Technical Report NEESgrid-2002-03
www.neesgrid.org

Version 1.0
Last modified 3/1/2002

NEESgrid TeleOperations Protocol Server Design

Laura Pearlman[1]

[1] University of Southern California, Marina del Rey, CA 90292

Feedback on this document should be directed to neesgrid-si@neesgrid.org

Table of Contents

1 Introduction

A typical earthquake engineering experiment today (Figure 1) involves at least one data acquisition and control system (DAQ), zero or more *control points* (shake tables, actuators, etc.) and one or more *sensors* (accelerometers, strain gauges, etc). A DAQ sends control messages to control points and/or receives data from sensors; a typical experiment might involve one DAQ that sends control messages to a shake table (or to a small number of actuators) and receives data from several sensors, and a second DAQ that receives data from many additional sensors.

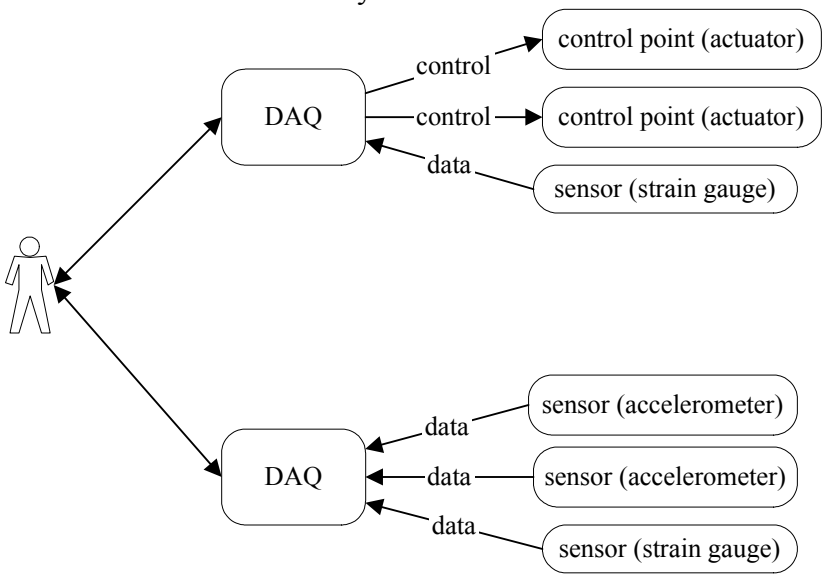

Figure 1: An experiment (without NTOP)

The experiment is typically controlled by a person via the user interface provided by each DAQ. In hybrid experiments, the experiment (the physical simulation) is controlled by a software simulation via an interface particular to the DAQ system being used.

The purpose of the NEESgrid TeleOperations Protocol (NTOP) service[1] is to provide a common protocol that can be used by remote applications (as shown in Figure 2) to interact with a running experiment over NEESgrid.

