GRID PROTECTION ALLIANCE

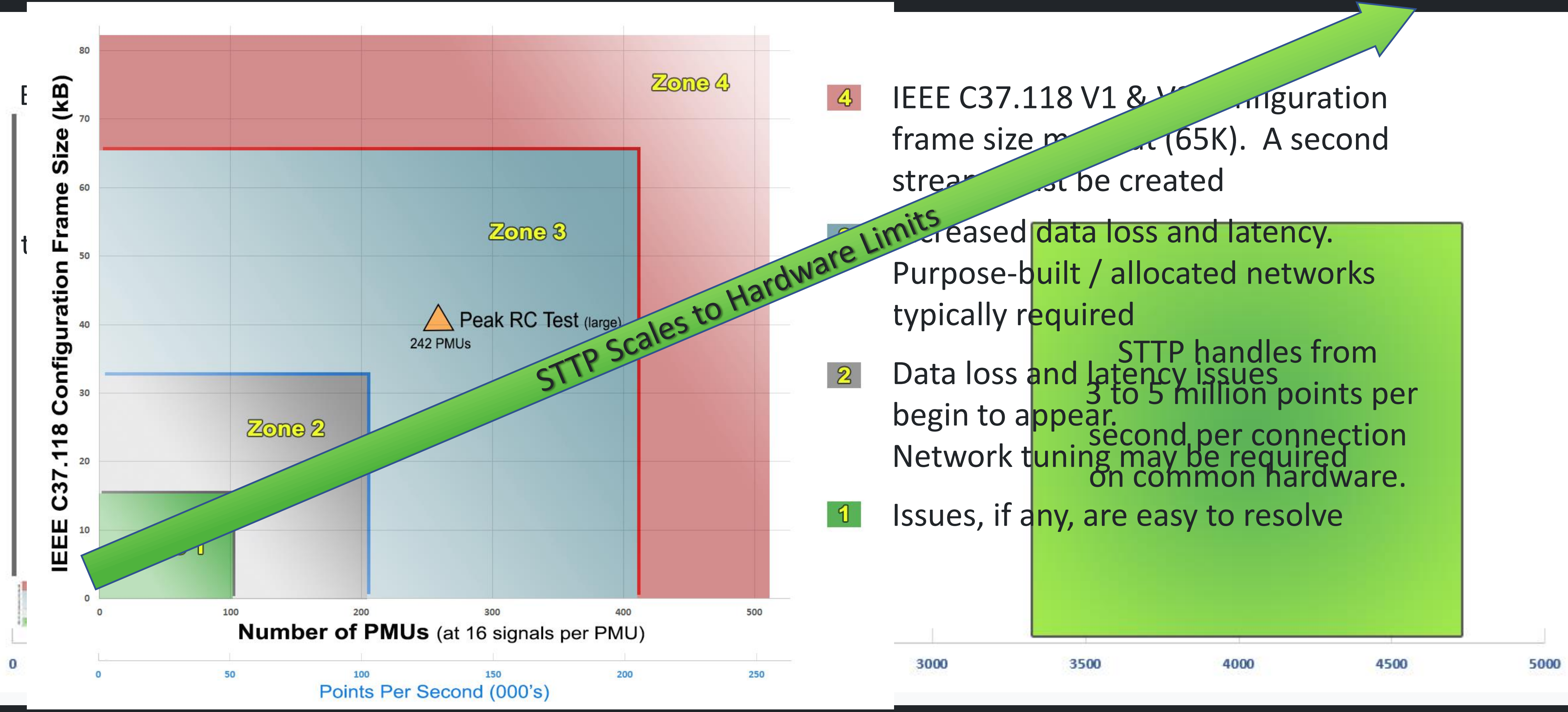
Connectivity and Performance Updates

IEEE 2664-2024 (STTP)

Streaming Telemetry Transport Protocol

- US DOE Project Funding (SIEGate / ASP)
- 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 Difference: Scalability



