Version: 0.0.33 - July 7, 2017

Status: Initial Development

Abstract: This specification defines a publish-subscribe data transfer protocol that has been optimized for exchanging streaming time series style data, such as synchrophasor data that is used in the electric power industry, over Internet Protocol (IP). The protocol supports transferring both real-time and historical time series data at full or down-sampled resolutions. Protocol benefits are realized at scale when multiplexing very large numbers of time series data points at high speed, such as, hundreds of times per second per data point.

Disclaimer

License

Table of Contents

Introduction

Use of synchrophasors by U.S. utilities continues to grow following the jump start provided by the Smart Grid Investment Grants. Even so, the dominant method to exchange synchrophasor data remains the IEEE C37.118 [1] protocol that was designed for and continues to be the preferred solution for substation-to-control room communications. It achieves its advantages through use of an ordered set (a frame) of information that is associated with a specific measurement time. When IEEE C37.118 is used for PDC-to-PDC communication or for PDC-to-Application communication, large data frames are typically distributed to multiple systems. To address the challenges presented by these large frame sizes, many utilities implement purpose-built networks for synchrophasor data only. Even with these purpose-built networks, large frame sizes result in an increased probability of UDP frame loss, or in the case of TCP, increased communication latency. In addition, IEEE C37.118 has only prescriptive methods for the management of measurement metadata which is well-suited for substation-to-control-center use but which becomes difficult to manage as this metadata spans analytic solutions and is used by multiple configuration owners in a wide-area context.

To address these issues, the Advanced Synchrophasor Protocol (ASP) Project was proposed to DOE in response to FOA-1492. In this project proposal, the argument was made for a new protocol that overcomes the limitations of IEEE C37.118 for large-scale synchrophasor data system deployments. The new protocol proposed leveraged the successful design elements of the secure Gateway Exchange Protocol (GEP) that was originally developed by the Grid Protection Alliance (GPA) as part of the SIEGate project (DE-OE-536).

On May 1, 2017, a DOE grant (DE-OE-859) was awarded to GPA and the other 25 collaborators on ASP Project (see Contributors section) to develop: (1) a detailed definition of new publish-subscribe protocol, now called the Streaming Time-series Transport Protocol (STTP) and (2) software to support it including production-grade implementations of STTP API's in multiple development platforms along with a collection of tools to test and validate the new protocol.

Scope of this Document

The purpose of this document is to define STTP and to include, as appendices, descriptions as to how to use its supporting software tools. This STTP specification is focused on effective "streaming data" delivery of which synchrophasor data is a very important use case.

In the Overview section of this specification, high-level features and the business value of STTP are presented. The balance of the sections of the specification provide the details of protocol design.

Appendix A - STTP API Reference provides instructions to enable software developers to integrate and use of STTP within other software systems.

Appendix B - IEEE C37.118 Mapping provides a detailed look at the process of transforming IEEE C37.118 into STTP as well as creating IEEE C37.118 streams from STTP.

While the format and structure of this document, established to facilitate collaboration, is different than that used by standards bodies, it is hoped that the content within this document can meet all the information requirements needed to enable repackaging of this specification into draft standard formats.

Business case

At the conclusion of the STTP project in April 2019, STTP will be a well-tested, thoroughly vetted, production-grade protocol that will be supported by project team vendors. An open source tool suite for STTP will be developed as part of the project (see Appendix A) that will include a test harness that will allow utilities and vendors outside the project to test and validate STTP in their systems and API's.

STTP offers both short-term cost savings and strategic value in that it is:

Intrinsically Robust

By design, STTP packet sizes are small and are optimized for network MTU size reducing fragmentation which results in more efficient TCP performance and less overall data loss with UDP. STTP also puts significantly less stress on network routing equipment and facilitates mixing of streaming data traffic and other general network communications. With STTP, purpose built networks are not required to reliably support very large phasor data streams.

Security Centric

STTP has been built using a "security first" design approach. Authentication to establish a connection with other parties requires a certificate. While public certificate providers can be used, it is recommended that symmetric certificates be exchanged out-of-band to avoid the risk and cost of management of public keys. Best-practice encryption is natively available in STTP but not required given the common practice to manage encryption at the network layer.

Reduces First Cost

GEP has been measured [5] to have less than half the band width requirements of IEEE C37.118 [1] when used with TCP and simple methods for lossless compression. With the compression, a single signal or measurement point (i.e., an identifier, timestamp, value and quality code) requires only 2.5 bytes. By comparison, IEEE C37.118 requires 4.5 bytes per measurement on average. The signal-based GEP protocol incorporates Pub/Sub data exchange methods so that unnecessary data points need not be exchanged â€" thereby further reducing overall bandwidth requirements as compared to IEEE C37.118.

