

The Data Transport Network Turns 15!

Dr. Todd Valentic

`todd.valentic@sri.com`

**Center for Geospace Studies
SRI International**

**Polar Technology Conference
US Naval Academy
Annapolis, MD
April 2 – 4, 2013**

Remote Data Retrieval

NSF Research Facility in Sondrestrom, Greenland



What Are We Trying To Do?

- Collect data from instruments
- Transfer data offsite to multiple users
- Monitor instrument health and status
- Remotely schedule and control
- Automate data processing
- Manage bandwidth



Small Systems Have The Same Needs



What Makes This Difficult?

- Bandwidth constraints
- Unreliable or on-demand connections
- Instrument operations
- Multiple data sources
- Scale to multiple instruments and sites
- Legacy hardware and software



The Data Transport Network

Open-Source, NSF Information Technology Project

A system to access, process and transfer data from multiple instruments over unreliable networks.

Access: common method to organize and access data

Process: monitor status, compute results, trigger actions

Transfer: transmit data between sites

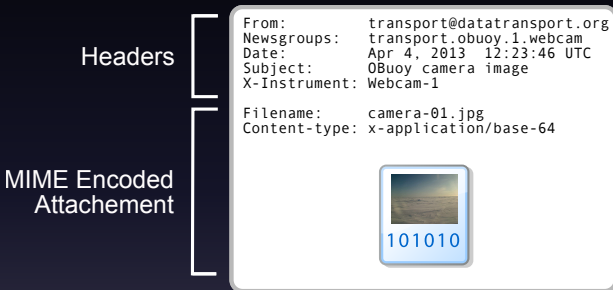
Multiple instruments: many heterogeneous data feeds

Unreliable network: store-and-forward, on demand

First code written in 1998 – 15 years and going strong!

How Does It Work?

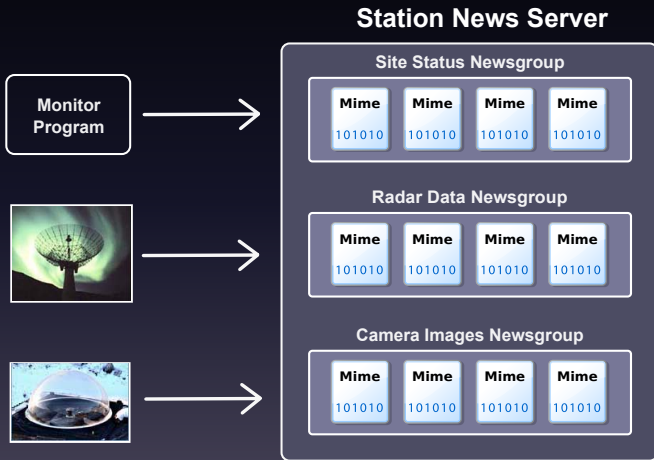
Usenet Messages and Attachments



- Data files are sent as attachments
- Headers provide metadata
- Any type of data can be sent (text, images, binaries)

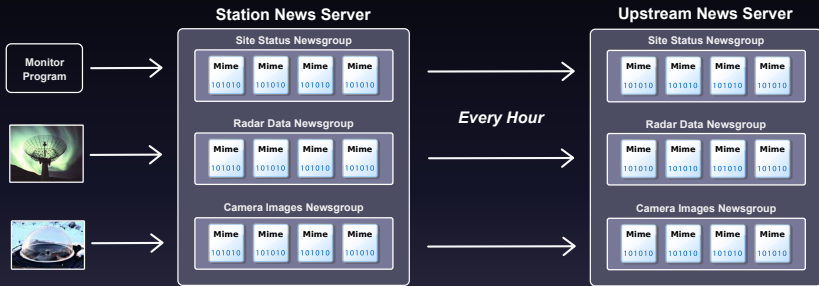
Messages Stored In Newsgroups

Instruments Post Data Files Into Separate Newsgroups



News Server Message Exchange

Store and Forward



- Periodic transfers (typically every hour)
- Access controls
- Bidirectional
- Data replication

