PNNL-28499

Pacific Northwest
NATIONAL LABORATORY

A Comparison of Phasor Communication Protocols

A Comparison of IEEE C37.118.2-2011, IEC TR 61850 90 5 and the Streaming Telemetry Transport Protocol (STTP) for the Transfer of Synchrophasor and Other Streaming Data

February 2019

J. Ritchie Carroll
F. Russell Robertson

U.S. DEPARTMENT OF ENERGY

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service,
U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161
ph: (800) 553-6847
fax: (703) 605-6900
email: orders@ntis.fedworld.gov
online ordering: http://www.ntis.gov/ordering.htm

This document was printed on recycled paper.
(9/2003)

This paper was prepared by:

Grid Protection Alliance. Inc.
1206 Broad Street
Chattanooga, TN 37402

https://gridprotectionalliance.org

Principal Authors

J. Ritchie Carroll
F. Russell Robertson

The authors want to thank the following editors for comments and suggestions in review of this paper:

Ken Martin
Dan Brancaccio
Matt Donnelly

The development of this paper was funded by the Department of Energy via funding provided by Pacific Northwest National Laboratory.

CONTENTS

I. ABSTRACT

In this paper, we compare three approaches used for continuous transfer of real-time synchrophasor data: IEEE C37.118.2-2011, IEC TR 61850-90-5 and a new protocol being developed under the Department of Energy (DOE) project DOE-OE-859 called the Streaming Telemetry Transport Protocol (STTP). STTP is currently being advanced as a potential third standard protocol via the IEEE Power Engineering Society STTP P10 Work Group (P2664). Each of these three synchrophasor protocols is described in detail in this paper along with the basis for their operating characteristics using Internet Protocol (IP) transport.

The dominant protocol for the exchange of synchrophasor data is IEEE C37.118, both in the U.S. and internationally. The most recent IEEE C37.118 standard is broken into two parts where IEEE C37.118.1-2011 (Part 1) defines the normative synchrophasor measurement requirements and IEEE C37.118.2-2011 (Part 2) defines the protocol's data transmission format. Both the IEC TR 61850-90-5 and the emergent STTP specifications only address synchrophasor data transmission; therefore, this paper focuses on comparing the data transmission protocol elements of these standards.

The primary dimensions of comparison of these three protocols explored in this paper are: structure, efficiency, susceptibility to data loss, scalability, security, and other operability functionality. Additionally, though not specifically relevant to synchrophasor data telemetry, the three protocols are evaluated with respect to flexibility for transporting non-synchrophasor precise-time data streams.

II. INTRODUCTION

Standardized data communication protocols are needed to transfer time-synchronized voltage and current phasor measurements made by substation devices, such as phasor measurement units (PMUs), to upstream systems and analytic tools [1]. Each of these "synchrophasor" measurements is associated with a time stamp that is acquired from a precise time source, such as a Global Positioning System (GPS) clock [2]. By making synchrophasor measurements at multiple locations on the grid then combining and time-aligning these measurements, a wide-area coherent data set is created to enable power system analysis and control [3].

PMU functionality is built into many intelligent substation electronic devices including digital relays and fault recorders but can also exist as hardware dedicated to the task. Substation devices that have PMU functionality sample voltages and currents using potential transformer (PT) and current transformer (CT) inputs, at high sampling rates (up to many kilohertz). This rapidly sampled data is used to produce derived synchrophasor measurement values at a lower streaming data rate, typically 30 samples per second in the U.S. The algorithms used to calculate the derived synchrophasor quantities are prescribed in detail in the IEEE C37.118.1-2011 standard and its IEEE C37.118.1a-2014 amendment [4].

In this document, three protocols identified as the most widely used for the transfer of synchrophasor data, IEEE C37.118.2-2011, IEC TR 61850-90-5 and STTP / IEEE P2664[1], will be described so that they can be compared from the perspective of (1) structure, (2) efficiency, (3) susceptibility to data loss, (4) scalability, (5) security, and (6) other operability functionality. An overview of how these protocols emerged follows.

A Brief History of Synchrophasor Protocols

The first standard synchrophasor protocol, IEEE 1344, was created in 1995 based on the original development at Virginia Tech. This protocol included time synchronization and measurement standards defined around sample timing. Its simple structure was loosely based on COMTRADE and focused on the delivery of data from a single measurement device to the control center. This simple protocol was extended with an interim protocol called PDCstream

[1] Although at the time of writing STTP / IEEE P2664 is still being developed, it is based upon the existing Gateway Exchange Protocol (GEP) which is in wide use for the exchange of synchrophasor data.

developed by Western Electricity Coordinating Council (WECC) utilities which added data quality indications and allowed exchange of data from multiple measurement devices between phasor data concentrators (PDCs) and higher-level applications.

The next synchrophasor standard, IEEE C37.118, was completed in 2005. It introduced the total vector error (TVE) concept for evaluating measurements, established steady-state performance requirements and extended the data communication profile using concepts from IEEE 1344 and PDCstream, such as being able to combine data received from multiple devices into a single larger frame of data. As was the case with IEEE 1344, the IEEE C37.118 protocol was crafted to fill what was at the time a major gap in utility deployment of synchrophasor data systems – the need for an efficient protocol to support the reliable communication of high-volume (relative to other substation data flows) synchrophasor data from the measurement device to the control center.

In 2009, the IEEE made a request to the IEC for an IEEE C37.118 dual logo but the request was declined by the IEC because of a preexisting protocol technology, i.e., IEC 61850-9-2, which could convey synchrophasor information. Following this, a joint task force was formed between the IEEE and IEC which worked on methodologies and agreements that led to changes in IEEE C37.118 and the ultimate creation of IEC TR 61850-90-5.

To facilitate harmonizing the IEEE and IEC work, IEEE C37.118 was split into two parts before further development. Part 1 focused on the metrology of synchrophasor measurements and added requirements for dynamic operating conditions, frequency, and rate of change of frequency. Part 2 focused on defining the protocol's binary data format. Both revisions were completed in 2011. Although Part 2 added new communication functionality, such as a new configuration frame format, its development was limited so that it would be backwards compatible with the original, then widely deployed, 2005 standard.

The IEC 61850-90-5 technical report was completed in 2012. Given what at the time was the ubiquitous use of IEEE C37.118, the IEC standard included technical documents to facilitate migration from IEEE C37.118 to 90-5. These documents addressed updates to the IEC 61850-6 standard so that the existing IEEE C37.118 configuration frame could be used to define measurements being published in 90-5. The documents also included information on the use of Generic Object-Oriented Substation Event (GOOSE) messages and Sampled Values for synchrophasor data as prescribed in the umbrella IEC 61850 standard. The first implementation

of the IEC TR 61850-90-5 protocol was deployed at Pacific Gas and Electric Company (PG&E) between GE Multilin N60 and openPDC systems [5].

In 2014, as part of the DOE funded Secure Information Exchange Gateway (SIEGate) Project (DE-OE0000536), a new protocol called the Gateway Exchange Protocol (GEP) was introduced to handle an expanded set of requirements for the secure exchange of the data necessary to support real-time (i.e., current day) grid operations. This real-time data exchange requirement included synchrophasor data, SCADA data, and file-based data. The GEP protocol allowed SIEGate to control access to data at an individual measurement, or point, level. In addition, GEP was focused on control-center-to-control-center communications and was designed for very high-volume data flows.

Although GEP was open source and in common use by utilities, the protocol was not a formal standard and did not have wide multi-vendor adoption. In 2017, the DOE funded the Advanced Synchrophasor Protocol (ASP) Development and Demonstration Project (DE-OE0000859) to create a new protocol, extending and improving on GEP, with the goal of standardizing the protocol. Like GEP, the new protocol was optimized for the demands of transporting high-volumes of streaming synchrophasor data, however, it was not limited to synchrophasor data as the protocol allows for the transmission of any information that can be represented longitudinally, e.g., time-series data. The new protocol was called the Streaming Telemetry Transport Protocol (STTP), so named to emphasize its generalized applicability to the transfer of streaming data.

In 2018, the IEEE Power Engineering Society P10 STTP Working Group was established to develop a project authorization request (PAR) to put STTP on a path for standardization. The PAR was approved by the IEEE-SA New Standards Committee on September 27, 2018 and given a proposed IEEE standard number of P2664.

III. COMMUNICATIONS BACKGROUND

All synchrophasor protocols require an underlying communications transport layer. Protocol operation and data transmission quality are directly affected by the transport layer behavioral specifics; therefore, consideration of transport layer properties and behaviors is a crucial factor in understanding a protocol's design and operation.

Internet Protocol

The Internet Protocol (IP) is by far the most common protocol for the transmission of most any kind of data. The physical medium supporting IP is designed for general purpose transfer of data between any number of networked devices. Transport of variable sized blocks of data is supported in IP by breaking the data into smaller fragments called "data packets". IP supports a variety of higher-level transport protocols to control the behavior of the transmission of data packets over a network. For synchrophasor protocols, the most common high-level protocols for IP are the transmission control protocol (TCP) and the user datagram protocol (UDP) [2], each of which behave differently when dealing with data packet loss. Consequently, many of the impacts a large frame of synchrophasor data has on an IP network as well as its probability of it being delivered without loss is dependent upon the high-level transport protocol used to send the frame of data.

For IP, a block of data that exceeds the negotiated maximum transmission unit (MTU) size, i.e., the maximum size of a data packet, will be divided into multiple fragments where each fragment is an ordered data packet, see Figure 1. The typical MTU size for Ethernet networks is 1,500 bytes [6], however, the actual bytes available to the payload will be less than the MTU size since a portion of the data packet, the "header", is used by the IP protocol itself to identify packet source and destination information. As an example, if a frame of IEEE C37.118.2-2011 data is 1,914 bytes – the approximate size of a configuration frame with four average sized PMUs – it will take at least 2 IP data packets to send the frame. If a frame of data is 65,536 bytes, the absolute largest frame size allowed by either IEEE C37.118.2-2011 or IEC TR 61850-90-5, it will take at least 44 IP data packets to send.

[2] These protocols are often labeled as TCP/IP and UDP/IP for high-level protocol over Internet Protocol

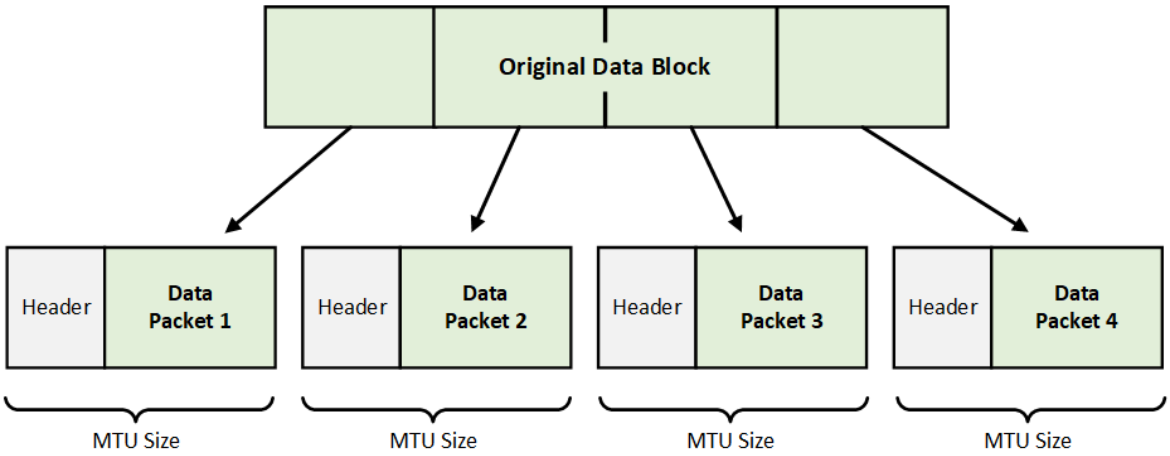

Figure 1. Packet Fragmentation

Data packets can only be transmitted over a physical network connection one packet at a time. When two or more data 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 are "dropped" [7]. Network collisions are not common on modern network infrastructure because of the increasing use of full-duplex and switch-based technology [8]; however, with heavy IP network traffic, data packet loss still occurs [9].

Non-collision-based data transport issues can occur when several devices are simultaneously transmitting data at high speed thereby putting a specific piece of supporting network equipment in a position where it cannot send all the traffic to the destination port. The reasons for the large data transport latency or for the dropped packets are typically CPU contention and memory limitations for data buffering[3].

Because data packets can (and will) be dropped, IP is inherently unreliable. If any data packet existing as an ordered fragment of a larger data block gets dropped, the original data block can no longer be reconstituted on the receiving machine. It is the higher-level IP protocols, e.g., TCP, that handle the retransmission of dropped packets to ensure reliable delivery of a block of data. Because of the protocol design of IEEE C37.118.2-2011 and IEC TR 61850-90-5, without retransmission dropped data packets result in loss of the entire original data block.

[3] Switch technology can also allow for a pause frame that is used for flow control at the Ethernet layer, however, if the buffers are at capacity the result is still the same, dropped packets.

With retransmission, a large data block will be reliably delivered, but retransmissions can have a negative impact on latency.

TCP/IP - The TCP protocol for IP is focused on reliability – assuring the data packet order and integrity of the originally published block of data. The protocol guarantees that, even as packets are dropped, data packets get retransmitted and are inserted into their original order so that a received block of data exactly matches what was sent. Retransmission of data packets increases delivery latency and subsequent data blocks are held-up in memory on the receiving device until all prior blocks have been received to maintain packet order. This process requires "acknowledgement of delivery" and includes error checking.

As a TCP connection is set up between two systems, one system is considered a client initiating a connection and the other a server listening for a connection. This creation of a connection between a server and client is called handshaking and is used to establish a unique communications path between the server and client systems. Reliable data delivery in a heterogenous network environment is an absolute requirement for many data transfer use cases, hence TCP/IP's popularity. However, all these activities to assure reliable data delivery result in TCP being considered a "heavy-weight protocol".

In the case of synchrophasor data transmission, TCP is the protocol of choice when data completeness is valued over timeliness of delivery, e.g., for transmission of data to a permanent archive. Even with the possibility of increased latencies, many utilities select TCP as their synchrophasor data transport of choice and often overprovision network resources to reduce packet retransmission to assure low-latency, reliable data delivery.

UDP/IP - The UDP protocol for IP is focused on timely data delivery at the expense of reliability. The protocol employs no functionality to assure the proper order of data packets nor does it include functionality to retransmit dropped packets. There is also no provisioning for flow control, congestion control or handshake mechanisms. The lack of handshake negotiation between the sender and receiver results in UDP being considered a "connectionless" protocol. In short, UDP is a "fire-and-forget" protocol. In comparison to TCP, the UDP protocol is simple and is consequently considered a "light-weight protocol".

The UDP protocol is not as popular as TCP since it allows for data loss; however, UDP packet losses on a network are measurable [10]. If measured losses are low and the use case for the data transfer can tolerate some amount of loss, then the UDP protocol is an attractive

approach. Unlike TCP, by not retransmitting data packets UDP does not suffer with increased transmission latencies during times of high network stress.

Another cause of UDP data loss is improper buffer sizing as managed by the operating system socket implementation. Unlike losses that occur due to collisions or network stress, the configurable sized UDP buffer physically located on a sending or receiving system is used to temporarily hold UDP data until it can be processed – when this buffer is full, any further incoming or outgoing data will be dropped. This data loss occurs at the application level and often results in a mystery data loss diagnosis for those certain that the data was received from a wire analysis but never entered the protocol parser. For data that is continuously transmitted over UDP, creating a properly sized socket buffer, for both receive and transmit, is an important configuration option to reduce data loss.

In the case of synchrophasor data transmission, UDP is the protocol of choice when timeliness of delivery is valued over data completeness such as may be the case for a time-sensitive analytic engine. Even with the possibility of data loss, many utilities select UDP as their synchrophasor data transport of choice to ensure low-latency data delivery. For example, analytics results that provide operational insights during rapidly changing system conditions need up-to-date data – for this use case, operator reaction to delayed information could have unintended harmful consequences.

UDP Multicast - The UDP protocol for IP includes a feature for group-based data transmission called multicast. UDP multicast uses as a special IP address range reserved for traffic that is designed to broadcast a sole source stream to multiple destinations; a typical use case being the transmission of media streams, e.g., radio or live video. Multicast streams are handled by routers in a local network. By default, most routers will not allow UDP multicast traffic to cross into other network segments without explicit configuration, so its use within a routable network infrastructure must be a planned activity.

In the case of synchrophasors where the source data stream will have benefits when shared by multiple parties, UDP multicast is often used. A common use case is to use a single combined data stream of all available synchrophasor data and distribute the combined data stream to multiple receiving applications [11], e.g., in a laboratory environment. However, the distribution of data in this fashion means that all applications receive all data, which is not always an efficient use of bandwidth. Multicast deployments are best suited to single, smaller streams of data allowing sole source data streams to be distributed to multiple parties, e.g., data

from a single device in a substation with limited network bandwidth being transmitted to multiple systems at a control-center in a network with ample bandwidth availability.

The challenge with traditional multicast, i.e., "any-source" multicast, for synchrophasor data is that any party can initiate a publication on a specific multicast IP address/port "endpoint" combination. When two or more parties start publishing on the same multicast endpoint, subscribers to the endpoint will receive all data from all sources; current frame-based synchrophasor protocols are not designed to work with interleaved data frames with different configurations from multiple sources. To alleviate this potential issue, multicast synchrophasor implementations typically operate with "source-specific" multicast, also called Protocol Independent Multicast Source-Specific Mode (PIM-SSM) as defined in Internet Group Management Protocol Version 3 (IGMPv3) [12]. Put simply, the source-specific multicast mode requires that the subscriber identify the source IP address of the desired publisher when initiating a subscription. Using the source IP address, the router can intelligently route only the desired data to the subscriber.

Serial Communications

Unlike IP, serial communication is not used as a general-purpose data transport, instead it is used to transfer data between two physical devices, i.e., it is a point-to-point communications protocol. Serial communications technology has advanced significantly over the years with standards such as the universal serial bus (USB) and the high-definition multimedia interface (HDMI) coming into common use in general purpose computing [13].

Although almost non-existent in modern telecommunication and computing centers, the 1960's era universal asynchronous transmitter and receiver (UART) protocol paired with the RS-232 electrical standard [13] [14] in still in wide use in the electric power industry. UART with RS-2322 is a simple data transmission technology that incorporates two separate wires for transmitting and receiving data [15]. RS-232 communication media and modulation methods limit data transmission to 100s of kilobits per second.

In environments where the infrastructure for serial communications already exists, electric utilities sometimes opt to use this infrastructure for the transmission of synchrophasor data. A common use of serial communications for synchrophasor data is inside substations between the measurement device and local communications or aggregation appliances.

IV. PROTOCOL DATA CHARACTERISTICS

Synchrophasor protocols are characteristically limited to a set of standard data types, groupings, publication frequencies, and serialization formats. The following paragraphs provide detail on the common data characteristics for synchrophasor protocols.

Types of Measured Quantities

All synchrophasor protocols define a specific format for the representation of time and the measured or calculated values that are associated with a given timestamp. These values include electrical system values, i.e., the "synchrophasor measurements", and scalar power system measurements, such as frequency, and binary flags that represent asset, metering, or data quality state.