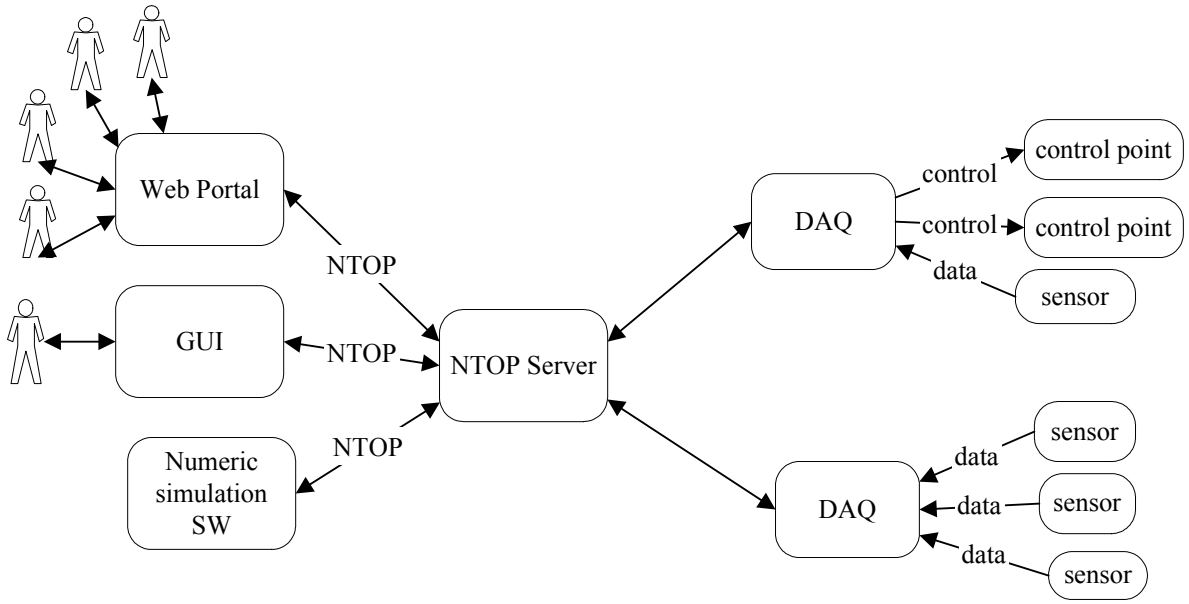

Figure 2: An experiment using an NTOP server

Specifically, the NTOP server should:

- accept control requests and forward them to the DAQ;
- accept requests to "subscribe to" (or "unsubscribe from") the streaming data for a sensor;
- provide access control for the requests described above;
- handle requests from multiple remote clients simultaneously;
- control (and receive data from) multiple local experiments simultaneously;
- support dynamic configuration (adding/deleting control points and sensors, and associating individual control points and sensors with different experiments over time, responding to queries about which experiment is associated with a specific sensor or which sensors are associated with a specific experiment, etc.); and
- support decimation of sensor data [I'm not sure exactly what the requirements are here -- do we need to allow the user to specify an arbitrary decimation factor? Do we just support a flag for whether the user gets decimated or non-decimated data, and determine the factor ourselves? Or do we not even give the user an option, and determine that high-volume data streams are always decimated?]

The NTOP server will not be the only interface to send control requests or receive data from the local DAQs. We expect that, for safety reasons, sites will continue to control most of their experiments using their local DAQ interface; streaming data from these experiments should still be available via the NTOP server. We also do not anticipate the use of the NTOP server to collect data for use after the experiment has been completed; the NEESgrid architecture specifies separate protocols and services for that purpose.

[Note: I don't think the NTOP server needs to support queries about sensor/control point metadata, such as calibration info, but I'm not 100% sure.]

2 Protocols Used by the NTOP Server

2.1 Communication between the NTOP Server and Clients: NTOP

NTOP is the protocol used between the NTOP server and client applications. It should be relatively easy to parse and extend; for that reason we believe that it should be encoded in SOAP and should be versioned [or does SOAP itself require versioning, i.e. do the SOAP headers call for a service-specific version number?].

Many of the NTOP requests will involve a particular sensor or control point. The remote user should not need to know the particular local configuration in order to make these requests; in an NTOP request, a sensor (or control point) should be expressed in terms of the name of the experiment and the name of the sensor (or control point) relative to that experiment.

NTOP requests include:

- **Subscribe** (subscribe to a data channel)
  - inputs: sensor name (experiment name plus relative sensor name), listener ip/port
  - result: data from the named sensor is sent to the listener ip/port until an *unsubscribe* request is received or the experiment trial is finished. For each data point, the output should include:
    - the data channel name (experiment/sensor name), to support applications that receive data from more than one sensor on the same listener port
    - a timestamp (relative to the start of the trial), to support synchronization within a trial
    - the actual data value (in units of voltage -- unit conversion data should be available in the metadata database)
  - reply: success or failure
  - access control requirements: *read* permission on the data channel (access control is discussed in Section 3).
  - other possible extensions:
    - decimation (results in only a sampling of data being sent to the listener channel),
    - other stop criteria (send at most N data points, stop at a particular time, etc.),
    - alternate output format (some applications may expect columnar input rather than SOAP-encapsulated data; it might make sense to offer this as an option. Or of course we could provide a simple filter to run on the client side).
- **Unsubscribe** (cancel a subscription):
  - inputs: sensor name, listener ip/port, as in *subscribe* request
  - result: data from the named sensor will no longer be sent to the specified listener ip/port.
  - reply: success or failure
  - access control: must be same entity that did the associated *subscribe* request.
- **Control** (send a request to a control point):
  - inputs: control point name (experiment name plus relative control point name), requested action and parameters (e.g., "move actuator forward 1 cm" or "run shake table with this waveform")