Buoy Server Newsgroups

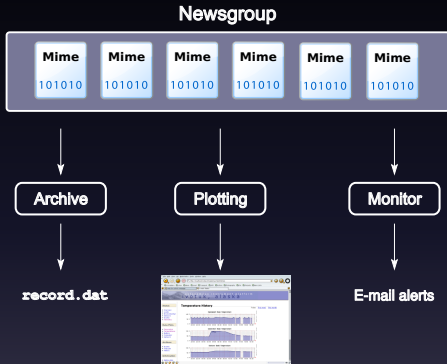
228 newsgroups for 3 different buoy projects

transport.obuoy.00000009211ffc9.tobuoy
transport.obuoy.0000000922159fa.frombuoy.camera
transport.obuoy.0000000922159fa.frombuoy.campbell
transport.obuoy.0000000922159fa.frombuoy.co2
transport.obuoy.0000000922159fa.frombuoy.doas
transport.obuoy.0000000922159fa.frombuoy.gps
transport.obuoy.0000000922159fa.frombuoy.iridium
transport.obuoy.0000000922159fa.frombuoy.logs
transport.obuoy.0000000922159fa.frombuoy.ozone
transport.obuoy.0000000922159fa.frombuoy.sbc
transport.obuoy.0000000922159fa.frombuoy.schedules
transport.obuoy.0000000922159fa.frombuoy.system
transport.obuoy.0000000922159fa.frombuoy.updates
transport.obuoy.0000000922159fa.tobuoy
transport.obuoy.000000092250547.frombuoy.camera
transport.obuoy.000000092250547.frombuoy.campbell
transport.obuoy.000000092250547.frombuoy.co2
transport.obuoy.000000092250547.frombuoy.doas
transport.obuoy.000000092250547.frombuoy.gps
transport.obuoy.000000092250547.frombuoy.iridium
transport.obuoy.000000092250547.frombuoy.logs
transport.obuoy.000000092250547.frombuoy.ozone
transport.obuoy.000000092250547.frombuoy.sbc
transport.obuoy.000000092250547.frombuoy.system
transport.obuoy.000000092250547.frombuoy.updates
transport.obuoy.000000092250547.tobuoy
transport.obuoy.00000009225cd12.frombuoy.ack
transport.obuoy.00000009225cd12.frombuoy.camera
transport.obuoy.00000009225cd12.frombuoy.campbell
transport.obuoy.00000009225cd12.frombuoy.co2
transport.obuoy.00000009225cd12.frombuoy.doas
transport.obuoy.00000009225cd12.frombuoy.gps
transport.obuoy.00000009225cd12.frombuoy.iridium
transport.obuoy.00000009225cd12.frombuoy.logs
transport.obuoy.00000009225cd12.frombuoy.ozone
transport.obuoy.00000009225cd12.frombuoy.sbc
transport.obuoy.00000009225cd12.frombuoy.schedules
transport.obuoy.00000009225cd12.frombuoy.system

transport.obuoy.status.history
transport.obuoy.status.online
transport.usna.history
transport.usna.icegoat-1c.argos
transport.usna.icegoat-1c.frombuoy.camera0
transport.usna.icegoat-1c.frombuoy.cameral
transport.usna.icegoat-1c.frombuoy.iridium
transport.usna.icegoat-1c.frombuoy.sbc
transport.usna.icegoat-1c.frombuoy.schedules
transport.usna.icegoat-1c.frombuoy.system
transport.usna.icegoat-1c.frombuoy.updates
transport.usna.icegoat-1c.tobuoy
transport.usna.icegoat-2a.tobuoy
transport.usna.icekid-1.frombuoy.ack
transport.usna.icekid-1.frombuoy.camera0
transport.usna.icekid-1.frombuoy.cameral
transport.usna.icekid-1.frombuoy.iridium
transport.usna.icekid-1.frombuoy.sbc
transport.usna.icekid-1.frombuoy.schedules
transport.usna.icekid-1.frombuoy.system
transport.usna.icekid-1.frombuoy.updates
transport.usna.icekid-1.tobuoy
transport.usna.icekid-2a.frombuoy.ack
transport.usna.icekid-2a.frombuoy.camera0
transport.usna.icekid-2a.frombuoy.cameral
transport.usna.icekid-2a.frombuoy.gps
transport.usna.icekid-2a.frombuoy.hydrophone
transport.usna.icekid-2a.frombuoy.iridium
transport.usna.icekid-2a.frombuoy.sbc
transport.usna.icekid-2a.frombuoy.schedules
transport.usna.icekid-2a.frombuoy.system
transport.usna.icekid-2a.frombuoy.updates
transport.usna.icekid-2a.tobuoy
transport.usna.icekid-3.frombuoy.ack
transport.usna.icekid-3.frombuoy.camera0
transport.usna.icekid-3.frombuoy.cameral
transport.usna.icekid-3.frombuoy.iridium
transport.usna.icekid-3.frombuoy.sbc