All three protocols recommend that synchrophasor timestamp values be provided in Coordinated Universal Time (UTC) with accuracy requirements defined in the IEEE C37.118.1-2011 standard. To produce measurement timing with sufficient accuracy to meet the standard, measurement systems require a precise and reliable time source such as a GPS clock. The actual bitwise serialization of timestamp values with included sub-second detail is protocol specific and typically idiosyncratic.

Synchrophasor measurements include "phasor values", as either a voltage or a current, which are described as a tuple of two floating-point or integer quantities. A phasor is a complex equivalent of a simple cosine wave quantity where the complex modulus (in polar format) is the cosine wave amplitude and the complex angle is the cosine wave phase angle. A phasor measurement is an estimate of the actual angle calculated from many samples using the GPS common time signal as the reference for the measurement. As such, phasor measurements from all sites are considered synchronized and have a common phase relationship. The tuple that represents the complex phasor value have either a polar representation (a magnitude and an angle) or a rectangular representation (real and imaginary values). The actual coordinate representation used is denoted in the protocol configuration which defines metadata for the measured values. Although a typical substation measurement device will have three physical PT and CT inputs for an individual transmission line, it is common for a substation measurement device to be configured to restrict the reported voltage and current phasor values to only the computed positive sequence values.

Synchrophasor protocols also include formats for an estimated frequency value along with a value that represents the rate-of-change of the frequency (ROCOF). Frequency values are often transmitted as an offset, i.e., difference, from the nominal frequency.

The specifications allow for other optional timestamped measurements. These measurements are categorized as either a floating-point quantity, called an analog value, or a set of binary states (bits), called a digital value, which consists of 16 distinct state measurements per value. Analog values are typically used to carry meter-calculated data, such as real and reactive power. Digital values are a group of Boolean values that typically record the state of power system equipment such as a switch, breaker, or alarm state.

Finally, a single integer value, overloaded to contain a set of 16 "status flags" (bits), is specified to provide information on the state of metering such as the quality of the measured values, and known accuracy of the time source, among others.

Data Publication Rate

Synchrophasor protocol standards define a set of expected data publication rates with the actual publication rate being a major element of substation device configuration. The most widely used publication rate for synchrophasor data in the U.S. is 30 reports per second – also called samples per second. However, some utilities have configured substation devices to publish synchrophasor data at 60 or 120 samples per second. A hybrid approach is also used where the substation device is publishing locally within the substation at a high rate and that synchrophasor data is then down-sampled prior to communication to the control center. Other less common synchrophasor publication rates include 10, 12, 15, 20, 25, 50, 100, 240 and 480 samples per second. Selectable rates are a multiple of 50 or 60, the common nominal frequency rates of power systems.

Measurement Groupings

As standards for substation to control-center communication, IEEE C37.118.2-2011 and IEC TR 61850-90-5[4] are device-centric protocols and as such it is natural for these protocols to create groups of measurements organized by substation measurement device. These device-

[4] Although the use case for data transport of IEC TR 61850-90-5 matches that of IEEE C37.118.2-2011 per implementation agreement, technically 90-5 can be configured to transport any sequence of defined measured values, however, all the values in the group must be for the same measured timestamp.

centric protocols also support multiple data groups – i.e., data from multiple devices – within a single data stream.

Since PMU functions are typically an addition to existing functionality in mainstream substation measurement devices, the phasor measurements available for publication from a device are usually limited to the physical PT and CT inputs of the device. For example, a protective relay will normally have 3-phase voltage and current measurements available for a single transmission line; however, a DFR will typically measure all 3-phase currents and voltages for every transmission line in the substation[5]. Regardless of the number of measurements or the number of transmission assets monitored, the typical paradigm of measurement grouping is by measurement device. Both the IEEE C37.118.2-2011 and the IEC TR 61850-90-5 synchrophasor standards are designed to operate on sets of data grouped by measurement device, e.g., a PMU, and are consequently considered "device-centric" protocols. The STTP protocol is instead a "measurement-centric" protocol. STTP focuses on individual measured values without imposing time-alignment restrictions, or device-centric notions, allowing measurement groupings, i.e., "subscriptions", to be based on application need rather than imposing the use of predefined high-level groupings. As an example, the groups can be category driven, e.g., all available frequencies, with the availability of measurements changing dynamically, increasing or decreasing while data continues to stream, as source data made available to the publisher changes.

Measurement Serialization

Software that takes newly acquired measurements and groups them together into a serialized package to prepare it for transport is called a "protocol generator". Unless otherwise specified, serialization of synchrophasor measurement quantities by a protocol generator into a protocol's specific binary format always uses network bit ordering, known as big-endian encoding. Most host computers manage numeric quantities with little-endian encoding; therefore, the bits of the values to be transmitted are typically reversed during the serialization process.

---

[5] Data grouping illusions often develop where a group of synchrophasor data is associated with all data from a substation or a single transmission line. If modeling of synchrophasor data develops on a device-equals-line or device-equals-substation notion, this can lead to modeling issues when devices that monitor phasors measured on individual or multiple lines are later added to the system.

V. DATA FRAMING

Synchrophasor protocols are implemented by software applications from the perspective of being either a data stream producer or a data stream consumer. As mentioned above, the data stream producer application implements a "protocol generator". The consuming application implements a data stream deserializer called a "protocol parser". The source applications hosting protocol generators will compose a block of data, which is held in memory and structured according to the protocol semantics, 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 and sequence to provide useful context and meaning.

The actual binary format of the data being sent will vary based on the specific protocol used. In the case of IEEE C37.118.2-2011 and IEC TR 61850-90-5, data will be structured into logical blocks called "frames". A frame exists as a serialization of ordered bytes representing the desired data to be transmitted, typically for a given timestamp, where the layout of the frame is organized in a logical fashion to accommodate later deserialization by a protocol parser. The kind of data in the frame determines its type, for example, a "configuration" frame holds meta-data, such as the order and names of measurements being transmitted – and a "data frame" contains measured synchrophasor values in the order specified by the configuration frame, as well as the precise timestamp of measurement for the contained values.

Checksums

For synchrophasor protocols that are designed to be communication transport neutral, such as IEEE C37.118.2-2011 which can be used over serial and IP, a cyclic redundancy check (CRC) checksum is computed over the frame of data to be transmitted and added to the end of the frame. The synchrophasor protocols often refer to the checksum as a "checkword" since the typical size of the sum is 16-bits, i.e., a "word". The checksum value is used to validate frame data integrity by the data parser and is particularly useful in detecting transport errors in connectionless data streams, e.g., UDP. It is also useful for serial connections since noise errors can occur during communication making the channel data susceptible to errors. When error correction over a lossy data connection is relegated to the application layer, a protocol-included CRC to check data validity is important.

Even though Internet Protocol already includes checksum values for every data packet, a frame of synchrophasor data can span many IP data packets. While uncommon, IP packets can arrive out of order[6], in this case the frame-based checksum also allows the protocol parser to detect out of order packets [16].

Synchronization Bytes

Connectionless transport protocols, like serial and UDP/IP, provide no direct protocol-level support for framing source data[7]. As a result, the synchrophasor protocols for connectionless transport protocols must be designed so that the start of a frame of data can be recognized. Marking the beginning of a frame is accomplished with a "synchronization byte" (or bytes). A synchronization byte is a generally unique byte[8] used to orient the protocol parser to the beginning of a frame. Although it is possible for the synchronization byte to appear as part of valid normal data within a data frame, in these cases the protocol parser will determine that this is not the beginning of the frame for other reasons, e.g., an invalid checksum, and then move on to the next synchronization byte to reattempt start-of-frame orientation.

When using TCP/IP, data transfer begins only after a connection has been established. As a result, the first received byte in a data packet will be the synchronization byte and data will remain aligned. From the perspective of the protocol parser using the TCP transport protocol, the synchronization byte is not necessary but is commonly used as additional validation of data alignment.

Frame Concentration

The IEEE C37.118.2-2011 and IEC TR 61850-90-5 standards define data frames such that the content from multiple frames, e.g., those received from multiple PMUs, can be conflated together for a given timestamp. In this way, once frames of data have arrived from

---

[6] TCP automatically manages out of order packets, but UDP does not. Out of order packets do not tend to occur very often on intranets with switch-based hub and spoke networks, but are more common on meshed and MPLS networks, i.e., in environments that support and use multi-path, parallel routing [16].

[7] If data is already publishing, serial connections can be established anywhere in a stream once connected. UDP/IP can drop packets, so the first data packet received upon connection may not be the first in a frame.

8 A byte is considered *generally unique* in context of all the bytes typically sent by the protocol, but not guaranteed to be unique, e.g., p-value < 2%.

multiple source devices (where all data was measured at a given timestamp) one super frame can be created to hold the data from all sources. The operation of combining frames together for the same timestamp is called "concentration". The function of concentrating data arriving from distributed locations requires that the system must wait for all expected data to arrive. Since data may never arrive, a timeout must be specified so that operations can eventually continue. This wait timeout is often referred to as the concentration "wait time"[9].

The functionality of concentration for synchrophasor data is typically handled by a "phasor data concentrator" (PDC) [17]. A PDC requires configuration to specify the incoming synchrophasor data streams that will be concentrated as well as the measurements that will be part of outgoing data streams. Other PDC configuration is also required, such as a concentration wait time value based on the most delayed inputs.

The process of concentrating multiple frames together into a compact binary data block can often be an effective and bandwidth efficient way to send large amounts of synchrophasor data. However, as the frames of synchrophasor data become larger, the process of serialization and deserialization becomes costlier in terms of both memory allocation and CPU utilization. Additionally, and importantly, there are also penalties that occur with large frames at the network transport layer.

[9] Some synchrophasor concentrator implementations instead refer to "wait time" as "lag time". This name is used to harmonize nomenclature with the other common concentration parameter called "lead time" which refers the allowed future-time reasonability validation that occurs when incoming timestamps are compared to the local system clock.

VI. LARGE FRAME IMPACT ON IP

Synchrophasor data is commonly transported over IP using the TCP and UDP transport protocols – both individually and in combination. Deployments that use both protocols simultaneously are characteristically configured such that the reliable TCP channel is used for transmission of commands and configuration and the UDP channel is reserved for transmission of synchrophasor measurement values where some loss can be tolerated.

Regardless of transport protocol, as a contiguous set of data (i.e., a data frame) is sent that is large enough to require fragmentation into multiple IP packets, the loss of any packet requires either the individual packet be resent as the dataset is being assembled or the retransmission of the entire dataset. Continual retransmissions can adversely impact latency and throughput and retransmission of the entire data set is particularly impactful [18]. Continuous transmission of large frames of data over IP causes increased network stress which compounds packet loss [9]. For these reasons, other popular general-purpose data serialization technologies, such as Google Protocol Buffers and Apache Thrift, recommend against the use of large data frames [19] [20].

The adverse impact of large data frames differs based on the high-level protocol being used, e.g., TCP or UDP. Therefore, the selection of an appropriate high-level protocol for a given use case is an important consideration for synchrophasor data transport reliability.

Large Frame Impacts using TCP

The most common Internet protocol, TCP/IP, creates an index for each network packet being sent for a frame of data and verifies that each packet is successfully delivered. TCP/IP is a “reliable protocol” in that it retransmits packets as many times as needed to assure delivery. However, because retransmission can increase latency, concentration of data frames that timeout can effectively cause the source data to never arrive at its destination.

Since each packet of data for the transmitted frame is sequentially ordered, TCP can fully reconstruct and deliver the original frame once all the packets have arrived. However, continuous streaming of large frames of data causes network hardware supporting TCP to suffer with increased memory allocation and computational burden, increasing data loss and retransmissions, which can lead to network slowdown. One unique distinction for IP based protocols is that at some level these issues will 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 retransmissions of dropped packets can induce cumulative time delays [21], especially as large data frames are published at rapid rates as is the case for synchrophasor data. Time delays are also exacerbated during periods of increased network activity which induces congestion and a higher rate of collisions or buffering – the effects on latency being pernicious. Real-time synchrophasor data must be accurate, dependable, and timely to be useful for grid operators [22].

Large Frame Impacts on UDP

The other common Internet protocol used for synchrophasor data is UDP/IP. Transmission of data over UDP differs from TCP in that UDP does not retransmit lost packets nor does it make any attempt to maintain the order of the transmitted packets. As such, UDP is considered a lossy data transmission protocol.

Even with the unreliable delivery caveats, UDP is still limited to packet sizes as defined by the MTU. Any packet larger than the MTU size must be fragmented – i.e., split into multiple smaller packets as described in section III. Like the TCP communication protocol, UDP attempts to reconstruct and deliver the originally transmitted frame of data; however, if even a single network packet is dropped, the entire data frame is lost and any packets that were already accumulated are discarded. In other words, there are no partial frame deliveries – frame reception with UDP is an all or nothing operation.

As was described for TCP, use of UDP results in similar stress on network equipment as data frame sizes increase requiring memory allocation and computational processing. The more problematic impact of large data frames with UDP is that the increased number of network packets needed to send a large frame increases the probability of dropping one of the individual packets in the frame. Since the loss of any one packet results in the loss of the entire frame of data, as frame sizes increase so does overall data loss.

VII. IEEE C37.118.2-2011 PROTOCOL OVERVIEW

The 2011 version of the IEEE C37.118 standard, including the later amendment in 2014 to cover performance requirements, is broken into two parts. Part 1 defines the measurement of synchrophasors, frequency, and rate of change of frequency under all operating conditions and it specifies methods for evaluating these measurements and requirements for compliance with the standard under both steady-state and dynamic conditions. Part one also defines requirements for time tags and synchronization.

Part 2 of IEEE C37.118 defines methods for the real-time exchange of synchrophasor data and specifies the messaging format, including message types, contents, and their use. The standard defines a simple and direct method of data transmission using an open access method to facilitate development and use of synchrophasors. It is Part 2 of the IEEE C37.118 standard that is the focus of this document.

Protocol Summary

The IEEE C37.118.2-2011 standard defines a method for exchange of synchronized phasor measurement data between electronic power system devices. The protocol specifies the messaging structure including types, use, contents, and data formats for real-time communication between PMUs, PDCs, and other applications.

Its simplicity and efficiency help to make the IEEE C37.118.2-2011 standard the most widely used protocol used to exchange synchrophasor data among measurement, data collection, and application equipment – including visualization systems, time-series data historians, PMUs and PDCs.

Protocol Structure

All data sent or received by the IEEE C37.118.2-2011 protocol is formatted into binary frames. The standard defines four frame types: *command frame*, *configuration frame*, *data frame* and *header frame*. Each frame type includes a common set of header values.

Common Header - Each of the frame types sent or received by the IEEE C37.118.2-2011 protocol include a common set of values at the start of each frame as shown in the table below:

Field	Byte Size	Description
SYNC	2	Sync byte followed by frame type and version number
FRAMESIZE	2	Number of bytes in frame (UINT16)
IDCODE	2	Stream source ID number (UINT16)
SOC	4	SOC time stamp (UINT32)
FRACSEC	4	Fraction of Second and Message Time Quality

Table 1. Common Header Fields

Command Frame - A command frame is used to control protocol behavior for an IEEE C37.118.2-2011 connection. Unlike other frames which are typically transmitted to a client which is handling protocol parsing, a command frame is sent to a server which will be primarily handling protocol generation. In the context of typical request / reply style communications paradigms, the command frame can be considered a request and all other frames the replies. Examples of available commands include *begin transmission of data frames* (CMD = 2), *stop transmission of data frames* (CMD = 1), *send a configuration frame* (CMD = 5 for CFG-2), and *send a header frame* (CMD = 3). Below are the elements of a command frame that exist beyond the common header:

Field	Byte Size	Description
CMD	2	Command being sent to the server (PMU/PDC)
EXTFRAME	0-65518	Extended frame data – FRAMESIZE controls length
CHK	2	CRC-CCITT

Table 2. Command Frame Fields

Configuration Frame - A configuration frame in the IEEE C37.118.2-2011 protocol contains information for processing a synchrophasor data stream; this includes the sequence of the data contained in data frames – as a result, a configuration frame must be received and processed by a protocol parser before any data frames can be fully parsed. The standard defines three configuration frame types labeled CFG-1, CFG-2 and CFG-3. The first two configuration frame types, i.e., CFG-1 and CFG-2, are structurally identical – they only differ in usage

context. CFG-1 is used to relay all the data that a device has to offer where CFG-2 reports the current active configuration, i.e., what is being reported in the data frames. Consequently, from the perspective of a protocol parser, it is CFG-2 that is required. The last configuration frame type, CFG-3, includes more meta-data information as well as the ability to span multiple frames. CFG-3 is optional, i.e., protocol generators can opt not to implement this frame type – as a result, protocol parsers cannot depend on its existence. Below are the elements of a configuration frame (for CFG-2) that exist beyond the common header:

	Field	Byte Size	Description
	TIME_BASE	4	Resolution of FRACSEC time stamp
	NUM_PMU	2	Number of PMUs in the data frame (UINT16)
┌	STN	16	Station Name—16 bytes in ASCII format
│	IDCODE	2	Data source ID identifies source of each data block
│	FORMAT	2	Data format within the data frame
│	PHNMR	2	Number of phasors (UINT16)
│	ANNMR	2	Number of analog values (UINT16)
│	DGNMR	2	Number of digital status words (UINT16)
│	CHNAM	16 × (PHNMR + ANNMR + 16 × DGNMR)	Phasor and channel names—16 bytes for each phasor, analog, and each digital channel (16 channels in each digital word) in ASCII format in the same order as they are transmitted
│	PHUNIT	4 × PHNMR	Conversion factor for phasor channels
│	ANUNIT	4 × ANNMR	Conversion factor for analog channels
│	DIGUNIT	4 × DGNMR	Mask words for digital status words
│	FNOM	2	Nominal line frequency code and flags
└	CFGCNT	2	Configuration change count
	Repeat Fields		Repeat fields STN to CFGCNT for NUM_PMU times
	DATA_RATE	2	Rate of data transmissions
	CHK	2	CRC-CCITT

Table 3. Configuration Frame (CFG-2) Fields

Data Frame - A data frame in the IEEE C37.118.2-2011 protocol is used to transmit synchrophasor measurement data[10] and a set of status bits for each of the included data blocks. The sequence of items defined in a data frame are outlined in the configuration frame, specifically CFG-2 – as a result, the configuration frame must be received before data frames can begin to be parsed. Below are the elements of a data frame that exist beyond the common header:

	Field	Byte Size	Description
┌	STAT	2	Bit-mapped flags defining current state and quality info
│	PHASORS[10]	4 x PHNMR or 8 x PHNMR	Phasor estimate value tuple 4-bytes per tuple (2 per value) in scaled integer format 8-bytes per tuple (4 per value) in floating-point format
│	FREQ	2 or 4	Frequency value (2-bytes scaled / 4-bytes floating-point)
│	DFREQ	2 or 4	ROCOF value (2-bytes scaled / 4-bytes floating-point)
│	ANALOG	2 × ANNMR or 4 x ANNMR	Analog value 2-bytes per value in scaled integer format 4-bytes per value in floating-point format
└	DIGITAL	2 x DGNMR	Digital data, 16-bit flags per value
	Repeat Fields		Repeat fields STAT to DIGITAL for NUM_PMU times
	CHK	2	CRC-CCITT

Table 4. Data Frame Fields

Header Frame - The payload for a header frame in the IEEE C37.118.2-2011 protocol is expected to be free-form, human readable information. A header frame is designated to provide ancillary information from a device that is measuring and publishing synchrophasor

[10] Phasors can be in polar (angle / magnitude) or rectangular (real / imaginary) format as specified in the FORMAT field in the configuration frame. Angle values are always represented as radians.

measurements, e.g., its data sources, filtering algorithms in use, etc. Below are the elements of a header frame that exist beyond the common header:

Field	Byte Size	Description
CMD	2	Command being sent to the server (PMU/PDC)
EXTFRAME	0-65518	Extended frame data – FRAMESIZE controls length
CHK	2	CRC-CCITT

Table 5. Header Frame Fields

Protocol Timestamp Format

Timestamps in IEEE C37.118.2-2011 are encoded as a 4-byte second-of-century, i.e., the 32-bit SOC value defined in the common header of all frames, and a 24-bit fraction of second, i.e., the lower 24-bits of the 32-bit FRACSEC value as defined in the common header. The second-of-century epoch is UNIX based representing the number of seconds since midnight on 1/1/1970 UTC. For non-data frames, the timestamp used is always provided in whole seconds without any fractional value. For data frames, timestamps include fractional time. Calculation of fractional time requires the use of the time base, i.e., the lower 24-bits of the 32-bit TIME_BASE value defined in the configuration frame. Time, including sub-second fraction, can be calculated using an expression like the following:

```
Timestamp = SOC + (FRACSEC & 0xFFFFFF) / (TIME_BASE & 0xFFFFFF)
```

Using the full 24-bits of the available fractional time base, the minimum fractional time interval that can be represented is 59.6 nanoseconds.

