Comet Documentation

Release 2.0.1

John Swinbank

March 17, 2016

Contents

VOEvent is the International Virtual Observatory Alliance standard for describing transient celestial events. Comet is an implementation of the VOEvent Transport Protocol, which provides a mechanism for fast and reliable distribution of VOEvents to the community. Comet enables you to subscribe to and act upon real-time streams of ongoing events, to use filters to select only events relevant to your science case and to publish your own events to the global VOEvent backbone.

CHAPTER 1

Obtaining and Installing Comet

1.1 Installation using pip

The latest version of Comet and all of the tools it depends upon can be installed using pip. It is generally a good idea to use virtualenv to create an isolated, self-contained installation:

```
$ virtualenv comet
$ . comet/bin/activate
$ pip install comet
```

1.2 Manual installation

1.2.1 Requirements

Comet is developed targeting Python 2.7. It depends upon:

- Twisted (version 11.1.0 or later)
- lxml (version 2.3 or later)
- zope.interface (versions 3.6.0 or later)
- py2-ipaddress.

How you make these dependencies available on your system is up to your (or, perhaps, to your system administrator). However, the author strongly suggests you might start by taking a look at virtualenv.

1.2.2 Downloading

See the *release history* to obtain the latest version of Comet or check out the source from the GitHub repository. The latest version of the source can be obtained using git:

```
$ git clone https://github.com/jdswinbank/Comet.git
```

1.2.3 Installation

Comet includes a distutils setup script which can be used for installation. To install in your system-default location, run:

```
$ python setup.py install
```

A number of other options are available: see also:

```
$ python setup.py --help
```

1.3 Testing

After installation, you should check that Comet is working properly. The Twisted framework used by Comet makes this easy with its `trial` tool. Simply run:

```
$ trial comet
```

No failures or errors are expected in the test suite. If you see a problem, please contact the author for help.

CHAPTER 2

Usage

The VTP system defines three types of nodes—Author, Broker and Subscriber—and two types of conenction—Author to Broker and Broker to Subscriber. Comet has the capability to act in all of these roles. To do so, it provides two different tools.

The first is `comet-sendvo`. This provides the capability of publishing a VOEvent packet to a remote broker. In other words, `comet-sendvo` assumes the role of Author in the above terminology, connects to a remote Broker, and delivers an event to it. The user invoking Comet in this mode is, of course, responsible for actually providing the event text to be sent.

The second tool is the Comet broker itself. This runs as a background process (daemon), and can:

- Accept submissions from Authors (such as `comet-sendvo`);
- Subscribe to streams of events from one or more remote Brokers;
- Distribute events received (whether by direct author submission or by subscription) to its own subscribers;
- Perform arbitrary actions upon events received.

2.1 Event Publisher

`comet-sendvo` is a simple event publisher: it forwards messages from an author to a broker.

After installation, it should be possible to execute `comet-sendvo` directly from the command line. Use the `--help` option to display a brief usage message:

```
$ comet-sendvo --help
Usage: comet-sendvo [options]
Options:
  -h, --host=             Host to send to. [default: localhost]
  -p, --port=             Port to send to. [default: 8098]
  -f, --file=             Where to read XML text (- is stdin). [default: -]
      --version           Display Twisted version and exit.
      --help              Display this help and exit.
```

Basic usage is straightforward: simply supply `comet-sendvo` with the details of the broker to connect to using the `--host` and `--port` options (or accept the defaults), and give it the text of a VOEvent either directly on standard input or by giving the path to a file. For example:

```
$ comet-sendvo --host=remote.invalid --port=8098 < voevent_to_publish.xml
```

Or:

```
$ comet-sendvo -h remote.invalid -f voevent_to_publish.xml
```

2.2 Broker

The Broker is the part of Comet which receives messages from authors and distributes them to subscribers. It can also subscribe to other brokers and act upon events received.

2.2.1 Twisted Applications

The Comet broker is implemented as a Twisted application. This means that it is not invoked directly on the command line, but rather using the `twistd` tool provided as part of Twisted.

`twistd` provides a wide range of generic configuration options related to daemonizing, logging, profiling, and so on. This fall outside the scope of this documentation, but the interested reader is encouraged to familiarize themselves with the contents of `twistd(1)`.