Reduces Operating Cost

GEP automatically exchanges and synchronizes measurement level meta-data using a GUID as the key value to allow the self-initialization and integration of rich meta-data with points from multiple connected synchrophasor networks. This eliminates the need to map measurements to a pre-defined set identifiers and dispenses with the cost and hassles of synchronization of individual utility configuration with a centralized registry. Permissions for data subscriptions can be grouped and filtered using expressions to assure that only the signals that are authorized are shared (for example, all phasors from a specified substation) while the set of points available is dynamically adjusted as PMUs come and go without the need for point-by-point administrator approval.

An Enabling Technology

STTP provides an alternative to the existing method for utility data exchange that will enable future generations of SCADA/EMS systems to both (1) utilize full-resolution synchrophasor data streams and (2) significantly reduce the cost of maintaining the configuration of components to exchange other real-time data. An ISO/RTO will typically exchange hundreds of thousands of data points every few seconds with its members and neighbors.

> ℹ️ ICCP (IEC 60870-6/TASE.2) is the international standard used to exchange "real-time" SCADA data among electric utilities. Analog measurement data is typically exchanged continuously every 2 to 10 seconds

> with bi-modal data such as breaker status information only being exchanged "on change". ICCP came into coordinated use in North America in the mid-1990s.

Promising technologies are being developed for cloud computing and these technologies are moving toward native implementations at individual utilities and ISOs -- and can be leveraged to support larger native implementations such as those to support an interconnect. The common theme among these technologies is the ability to process significantly more data very quickly with improved reliability.

It's possible that a protocol like STTP which allows secure, low-latency, high-volume data exchange among utilities at low cost can be a major factor in driving change toward these new technologies. New higher-speed forms of inter-utility interaction will be possible, and new approaches for providing utility information services will be realizable.

Built Upon A Proven Approach

STTP will enhance the successful design elements of the Gateway Exchange Protocol (GEP) as a foundation and improve upon it. GEP is currently in production use by Dominion, Entergy, MISO, PeakRC, TVA, FP&L, Southern Company, among others.

Definitions and Nomenclature

🚧 Please add liberally to this section as terms are introduced in the spec

Definition of Key Terms

The words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are to be interpreted as described in RFC 2119 [3].

ℹ️ All the terms below are hyperlinked to a key source for the definition or to a reference where more information is available.

Term	Definition
data point	A measurement of identified data along with any associated state, e.g., time of measurement and quality of measured data.
data structure	An organized set of primitive data types where each element has a meaningful name.
frame	A data-structure composed of primitive data types that has been serialized into a discrete binary package.
endianess	The hardware prescribed ordinal direction of the bits used to represent a numerical value in computer memory; usually noted as either *big* or *little*.
Ethernet	Frame based data transmission technology used in local area networks.
fragmentation	Network fragmentation is the process that breaks frames into smaller fragments, called packets, that can pass over a network according to an MTU size limit. Fragments are reassembled by the receiver.
measurement	
packet	A block of data carried by a network whose size is dictated by the MTU. Also called *network packet*.
phasor	A complex equivalent of a simple cosine wave quantity such that the complex modulus is the cosine wave amplitude and the complex angle (in polar form) is the cosine wave phase angle.
primitive type	A specific type of data provided by a programming language referenced by a keyword that represents the most basic unit of data storage - examples can include integer, float and boolean values. Also called *primitive data type*.
publish/subscribe	A messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead characterize published messages into classes without knowledge of which subscribers, if any, there may be.
null	A value reserved for indicating that a reference, e.g., a pointer, is not initialized and does not refer to a valid object.
serialization	Process of transforming data structures into a format that is suitable for storage or transmission over a network.
signal	
synchrophasor	A phasor calculated from data samples using a standard time signal as the reference for the measurement. Synchronized phasors from remote sites have a defined common phase relationship.

time series	A series of data points indexed in time order, most commonly measured as a sequence taken at successive equally spaced points in time.
term	definition

Acronyms

Term	Definition
API	Application Program Interface
BES	Bulk Electric System
DOE	United States Department of Energy
DDS	Data Distribution Service
GEP	Gateway Exchange Protocol
GPA	Grid Protection Alliance, Inc.
GPS	Global Positioning System
GUID	Globally Unique Identifer
ICCP	Inter-Control Center Communications Protocol
IP	Internet Protocol
ISO	Independent System Operator
MTU	Maximum Transmission Unit
NaN	Not a Number
PDC	Phasor Data Concentrator
PMU	Phasor Measurement Unit
STTP	Streaming Telemetry Transport Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
ZeroMQ	Brokerless Messaging Queuing and Distribution Library

Document Conventions

Markdown notes in combination with the Github Emogi images are used as callouts. The standard callouts are:

ℹ️ This is a call out in the spec to provide background, instruction or additional information

⚠️ This note use used to highlight important or critical information.

🚧 A informal note to document authors to facilitate specification development

🍅❓ (author's initials): May be used by anyone to toss out questions and comments that are temporal. These may be inserted at any point in any of the markdown documents. These questions will preserved as they are migrated to the QuestionsSummary.md file from time-to-time.

Code blocks are shown as:

```
public function void DisplayHelloWorld()
{
    Console.WriteLine("Hello world!");
}
```

Code is also shown `inline` as well.