Protocol Security

No native security options are provided in the IEEE C37.118.2-2011 protocol. Since IEEE C37.118 is an application level messaging system, security must be provided by the underlying communications transport used to carry the messages. Some IEEE C37.118 device implementations do enforce a password-like feature such that no responses will be provided for command requests not specifying a matching IDCODE in the header of the provided command frame. However, this is limited to the maximum of 65,536 values per the 16-bit IDCODE field.

In lieu of native security options, some production implementations of IEEE C37.118 needing security have instead opted to deploy a virtual private network (VPN) between sources and syncs needing to exchange data thus creating a secure network tunnel for all network traffic flowing between the two endpoints.

Data Integrity - The IEEE C37.118.2-2011 standard employs the use of a CRC-CCITT [1] based checkword in its protocol implementation so that reconstitution of a frame of data at the application layer can be validated even when transported over an unreliable data transport, e.g., UDP or serial. Additionally, to help accommodate protocol parsing alignment when using connectionless transports, IEEE C37.118 also publishes a synchronization byte, i.e., value 0xAA, at the beginning of each of its frames – this is the first byte of the SYNC value defined in the common header of all frames.

Bandwidth Utilization

Application of compression techniques notwithstanding, e.g., those defined in STTP, the raw binary format of IEEE C37.118.2-2011 data frames is the most compact option available for the transmission of synchrophasor measurement data[11] [23] [24]. In addition, to help send more data over channels with limited bandwidth, e.g., a serial connection, the IEEE C37.118 protocol includes a mode of operation to send scaled integers instead of floating-point values. This mode optimizes payload size at the expense of less data resolution[12] to further reduce the required communications bandwidth, i.e., transmission of scaled 2-byte 16-bit integer values versus 4-byte 32-bit floating-point values.

[11] This assertion assumes that the configuration frame is only sent once per session. In deployments where the configuration frame gets broadcast on a periodic schedule, e.g., once per minute in a unicast only environment, bandwidth utilization will increase.

[12] For PMU devices that use a 16-bit A/D converter for source measurements, the integer-based scaling optimization may result in no loss of precision when scaling factors are properly configured.

VIII. IEC TR 61850-90-5 PROTOCOL OVERVIEW

The 2012 version of the IEC TR 61850-90-5 standard was developed to extend the widely used IEC 61850 communications protocol suite to include normative options for the transmission of synchrophasor data. Use of the IEC TR 61850-90-5 synchrophasor protocol is often a consideration for environments where IEC 61850 is already the primary communications protocol in use, e.g., within a substation, so that tools, processes, and standard naming conventions already provided by IEC 61850 can be leveraged.

The 90-5 technical report addition to IEC 61850 protocol is based on established practice and usage of IEEE C37.118 [25] with added native security options, see IEC 62351. The 90-5 standard focuses on using existing elements of the IEC 61850 protocol stack for publication of synchrophasor data but not on how the data is measured – instead, normative references to IEEE C37.118.1-2011 are included for the handling of synchrophasor metrology.

The 90-5 standard employs use cases to establish the protocol requirements and, building on tools and elements available in the existing IEC 61850 protocol stack, uses modeling to establish logical devices, nodes, and communications [26]. The protocol stack is used to define mappings to common synchrophasor functions, for example, a PMU is mapped to an IEC 61850 MMXU or MSQI logical node, and a PDC is an IEC 61850 proxy-server or gateway. The existing protocol stack is also utilized for the actual transmission of synchrophasor measurements by using IEC 61850 Generic Object-Oriented Substation Event (GOOSE) messages or Sampled Values, both of which now support routable implementations over IP[13].

Protocol Summary

The goal of IEC TR 61850-90-5 is to standardize communications protocols and interfaces for synchrophasor data in environments where IEC 61850 is already deployed, such as a substation, where equipment interoperability can lower equipment acquisition and installation costs.

The availability of standard object models provided through the IEC 61850 protocol is intended to lower substation engineering and design costs by enabling automated system engineering tools and processes and new substation designs. The design of IEC 61850 enables

[13] IEC 61850 GOOSE and Sample Value broadcasts were previously restricted to a local subnet [25].

in-substation wireless (no copper wires) communications that will lower substation construction and commissioning costs by reducing or eliminating relay-to-relay wiring.

IEC TR 61850-90-5 adds asset security inherited from IEC 61850 that incorporates the IEC 62351 cyber-security standard and transmits waveform samples in real-time, enabling high-speed data services that can support real-time protection and control actions. Using UDP multicast, IEC TR 61850-90-5 enables the use of single measurements (e.g., CT and PT transducer signals) by many users or devices and applications, which enhances efficiency and redundancy and reduces equipment connection and wiring costs [27].

Devices defined with IEC 61850 also have standard object naming conventions that are self-describing and discoverable by other IEC 61850 devices and controllers – this is expected to reduce the cost and time required for design, specification, configuration, testing, commissioning, and maintenance.

Protocol Structure

All data sent or received by the IEC TR 61850-90-5 protocol is formatted into binary frames. The 90-5 standard defines an IEC 61850 based implementation for a frame of data on a given session using either GOOSE or Sampled Values, referred to here as a *data frame*. The 90-5 protocol also allows for native protocol options for defining the configuration, i.e., names and sequences of values defined in a data frame using Substation Control Language (SCL) [26] as described in IEC 61850-6-1. To support transitional environments for synchrophasor implementations wanting to switch to IEC TR 61850-90-5, many 90-5 protocol implementations also support use of the IEEE C37.118.2-2011 configuration frame and a limited set of the IEEE C37.118.2-2011 command frame functionality. Since the configuration and command frames for IEEE C37.118 have been defined prior and describing SCL in detail is beyond the scope of this document, this overview focuses only on the structure of IEC headers and footers and the 90-5 data frame payload.

Sampled Value Tag Encoding - The transport options available for data frames in the IEC TR 61850-90-5 protocol are GOOSE and Sample Values – both of which use ANS.1 Basic Encoding Rules (BER) for serializing a Tag, Length, and Value (TVL) triplet. Serialization of a TVL triplet employs the use of byte-based markers to identify a distinct quantity. These markers are called “tags” and can represent data of any type, e.g., an integer value or a string. The format of the tags is a one-byte value representing the tag’s identification, followed by an

encoded tag length representing the length of the value, followed by the actual bytes of the value. The length is encoded using a custom algorithm that operates in an analogous manner to the common 7-bit integer variable length encoding algorithm but optimized for lengths that are less than 128 bytes in size. For example, given an array of bytes to hold the encoded length, "buffer", an integer-based "length" representing the size of the value to encode and an integer-based current "index" into the buffer, an implementation of the custom TVL triplet tag length encoding algorithm in C could look like the following:

```
if (length > 0x7F)
{
    if (length > 0xFF)
    {
        // 16-bit length value
        buffer[index++] = 0x80 | 2;
        buffer[index++] = (length & 0xFF00) >> 8;
        buffer[index++] = length & 0x00FF;
    }
    else
    {
        // 8-bit length value > 127
        buffer[index++] = 0x80 | 1;
        buffer[index++] = length & 0xFF;
    }
}
else
{
    // 8-bit length value < 128
    buffer[index++] = length & 0xFF;
}
```

This algorithm allows a variable length, e.g., the length of a string, to be serialized into 1, 2, or 3 bytes. Since the algorithm's maximum encoding size, i.e., 3 bytes, is greater than the size it would take to naturally encode a 16-bit integer, i.e., 2 bytes, its use presumes a nominal use-case where lengths are usually expected to be less than 128. For example, lengths are encoded into 1 byte when length values are less than 128 – this represents a serialization size optimization for length-prefixed string encoding if length values are expected to be typically less than 128.

IEC Headers - The IEC TR 61850-90-5 data frame and the associated IEC standard headers are part of an overall IEC 61850 session structure. This structure is more complex and dynamic than the simpler frames defined in IEEE C37.118.2-2011 and hence are larger in size. These IEC structures define a wrapping architecture to allow all the various kinds of IEC 61850

information to be exchanged as payload while simultaneously supporting multiple contexts, security implementations and routing options. Instead of being fixed, the structure is container based and defined as a hierarchy of levels representing layers of implementation that can adjust as needed to accommodate diverse types of data exchanges. Below are the elements that make up the header, applied in the context of a data frame for routable Sampled Value messages (content beyond APDU_LEN would differ for GOOSE[14]):

				Field[15]	Byte Size	Description
┌				LI	1	Length identifier for Transport Unit (0x01)
└				TRANS_TYPE	1	Transport Unit Type Marker (Connectionless = 0x40)
				SESSION_TYPE	1	Session Type Marker (Tunneled = 0xA0, GOOSE = 0xA1, Sampled Value = 0xA2)
┌				LI	1	Length Identifier (0x18)
│				COM_HDR	1	Common Header (0x80)
│	┌			LI	1	Length Identifier (0x16)
│	│	┌		SPDU_LEN	2	SPDU Length (UINT16)
│	│	│		SPDU_NO	2	SPDU Sequence Number (UINT16)
│	│	│		VERSION	2	Version Number (0x01)
│	│	│		KEY_TIME	4	UNIX SOC Time of Current Key (UINT32)
│	│	│		NEXT_KEY	2	Time of Next Key (UINT16)
│	│	│		SEC_TYPE	1	Security Algorithm Type Marker (None = 0x00, AES128 = 0x01, AES256 = 0x02)
│	│	│		SIG_TYPE	1	Signature Algorithm Type Marker (None = 0x00, SHA80 = 0x01,

[14] GOOSE messages are several times larger than those for Sampled Values, so this document focuses on the smaller Sampled Values implementation, especially as it relates to bandwidth comparisons with other protocols.

[15] The field names presented here are intended to establish an identifiable reference to a sequential point in the IEC TR 61850-90-5 data frame to assist with locally describing structural relationships and repeating data sections. The chosen name does not necessarily match field names otherwise defined in the IEC standard.

						SHA128 = 0x02, SHA256 = 0x03, AES64 = 0x04, AES128 = 0x05)
				KEY_ID	4	Index into Key Table (UINT32)
				PAYLOAD_LEN	4	Length of Payload (UINT32)
				PAYLOAD_TYPE	1	Payload Type Marker (UINT8) (GOOSE = 0x81, Sampled Value = 0x82)
				SIMULATION	1	Simulated Data Marker (No = 0x00, Yes = 0x01)
				APPID	2	Application ID (UINT16)
				APDU_LEN	2	APDU (Payload) Length (UINT16)
				TAG_SVPDU	1	Sampled Value Protocol Tag (0x60)
				PAYLOAD_SIZE	3	Encoded Payload Size (UINT16)
				TAG_NOASDU	1	Number of ASDU's Tag (0x80)
				NOASDU	2	Encoded ASDU Count (UINT8)
				TAG_SQASDU	1	Sequence of ASDU Length Tag (0xA2)
				SEQ _ASDU	3	Encoded Sequence Length (UINT16)
				Payload		(See Data Frame)

Table 6. IEC Header Fields

IEC Footers - The IEC footers directly follow the payload as described for a data frame. The footer elements close the hierarchy of containers that remained open in the header before the data frame payload. Below are the elements that make up the footer:

				Field	**Byte Size**	**Description**
				Payload		(See Data Frame)
				SIGNATURE	1	Signature Marker (0x85)
				HMAC_LEN	0 or 2	HMAC Length (UINT16)
				HMAC	0 to 33	Hash-based Message Authentication Code, Length based on SIG_TYPE: None = 0, SHA80 = 11, SHA128 = 17, SHA256 = 33, AES64 = 9, AES128 = 17

Table 7. IEC Footer Fields

Data Frame - The data frame of IEC 61850-90-5 is unique in the sense that it can repeat any number of past datasets within the current frame. This capability was added to help reduce overall data loss over a connectionless transport, e.g., UDP, by making sure that the current set of synchrophasor measurements, at timestamp T, can also include one or more earlier sets of measurements, such as, measurements at T - 2 and T - 1. Including past datasets in each frame with a large configuration can negatively impact data quality[16], however, the number of past datasets to send within each data frame is adjustable through configuration. The current data set, i.e., T - 0, is always the last data set in the frame.

The order of the actual synchrophasor measurement data in a frame of IEC TR 61850-90-5, regardless of GOOSE or Sampled Value implementation, is structurally in the same order as data defined in the IEEE C37.118.2-2011 protocol. Even though the data is in the same sequence, unlike IEEE C37.118, the data formats are fixed, i.e., all data values are delivered in 32-bit floating-point (no scaled integers) and phasors are in polar format (angle, in degrees[17], and magnitude value – no imaginary option). Below are the elements that make up the data frame in Sampled Value format:

[16] Larger frame sizes can have a negative impact on the quality of data transmissions over UDP/IP, see prior section LARGE FRAME IMPACT ON IP. For smaller frame sizes, repetition of past datasets over UDP can have the net effect of reducing overall data loss at the expense of increased bandwidth and processing costs, as incurred due to the need to ignore duplicate data.

[17] This differs from IEEE C37.118 where polar formatted phasor angles are always represented in radians.

		Field	Byte Size	Description
┌		TAG_ASDUSQ	1	ASDU Sequence Length Tag (0x30)
│		ASDU_SEQLEN	3	Encoded Sequence Length (UINT16)
│		TAG_MSVID	1	Multicast Sampled Value ID Tag (0x80)
│		MSVID[18]	Variable	Encoded Length-Prefixed Multicast Sampled Value Identifier (STRING)
│		TAG_SMPCNT	1	Sample Count Tag (0x82)
│		SAMPLE_CNT	3	Encoded Sample Count (UINT16)
│		TAG_CONFREV	1	Configuration Revision Tag (0x83)
│		CONF_REV	3	Encoded Configuration Revision (UINT16)
│		TAG_REFRTM	1	Refresh Timestamp Tag (0x84)
│		REFR_TM	9	Encoded Refresh Timestamp (UINT64)
│		TAG_SMPSYNC	1	Sample Synchronized Tag (0x85)
│		SMP_SYNC	2	Encoded Sample Synchronized (UINT8)
│	┌	STAT	2	Bit-mapped flags defining current state and quality info
│	│	PHASORS	8 x PHNMR	Phasor estimate value 2-part tuple 8-bytes per tuple (4-bytes per value)
│	│	FREQ	4	Frequency value
│	│	DFREQ	4	ROCOF value
│	│	ANALOG	4 x ANNMR	Analog value (4-bytes per value)
│	└	DIGITAL	2 x DGNMR	Digital data, 16-bit flags per value
└		*Repeat Fields*		Repeat fields STAT to DIGITAL for NUM_PMU times
		Repeat Fields		Repeat fields TAG_ASDUSQ to DIGITAL for NOASDU times (i.e., repeat for each included past data set plus the current one)

Table 8. 90-5 Data Frame Fields

[18] According to the established implementation agreement for interoperability with IEEE C37.118, the MSVID field will contain an ID code and station name separated by an underscore that will map to the IEEE C37.118 configuration frame IDCODE and STN fields.

Protocol Timestamp Format

Timestamps in IEC TR 61850-90-5 are encoded as a 4-byte second-of-century, i.e., the higher 32-bits of the 64-bit REFR_TM value defined in the data frame, and a 24-bit fraction of second, i.e., the lower 24-bits of the 64-bit REFR_TM value – the remaining 8-bits are reserved for time-quality flags. The second-of-century epoch is UNIX based representing the number of seconds since midnight on 1/1/1970 UTC. Calculation of fractional time in 90-5 uses a constant divisor value of 16,777,216. Time, including sub-second fraction, can be calculated using an expression like the following:

```
Timestamp = (REFR_TM & 0xFFFFFFFF00000000) +
            (REFR_TM & 0x00000000FFFFFF00) / 16777216
```

The minimum fractional time interval that can be represented is 59.6 nanoseconds.

Protocol Security

The IEC TR 61850-90-5 standard works with the pre-existing IEC 61850 security options, as defined in IEC 62351-9, to allow the secure transport of synchrophasor measurements. The security options make use of digital signatures, exchanged between authenticated parties, to ensure only authenticated users have access to the desired data and specifies the use of the Group Domain of Interpretation (GDOI) protocol[19] to handle distributed security with UDP multicast with group authentication of broadcast packets using the shared, group key.

When two parties have already authenticated and exchanged keys using native 61850 security options[20], the appropriate key, as referenced by the KEY_ID field in the IEC headers, will be used to decrypt the payload contents defined in the data frame. This implementation works to protect streaming data even in UDP multicast environments where any recipient can use standard multicast mechanisms to receive a stream of the synchrophasor data but must

[19] For more information on the GDOI protocol see RFC 3547 and RFC 6407.

[20] This is normally handled with the use of a group controller / key server, also known as a key distribution center (KDC), used to provide symmetric key coordination between multiple parties, e.g., publishers and subscribers.

properly authenticate and receive the needed keys from the publishing source (or its proxy) before the data can be decrypted.

Data Integrity - The IEC TR 61850-90-5 standard optionally employs the use of a Hash-based Message Authentication Code (HMAC) style checksum in its protocol implementation so that reconstitution of a frame of data at the application layer can be validated when transported over an unreliable data transport, e.g., UDP. Several variants of HMAC algorithms are supported to accommodate the desired balance between accuracy of the checksum and cost of the calculation.

The 90-5 protocol does not specifically dedicate a common synchronization byte in its IEC headers, however, the first byte of any header should be a length indicator (LI) for the type of transport unit marker that follows – which will always be 1 byte. As a result, the first LI will always be 0x01 and this value is immediately followed by the connectionless transport unit marker of 0x40 – these values, in sequence, can be used in lieu of a specified synchronization byte and be searched to help find the start of a data frame when establishing a new parsing session over a lossy protocol. Since many implementations of 90-5 also have transitional support for IEEE C37.118 configuration and command frames, the synchronization byte of 0xAA can be used as well when searching for these frame types.