It is worth noting that, by default, processes invoked with `twistd` are immediately daemonized. That is, `twistd` is invoked, it returns control to your shell prompt almost immediately, but Comet is now running in the background. This is generally very convenient, but sometimes it's useful to have the application run in the foreground when testing and debugging: to achieve this, invoke `twistd -n`.

2.2.2 Invoking Comet

Specify name of the Twisted application to run after `twistd` and its options. In this case, we run `comet`, and pass it the `--help` option to provide a brief usage message:

```
$ twistd comet --help
Usage: twistd [options] comet [options]
Options:
  -r, --receive                   Listen for TCP connections from authors.
  -b, --broadcast                 Re-broadcast VOEvents received.
  -v, --verbose                   Increase verbosity.
  -q, --quiet                     Decrease verbosity.
      --print-event               Enable the print-event plugin.
      --save-event                Enable the save-event plugin.
      --local-ivo=                IVOA identifier for this system (required).
      --eventdb=                  Event database root. [default: /tmp]
      --receive-port=             TCP port for receiving events. [default: 8098]
      --broadcast-port=           TCP port for broadcasting events. [default:
                                  8099]
      --broadcast-test-interval=  Interval between test event brodcasts (in
                                  seconds; 0 to disable). [default: 3600]
      --whitelist=                Network to be included in submission
                                  whitelist. [default: 0.0.0.0/0]
      --remote=                   Remote broadcaster to subscribe to
                                  (host[:port]).
      --filter=                   XPath filter applied to events broadcast by
                                  remote.
      --cmd=                      Spawn external command on event receipt.
      --save-event-directory=     Target directory [default:
                                  current working directory]
      --help                      Display this help and exit.
      --version                   Display Twisted version and exit.
```

Basic Modes of Operation

If the `--receive` option is supplied, Comet will fulfil the Broker role in an Author to Broker connection. In other words, it will listen for TCP connections from remote authors and accept events for distribution. The TCP port on which Comet will listen may be specified with the `--receive-port` option.

If the `--broadcast` option is supplied, Comet will listen for Subscribers to connect and then it will fulfil the Broker role in a Broker to Subscriber connection with each of the Subscribers. Any VOEvents received (either by direct connection from authors, or by subscribing to remote brokers) are rebroadcast to subscribers. The TCP port on which Comet will allow subscribers to connect may be specified with the `--broadcast-port` option.

If one or more `--remote` options are supplied, Comet will subscribe to the remote host specified and fulfil the Subscriber role in the resulting Broker to Subscriber connection. If just given a hostname Comet will attempt to subscribe on port 8099. Optionally, a different port may be specified by appending it to the hostname, separated by a `:`.

A single Comet daemon will accept any combination of `--receiver`, `--broadcast` and one or more `--remote` options and play all of the specified roles simultaneously. If none of `--receiver`, `--broadcast` or `--remote` are supplied, there is no work to be done and Comet will exit immediately.

Identification

Whatever the mode of operation, Comet identifies itself by means of an *International Virtual Observatory Resource Name* or *IVORN*: see the VOEvent standard for details. You should specify some appropriate IVORN for your site using the `--local-ivo` option. This should be of the format `ivo://${organization}/${name}`; for example, `ivo://org.transientskp/comet_broker`.

VOEvent Network Maintenance

In order to prevent looping on the network (ie, two brokers exchanging the same event ad infinitum), a database of previously seen event is maintained. This database is written to the filesystem in the location specified by the `--eventdb` option. Events which are recorded in the database are not forwarded by Comet. This is important: looping would degrade the quality of the VOEvent network for all users! Note that events persist in the database for 30 days, after which they are expired to save space.

Receiver Options

When acting as a receiving broker (with `--receive`), Comet will only accept new events for publication from hosts which have been specified as "whitelisted". Hosts (or, indeed, networks) may be included in the whitelist using the `--whitelist` option. This option accepts either CIDR or dot-decimal notation including a subnet mask. For example, `--whitelist 127.0.0.1/32` and `--whitelist 127.0.0.1/255.255.255.255` both permit the local host to submit events to the broker. This option may be specified multiple times and the results are cumulative. To accept submissions from any host, specify `--whitelist 0.0.0.0/0`; this is the default if no `--whitelist` option is supplied.

