Gateway Exchange Protocol Overview

The Gateway Exchange Protocol is an open source measurement-based publish/subscribe transport protocol used to securely exchange time-series data and automatically synchronize meta-data between two 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. GEP streaming communication speed can be set to "as-quickly-as-possible" as is typically desirable for system-to-system communication.

For synchrophasor data, use of GEP overcomes the scaling limits imposed by frame-based protocols. By their nature, configuration frames in frame-based protocols are much larger than data frames and can quickly exceed practical UDP data packet size limits. Synchrophasor implementations often use a combination of TCP and UDP to help postpone frame size limitations using TCP to send the configuration frame and UDP to send data frame; however, frame based protocols still have a fixed maximum frame size of 64K that can be transmitted in one frame even when the transport is TCP. Use of a measurement-based protocol, like GEP, overcomes these issues.

Additionally, using GEP provides the following benefits over frame-based protocols:

- Automatic exchange of authorized metadata information between GEP appliances (the publisher approves subscriber data access based on data filters – which can be broad or as specific as measurement level approval.)
- Can reduce bandwidth required for communicating data by using lossless compression techniques.
- Allows the subscribing GEP appliance to start and stop the data stream as needed.
- Allows the subscribing GEP appliance to dynamically change the measurements points which are being received (within set of what publisher allows).
- Reduces latency for most synchrophasor data architectures since data is communicated "on receipt" without the need for data concentration into time-based frames.

To test the impact of sending data in small packets rather that large (often very large) frames, GEP has been tested using UDP over the internet with a set of approximately 125 million synchrophasor measurements using 4 different protocols: IEEE C7.118-2005, IEC 61850-90-5 with no frame retransmission, IEC 61850-90-5 with one frame retransmission, and GEP. The results shown in Figure 1 below show the advantage of the small GEP packet size on reducing data loss.

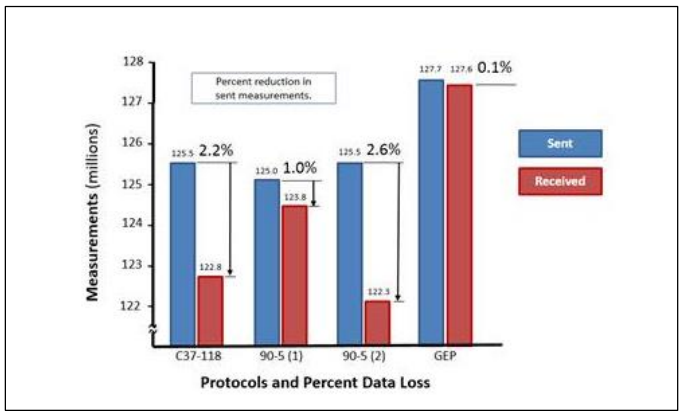

Figure 1. GEP Data Loss vs. Frame-based Protocols

The GEP Protocol

The wire protocol employed by GEP implements a publish/subscribe data exchange model using a simple command driven service with tightly compressed, fast binary serialization of time-series values. The protocol does not require a predefined or fixed configuration – that is, the time-series values arriving in one data packet can be different than those arriving in another. Each packet of data consists of a collection of time-series values; each time-series value is a structure containing an ID, a time-stamp, a value and associated flags.

The GEP is implemented using a TCP/IP command channel (for actions such as subscription) and optionally a UDP/IP data channel (the actual data to be transmitted). The TCP command channel is used to reliably negotiate session specific required communication, state and protocol parameters. It is used to authenticate with other GEP communications appliances, exchange metadata on points with them, and request points for subscription. This channel can also use transport layer security (TLS) for secure communications. The UDP data channel is used to send compact, binary encoded packets of identifiable measured values along with a timestamp accurate to one ten-millionth of a second (e.g., a tick) and flags that can be used to indicate time and data quality. The data on the UDP channel can also be encrypted using keys that are dynamically exchanged on the command channel. When the UDP data channel is not used, data is transmitted on the TCP channel.

Subscriber Command Format -- The subscriber command consists of a payload marker, payload length, command code and actual payload bytes. To easily separate multiple commands on the wire, the first four bytes in the command wire format is a payload marker, specifically: 0xAA, 0xBB, 0xCC, 0xDD. The next four bytes (a 32-bit integer) represent the total size of the payload including the command code. The next byte is the subscriber command code. See: Appendix A, GEP Subscriber Commands.