Bandwidth Utilization

Compared to IEEE C37.118.2-2011, the IEC standard headers add overhead that increase overall bandwidth requirements for IEC TR 61850-90-5 data frames [23] [28]. The IEC header in 90-5 is 45 bytes; then, each data frame contains a prefix of an additional 45 or more bytes before the actual synchrophasor data begins. Moreover, when the feature to repeat past data sets within the current frame is enabled, frame size will grow by data frame size with the configured NOASDU value as the coefficient – which tends to be quixotic for any sizable dataset[16]. Accordingly, enabling use of multiple prior ADSU data sets per frame should only be considered for smaller synchrophasor measurement sets. However, the common deployment use case for 90-5, where the protocol is utilized in a substation within a larger existing IEC 61850 ecosystem, typically already represents a limited, smaller synchrophasor measurement footprint – as such, enabling the prior data set feature in environments with smaller data sets could yield the functionality's original intention of reduced data loss under ideal network conditions at the cost of increased bandwidth.

IX. STTP PROTOCOL OVERVIEW

The Streaming Telemetry Transport Protocol (STTP) is being developed under the DOE Advanced Synchrophasor Protocol (ASP) Development and Demonstration Project (DE-OE0000859). This two-year project began on May 1, 2017 and includes 25 collaborators with the objective to document and demonstrate STTP and to work with standards bodies to put STTP on a track for consideration as a standard protocol.

The proposal to DOE for the ASP project argued that a new standard protocol is needed to overcome the issues being encountered in large-scale synchrophasor data system deployments using existing protocols – specifically with issues of scalability, data loss, bandwidth utilization, data access control, transport security options and cost of configuration management.

STTP leverages the successful design elements of Gateway Exchange Protocol (GEP) that was developed under DOE funded Secure Information Exchange Gateway (SIEGate) Project (DE-OE0000536). GEP was developed for the secure exchange of data necessary to support real-time (i.e., current day) grid operations. The real-time data exchange requirement includes synchrophasor data, SCADA data and file-based data. Although GEP is open source with a permissive license[21] and is widely used by utilities, multi-vendor adoption has been slow. Moreover, regardless of benefits, GEP is not a standard, just an open protocol. Through the ASP project, STTP will build on GEP's successful features and is on track to become a new standard, IEEE 2664.

Protocol Summary

STTP is a data measurement centric, publish/subscribe transport protocol that can be used to securely exchange time-series style data and synchronize metadata among applications. The protocol supports sending real-time and historical data at full or down-sampled resolutions. When sending historical data, the replay speed can be controlled dynamically for use in visualizations to enable users to see data faster or slower than recorded in real-time.

The wire protocol defined by STTP is targeted for packet-based transport protocols, specifically Internet Protocol. STTP implements a publish/subscribe data exchange model using

[21] SIEGate and the GEP protocol are licensed using the MIT license, a short and simple vendor permissive license with conditions only requiring preservation of copyright and license notices.

simple commands with a compressed binary serialization of data points. The protocol does not require a predefined or fixed configuration - that is, the identifiable data point values arriving in one data packet can be different than those arriving in another. Each packet of measurement data consists of a collection of data values where each value is defined by a compact structure containing an identifier; a timestamp or sequence index; a value; and any associated asset state or data quality flags.

STTP is implemented using two different communication channels. STTP calls the first the "command channel" and the second the "data channel". In IP based communication, each of these channels is handled by one or more IP endpoints for sending and receiving data (called a "socket") where the IP transport protocols for these channels can vary based on need. The two most common configurations are: (1) a single TCP transport for both the command and data channel, i.e., traffic for both channels share the same socket, and (2) a TCP based command channel with a UDP based data channel.

The command channel is used to reliably negotiate session specific required communication, state, and protocol parameters. The command channel is also used to manage authentication with other STTP instances, exchange metadata on available data points, and request specific data points for subscription. The data channel is primarily used to send compact, binary encoded packets of data points.

Protocol Structure

Since at the time of writing development of STTP is still a work in progress, some of the serialization details described here may differ from those published when the protocol specification is complete[22]. Furthermore, the process of STTP standardization, through the IEEE P10 working group for P2664, is certain to result in changes to the specification as the group of collaborators expand as part of the standardization process. However, the fundamental invariable tenets of STTP, based on the successful design elements of GEP, are well defined and production-proven. These are:

- Simple command and response architecture

[22] Documentation is focused on serialization details of the beta deliverable of STTP which still closely match those of GEP. However, many protocol improvements are expected that could affect serialization, see the current STTP specification document and test implementation for more information: https://github.com/sttp/.

- Publisher capability to control both data and metadata accessibility for individual subscribers at the measurement level
- Subscribers limited to only the data and metadata they are authorized to receive
- High-volume, high-speed, compact transfer of time-series data from publisher to subscriber with minimal loss through controlled packet sizing
- Ability to encrypt data, specific to subscriber, and use rotating keys

All STTP protocol traffic is composed of simple commands and responses that can each optionally carry a payload. Typically, a subscriber will issue commands and a publisher will answer with responses. Commands from the subscriber include a command type, an optional payload length and optional payload bytes. Responses from the publisher include a response type, an in-response-to command type, optional payload length and optional payload bytes. Often the response type will simply indicate success or failure with an associated message, otherwise the response type will indicate that a specific payload format is in use, e.g., a data packet.

Payloads with Strings - Some of the payloads of command and response messages will be a string or will contain strings. In STTP, all strings are encoded in natural order from left to right, using the character encoding method established in the initial DEFINE OPERATIONAL MODES command, e.g., UTF-8 or UTF-16, see Table 10. STTP Command Types.

STTP Commands - The subscriber command message consists of a command type, an optional payload length, and any payload bytes. The elements that make up the STTP command message are described in the table below:

Field	Byte Size	Description
COMMAND TYPE	1	Command type – see Table 10. STTP Command Types
PAYLOAD LENGTH	0 or 4	Number of bytes of payload, if specified command type includes payload (INT32)
PAYLOAD	VARIABLE	Actual bytes of payload, if any

Table 9. STTP Command Message Fields

The STTP command message types that can be sent by a subscriber and received by a publisher are shown below in Table 10. STTP Command Types. Note that after sending a

solicited command message to the publisher, the subscriber will normally receive a SUCCEEDED or FAILED response message along with an associated message in the payload, i.e., a string of text, detailing the success or failure of the command operation. The payload type for other response successes will be based on the response type. For example, the publisher response for a successful METADATA REFRESH command will be a serialized dataset of the available publisher metadata specifically allowed for the subscriber. The payload content for failed responses will always be a string of text representing the error message. Many command types also require a payload which is specific to the requested command, these are described in APPENDIX C – STTP COMMAND PAYLOADS.

Command	Type	Description	Payload
METADATA REFRESH	0x01	Requests that the publisher send an updated set of metadata so that the subscriber can update its dataset. The successful response message type is a dataset containing the server device and measurement metadata. Devices and measurements contain GUIDs that are used to uniquely identify metadata in local repository allowing datasets to be merged as received from multiple subscriptions.	Optional
SUBSCRIBE	0x02	Requests a subscription of streaming data from the publisher based on a provided connection string. The connection string contains parameters that specify the desired measurements and control if the subscription is for real-time or historical data, when supported. It is not necessary to stop an existing subscription before requesting a new one. The successful response is a message indicating the total number of allowed measurements. Upon successful subscription the subscriber will also receive an UPDATE SIGNAL INDEX CACHE response that will allow parsing of newly subscribed measurement data.	Yes
UNSUBSCRIBE	0x03	Requests that the publisher stop sending streaming data to the subscriber. This command cancels the current subscription.	No
ROTATE CIPHER KEYS	0x04	Requests that the publisher send a new set of cipher keys for encryption of information transmitted on the data channel. The response will include two keys (an old one and a new one) to accommodate any time-slew with transitioning from one key to another. To protect symmetric key exchange, this command should only be used in conjunction with a TLS-based command channel.	No

UPDATE PROCESSING INTERVAL	0x05	Requests that the publisher update its historical data replay processing interval with the specified value in milliseconds – this only applies for a subscription setup to replay historical data, not to a real-time stream. Except for the values of -1 and 0, this value specifies the desired processing interval for data, i.e., basically a delay, or timer interval, over which to process data. A value of -1 means to use the default processing interval while a value of 0 means to process data as fast as possible.	Yes
DEFINE OPERATIONAL MODES	0x06	Defines the protocol version and operational modes for a subscriber connection. As soon as a connection with a publisher is established, this command defines the server operational modes used for subscriber and publisher communication (e.g., compression options or text encoding style). This command can only be used only once per connection and must the first command to be sent by the subscriber to the publisher.	Yes
CONFIRM NOTIFICATION	0x07	This command is sent to a publisher to confirm that a NOTIFY response was received. This is used to verify delivery of critical messages from publisher to subscriber, e.g., a control operation.	Yes
CONFIRM BUFFER BLOCK	0x08	This command is sent to a publisher to confirm that a BUFFER BLOCK response was received. This is used to verify delivery of a data blocks that may require continuity and sequencing, e.g., a file data block.	Yes

Table 10. STTP Command Types

STTP Responses - The publisher response message consists of a response type, an in-response-to command type, payload length and actual payload bytes. The in-response-to command type is required even if the response is unsolicited. In Table 11 below the elements that make up the response message are described:

Field	Byte Size	Description
RESPONSE TYPE	1	Response type – see Table 12. STTP Response Types
IN RESPONSE TO COMMAND TYPE	1	The in-response-to command type – see Table 10. STTP Command Types
PAYLOAD LENGTH	0 or 4	Number of bytes of payload, if specified response type includes payload (INT32)
PAYLOAD	VARIABLE	Actual bytes of payload, if any

Table 11. STTP Response Message Fields

The size of the payload, if any, are specific to the response type. For example, in the case of a data packet response, the payload will contain serialized measurements. Although the subscriber commands and publisher responses will be on two different paths, the value code used for response types are defined as distinct from those used for command types to make it easier to identify values from a wire analysis. Normally response messages are issued in response to subscriber commands, however, they may also act like commands sent by the publisher to the subscriber that were not necessarily solicited – regardless, they are still referred to here as responses for clarity in communicating data flow direction. Many response types also require a payload which is specific to the specified response, these are described in APPENDIX D – STTP RESPONSE PAYLOADS.

Response	Type	Description	Payload
SUCCEEDED	0x80	Informs the client that its solicited server command succeeded, a success message payload follows.	Yes
FAILED	0x81	Informs the client that its solicited server command failed, a failure message payload follows.	Yes
DATA PACKET	0x82	Informs the client that a data packet follows.	Yes
UPDATE SIGNAL INDEX CACHE	0x83	Requests that the client update its runtime signal index cache with the one that follows.	Yes
UPDATE BASE TIMES	0x84	Requests that client update its runtime base-timestamp offsets with those that follow.	Yes
UPDATE CIPHER KEYS	0x85	Requests the client update its runtime symmetric encryption keys with those that follow and use the keys to decrypt data. This only applicable when the data channel is using a UDP socket. Keys should only be transferred used in conjunction with a TLS-based command channel.	Yes
DATA START TIME	0x86	Provides the start time of data being processed from the first measurement.	Yes
PROCESSING COMPLETE	0x87	Provides notification that historical replay processing has completed, established with via temporal constraints parameters, i.e., *StartTimeContraint* and *StopTimeConstraint* connection string parameters.	No
BUFFER BLOCK	0x88	Informs the subscriber of a buffer block (included in payload). This works with a specially defined measurement, still requiring subscription, that allows a free form transfer of data that does not conform to a time-series value. Data in block must still be partitioned to fit within a minimal number of network packets. Subscriber will be required to acknowledge reception of BUFFER BLOCK with a CONFIRM BUFFER	Yes

		BLOCK command since blocks may exist as a sequence of packets and require retransmission when used over a lossy communications transport, e.g., UDP.	
NOTIFY	0x89	Informs the subscriber of a critical notification (included in payload). This works with a specially defined measurement, still requiring subscription, that allows subscriber to receive messages with verified delivery. Since message is considered critical, subscriber must respond with a CONFIRM NOTIFICATION command since message may require retransmission when used over a lossy communications transport, e.g., UDP.	Yes
CONFIGURATION CHANGED	0x8A	Provides a notification that the publisher's source configuration has changed, and that client should make a request to refresh metadata.	No
NO OP	0xFF	Informs the subscriber that communications channel is still active. Since it is possible for the command channel to remain quiet for some time, this command allows a periodic test of continued connectivity.	No

Table 12. STTP Response Types

Protocol Timestamp Format

Timestamps in STTP are encoded as a 64-bit integer representing the number of 100-nanosecond intervals since 0/0/0001, often referred to as a tick. This provides a very high-resolution timestamp, accurate to one ten-millionth of a second with a long year-range, specifically: 00:00:00.0000000 UTC, January 1, 0001 to 23:59:59.9999999 UTC, December 31, 9999, exactly one 100-nanosecond tick before 00:00:00.0000000 UTC, January 1, 10,000 – Gregorian calendar. The 64-bit timestamp is rarely transmitted in its full form since most of the bits representing the time change slowly, see STTP Data Compression.

Protocol Security

STTP requires the use of a TCP-based command channel for actions such as a subscription. The TCP-based command channel is used to reliably negotiate session specific required communication, state, and protocol parameters. It is also used to authenticate with other STTP communications appliances, exchange metadata on points and request points for subscription. This same channel is also used to apply transport layer security (TLS) for publisher/subscriber authentication using public-key cryptography and secure communications. TLS uses X.509 identity certificates for authentication, strong identity verification and encryption. STTP publishers can use a locally accessible subscriber certificate to validate the

identity of a subscriber connection, this can be done in conjunction with a mutually trusted certificate authority or managed privately.

STTP is configurable to allow use of private, i.e., self-signed, certificates in highly isolated environments. In the absence of key management infrastructure, such as deployments with no Internet access or mutually accessible certificate authority, STTP can use self-signed X.509 identity certificates that are securely exchanged between publisher and subscriber, out-of-band, i.e., not on the same communications channel used for data exchange.

When the data is optionally enabled over a UDP socket, the data transmitted on this channel can be encrypted using symmetric encryption keys that are dynamically exchanged over the existing TLS secured TCP-based command channel. When UDP is not used, the data channel information is transmitted over the existing command channel TCP socket.

STTP also incorporates access control at the measurement level. Subscriptions allow for dynamic data and metadata exchange with availability change notifications and the ability to automatically update streaming data values. However, all data and metadata available to a subscriber are subject to publisher discretion and can be changed even when a subscription is active. Although STTP can be configured without its security features enabled, the publisher can always decide if it will allow unsecured connections or data access to unrecognized subscribers. Applications implementing STTP publisher functionality will need to log changes to configuration and administrative actions, which includes data access control, to facilitate compliance with the North American Electric Reliability Corporation (NERC) CIP standards.

Although STTP can be used over a VPN tunnel to provide security, there may be benefits in many deployments to directly using a TLS implementation instead. For one, in order for a publisher to strongly validate the identity of a subscriber, an X.509 certificate will be required even if a VPN tunnel is used to exchange data. As seen in Table 13 - Comparison of STTP / VPN to STTP / TLS Security below, STTP over TLS is an alternative to the use of VPN for securing the transfer of streaming utility operating data.

STTP / VPN APPROACH	STTP / TLS APPROACH
Security managed at the network interface level	Security managed at the application layer with flexible pairwise, i.e., per publisher / subscriber, security
Traffic protected only if it reaches the VPN tunnel – susceptible at previous levels	Traffic is encrypted from source minimizing internal exposure
VPN failure can result in (1) unencrypted data flows, or (2) failure of data flows	Connection failure results in automatic retries, renegotiating keys at each attempt
Network issues may require human intervention to restart data flows	Network issues cause data flows to be automatically reestablished

Table 13 - Comparison of STTP / VPN to STTP / TLS Security

Data Integrity - Unlike the IEEE C37.118.2-2011 and IEC TR 61850-90-5 standards which employ the use of an additional checksum in their protocol implementation, STTP, which is only designed for IP based transport, does not include checksums in its payload since, (1) checksums are already applied at the IP transport layer, and, (2) because payload sizes are targeted to fit within a minimal number of data packets, ideally just one. Since STTP is an IP only protocol and existing data packet checksums are already validated by operating system IP implementations, no extra time or space is allocated to an additional application layer checksum value.

Bandwidth Utilization

Compared to frame-based protocols which include a single timestamp per frame of simultaneously measured data and define a fixed order to identify measurements, the raw binary format for an equivalent group of STTP measurements, where each serialized measurement has its own timestamp and identification, will be always larger. However, production deployments

of STTP never send data in a raw binary form, instead, data is always compressed before transmission. Compression techniques result in TCP based deployments being smaller than IEEE C37.118.2.2011 and UDP based deployments being on-par with IEC TR 61850-90-5 deployments.

STTP Data Compression

Synchrophasor data is comprised of periodic measurements that are recorded at a data sampling rate that is sufficiently high to infer the gradient of change of this data, as such, synchrophasor data is a suitable candidate for compression. Compression algorithms are typically classified into two categories, “lossless” and “lossy”.

As their names infer, lossless compression allows reconstruction, i.e., “inflation”, of the compressed data back to the original source data with full fidelity; whereas inflation of data compressed using a lossy compression algorithm will produce data that is only an approximation of the original data. In terms of computational costs, lossy compression is typically less expensive than lossless compression.

The field of computer science is replete with algorithms for data compression, lossless and lossy, each of which offer tradeoffs in the needed time required to compute the greatest reduction in size. The more CPU time that can be invested into the computation, the better the size reduction; however, the results from increased time investment are non-linear and subject to diminishing returns. Since synchrophasor data is recorded at a high sampling rate and any additional computational activity, like compression, will subsequently increase delivery latency, the available time investment for compression is small and any selected algorithm must be fast and computationally inexpensive.

Although use-cases could be envisioned where a lossy compression algorithm could be tolerated, STTP instead always uses a lossless compression algorithm so that the original data is retained in its full fidelity. The choice of the best lossless compression algorithm to apply depends on the nature of the transmission protocol selected.

When a TCP socket is being used for data channel traffic, it can be assumed that there will be no data packets dropped over a given connection between a publisher and subscriber, as a result, a compression algorithm that works across a longer window of gradually changing waveform data can be selected. When a UDP socket is being used for data channel traffic, the lossless compression algorithm will have no choice but to focus on compression of the data for

an individual packet, not across multiple packets – this is because with UDP, packets can be dropped and the process of lossless inflation of compressed data with common algorithms does not tolerate loss.

An additional compression issue exists with attempting to optimize data packet utilization. With the target compressed size being one network data packet, there can be complications with trying to estimate and balance the total amount of uncompressed source data that will to go into a single compressed data packet as compression ratios depend entirely on the compressibility of the source data with selected compression algorithm. This is often less of a concern for TCP since compressed source data can be easily partitioned into the desired target data packets, but for UDP, the choice is to instead take a conservative approach and try to minimize the number of target frames based on typical compression ratios for the target data and given number of source measurements.

STTP TCP Compression

The current compression algorithm developed for STTP when used over TCP is called Time Series Special Compression (TSSC). Although this is the current compression algorithm in use, it is expected that new algorithms may be developed in the future with better applicability for given use cases, as such STTP requires the flexibility to accommodate new algorithms that can be specified and negotiated when a connection is established without requiring changes to the protocol, see Define Operational Modes Command Payload.