Protocol Overview

In typical messaging exchange paradigms a source application hosts a block of structured data, composed in memory, with the intent to transmit the data to one or more receiving applications. The data has *structure* in the sense that it exists as a collection of simpler primitive data types where each of the data elements is given a name to provide useful context and meaning; most programming languages represent data structures using a primary key word, e.g., `class` or `struct`. Before transmission, the data structure must be serialized - this is necessary because the programming language of the source application which hosts the data structure defines the structure in memory using a format that is optimized for use in the application. The process of serializing the data structure causes each of the data elements to be translated into a format that is easily transmitted over a network and is suitable for deserialization by a receiving application.

The applications that are sending and receiving data structures can be running on the same machine or on different physical hardware with disparate operating systems. As a result, the details of the data structure serialization format can be complex and diverse. Such complexities can include issues with proper handling of the endianess of the primitive data types during serialization which may differ from the system that is deserializing the data, or differences in the interpretation of how character data is encoded [6].

The subject of serializing data structures in the field of computer science has become very mature; many solutions exist to manage the complexities of serialization. Today most computer programming languages, or their associated frameworks, include various options for serializing data structures in multiple formats. However, these solutions tend to only work within their target ecosystems and are usually not very interoperable with other frameworks or languages.

When interoperability is important, other technologies exist that focus on data structure serialization that works regardless of hardware, operating system or programming language. Two of these serialization technologies that are in wide use are Google Protocol Buffers [7] and the Facebook developed Apache Thrift [8]. Both of these serialization frameworks create highly compact, cross-platform serializations of data structures with APIs that exist in many commonly used programming languages.

For smaller sized, discrete data structures, the existing available serialization technologies are very fast and highly effective. However, as the data structures become larger, the process of serialization and deserialization becomes more costly in terms of both memory allocation and computational processing. Because of this, large frames of data are not recommended for use by these serialization technologies [9] [10]. Additionally, and perhaps more importantly, there are also penalties that occur at the network transport layer.

> ℹ️ For the purposes of this specification, serialized data structures will be referred to as a *frames*, regardless of the actual binary format.

Data Structure Serialization in the Power Industry

In the electric power industry, the IEEE C37.118 [1] protocol exists as a standard serialization format for the exchange of synchrophasor data. Synchrophasor data is typically measured with an accurate time source, e.g., a GPS clock, and transmitted at high-speed data rates, up to 120 frames per second. Measured data sent by this protocol is still simply a frame of serialized primitive types which includes data elements such as a timestamp, status flags, phasor angle / magnitude pairs, etc. The IEEE C37.118 protocol also prescribes the combination of data frames received from multiple source devices for the same timestamp into one large combined frame in a process known as concentration. The concentration process demands that a waiting period be established to make sure all the expected data frames for a given timestamp arrive. If any frames of data do not arrive before the waiting period expires, the overall combined frame is published anyway. Since the frame format is fixed, empty data elements that have no defined value, e.g., NaN or null, still occupy space for the missing frames.

Large Frame Network Impact

In terms of Internet Protocol (IP), all frames of data to be transmitted that exceed the negotiated maximum transmission unit (MTU) size (typically 1,500 bytes for Ethernet networks [11]) are divided into multiple fragments where each fragment is called a network packet.
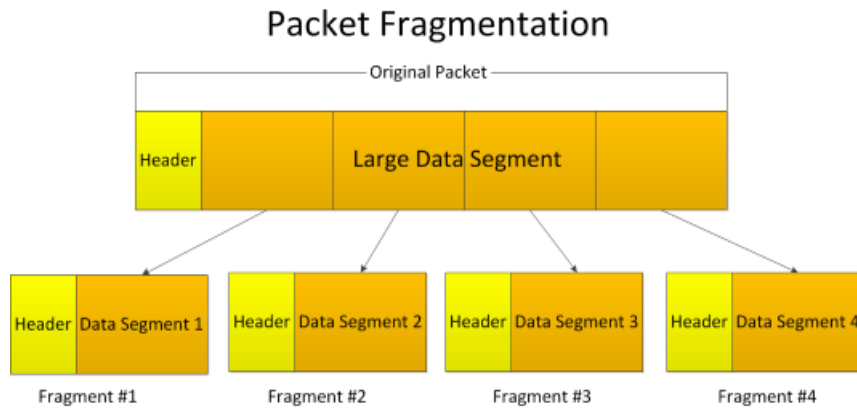
Packet Fragmentation

Figure 1

The impacts of large frames on an IP network are determined by the number of network packets required to send the frame and the fact that IP is inherently unreliable by design. Network packets can only be transmitted over a connection one packet at a time; when two or more network packets arrive for transmission at the same time on any physical network media, the result is a collision. When a collision occurs, only one packet gets sent and the others get dropped [12]. IP defines a variety of different transport protocols for network packet transmission, each of which behave in different manners when dealing with packet loss. Consequently, many of the impacts a large frame has on an IP network is dependent upon the transport protocol used to send the frame.

