GRID
PROTECTION
ALLIANCE

STTP Compatible
Point on Wave Compression

IEEE 2664-2024 (STTP)

sttp IEEE 2664
Streaming Telemetry Transport Protocol

GRID PROTECTION ALLIANCE
BONNEVILLE POWER ADMINISTRATION
WASHINGTON STATE UNIVERSITY
Electric Power Group
SEL SCHWEITZER ENGINEERING LABORATORIES
SPP Southwest Power Pool
OG&E
TVA
SDGE
Dominion Energy
pjm
Utilicast
PEAKRELIABILITY assuring the wide area view
SOUTHERN CALIFORNIA EDISON
ercot
ISO new england
SOUTHERN COMPANY
EPRI ELECTRIC POWER RESEARCH INSTITUTE
USC University of Southern California
OSIsoft.
BRIDGE ENERGY GROUP
SPACE-TIME INSIGHT
PingThings
T&D Consulting Engineers
MEHTA TECH, INC.

Streaming Telemetry Transport Protocol

- US DOE Project
- Intrinsically reduces losses and latency compared to frame-based protocols
- Allows the safe co-mingling of phasor data with other operational data network traffic
- Detailed metadata exchanged as part of protocol
- Includes lossless compression to reduce bandwidth utilization
- Security-first design with strong authentication and option for encryption

STTP Compression Algorithm: TSCC

- IEEE 2664 Standard (STTP) includes a compression algorithm:
  - Time Series Special Compression (TSCC)
- Tuned for Synchrophasor Data and Streaming Data
- Algorithm uses multiple algorithms for different time-series elements, with special focus on “Value”:
  - ID
  - Time
  - Value (differential / 7-bit encoding / last result cache / zero handling)
  - Quality

TSSC Testing with Point of Wave

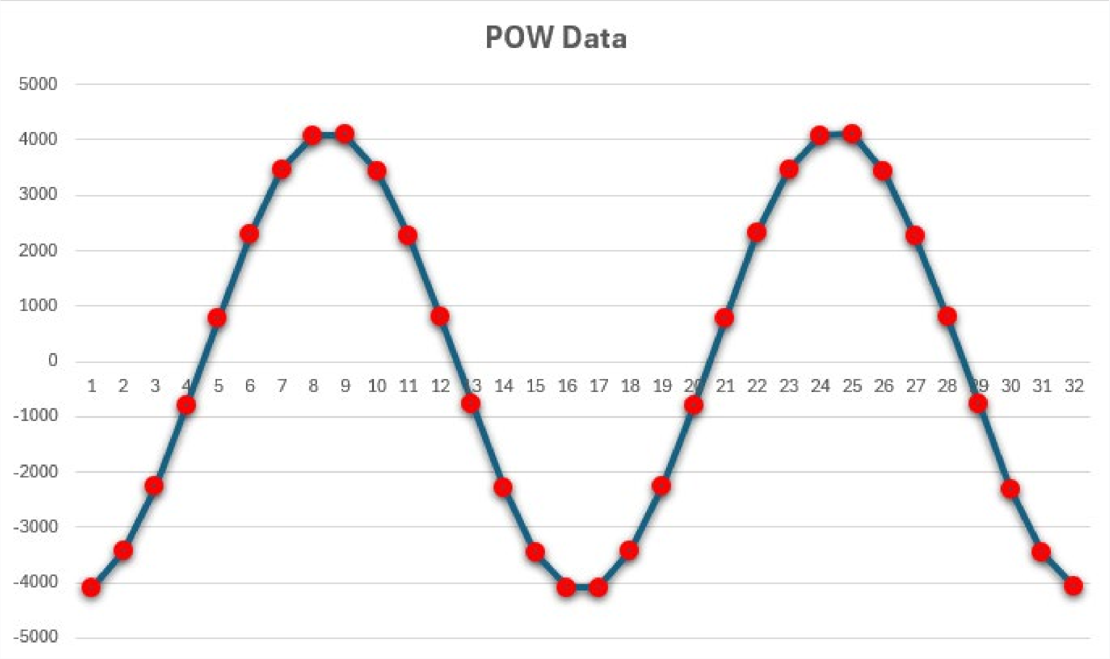
- During last NASPI meeting, a question came up about use of STTP TSSC for POW data
  - So, we collected some POW sample data and ran some tests
- Compression is very good for streaming phasor data
  - Low latency, low CPU impact, and fast
- Tests with streaming audio data also compressed well
  - Streaming signals at 44100 Hz data compressed well
- TSSC was expected to perform well with point on wave data…
  - Test data was recorded at 960Hz

It did not…

Why? Rate of Change

- TSSC performs well for data sets where there is a slow gradient of change:
  - This works well for phasor data (30/60Hz)
  - This works well for audio data (44100Hz)
- What makes 960Hz special?
  - Within 16 measurements, you move through 360 degrees →

WARNING: Curves Ahead!