Since STTP can transport most any kind of data, applying a compression algorithm that is used for general purpose data might seem ideal. However, most commonly available lossless streaming compression algorithms, e.g., LZ4, tend to perform worse than simply applying the ubiquitous Gzip algorithm over a single data packet. However, with a little insight into the data being processed it is possible to apply streaming compression rules to specific data elements longitudinally and, based on the nature of the data, produce exceptionally good compression ratios with minimal CPU impact.

TSSC takes each of the elements of a data packet, see Data Packet Response Payload, and handles each data type with separate compression algorithms, creating parallel compression streams for each data element in the data packet. The nature of the data element being compressed then infers the necessary compression algorithm tuning to produce the best results. As an example, with a typical subscription, timestamps tend to be near each other, normally

varying by no more than a few seconds. For the 64-bit timestamps in STTP, this means the data variation may only occur in the bottom 16 of the total 64 bits of the timestamp. With the bulk of the bits repeating invariably, the total bit set needs to be only transmitted once or on substantial change, then only the changing bits need to be sent. Additionally, if the timestamps vary less, the algorithm can automatically adjust and send even fewer bits. This same type of pattern works well for identification numbers, which are finite in number, and state flags, which vary little. Data values, however, need special attention.

Data value elements for a given data packet can seem to be quite random, however, many values change slowly over time, from packet to packet. For example, a measured frequency value tends to change only incrementally over several data packets; in fact, other frequencies in the same subscription may only differ by just a few bits. With this knowledge a table of various base values can be maintained that represent the unvarying bits of many types of measurements. Now only the changed bit values need to be encoded into the stream with enough detail, such as which table entry to use, so that the stream can be losslessly reinflated upon reception.

The actual data going in the data packet payload will now be a chunk of the compressed data stream instead of individual serialized measurements as described in the Data Packet Response Payload. The STTP API will simply now target chunk sizes to meet configured maximum packet size, still fragmenting at the application layer to eliminate the need for buffer reconstruction at the network layer thereby reducing latency.

In practice for synchrophasor data this algorithm has negligible impact on memory and CPU and produces data compression that reduces STTP bandwidth consumption to less than that required by IEEE C37.118 for the same data. However, since the algorithm depends on the evolving states of data over time it can tolerate no intermediate data loss, so a TCP data transport channel is required to use this algorithm.

STTP UDP Compression

Using UDP for data transmission means the possibility exists for data packet loss and, at present, lossless compression algorithms do operate with this caveat. Consequently, the current best option is to apply compression over an individual data packet, ignoring gains that might be acquired over several packets. However, since compressed data is expected to be smaller than

the source data, some tuning can go into how much source data can be used to create the configured maximum packet size.

The current algorithm used for STTP over UDP is the common Gzip algorithm. Application of the compression takes the simple approach of serializing the data packets as normal, see Data Packet Response Payload, then applying Gzip compression over the payload. In practice for synchrophasor data this algorithm has minimal impact on memory and CPU and produces data compression that reduces overall STTP bandwidth consumption; however, even after compression the size is still more than required by IEEE C37.118 for the same data, although comparable to IEC TR 61850-90-5.

The current simple Gzip compression operation does not produce the notable results like those seen with TSSC. However, since a TCP based command channel is always available with STTP, it is likely that a variation of the TSSC algorithm, using a persistent compression table maintained reliably over the command channel, could be developed to produce much better compression ratios when using UDP, perhaps on par with those seen in a TCP only connection.

X. PLANNED TESTING

Bench testing by project participants of the three protocols is planned as part of the ASP project in early 2019. The tests will be conducted on a common set of hardware with a private network. Tests will be executed for scalability, data loss and performance in terms of bandwidth utilization, CPU loading and memory impact. Where necessary, such as with tests for data loss, varying levels of network traffic will be induced on the private network to simulate loading conditions that normally exist in heterogenous IP networks treated as a shared resource. An open source load generator will be used for this testing.

Scalability Testing - Scalability will test maximum data throughput for all protocols, on their own terms, and measure impact on CPU and memory. For example, a nominally sized PMU will be used with IEEE C37.118 to test 10, 100 and the maximum number of PMUs that will fit into a frame and still operate reliably. IEC 61850-90-5 will do the same test, but for varying numbers of ASDU counts (i.e., protocol-specific feature that includes past data sets for current frame in an attempt stave off data loss) from 0 to 4 – note that the maximum number of PMUs will go down per increase in ASDU. STTP will be tested for the same nominally sized PMUs, at comparable scales, but also continuing to increase scale until hardware limitations prevent further reliable throughput.

Ignoring size limitations on the IEEE C37.118 configuration frame 2, because the optional configuration frame 3 can span multiple frames, limits for a data frame of IEEE C37.118 and IEC TR 61850-90-5 are restricted to 65K bytes. As a result, conclusions similar to the indications found in PeakRC testing [24] of GEP and IEEE C37.118 is expected once tests are complete, specifically that STTP will scale to hardware limits, typically 3-5 million points per second depending on hardware, but the 65K limit on IEEE C37.118 and IEC TR 61850-90-5 frames limit data throughput to no more than ~200,000 points per second.

CPU / memory comparisons for all protocols will use the smallest number in the set of max-number of nominally sized PMUs that can be supported by each protocol, as discovered per testing, and compare impacts. As part of scalability testing, comparisons should highlight impacts of STTP compression on memory and CPU loading. Conclusions like the results found in PeakRC testing are expected once tests are complete, specifically that compression does not adversely affect CPU or memory loading.

Data Loss Testing - Data losses will be tested using both TCP only and TCP with UDP. Including TCP only may seem unnecessary since no data loss is expected, however, previous

tests with PeakRC have shown that losses over TCP can still be incurred because of failure to receive all expected data in a configured wait-time – as time allows, multiple TCP tests with varying wait-times will be executed to show the level at which wait-time lengths produce no loss.

UDP tests are expected to highlight data loss minimization effectiveness of STTP. All tests will be executed with varying pre-existing network loads ranging from 0% to 90%. Multiple tests will be executed for varying scalability options – at a minimum three representative tests for small (10 PMUs), medium (50 PMUs) and large (200 PMUs) infrastructures. A traffic impact test of data loss for STTP will also be conducted to show how preexisting network loads affect STTP scale and overall loss (UDP only).

Efficiency Testing - Bandwidth utilization will be tested using both TCP only and TCP with UDP. In the case of TCP only, STTP compression options show bandwidth performance benefits over IEEE C37.118. Conclusions similar to the results found in PeakRC testing are expected once tests are complete, specifically that when compression is enabled, STTP over UDP is expected to perform worse, ~1.8x, than IEEE C37.118, but on par with IEC TR 61850-90-5; and STTP over TCP is expected to perform better that IEEE C37.118, a ~30% reduction.

Virtual Machine Testing - In addition to the tests that will be executed on standalone machines, a subset of the tests, specifically related to scalability and data loss, will also be run on virtual machines (VMs) to ascertain the impact that this technology can have on these protocols. As VMs are a quite common deployment option for IT environments, these tests will highlight any possible negative impacts VMs can have on the reliable transmission of synchrophasor data. VM testing will execute on host hardware where other active VMs will be varied in number and CPU loading on the same host machine.

XI. COMPARISON CONCLUSIONS

This section includes data from the PeakRC testing that compared IEEE C37.118 and GEP, using TCP and UDP, that was conducted in September of 2016 [24]. Testing was conducted at Peak Reliability (PeakRC) in the Vancouver and Loveland operations centers. To simulate a range of operating conditions, the performance of the protocols was evaluated at three data volumes: (1) small scale – simulating a phasor data flow from one of PeakRC's smaller phasor data contributors, (2) medium scale – simulating a phasor data flow from one of PeakRC's bigger phasor data contributors, and (3) large scale – the aggregated PeakRC synchrophasor data stream from all its members. To assure that the protocols were evaluated under identical conditions, all tests were executed simultaneously, side-by-side. Multiple 2-hour tests were run for each data volume to verify that the results were repeatable.

Structure

The IEEE C37.118.2-2011 and IEC TR 61850-90-5 are frame based. While efficient at all data volumes and effective with small data volumes, when used at scale (e.g., for systems involving hundreds of PMUs) the frame-based nature of these protocols present network design and operational challenges that the protocols were never designed to handle. Large frame sizes can also have adverse effects on data completeness; as more devices are concentrated into a single frame of data, the larger frame sizes contribute to higher overall data losses.

STTP is measurement based without a fixed configuration, i.e., the identifiable data arriving in one packet of STTP data will differ from the identifiable data arriving in the next packet and timestamps of data included within a data packet are not necessarily time-aligned. With STTP, data is partitioned at the application layer to minimize network fragmentation at the communications layer. Ideally the number of values sent per partitioned data packet are the total that will conveniently fit into one network packet, i.e., MTU size minus required headers. By reducing network fragmentation, the loss of a single packet over UDP does not constitute the loss of an entire frame of data and retransmission of the packet over TCP is not inducing increased latency and equipment stress due to frame reconstruction.

Efficiency

Because of the extra information required to be transmitted per measurement, the natural (raw) bandwidth requirements of STTP will be higher than a fixed format frame-based protocol such as IEEE C37.118; however, production STTP deployments are always configured with lossless compression. When STTP is used over UDP, each group of measurements is compressed before transmission making the bandwidth requirements more comparable to IEEE C37.118 and other synchrophasor frame-based protocols. Testing showed that after packet-level compression, STTP/UDP is roughly 1.8 times larger than IEEE C37.118 for the same data. However, when using STTP over TCP, stateful compression is used which allows for better time-series based compression over many groups of data resulting in the total bandwidth requirement for STTP/TCP being less than IEEE C37.118. Test results show that STTP is at least 30% smaller than IEEE C37.118 for the same data when using TCP, see Figure 2 below.

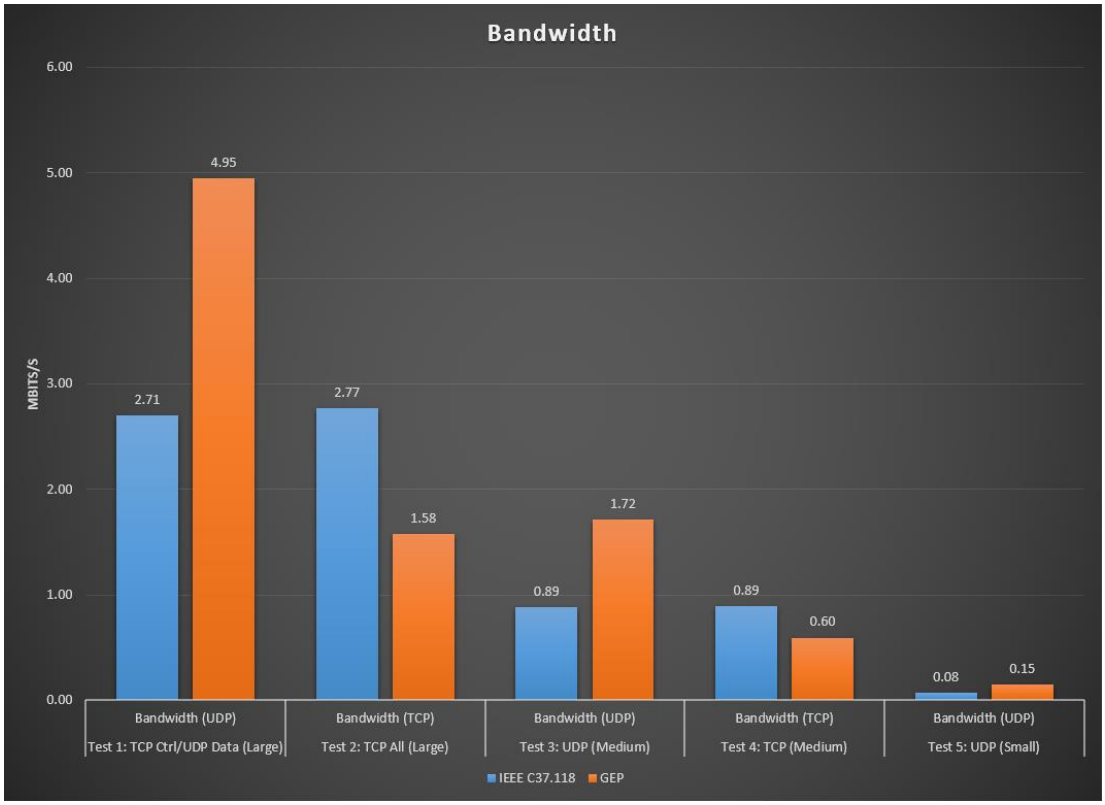

Figure 2. Bandwidth Utilization, IEEE C37.118 vs. GEP

The IEC TR 61850-90-5 protocol has been demonstrated as an alternative to IEEE C37.118.2-2011 [27]. Like IEEE C37.118, the 90-5 protocol is frame-based and for the same

data and has a larger frame size than IEEE C37.118[23]. The 90-5 protocol also includes a feature to repeat past datasets in the current frame to help reduce overall loss when used over UDP, however, this feature should be used judiciously because for large data sets this coefficient quickly causes frame growth which will contribute to increased data loss. Therefore, any existing scalability issues encountered with IEEE C37.118.2-2011 will be exacerbated with IEC 61850-90-5.

Since STTP always applies compression to achieve desired bandwidth utilization, there are legitimate concerns on the impact the compression algorithms will have on both CPU loading and memory utilization. However, since there is no concentration involved with STTP the impacts are significantly less than those compared to IEEE C37.118.2-2011 and IEC TR 61850-90-5 mainly because no frames are being constructed and held in memory, see Figure 3 and Figure 4 below.

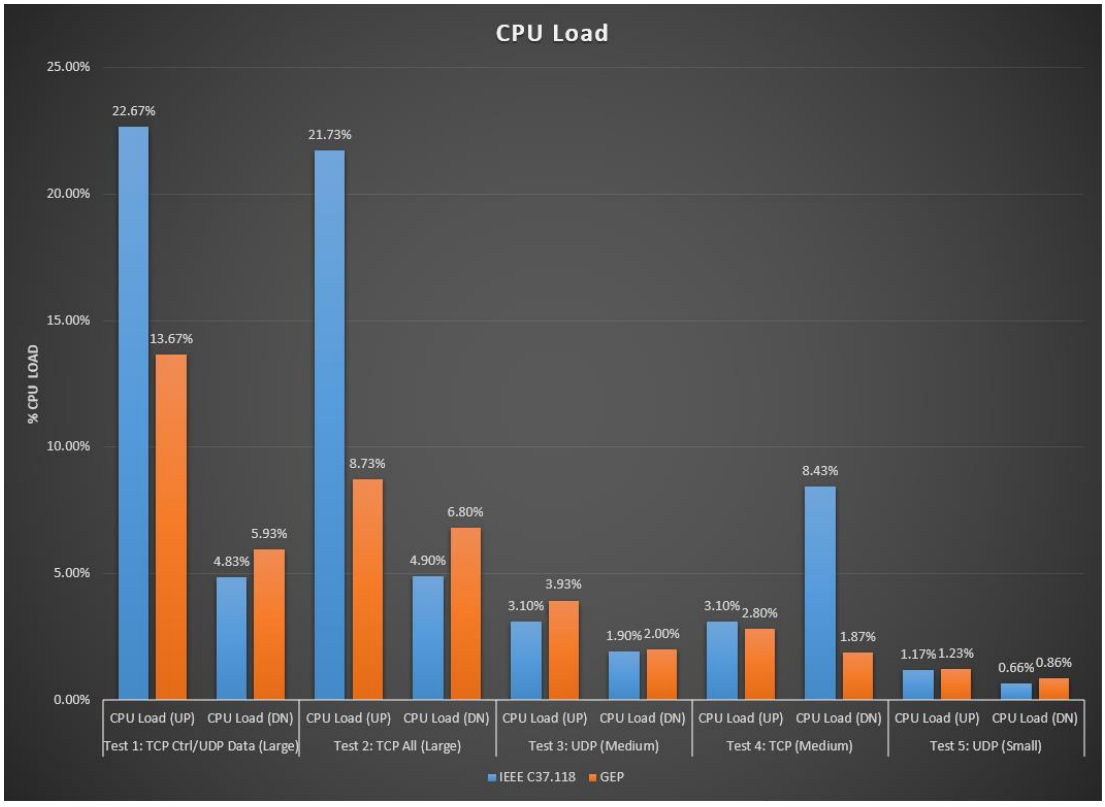

Figure 3. CPU Utilization, IEEE C37.118 vs. GEP

[23] At least 90-bytes more per frame when ASDU count is zero.

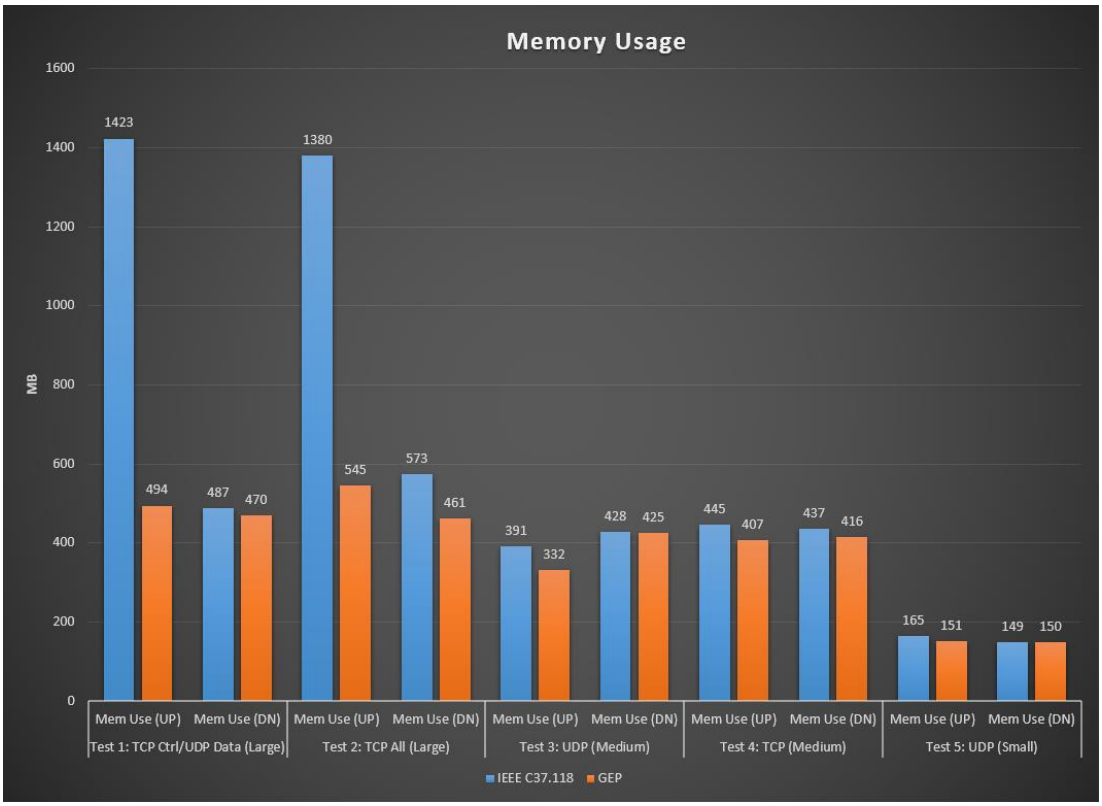

Figure 4. Memory Utilization IEEE C37.118 vs. GEP

Susceptibility to Data Loss

To address the challenges with data loss caused by large frame sizes inherent to the standard IEEE C37.118.2-2011 and IEC TR 61850-90-5 synchrophasor protocols, some utilities have opted to implement purpose-built, dedicated networks exclusively used for synchrophasor traffic [29]. Companies that have not implemented purpose-based networks have also used non-critical network infrastructure, including the internet, to share synchrophasor data due to the fear of over using bandwidth on their respective wide area networks. Although a dedicated network is ideal at reducing data loss (minimizing simultaneous network traffic results in fewer collisions), most networks are a shared resource for many kinds of heterogeneous traffic – in these networks, the continual streaming of large frames of synchrophasor data result in an increased probability of UDP frame loss, or in the case of TCP, increased communication latency due to the higher than normal data packet retransmission rates. Over provisioning of bandwidth on shared networks is a common approach used to resolve issues with sending data with large frame sizes.