Newsgroup Features

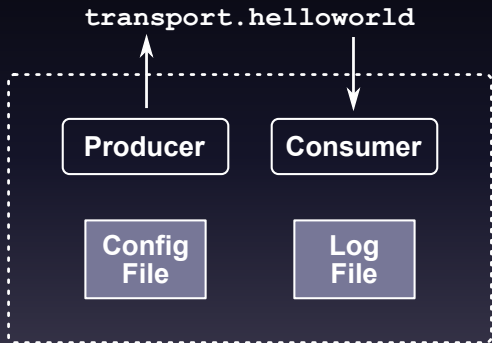
Publish and Subscribe



- Single producer, many consumers
- Copies of data are accessed
- Short-term history, automatic expiration (2 weeks)
- “Market place” for data

Data Transport “Hello World”

A Producer/Consumer Example



Hello World Example

Print Message To Log File Every 15-seconds

helloworld.conf

```
[ProcessGroup]
clients: hello
```

```
[hello]
command: helloworld.py
```

helloworld.py

```
from Transport import ProcessClient

class Client (ProcessClient):

    def __init__(self,args):
        ProcessClient.__init__(self,args)

    def run(self):
        while self.wait(15):
            self.log.info('Hello world!')

if __name__ == '__main__':
    import sys
    Client(sys.argv).run()
```

```
transportctl start helloworld
```

```
[2013-04-03 01:13:43.327 INFO] helloworld: Starting client hello
[2013-04-03 01:13:43.413 INFO] helloworld: Logging in client: hello (pid=30148)
[2013-04-03 01:13:58.440 INFO] helloworld/hello: Hello world!
[2013-04-03 01:14:13.466 INFO] helloworld/hello: Hello world!
```

Hello World Example

Reading From The Config File

helloworld.conf

```
[ProcessGroup]
clients: hello
```

```
[hello]
command: helloworld.py
rate: 00:15
text: Hello PTC!
```

helloworld.py

```
from Transport import ProcessClient

class Client (ProcessClient):

    def __init__(self,args):
        ProcessClient.__init__(self,args)

        self.rate = self.getRate('rate')
        self.text = self.get('text')

    def run(self):
        while self.wait(self.rate):
            self.log.info(self.text)

if __name__ == '__main__':
    import sys
    Client(sys.argv).run()
```

Publisher Example

Posting To A News Group

helloworld.conf

```
[ProcessGroup]
clients: hello
```

```
[hello]
command: helloworld.py
rate: 00:15
text: Hello PTC!
post.newsgroup: transport.helloworld
```

helloworld.py

```
from Transport import ProcessClient, NewsPostMixin

class Client (ProcessClient,NewsPostMixin):

    def __init__(self,args):
        ProcessClient.__init__(self,args)
        NewsPostMixin.__init__(self)

        self.rate = self.getRate('rate')
        self.text = self.get('text')

    def run(self):
        while self.wait(self.rate):
            self.log.info(self.text)
            self.newsPoster.postText(self.text)

if __name__ == '__main__':
    import sys
    Client(sys.argv).run()
```

Subscriber Example

Reading From A News Group

helloworld.conf

[DEFAULT]

newsgroup: transport.helloworld
rate: 00:15

[ProcessGroup]

clients: hello poller

[hello]

command: helloworld.py
text: Hello PTC!
post.newsgroup: %(newsgroup)s

[poller]

command: poller.py
poll.newsgroup: %(newsgroup)s
poll.rate: %(rate)s

poller.py

```
from Transport import ProcessClient, NewsPollMixin
```

```
class Poller (ProcessClient,NewsPollMixin):
```

```
    def __init__(self,args):  
        ProcessClient.__init__(self,args)  
        NewsPollMixin.__init__(self,callback=self.process)
```