Large Frame Impacts on TCP/IP

The most common Internet protocol, TCP/IP, creates an index for each of the network packets being sent for a frame of data and verifies that each are successfully delivered, retransmitting packets as many times as needed in the case of loss. This functionality is the basis for TCP being considered a *reliable* data transmission protocol.

Since each packet of data for the transmitted frame is sequentially ordered, TCP is able to fully reconstruct and deliver the original frame once all the packets have arrived. However, for very large frames of data this causes TCP to suffer from the same kinds impacts on memory allocation and computational burden as the aforementioned serialization technologies, i.e., Protocol Buffers and Thrift. The unique distinction for IP based protocols is that at some level, these issues also affect every element of the interconnected network infrastructure between the source and sync of the data being exchanged.

Another critical impact that is unique to TCP is that for data that needs to be delivered in a timely fashion, retransmissions of dropped frames can also cause cumulative time delays [13], especially as large data frames are published at rapid frame rates. Time delays are also exacerbated during periods of increased network activity which induces congestion and a higher rate of collisions.

> ℹ️ Synchrophasor data is the source for real-time visualization and analysis tools which are used to operate the bulk electric system (BES). This real-time data is required to be accurate, dependable and timely in order to be useful for grid operators [14]. Any delays in the delivery of this data could have adverse affects on operational decisions impacting the BES.

Large Frame Impacts on UDP/IP

Another common Internet protocol is UDP/IP. Transmission of data over UDP differs from TCP in the fact that UDP does not attempt to retransmit data nor does it make any attempts to maintain the order of the transmitted packets. This functionality is the basis for UDP being considered a *lossy* data transmission protocol, but more lightweight than TCP.

Even with the unreliable delivery caveats, UDP will still attempt to reconstruct and deliver the originally transmitted frame of data. However, even if a single network packet is dropped, the entire original frame will be lost and any packets that were already accumulated get discarded [15]. In other words, there are no partial frame deliveries - frame reception with UDP is an all or nothing operation.

Since UDP attempts frame reconstruction with the received packets, the impact of very large frames of data with UDP are similar to those with TCP and serialization technologies in that there is increased memory allocation and computational processing throughout the network infrastructure.

The more problematic impact with UDP and large frames of data is that the increased number of network packets needed to send a large frame also increases the probability of dropping one of those packets due to a collision. Since the loss of any one packet results in the loss of the entire frame of data, as frame size increases, so does volume of data loss.

Impacts of UDP Loss on Synchrophasor Data

For synchrophasor data, UDP is often the protocol of choice. The density of synchrophasor data allows analytical applications to tolerate *some* loss. The amount of loss that can be tolerated depends on the nature of the analytic because as the loss increases, the confidence in the analytic results decreases [citation needed]. Another reason UDP is used for synchrophasor data is its lightweight nature; use of UDP reduces overall network bandwidth requirements as compared to TCP [16]. Perhaps the most critical reason for use of UDP for synchrophasor data is that UDP does not suffer from issues with induced time delays caused by retransmission of dropped network packets.

For IEEE C37.118 [1] deployments, large frame sizes can have adverse affects on data completeness; as more and more devices are concentrated into a single frame of data, the larger frame sizes contribute to higher overall data losses. In tests conducted by PeakRC, measured overall data loss for the transmission of all of its synchrophasor data using IEEE C37.118 averaged over 2% [5] when using a data rate of 30 frames per second and more than 3,100 data values per frame. To help mitigate the data losses when using UDP, some companies have resorted to purpose-built, dedicated synchrophasor networks [17]. Although a dedicated network is ideal at reducing data loss (minimizing simultaneous network traffic results in fewer collisions), this will not be an option for most companies that treat the network as a shared resource.

Changing the Paradigm with STTP

Existing serialization technologies are not designed for messaging exchange use cases that demand sending large frames of data at high speeds, often falling short in terms of timely delivery or data loss depending on the IP transport protocol used. The obvious solution is to break large data structures into smaller ones, recombining them as needed in receiving applications [9]. Although this strategy can work fine for one-off solutions where data structures are manually partitioned into smaller units for transport, this does not lend itself to an abstract, versatile long term solution.

Instead of serializing an entire data structure as a unit, STTP is designed to package each of the distinct elements of the data structure into small groups. Serialization is managed for each data element, typically a primitive type, that gets individually identified along with any associated state, e.g., time and/or quality information. Ultimately more information is being sent, but it is being packaged differently.

ⓘ For the purposes of this specification a data element, its identification and any associated state, e.g., time and quality, will be referred to as a *data point*.

