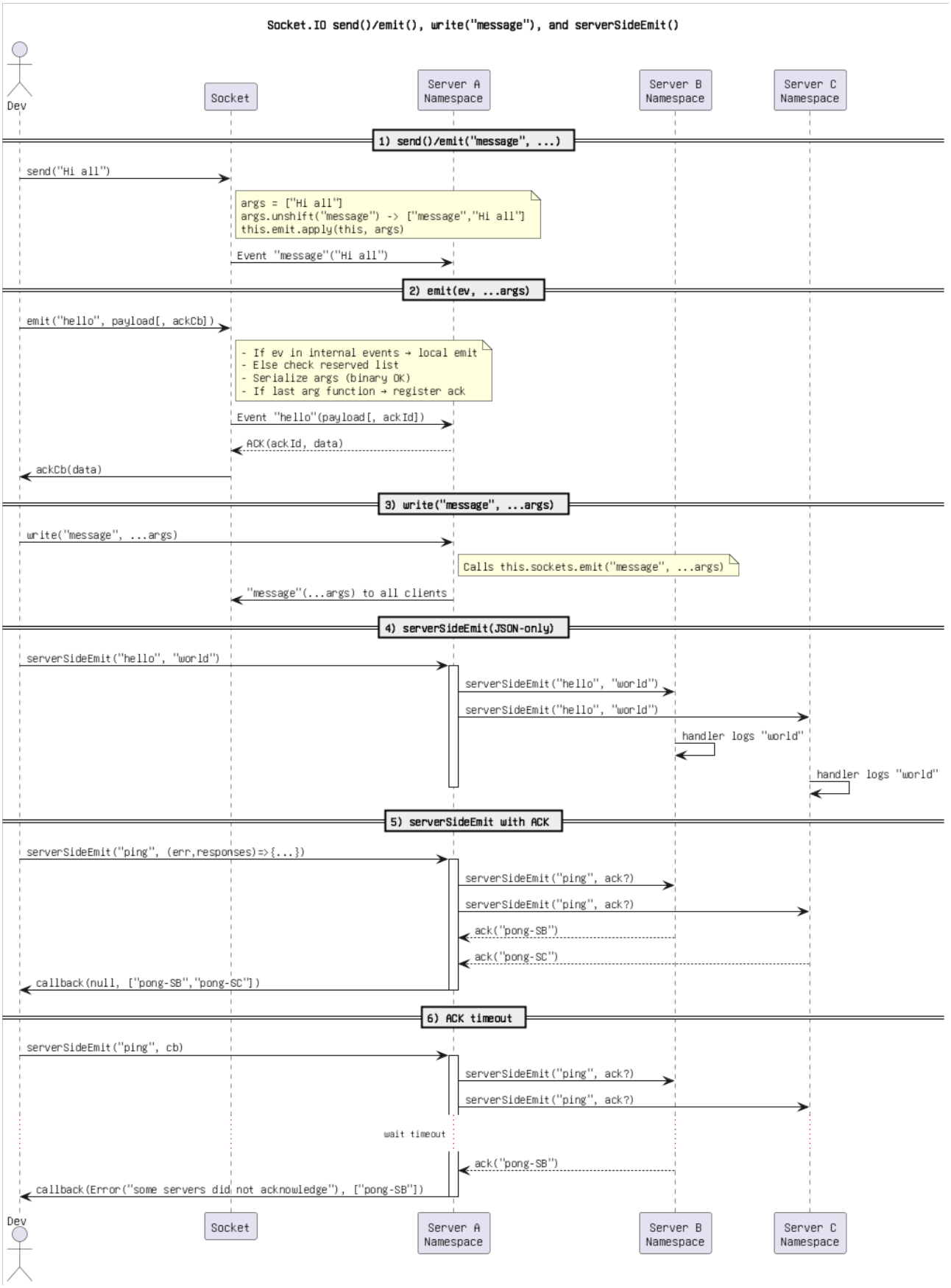

Figure: 26. Socket.IO send()_emit(), write(_message_), and serverSideEmit().png

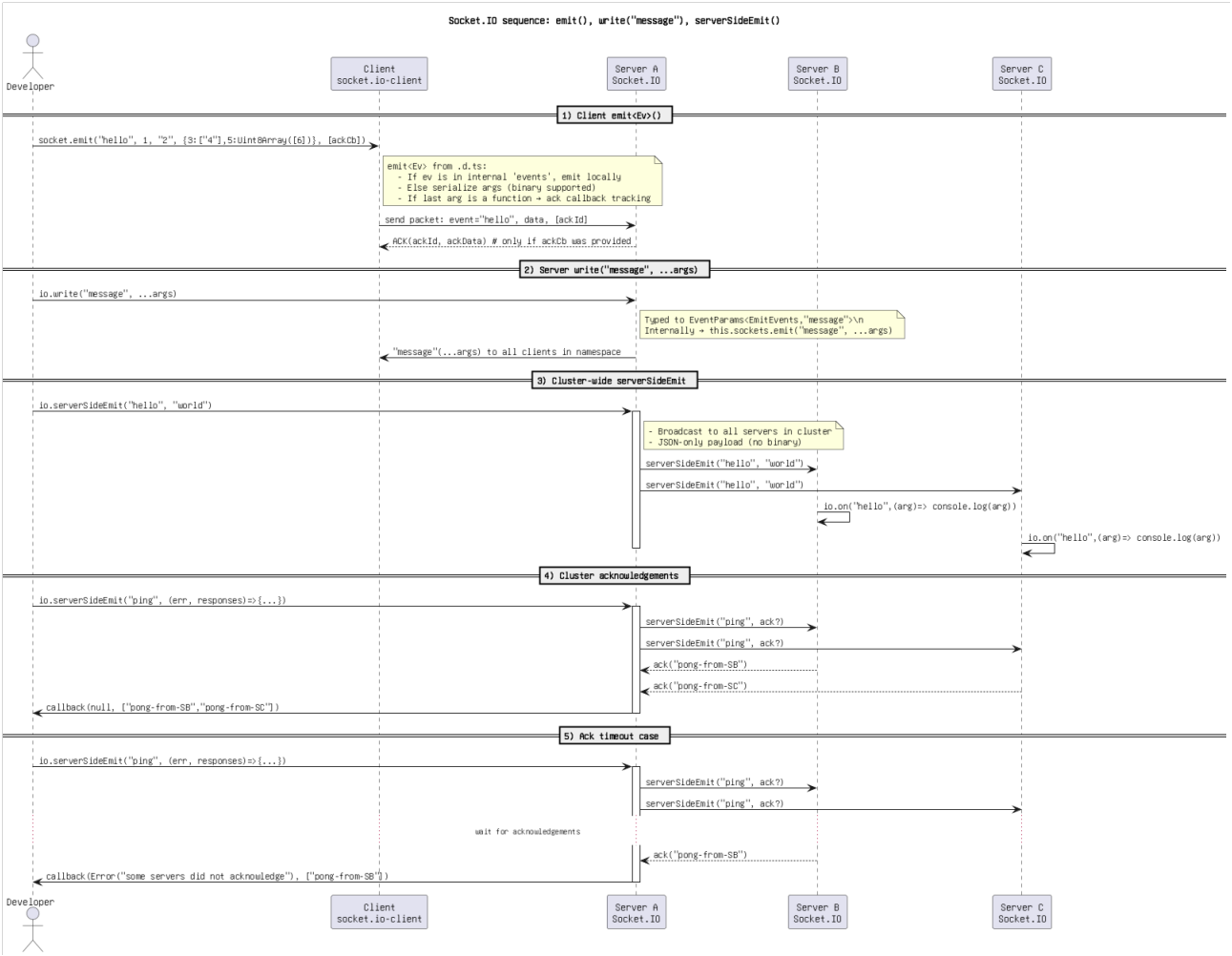

Figure: 27. Socket.IO sequence_ emit(), write(_message_), serverSideEmit().png

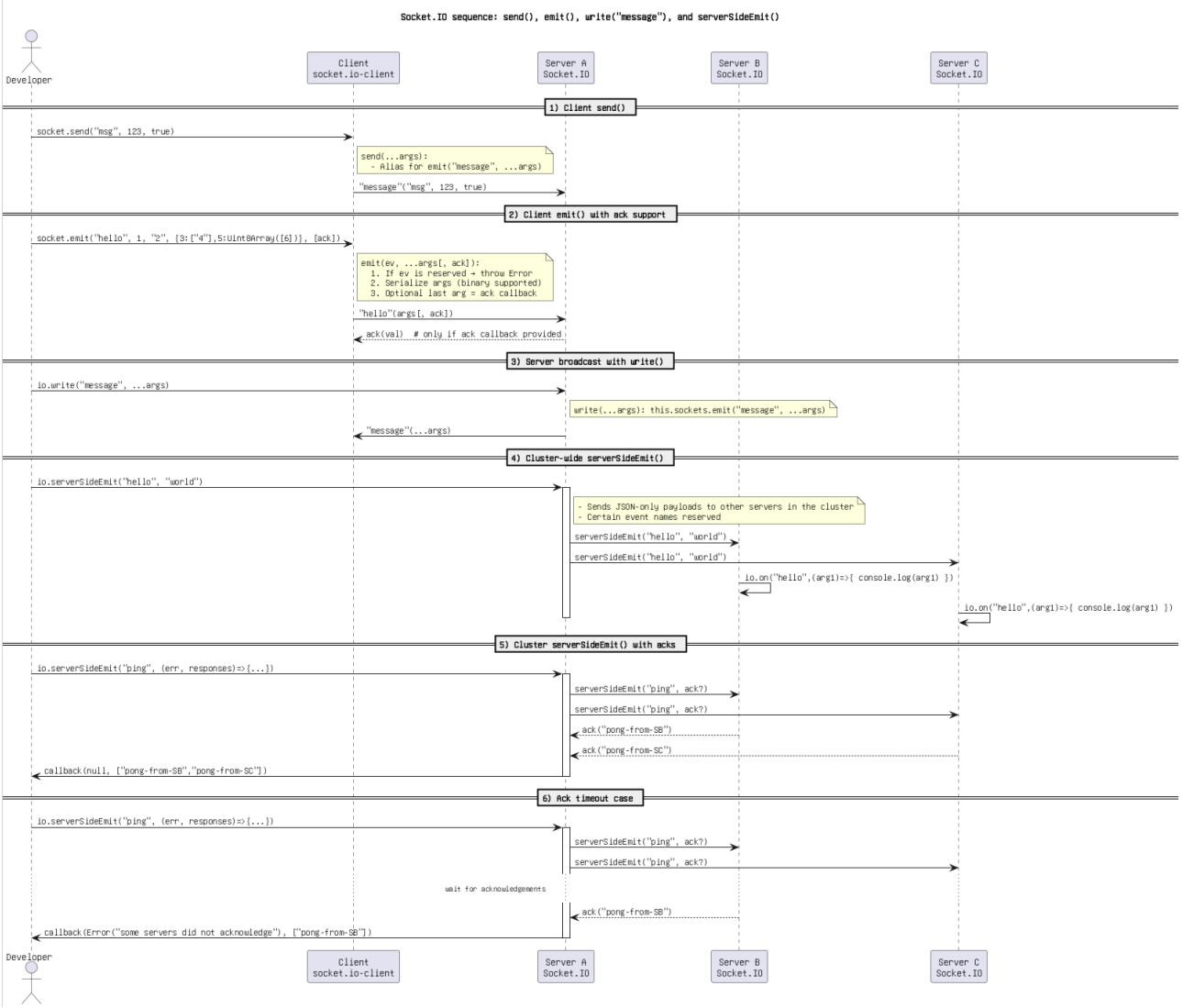

Figure: 28. Socket.IO sequence_ send(), emit(), write(_message_), and serverSideEmit().png

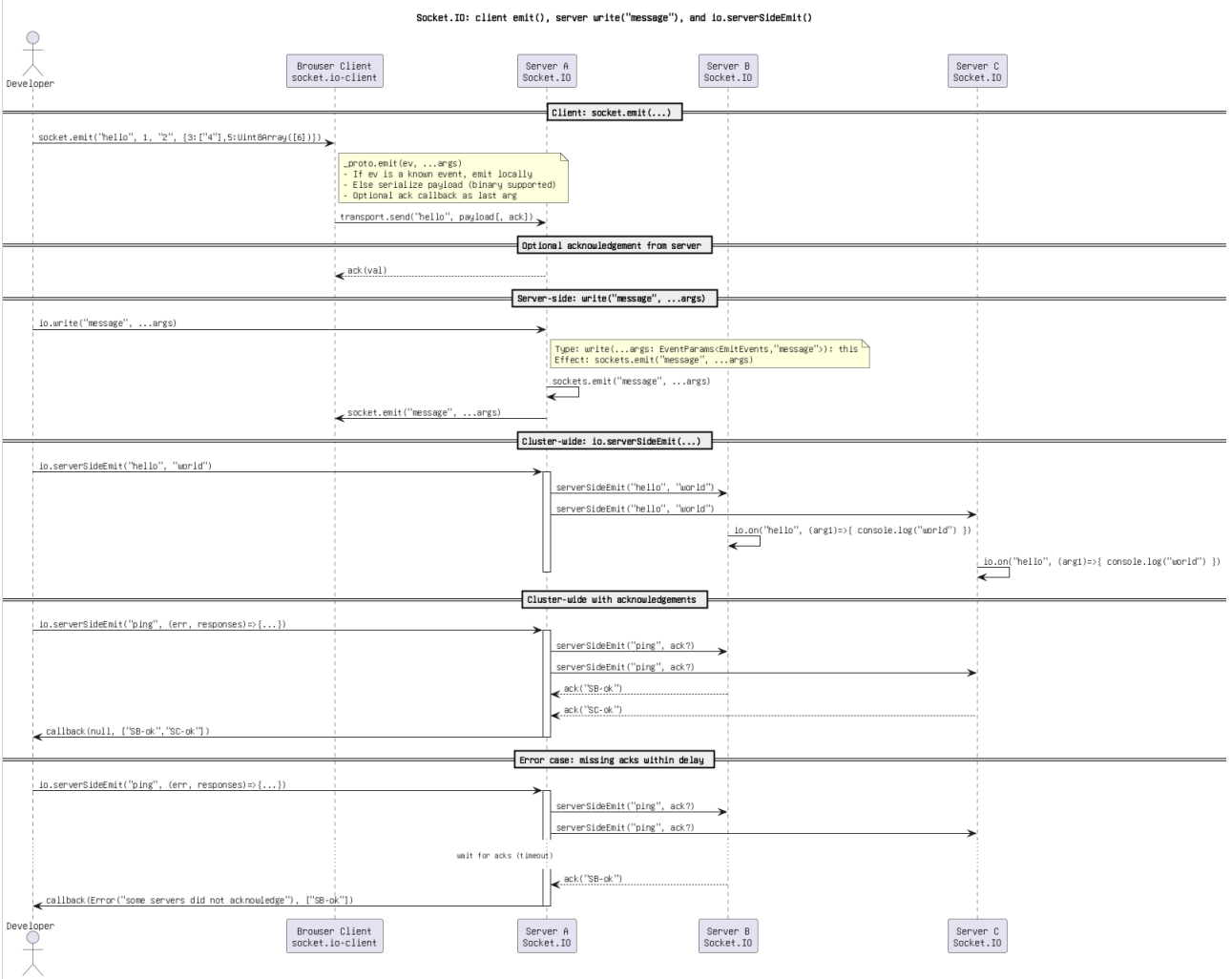

Figure: 29. Socket.IO_ client emit(), server write(_message_), and io.serverSideEmit().png