Publisher Response Format -- The publisher response consists of a response code, an in-response-to command code, payload length and actual payload bytes. The first byte is the response code. See: Appendix B, GEP Publisher Responses. The second byte is command code for which this response applies. The command code is required even if the response is unsolicited. The next

four bytes (a 32-bit integer) represent the payload length. The actual bytes of the payload follow, if any, and are specific to the actual response. For example, in the case of a data packet response, the payload will contain serialized measurements.

The serialized measurements, "the data", are constructed in GEP to be easy to parse so that third-party systems can easily consume and use data. The data structure is a repeated binary encoding of an ID, timestamp, measured value and flags (e.g., time and data quality).

- Point ID – A 128-bit GUID identifier provided in the command channel. This value is compressed to 2 bytes on the wire using run-time cache.
- Time – A 64-bit integer based timestamp in "ticks" (100's of nanoseconds). This value is also compressed to 2 bytes on the wire using an offset to an absolute time value which is updated frequently.
- Value – Typically a 32-bit floating point real number (4 bytes). GEP supports other data payload sizes.
- Flags – For synchrophasor data these flags are embedded in a 16-bit integer representing the IEEE C37.118 data quality flags.

Security

GEP can be implemented with or without its security features. GEP enables implementation of both strong access control and encryption. For GPA's products, security is managed through components in the Grid Solutions Framework (GSF). These features include:

Administrator access control where multiple role-based options are available. This access control can be implemented to integrate with existing enterprise authentication, such as, Microsoft Active Directory, Kerberos, and local accounts. The GSF also provides the capability for multi-factor authentication strategies using hardware/software tokens, e.g., RSA SecurID Hardware Tokens.

Authentication / access control for data communication includes strong authentication of trusted appliances through the out of band exchange of symmetric keys using transport layer security (TLS). Publishers have a fine-grained mechanism to control access to specific data by authenticated partner (or trusted) GEP appliances.

Integrity-protected logging for operating logs and configuration logs as well as remote log storage capability for additional security. The GSF leverages standardized logging to the OS so that errors and events can be captured through enterprise log integration systems.

Key Management – GEP is configurable to allows use and manage private keys in a highly isolated environment. Using GSF transport security features, GEP is also capable of utilizing key management services that offer X.509 identity certificates for authentication. In the absence of that infrastructure, GSF is able to use self-signed X.509 identity certificates that are securely communicated out-of-band.

GEP facilitates full compliance with NERC CIP standards with complete logging of configuration changes and administrative actions.

As seen in Table 1 below, GEP is an alternative to the use of VPN for securing communication of streaming utility operating data.

VPN Approach	**GEP Approach**
• Security managed at a network interface level	• Security managed at the application layer, with fully flexible pairwise security
• Traffic only protected once it reaches the VPN tunnel, susceptible at every previous level	• Traffic is protected from the very beginning, protecting it directly in the application which eliminates exposure via other apps on the system
• VPN failure can result in either unencrypted data being sent, or complete blockage of transmission until the network issue can be resolved	• Connection failure results in retried connections, renegotiating the key at each try
• If network issues are present, the connection may require intervention to either start or stop once the network issues are corrected	• If network issues are present, the connection will be re-established without intervention once they are corrected

Table 1. Comparison of VPN and GEP

GEP APIs

GEP API's are available in C/C++, .NET (including Mono and Unity 3D) and Java to enable these applications to easily be integrated with the applications that use GEP and avoid limitations imposed by use of frame-based protocols. An open source application, the GEP Subscription Tester is a multi-platform graphical application that can be used to verify connectivity to applications implementing a GEP data publisher.