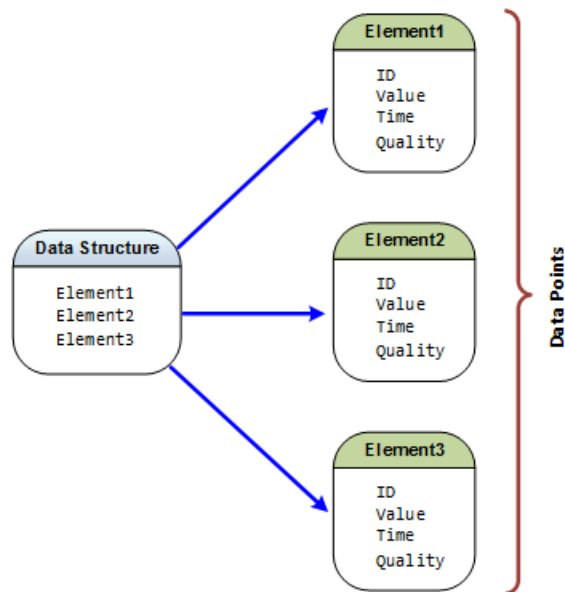**Mapping Data Structure Elements to Data Points**

Figure 2

To resolve issues with large frame impacts on IP based networks, a primary tenet of the STTP design strategy is to reduce fragmentation; as a result, STTP intentionally limits the number of data points that will be grouped together to form a frame to ensure its size is optimized for transmission over an IP network with minimal fragmentation.

Because each data point is uniquely identified, the elements that appear from one frame to another are not fixed allowing interleaving of data from multiple simultaneous data exchanges - this notion supports the delivery of any number of data structures where each can have a different publication interval.
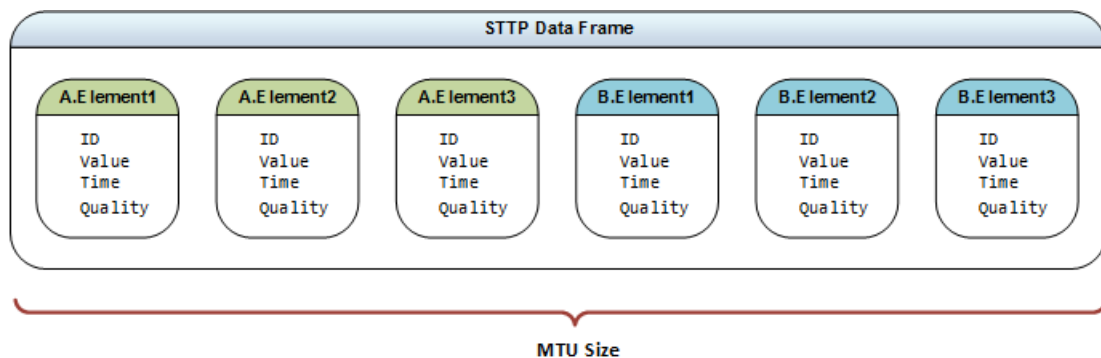
Figure 3

⚠ While it is possible to always target zero fragmentation by making sure the frame size is below the current MTU size, STTP implementations should allow tuning for some fragmentation to accommodate different deployment scenarios and use cases, i.e., allowing target frame sizes that are larger than the current MTU size. For deployments in high-performance network environments, overall loss due to data collisions may be statistically the same for frame sizes that are a few multiples of the MTU.

STTP Bandwidth Impact

Since data points include identity and state along with the primitive type value, serializations of STTP data carry extra information; so by its very nature uncompressed STTP will often require more bandwidth as compared to

traditional data structure serialization technologies.

Although it will be common for use cases that demand a protocol like STTP, e.g., transmission of large data sets with variable availability at high speeds, to be deployed in environments that are not bandwidth constrained - simple testing has shown that deviation based compression techniques that have negligible processing impact can yield overall bandwidth requirements for STTP that are equal to or less than other serialization technologies, even when carrying extra information. For synchrophasor data, tests have shown data point serializations to have less than half the bandwidth requirements of IEEE C37.118 [1] when used over TCP with simple stateful methods for lossless compression [5].

Bandwidth requirements for STTP can often be further lowered by reducing the amount of data being transmitted. For most data structure serialization technologies and protocols, the very process of packaging and sending data in the form of data structures means that some data ends up being transmitted that is not used nor needed by receiving applications. Data reduction for these technologies means creating smaller data structures where it can be costly to maintain separate configuration models for multiple data structures just to achieve bandwidth improvements. Since STTP is designed as a publish / subscribe technology, a receiving application can choose to subscribe to only the individual data points it needs.

Data Point Level Publish / Subscribe

STTP intrinsically manages data at its most fundamental level, primitive types. Each uniquely identified primitive type value represents some form of physical measurement. When measured with periodicity and associated with a timestamp at the moment of measurement, the resulting sequence of measured values and associated timestamps are known as *time series* data. Since data points that are serialized by STTP can include time as part the state information for a value, STTP can be considered a time series data transmission protocol. However, the state information for values being transmitted is flexible - what is *time* for one data point could simply be a *sequence* for another. Additionally, the existence of some data points can be temporal, for example, to exchange a set of binary data, a temporary data point ID may be created that only exists until the binary data transfer is complete.