  - outputs: success or failure, plus detailed output (e.g. "actual distance moved was 1.05 cm")
  - note: obviously, we need to work out many more details about the actions/parameters.
  - access control: at least something like *control* permission on the control channel. Maybe more fine-grained actions as more details are worked out.
- **Query sensor/control instantiation** (maybe?)
  - inputs: sensor or control point name (as would be specified to the *subscribe* or *control* request)
  - output: true or false -- indication of whether there is a physical sensor/control point associated with this name, and possibly more detailed information if the local DAQ supports this kind of query
  - access control: *query* permission on the sensor or control point
  - note: this request is probably not essential -- the metadata database should be used for basic information, such as whether a sensor is a strain gauge or an accelerometer.

2.2 Communication between the NTOP Server and Local DAQs

The protocols used to communicate between the NTOP server and the local DAQs may vary based on the capabilities of each local DAQ; it is, therefore, important to structure the NTOP server in such a way that it is relatively easy to install different drivers for different DAQs (and possibly to configure the NTOP server to use different drivers for different DAQs simultaneously). However, we will attempt to define a single protocol that will be supported by several of the most common DAQ systems, and we'll provide a driver for that protocol.

This protocol should support:

- Establishing connections for sensors and control points -- either in advance, before a trial begins, or (if supported by the DAQ) creating and shutting down connections as needed during the course of a trial.
- Streaming sensor data from the DAQ to the NTOP server.
- Control requests from the DAQ to the NTOP server.
- Notifying the NTOP server when a trial is finished.

2.3 Dynamic Local Administration

Some day-to-day administration tasks will need to be done by local site personnel. Each of these tasks could be accomplished via an administrative protocol, or by updating local configuration files (and either signaling the NTOP server to reread them, or having the server poll those files periodically). Of course, if an administrative protocol is chosen, the server should make sure that the configuration change will persist if the server is restarted (e.g., by writing the information to a database). These tasks include:

- Assigning a sensor or control point to an experiment: mapping an (experiment name, relative sensor (or control point) name) pair to a (DAQ, channel) pair.
- Freeing a sensor or control point: removing a sensor or control point name mapping.
- Granting or revoking permissions.

3 Access Control

Local site administrators will be able to grant permissions to perform the *subscribe*, *control*, or *query* actions (see section 2.1) on control points and sensors, using a local configuration file. If the local site administrators grant these permissions to a CAS server, community administrators may grant these permissions to other community members. For example, a typical scenario might be:

1. Alice is designated as a site administrator at *site.edu*. She uses the local configuration file to grant, to the CAS server, permission to perform all actions on all control points and servers. The central CAS administrators for the NEESgrid community grant her permission, within the CAS server, to create CAS resources under *site.edu*.
2. Alice finds out that Bob is the PI for experiment XYZ, which will run on resources at site.edu. She uses CAS to:
   - create an object ("/site.edu/XYZ") that represents that experiment at her site, and
   - grant permission to Bob to create CAS objects within that experiment (such as "/site.edu/XYZ/sensor1") that represent sensors and control points within that experiment and to grant permissions on them.
3. Bob then either:
   - create CAS objects for each sensor and control point and use CAS to grant permissions on them ("this group of people can read sensor /site.edu/XYZ/sensor1"), or
   - grant permissions that span the entire experiment ("this group of people can read all the sensors in experiment /site.edu/XZY").
4. In the meantime, Alice performs the other configuration tasks associated with setting up an experiment (e.g., at her local site, she creates the mappings that map "experiment XYZ, sensor 1" to "DAQ 2, channel 4").

4 NTOP Server Architecture

4.1 Overview

The NTOP server will consist of these modules:

- The *i/o and authentication module* handles low-level network i/o and authentication (and parses any proxy restrictions present in the authentication credential); this is a standard Globus component.
- The *NTOP parsing module* parses an NTOP request.
- The *authorization module* determines whether or not an NTOP request is authorized, based on the local configuration and any proxy restrictions.
- The *subscription module* handles *subscribe* requests: it communicates with the *data streaming module* to locate (or create) the appropriate *input data stream* and *output data stream*, and creates a mapping between the two. It also handles *unsubscribe* requests in a similar fashion

- The *data streaming module* reads input data streams from the *DAQ communication module*, formats the data according to the NTOP protocol, and writes it out to the appropriate *output data streams*.
- The *control request module* handles control requests by forwarding them to the *DAQ communication module*.
- The *administration* module handles local administration requests.