Broadcaster Options

By default, Comet will broadcast a content-free test event to all subscribers every hour. The aim is to help with network debugging. The interval between test events may be configured using the `--broadcast-test-interval` option, which accepts a value in seconds. Set it to `0` to disable the test broadcast completely.

Subscriber Options

When subscribing to a remote broker (with `--remote`), one or more filters may be specified which limit the events which will be received. These filters are specified with `--filter`, in the form of XPath 1.0 expressions. The broker will evaluate the expression against each event it processes, and only forward the event to the subscriber if it produces a non-empty result. For more details see the section on *filtering*.

Common Options

Plugins

Custom code may be run to perform arbitrary local processing on an event when it is received. For more details, see the section on *event handlers*. Plugin actions will be taken whether Comet receives an event from an author (`--receive`) or an upstream broker (`--remote`). A plugin is enabled by giving its name as a command line option (`--plugin-name`). Plugins may also take arguments from the command line. These are given in the form `--plugin-name-argument=value`.

Comet ships with two plugins which both serve as examples of how to write event handlers and which may be useful in their own right. The first simply writes events to Comet's log as they are received. This is the `print-event` plugin: enable it by invoking Comet with the `--print-event` option.

The second plugin shipped with Comet is `save-event`, which writes events to file. It is enabled with the `--save-event` option. By default, events are written to the default working directory (normally the directory in which you invoked Comet): this may be customized using the `--save-event-directory=` option. The filename under which an event is saved is based on its IVORN, but modified to avoid characters which are awkard to work with on standard filesystems.

Spawning External Commands

Similarly, received events may be sent to one or more external commands for processing. These are specified using the `--cmd` option. They should accept the event on standard input and perform whatever processing is required before exiting. The standard output and error from the external process is ignored. If it returns a value other than 0, it will be logged as a failure. Note that external commands are run in a separate thread, so will not block the subscriber from processing new events; however, the user is nevertheless responsible for ensuring that they terminate in a timely fashion.

Logging

The amount of information Comet writes to its log may be adjusted using the `--verbose` and `--quiet` options.

CHAPTER 3

Filtering

As the number of events on the VOEvent backbone increases, it is unlikely that individual subscribers will wish to receive or act upon all of them. Comet therefore implements an experimental filtering system which enables subscribers to express their preferences as to which events to receive.

At any time, the subscriber may send the broker an authentication response message. (Note that in the current implementation no authentication is actually requred, and the processing of digital signatures is not supported). Within the `<Meta />` section of the authentication packet, one or more XPath expressions may be supplied in `<Param />` elements with a `name` attribute equal to `xpath-filter`. For example, the following will select all VOEvent packets which are not marked as a test:

```
<trn:Transport version="1.0" role="authenticate"
  xmlns:trn="http://www.telescope-networks.org/xml/Transport/v1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://telescope-networks.org/schema/Transport/v1.1
    http://www.telescope-networks.org/schema/Transport-v1.1.xsd">
  <Origin>ivo://origin</Origin>
  <Response>ivo://response</Response>
  <TimeStamp>2012-02-08T21:13:53Z</TimeStamp>
  <Meta>
    <Param name"xpath-filter" value="/*[local-name()=&amp;quot;VOEvent&amp;quot; and @role!=&amp;quot;test&amp;quot;]
  </Meta>
</trn:Transport>
```

The broker will evaluate each filter against each VOEvent packet it processes, and only forward it to the subscriber if one (or more) of the filters returns a positive result.

It is worth noting that XPath expressions may return one of four different types of result: a boolean, a floating point number, a string, or a node-set. For the purposes of filtering, we regard a positive result as a boolean true, a non-zero number, a non-empty string, or a non-empty node-set.

When evaluating the XPath expression, no namespaces are defined. In other words, an expression such as `//voe::VOEvent` will not match anything (and hence the use of `local-name()` in the example above).

The filtering capabilities of XPath are quite extensive, and the user is encouraged to experiment. For example, the names and values of individual paramters within the VOEvent message can be checked:

```
//Param[@name="SC_Lat" and @value>600]
```

Or messages from particular senders selected:

```
//Who[AuthorIVORN="ivo://lofar.transients/"]
```

CHAPTER 4

Event Handlers

Comet aims to server as a fairly complete and fully-functional broker. However, it is anticipated that those interested in subscribing to VOEvent feeds may have many and varied requirements: it is impossible to take account of all of them. For these users, Comet serves as a template and development platform, and they are encouraged to develop it further to meet their needs.