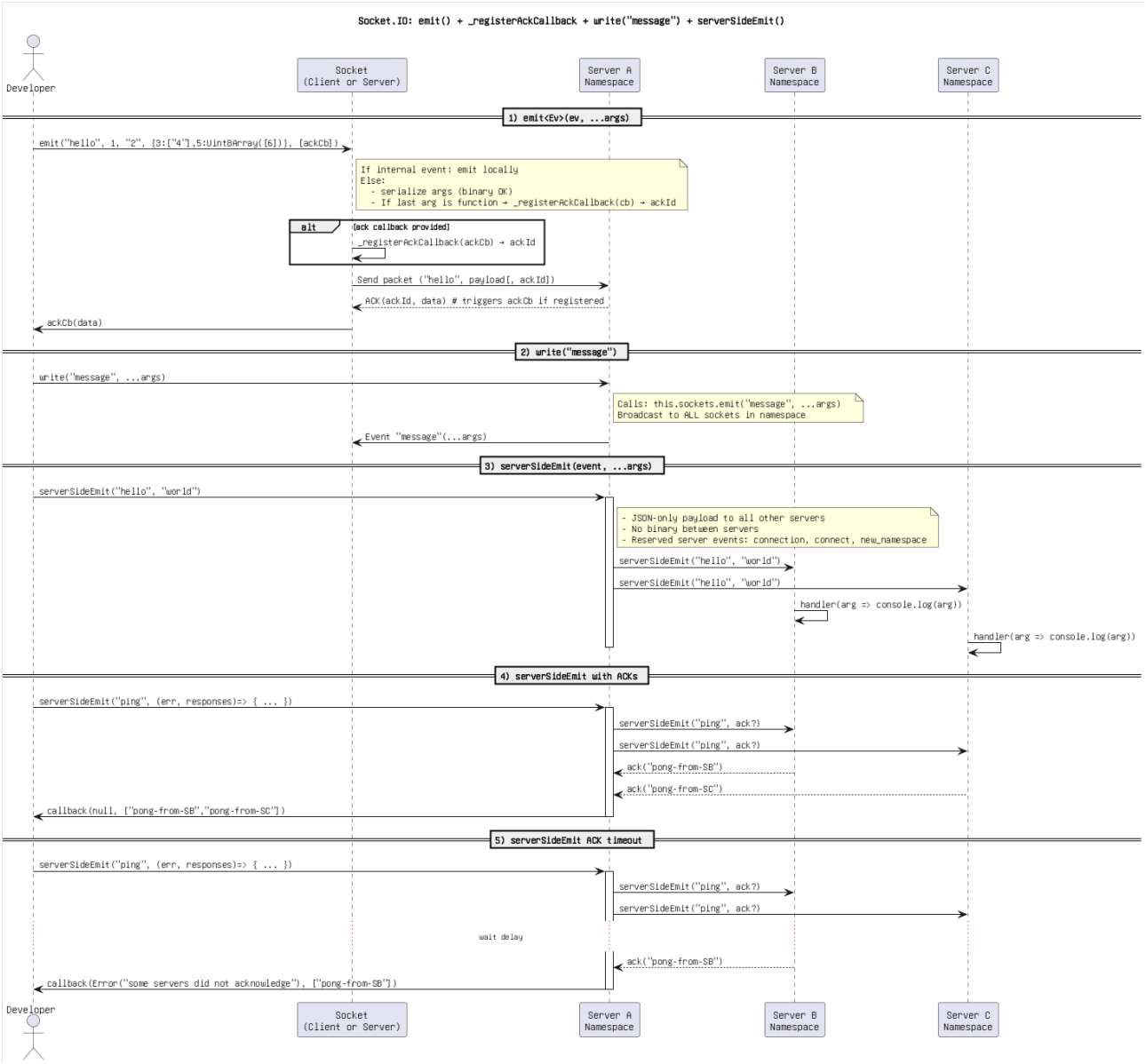

Figure: 30. Socket.IO_ emit() + _registerAckCallback + write(_message_) + serverSideEmit().png

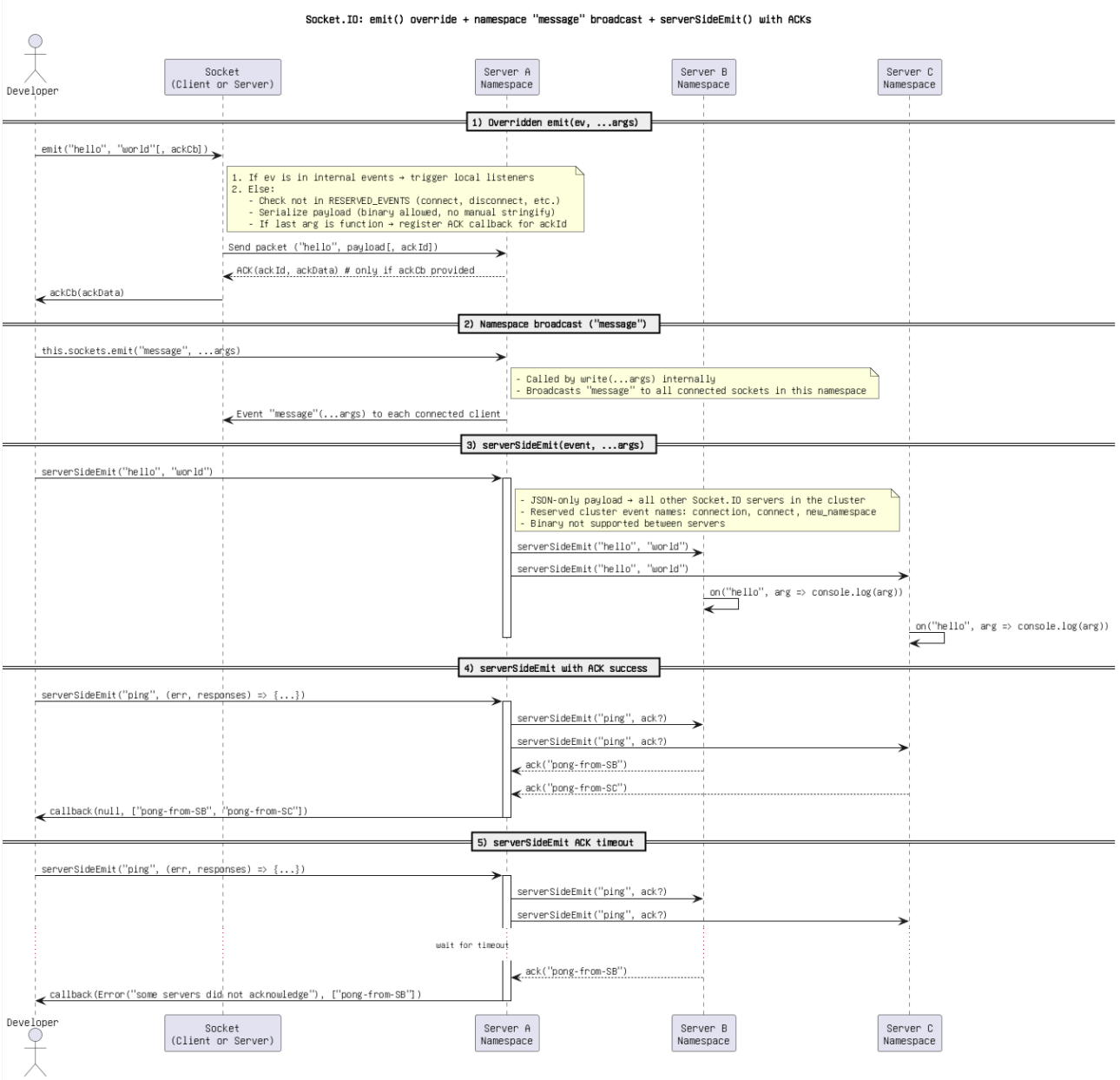

Figure: 31. Socket.IO_ emit() override + namespace _message_ broadcast + serverSideEmit() with ACKs.png

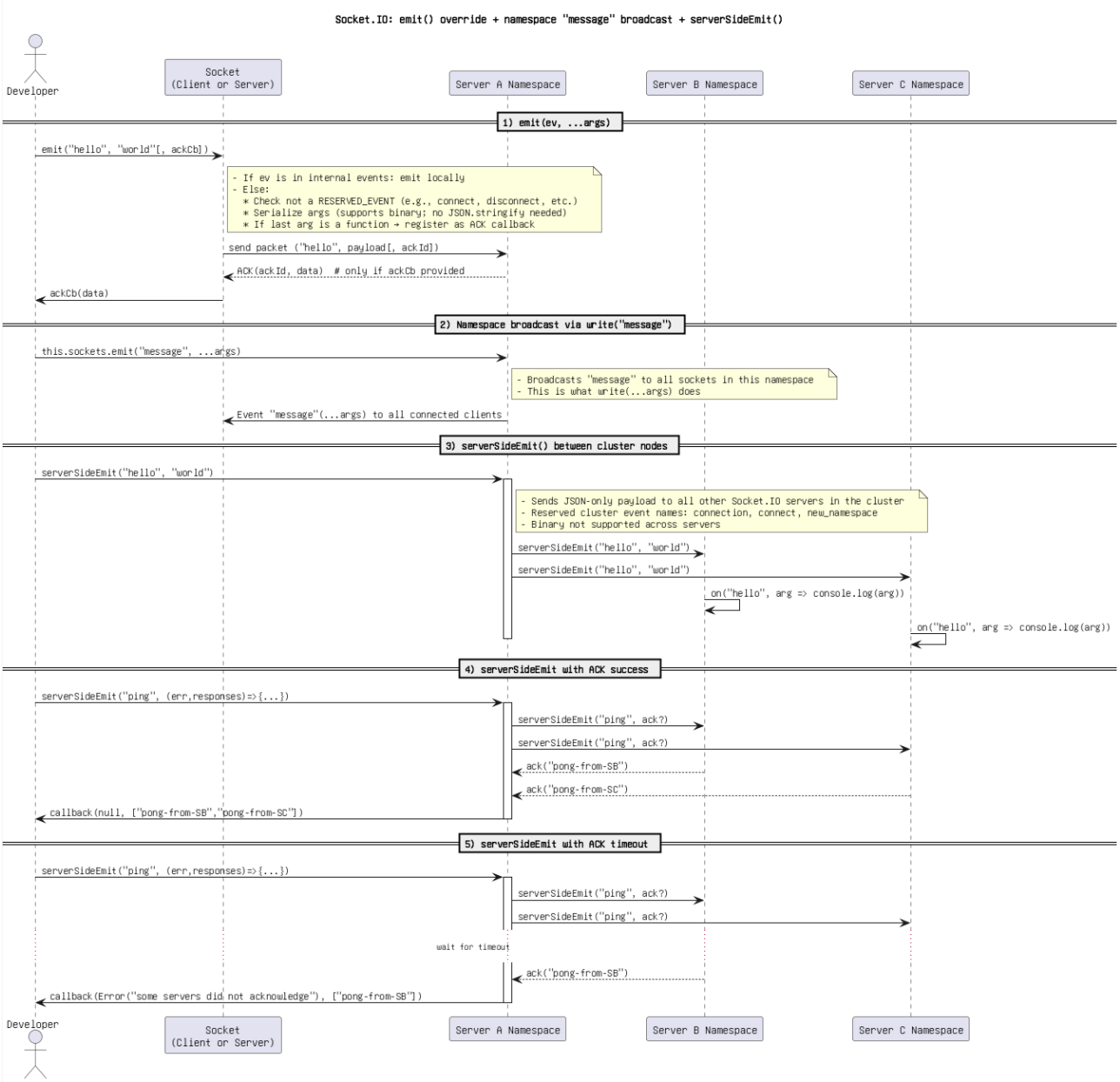

Figure: 32. Socket.IO_ emit() override + namespace _message_ broadcast + serverSideEmit().png

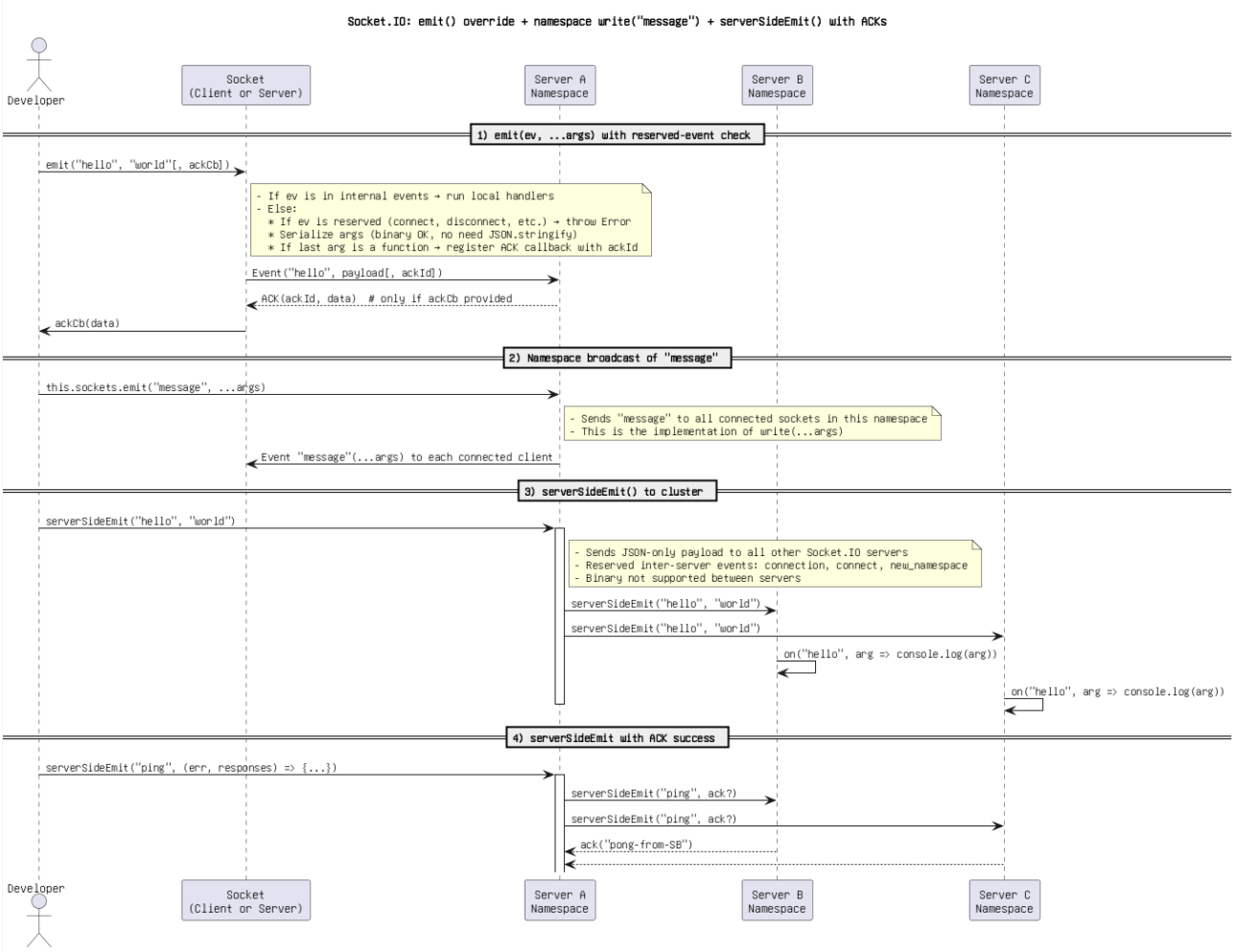

Figure: 33. Socket.IO_ emit() override + namespace write(_message_) + serverSideEmit() with ACKs.png