Lots of experimentation ensued…

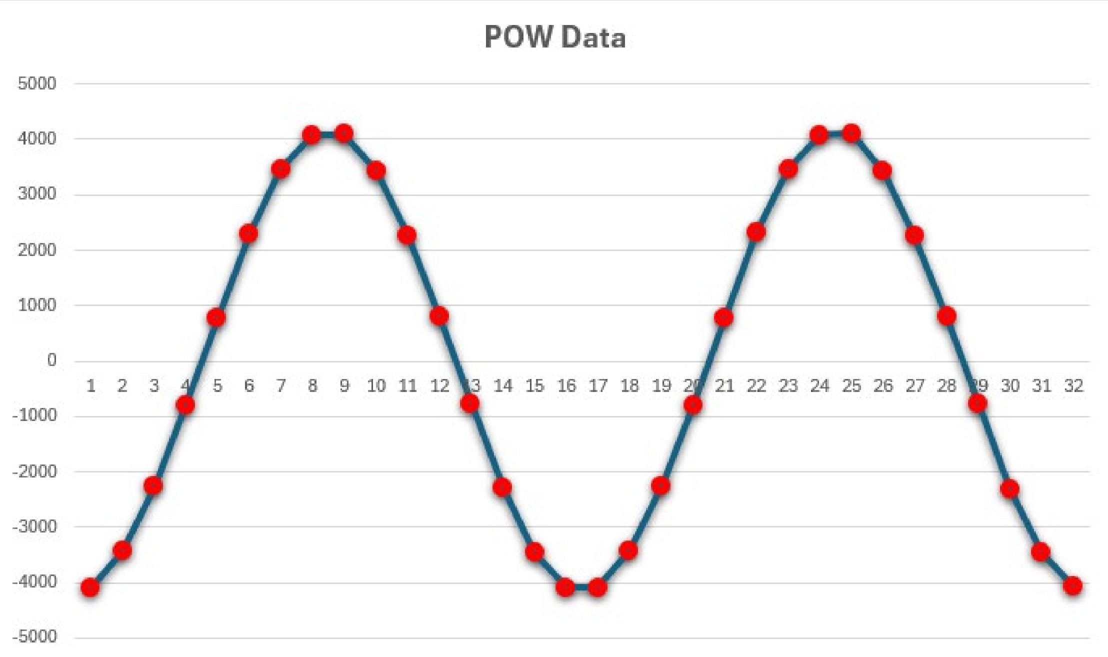
- After playing around with many compression techniques, you find that standard compression algorithms work fairly well.
- Of all the common players, LZMA (a.k.a. 7-zip) seemed to do the best job.
  - For a 5.6GB POW file representing a full day of data, 7-zip would reduce size by 65.6% (34.44% compression ratio)
- In terms of compression, we felt like this ratio could be improved, especially by understanding the sinusoidal nature of the data – something LZMA would not “assume”
- An idea: match the curve, producing small residuals, should be able to compress based on small values