One way in which the Comet's capabilties may be developed is by providing "event handlers": Python code which is executed when a new event is received. In order to make use of this facility, the developer should be familiar with Twisted's component architecture. Handlers may then be written to follow Comet's `comet.icomet.IHandler` interface, and then installed into the `comet/plugins directory`. A simple example is provided in `comet.plugins.eventprinter`. Note that the plugin class defines a `__call__()` method which is invoked with the event being received as its argument. To be more specific, `__call__()` is handed an instance of `comet.utility.xml.xml_document`.

Each handler must provide a name attribute (e.g. `print-event`). The user may specify that a particular plugin be loaded by specifying its name as a command line argument when invoking comet (`--print-event`).

In some cases, a plugin requires additional configuration. This can be provided through the use of command line arguments. In this case, the plugin must also implement the `comet.icomet.IHasOptions` interface. This involves two further methods: `get_options()`, which returns a list of options which are accepted, and `set_option()`, which provides a means for setting those options. Options declared in plugins will automatically be added to the command line options of the *Comet broker*.

Again, an example of how such a plugin with options may be implemented is likely the best documentation: fortunately, one is available in the form of `comet.plugins.eventwriter`.

CHAPTER 5

Release History

See the *release procedure* section for more details on version numbering and how releases are made.

5.1 Version 2.0.x

2.0.1 (2016-03-17) [Download] Remove incorrect calls to old (pre-2.0.0) logging API. Thanks to Tim-Oliver Husser & Tim Staley.

Reformat documentation to add *Acknowledgements*.

2.0.0 (2016-01-31) [Download] Switch dependency from ipaddr-py to py2-ipaddress. The latter is a backport of the Python 3 functionality, so this helps clear the way for an eventual Python 3 version of Comet.

Use the `$TMPDIR` environment variable, if set, to store the event database. Otherwise, fall back to `tmp`.

Drop support for Python 2.6, following the same change made in Twisted.

Improve checking for valid IVORNs.

Some extremely old versions of Comet (dating from before the 1.0.0 release) used a different format for the database of seen events. All released versions through 1.2.2 automatically update old-style databases to the new format when run. As of this release, this support for legacy databases has been dropped. It is necessary to use a previous Comet release to update the database format before upgrading to this version.

Refactor the codebase caused a minor API change: logging facilities are now available from the `comet.log` module. End user code — notably event handling plugins — should replace statements to the effect of `from comet.utility import log` with `import comet.log as log`. The convenience aliases `log.msg` and `log.warning` have been removed: use `log.info` and `log.warn` instead.

5.2 Version 1.2.x

1.2.2 (2015-04-20) [Download] Disable XML entity expansion for documents received from the network. This eliminates a class of potential resource exhaustion attacks.

Update documentation to request citation of the paper in published work which makes use of Comet.

1.2.1 (2014-09-02) [Download] Correctly check that the (required) `--local-ivo` command line option was provided (GitHub #35).

1.2.0 (2014-08-26) [Download] When subscribing to a remote broker, we wait for a short period after the initial connection is made before marking it as successful. This means that if the broker rapidly drops the connection

(e.g. due to an authentication failure), we retry the connection with an exponential back-off rather than an immediate reconnection (GitHub #29).

Timestamps in `iamalive` messages are marked as being in UTC.

`authenticate` messages which specify XPath filters are schema compliant (GitHub #31).

Subscriber refuses to start if an XPath `--filter` is specified with invalid syntax (GitHub #33).

Require that a valid IVOA identifier (IVORN) be supplied by the end user when starting Comet rather than relying on a default.

Require that events submitted to the broker by authors have valid IVORNs.

5.3 Version 1.1.x

1.1.2 (2014-08-26) [Download] Fix a bug which could result in malformed event IVORNs exhausting the available resources and ultimately rendering Comet unable to process more events (GitHub #34).

1.1.1 (2014-07-08) [Download] Fix a bug which could result in the same VOEvent message being processed multiple times (GitHub #30).

Add compatibility with DBM-style databases which do not provide an `.items()` method.

1.1.0 (2014-02-26) [Download] Improved documentation.

