System Design

Modules

1. PCAP Ingest Module

- **Role:** Reads raw .pcap files and extracts packet-level data.
- **Functionality:**
    - Uses external libraries (or internal code) to extract payloads.
    - Strips redundant Ethernet/IP/UDP headers (step 1).
- **Input:** Raw .pcap file (A LOT of GB).
- **Output:** Sequence of decoded IEX DEEP packets (payload-only).

---

2. Packet Parser

- **Role:** Decodes DEEP messages and splits data by symbol and message type.
- **Functionality:**
    - Parses payloads using IEX DEEP using the official spec.
    - Splits each message into per-symbol files (step 2)
    - Further categorizes each message by type (e.g., trade, quote, LULD, etc.) (step 3).
    - Rare event types go into a fallback "misc" bucket.

---

3. Event Histogram Generator

- **Role:** Gathers metadata/statistics on message type frequencies.
- **Functionality:**
    - Scans parsed data to count frequency of each event type per symbol.
    - Helps decide which events get dedicated files vs go in fallback.
    - Also assists in optimizing future formats (extension)
- **Note:** Not really sure if this is needed. David suggested this.

---

4. Column Based Field Splitter

- **Role:** Optimizes compression via field-level decomposition.
- **Functionality:**
    - Instead of storing messages as binary blobs, it splits into fields (e.g., symbol, timestamp, price, size).

    - Stores these fields in columnar format, e.g., CSV or binary arrays.
- **Why:** Columnar storage often compresses better (especially for numerical time series).

---

5. Compressor

- **Role:** Compresses per-symbol/event/field files.
- **Functionality:**
    - Use zstd, LZ4, or custom schemes.
    - Optionally apply dictionary compression for repeated symbols, prices, etc.
- **Output:** Folder of compressed files (small, queryable, per-symbol).

---

6. FUSE Virtual Filesystem Overlay

- **Role:** Reconstruct .pcap files in real-time from compressed data (step 5).
- **Functionality:**
    - Implements a FUSE user-mode filesystem.
    - When user "reads" a file like mnt/virtual/AAPL.pcap, the system:
        - Pulls symbol + event files from disk
        - Re-encodes messages
        - Adds Ethernet/IP/UDP headers
        - Streams a virtual .pcap on the fly
- **Use case:** Compatibility with tools that expect .pcap, like Wireshark.
- **Note:** Not sure how this works yet

Intended I/O

```
[ PCAP File (.pcap) ]
      │
      ▼
[ PCAP Ingest Module ]
      │  extracts information
      ▼
[ Raw DEEP Payloads (binary) ]
      │  parsed + routed
      ▼
[ Packet Parser & Demuxer ]
      │  writes per-symbol/type
      ▼
[ /parsed/<SYMBOL>/<TYPE>.bin files ]
```

What parsed might look like:

```
/parsed/
├── AAPL/
│   ├── trade.bin
│   ├── quote.bin
│   └── misc.bin
├── MSFT/
│   ├── trade.bin
│   └── misc.bin
```

After parsing:

```
/compressed/
├── AAPL/
│   ├── trade.zst
│   ├── quote.zst
│   └── misc.zst
├── MSFT/
│   ├── trade.zst
│   └── misc.zst
```

CS Fundamentals

- Multithreading and Concurrency. std::thread
- ifstream/ofstream and fread/fwrite for fast binary reads/writes. Avoid any conversions from bin to other formats
- Decoding Algorithms. Research on the best algos for our case. Ideally use and test all, then pick the best one.
- … more to come. (Optimize speed and space)

Module 1: PCAP Ingest Module

Goal

- Strip away all redundant headers. Go from pcap file to bin file (binary)

Considerations

- Headers might vary in size
- Need to research pcap formats through IEX resources
- Data processor must support files that are 100+ gb (Real pcap files)
- It needs to be fast. How -> don’t know yet

Steps

Open .pcap file in binary mode

Parse global header (if exists, I couldn’t find it)

For each packet:

  - Skip global + per-packet headers
  - Extract **Ethernet frame**
  - Validate protocol and other header data
  - Skip 42 (or necessary) bytes total
  - Write the remaining **payload** to a binary output file

Continue until EOF

Packages

- libpcap (if using library)
- Or any other packet reading libs (need to research)
- Or just c++ file I/O?
- C++ file I/O:
- ifstream + seekg() and read()
- Or fread() for buffered speed

Unknowns

- Do we validate checksum / header fields or just skip fixed bytes?
- What exactly do we store in payload files? Exact specifications.
- Where do we store redundant headers, and how? Needed for reconstruction
- Use a map if there’s multiple headers? Map[key, val] -> [headers: list of ALL pcaps associated with that header]
- Should we store timestamps with each payload for reassembly later?
- Will payload length be prepended to each entry in .bin (to allow re-parsing)?
- Need to benchmark: is libpcap slower than raw reads? Or other libraries