STTP uses a publish / subscribe based model for control of the data to be exchanged. This exchange is managed at the data point level where data sourced at a sending application, i.e., the *publisher*, will make a set of data points available for publication. A receiving application, i.e., the *subscriber*, will select a subset of the available points for subscription. As new data is made available at the publisher, the subset of the data as selected by the subscriber will be transmitted.

Data Point Metadata

A critical part of the publish / subscribe process is defining the data points that are available for subscription. An STTP publisher will define a tabular list of available data point identifiers and associated descriptive information as the *metadata* that is available to a subscriber.

Each data point includes a unique identifier; regardless of the binary transmission format, this identifier will exist as a statistically unique GUID in the defined metadata for the available data points. This makes the metadata from multiple publishers easier to merge into local repositories used by a subscriber.

At a minimum, each row in the STTP publisher metadata will include the GUID based data point identifier, a short human readable alpha-numeric *tag*, the primitive data type used for the value of the data point, a description, the enabled state and timestamps for the creation, last update and deletion of the data point.

Metadata in STTP is designed to be extensible. Different industries may require different kinds of available metadata in order to properly map and integrate with other protocols and environments. To accommodate the extensibility, other tabular datasets can be made available by a publisher as needed.

Data Point Access Control

STTP puts publishers in full control of access to data. A publisher can choose not to allow connections and/or expose any data to a subscriber that is not strongly identified. Publishers can choose to restrict data access at an individual data point level, a group level or at an identified subscriber level.

Selection of available points for an identified subscriber or a group can be controlled by an expression. Expression based access control means that the even as the data sources available to a publisher change, the expressions will still apply and need not be updated. For example, metadata will need to contain information about the primitive data type for a given data point - an expression based on this data type may look like the following:

```
ALLOW WHERE DataType='BOOL'
```

For this expression, all data points as defined in the metadata that have a data type of `BOOL` would be allowed for the group or identified subscriber. This expression would cause the allowed metadata to dynamically change as the available source data configured in the publisher changed.

Data Transport Channels

STTP data transport requires the use of a command channel using TCP/IP for reliable delivery of important commands. Optionally a secondary data channel can be established using UDP/IP for the transport of data that can tolerate loss. When no secondary UDP/IP is used, both commands and data will share use of the TCP/IP channel for communications.

Although not precluded from use over other data transports, the design of STTP is targeted and optimized for use over IP, specifically TCP/IP and UDP/IP. Even so, since the command/response implementation and data packet distribution of the STTP protocol is fairly simple, it is expected that commonly available middleware data transport layers, such as ZeroMQ or DDS, could easily support and transmit data using the STTP protocol should any of the messaging distribution and management benefits of these transport layers be useful to a particular deployment environment. However, these types of deployments are outside the scope of this documentation. If needed, STTP integrations with middleware layers should be added as reference implementation repositories to the STTP organizational site [4].

> 🍅 **?** JRC: *The question has been raised if a UDP only transport should be allowed? In this mode, any critical commands and responses would basically be sent over UDP. Thought would need to be given to commands and/or responses that never arrive and the consequences thereof.*

STTP Feature Summary

- Perform at high volume / large scale
- Minimize data losses (e.g., over UDP)
- Lower bandwidth requirements (e.g., over TCP)
- Optimized for the performant delivery of individual data points
- Automated exchange of metadata (no centralized registry required)
- Detect and expose communication issues
- Security and availability features that enable use on critical systems to support critical operations
- Publish/subscribe at data point level
- API implemented in multiple languages on multiple platforms
- Metadata will be versioned and tabular in nature
- Sets of metadata from multiple parties will be easy to merge

- Points defined in metadata will have a clear ownership path
- A minimal set of metadata will exist to support any STTP deployments
- Industry specific metadata extensions will exist to support specific industry deployments

Data Point Structure

🚧 Lead with paragraph on purpose / value of the section - (1) what is a data point structure and (2) why have a data point structure / value? Next paragraph would be contents of section...

... this section includes:

- Identification - maps to 128-bit Guid, transport mapping should be small
- Timestamp (required? could simply be a auto-incrementing counter)
- Value - multiple native types supports
- Flags - standardize minimal set of simple flags, complex state can be new data point

Data Point Value Types

- Null
- Byte
- Int16
- Int32
- Int64
- UInt16
- UInt32
- UInt64
- Decimal
- Double
- Single
- DateTime (need some thought on proper encoding, perhaps options)
- TimeSpan (Tick level resolution, or better, would be ideal)
- Char (2-byte Unicode)
- Bool
- Guid
- String (encoding support for UTF-16, UTF-8, ANSI and ASCII)
- Byte[]

🚧 Need to determine safe maximum upper limit of per-packet strings and byte[] data, especially since implementation could simply *span* multiple data points to collate a larger string or buffer back together.

🍅❓ *Should API automatically handle collation of larger data types, e.g., strings and buffers?*

Commands and Responses

🚧 Purpose of command/response structure, fundamentals of how it works, why it is needed

Commands

All commands must be sent over the command channel.