Interval between broadcast test events is user configurable, and they may be disabled. See the `--broadcast-test-interval` option.

Test events now include details of the version of Comet used to generate them.

Event handler plugin system reworked. Plugins may now take command line options. See the *event handler documentation* for details. Note that the syntax for invoking the `print-event` handler has changed (now `--print-event` rather than `--action=print-event`).

Plugin which writes events received to file (`--save-event`).

5.4 Version 1.0.x

1.0.4 (2013-11-13) [Download] `comet-sendvo` will choose its Python interpreter based on the environment.

1.0.3 (2013-11-12) [Download] Update `MANIFEST.in` so that `requirements.txt` is included in the distribution. This changes nothing on an installed system.

1.0.2 (2013-11-12) [Download] Add a `requirements.txt` file and specify the installation requirements in `setup.py`. This makes installation easier, but changes nothing on an installed system.

1.0.1 (2012-08-28) [Download] Fix for badly formed XML `Transport` element.

1.0.0 (2012-08-27) [Download] Initial public release

5.5 Future Plans

- Cryptographic authentication of VOEvent messages and subscribers.
- Port to Python 3.

CHAPTER 6

Acknowledgements

Comet was originally developed by John Swinbank as part of the LOFAR Transients Key Project. It is indexed by the Astrophysics Source Code Library as ascl:1404.008.

If you make use of Comet in published research, please cite Swinbank (2014).

Many thanks to these contributors:

- Tim-Oliver Husser
- Tim Staley

CHAPTER 7

Appendices

7.1 Release Procedure

7.1.1 Version Numbering

Releases have a `<major>.<minor>.<patch>` versioning scheme, and broadly follow the Semantic Versioning conventions:

- Changes to the major version number indicate backwards-incompatible changes to the feature set and/or interface;
- Changes to the minor version indicate backwards-compatible feature enhancements;
- Changes to the patch version indicate backwards-compatible bug fixes.

There is no formal end-of-life for Comet releases, but generally only critical bugs will be fixed in major or minor versions earlier than the most recent release.

7.1.2 Making a Release

Major and minor releases happen on a branch named `release-N.M`, where `N` is the major version number and `M` the minor. Create the branch as follows:

```
$ git checkout -b release-N.M
```

Patch releases happen on existing branches. Check out the branch as follows:

```
$ git checkout release-N.M
```

The release will correspond to a particular commit in which we set the version number. After all the other commits which will constitute the release have been committed to the release branch, edit the file `comet/__init__.py` and set the `__version__` variable appropriately. Also make sure the release history page `docs/appendix/history.rst` contains the date of the new release. Commit this change with an appropriate log message:

```
$ vim comet/__init__.py
$ vim docs/appendix/history.rst
$ git commit comet/__init__.py -m "Set version N.M.P"
```

Next tag the release with the version number:

```
$ git tag -a "N.M.P" -m "Comet release N.M.P"
```

Push everything, including the tag, to GitHub:

```
$ git push --tags origin release-N.M
```

Visit the ReadTheDocs Dashboard and ensure that the release branch is marked as active (i.e. that documentation for that branch will be built). Normally, the most recent release should be marked as the "default version".

Push an update to PyPI. Should be as simple as:

```
$ python setup.py sdist upload.
```

Chance back to the `master` branch and increment the version number to indicate that it is now a pre-release of the next version of Comet (e.g., `N.M+1.0-pre`). Make sure that the release history is correct here too. Commit and push.

E-mail an announcement to the IVOA Time Domain Interest Group mailing list.

7.2 OpenPGP Signatures, Comet and VOEvents

This document, originally written in August 2012, provides some background on the implementation of event and signature authentication in Comet. It is provided here for historical purposes only: please refer to current documentation for up-to-date information on how to use authentication in Comet.

7.2.1 Introduction

The basic VOEvent packet structure provides no guarantees as to the integrity of the data contained: it does not guarantee the identity of the event author or that the event has not been tampered with in transport. It is the position of the author that this information is fundamental: as we move to a future of autonomous, automatic telescopes, it is essential that valuable obsevating time and resources are not triggered by malicious or mistaken VOEvent messages.