4 IEEE C37.118 V1 & V2 configuration frame size max limit (65K). A second stream must be created

3 Increased data loss and latency. Purpose-built / allocated networks typically required

2 Data loss and latency issues begin to appear. Network tuning may be required

1 Issues, if any, are easy to resolve

STTP Open-Source Implementations

New Features!

https://github.com/sttp

Streaming Telemetry Transport Protocol

python

Python STTP Implementation
https://github.com/sttp/pyapi

GO

Go STTP Implementation
https://github.com/sttp/goapi

.NET

.NET STTP Implementation
https://github.com/sttp/dotnetapi

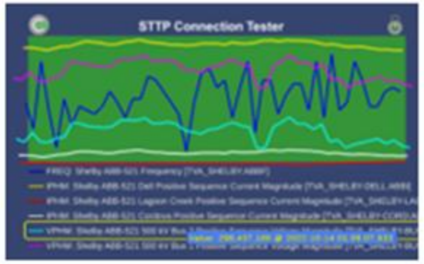
C++ STTP Implementation
https://github.com/sttp/cppapi

sttp Connection Tester

STTP Connection Tester
https://github.com/sttp/connection-tester

Open Source

All STTP reference implementations are Open Source Software (OSS) published on GitHub under the permissive MIT license.



Work continues to make STTP implementations 100% compatible with IEEE 2664-2024

New Python Code: Concentration

pyapi/examples/groupeddatasubscribe/main.py at main · sttp/pyapi

- Grouped Data Subscribe
  - For a collection of incoming signals, applies time alignment operations providing synchronized data to a consuming algorithm
- Allows custom Python algorithms to operate on time-aligned groups of data received over STTP
  - This allows grouping of data by timestamp in Python code
  - Also known as *data concentration*
- STTP does not time align data natively; it sends data as it is received to speed delivery
  - This leaves concentration function to the consuming application

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

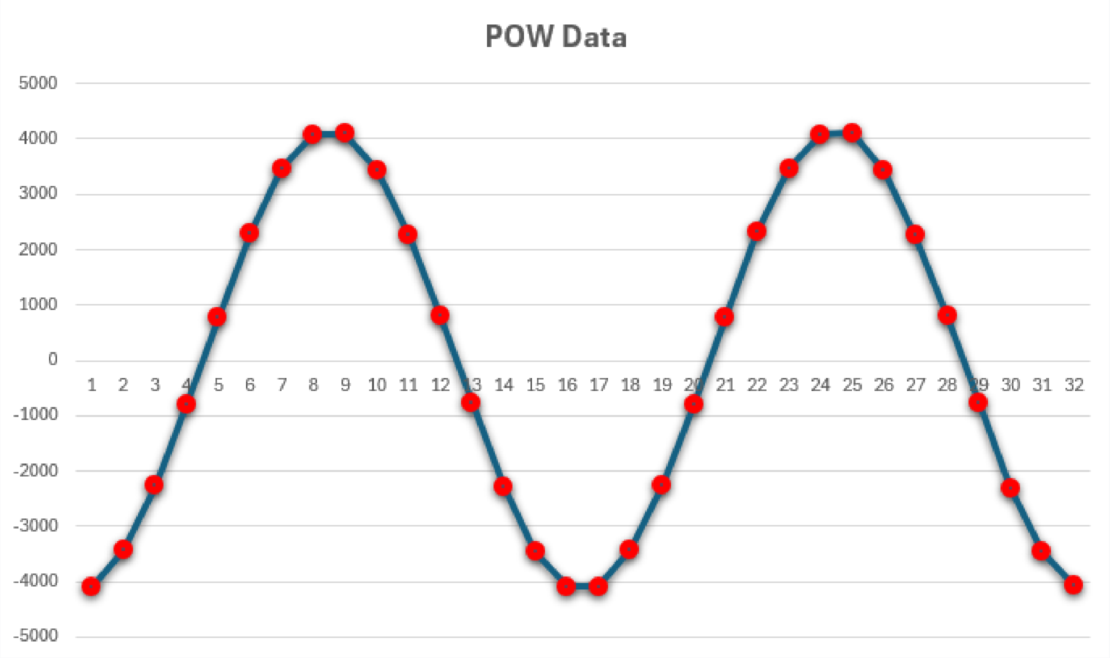
TSSC Testing with Point of Wave

- 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 does not perform as well with 960Hz point on wave data
  - Multiple channels of test data were recorded at 960Hz