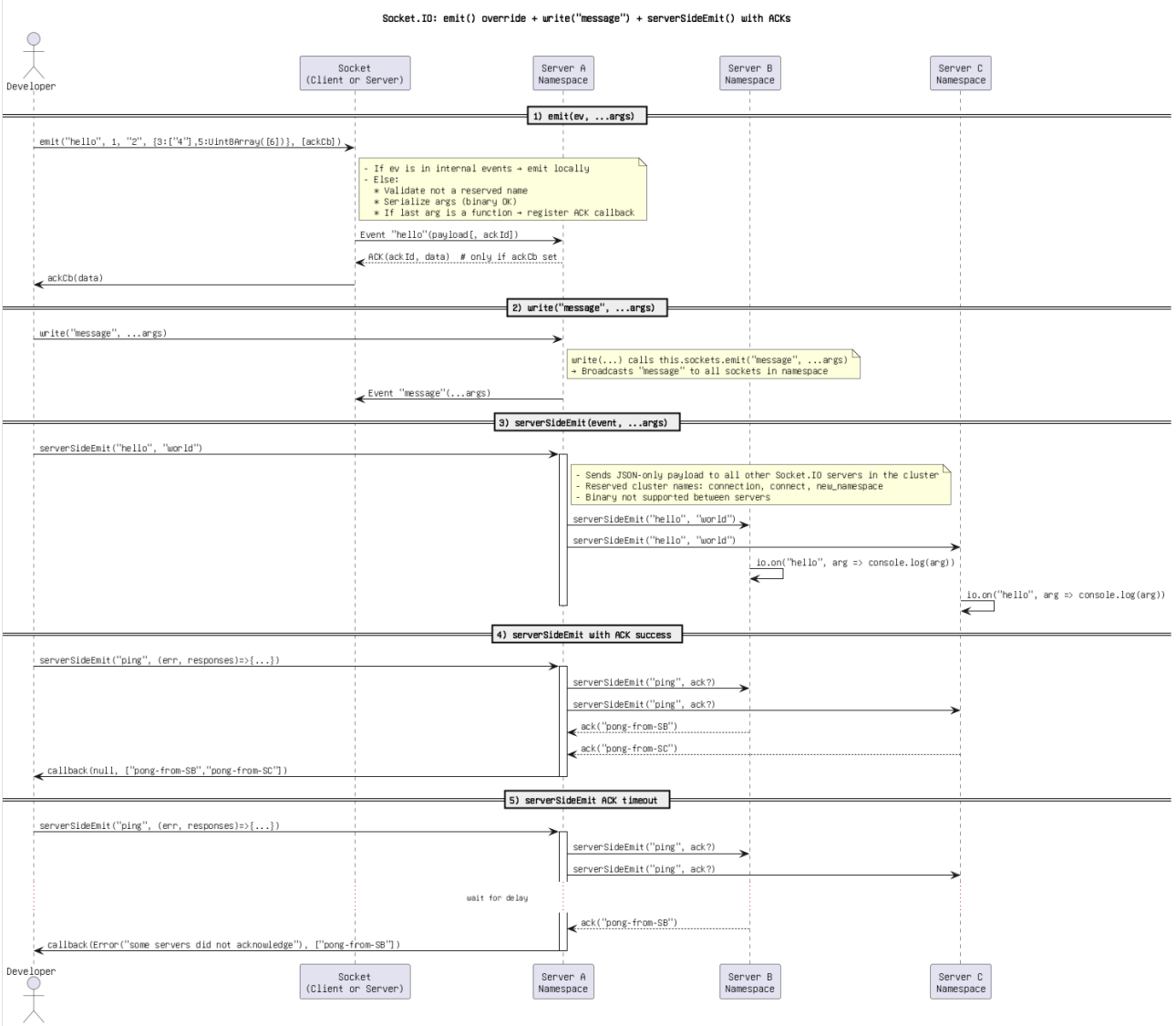

Figure: 34. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs(1).png

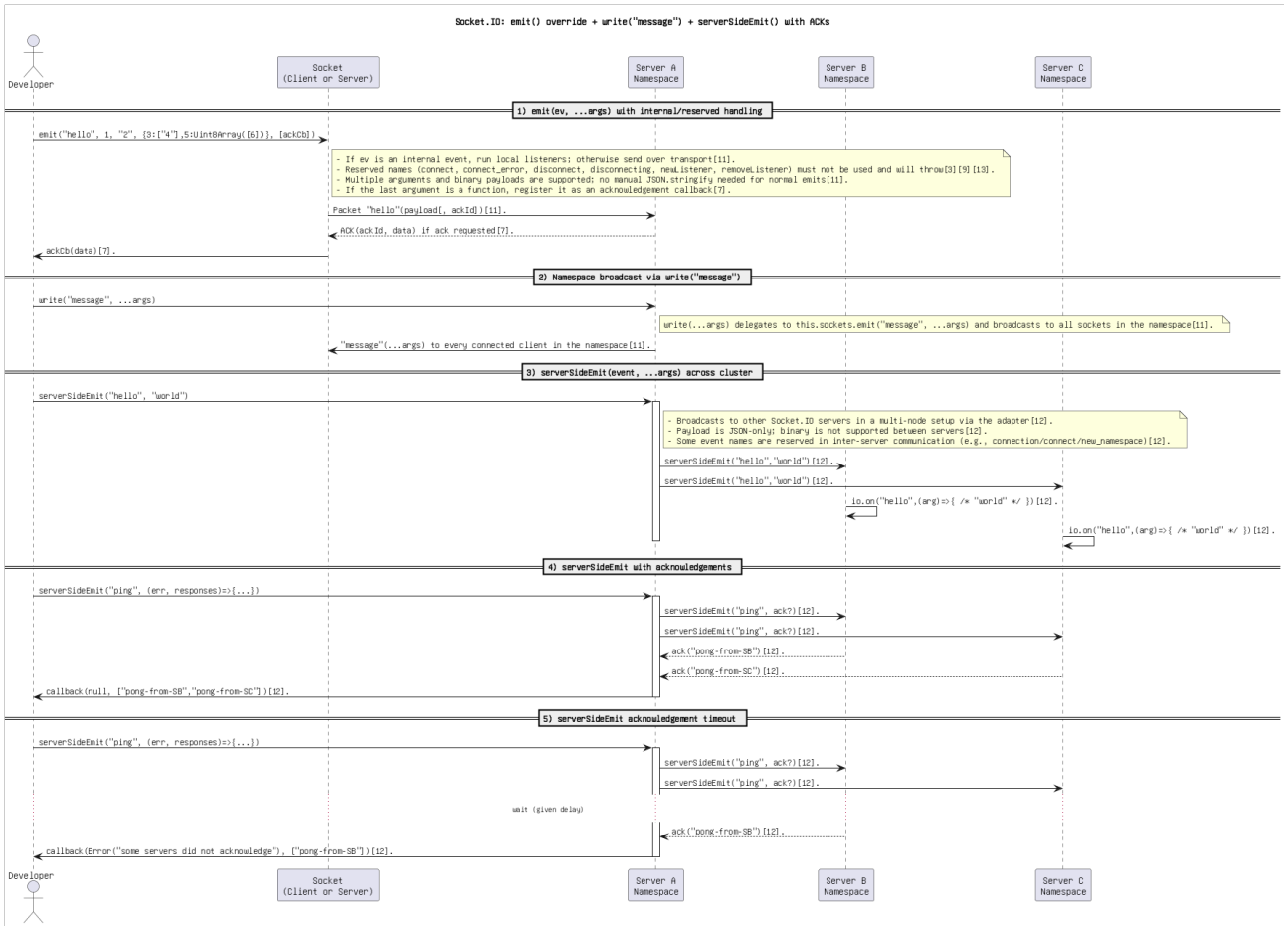

Figure: 35. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs(2).png

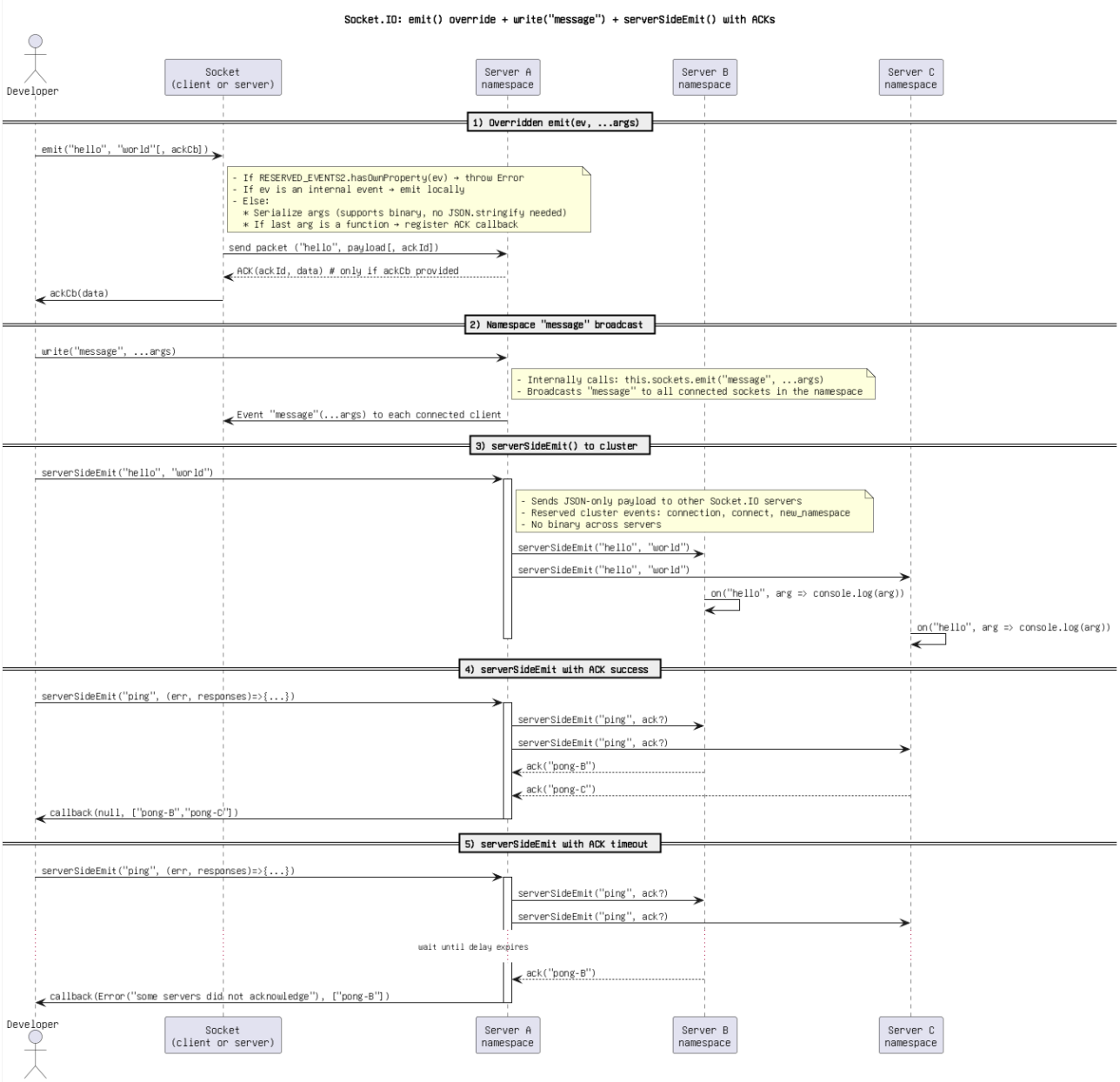

Figure: 36. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs(3).png

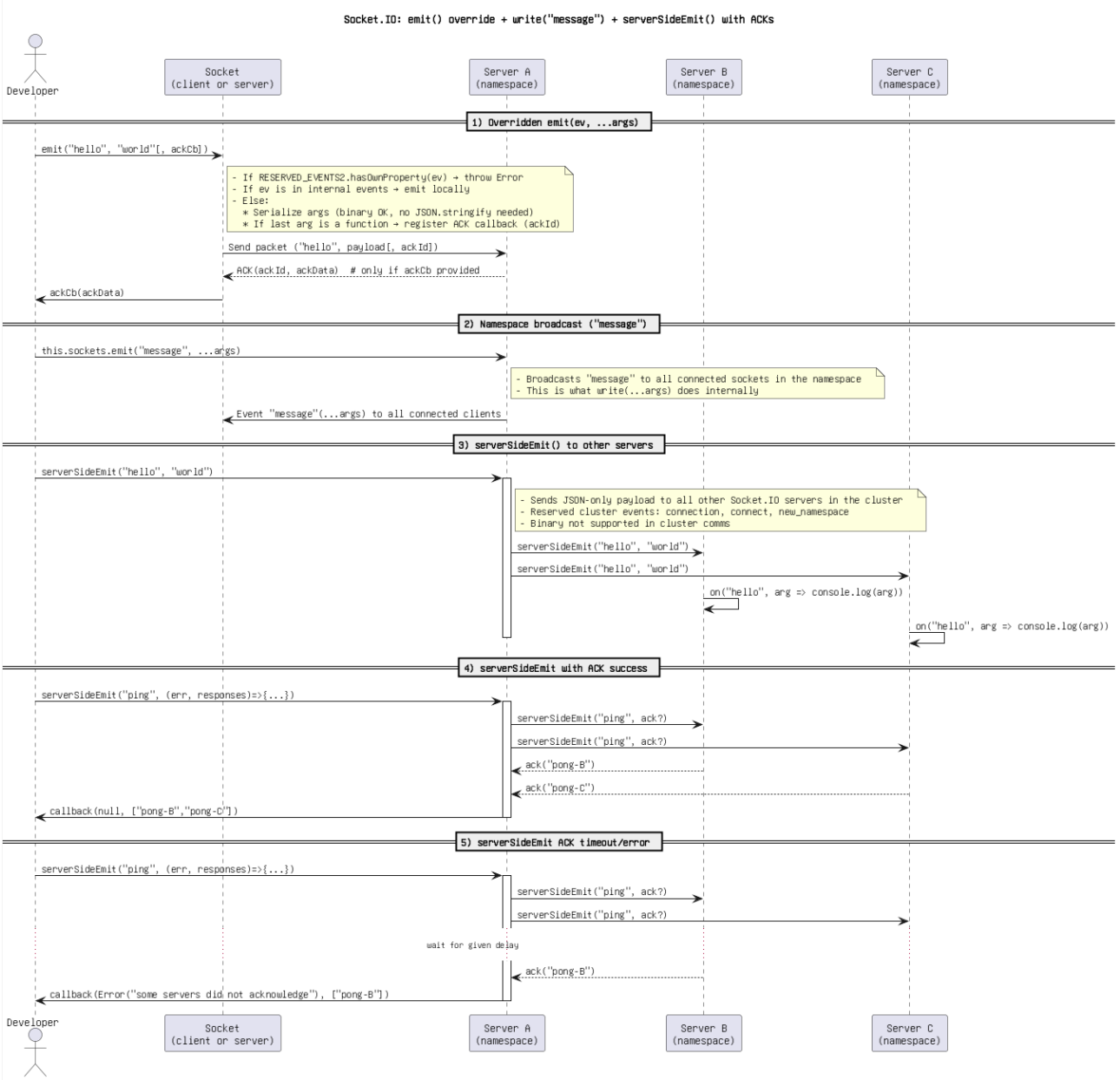

Figure: 37. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs(4) (Copy).png

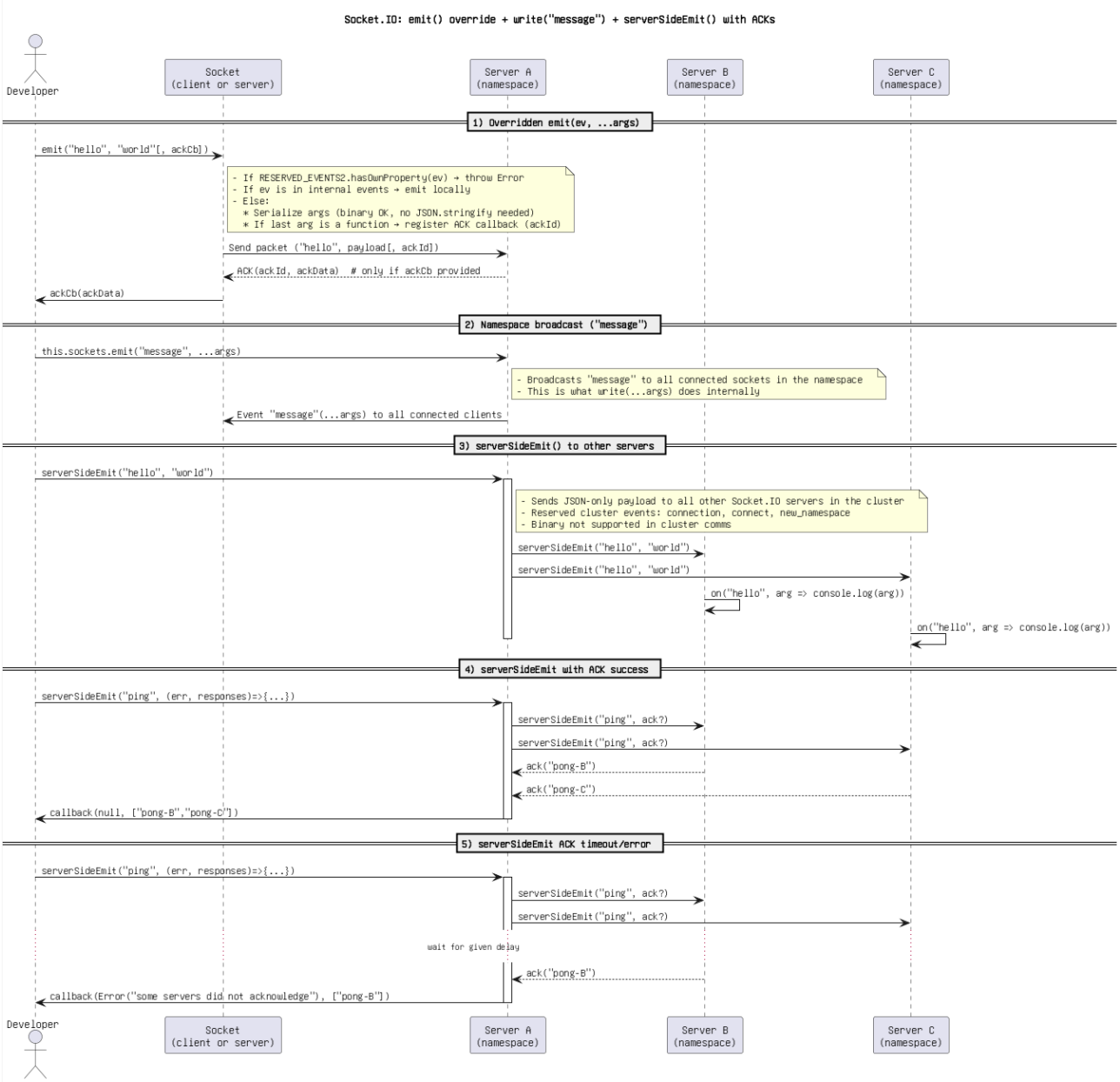

