Flute项目raptorq测试报告

虚拟网卡对

```
# 创建一对虚拟网卡（veth-sender ↔ veth-receiver）
sudo ip link add veth-sender type veth peer name veth-receiver

# 给 veth-sender 分配 IP 192.168.100.1
sudo ip addr add 192.168.100.1/24 dev veth-sender

# 给 veth-receiver 分配 IP 192.168.100.2
sudo ip addr add 192.168.100.2/24 dev veth-receiver

sudo ip link set veth-sender up
sudo ip link set veth-receiver up

# 检查 veth-sender 的 IP
ip addr show veth-sender
# 检查 veth-receiver 的 IP
ip addr show veth-receiver

# 清理 veth pair
sudo ip link del veth-sender
```

RaptorQ批量基准单源块测试

```
cargo bench --bench encode_benchmark

cargo bench --bench decode_benchmark
```

RaptorQ参数关系

Encode & Decode参数

参数名称	说明
`encoding_symbol_length`	每个符号的字节数
`max_number_of_parity_symbols`	最大冗余符号数
`maximum_source_block_length`	单个源块的最大符号数量

参数名称	说明
`symbol_alignment`	符号对齐
`sub_blocks_length`	子块长度
`buffer_size`	发送端缓冲区大小
`transfer_length`	文件传输总长度（编码前）
`symbol_count`	总符号数（编码前）
`nb_block`	总块数
`a_large`	大块尺寸
`a_small`	小块尺寸
`nb_a_large`	大块数量
`block_length_small`	小块尺寸（sbn >= nb_a_large）
`block_length_large`	大块尺寸（sbn < nb_a_large）
`block_length`	当前数据块大小
`buffer_len`	发送缓冲区长度
`nb_source_symbols`	当前数据块包含源符号数量
`nb_parity_symbols`	当前数据块包含冗余符号数量
`transfer_length_after`	编码后的文件尺寸
`MAX_SYMBOLS`	RaptorQ支持的符号最大字节数
`MAX_TRANSFER_LENGTH`	文件传输最大尺寸

Encode & Decode基准测试（单源块）

常量定义

```
const TARGET_TOTAL_BYTES: usize = 10 * 1024 * 1024;  // 编码前单块大小
const MAX_SYMBOLS: usize = 56403;
const KB: usize = 1024;
const MB: usize = 1024 * KB;
const MAX_TRANSFER_LENGTH: usize = 0xFFFFFFFFFF; // 40 bits max
const PRE_PLAN: bool = false; // 是否预编码（Encode测试特有参数）
```

计算发送缓冲区大小

```
fn calculate_buffer_size(
    transfer_length: usize,
    encoding_symbol_length: u16,
    maximum_source_block_length: u64
) -> usize {
    // 计算总符号数
    let symbol_count = div_ceil(transfer_length, encoding_symbol_length as
usize);

    // 计算总块数
    let nb_blocks = div_ceil(symbol_count as u64,
maximum_source_block_length);

    // 计算大块和小块尺寸
    let a_large = div_ceil(symbol_count as u64, nb_blocks);
    let a_small = div_floor(symbol_count as u64, nb_blocks);

    // 计算大块数量
    let nb_a_large = symbol_count as u64 - (a_small * nb_blocks);

    // 对于基准测试，我们使用大块尺寸作为buffer_size
    (a_large * encoding_symbol_length as u64) as usize
}
```

Decode参数配置

```
struct BenchmarkConfig {
    encoding_symbol_length: u16,
    symbol_count: usize,
    overhead: f64,
    maximum_source_block_length: u16,
    symbol_alignment: u8,
    buffer_size: usize,
    transfer_length: usize,
}
```

Encode参数配置

```
struct BenchmarkConfig {
    encoding_symbol_length: u16,
    symbol_count: usize,
    pre_plan: bool,
    maximum_source_block_length: u16,
    symbol_alignment: u8,
    buffer_size_mb: usize,
}
```