In tests conducted by PeakRC, measured overall data loss for the transmission of all of its synchrophasor data using IEEE C37.118 over UDP averaged over 2% when using a data rate of 30 frames per second and more than 3,100 data values per frame, see Figure 5 below.

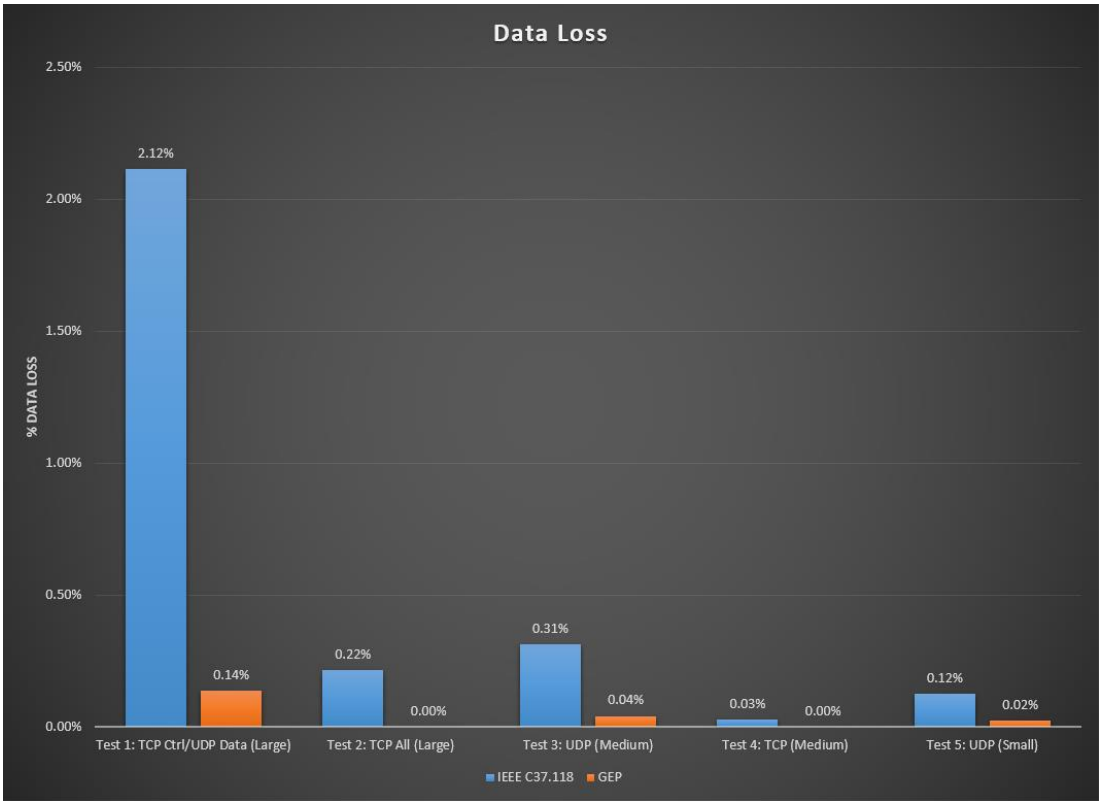

Figure 5. Data Loss IEEE C37.118 vs. GEP

The results from PeakRC testing show that using STTP results in less synchrophasor data loss as compared to other protocols, however, the loss is only significant at scale, i.e., for larger data sets. For the PeakRC large data volume test with UDP with 3,145 measurements published at 30 samples per second, IEEE C37.118 was measured to have 2.1% data loss vs. 0.14% for STTP. Although the data loss for STTP in the smaller datasets is about 6 times less for each, 0.31% data loss vs. 0.04% for the medium data volume and 0.12% data loss vs. 0.02% for the small data volume, the losses are still fractional. Consequently, only when the number of

measurements included in a frame start to create frame sizes that require 20 or more network fragments for IP transmission do the losses become significant, see Figure 6.

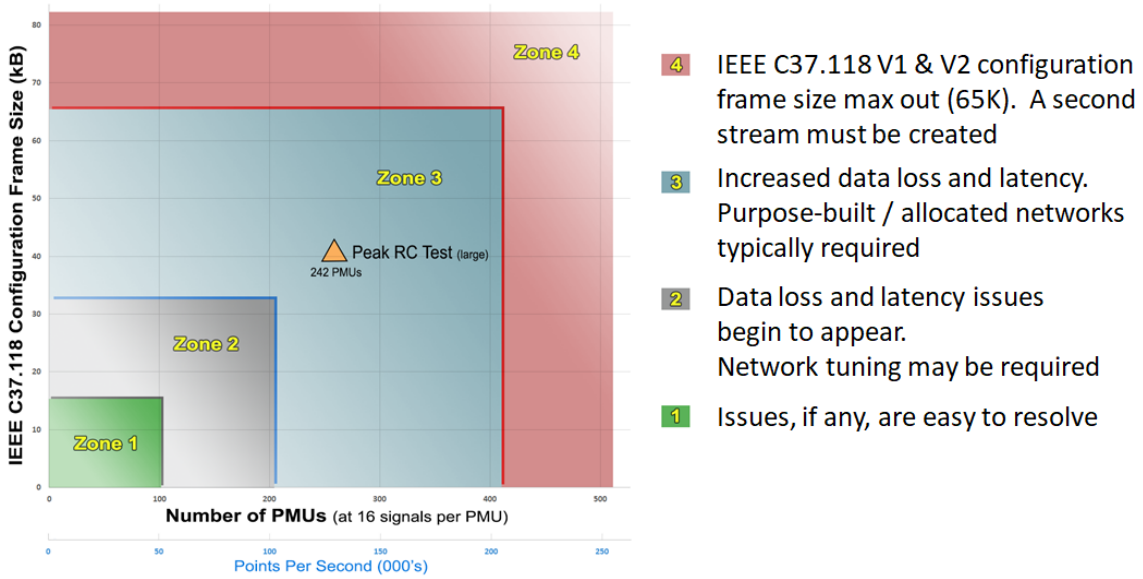

Figure 6. IEEE C37.118 Frame Size as a Function of Signals per PMU

Scalability

Both IEEE C37.118.2-2011 and IEC TR 61850-90-5 have a data frame size ceiling of 65K bytes – this creates a limit for information that can be exchanged between two parties in a single session. For example, with a synchrophasor stream that is publishing data at 30 samples per second, the maximum throughput is about 200,000 measurements per second[24]. The 65K byte limit, as referenced in Zone 4 of Figure 6, means that no more than about 6,700 uniquely identifiable measurements can be published in one stream[25]. In contrast, STTP does not require a fixed configuration nor does it specify a maximum limit on the number of identifiable measurements that can be exchanged – as such, STTP will scale to much higher volumes for data exchange, up to hardware limitations. Current STTP implementations are limited only by CPU processing power used to serialize data. On testing with pedestrian hardware, systems are able to scale from 3 to 5 million measurements per second depending on hardware capabilities.

[24] Increasing the samples per second will increase the throughput, but not the total number of distinctly identified measurements in a configured frame.

[25] In IEEE C37.118.2-2011 using scaled integers, i.e., 2-byte word values instead of 4-byte floating-point values, the total measurements would be doubled to 13,400 measurements.

At 3 million measurements per second, assuming all input sources are streaming at 30 samples per second, a system could support 100,000 distinct identifiable measurements in a single stream. Multi-core server systems should be process 10 to 20 times as much, based on increased core counts, as the constraint is processing based, not network hardware constrained and plenty of remaining bandwidth overhead will exist on Gigabit network infrastructures even at these higher data processing volumes. As an example of need, at current growth rates synchrophasor implementations at the ISO level, with data being received from both members and peers, data volumes could easily approach 25,000 or more distinct measurements within the next few years.

Current synchrophasor deployments at ISO levels are aware of the limits when using IEEE C37.118.2-2011 and IEC TR 61850-90-5; their use forces the creation of multiple streams of data once a frame (typically the configuration, the larger of the frames) reaches its 65K limit. Although creation of multiple streams is a viable option when needing to send all data between two parties using an existing protocol, it is not ideal. Each new stream will need to be on a unique port and new firewall paths will need to be opened and established for each unique stream.

Security

Since the IEEE C37.118.2-2011 protocol has no native security options, this is often a cited reason for using IEC TR 61850-90-5 as an alternative protocol. Certainly in lieu of native security options, many implementations of IEEE C37.118 have instead opted to deploy a VPN between sources and syncs needing to exchange data.

The IEC TR 61850-90-5 standard works with the pre-existing IEC 61850 security options, specifically IEC 62351-9. These security options make use of digital signatures ensuring only authenticated users have access to the desired data using the GDOI protocol. GDOI handles security with UDP multicast security using a shared, group key. This security implementation works out well since IEC TR 61850-90-5 is normally deployed over UDP multicast. For GDOI to function properly, a centrally accessible KDC – i.e., a KDC accessible to both publishers and subscribers – must be available. One of the challenges here is the necessity of establishing a centrally accessible KDC, especially when synchrophasor data will be traversing CIP security zones where each higher security zone will not allow the ingress of connections – ultimately this requires that the KDC be in the least secure zone.

Security in STTP uses standard TLS over TCP. When a UDP channel is enabled, symmetric keys are exchanged over the TLS secured channel to secure the UDP traffic where the keys are specific to each subscriber. If desired and allowed by the publisher, multicast scenarios can also be supported; in this case each subscriber to the multicast stream would each receive the same shared keys after successfully establishing a TLS connection to the publisher. The certificates used in the STTP TLS connections are also used to authenticate and strongly identify subscribers. STTP adds an additional level of security not offered by the other synchrophasor protocols being compared, i.e., measurement-level access control per subscriber as controlled by publisher configuration. In this way a specific subscriber will only have access to data as explicitly authorized by a subscriber. Finally, to accommodate traversal of CIP security zones, STTP supports reverse connection options. A reverse connection allows publishers in a higher security zone to be configured to connect out to subscribers in a lower security zone, which is necessary because initiating a connection in the other direction is otherwise not allowed. Through configuration, a subscriber with a listening socket in a reverse connection will validate the connecting publisher's TLS certificate, but once the connection is established, all other functions proceed as normal.

Non-Synchrophasor Data Transport

The IEEE C37.118.2-2011 and IEC TR 61850-90-5 both accommodate the transmission of other, non-synchrophasor data using "analog" values, such as calculated data. Transmission of analog values is always within an existing frame along with other synchrophasor data at a specific timestamp. Publication of extra data at predefined timestamp creates a caveat that the analog values, ideally, should be published at the same rate as other synchrophasor data. Otherwise, if the publication frequency, i.e., the rate of calculation or measurement, of an analog value is less than the configured publication frequency of the host frame, a sentinel value will need to occupy the space in the frame for publication periods with no measurement, e.g., a not-a-number (NaN) value. Publishing a sentinel value means that bandwidth is being unnecessarily consumed in cases where there is no data. Also, using a sentinel value can create a dilemma for meaning as NaN may represent a valid result for a calculated analog value. If the publication frequency of an analog value is higher than the configured publication frequency of the host frame, then the measured values will need to be down-sampled into the target frequency. In

either case, when the publication frequencies differ, the situation for analog value transport in a frame-based protocol is not ideal.

By contrast, in STTP each individual measurement can have its own publication frequency. This is easily accommodated because each STTP data packet does not have a fixed configuration and every measured value has its own associated identification, timestamp, and quality flags, see Data Packet Response Payload. The per measurement quality flags allow for an unambiguous representation of a value's meaning – that is, the value is always exactly what is measured or calculated with no sentinels required and the flags convey known state about the analog value, such as its quality or the accuracy of the timestamp.

Other Operating Functionality

Existing frame-based synchrophasor protocols only have prescriptive methods for the management of measurement metadata. While this prescriptive method can be well-suited for substation-to-control-center use, it becomes difficult to manage as measurement metadata that spans multiple analytic solutions and control centers, for example, inter-company data exchange where it becomes difficult to describe data when measurements with shared configuration owners or in a wide-area context, due to merging complexities. To help with merging disparate data sources, STTP allows for extensible metadata sets so that industry specific information about the data being exchanged can be included and all STTP metadata is identified with 128-bit statistically unique GUID values to support dataset conflation, see APPENDIX B – STTP METADATA.

Protocol Comparison Summary

A table summarizing features of the three compared protocols is provided below:

FEATURE	IEEE C37.118	IEC 61850 90-5	STTP
Structure	Frame	Frame	Dynamic
Efficiency	Good	Fair	Excellent - TCP Fair - UDP
Data Loss (low volume)	None - TCP	None - TCP	None
Data Loss (high volume)	Low - TCP[26] Some - UDP[27]	Low - TCP Some - UDP	None - TCP Minimal - UDP
Scalability	Fair	Fair	Excellent
Encryption	No	Yes	Yes
Extensible Metadata	No	No[28]	Yes
Multicast Supported	Yes	Yes	Limited

Table 14. Protocol Comparison Summary

[26] Latencies with large data sets when using concentration can cause timeout expirations such that source data will to not be included in final output stream thus inducing some data loss even when using TCP.

[27] Measured to be at 2% during PeakRC testing [24].

[28] SCL can be mapped to CIM giving deep insights into utility infrastructure, however, the protocol does not allow for the exchange of dynamic datasets that may be required for other industries.

XII. REFERENCES

[1] K. E. Martin, et. al., "Exploring the IEEE Standard C37.118–2005 Synchrophasors for Power Systems," IEEE Transactions on Power Delivery, vol. 23, no. 4, 2008.

[2] S. Thakur and A. Chakrabortty, "Multi-dimensional wide-area visualization of power system dynamics using Synchrophasors," In 2013 IEEE Power & Energy Society General Meeting, Vancouver, BC, Canada, 2013.

[3] Schweitzer III, Edmund O.; Whitehead, David; Zweigle, Greg; Ravikumar, Krishnanjan Gubba; Rzepka , Greg; Schweitzer Engineering Laboratories, Inc., "Synchrophasor-Based Power System Protection and Control Applications, Wroclaw, Poland, 2010.

[4] "C37.118.1a-2014 - IEEE Standard for Synchrophasor Measurements for Power Systems -- Amendment 1: Modification of Selected Performance Requirements, IEEE, 2014.

[5] V. Madani and J. R. Carroll, "IEC 61850-90-5 Implementation at PG&E," Knoxville, TN, 2014.

[6] Microsoft, "The default MTU sizes for different network topologies, Article ID: 314496," 19 June 2014. [Online]. Available: https://support.microsoft.com/en-us/help/314496/the-default-mtu-sizes-for-different-network-topologies.

[7] V. Popeskic, "Collisions and collision detection - What are collisions in Ethernet?," 16 November 2011. [Online]. Available: https://howdoesinternetwork.com/2011/collisions.

[8] H. K. Huang, T. Suda and Y. Noguchi, "LAN with collision avoidance: switch implementation and simulation study," In 15th Conference on Local Computer Networks, Minneapolis, MN, USA, 1990.

[9] P. M. Dickens, J. W. Larson and D. M. Nicol, "Diagnostics for causes of packet loss in a high performance data transfer system," In 18th International Parallel and Distributed Processing Symposium, Santa Fe, NM, USA, USA, 2004.

[10] M. S. Borella, "Measurement and interpretation of Internet packet loss," IEEE Journal of Communications and Networks, Vol. 2, Issue 2, pp. 93-102, June 2000.

[11] M. Seewald, "Building an architecture based on IP-Multicast for large phasor measurement unit (PMU) networks," Washington, DC, 2013.

[12] H. Holbrook, B. Cain, Arastra, Inc. and Acopia Networks, RFC 4607: Source-Specific Multicast for IP,' Internet Engineering Task Force - Network Working Group, 2006.

[13] L. Frenzel, Serial I/O Interfaces Dominate Data Communications,' 22 September 2015. [Online]. Available: https://www.electronicdesign.com/communications/serial-io-interfaces-dominate-data-communications.

[14] W. T. Shaw, Cybersecurity for SCADA Systems, PennWell Books, 2006, pp. 335-336.

[15] Jimb0, RS-232 vs. TTL Serial Communication,' 23 November 2010. [Online]. Available: https://www.sparkfun.com/tutorials/215. [Accessed 16 August 2018].

[16] J. C. R. Bennett, C. Partridge and N. Shectman, Packet Reordering is Not Pathological,' IEEE/ACM Transactions on Networking, Vol. 7, No. 6, pp. 789-798, December 1999.

[17] C. a. M. C37.244-2013 - IEEE Guide for Phasor Data Concentrator Requirements for Power System Protection, 2013.

[18] C. A. Kent and J. C. Mogul, Fragmentation Considered Harmful,' Proceedings of Frontiers in Computer Communications Technology, Stowe, Vermont, 1987.

[19] Google, Protocol Buffers - Techniques - Large Data Sets,' 14 December 2017. [Online]. Available: https://developers.google.com/protocol-buffers/docs/techniques#large-data.

[20] Apache Software Foundation, Thrift Remote Procedure Call - Protocol considerations - Framed vs. unframed transport,' 21 September 2016. [Online]. Available: https://github.com/apache/thrift/blob/master/doc/specs/thrift-rpc.md#framed-vs-unframed-transport.

[21] E. Brosh, S. A. Baset, V. Misra, D. Rubenstein and H. Schulzrinne, The Delay-Friendliness of TCP,' Columbia University, NY, USA, 2010.

[22] RAPIR Task Force, Real-Time Application of Synchrophasors for Improving Reliability,' 2010.

[23] R. Khan, K. McLaughlin, D. Laverty and S. Sezer, Analysis of IEEE C37.118 and IEC 61850-90-5 Synchrophasor Communication Frameworks,' Boston, USA, 2016.

[24] Peak Reliability, PRSP Phasor Gateway Evaluation Report,' NASPI - https://www.naspi.org/naspi/sites/default/files/2017-03/PRSP_Phasor_Gateway_Whitepaper_Final_with_disclaimer_Final.pdf, Vancouver,WA, 2016.

[25] H. Falk, "IEC 61850-90-5 - an Overview (https://www.pacw.org/issue/december_2012_issue/iec_61850905_an_overview/iec_61850905_an_overview/complete_article/1.html)," PAC World, no. December, 2012.

[26] S. R. Firouzi, "Design, Implementation and Validation of an IEC 61850-90-5 Gateway for IEEE C37.118.2 Synchrophasor Data Transfer," Escola Tecnica Superior d'Enginyeria Industrial de Barcelona, Barcelona, Spain, 2015.