There is widespread agreement that the solution is cryptographic: some form of digitial signature is applied to the event by its author, and this can then be verified by the receiver before the event is acted upon. Unfortunately, the application of cryptographic signatures to XML documents is non-trivial: the "standard" solution defined in the W3C recommendation XML Signature Syntax and Processing (hereafter "XML-DSig") is long, complex and widely criticised, and library support for many common programming langauges is hard to come by. Further, adopting the "enveloped" signature system described by XML-DSig would require changes to the existing VOEvent schema, while the alternative "detached" mechanism introduces extra complexity for transport protocols.

Work has already been done on this topic within the VOEvent community. Allen described the application of XML-DSig to VOEvent messages, while Denny proposed an alternative approach based on the OpenPGP system. Denny describes the infrastructure around the OpenPGP system in some detail, and the reader is encouraged to familiarize themself with that document for an overview of the situation.

It is the opinion of the present author that neither of these systems provide a panacea for the issues surrounding VOEvent authentication. However, the complexity surrounding XML-DSig makes the barrier-to-entry very high. Until either the XML-DSig technology matures to overcome the technical criticisms and lower the development effort required, or the OpenPGP solution is demonstrated to be inadequate to the requirements of the community, Comet development will concentrate on adding support for OpenPGP signatures.

7.2.2 Infosets, Serialization and Canonicalization

It is important to distinguish between the information contents of a VOEvent – the "infoset" – and a particular representation of that information. The same infoset can be described by many different VOEvent serializations, all equally valid according to the relevant standards. The differences may be as simple, for instance, as a change in the white space used within the document.

All of the cryptographic signature systems discussed apply at the fundamental level to a stream of bytes. Therefore, while changing the serialization of a particular infoset may be valid within the scope of the VOEvent standard, it will invalidate a signature which has been calculated over the original serialization.

The XML-DSig standard attempts to overcome many of these limitations by invoking a "canonicalization" process which transforms the infoset into a standard representation which both the signing and validating party can unambiguously agree upon. Unfortunately, this procedure is complex and can be slow and fragile.

OpenPGP makes no attempt to canonicalize: it simply provides a signature which applies to the stream of bytes supplied to it. This means that it is not possible to use a signature calculated over one serialization of a VOEvent to validate the contents of another serialization, *even if the information content is identical*. While this is not relevant for simple transmission from an author through one or more brokers to a receiver, it does significantly limit the more general applicability of the signature. For example, it would not be possible to run a service which receives signed VOEvents, deserializes the information a store such as a relation database, and then later reconstitutes them as XML including a valid signature from the original author.

7.2.3 Design Goals

Given the caveat above, the goals for the system set out by this document necessarily limited.

- The aim of the system being proposed is to make it possible to append an OpenPGP signature to an XML element being transmitted using the TCP-based VOEvent Transport Protocol (VTP).
- The signature enables the receiver of the element to verify the identity of the sender, using the standard OpenPGP "web of trust" principle. This document does not describe how the systems at either end of the transmission should manage their keychains, nor does it mandate what the receiver should regard as sufficiently trustworthy: these decisions should be made according to local requriements.
- The signature is valid for the particular bytestream being transmistted, and is invalidated by any manipulation or reserialization of that bytestream.
- The signature should not interfere with processing of the VOEvent by systems which do not understand or (wish to) participate in the authentication infrastucture: these should be able to treat the event as if it were unsigned.
- The application or removal of the signature should not in any way alter the contents of the XML element being transmitted.

Note that the above goals make this proposal suitable for use not only in signing VOEvents transmitted by VTP but also for use in the subscriber authentication scheme defined in the VTP standard.

7.2.4 The Cleartext Signature Framework

Denny proposes that VOEvents should be signed using the cleartext signature framework defined in RFC 4880 (OpenPGP Message Format). However, further investigation demonstrates the shortcomings of this technique, and it is here that the present document diverges from Denny's proposal.

RFC 4880 section 7 states

> this [cleartext signature] framework is not intended to be reversible.

In other words, cleartext signatures are transformational: applying one to an XML document could alter its contents. This renders it unsuitable according to the *Design Goals* outlined above.

It is worth noting that the circumstances under which cleartext signatures are transformation are quite limited: they concern escape sequences applied to lines starting with a dash "-" (0x2D). The cleartext signature framework is therefore likely to work perfectly well for the vast majority of VOEvents, but the chance of an error is ever-present.

