Protocol prototypes and some thoughts about software

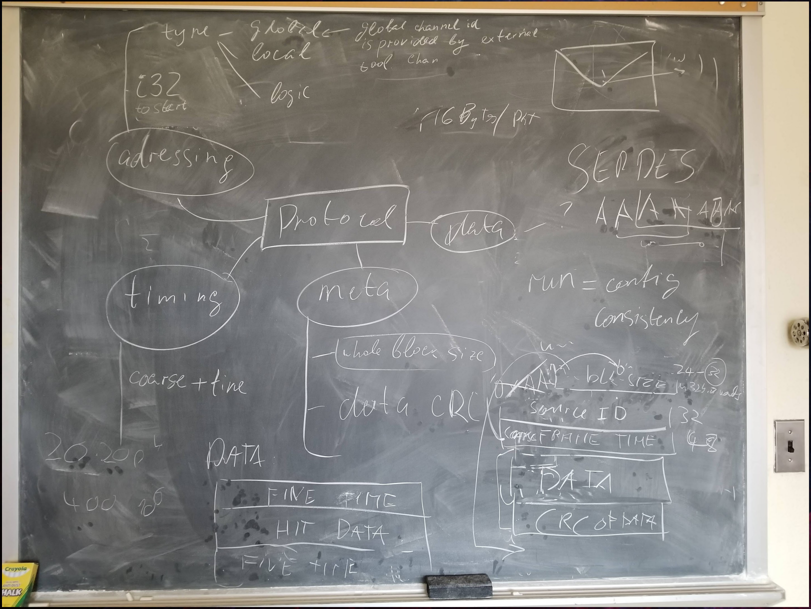
Jan C. Bernauer

SRC workshop IV, May 2019

RBRC RIKEN BNL Research Center
Stony Brook University
Massachusetts Institute of Technology

A first pass at a protocol

Early April, Dimitri, Markus and I met at BNL:



Structure and Rational

- A constant start byte (0xAA) to resynchronize
- 24 bit unsigned int for block size
- 32 bit unsigned int for channel id
- 48 bit unsigned int for coarse frame time
  - We think an (approximately) absolute time would be very helpful.
  - Assuming 32 bit fine times at ps resolution (=4ms), we can cover 38k years with 48 bit. Could reduce to 40 bit and have 8 bit flags?)
- 32 bit aligned semi-opaque payload
  - 32 bit unsigned int for fine time (ps level)
  - 32 bit aligned detector specific hit data.
- 32 bit unsigned int for checksum

Things I learned/thought about while implementing this

- It's way too long I have written C/C++ code. Needs to change.

Things I learned/thought about while implementing this

- It's way too long I have written C/C++ code. Needs to change.
- C bitfield layout is compiler implementation specific. Can be managed, but maybe as awkward as using bit shifts/masks...

Things I learned/thought about while implementing this

- It's way too long I have written C/C++ code. Needs to change.
- C bitfield layout is compiler implementation specific. Can be managed, but maybe as awkward as using bit shifts/masks...
- We might want to get rid of the checksum on-wire, but certainly not on-disk.

Things I learned/thought about while implementing this

- It's way too long I have written C/C++ code. Needs to change.
- C bitfield layout is compiler implementation specific. Can be managed, but maybe as awkward as using bit shifts/masks...
- We might want to get rid of the checksum on-wire, but certainly not on-disk.
- We need a sequence number so we can reorder efficiently.
  - 24 sequence number as first field.
  - Promote block size to 32 bit.

Things I learned/thought about while implementing this

- It's way too long I have written C/C++ code. Needs to change.
- C bitfield layout is compiler implementation specific. Can be managed, but maybe as awkward as using bit shifts/masks...
- We might want to get rid of the checksum on-wire, but certainly not on-disk.
- We need a sequence number so we can reorder efficiently.
  - 24 sequence number as first field.
  - Promote block size to 32 bit.
- We are only word aligned. Could add short for more flags / version / data pitch.