[27] V. Madani, R. Farquharson, M. Adamiak, R. Mackiewicz, H. Falk, C. Brunner and F. Rahmatian, "NASPI Synchrophasor Technical Report - NASPI IEC 61850-90-5 Tutorial," smartgrid.gov, Atlanta, GA, 2016.

[28] I. Ali, M. A. Aftab and S. M. S. Hussain, "Performance comparison of IEC 61850-90-5 and IEEE C37.118.2 based wide area PMU communication networks," Journal of Modern Power Systems and Clean Energy, vol. 4, no. 3, pp. 487-495, 2016.

[29] D. Kosterev, L. Carter and S. Lissit, "The Pulse of the Grid," vol. 66, no. 1, 2013.

XIII. APPENDIX A – STTP FILTER EXPRESSIONS

Subscribers use filter expressions to condense XML-based datasets, e.g., metadata, provided by STTP publishers to a desired set of information. The metadata defined by STTP is in tabular format – such as the data from tables in a database – so the syntax used for filter expressions is like that defined in Structured Query Language (SQL). Filter expressions focus on operations that work in a similar fashion to the SQL expression syntax associated with the WHERE clause in a SELECT statement but does not implement the full set of SQL language options for the clause.

Limited Implementation Requirements

The purpose of a filter expression is to reduce the set of available metadata down to the desired set of values, as such, any of the more complex SQL WHERE expression operations, like those that use aggregation functions or parent-child foreign-key relationships, need not be implemented.

At a minimum, filter expressions should allow string literals demarked by single quotes, e.g., `'String Literal'`, and numeric literals. Expressions should also support comparison operators of `=`, `<>`, `<`, `<=`, `>`, and `>=`, where strings comparisons operate as case-insensitive in the locale of the host system. A SQL `LIKE` expression should so also be supported with an `*` or `%` representing a wildcard for pattern matching that can be at the beginning of a pattern `'*value'`, at the end `'value*'`, or at both `'*value*'`. A wildcard in the middle of a pattern `'va*lue'` is not allowed. Boolean operators `AND`, `OR` and `NOT` should be supported to allow concatenation of expressions. The `NOT` operator has precedence over the `AND` operator and it has precedence over the `OR` operator. Other operators, such as standard SQL arithmetic operations are encouraged, but not required.

Implementations of STTP are encouraged not to pass filter expressions along to an actual database engine for parsing since this could introduce security issues on the host database related to SQL injection. Instead implementations should a use a custom SQL expression parser, with limited implementation, against a read-only or in-memory dataset.

Syntax

The filtering syntax uses the custom key word `FILTER` instead of the standard SQL `SELECT` so that it is clear to the user that the operation is not a standard SQL operation and hence restriction to capabilities of the expression may apply. The structure of an expression should be as follows:

```
FILTER <TableName> [TOP n] WHERE <Expression> [ORDER BY <SortField>]
```

The following table defines the options and clauses that make up the STTP filter expression:

Keyword	Example	Description	Required
`FILTER`	See Examples below	Starts the filter expression	Yes
`TOP n`	`TOP 100`	Selects only the first number of items	No
`WHERE <Expression>`	`WHERE SignalType='FREQ'`	SQL `WHERE` expression with limited functionality support	Yes
`ORDER BY <ColumnName>`	`ORDER BY SignalType`	Orders the results by the selected field	No

Table 15. STTP Filter Expression Structure

Examples

The following filter expression will request to subscribe to the first 20 measurements with the company name of BPA and signal type of frequency (FREQ):

```
FILTER TOP 20 ActiveMeasurements WHERE Company='BPA' AND SignalType='FREQ'
```

The following filter expression will request to subscribe to only positive sequence current and voltage phase angles:

```
FILTER ActiveMeasurements WHERE SignalType IN ('IPHA','VPHA') AND Phase='+'
ORDER BY PhasorID
```

The following filter expression will request to filter incoming metadata to exclude statistics, such as might be used with the METADATA REFRESH command payload:

```
FILTER MeasurementDetail WHERE SignalAcronym <> 'STAT'
```

XIV. APPENDIX B – STTP METADATA

Metadata in STTP is XML data that is represented in a tabular format, specifically with an XML Schema (https://www.w3.org/TR/xmlschema-2/), much like that of data that can be found in a spreadsheet or table of data in a database, see Example Metadata below. Additionally, STTP allows for the transport of multiple tables of data in a single dataset, such as, a table of devices and measurements. This flexibility allows for the transport of any number of tables of metadata that may be required to describe measurements in enough detail to accommodate a given use case. Regardless, a minimum set of metadata is required for STTP to function, specifically all measurements need a unique GUID based identifier along with its data type, an alpha-numeric tag name, description, and last update timestamp.

Metadata Tables and Fields

The following tables detail the metadata that are currently used in STTP, specific for the utility industry, that allow for interoperability with other synchrophasor protocols such as IEEE C37.118.

MeasurementDetail Table

At a minimum, STTP will require a table of measurements in order to function. Of the fields in this table, only four fields are required, that is: SignalID, PointTag, Description and UpdatedOn. All other fields are optional from the perspective of STTP but may be required for an industry specific use case.

Field	Type	Description	Required
SignalID	Guid	Unique UUID of this individual measurement	Yes
PointTag	String	Well formatted tag name for historians	Yes
Description	String	Detailed measurement description (free-form)	Yes
ID	String	Measurement key string, format: "source:index"	No
SignalReference	String	Frame-based protocol mapping field (type / index)	No
PhasorSourceIndex	Integer	Phasor ordered index, uses 1-based indexing	No
DeviceAcronym	String	Name of associated parent device (if any)	No
UpdatedOn	DateTime	Time of last meta-data update	Yes

DeviceDetail Table

The device table defines the devices, such as PMUs, that are the sources of measurements. This table is useful for mapping STTP to and from IEEE C37.118.

Field	Type	Description
Acronym	String	Alpha-numeric device, e.g., pmu/station name (all-caps)
Name	String	User-defined device name / description (free-form)
UniqueID	Guid	Device unique UUID (used for IEEE C37.118 CFG-3 frame)
AccessID	Integer	ID code used for device connection / reference
ParentAcronym	String	Original PDC name (none for direct connected devices)
ProtocolName	String	Original protocol name
FramesPerSecond	Integer	Device reporting rate, e.g., 30 fps
CompanyAcronym	String	Original device company name
VendorAcronym	String	Original device vendor name (if provided)
VendorDeviceAcronym	String	Original vendor device name, e.g., PMU brand
Longitude	Float	Device longitude (if reported)
Latitude	Float	Device latitude (if reported)
UpdatedOn	DateTime	Time of last meta-data update

PhasorDetail Table

The phasor table defines phasors, as described in IEEE C37.118, that are associated with devices. This table is required to construct a frame of data in IEEE C37.118 format using input measurements received from STTP.

Field	Type	Description
DeviceAcronym	String	Name of associated parent device (required)
Label	String	Phasor label (16-characters) for CHNAM
Type	String	Current (I) or Voltage (V)
Phase	String	Phase, e.g., A, B, C, +, -, 0
SourceIndex	Integer	Phasor ordered index, uses 1-based indexing
UpdatedOn	DateTime	Time of last meta-data update

Example Metadata

```
<?xml version="1.0" standalone="yes"?>
<Metadata>
  <xs:schema id="Metadata" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Metadata">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="DeviceDetail">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="NodeID" type="xs:string" minOccurs="0" />
                <xs:element name="UniqueID" type="xs:string" minOccurs="0" />
                <xs:element name="OriginalSource" type="xs:string" minOccurs="0" />
                <xs:element name="IsConcentrator" type="xs:boolean" minOccurs="0" />
                <xs:element name="Acronym" type="xs:string" minOccurs="0" />
                <xs:element name="Name" type="xs:string" minOccurs="0" />
                <xs:element name="AccessID" type="xs:long" minOccurs="0" />
                <xs:element name="ParentAcronym" type="xs:string" minOccurs="0" />
                <xs:element name="ProtocolName" type="xs:string" minOccurs="0" />
                <xs:element name="FramesPerSecond" type="xs:long" minOccurs="0" />
                <xs:element name="CompanyAcronym" type="xs:string" minOccurs="0" />
                <xs:element name="VendorAcronym" type="xs:string" minOccurs="0" />
                <xs:element name="VendorDeviceName" type="xs:string" minOccurs="0" />
                <xs:element name="Longitude" type="xs:decimal" minOccurs="0" />
                <xs:element name="Latitude" type="xs:decimal" minOccurs="0" />
                <xs:element name="InterconnectionName" type="xs:string" minOccurs="0" />
                <xs:element name="ContactList" type="xs:string" minOccurs="0" />
                <xs:element name="Enabled" type="xs:boolean" minOccurs="0" />
                <xs:element name="UpdatedOn" type="xs:dateTime" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="MeasurementDetail">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="DeviceAcronym" type="xs:string" minOccurs="0" />
                <xs:element name="ID" type="xs:string" minOccurs="0" />
                <xs:element name="SignalID" type="xs:string" minOccurs="0" />
                <xs:element name="PointTag" type="xs:string" minOccurs="0" />
                <xs:element name="SignalReference" type="xs:string" minOccurs="0" />
                <xs:element name="SignalAcronym" type="xs:string" minOccurs="0" />
                <xs:element name="PhasorSourceIndex" type="xs:long" minOccurs="0" />
                <xs:element name="Description" type="xs:string" minOccurs="0" />
                <xs:element name="Internal" type="xs:boolean" minOccurs="0" />
                <xs:element name="Enabled" type="xs:boolean" minOccurs="0" />
                <xs:element name="UpdatedOn" type="xs:dateTime" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="PhasorDetail">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ID" type="xs:long" minOccurs="0" />
                <xs:element name="DeviceAcronym" type="xs:string" minOccurs="0" />
                <xs:element name="Label" type="xs:string" minOccurs="0" />
                <xs:element name="Type" type="xs:string" minOccurs="0" />
                <xs:element name="Phase" type="xs:string" minOccurs="0" />
                <xs:element name="DestinationPhasorID" type="xs:long" minOccurs="0" />
                <xs:element name="SourceIndex" type="xs:long" minOccurs="0" />
```

```
              <xs:element name="UpdatedOn" type="xs:dateTime" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="SchemaVersion">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="VersionNumber" type="xs:long" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
<DeviceDetail>
  <NodeID>8736f6c7-ad41-4b43-b4f6-e684e0d4ad20</NodeID>
  <UniqueID>c8283c22-8aed-4a0d-bf9d-76111932afd9</UniqueID>
  <IsConcentrator>false</IsConcentrator>
  <Acronym>TESTDEVICE</Acronym>
  <Name>Test Device</Name>
  <AccessID>2</AccessID>
  <ParentAcronym />
  <ProtocolName>IEEE 1344-1995</ProtocolName>
  <FramesPerSecond>30</FramesPerSecond>
  <CompanyAcronym>TVA</CompanyAcronym>
  <VendorAcronym>ABB</VendorAcronym>
  <VendorDeviceName>ABB-521</VendorDeviceName>
  <Longitude>-89.8038</Longitude>
  <Latitude>35.3871</Latitude>
  <InterconnectionName>Eastern Interconnection</InterconnectionName>
  <ContactList />
  <Enabled>true</Enabled>
  <UpdatedOn>2018-03-14T19:23:10.321-04:00</UpdatedOn>
</DeviceDetail>
<MeasurementDetail>
  <DeviceAcronym>TESTDEVICE</DeviceAcronym>
  <ID>PPA:1</ID>
  <SignalID>29b80c23-b76e-4f3d-a0bd-855b0f8ef08d</SignalID>
  <PointTag>TVA_TESTDEVICE:ABBS</PointTag>
  <SignalReference>TESTDEVICE-SF</SignalReference>
  <SignalAcronym>FLAG</SignalAcronym>
  <Description>Test Device ABB-521 Status Flags</Description>
  <Internal>true</Internal>
  <Enabled>true</Enabled>
  <UpdatedOn>2018-03-14T19:23:11.477-04:00</UpdatedOn>
</MeasurementDetail>
<MeasurementDetail>
  <DeviceAcronym>TESTDEVICE</DeviceAcronym>
  <ID>PPA:2</ID>
  <SignalID>285dfa57-7b51-47b5-919b-b1dc7140d01a</SignalID>
  <PointTag>TVA_TESTDEVICE:ABBF</PointTag>
  <SignalReference>TESTDEVICE-FQ</SignalReference>
  <SignalAcronym>FREQ</SignalAcronym>
  <Description>Test Device ABB-521 Frequency</Description>
  <Internal>true</Internal>
  <Enabled>true</Enabled>
  <UpdatedOn>2018-03-14T19:23:11.665-04:00</UpdatedOn>
</MeasurementDetail>
<MeasurementDetail>
  <DeviceAcronym>TESTDEVICE</DeviceAcronym>
```

```
    <ID>PPA:5</ID>
    <SignalID>977747f8-056f-4fc6-88b2-f0cfb51ec139</SignalID>
    <PointTag>TVA_TESTDEVICE-BUS1:ABBV</PointTag>
    <SignalReference>TESTDEVICE-PM1</SignalReference>
    <SignalAcronym>VPHM</SignalAcronym>
    <PhasorSourceIndex>1</PhasorSourceIndex>
    <Description>Test Device Bus 1 Positive Sequence Voltage Magnitude</Description>
    <Internal>true</Internal>
    <Enabled>true</Enabled>
    <UpdatedOn>2018-03-14T19:23:12.227-04:00</UpdatedOn>
  </MeasurementDetail>
  <MeasurementDetail>
    <DeviceAcronym>TESTDEVICE</DeviceAcronym>
    <ID>PPA:6</ID>
    <SignalID>952ed494-377f-4be3-9151-86c78ead9231</SignalID>
    <PointTag>TVA_TESTDEVICE-BUS1:ABBVH</PointTag>
    <SignalReference>TESTDEVICE-PA1</SignalReference>
    <SignalAcronym>VPHA</SignalAcronym>
    <PhasorSourceIndex>1</PhasorSourceIndex>
    <Description>Test Device Bus 1 Positive Sequence Voltage Phase Angle</Description>
    <Internal>true</Internal>
    <Enabled>true</Enabled>
    <UpdatedOn>2018-03-14T19:23:12.415-04:00</UpdatedOn>
  </MeasurementDetail>
  <MeasurementDetail>
    <DeviceAcronym>TESTDEVICE</DeviceAcronym>
    <ID>PPA:4</ID>
    <SignalID>98c93a54-9435-48cf-987d-e897b793441a</SignalID>
    <PointTag>TVA_TESTDEVICE:ABBDF</PointTag>
    <SignalReference>TESTDEVICE-DF</SignalReference>
    <SignalAcronym>DFDT</SignalAcronym>
    <Description>Test Device ABB-521 Frequency Delta (dF/dt)</Description>
    <Internal>true</Internal>
    <Enabled>true</Enabled>
    <UpdatedOn>2018-03-14T19:23:12.04-04:00</UpdatedOn>
  </MeasurementDetail>
  <MeasurementDetail>
    <DeviceAcronym>TESTDEVICE</DeviceAcronym>
    <ID>STAT:35</ID>
    <SignalID>13f86d22-bf54-4ceb-b345-6a423118a1bc</SignalID>
    <PointTag>TVA_TESTDEVICE!IS:ST1</PointTag>
    <SignalReference>TESTDEVICE!IS-ST1</SignalReference>
    <SignalAcronym>STAT</SignalAcronym>
    <Description>Total frames received during last reporting interval.</Description>
    <Internal>true</Internal>
    <Enabled>true</Enabled>
    <UpdatedOn>2018-03-20T22:12:26.933-04:00</UpdatedOn>
  </MeasurementDetail>
  <PhasorDetail>
    <ID>12</ID>
    <DeviceAcronym>TESTDEVICE</DeviceAcronym>
    <Label>500 kV Bus 1</Label>
    <Type>V</Type>
    <Phase>+</Phase>
    <SourceIndex>1</SourceIndex>
    <UpdatedOn>2018-03-14T19:23:10.509-04:00</UpdatedOn>
  </PhasorDetail>
  <SchemaVersion>
    <VersionNumber>8</VersionNumber>
  </SchemaVersion>
</Metadata>
```

XV. APPENDIX C – STTP COMMAND PAYLOADS

The following are the payload definitions associated with specific STTP command types, see Table 10. STTP Command Types.

Metadata Refresh Command Payload

The payload for the METADATA REFRESH command is optional. If the payload length is zero, no filters are applied to the publisher metadata and all records that the subscriber is authorized to view will be returned in the dataset. When a payload is specified, i.e., the payload length is greater than zero, the payload allows the subscriber to provide text-based filtering expressions to reduce the metadata dataset provided by the publisher. For example, the subscriber may want to reduce the metadata to records for a certain company or signal type to reduce the number of records received. The format of the filter expression is defined in APPENDIX A – STTP FILTER EXPRESSIONS. Multiple expressions can be defined, each separated by a semi-colon. Below are the elements that make up the optional metadata refresh command payload:

Field	Byte Size	Description
EXPRESSION LENGTH	4	Number of bytes in the expression
EXPRESSION	EXPRESSION LENGTH	String-based filter expressions separated by semi-colons

Table 16. Metadata Refresh Command Payload Fields

Subscribe Command Payload

The payload for the SUBSCRIBE command defines the desired data packet options and connection string expression used to start data streaming from the publisher. Requests for

measurements that do not exist or are not allowed by the publisher for the subscriber will be ignored. Below are the elements that make up the subscribe command payload:

Field	Byte Size	Description
DATA PACKET OPTIONS	1	Currently a fixed value of 0x02 specifying compact measurement format, no synchronization[29]
EXPRESSION LENGTH	4	Number of bytes in the key/value pair connection string expression
EXPRESSION	EXPRESSION LENGTH	String-based key/value pair connection string expression that defines desired subscription parameters, see Table 18. Subscribe Command Connection String Parameters

Table 17. Subscribe Command Payload Fields

The connection string expression used in the subscribe command is formatted as a series of key/value pairs where an equals-sign (=) separates the key and value and a semi-colon (;) separates the pairs. The key names are not case-sensitive. The available keys and value types for the subscription are defined as follows:

Key[30]	Value Type	Description
TrackLatestMeasurements	Boolean	Enables measurement down-sampling, speed controlled by PublishInterval. Defaults to false if not specified.
PublishInterval	Double	The interval, in seconds, over which to deliver streaming data. Only applicable when TrackLatestMeasurements = true.
IncludeTime	Boolean	Determines if measurements should include timestamp. Defaults to true if not specified.

[29] Future implementations of STTP will drop this value. GEP supports requests for a non-compact measurement format to receive measurements using a simple serialization, however, this option is never used in practice. Another GEP subscription option exists to request server-side pre-concentration of data which delivers measurements in time-sorted order – however, the publisher has the right to deny this, and by default does, because it induces extra memory and CPU burden per subscription. The decision made was that if subscribers need data in a time-sorted manner, it will need to be concentrated locally post reception.

[30] These key names are subject to change with ongoing STTP updates and improvements. STTP API implementations typically hide these key/value details through properties and configuration parameters.