TSSC Best for Slow 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 →

Non-linear Data



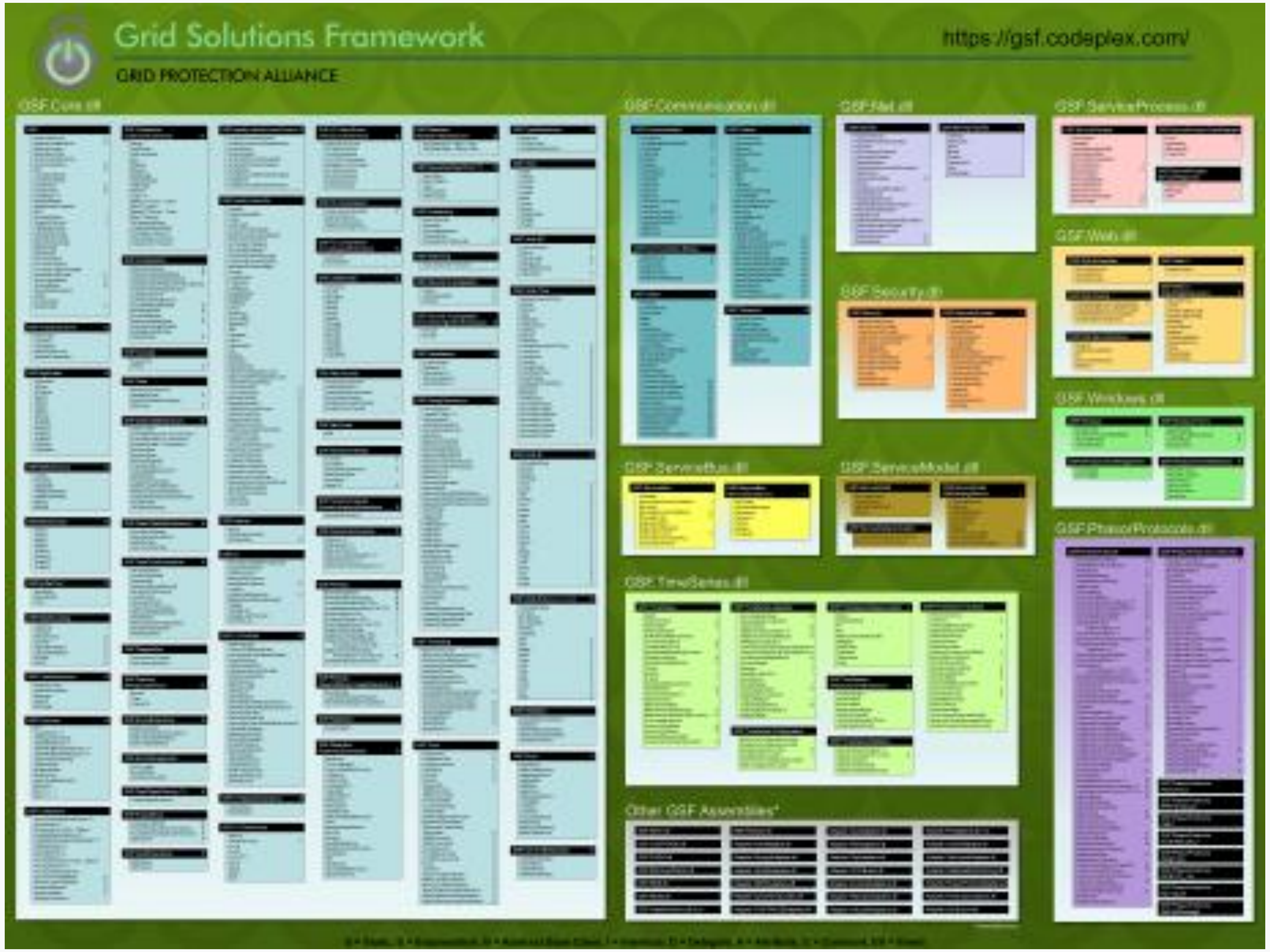
New STTP Compatible Algorithm: Harmonic Differential Compression