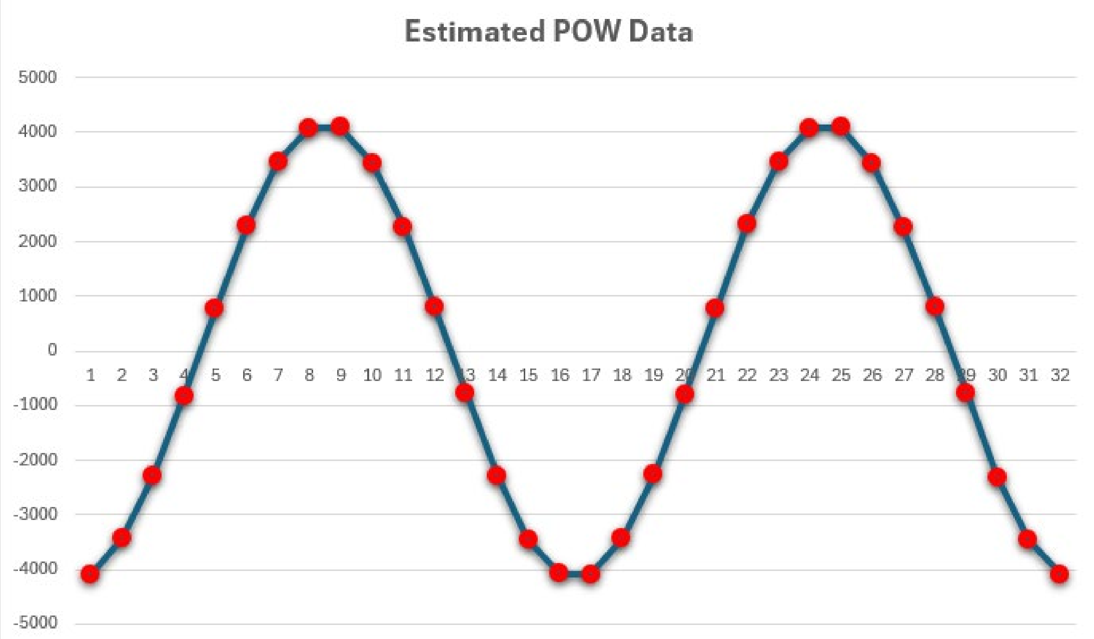
Trying to match the curve…

- Started with a goal of trying to emulate the source curve as close as possible, using Excel as a test bed
- Tried lots of frequency estimators with simple sine wave:
  - Zero crossing / FFT / and just assuming fixed 60hz
- For several sample files, narrowed in on the following solution
  - NOTE: *This was based on empirical work and intuition, not some mathematical hypothesis, which may produce better results*
- Emulating the POW curves with harmonic estimation, narrowing in on the 8th harmonic – simply because it produced the best match to original curve
  - For available data sources, anything higher or lower did not do as well

Getting as close as possible

-4076	-4080.34
-4088	-4092.92
-3436	-3436.72
-2274	-2285.47
-814	-822.08
770	761.8105
2306	2305.189
3468	3460.816
4082	4081.503
4097	4094.488
3437	3436.564
2257	2268.652
806	811.8743
-760	-759.954
-2302	-2301.64
-3469	-3468.84
-4087	-4082.08
-4093	-4090.85
-3423	-3430.22
-2260	-2268.38
-811	-814.821
779	771.1554
2314	2309.681
3471	3465.27
4076	4078.375
4103	4097.442
3421	3427.474
2254	2254.339
799	801.5861
-782	-771.874
-2315	-2309.46





Tiny residuals – *we can compress*!

Measured	Predicted	Residual
-4076	-4080.3429	4
-4088	-4092.9206	5
-3436	-3436.7209	1
-2274	-2285.4707	11
-814	-822.07975	8
770	761.810509	8
2306	2305.1895	1
3468	3460.81555	7
4082	4081.50316	0
4097	4094.48761	3
3437	3436.56381	0
2257	2268.65189	-12
806	811.874317	-6
-760	-759.95442	0
-2302	-2301.641	0
-3469	-3468.8422	0
-4087	-4082.0751	-5
-4093	-4090.8462	-2
-3423	-3430.219	7
-2260	-2268.3752	8
-811	-814.82135	4
779	771.155417	8
2314	2309.68149	4
3471	3465.26997	6
4076	4078.37499	-2
4103	4097.44182	6
3421	3427.47357	-6
2254	2254.33865	0
799	801.58615	-3
-782	-771.87386	-10
-2315	-2309.455	-6

- Very small residuals allows for interesting compression options
- For one, if most values are in the range of -8 to +8, then you can compress the value into 3-bits, saving a bit as a marker. So, 4-bits, i.e., a “nibble” – in other words, can fit two values into one byte
- From 3-bits, you can then move up to 7-bits and then 13-bits and finally, full value, with a marker
- In testing most values fell within either the 3-bit to 7-bit range, which meant good compression