Figure: 38. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs(4).png

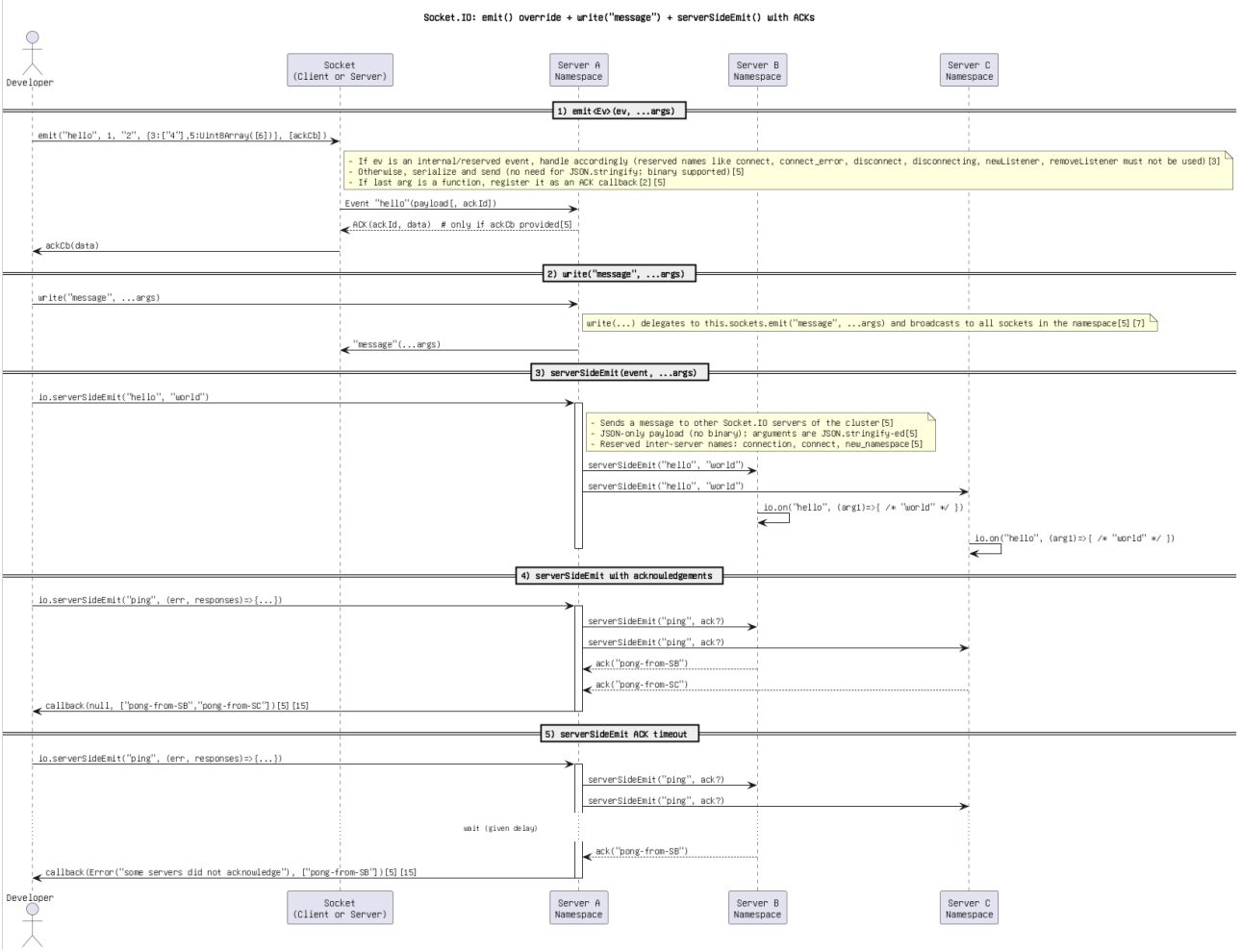

Figure: 39. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs..png

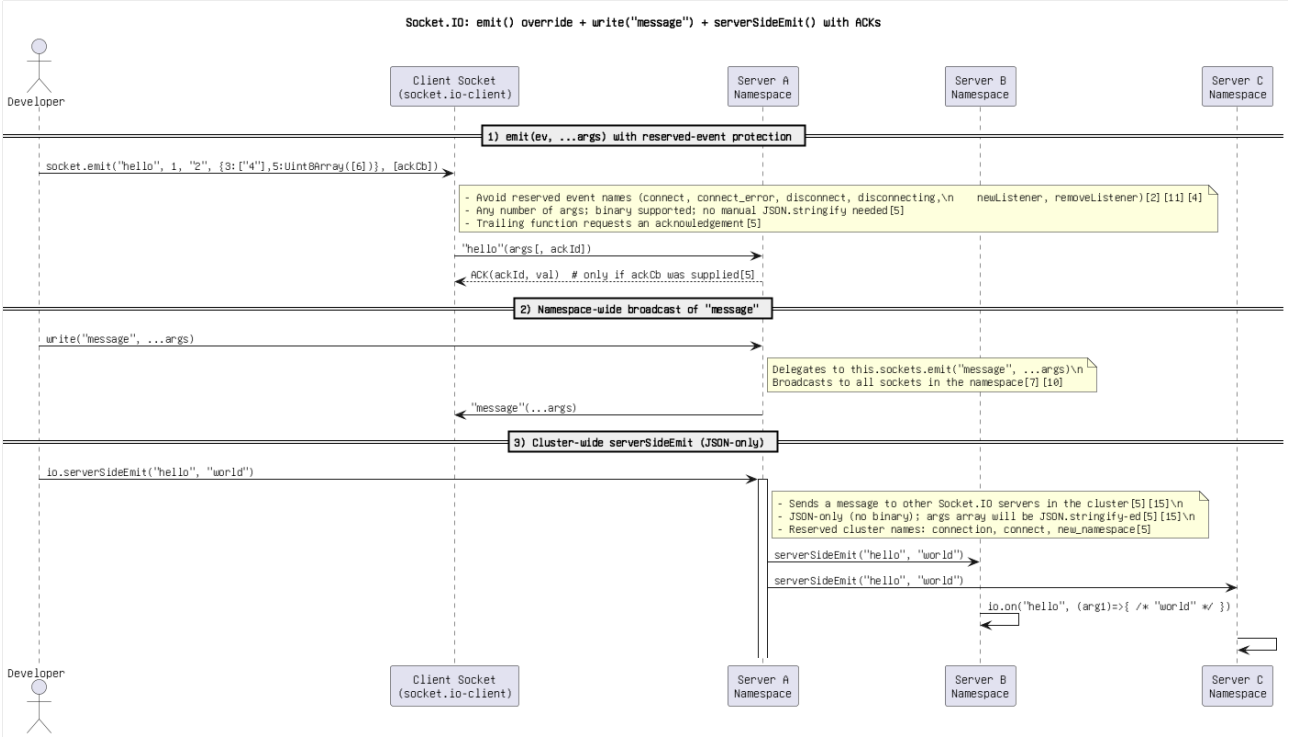

Figure: 40. Socket.IO_ emit() override + write(_message_) + serverSideEmit() with ACKs.png

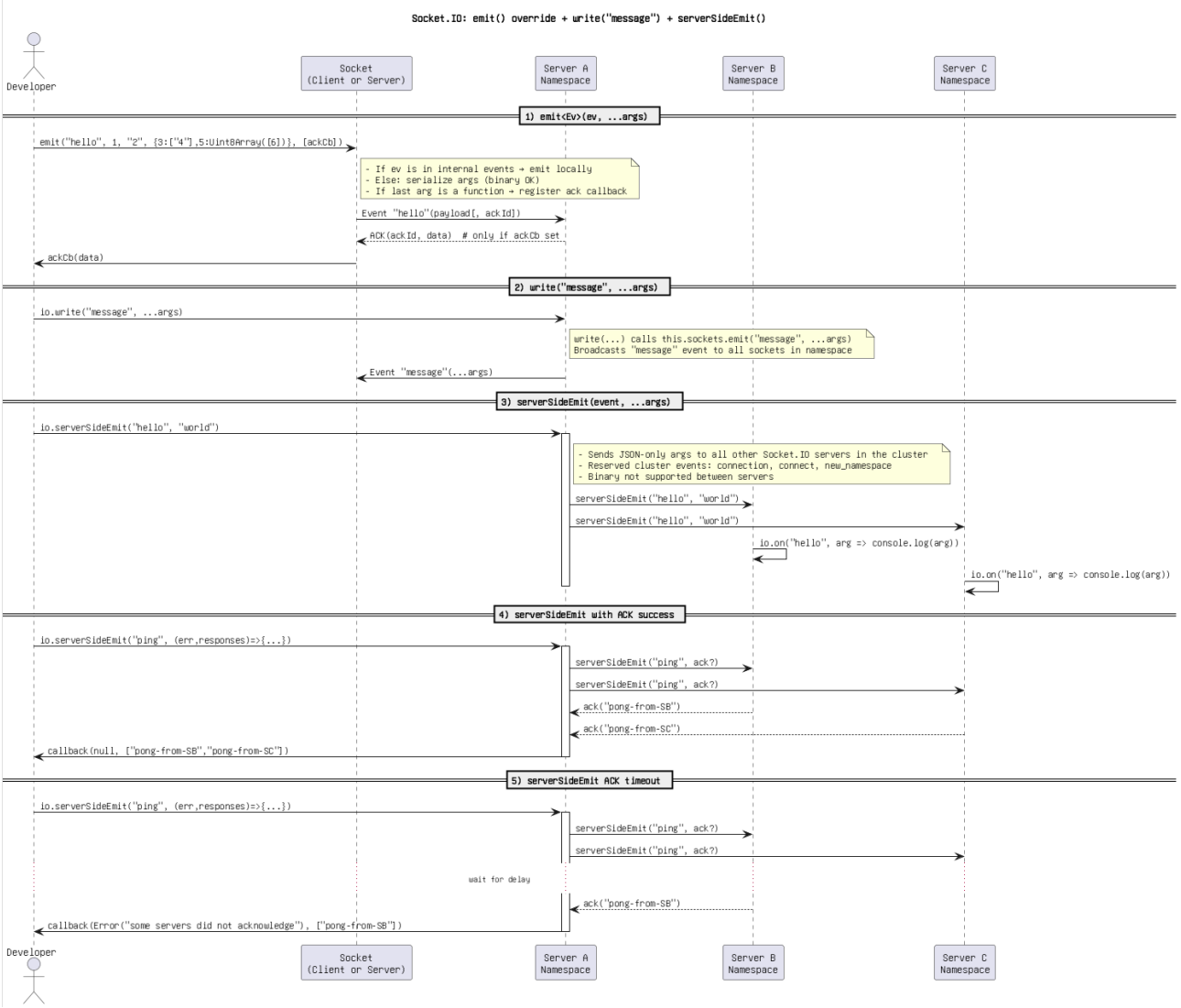

Figure: 41. Socket.IO_ emit() override + write(_message_) + serverSideEmit()(1).png

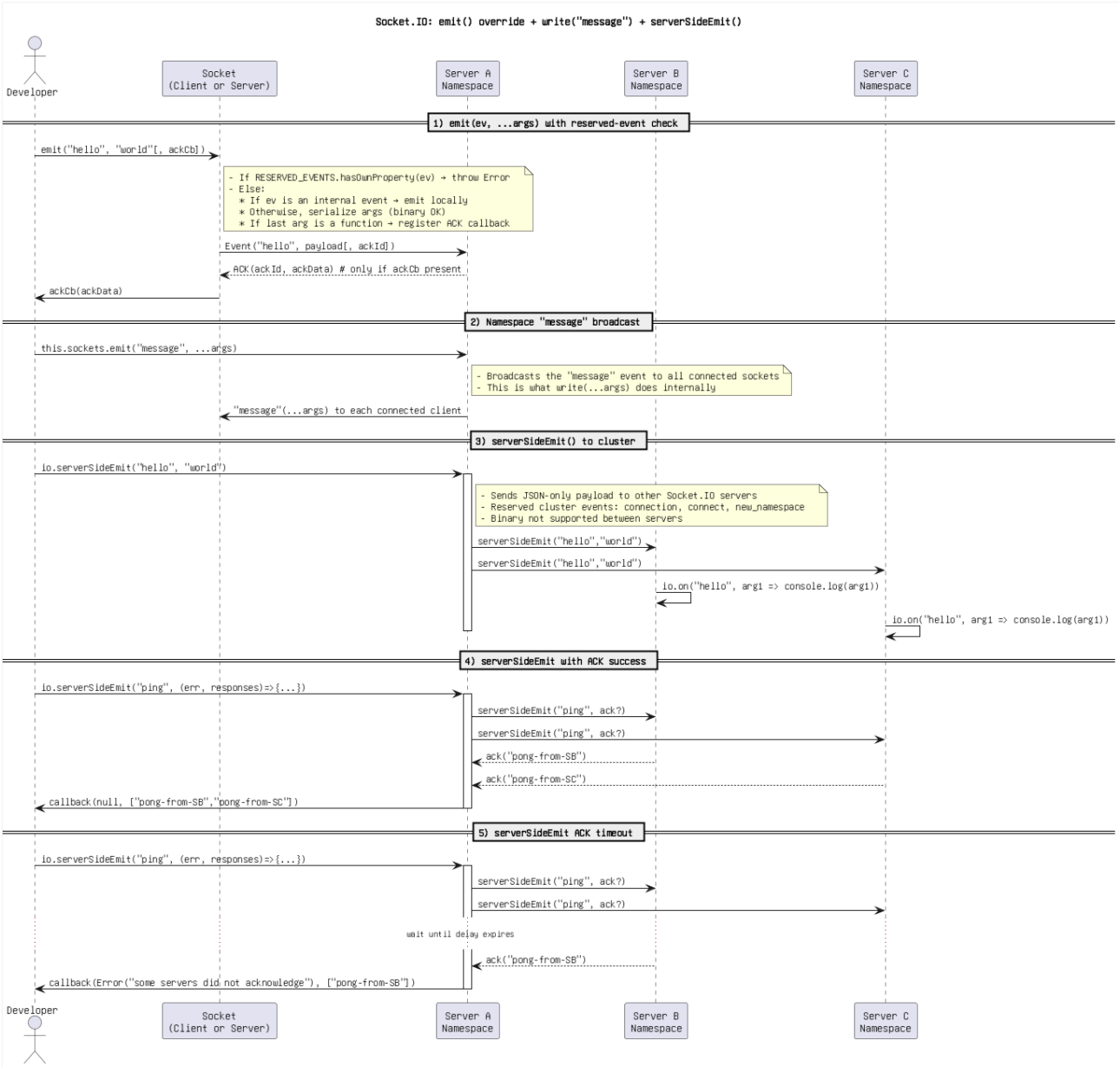

Figure: 42. Socket.IO_ emit() override + write(_message_) + serverSideEmit()(2).png