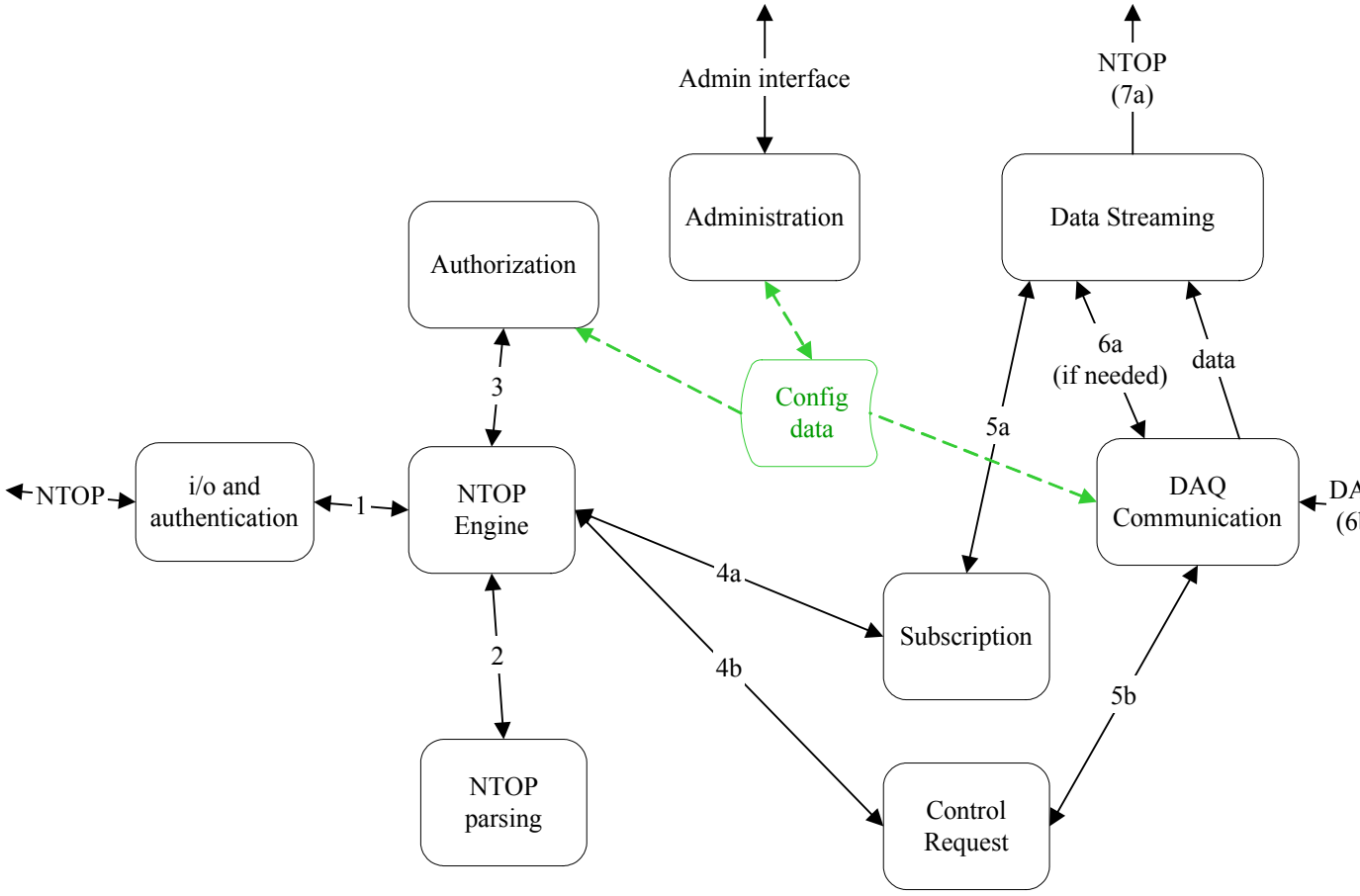

Figure 3: NTOP server modules

For example, a subscription request is read by the i/o and authentication module and passed to the NTOP engine (1), which sends it to the NTOP parsing module (2) for parsing, the authorization module (3) to check authorization, and then to the subscription module (4a). The subscription module forwards the request to the data streaming module (5a), which locates the appropriate input data stream (or, if necessary, contacts the DAQ communication module (6a) to create one), creates or locates the appropriate output data stream, creates a mapping between the two, and returns status to the NTOP engine, which communicates the result to the client.

