On Zenoh transmission pipeline design

Chen, Hsin-Jui

November 22, 2024

Table of Contents

Transmission Pipeline Philosophy (I)

- Zenoh transmission pipeline is a MPSC (**multiple producer single consumer**) pipeline.
- Transmissions are divided into two pipelines two stages `StageIn` and `StageOut`. Both of them have implemented method to handle pushing and pulling message into queues.

Transmission Pipeline Philosiphy (II)

- The messages are put into the unit of transmissions: `WBatch`.

  Briefly speaking, this structure is served as the buffer.

- Two circular queues of `WBatch` are maintained. One of the queues stores the available batch to put data, and the other is for batches ready to be consumed.

  For simplicity, I will refer to the first queue as the producer queue and the second one as the consumer queue.

The components of the StageIn structure (I)

```
1  struct StageInRefill {
2      n_ref_r: Waiter,
3      s_ref_r: RingBufferReader<WBatch, RBLEN>,
4  }
5  struct StageInOut {
6      n_out_w: Notifier,
7      s_out_w: RingBufferWriter<WBatch, RBLEN>,
8      atomic_backoff: Arc<AtomicBackoff>,
9  }
10 struct StageInMutex {
11     current: Arc<Mutex<Option<WBatch>>>,
12     priority: TransportPriorityTx,
13 }
```

The components of the `StageIn` structure (II)

- `StageInRefill` implements the method that handles the scenarios that `WBatch` is unavailable, the `Waiter` object wait for the `WBatch` release by the consumer.
- `StageInOut` notifies the consumer once a `WBatch` is ready to consume.
- `StageInMutex` make the implementations thread-safe.
- To avoid congestion, some kind of backoff mechanism is adopted, but I have not fully learned it yet, possibly a simple constant backoff.

The components of the StageOut structure (I)

```
1 struct StageOutIn {
2     s_out_r: RingBufferReader<WBatch, RBLEN>,
3     current: Arc<Mutex<Option<WBatch>>>,
4     backoff: Backoff,
5 }
6 struct StageOutRefill {
7     n_ref_w: Notifier,
8     s_ref_w: RingBufferWriter<WBatch, RBLEN>,
9 }
```

The components of the `StageOut` structure (II)

The ideas are very similar to `StageIn`.

- `StageOutIn` implements the method that pulls `WBatch` in the consumer queue. If no `WBatch` is available, then it will wait for the notification.
- `StageOutRefill` implements the method that refill the used `WBatch` back to the producer queue after the transmission.

Table of Contents

The push_network_message() Implementation (I)

```
let mut c_guard = self.mutex.current();
macro_rules! zgetbatch_rets {
  ($fragment:expr, $restore_sn:expr) => {
    loop {
      match c_guard.take() {
        Some(batch) => break batch,
        None => match self.s_ref.pull() {
          Some(mut batch) => {
            batch.clear();
            self.s_out.atomic_backoff.first_write
                .store(/* ommitted */);
            break batch;
          }
```

The push_network_message() Implementation (II)

```
14              None => {
15                drop(c_guard);
16                match deadline_before_drop {
17                  Some(ddl) if !$fragment => {
18                    if !self.s_ref.wait_deadline(ddl) {
19                      $restore_sn;
20                      return false
21                    }
22                  }
23                  _ => {
24                    if !self.s_ref.wait() {
25                      $restore_sn;
26                      return false;
27                    }
28                  }
29                }
30                c_guard = self.mutex.current();
31              }
```

The `push_network_message()` Implementation (III)

This macro is used to get a `WBatch` and if successfully get a `WBatch`, then it will try to write message to the batch by calling the following method:

```
1 batch.encode((&*msg, &frame)).is_ok()
```

The `push_network_message()` Implementation (IV)

The `batch.encode()` will actually be called three time if necessary. The reason is that the `WBatch` may be shared among messages, then the message may not fit into the remaining buffer in the `WBatch`. Even if it get an empty block, the messages still may be too large to fit in, then it will get the `WBatch` until all fragment are written.

The `push_network_message()` Implementation (V)

It is worth noting that `push_transport_message()` will only be called at most two times. I believe the reason is transport messages are mainly flow control messages, which have small sizes. More description on `StageIn` can be found from this link.

The TransmissionPipeline structure

Then StageIn and StageOut structures are encapsulate into TransmissionPipeline structure.

```
impl TransmissionPipeline {                                    // A MPSC pipeline
    pub(crate) fn make(
        config: TransmissionPipelineConf,
        priority: &[TransportPriorityTx],
    ) -> (TransmissionPipelineProducer,
        TransmissionPipelineConsumer) {
    /* This method makes each priority level a StageIn and StageOut structure
     * and then encapsulate the StageIn array (vector) and StageOut array into
     * TransmissionPipelineProducer and TransmissionPipelineConsumer struct.
     **/
}
```

Table of Contents

The TransportStats structure

In the file `io/zenoh-transport/src/common/stats.rs`, it defines the `TransportStats` structure via macros.

```
1  stats_struct! {
2      #[derive(Clone, Debug, Deserialize, Serialize)]
3      pub struct TransportStats {
4          # HELP "Counter of sent bytes."
5          # TYPE "counter"
6          pub tx_bytes,
7
8          # HELP "Counter of sent transport messages."
9          # TYPE "counter"
10         pub tx_t_msgs,
11
12         /* other statistics are omitted */
13     }
14 }
```

The `TransportStats` structure

I have some statistics on number of sent bytes, sent messages, drop messages, etc. To enable this feature, it need to use `"stats"` feature. I do not learn how to include this at compile time and I am not sure whether it will automatically show the statistics on closing.

Discussion: Do we need more detailed measurements?

Since there are already some profiling implementations so do we actually need to have a more detailed measurement.

Precisely speaking, do we need to implement a version that profiles each queue, and not just the whole transport layer statistics.

There are also some statistics that are not provided in the `TransportStats` structure, such as queueing delay.