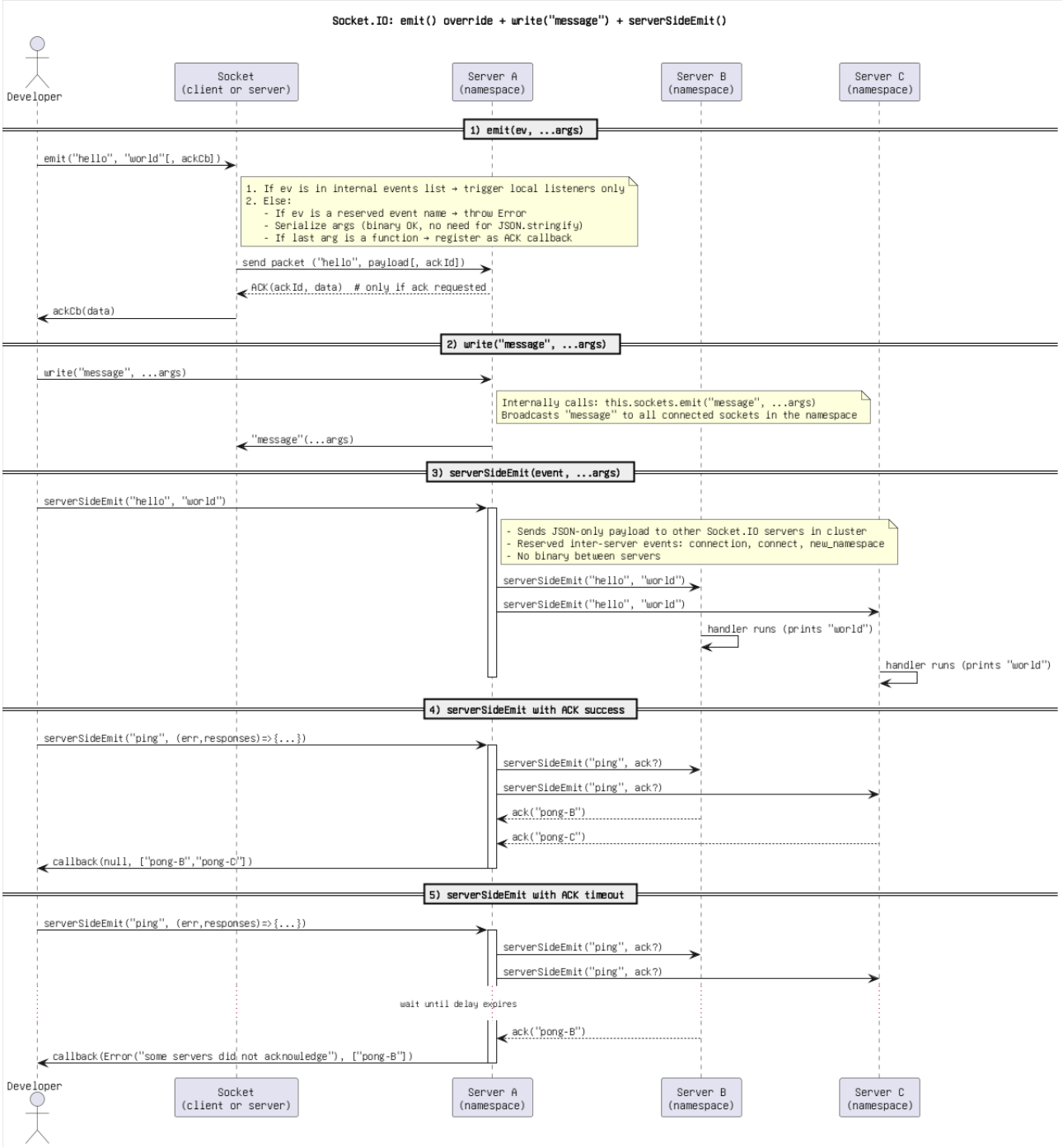

Figure: 43. Socket.IO_ emit() override + write(_message_) + serverSideEmit()(3).png

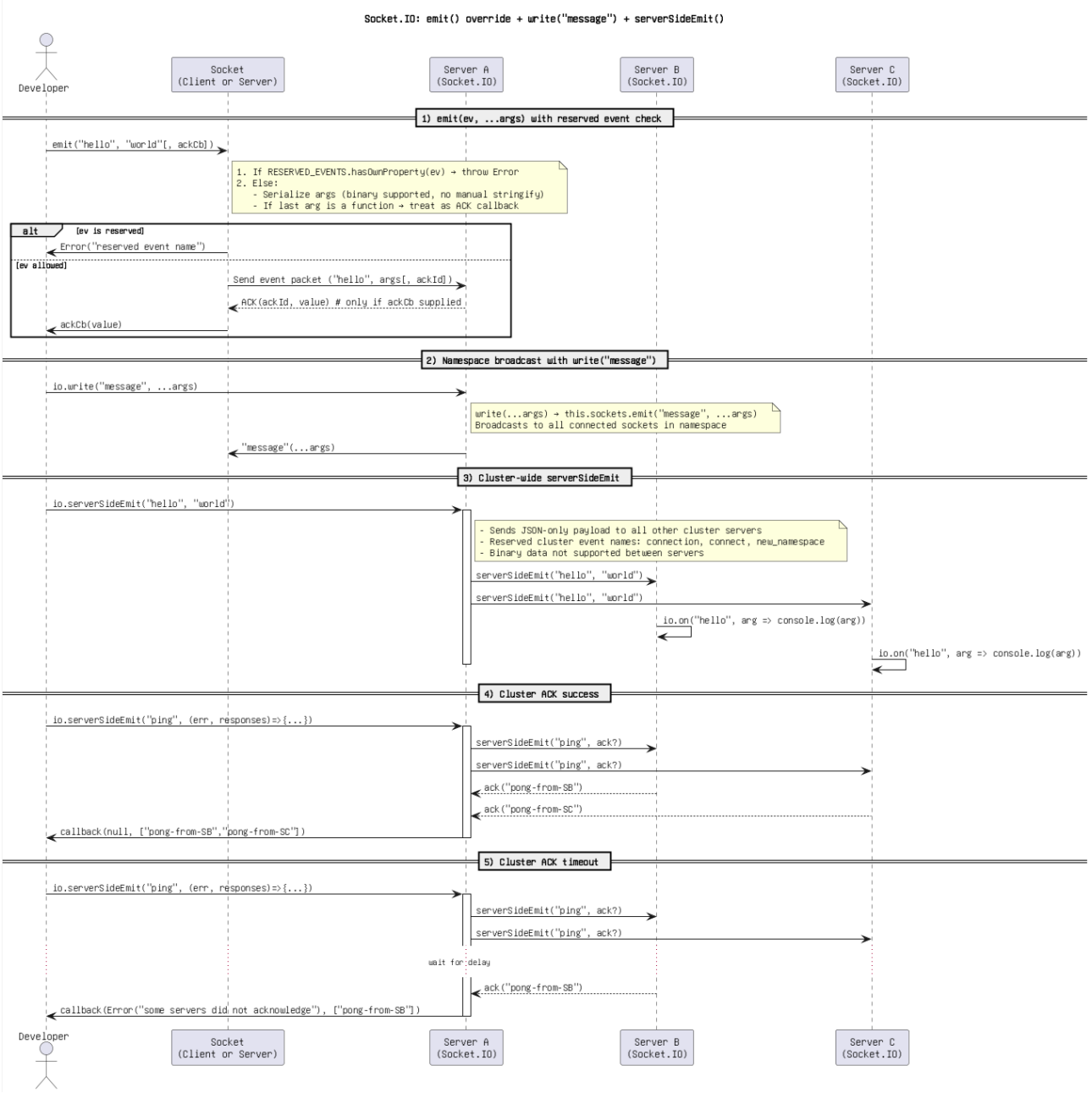

Figure: 44. Socket.IO_ emit() override + write(_message_) + serverSideEmit().png

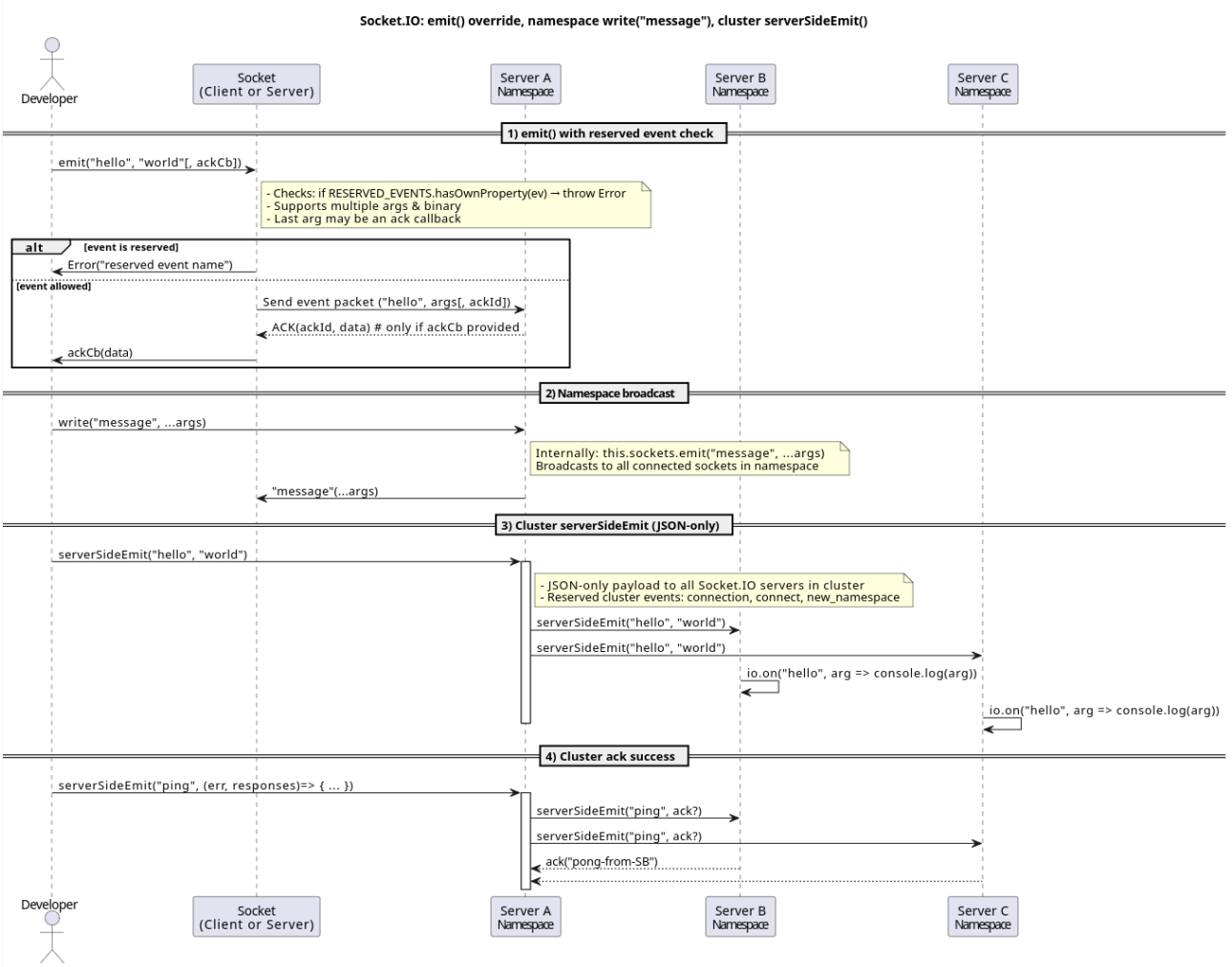

Figure: 45. Socket.IO_ emit() override, namespace write(_message_), cluster serverSideEmit().png

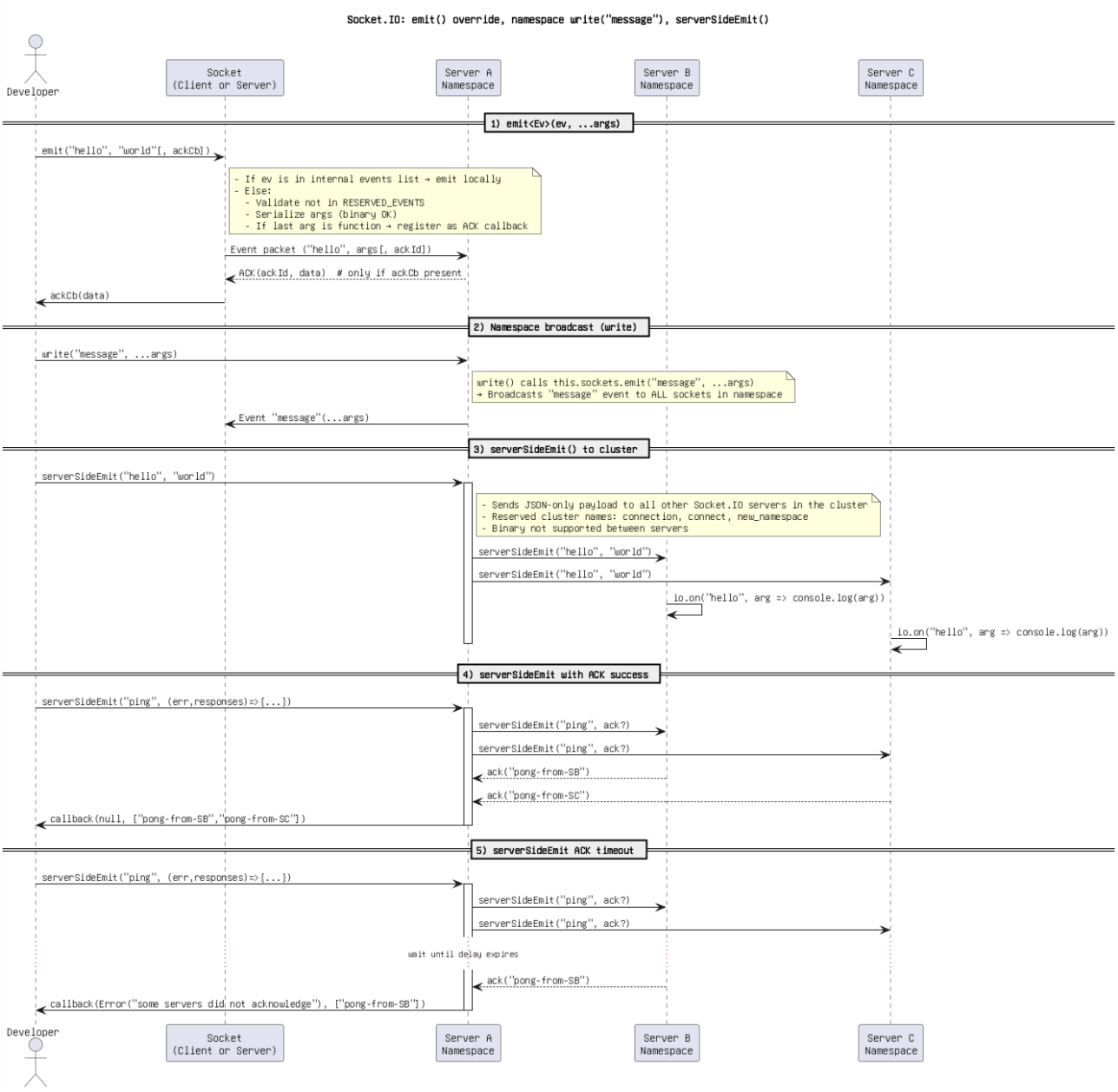

Figure: 46. Socket.IO_ emit() override, namespace write(_message_), serverSideEmit().png