Code	Command	Source	Description
0x00	Set Operational Modes	Subscriber	Defines desired set of operational modes.
0x01	Metadata Refresh	Subscriber	Requests publisher send updated metadata.
0x02	Subscribe	Subscriber	Defines desired set of data points to begin receiving.
0x03	Unsubscribe	Subscriber	Requests publisher terminate current subscription.
0x0n	etc.		
0xFF	NoOp	Any	Periodic message to allow validation of connectivity.

Set Operational Modes Command

This must be the first command sent after a successful connection - the command must be sent before any other commands or responses are exchanged so that the "ground-rules" for the communications session can be established. The rule for this operational mode negotiation is that once these modes have been established, they will not change for the lifetime of the connection.

The subscriber must send the command and the publisher must await its reception. If the publisher does not receive the command in a timely fashion (time interval controlled by configuration), it will disconnect the subscriber.

As part of this initial exchange, the subscriber will propose the desired protocol version to use.

ℹ️ In modes of operations where the publisher is initiating the connection, the publisher will still be waiting for subscriber to initiate communications with a `Set Operational Modes` command.

- Wire Format: Binary
- Requested operational mode negotiations
  - String encoding
  - Compression modes
  - UDP data channel usage / port

Metadata Refresh Command

- Wire Format: Binary
  - Includes current metadata version number

Subscribe Command

- Wire Format: Binary
  - Includes metadata expression and/or individual Guids for desired data points

Unsubscribe Command

- Wire Format: Binary

NoOp Command

No operation keep-alive ping. It is possible for the command channel to remain quiet for some time if most data is being transmitted over the data channel, this command allows a periodic test of client connectivity.

- Wire Format: Binary

Responses

Responses are sent over a designated channel based on the nature of the response.

Code	Response	Source	Channel	Description
0x80	Succeeded	Publisher	Command	Command request succeeded. Response details follow.
0x81	Failed	Publisher	Command	Command request failed. Response error details follow.
0x82	Data Point Packet	Any	Data	Response contains data points.
0x83	Signal Mapping	Any	Command	Response contains data point Guid to run-time ID mappings.
0x8n	etc.			

ℹ️ For the response table above, when a response is destined for the data channel, it should be understood that a connection can be established where both the command and data channel use the same TCP connection.

Succeeded Response

- Wire Format: Binary (header)
  - Base wire format includes *in-response-to* command code
  - Can include response that is specific to source command:

Succeeded Response for Metadata Refresh

- Wire Format: String + Binary
  - Includes response message with stats like size, number of tables etc.
  - Includes temporal data point ID for "chunked" metadata responses
  - Includes number of metadata data points to be expected

Succeeded Response for Subscribe

Subscriber will need to wait for

- Wire Format: String + Binary
  - Includes response message with stats like number of actual points subscribed, count may not match requested points due to rights or points may no longer exist, etc.
  - Includes temporal data point ID for "chunked" signal mapping responses
  - Includes number of signal mapping data points to be expected

Succeeded Response for Unsubscribe

- Wire Format: String
    - Includes message as to successful unsubscribe with stats like connection time

Failed Response

- Wire Format: String + Binary (header)
    - Base wire format includes *in-response-to* command code
    - Includes error message as why command request failed
    - Can include response that is specific to source command:

Failed Response for Set Operational Modes

Failed responses to operational modes usually indicate lack of support by publisher. Failure response should include, per failed operational mode option, what options the publisher supports so that the operational modes can be re-negotiated by resending operational modes with a set of *supported* options.

- Wire Format: Binary
    - Includes operational mode that failed followed by available operational mode options

Data Point Packet Response

- Wire Format: Binary
    - Includes a byte flag indicating content, e.g.:
    - Data compression mode, if any
    - Total data points in packet
    - Includes serialized data points

ℹ️ The data point packet is technically classified as a response to a `subscribe` command. However, unlike most responses that operate as a sole response to a parent command, data-packet responses will continue to flow for available measurements until an `unsubscribe` command is issued.

Signal Mapping Response

- Wire Format: Binary
    - Includes a mapping of data point Guids to run-time signal IDs
    - Includes per data point ownership state, rights and delivery characteristic details

Data Point Characteristics

- Priority (e.g., control over data delivery priority)
- Reliability (e.g., must be sent over TCP channel)
- Verification (e.g., notification of successful transport)
- Exception (e.g., delivery on change)
- Resolution (e.g., down-sampling)

Metadata

- Wire Format: Tabular XML format (XML) - highly compressible
- Primary data point identifier is Guid (describe)
- Extensibility
- Rights based content restriction

Dataset Contents

- Minimum required dataset for STTP operation
- Industry specific dataset extensions (outside scope of this doc)

Dataset Filtering

- Format of expressions that work against metadata
  - SQL style expressions
  - Regex style expressions
- Application of expressions
  - Metadata reduction
  - Data point access security