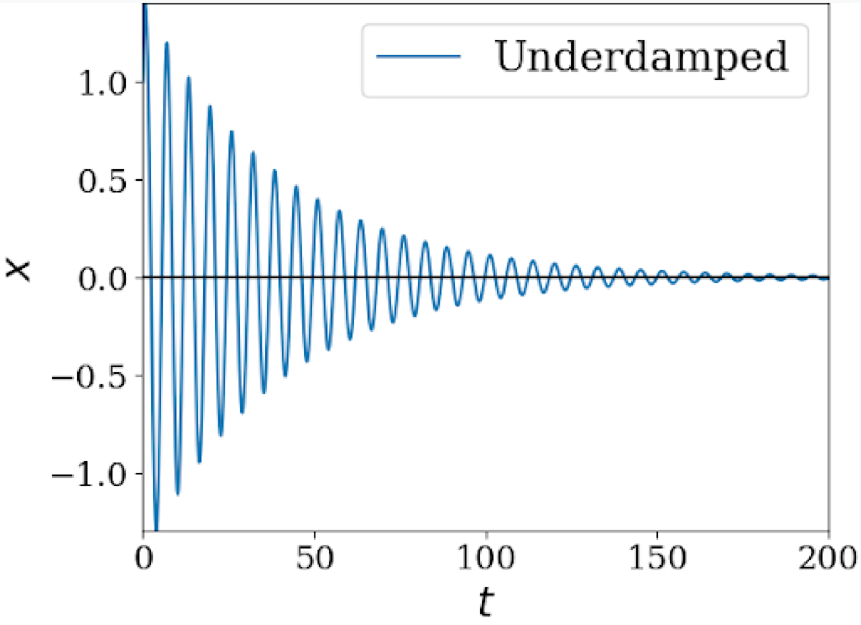
Harmonic Differential Compression

- For the current implementation, some default parameters (all configurable):
  - Harmonic count: 8
  - Supplemental compression algorithm: LZMA
  - Buffer size: 64K
  - Window size: 2 cycles
  - Frequency estimation: Fixed (options for zero crossing / FFT)
  - Target compression ratio: 26%
- Optimizations:
  - Caching of calculated omegas – reduces calls to trig functions
  - 3/7/13-bit encoding

Everything is peaches and cream…

Until it's not…

- Pretty, predictable curves aren't always so pretty
- Sometimes they get angry and noisy
- So, you need a "plan B" for compression in these cases
  - As previously tested, LZMA is a good "general choice" for compression
- When things don't compress well, e.g., less than a target of 26%, use a common compression algorithm, e.g., LZMA



Some results… *~25% compression ratio*

- *Note*: excludes results that used around 100% supplemental compression
- Smaller compression ratios values are better:

Example 1:
Encoded Size: 39.82 megabytes / 158 megabytes (**25.17%**)
Supplemental Compression: 1,028 / 1,265 (81.26%)

Example 2:
Encoded Size: 39.47 megabytes / 158 megabytes (**24.95%**)
Supplemental Compression: 319 / 1,265 (25.22%)

Example 3:
Encoded Size: 36.25 megabytes / 158 megabytes (**22.91%**)
Supplemental Compression: 436 / 1,265 (34.47%)

Example 4:
Encoded Size: 40.01 megabytes / 158 megabytes (**25.29%**)
Supplemental Compression: 0 / 1,265 (0.00%)

Example 5:
Encoded Size: 40.21 megabytes / 158 megabytes (**25.42%**)
Supplemental Compression: 0 / 1,265 (0.00%)

Example 6:
Encoded Size: 40 megabytes / 158 megabytes (**25.29%**)
Supplemental Compression: 598 / 1,265 (47.27%)

Example 7:
Encoded Size: 39.76 megabytes / 158 megabytes (**25.13%**)
Supplemental Compression: 518 / 1,265 (40.95%)

Example 8:
Encoded Size: 39.58 megabytes / 158 megabytes (**25.02%**)
Supplemental Compression: 0 / 1,265 (0.00%)

Example 9:
Encoded Size: 40.36 megabytes / 158 megabytes (**25.51%**)
Supplemental Compression: 0 / 1,265 (0.00%)

Example 10:
Encoded Size: 40.13 megabytes / 158 megabytes (**25.37%**)
Supplemental Compression: 0 / 1,265 (0.00%)

Conclusions

- For a sinusoidal inputs, results were better than LZMA alone
- For wave forms that don’t “fit”, LZMA produced better results
- The current implementation operates by using both, again, when ratio is less than (configurable) 26%, use LZMA
- ***Some math may go a long way at producing better results!***

- ***Pros***:
  - Good compression, ~25%
  - Suitable for streaming compression, e.g., STTP
  - Reduces bandwidth for streaming and file transfers in reduced bandwidth environments, e.g., sub-station

- ***Cons***:
  - CPU costs are high – lots of calculation required – so better suited for single value streams
  - More compression would be better, more work to be done on improving algorithm results