ProcessingInterval	32-bit Integer	Specifies the initial historical data replay processing interval to use, in milliseconds. Only applicable when StartTimeContraint and StopTimeConstraint are provided. A value of -1 means to use the default processing interval and a value of 0 means to process data as fast as possible. Defaults to -1 if not specified.
UseMillisecondResolution	Boolean	Requests that millisecond time resolution be used for subscription to use less space when encoding timestamps. Defaults to false if not specified.
RequestNaNValueFilter	Boolean	Determines if values that contain not-a-number (NaN) should be published. Defaults to false if not specified, meaning values that NaN will be delivered to subscriber.
InputMeasurementKeys	String	Defines the list of measurement identifiers or filter expression that the subscriber desires for subscription. When a list of identifiers is provided – either the GUID based signal ID, measurement key string, or point tag – the values are separated by a semi-colon. See APPENDIX A – STTP FILTER EXPRESSIONS for more details on filter expressions.
StartTimeConstraint	String	When supported by the publisher, defines the start time, in UTC, for a subscription used to start a historical subscription. Time format can be absolute, e.g., 12-30-2000 23:59:59.033, or relative, e.g., *-20s meaning start twenty seconds before current time. For relative time specifications an “s” suffix is for seconds, “m” is for minutes, “h” is for hours and “d” is for days.
StopTimeConstraint	String	When supported by the publisher, defines the stop time, in UTC, for a subscription used to mark the end of a historical subscription. See StartTimeConstraint for time format options. When historical replay is complete, subscriber will receive a PROCESSING COMPLETE response.

Table 18. Subscribe Command Connection String Parameters

Update Processing Interval Command Payload

The payload for the UPDATE PROCESSING INTERVAL command defines the new processing interval to apply to an ongoing historical data replay subscription. This command can be issued at any time while historical data is streaming back to the subscriber, this allows dynamic control of the speed of the replay. Except for the values of -1 and 0, the new value specifies the desired processing interval for data in milliseconds. Implementations of STTP use the interval to induce a delay in historical replay. A value of -1 means to use the default processing interval while a value of 0 means to process data as fast as possible. Below are the elements that make up the update processing interval command payload:

Field	Byte Size	Description
VALUE	4	New interval to apply (INT32)

Table 19. Update Processing Interval Command Payload Fields

Define Operational Modes Command Payload

The payload for the DEFINE OPERATIONAL MODES command defines the desired protocol version and bit flags used to set up the desired communication rules between the publisher and subscriber. As soon as a connection with a publisher is established, the subscriber sends this command. The command can be used only once per connection and it must be the first command to the publisher. Any other commands received by the publisher before this command may cause a session termination since the connection may not look like a valid STTP protocol session. Below are the elements that make up the define operational modes command payload:

Field	Byte Size	Description
VALUE	4	Desired operational modes to use (INT32) See Table 21. Operational Modes – Bit Masks and Values

Table 20. Define Operational Modes Command Payload Fields

Field	Value	Description
VERSION MASK	0x0000001F	Bit mask used to apply protocol version number (0 to 31)
COMPRESSION MODE MASK	0x000000E0	Bit mask used to set desired compression algorithms, see Table 22. Compression Algorithm Flags
ENCODING MASK	0x00000300	Bit mask used to set text encoding mode, see Table 23. Text Encoding Options
COMPRESS PAYLOAD DATA	0x20000000	Determines whether payload data is compressed when exchanged between publisher and subscriber
COMPRESS SIGNAL INDEX CACHE	0x40000000	Determines whether the signal index cache is compressed when exchanged between publisher and subscriber
COMPRESS METADATA	0x80000000	Determines whether metadata is compressed when exchanged between publisher and subscriber

Table 21. Operational Modes – Bit Masks and Values

Field	Value	Description
GZIP	0x20	Enable GZip style compression – applies to METADATA REFRESH success response payload when COMPRESS METADATA flag is set, UPDATE SIGNAL INDEX CACHE payload when COMPRESS SIGNAL INDEX CACHE flag is set, and DATA PACKET payload when data channel operates over a UDP socket and COMPRESS PAYLOAD DATA is set
TSSC	0x40	Enable TSSC style compression – applies to DATA PACKET payload when data channel operates over a TCP socket and COMPRESS PAYLOAD DATA is set
NONE	0x00	No compression algorithms are selected

Table 22. Compression Algorithm Flags[31]

[31] Instead of requiring fixed bit flags for compression algorithms, future versions of STTP may instead use string values to specify the desired algorithm to use. Using a string name creates a more flexible algorithm negotiation option – the publisher need only respond with a failure if a requested algorithm is not supported, perhaps including a list of the supported compression algorithms.

Field	Value	Description
UNICODE	0x000	Process strings using UTF-16 (a.k.a., Unicode) encoding
BIG ENDIAN UNICODE	0x100	Process strings using UTF-16 (a.k.a., Unicode) encoding where each 16-bit text character is serialized in big-endian order
UTF8	0x200	Process strings using UTF-8 encoding
ANSI	0x300	Process strings using ANSI encoding

Table 23. Text Encoding Options

Confirm Notification Command Payload

The payload for the CONFIRM NOTIFICATION command defines the hash code, received in the NOTIFY response payload, used to inform the publisher that the notification sent to the subscriber was received.

Field	Byte Size	Description
VALUE	4	Notification hash to confirm (INT32)

Table 24. Confirm Notification Command Payload Fields

Confirm Buffer Block Command Payload

The payload for the CONFIRM BUFFER BLOCK command defines the sequence number, received in the BUFFER BLOCK response payload, used to inform the publisher that the buffer block sequence was received.

Field	Byte Size	Description
VALUE	4	Buffer block sequence to confirm (UINT32)

Table 25. Confirm Buffer Block Command Payload Fields

XVI. APPENDIX D – STTP RESPONSE PAYLOADS

The following are the payload definitions associated with specific STTP response types, see Table 12. STTP Response Types.

Succeeded Response Payload for Simple Commands

The payload for the SUCCESS response defines the message content for a successful response to a solicited command. In the case of the SUBSCRIBE, UNSUBSCRIBE, and ROTATE CIPHER KEYS[32] commands the payload will be a text message, suitable for logging, that provides details about the operation, e.g., how many points were successfully subscribed. Below are the elements that make up the payload for a simple response:

Field	Byte Size	Description
MESSAGE	PAYLOAD LENGTH	String-based response message

Table 26. Success Response Payload Fields for Simple Commands

Succeeded Response Payload for METADATA REFRESH Command

The payload for the SUCCESS response for the METADATA REFRESH command is XML dataset containing all the metadata the subscriber is authorized to see[33]. The payload contents will be compressed if the GZip compression mode is enabled with the DEFINE OPERATIONAL MODES command when the connection is established, see Table 22. Compression Algorithm Flags. The content of XML metadata is extensible and subject to industry needs, however, at a minimum the fields as defined in APPENDIX B – STTP

[32] The success or failure response to the ROTATE CIPHER KEYS is simply an operational confirmation to the solicited command. The subscriber will receive a second response, i.e., UPDATE CIPHER KEYS, that actually contains the new symmetric encryption keys – this is because UPDATE CIPHER KEYS can be unsolicited, i.e., the publisher can update the keys at any time.

[33] If the subscriber specified any filter expressions as part of the METADATA REFRESH command payload, these filter expressions will be applied and will reduce the metadata available in the dataset.

METADATA are available. Below are the elements that make up the payload for a simple response:

Field	Byte Size	Description
XML METADATA	PAYLOAD LENGTH	Compressed GZip or raw string-based response message that contains XML tables representing metadata.

Table 27. Success Response Payload Fields for METADATA REFRESH Command

Failed Response Payload

The payload for the FAILED response defines the message content for a failed response to a solicited command. Any failed response will be a text message, suitable for logging, that provides details about the unsuccessful operation. Below are the elements that make the payload for a failed response:

Field	Byte Size	Description
MESSAGE	PAYLOAD LENGTH	String-based response message

Table 28. Failed Response Payload Fields

Data Packet Response Payload

The payload for the DATA PACKET response defines the serialized time-series measurement values, i.e., the data, that are streaming back to a subscriber. The contents of the data packets are repeated binary encoding of an identifier, timestamp, measured value and flags (e.g., time and data quality) – that are in no fixed order, i.e., the measurements in one data packet may be different than those in the next. Data packet size is managed at the application layer such that only a finite group of measurements are sent at once – ideally one group with will fit neatly within a single network packet to reduce (or eliminate) data block fragmentation, see

LARGE FRAME IMPACT ON IP. The data packets are constructed to be easy to parse so that third party systems can easily consume and use data. The raw native format of the data packet response uses a compact serialization format to help conserve bandwidth. Below are the elements that make up the payload for a data packet response:

	Field	Byte Size	Description
	DATA PACKET FLAGS	1	Defines a marker used to define the data packet payload format, e.g., if the content is compressed, see Table 30. Data Packet Flags
⌐	MEASUREMENT COUNT	4	Number of measurements defined in data packet payload (INT32)
⌊	DATA PACKET PAYLOAD	PAYLOAD LENGTH - 5	Format depends on if COMPRESS PAYLOAD DATA was specified in DEFINE OPERATIONAL MODES and selected compression algorithm, see Table 22. Compression Algorithm Flags
	Encrypt Range		When UDP encryption is enabled with the UPDATE CIPHER KEYS command, data will be encrypted starting with MEASUREMENT COUNT field and include all of the DATA PACKET PAYLOAD

Table 29. Data Packet Response Payload Fields

Field	Value	Description
COMPACT	0x02	Determines if data packet payload format is using compact serialization
CIPHER INDEX	0x04	Determines which cipher index to use when decrypting data packet payload: Bit set = use odd cipher index (i.e., 1), Bit clear = use even cipher index (i.e., 0)
COMPRESSED	0x08	Determines if data packet payload is compressed – actual compression format will be based on data channel socket and selected compression algorithm, a TCP socket will use TSCC and a UDP socket will use GZip

Table 30. Data Packet Flags

Specifics of the compression of a data packet are covered in the STTP Data Compression section. The details that follow describe the native STTP encoding of time-series values. Each block of data to be published, i.e., the data packet response payload, consists of a collection of time-series values where each value is a serialized structure containing a 128-bit GUID-based identifier, a 64-bit high resolution timestamp, a 32-bit floating-point value, and one byte (i.e., 8-bits) of associated quality flags, see Figure 7. The GUID-based identifier will be directly referenceable as a lookup key into the received metadata so that the measurement type, description, and other information can be used by a consuming application at runtime. The total data block size is dynamically configurable, this way it can be adjusted at run-time to accommodate varying network conditions to reduce packet fragmentation – but ideally, the data block size will target the current network MTU size minus headers so that one block of data will fit within one network packet.

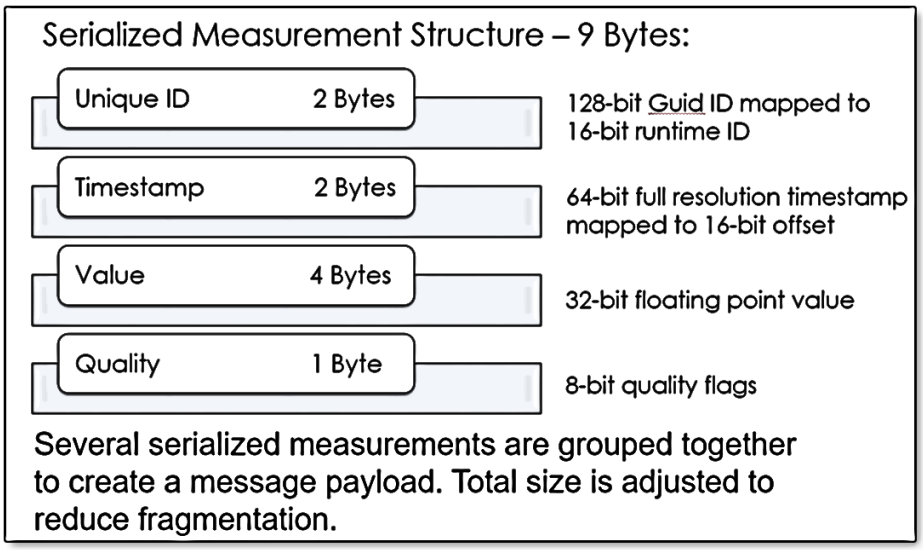

Figure 7. Serialized Measurement Structure

To reduce the size of transmission of serialized measurements the 128-bit GUID is mapped to a 16-bit[34] identifier established by the publisher during subscription, see Update Signal Index Cache Response Payload. Additionally, since publication of timestamps are near other timestamps in value, offsets are provided to the subscriber by the publisher to provide a

[34] Future versions of STTP are expected to change this runtime identifier to a 32-bit value used with 7-bit encoding to allow subscriptions with more than 65K subscribed measurement values.

common base for timestamp values so that the size can be reduced. When the offset is unavailable or cannot be used (because it is out of range for the timestamp to be encoded), the full resolution timestamp will be sent. Below are the elements that make up the format of a compact measurement:

Field	Byte Size	Description
MEASUREMENT FLAGS	1	Defines the compact measurement flags. These flags also include the base time offset, if it is in use, to designate the current base time offset index, see Table 32. STTP Compact Measurement Flags
SIGNAL INDEX[34]	2	Index from the Signal Index Cache table that maps to the 128-bit measurement GUID (UINT16)
MEASUREMENT VALUE	4	32-bit floating-point value of the measurement
TIMESTAMP	0, 2, 4 or 8	Encoded timestamp: 0 bytes if subscription does not include timestamps 2 bytes UINT16 offset added to current time base if millisecond resolution has been selected 4 bytes UINT32 added to current time base if not using millisecond resolution 8 bytes when full timestamp encoding is needed, i.e., value outside range of current base time offset

Table 31. STTP Compact Measurement Format

Field	Value	Description
BASE TIME OFFSET	0x40	Determines if base time offset is active for current measurement
TIME INDEX	0x80	Determines which base time offset index to use for current measurement (when active): Bit set = use odd time index (i.e., 1), Bit clear = use even time index (i.e., 0)
DATA RANGE	0x01	Marks measurement quality flags to show issue with the value range

DATA QUALITY	0x02	Marks measurement quality flags to show issue with value quality
TIME QUALITY	0x04	Marks measurement quality flags to show issue with the time quality
SYSTEM ISSUE	0x08	Marks measurement quality flags to show a system related issue
CALCULATED VALUE	0x10	Marks measurement quality flags to show that value was calculated / derived
DISCARDED VALUE	0x20	Marks measurement quality flags to show that value was discarded while processing

Table 32. STTP Compact Measurement Flags

Update Signal Index Cache Response Payload

The payload for the UPDATE SIGNAL INDEX CACHE response defines the lookup table that maps a 128-bit measurement ID to a 16-bit[34] run-time used to reduce the size of a serialized measurement. Since the signal index cache is maintained per subscriber, this cache will only map the authorized measurements available to a subscriber, however, for the set of requested measurements the cache also includes a list of the measurements that publisher did not authorize. Generated run-time IDs are temporal, i.e., they only have context for the current session. A subscriber must accept updates to the signal index cache when provided by a publisher because changes in metadata and access rights can occur dynamically. To reduce race conditions, publisher should attempt to maintain as much continuity and uniqueness as possible with run-time IDs when dynamically updating the signal index cache as time-slew can occur between data transmission of measurements and reception of new signal index cache. Below are the elements that make up payload for the update signal index cache:

Field	Byte Size	Description
SIGNAL INDEX CACHE PAYLOAD	PAYLOAD LENGTH	Compressed GZip or directly serialized signal index cache, see Table 34. STTP Signal Index Cache Format

Table 33. Update Signal Index Cache Response Payload Fields

	Field	Byte Size	Description
	SIGNAL INDEX CACHE PAYLOAD LENGTH	4	Number of bytes in the signal index cache payload (INT32) – this payload length is different from the response message payload length because message payload may be compressed
	SUBSCRIBER ID	16	Defines the 128-bit GUID-based identifier that uniquely identifies the subscriber, typically implemented as a run-time reference to a validated certificate
	REFERENCE COUNT	4	Total number of signal index references defined
┌	SIGNAL INDEX[34]	2	Run-time measurement index that maps to a 128-bit GUID (UINT16)
│	SIGNAL ID	16	Defines the 128-bit GUID-based identifier that uniquely defines the measurement
│	SOURCE SIZE	4	Number of bytes in the source string for the measurement (INT32)
│	SOURCE[35]	SOURCE SIZE	String-based source of measurement as defined by the publisher
└	SOURCE ID	4	Source identifier as defined by the publisher (UINT32)
	Repeat Fields		Repeat fields SIGNAL INDEX to SOURCE ID for REFERENCE COUNT times
	UNAUTHORIZED COUNT	4	Total number of measurements requested by subscriber that were unauthorized by publisher
┌	UNAUTHORIZED SIGNAL ID	16	Defines the 128-bit GUID-based identifier requested by subscriber that was unauthorized by publisher
└	*Repeat Field*		Repeat field UNAUTHORIZED SIGNAL ID for UNAUTHORIZED COUNT times

Table 34. STTP Signal Index Cache Format

[35] The source string and associated integer source ID will be dropped in future versions of STTP. These fields were implemented as part of GEP to define an alternate lookup key for the signal index.

Data Start Time Response Payload

The payload for the DATA START TIME response defines the timestamp of the first data point sent by the publisher. To reduce bandwidth, options exist to not transport timestamps when the subscriber does not need them. Additionally, the timestamp value may represent a sequence value. To provide a base timestamp that represents the time at the beginning of a transmission, this response includes the timestamp value of the first value to be transmitted when a subscription is started, should the value be useful. See Protocol Timestamp Format for information on timestamp encoding. Below are the elements that make up the payload for a data start time response:

Field	Byte Size	Description
VALUE	8	Start time of first data point (INT64)

Table 35. Data Start Time Response Payload Fields

Buffer Block Response Payload

The payload for the BUFFER BLOCK response defines a free form block of data that does not conform to a time-series value. A buffer block measurement must still be defined as a measurement and subscribed-to by the subscriber, but the implementation can be sourced from data other than time-series measurements, e.g., file data. Each data in block must be partitioned to fit within a minimal number of network packets, so data from a source larger than a network packet must be sequenced. The subscriber will be required to acknowledge reception of this BUFFER BLOCK response with a CONFIRM BUFFER BLOCK command since blocks may exist as a sequence of packets and could require retransmission, e.g., over UDP. Below are the elements that make up the payload for a buffer block response:

Field	Byte Size	Description
VALUE	4	Buffer block sequence used for confirmation (UINT32)
SIGNAL INDEX	2	Index from the Signal Index Cache table that maps to the 128-bit measurement GUID (UINT16)
BUFFER BLOCK PAYLOAD	PAYLOAD LENGTH - 6	Free-form payload of buffer block

Table 36. Buffer Block Response Payload Fields

Notify Response Payload

The payload for the NOTIFY response defines a critical notification as a string message. A notify operation works with a specially defined measurement, still requiring subscription, that allows the subscriber to receive messages with verified delivery. Since the message is considered critical, the subscriber must respond with a CONFIRM NOTIFICATION command since the message may require retransmission, not only when used over a lossy communications transport, e.g., UDP, but because the publisher requires verification of delivery even if subscriber has gone offline, e.g., restarting:

Field	Byte Size	Description
VALUE	4	Notification hash used for confirmation (INT32)
MESSAGE	PAYLOAD LENGTH - 4	String-based notification message

Table 37. Notify Response Payload Fields