Two API calls exist to subscribe to real-time data:

1)
```
bool SynchronizedSubscribe(bool compactFormat, int framesPerSecond, double lagTime, double
leadTime, string filterExpression, bool useLocalClockAsRealTime = false, bool
ignoreBadTimestamps = false, bool allowSortsByArrival = true, long timeResolution =
Ticks.PerMillisecond, bool allowPreemptivePublishing = true, DownsamplingMethod
downsamplingMethod = DownsamplingMethod.LastReceived)
```

2)
```
bool UnsynchronizedSubscribe(bool compactFormat, bool throttled, string filterExpression,
double lagTime = 10.0D, double leadTime = 5.0D, bool useLocalClockAsRealTime = false)
```

These two API calls provide the following possible real-time data subscriptions:

- A synchronized (i.e., concentrated by time) set of subscribed data points:
```
subscriber.SynchronizedSubscribe(true, 30, 0.5D, 1.0D, "FILTER ActiveMeasurements WHERE
Device = 'SHELBY' AND SignalType = 'VPHM'");
```

- An on-change unsynchronized set of subscribed data points:
```
subscriber.UnsynchronizedSubscribe(true, false, "PPA:87;PPA:92");
```

- A throttled (e.g., down-sampled to every few seconds) set of subscribed data points:
```
subscriber.UnsynchronizedSubscribe(true, true, "POINT:1;POINT:2", 5.0D,
1.0D, false);
```

Appendix A -- GEP Subscriber Commands

Table of commands sent by a subscriber and received by a publisher. Note that solicited server commands will receive a Succeeded or Failed response code along with an associated success or failure message. Message type for successful responses will be based on server command - for example, server response for a successful MetaDataRefresh command will return a serialized dataset of the available server metadata. Message type for failed responses will always be a string of text representing the error message.

Command	Command Code	Description
MetaDataRefresh	0x01	Meta data refresh command. Requests that server send an updated set of metadata so client can refresh its point list. Successful return message type will be a dataset containing server device and measurement metadata. Received device list should be defined as children of the "parent" server device connection similar to the way PMUs are defined as children of a parent PDC device connection. Devices and measurements contain unique Guids that should be used to key metadata updates in local repository.
Subscribe	0x02	Subscribe command. Requests a subscription of streaming data from server based on connection string that follows. It will not be necessary to stop an existing subscription before requesting a new one. Successful return message type will be string indicating total number of allowed points. Client should wait for UpdateSignalIndexCache response code before attempting to parse data.
Unsubscribe	0x03	Unsubscribe command. Requests that server stop sending streaming data to the client and cancel the current subscription.
RotateCipherKeys	0x04	Rotate cipher keys. Manually requests that server send a new set of cipher keys for data packet encryption for use on the UDP data channel. There are always two keys, the old one and the new one, to accommodate time slew in transitioning from one key to another. This should only be used in conjunction with a TLS-based command channel.
DefineOperationalModes	0x06	Define operational modes for subscriber connection. As soon as connection is established, requests that server set operational modes that affect how the subscriber and publisher will communicate (e.g., compression or text encoding style).
ConfirmNotification	0x07	Confirm receipt of a notification. This command is sent in when a Notify response is received.
ConfirmBufferBlock	0x08	Confirm receipt of a buffer block measurement. This command is sent when a BufferBlock response is received.

Appendix B -- GEP Publisher Responses

Table of responses sent by a publisher and received by a subscriber*. Although the subscriber commands and publisher responses will be on two different paths, the response codes are defined as distinct from the command codes to make it easier to identify values from a wire analysis.

Response	Response Code	Description
Succeeded	0x80	Command succeeded response. Informs client that its solicited server command succeeded, original command and success message follow.
Failed	0x81	Command failed response. Informs client that its solicited server command failed, original command and failure message follow.
DataPacket	0x82	Data packet response. Unsolicited response informs client that a data packet follows.
UpdateSignalIndexCache	0x83	Update signal index cache response. Unsolicited response requests that client update its runtime signal index cache with the one that follows.
UpdateCipherKeys	0x85	Update runtime cipher keys response. Response, solicited or unsolicited, requests that client update its runtime symmetric encryption keys with those that follow and use the keys to decrypt the data on the UDP data channel. This should only be used in conjunction with a TLS-based command channel.
DataStartTime	0x86	Data start time response packet. Unsolicited response provides the start time of data being processed from the first measurement.
BufferBlock	0x88	Buffer block response. Unsolicited response informs client that a raw buffer block follows.
Notify	0x89	Notify response. Unsolicited response provides a notification message to the client.
NoOP	0xFF	No operation keep-alive ping. It is possible for the command channel to remain quiet for some time, this command allows a periodic test of client connectivity.

* Technically these are responses to subscriber commands as well as commands sent by the publisher to the subscriber that were not necessarily solicited, but they are referred to only as responses for clarity in understanding and communicating data flow direction.