The DAQ communication module also reads data from the local DAQs; when data arrives, the DAQ communication module parses it and forwards it (as an input data stream) to the data streaming module. The data streaming module checks to see which

output data streams are subscribed to that input stream, formats the data, and sends it out to each waiting output stream.

4.2 Modules

4.2.1 Simple Modules

The i/o and authentication module performs raw i/o and authentication. The NTOP parsing module translates raw input into a data structure containing a parsed NTOP request. The authorization module combines that data structure with the configured access control policy (including any policy from the authentication credential) and gives a yes/no authorization decision. The control request module passes a request on to the DAQ module and returns the result [this may become more complicated when we know more about the control interface]. The subscription module translates a subscribe or unsubscribe request into a request for the data streaming module and returns the result.

4.2.2 The Data Streaming Module

The Data Streaming module performs two basic functions:

1. It maintains mappings of:
   - input stream names (experiment/sensor names) to input streams (data structures)
   - output stream names (ip address/port) to output streams (data structures)
   - input streams to output streams (subscriptions)
2. It reads data from the DAQ Communication module and forwards it out to clients.

It can be divided into two modules, a Data Stream Mapping Module and a Data Stream Flow Module. The Mapping module communicates with the Subscription Module (to handle subscribe and unsubscribe requests), answers "what output streams are subscribed to this input stream" queries from the Flow Module, and handles "remove all subscriptions to this input stream" requests from the DAQ Communication Module (these requests would be sent when an input stream closes down; e.g., when an experiment trial ends). The Flow module communicates with the DAQ Communication module (which sends it data) and with clients. [Note: it may make sense to subdivide the Flow module into submodules to do data formatting, decimation, etc.]

4.2.3 The DAQ Communication Module

The DAQ Communication Module performs these functions:

1. It handles requests from the Data Stream (Mapping) Module to create an input stream; that is, to take an (experiment name, relative name) pair, and find the corresponding DAQ name and channel name and do whatever is appropriate to get input from that channel.

2. It reads and parses input streams and forwards that data to the Data Stream (Flow) module).
3. It notifies the Data Stream (Mapping) module when an input stream has closed.
4. It reformats requests from the NTOP Control module and sends them to the appropriate control channel.

It can be divided into:

- A DAQ Driver, to handle the parsing of input streams, the DAQ-specific setting up of input and control streams (e.g., sending a request to a server running on a DAQ, or simply listening on a port and waiting an administrator to manually configure the DAQ), and the translation of control requests into something that the DAQ will understand; and
- A DAQ Channel Mapping Module, which handles the mapping of input stream names to physical DAQ names and channel names, and the selection of the appropriate driver for each DAQ.

4.3 Threading

The NTOP server will almost certainly need to be threaded for performance. This raises questions about what level of consistency is required; we may need to consult with the user community for answers.

1. *Should we guarantee that all data sent to an output stream (possibly from several input streams) be sent in the order that it was read?* Probably not, as the data is timestamped.
2. *Should we guarantee that all data sent from one input stream to one output stream be sent in the order that it was read from the input stream?* Again, probably not, as the data is timestamped.
3. *Should we guarantee that all control requests sent from one client connection to one control point be forwarded to that control point in the order received?* Probably.
4. *Should we guarantee that all control requests sent from one client connection to any of the control points in an experiment be forwarded in the order received?* Again, probably.
5. *Should we guarantee that all control requests from all clients to one control point be forwarded in the order received?* How likely is it that more than one client will be sending control requests to the same control point during one experiment?

Acknowledgments

This work was supported by the NSF NEESgrid project.

Bibliography

[1] Kesselman et al., NEESgrid System Architecture Version 1.0