7.2.5 Implementation Details

To meet the requirements of the section above, this document proposes signing relevant XML elements using a detached signature. A detached signature is non-transformational over the text being signed, and therefore avoids the problems described above.

The cleartext signature framework makes it clear exactly which bytes it is being applied over by the use of delimeters such as `=====BEGIN PGP SIGNED MESSAGE=====`. These do not apply to a detached signature. Therefore, we propose that the contents of the signed element consist of all bytes from the opening < to the closing > of the element being signed. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This text is not signed -->
<voe:VOEvent xmlns:voe="http://www.ivoa.net/xml/VOEvent/v2.0" ..>
  <!-- all of this text is signed -->
</voe:VOEvent>
```

In this case, all of the text from the first character of the string `<voe:VOEvent` to the last character of the string `</voe:VOEvent>` is signed, but no bytes outside those delimeters are included.

The signature is ASCII-armoured and appended to the message text as an XML comment. XML comments are started by the string `<!--` and closed by the string `-->`. With XML comments, the string `--` is forbidden. The string `-----` is used to delimit ASCII-armoured OpenPGP signature blocks. Within the context of the signed XML element, therefore, the sender must globally replace `-----` with the string `=====`. This substitution must be reversed by the receiver before the ASCII armoured signature is decoded. All other characters which are permitted in ASCII armoured OpenPGP signatures are also valid within XML comments, so no other substitution is required.

An example of a signed VOEvent with the above substitution performed is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This text is not signed -->
<voe:VOEvent xmlns:voe="http://www.ivoa.net/xml/VOEvent/v2.0" ..>
  <!-- all of this text is signed -->
</voe:VOEvent><!--
=====BEGIN PGP SIGNATURE=====
Version: GnuPG v1.4.12 (Darwin)

iQEcBAABAgAGBQJQIpY+AAoJEA7iIKe6Xi++k1AH/jW+7ql3coxbvJV41fhFTHOr
dPv+4woSXPvZXX2s3D0SEfSvtE2ofuQlzrGojGYgqZ9gwJS8/bjGGehTr29jA50e
92kYGenaCtti7BhatPVOwLETTsIx5Yj/3sbuIQhL8mWPW9oO6/0VNnbefaqZ7KZp
oBb8T3y2wkVF0Odz1lLKCVVyGZWdXM77m4PeVQeH8/6yqhrFl4npUPpR7Y4020+U
XkqZnERprPfiKF4j/OQpn4rtsKFlxwLgVUgalPAav0OjYyDjZrTG7vn4ZFCrInIT
F5P990K1jvSuA8TD7xUXZmceEM3yHm+/x5f5vCe6pZvRAsFZqAkfm11v0pxr5K4=
=nZgJ
=====END PGP SIGNATURE=====

-->
```

This system is unambiguously defined only when events are transmitted according to the VTP system, which specifies that only a single VOEvent or transport element is transmitted in each transaction. If multiple root-level XML elements

were to be transmitted, it would be ambiguous as to which the OpenPGP signature referred. This is therefore forbidden by the protocol.

7.2.6 Software

This system relies on the OpenPGP standard as set down in RFC 4880. Various implementations of the OpenPGP standard are available. All tests carried out while writing this document have been carried out using the GNU Privacy Guard, which is freely available and licensed under the GNU General Public License.

The Dakota VOEvent Tools provide a working implementation of the earlier proposal by Denny.

A version of Comet with basic support for this system is now being tested, and it will be merged into the released version soon. A preview version is available to interested parties on request.

7.2.7 Performance

The performance implications of this system are not negligible. The cryptographic operations obviously require some computation. Further, by design, there is no GnuPG shared library: signing or verifying operations cannot be handled in-process and instead involve forking a separate `gpg` executable.

The time taken for signing and verification obviously varies significantly both with the size of the data being signed and the key used for signing. Informal tests on a modest, 2009-vintage laptop running OS X 10.8 and GnuPG 1.4.12 indicate that signing a typical VOEvent message takes on the order of 0.1 seconds, including spawning the `gpg` executable, while verifying that signature takes around 0.01 seconds. On server grade hardware, one would imagine that this time would be substantially reduced.

Released versions of Comet are available for *download*, while the latest development version can be obtained from the GitHub repository.

Feedback is always welcome. For bug reports and feature requests, please use the issue tracker.