Dataset Versioning

- Versioned
- Difference based publication

Dataset Serialization

- Serialization for transport
  - Packet based publication using temporal data point
  - Publisher reduction by access rights and diff-version
  - Subscriber reduction by filter expression
- Serialization to local repository
  - Merging considerations
  - Conflict resolution
  - Ownership control

Compression

- Types of compression
  - Stateful data compression (TCP)
  - Per-packet data compression (UDP)
  - Metadata compression (GZip)
- Compression algorithm extensibility
  - Negotiating desired compression algorithm

Security

- Access control list (ACL) security is always on

Encrypted Communications

- Transport layer security (TLS) over TCP command channel
- UDP data channel traffic secured via AES keys exchanged over TCL command channel

Strong Identity Validation

- X.509 certificates
- Self-signed certificates

Publisher Initiated Security Considerations

How does publisher initiated connection, to cross security zones in desired direction, affect identity validation and TLS?

Access Control Lists

- Allow/deny for specific points (data point explicit)
- Allow/deny for group with specific points (group explicit)
- Allow/deny for filter expression (filter implicit)
- Allow/deny for group with filter expression (group implicit)

Expression based Access Control

- Expressions can be used to define filters and groups
- How do filters work against extensible metadata, missing columns?

Access Control Precedence

- (1) Data Point Explicit
- (2) Group Explicit
- (3) Filter Implicit
- (4) Group Implicit

References and Notes

1. C37.118.2-2011 - IEEE Standard for Synchrophasor Data Transfer for Power Systems, IEEE, 2011

2. The MIT Open Source Software License

3. RFC 2119, Current Best Practice Scott Bradner, Harvard University, 1997

4. STTP Repositories on GitHub, Various specification documents and reference implementations, 2017

5. New Technology Value, Phasor Gateway, Peak Reliability, Task 7 Data Delivery Efficiency Improvements, DE-OE-701., September 2016

6. Character Encoding, Jukka Korpela, February 27, 2012

7. Google Protocol Buffers Overview, Google, May 31, 2017

8. Thrift: Scalable Cross-Language Services Implementation, Mark Slee, Aditya Agarwal and Marc Kwiatkowski, April 1, 2007

9. Protocol Buffers - Techniques - Large Data Sets, Google, December 10, 2015

10. Thrift Remote Procedure Call - Protocol considerations - Framed vs. unframed transport, Apache Software Foundation, Apache Thrift Wiki, Sep 21, 2016

11. The default MTU sizes for different network topologies, Microsoft, Article ID: 314496, June 19, 2014

12. Collisions and collision detection - What are collisions in Ethernet?, Valter Popeskic, November 16, 2011

13. The Delay-Friendliness of TCP, Eli Brosh, Salman Abdul Base, Vishal Misra, Dan Rubenstein, Henning Schulzrinne, pages 7-8, October 2010

14. Real-Time Application of Synchrophasors for Improving Reliability, RAPIR Task Force, October 18, 2010

15. User Datagram Protocol (UDP) and IP Fragmentation, Shichao's Notes, Chapter 10

16. Synchrophasors and Communications Bandwidth, Schweitzer Engineering Laboratories, April 1, 2010

17. Implementation and Operating Experience with Oscillation Detection at Bonneville Power Administration, Matt Donnelly, March 2017

Contributors

The following individuals actively participated in the development of this standard.

- J. Ritchie Carroll, GPA
- F. Russell Robertson, GPA
- Stephen C. Wills, GPA

ASP Project Participants

Project Collaborators	Project Financial Partner	Vendor	Utility	Demonstration Host
Bonneville Power Administration	♦		♦	
Bridge Energy Group				
Dominion Energy	♦		♦	EPG
Electric Power Group	♦	♦		
Electric Power Research Institute				
ERCOT			♦	
Grid Protection Alliance (Prime)	♦	♦		
ISO New England			♦	
MehtaTech		♦		
Oklahoma Gas & Electric	♦		♦	WSU
OSIsoft		♦		
Peak Reliability			♦	
PingThings		♦		
PJM Interconnection			♦	EPG
Southern California Edison			♦	
San Diego Gas & Electric	♦		♦	WSU
Schweitzer Engineering Laboratories	♦	♦		
Southern Company Services			♦	
Southwest Power Pool	♦		♦	WSU
Space-Time Insight		♦		
Trudnowski & Donnelly Consulting Engineers		♦		
Utilicast	♦	♦		
Tennessee Valley Authority	♦		♦	WSU
University of Southern California				
V&R Energy		♦		
Washington State University	♦	♦		
	26	11	11	12	6

Specification Copyright Statement

- Copyright © 2017, Grid Protection Alliance, Inc., All rights reserved.

Major Version History

Version	Date	Notes
0.1	TBD, 2017	Initial draft for validation of use of markdown
0.0	June 15, 2017	Specification template

Appendix A - STTP API Reference

appendix body

appendix body

> 💡 Links to language specific auto-generated XML code comment based API documentation would be useful.

Appendix B - IEEE C37.118 Mapping

appendix body

appendix body