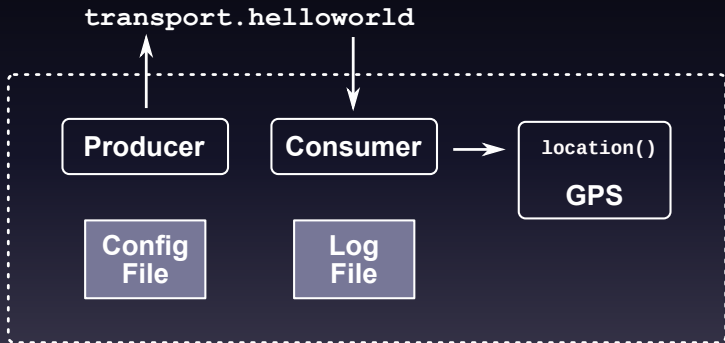
```
    def process(self,message):  
        text = message.get_payload()  
        self.log.info('Received: %s' % text)
```

```
if __name__ == '__main__':  
    import sys  
    Poller(sys.argv).run()
```

```
[2013-04-03 01:35:38.093 INFO] helloworld: Logging in client: poller (pid=631)  
[2013-04-03 01:35:38.592 INFO] helloworld: Logging in client: hello (pid=636)  
[2013-04-03 01:35:45.016 INFO] helloworld/hello: Hello PTC!  
[2013-04-03 01:35:53.142 INFO] helloworld/poller: Received: Hello PTC!
```


XML-RPC Server/Client Example

Synchronous Communications Between Programs



XML-RPC Server Example

GPS Position Service

helloworld.conf

[DEFAULT]

newsgroup: transport.helloworld
rate: 00:15

[ProcessGroup]

clients: hello poller gps

[hello]

command: helloworld.py
text: Hello PTC!
post.newsgroup: %(newsgroup)s

[poller]

command: poller.py
poll.newsgroup: %(newsgroup)s

[gps]

command: server.py
service.name: gps

server.py

```
from Transport import ProcessClient
from Transport import XMLRPCServerMixin
```

```
class GPSServer (ProcessClient,XMLRPCServerMixin)
```

```
    def __init__(self,args):
        ProcessClient.__init__(self,args)
        XMLRPCServerMixin.__init__(self)
```

```
        self.register_function(self.location)
```

```
    def location(self):
        lat = 87.6
        lon = 138.3
        return lat,lon
```

```
if __name__ == '__main__':
    import sys
    GPSServer(sys.argv).run()
```

Calling an XML-RPC Service

helloworld.conf

```
[DEFAULT]
newsgroup: transport.helloworld
rate: 00:15
```

```
[ProcessGroup]
clients: hello poller gps
```

```
[hello]
command: helloworld.py
text: Hello PTC!
post.newsgroup: %(newsgroup)s
```

```
[poller]
command: poller.py
poll.newsgroup: %(newsgroup)s
```

```
[gps]
command: server.py
service.name: gps
```

helloworld.py

```
from Transport import ProcessClient, NewsPostMixin
from Transport import DirectoryMixin
```

```
class Client (ProcessClient,NewsPostMixin,DirectoryMixin):
```

```
    def __init__(self,args):
        ProcessClient.__init__(self,args)
        NewsPostMixin.__init__(self)
        DirectoryMixin.__init__(self)
```

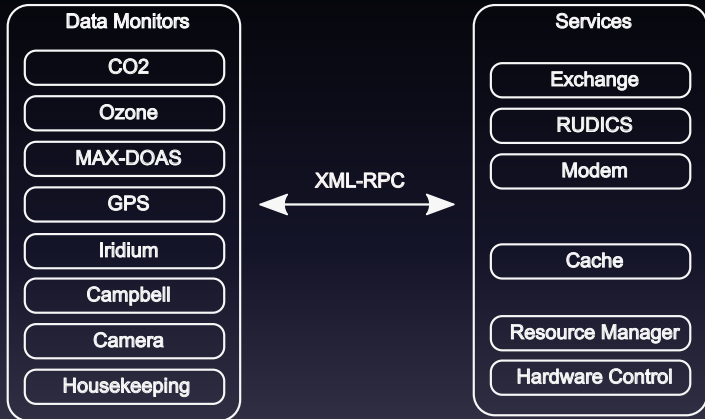
```
        self.rate = self.getRate('rate')
        self.text = self.get('text')
        self.gps = self.connect('gps')
```

```
    def run(self):
        while self.wait(self.rate):
            lat,lon = self.gps.location()
            msg = "Received: %s at (%s,%s)" % (self.text,lat,lon)
            self.log.info(msg)
            self.newsPoster.postText(msg)
```

```
[2013-04-03 01:40:00.025 INFO] helloworld/hello: Hello PTC!
```