检查编码后的传输长度

```
fn validate(&self) -> Result<(), String> {
    if self.symbol_count > MAX_SYMBOLS {
        return Err(format!(
            "symbol_count {} exceeds maximum {}",
            self.symbol_count, MAX_SYMBOLS
        ));
    }

    if self.encoding_symbol_length % self.symbol_alignment as u16 != 0 {
        return Err("encoding_symbol_length must be divisible by symbol_alignment".into());
    }

    // 检查编码后的传输长度是否超过限制
    // let total_symbols = (self.symbol_count as f64 * (1.0 + self.overhead)) as usize;
    let block_size: usize = self.encoding_symbol_length as usize * self.maximum_source_block_length as usize;
    let max_source_blocks_number: usize = u8::MAX as usize;
    let mut size = block_size * max_source_blocks_number;
    let transfer_length_after = self.transfer_length + (self.encoding_symbol_length as usize * self.symbol_count);
    if size > MAX_TRANSFER_LENGTH {
        size = MAX_TRANSFER_LENGTH;
    }
    if transfer_length_after > size {
        return Err(format!(
            "transfer_length_after {} exceeds maximum {}",
            transfer_length_after, size
        ));
    }

    Ok(())
}
```

Decode参数组合

```
const TARGET_TOTAL_BYTES: usize = 10 * 1024 * 1024;
let encoding_symbol_lengths = [5000, 10000, 20000, 40000, 50000, 55000, 60000];
let transfer_lengths = [1024 * MB]; // 测试不同大小的文件
let overheads = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35];
let memory_options = [50, 100, 200, 150, 250];
let alignment_options = [1, 4, 8];
```

Encode参数组合

```
const TARGET_TOTAL_BYTES: usize = 5 * 1024 * 1024;
let encoding_symbol_lengths = [5000, 10000, 20000, 40000, 50000, 55000,
60000];
let transfer_lengths = [1024 * MB]; // 测试不同大小的文件
let memory_options = [50, 100, 200, 150, 250];
let pre_plan_options = [PRE_PLAN];
let alignment_options = [1, 4, 8];
```

Decode最佳参数组合

```
=== FINAL RESULTS ===
Best Overall: 4622.7 Mbit/s
Config: BenchmarkConfig { encoding_symbol_length: 55000, symbol_count: 100,
overhead: 0.25, maximum_source_block_leng
th: 100, symbol_alignment: 8, buffer_size: 5500000, transfer_length:
1073741824 }
```

Encode最佳参数组合

```
=== FINAL RESULTS ===
Best Overall: 12979.7 Mbit/s
Config: BenchmarkConfig { encoding_symbol_length: 50000, symbol_count: 100,
pre_plan: false, maximum_source_block_len
gth: 100, symbol_alignment: 8, buffer_size: 5000000, transfer_length:
1073741824 }
```

测试结果

```
Total time: 158.32 seconds
Total packets: 44173
Total data: 2318.56 MB
Bytes_written: 1073741824 bytes
Average rate: 122.85 Mbps
Average rate: 15.36 MB/s
Packet rate: 279.01 packets/second
```

配置参数

```yaml
# config_1024mb_raptorq.yaml
# FLUTE 1024MB文件传输专用配置 (raptorq)# Rate under: unlimited Mbits/sec
sender:
  network:
    destination: "192.168.100.2:3500"  # 使用 veth-receiver 的 IP
bind_address: "192.168.100.1"
    bind_port: 0
    send_interval_micros: 1

  fec:
    type: "raptorq"
    encoding_symbol_length: 55000
    max_number_of_parity_symbols: 100
    encoding_symbol_id_length: 1 # raptorq无关参数
    maximum_source_block_length: 100
    symbol_alignment: 8
    sub_blocks_length: 1

  flute:
    tsi: 1
    interleave_blocks: 4     # 更高块交错

  logging:
    progress_interval: 100  # 更少日志

  files:
    - path: "/home/halllo/flute-main/examples/flute-
sender/src/files_send/test_1024mb.bin"
      content_type: "application/octet-stream"
      priority: 0
      version: 1

  # 新增速率控制参数
  max_rate_kbps: 0
  send_interval_micros: 1

receiver:
  network:
    bind_address: "192.168.100.2"  # 绑定到 veth-receiver    port: 3500

  storage:
    destination_dir: "/home/halllo/flute-main/examples/flute-
receiver/src/files_save"
    enable_md5_check: false

  logging:
    progress_interval: 500

  advanced:
    buffer_size: 8388608     # 4MB缓冲区
    cleanup_interval: 2000   # 减少清理频率
    log_interval: 500       # 减少日志频率
    max_memory_mb: 8192      # 8GB内存限制
```

存在问题

在提高冗余符号的同时，单块的编解码速率会提高，但考虑到大文件的多块传输，这样会导致发送端总共传输的数据增加，最后耗时几乎不变，甚至更长。基准测试只能进行单源块的编解码测试，无法测试多源块下的传输效率。