- **~25%** compression ratio (i.e., **75%** reduction in original size)
- 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

Other Protocols

- ICCP
  - Sending and receiving SCADA data into GPA Tools
  - New adapters available in EE versions
  - Testing new configuration scripts (Python)
- DNP-3
  - Updated configuration scripts (Python)
  - Existing adapter → Input
  - Under development → Output
- IEEE C37.118.2-2024
  - GPA participated in standard process, recently published
  - Will be implementing new protocol soon

Grid Solutions Framework



Analytics
- Alarming Module
- Bad Data Detection Core
- Linear State Estimation (NEW)
- Numerical Analysis Extensions
- SI Unit Primitives

Data Storage
- Data Historian
- Database Abstraction
- Metrics Historian

Core Services
- Adapter Framework
- Async Data Processing
- Bit Manipulations
- Installation Framework
- Interprocess Synchronization
- Native Core Extensions
- Object Pooling
- Security Subsystems
- System Services Framework
- Time Series Framework
- UI Base Services
- XML Extensions

Input/Output
- Byte Encoding
- Checksum Validation
- Communications Library
- Data Encryption
- Phasor Data Stream Parsing
- Pub/Sub Framework
- Stream Management
- String Manipulation
- Utility Protocol Parsing

System Management
- Configuration Management
- Error Management
- Performance Monitoring
- System Event Logging
- System Management
- Thread Management

GRID PROTECTION ALLIANCE

From GSF to Gemstone

Grid Solutions Framework → GPA Gemstone .NET LIBRARIES

- Migrating code from .NET Framework to .NET 8 (a.k.a. Core)
  - Makes code natively cross platform, supporting:
    - Windows
    - Linux
    - OS-X
  - Impressive performance boost!
    - ASP.NET UI testing boasts near 40% improvements many some cases
    - Allows ready-to-run native executable deployments
  - Complete backend code overhaul
    - New service and security architectures
    - Adding nullable language checks to code for improved safety

Ongoing work with New Versions

- Based on new cross-platform openPDC, running on Linux natively, allows unique deployments
- New version runs on existing substation hardware with tiny hardware constraints
- Allows reuse of existing hardware for new purposes:
  - Local storage, allowing data gap filling for com losses
  - Application of new protocols (STTP) to reduce bandwidth
  - Deployment of distributed calculations, e.g.:
    - Power
    - Sequence Calculations
    - Oscillation Detection

New openHistorian Performance Test Results

- Extraction testing using the C# socket based OH API writing data to a CSV
  - Extracted CSV data size: 7.17GB
- Query Time Reduction
  - Old Query Time: 90.02 seconds
  - New Query Time: 69.70 seconds
  - *Time Reduction*: **20.32 sec** (≈22.58%)
- Throughput Improvement
  - Old Extracted Points Per Second: 3,527,216.17
  - New Extracted Points Per Second: 4,555,523.67
  - *Throughput Increase*: **1,028,307.5 points per second** (≈29.15%)

Total Points:	2.00			
Point Freq:	44,100.00	Hz		
Points Per Sec:	88,200.00			
Time Range:	3,600.00	Seconds	(1 Hr)	
Extracted Points:	317,520,000.00			Extracted Points Per Sec:
Old Query Time:	90.02	Seconds		3,527,216.17
New Query Time:	69.70	Seconds		4,555,523.67

Performance Results Summary

- The transition to the new openHistorian system, querying the exact same data / time range, results in:
  - Significant reduction in query execution time (approx. 22.6%)
  - Notable increase in data processing throughput (approx. 29%)

This improvement can be classified as a major performance enhancement for openHistorian, yielding faster computations and more efficient data handling capabilities which is crucial for high-frequency data processing.