Backend Coding Challenge
Jan 2017
Programming Exercise

You have been asked to build a server to reassemble UDP packets into N discrete messages. Design an in-memory data model (eg. No database or filesystem). Leverage the data model to store the fragments from which the completed message can be reassembled. Provide an implementation of the consumer/reassembly process. The output of the system should be the completed message number and its corresponding sha256 hash. If all fragments arrived, the hash should line up with the sha256 hash output by the emitter. If the process crashes, there is no expectation of recovery for data in flight.

Fragments delivered in the packet should be stored in the data model. A packet with the EOF flag can arrive before other packets in the same message, and all packets need to be included. If the completed message has not been received within 30 seconds, an error should be issued. The error should indicate which pieces of the specific message did not arrive. (eg. perhaps byte ranges 3-5 and 72-79 did not arrive).

The implementation may be written in any language using corresponding tech stacks. If no extra credit is attempted, one instantiation of your process must run and consume data off the UDP port.

Please include a readme describing the design of the implementation, how to execute the server, and any known deficiencies or limitations. If a partial solution is delivered, describe which requirements are not met and how you would evolve the code to meet those requirements.

Multi-threading extra credit (Medium Difficulty)

Due to UDPs lack of reliable delivery and the high rate of incoming packets, you will need to run 4 consumer threads concurrently. This will keep incoming packets from being lost while the server is processing the last packet. The data model must be enhanced to support parallel worker threads. Each consumer will be listening for UDP communications over a single, shared port.

The implementation may be written in any language using corresponding tech stacks, however, it is a requirement that 4 threads of your process run concurrently consuming data off the UDP port.

Fault-tolerant and multi-process extra credit (Difficult)

Your implementation should run in four discrete processes. Your implementation should also be fault-tolerant and limit data loss in the event of a crash or a restart. This involves persisting in-flight data to disk or a database.

What is included in this ZIP

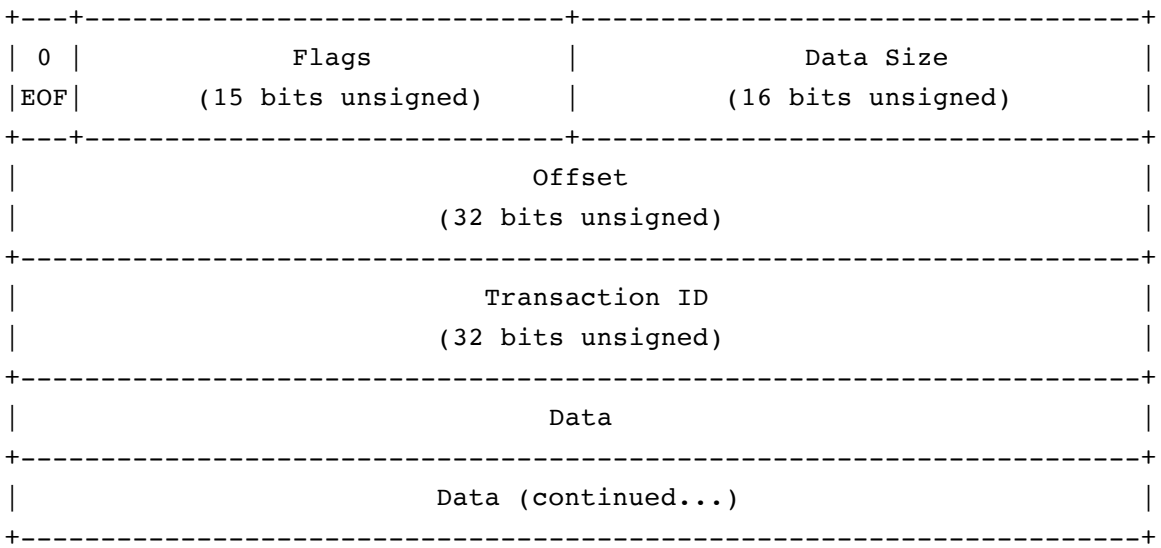
Included is a node.js client program that will emit 10 messages of variable size on UDP port 6789 on the local loopback. The sha256 for each message is computed and output as follows:

```
> node udp_emitter.js
Emitting message #1 of size:1022990
```

```
sha256:ccdbc4c6d006e4b600f784e90534e72ab2c952d333e821ed6f0753690f30b6e5
Emitting message #2 of size:196392
sha256:ffc846e16558fa7501f6deb71ee9c4e1d9db0eeb5421deadafbcd9e605f249e8
Emitting message #3 of size:865334
sha256:7b60a643bbcecc48f5277f0c58006f0644e557e1fa25cee220142d04b18f0402
Emitting message #4 of size:961815
sha256:c39d75497a7d9c7d758fec1bd98edf318f4819b06f185c80ab2b22b044b40b6c
Emitting message #5 of size:470530
sha256:2c349e023f6536e874f2695b32c562680cd488520ea902b4134e4e417756b347
Emitting message #6 of size:260786
sha256:98f7b66ba315e40f81a98c8072336ec360441c939f7f3328293ac677406486b2
Emitting message #7 of size:435191
sha256:1cad3ba5b864c1905cbc1f841ce7465ddefc81f44fc2c627948efdecac1b5c3a
Emitting message #8 of size:459077
sha256:d3644218d5d96c41f5cf18843adf71eb7dbbb4a9cdb2af1b310a8d8c53763f25
Emitting message #9 of size:22273
sha256:fd428f2c7e7bec03811d595320b1ad69ee79e143710e8ae330616321bc64a4bf
Emitting message #10 of size:10409
sha256:30a0022f6e89c4d28691ce15387201ffbbcc99f30773584d390c5984330557ef
```

This program should run on any recent version of node.js. Each time the program is run, M random size messages are created, each with random data. The message is decomposed into N random size fragments, which are emitted in random order. Each packet containing a fragment is emitted twice, so the receiver should handle duplicate packets without error.

The packet payload is as follows:



Example output is as follows:

```
> node ./udp_server.js
Message #1 length: 1022990
sha256:ccdbc4c6d006e4b600f784e90534e72ab2c952d333e821ed6f0753690f30b6e5
Message #2 length: 196392
```

sha256:ffc846e16558fa7501f6deb71ee9c4e1d9db0eeb5421deadafbcd9e605f249e8
Message #3 length: 865334
sha256:7b60a643bbcecc48f5277f0c58006f0644e557e1fa25cee220142d04b18f0402
Message #4 length: 961815
sha256:c39d75497a7d9c7d758fec1bd98edf318f4819b06f185c80ab2b22b044b40b6c
Message #5 length: 470530
sha256:2c349e023f6536e874f2695b32c562680cd488520ea902b4134e4e417756b347
Message #6 length: 260786
sha256:98f7b66ba315e40f81a98c8072336ec360441c939f7f3328293ac677406486b2
Message #7 Hole at: 66
Message #7 Hole at: 2324
Message #7 Hole at: 3722
Message #7 Hole at: 4416
Message #7 Hole at: 5490
Message #7 Hole at: 9873
Message #7 Hole at: 12500
Message #7 Hole at: 16474
Message #8 length: 459077
sha256:d3644218d5d96c41f5cf18843adf71eb7dbbb4a9cdb2af1b310a8d8c53763f25
Message #9 length: 22273
sha256:fd428f2c7e7bec03811d595320b1ad69ee79e143710e8ae330616321bc64a4bf
Message #10 length: 10409
sha256:30a0022f6e89c4d28691ce15387201ffbbcc99f30773584d390c5984330557ef