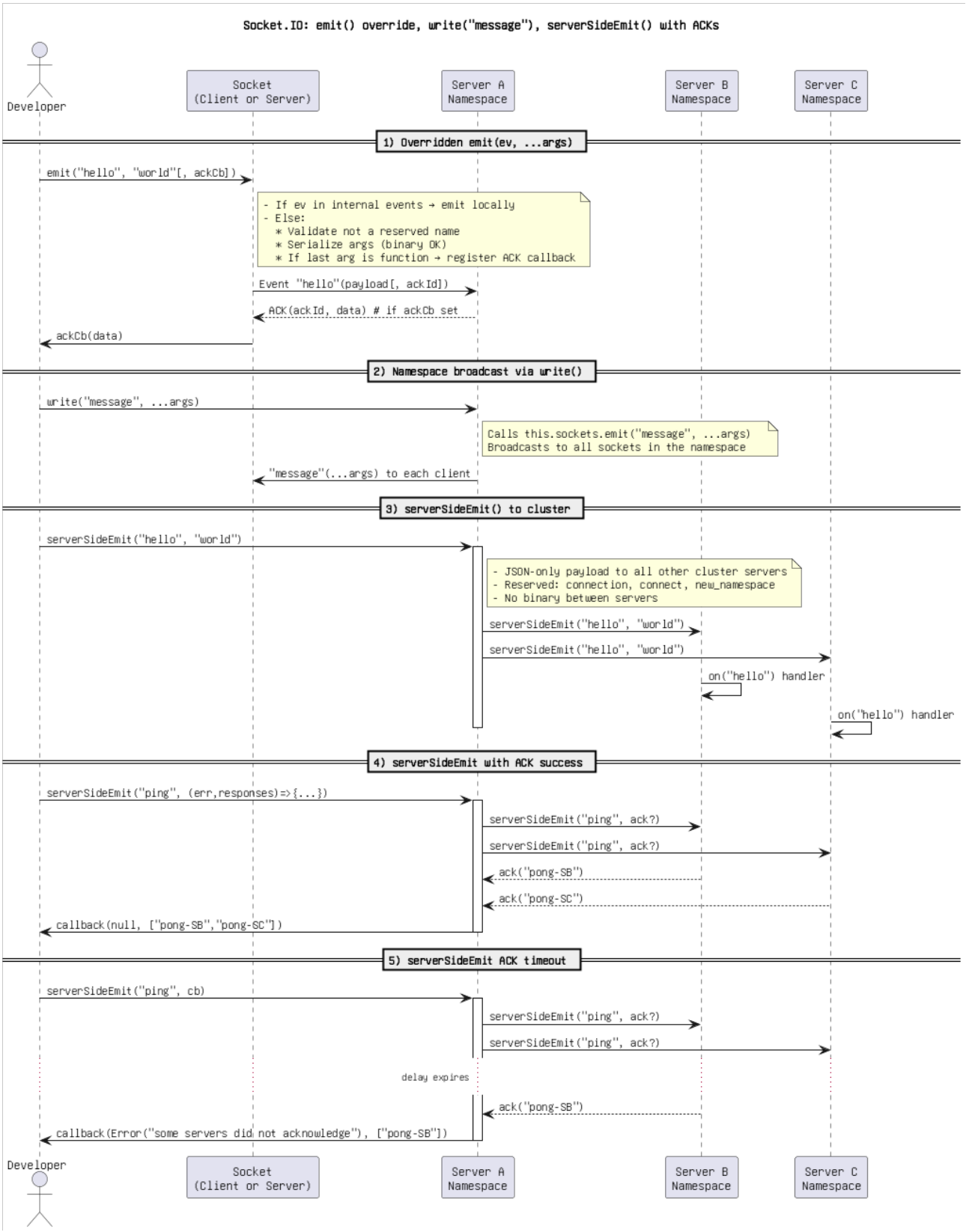

Figure: 47. Socket.IO_ emit() override, write(_message_), serverSideEmit() with ACKs.png

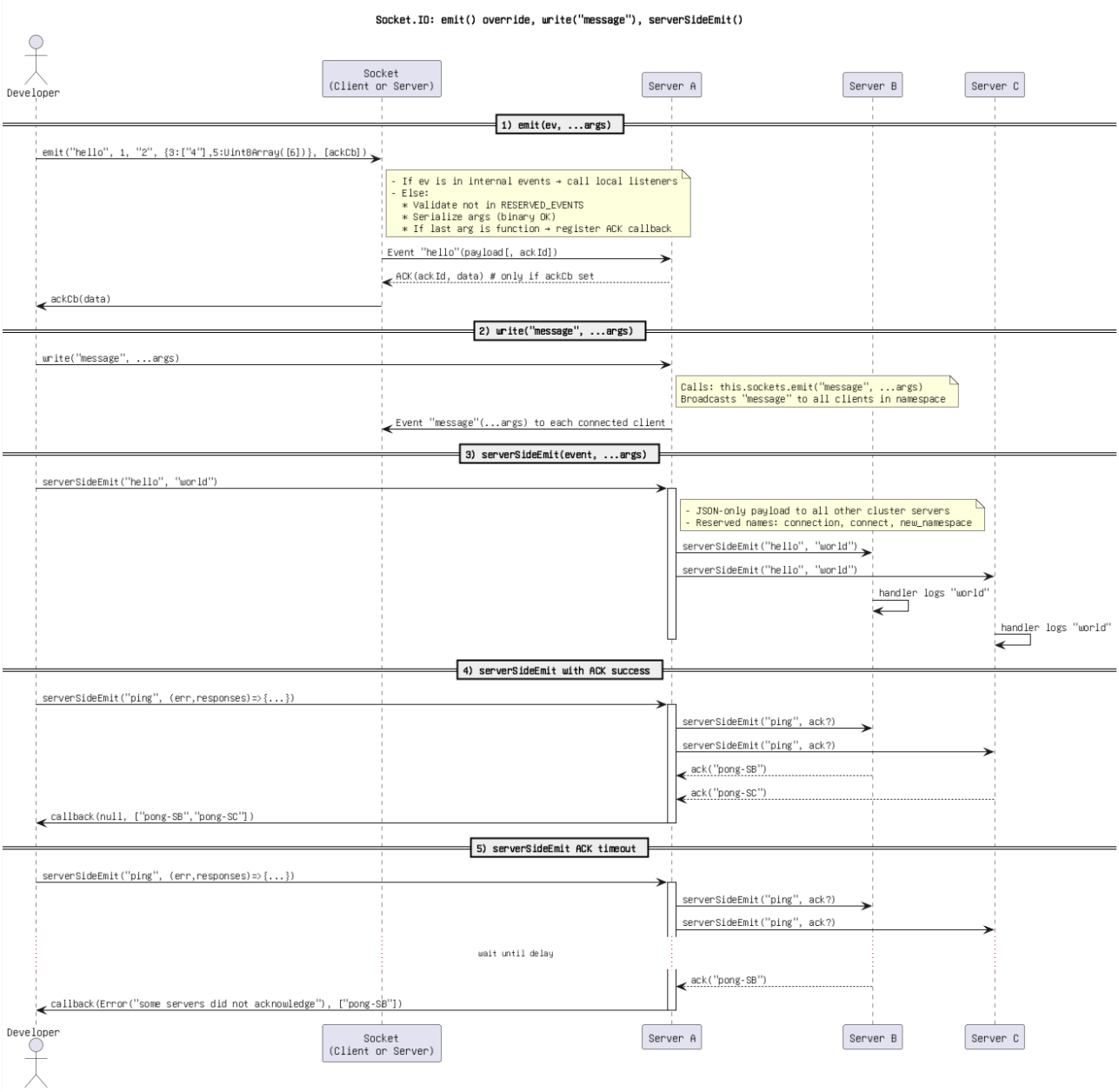

Figure: 48. Socket.IO_ emit() override, write(_message_), serverSideEmit().png

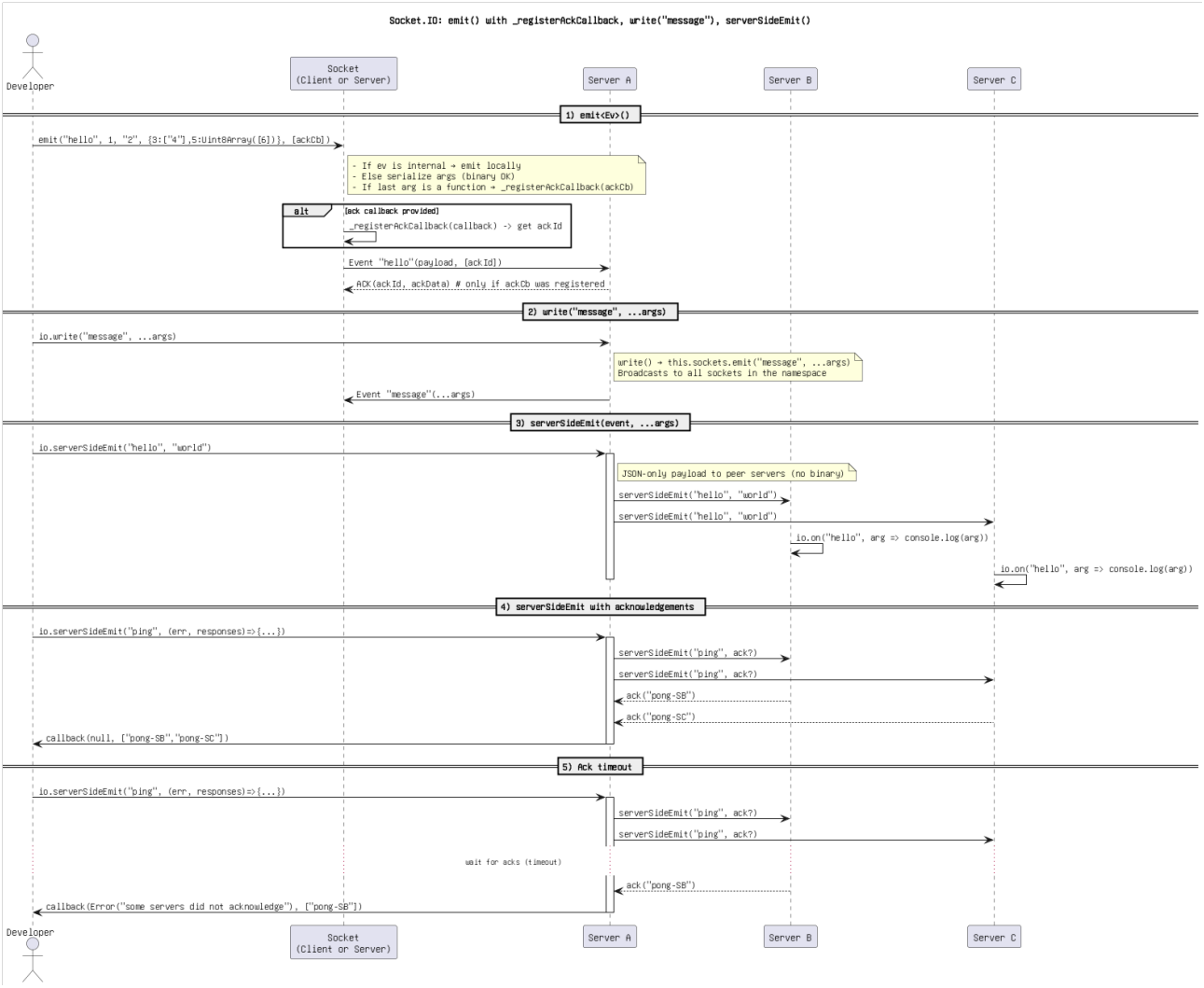

Figure: 49. Socket.IO_ emit() with _registerAckCallback, write(_message_), serverSideEmit().png