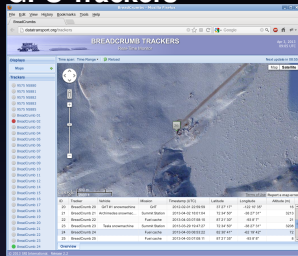
```
[2013-04-03 01:40:08.668 INFO] helloworld/poller: Received: Hello PTC! at (87.6,138.3)
```

Buoy System Architecture

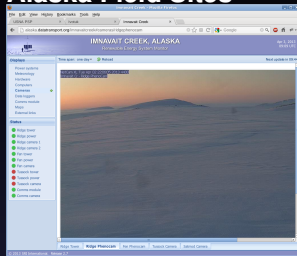


Example Uses

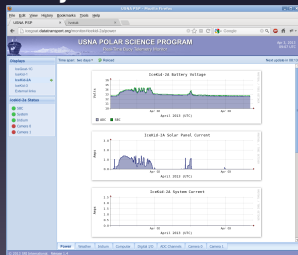
GPS Trackers



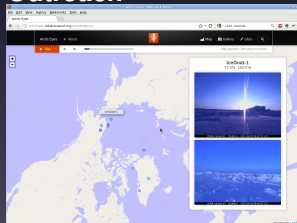
Alaska Field Sites



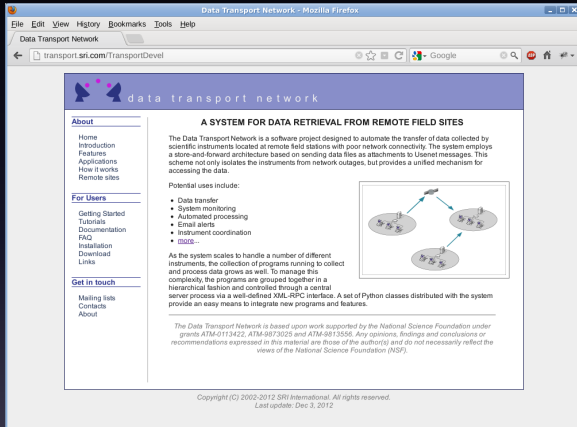
Buoys



Outreach



How To Get Started



The screenshot shows a Mozilla Firefox browser window displaying the Data Transport Network website. The browser's address bar shows the URL `transport.sri.com/TransportDev`. The website has a blue header with the text "data transport network" and a logo of two stylized figures. A left-hand navigation menu includes sections for "About", "For Users", and "Get in touch". The main content area features the heading "A SYSTEM FOR DATA RETRIEVAL FROM REMOTE FIELD SITES" and a detailed description of the system's purpose and architecture. A diagram illustrates the data flow between field stations and a central server. At the bottom, there is a copyright notice for SRI International, dated 2002-2012, with a last update of Dec 3, 2012.

About

- Home
- Introduction
- Features
- Applications
- How it works
- Remote sites

For Users

- Getting Started
- Tutorials
- Documentation
- FAQ
- Installation
- Download
- Links

Get in touch

- Mailing lists
- Contacts
- About

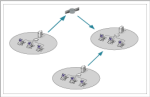
A SYSTEM FOR DATA RETRIEVAL FROM REMOTE FIELD SITES

The Data Transport Network is a software project designed to automate the transfer of data collected by scientific instruments located at remote field stations with poor network connectivity. The system employs a store-and-forward architecture based on sending data files as attachments to Usenet messages. This scheme not only isolates the instruments from network outages, but provides a unified mechanism for accessing the data.

Potential uses include:

- Data transfer
- System monitoring
- Automated processing
- Email alerts
- Instrument coordination
- [more...](#)

As the system scales to handle a number of different instruments, the collection of programs running to collect and process data grows as well. To manage this complexity, the programs are grouped together in a hierarchical fashion and controlled through a central server process via a well-defined XML-RPC interface. A set of Python classes distributed with the system provide an easy means to integrate new programs and features.



The Data Transport Network is based upon work supported by the National Science Foundation under grants ATM-0113422, ATM-9873025 and ATM-9813556. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Copyright (C) 2002-2012 SRI International. All rights reserved.
Last update: Dec 3, 2012

- **Development site:** `www.datatransport.org`
- **Email:** `todd.valentic@sri.com`