Things I learned/thought about while implementing this

- It's way too long I have written C/C++ code. Needs to change.
- C bitfield layout is compiler implementation specific. Can be managed, but maybe as awkward as using bit shifts/masks...
- We might want to get rid of the checksum on-wire, but certainly not on-disk.
- We need a sequence number so we can reorder efficiently.
    - 24 sequence number as first field.
    - Promote block size to 32 bit.
- We are only word aligned. Could add short for more flags / version / data pitch.
- At full frame size, we have an overhead of 5kByte/s per channel.

Large packets

- A frame can be up to 16 Megabytes.
- We might want to interleave on smaller time-scales / buffer depth

Large packets

- A frame can be up to 16 Megabytes.
- We might want to interleave on smaller time-scales / buffer depth
- Could do wrapping layer which chops up a stream into (fixed size?) packets. Similar to MPEG TS/PS.

Development at JLAB: Hardware

- (See Ben's talk)
- Work on FADC firmware for pulse extraction to forward to VTP (and then 10GbE)
- Work on VTP firmware to use DDR3 memory to buffer 128 Gbps (burst) input streams.
- C++ on FPGA implementation for TCP streaming (connect to zMQ)
- Some debug work on ethernet/TCP stack.

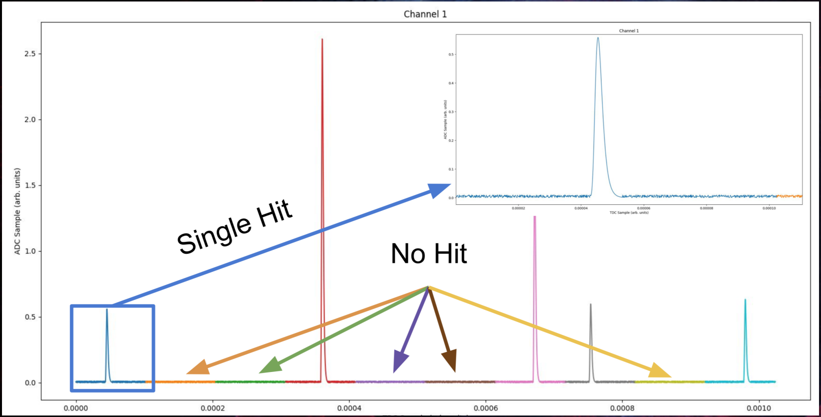
Development at JLAB: Software/Network

- Data source: Sends data from file/detector simulation over network with efficient packing.
- Streamer router: Receives data from source(s), measures statistics and re-broadcasts via ZeroMQ.
- Sink: Receives data from the Streamer router via ZeroMQ.

Data rates of up to 6 GBytes/s have been achieved in the INDRA lab. Streamed data from the VETROC TDC system.

Development at JLAB: Software/JANA

- Python simulation of ADC/TDC data from SAMPA (without zero suppression)
- Converter to JANA events (each event = one time window)



Next steps: Combine these tasks for end-to-end solution.

Message passing or streams?

- Natural streams, but work distribution etc. chops the streams up for parallel processing.

Message passing or streams?

- Natural streams, but work distribution etc. chops the streams up for parallel processing.
- A packet-oriented or message-passing interface suits this.
- ZeroMQ is big player, but others exists (nanomsg, nng).
- (Also interesting: libfabric, DPDK)

Things to check for zMQ/nanomsg

- Can we use one of these even for FEE?
- PIPELINE mode promises N:M work distribution! Check how well that works.
- How much overhead on the wire?
- Stable? Future proof?

Node configuration / naming of channels

- Nodes bring-up, especially FEE, needs basic configuration (network address, etc.)
- Probably will use DHCP/BOOTP/DNS for this.
- Since each node knows it's data formats, node should tell orchestrator. Maybe use DNS records (SRV,TXT)?
- Need mapping of channel name to channel-id. Can we auto-generate?
  - 32bit is probably enough to enumerate all channels in even a big system.
  - Could hash names to produce id.
  - Could autogenerate names of process nodes by hash(input channel names + operator name).
  - But with 32 bit, birthday problem limits to few k channels before hash collision.
  - Is that a problem? Do we want to go to 64 or 128 bits (IPv6) bits?