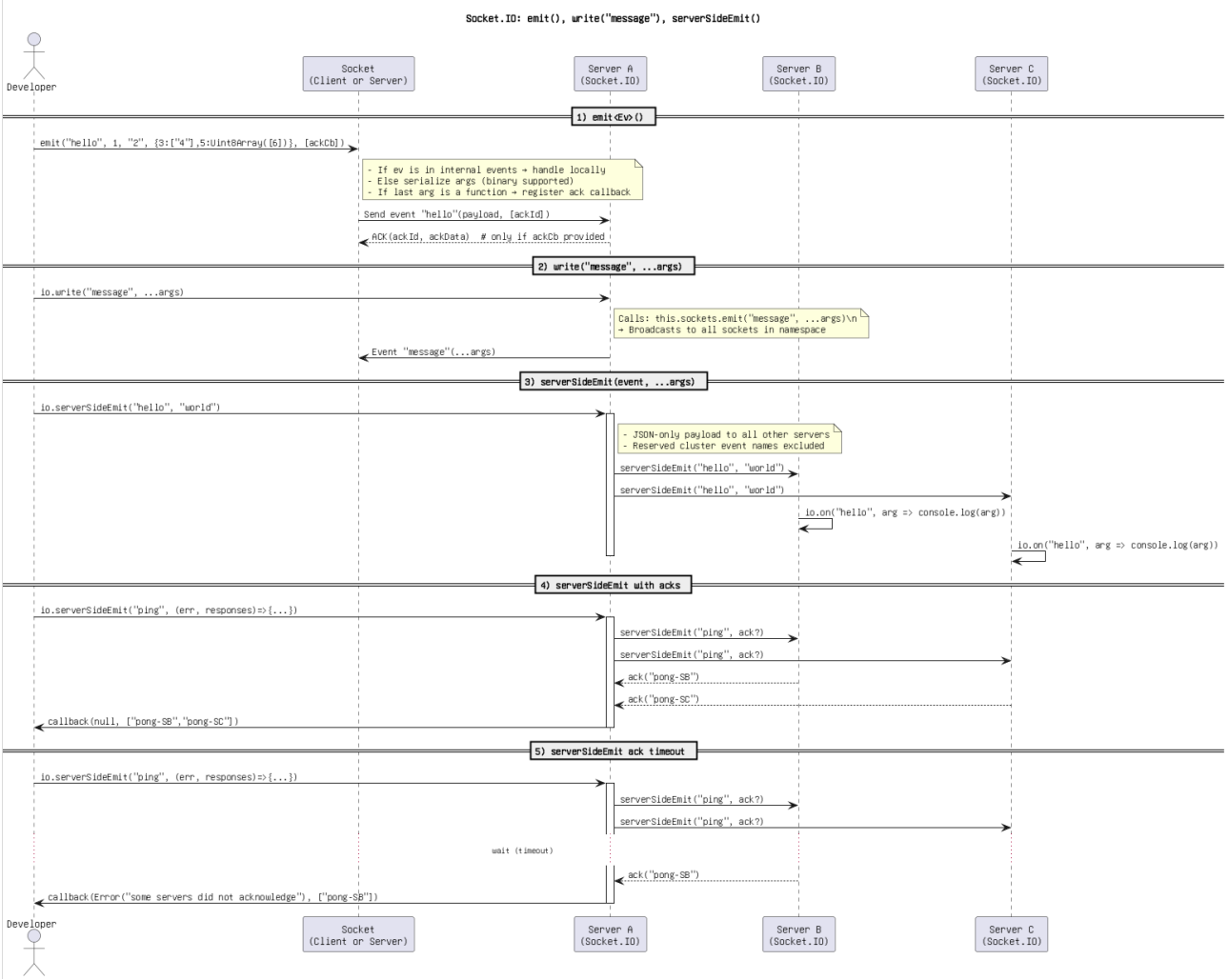

Figure: 50. Socket.IO_ emit(), write(_message_), serverSideEmit().png

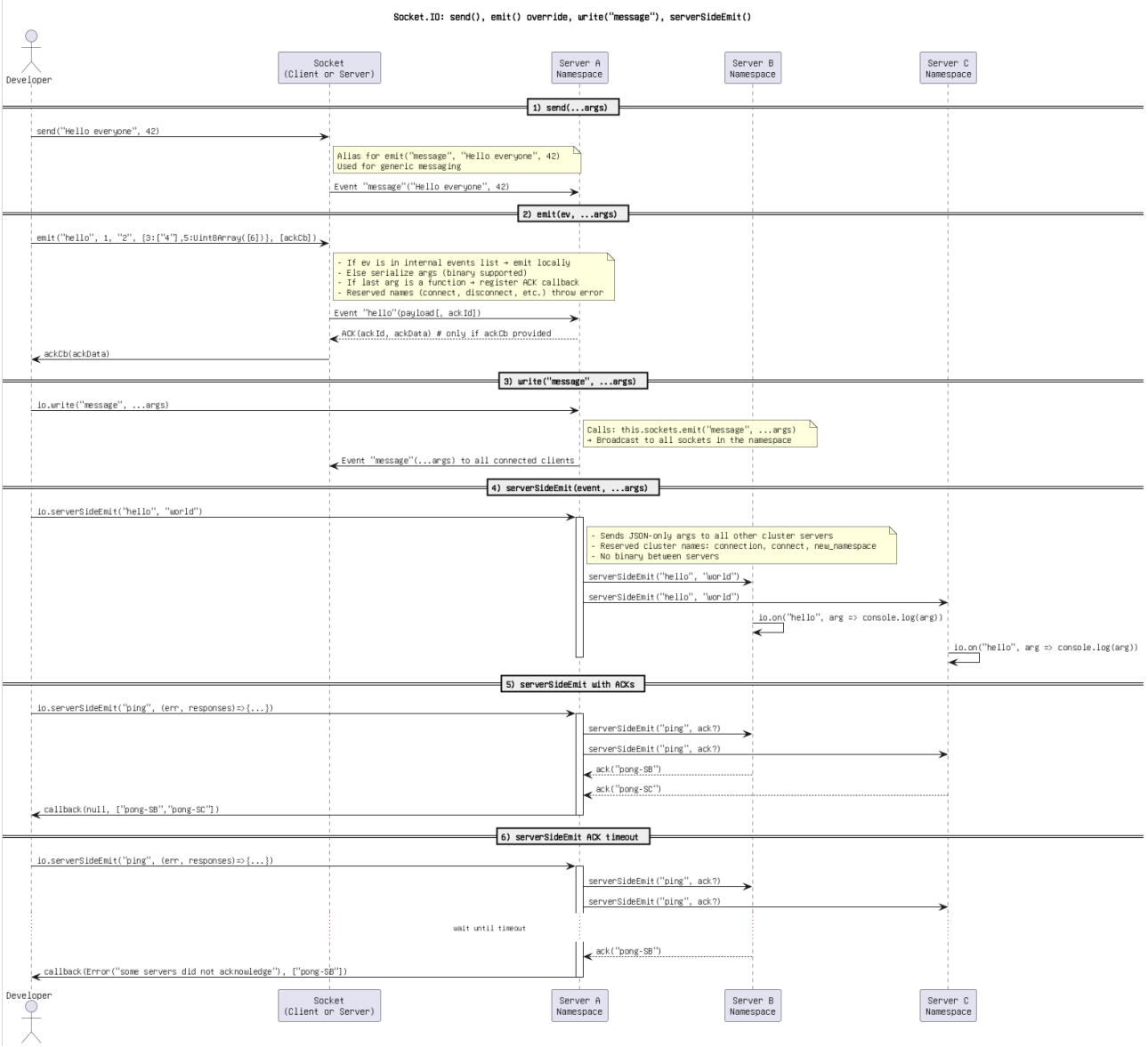

Figure: 51. Socket.IO_ send(), emit() override, write(_message_), serverSideEmit().png

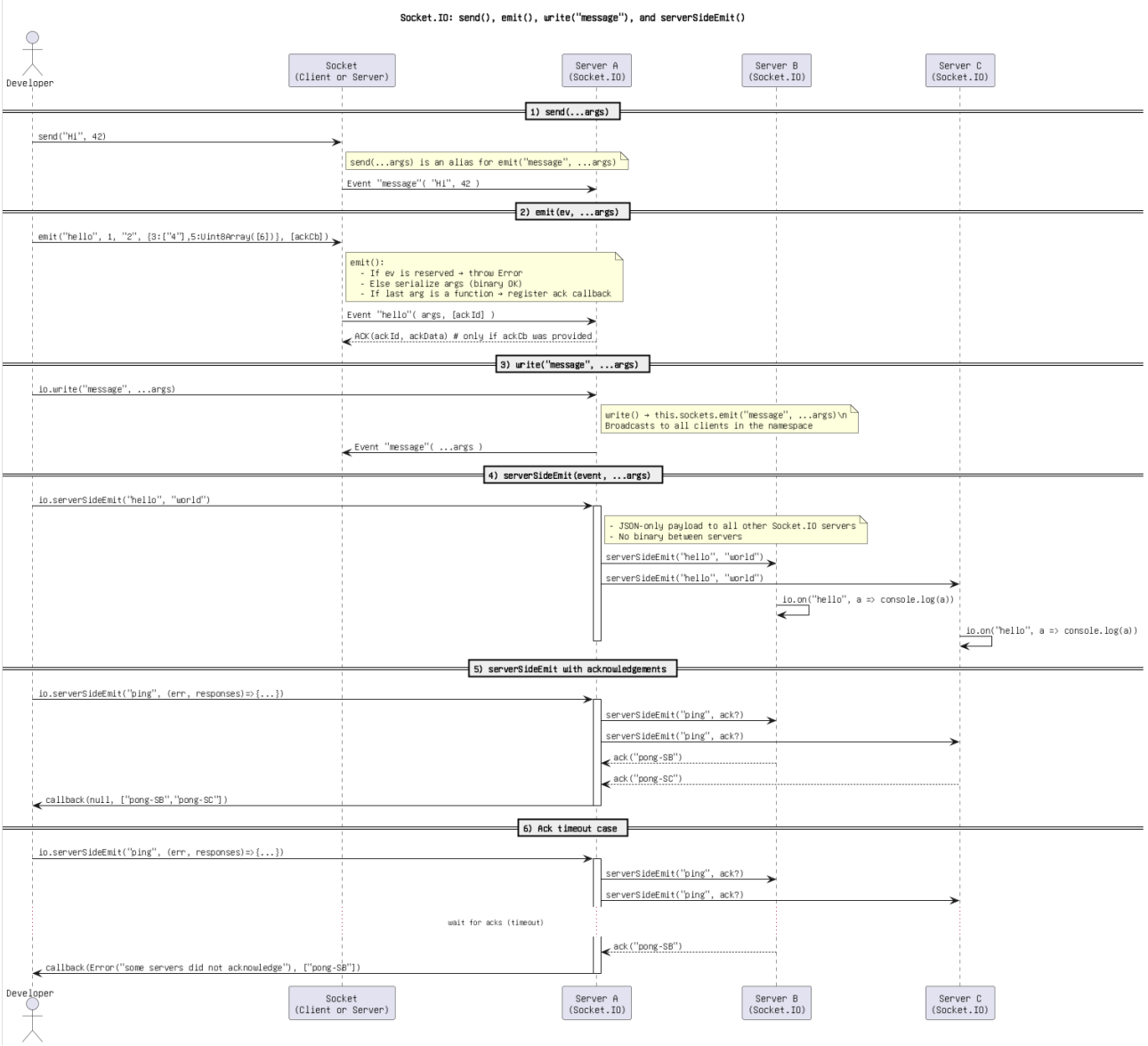

Figure: 52. Socket.IO_ send(), emit(), write(_message_), and serverSideEmit().png

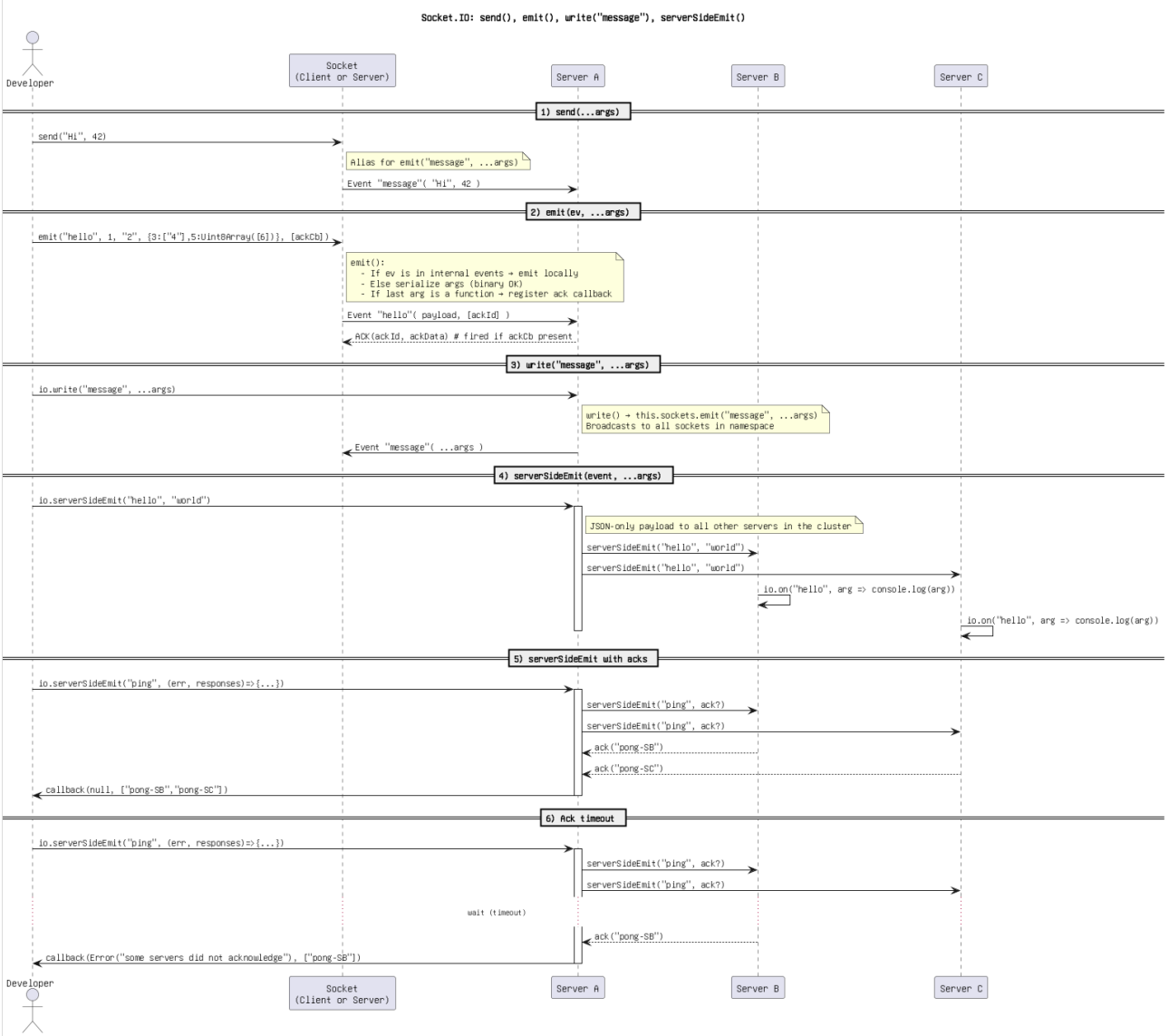

Figure: 53. Socket.IO_ send(), emit(), write(_message_), serverSideEmit().png

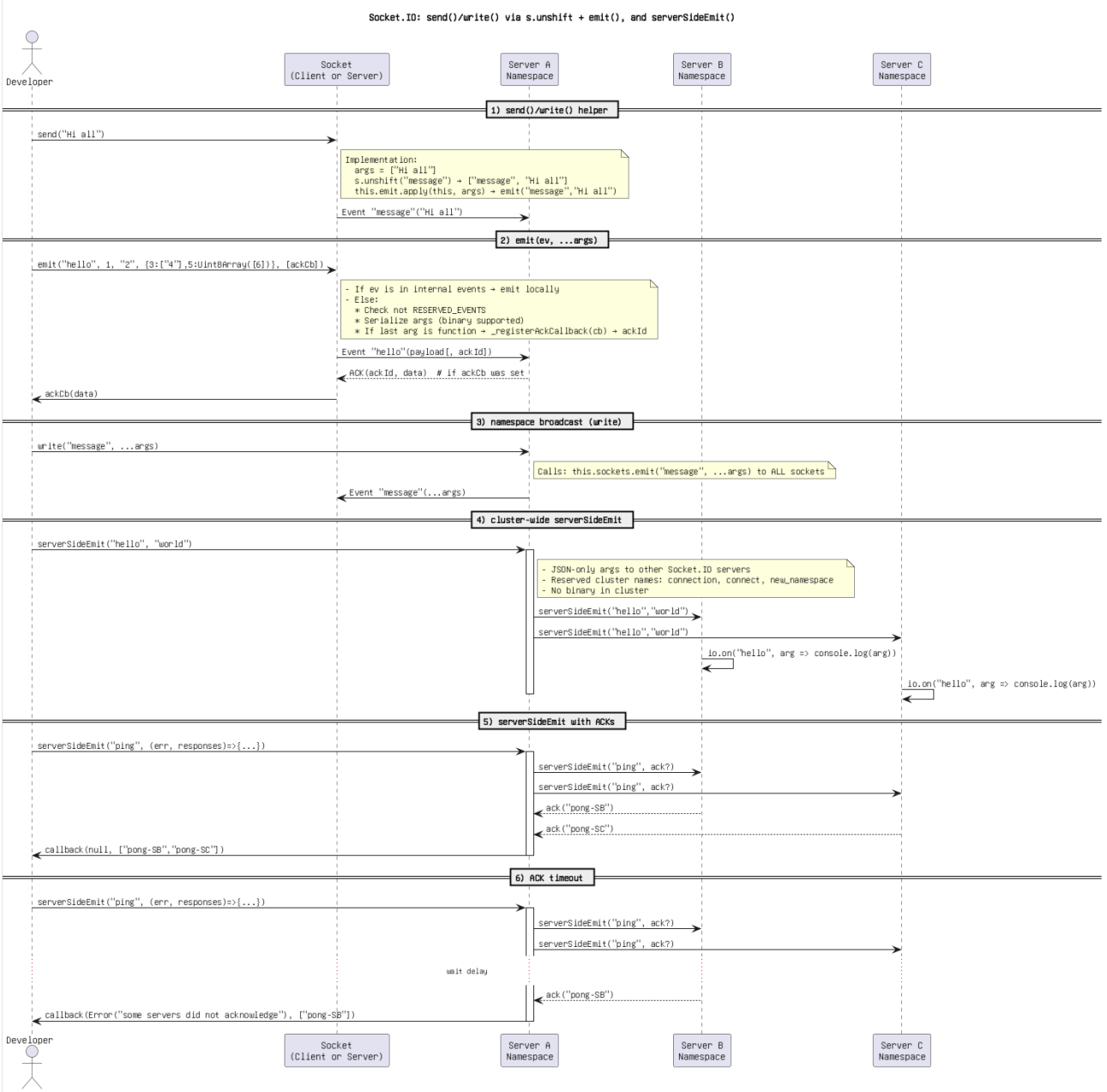

Figure: 54. Socket.IO_ send()_write() via s.unshift + emit(), and serverSideEmit().png

