perplexity

private void CallBackMetaData(MapCommands map)

есть
switch (_mode)
{
case SateMode.Initialization:
.....
case SateMode.Work:
есть команды
MdCommand.Ok
MdCommand.DataOk ⇒ "dataok",
MdCommand.Data ⇒ "data",

Помоги мне построить каркас логики
MdCommand.Ok - общая коменда подтверждения, что мы получили комманду, связь или что то еще (общая)
MdCommand.Data = передается в канал metadata[MdCommand.Data]="" передеем данные.
обрати внимание если мы формируем подтверждение предыдущим действиям и данные то посылка будет
metadata[MdCommand.Sate]="clientCUDA"
metadata[MdCommand.Ok]="ok"
metadata[MdCommand.Data]=""
если просто данные
metadata[MdCommand.Sate]="clientCUDA"
metadata[MdCommand.Data]=""

От сервера мы можем получить
metadata[MdCommand.Sate]="servrCUDA"
metadata[MdCommand.Ok]="ok"

подтверждении команды и данные
дальше мы можем получить
metadata[MdCommand.Sate]="servrCUDA"
metadata[MdCommand.Ok]="ok"
metadata[MdCommand.Data]="dataok"
это значит что какая то предыдущая команда прошла и данные получены

подтверждении команды и данные
дальше мы можем получить
metadata[MdCommand.Sate]="servrCUDA"
metadata[MdCommand.Data]="dataok"
это значит данные получены все хорошо нужно проверить, что данные есть и дернуть
private void TrySendNext()

подтверждении команды и данные
дальше мы можем получить
metadata[MdCommand.Sate]="servrCUDA"
metadata[MdCommand.Data]="error"
должны повторить операцию с этими данными

подтверждении команды и данные
дальше мы можем получить
metadata[MdCommand.Sate]="servrCUDA"
metadata[MdCommand.Data]=... присутствие этой записи говорит что мы делаем манипуляции с памятью.
в то же самое время могут присутствовать команды metadata[MdCommand.Ok]
MdCommand.Data, MdCommand.DataOk - команда отвечает только за работу с данными
сможешь построить каркас оптимальный по скорости
case SateMode.Work:
реадизации данного блока

Конечно, Алекс! Ниже предлагаю каркас логики для блока `case SateMode.Work:` с учётом твоих описаний и приоритетов:

- Обработка ключей `MdCommand.Ok`, `MdCommand.Data`, `MdCommand.DataOk`
- Разделение логики подтверждения команд и обработки данных
- Минимальная задержка (асинхронность там, где нужно)
- Обработка ошибок и повторов операций
- Вызов метода `TrySendNext()` при необходимости передачи следующих данных

```
case SateMode.Work:
{
    // Лог для входа в режим работы — можно убрать или оставить для отладки
```

```
    Console.WriteLine(">>> Работаем: получили данные в режиме CLIENT Work");

    // Если метаданные почти пусты — скорее всего не нужно ничего делать
    if (map == null || map.Count < 1)
        break;

    // Сброс таймера активности (по твоему примеру)
    _timer.ResetWork();
    _timer.ResetWorkSendCount();

    // Получаем состояние от сервера (например, "servrCUDA")
    map.TryGetValue(MdCommand.State.AsKey(), out var stateValue);

    // Обрабатываем команды подтверждений и данные

    // Обработка команды подтверждения MdCommand.Ok
    if (map.TryGetValue(MdCommand.Ok.AsKey(), out var okValue))
    {
        if (!string.IsNullOrEmpty(okValue) && okValue.Equals("ok", StringComparison.Ordin
        {
            Console.WriteLine(">>> [CLIENT] Получено подтверждение Ok от сервера");

            // Подтверждение, что предыдущий шаг принят
            _transferWaiting = TransferWaiting.Transfer;

            // Возможно, нужно удалить или сместить очередь на следующий пакет
            // Например:
            _dataQueue.TryDequeue(out _);

            // Запускаем проверку/отправку следующих данных
            TrySendNext();

            // Если есть дополнительные действия для Ok — сюда добавить
        }
    }

    // Обработка команды MdCommand.Data
    if (map.TryGetValue(MdCommand.Data.AsKey(), out var dataValue))
    {
        if (string.Equals(dataValue, "dataok", StringComparison.OrdinalIgnoreCase))
        {
            Console.WriteLine(">>> [CLIENT] Получено подтверждение данных dataok");

            // Данные успешно получены сервером — можно смещать очередь и формировать сле
            _dataQueue.TryDequeue(out _);

            TrySendNext();
        }
        else if (string.Equals(dataValue, "error", StringComparison.OrdinalIgnoreCase))
        {
            Console.WriteLine(">>> [CLIENT] Получена ошибка передачи данных — повторяем")

            // Повторяем отправку текущих данных (не удаляем из очереди!)
            // Можно вызвать повторный вызов или установить флаг повтора
            TrySendCurrentOrRetry();
        }
```

```
        else if (!string.IsNullOrEmpty(dataValue))
        {
            Console.WriteLine($">>> [CLIENT] Получены специальные данные: {dataValue}");

            // Если dataValue содержит другую информацию — возможно, нужно работать с пам
            // обработать служебные команды.

            // Если есть команда Ok, обработать её совместно
            if (okValue == null)
            {
                // Если только data (без ok), всё равно стоит проверить очередь и отправи
                TrySendNext();
            }
            // Для сложной логики с памятью сюда вписать дополнительные кейсы
        }
        else
        {
            // Если dataValue пустой — возможно, просто информативная команда
            Console.WriteLine(">>> [CLIENT] Получены данные, но без специального действия
        }
    }

    // Если ни Ok, ни Data не пришли — можно проверить другие команды, если необходимо

    // В конце можно следить за перезапуском таймеров или состояния
    break;
}
```

Дополнительные комментарии и рекомендации

- Метод `TrySendNext()` ты уже используешь как логику попытки передачи следующего элемента очереди.
- Метод `TrySendCurrentOrRetry()` — условный метод, который должен повторять попытку отправки текущих данных, например, повторно вызвать сериализацию и отправку.
- Важно **не удалять элемент очереди при получении ошибки передачи**.
- Если приходит и `Ok`, и `Data` одновременно — считается, что команда подтверждена и данные получены, очередь смещается.
- Таймеры (`_timer`) используем для контроля активности и таймаутов.
- Не забывай обрабатывать непредвиденные команды и возможные состояния — для расширения сделай в конце `default` или дополнительные проверки.
- При необходимости логи желательно отлаживать и отправлять в отдельный логгер, чтобы не тормозить основное исполнение.

Если хочешь — могу помочь и с конкретными шаблонами методов `TrySendNext()` и `TrySendCurrentOrRetry()`, или с детальной обработкой памяти и метаданных.

Сообщи, если надо будет добавить асинхронность или обработку исключений!

⁂