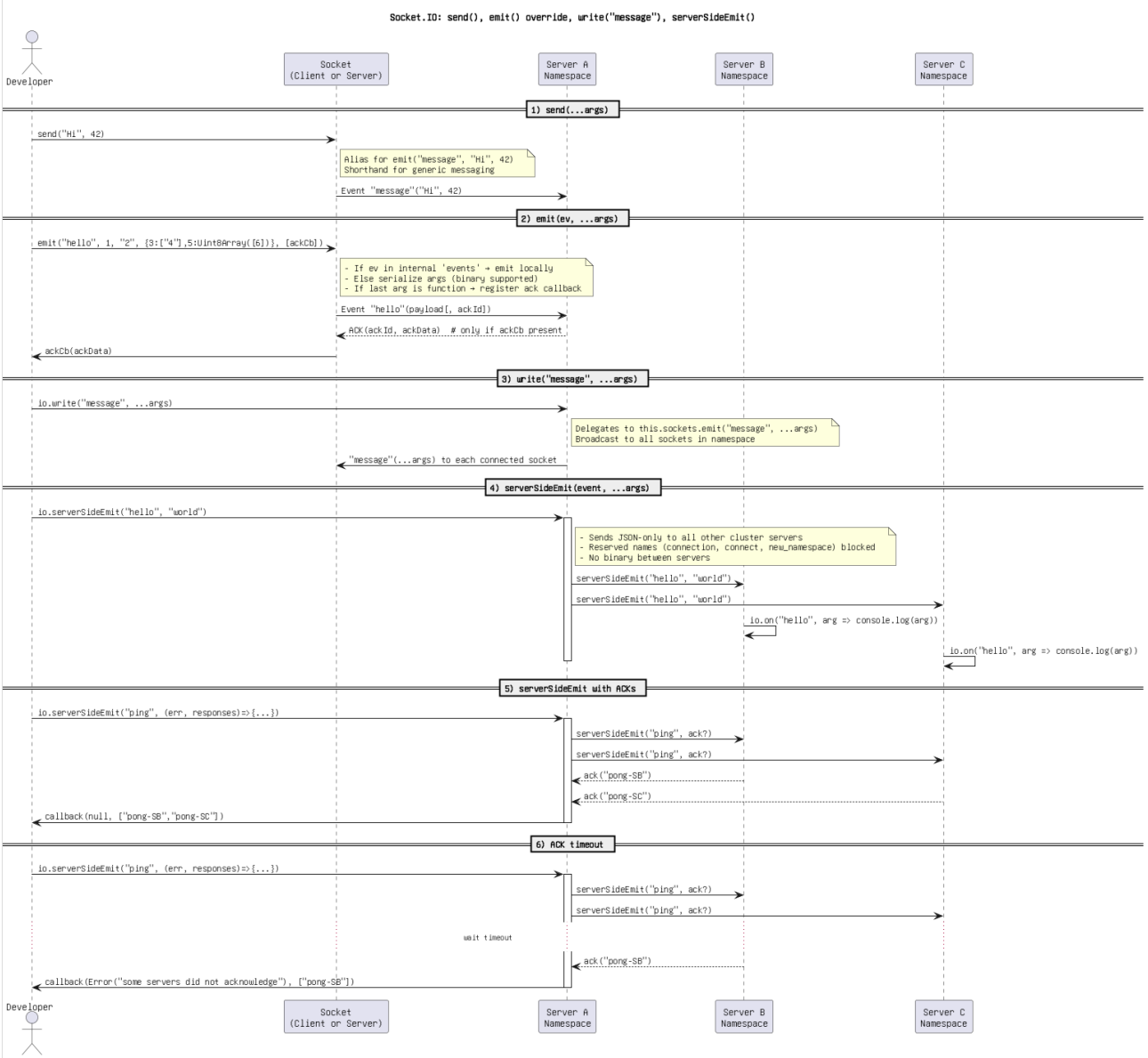

Figure: 55. Untitled DiSocket.IO_ send(), emit() override, write(_message_), serverSideEmit()agram.png

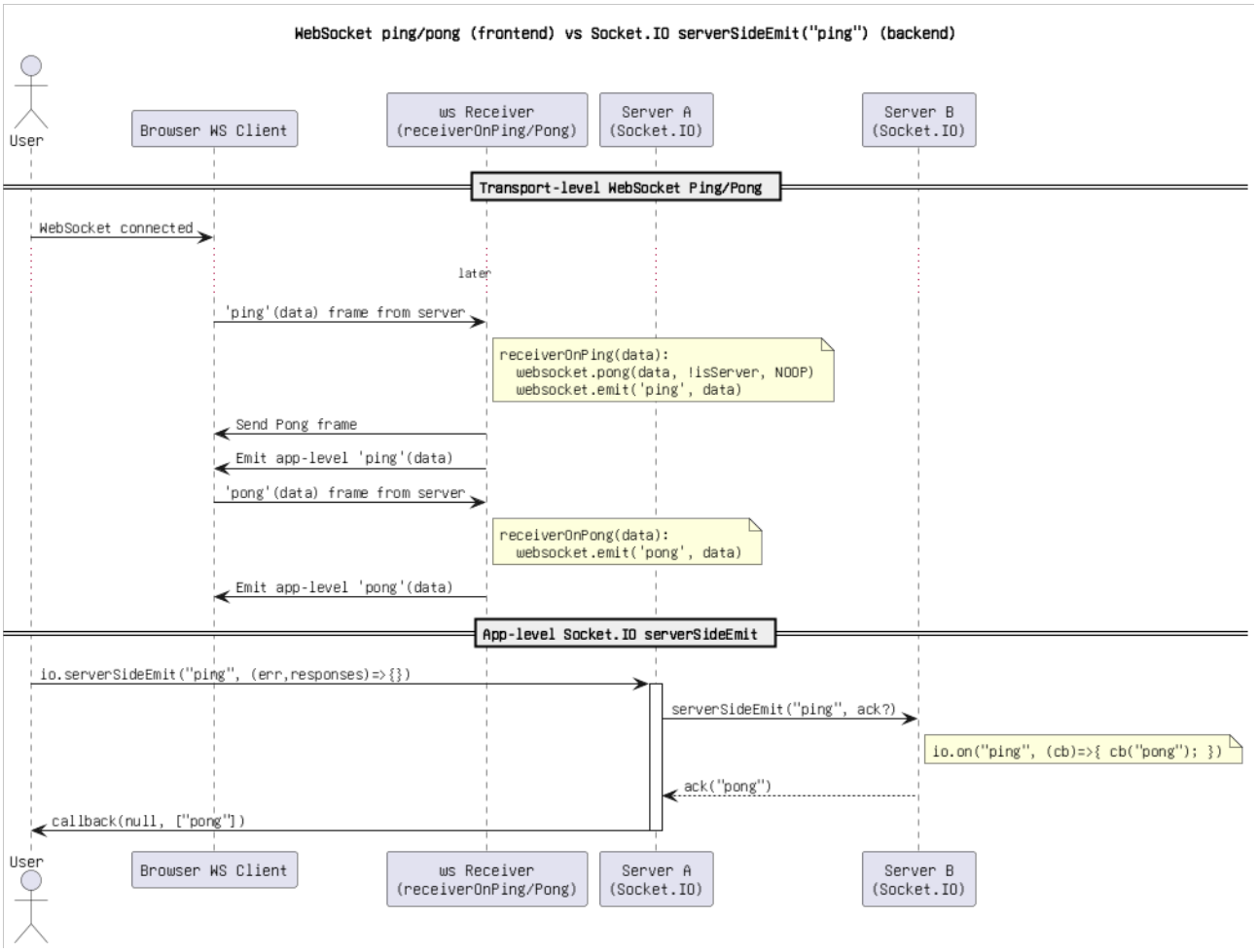

Figure: 56. WebSocket ping_pong (frontend) vs Socket.IO serverSideEmit(_ping_) (backend).png

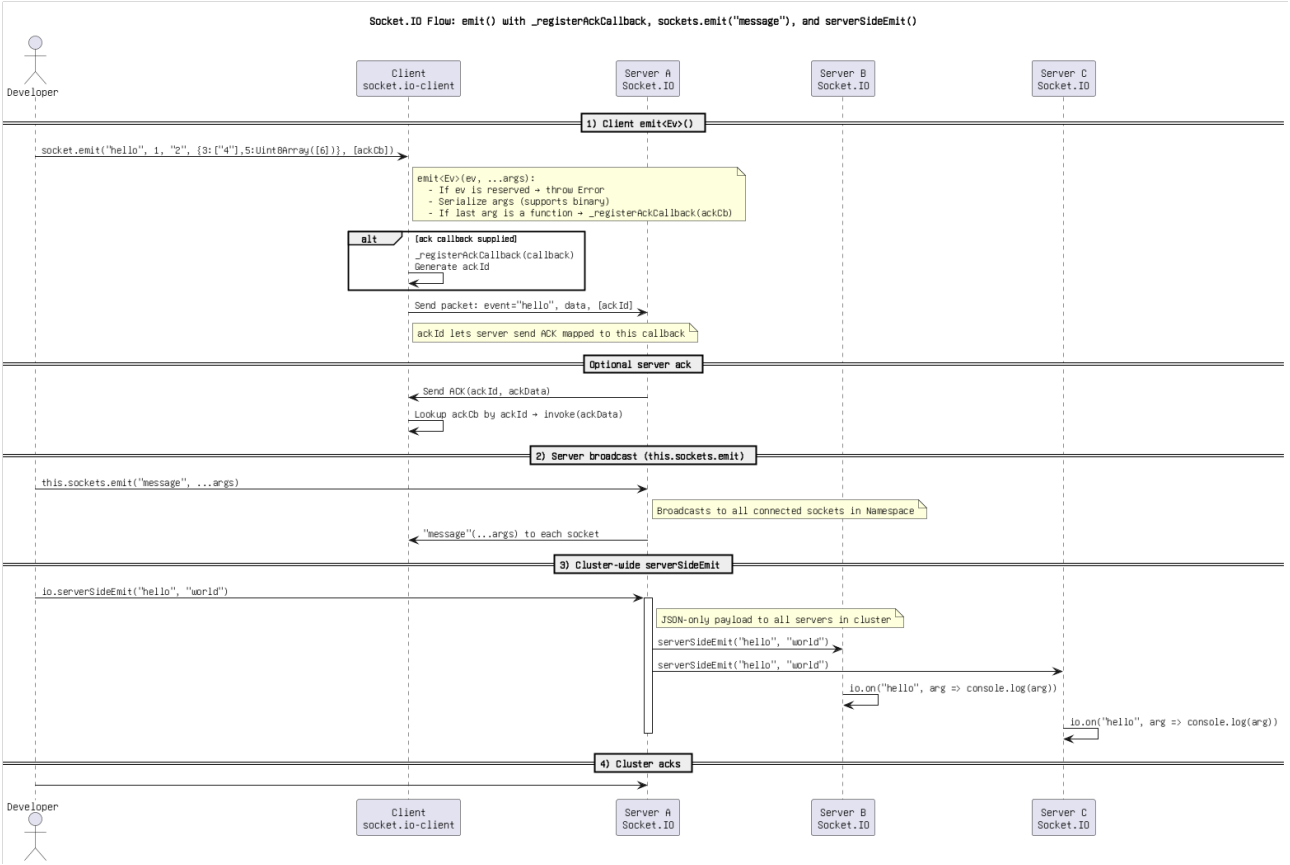

Figure: 57. socket.IO Flow_ emit() with _registerAckCallback, sockets.emit(_message_), and serverSideEmit().png

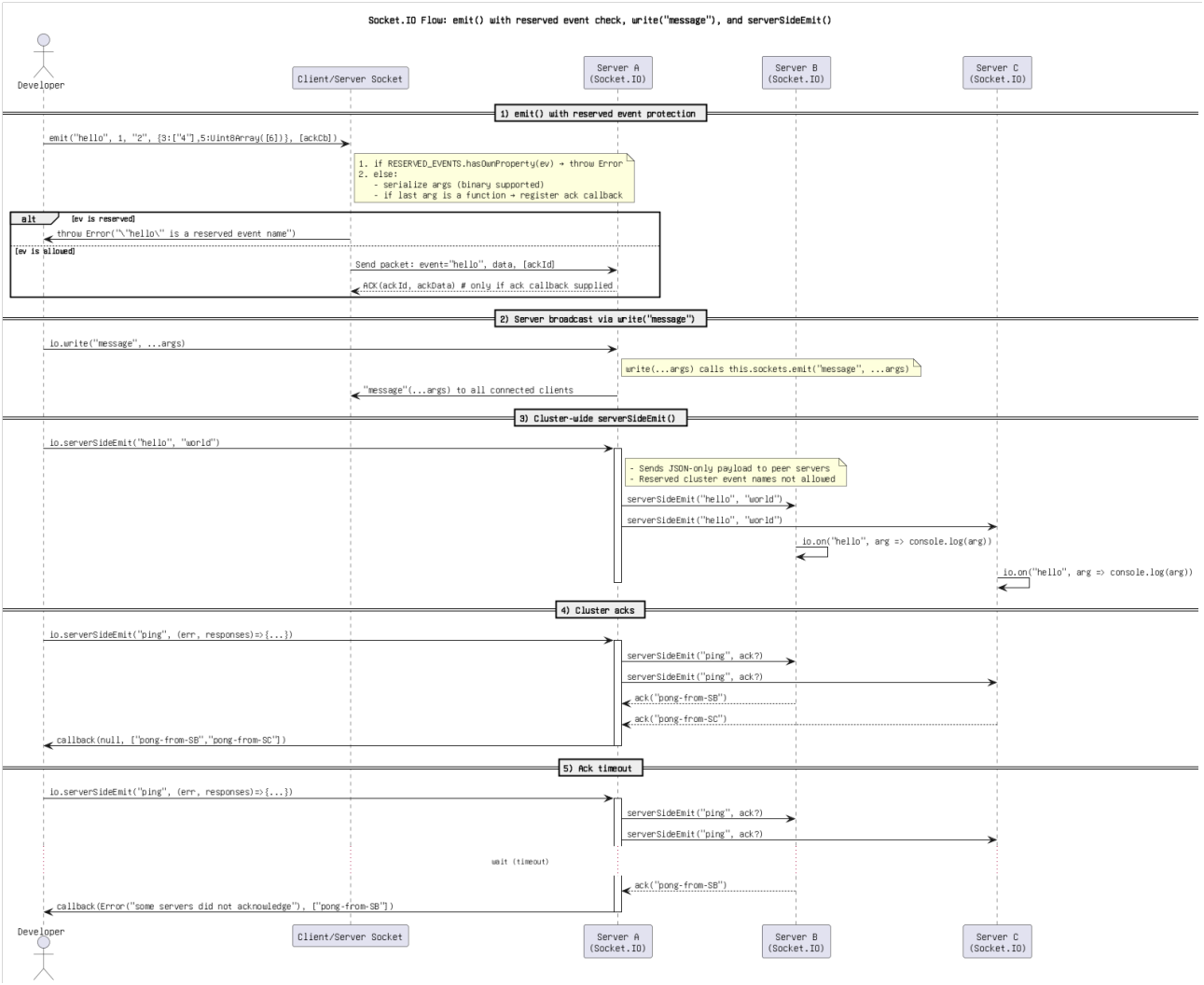

Figure: 58. socket.IO Flow_ emit() with reserved event check, write(_message_), and serverSideEmit().png