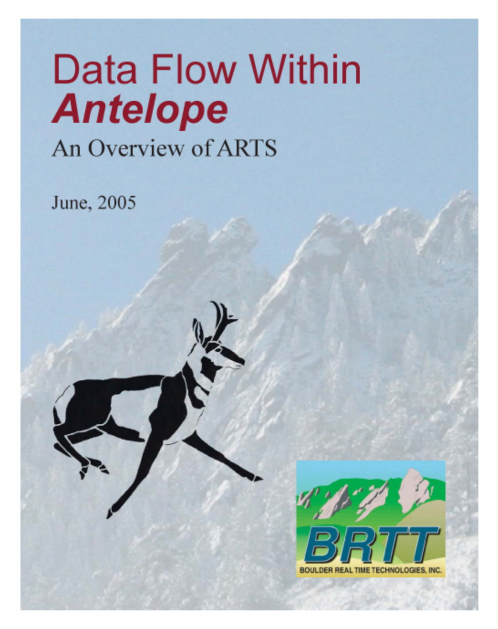


Data Flow Within *Antelope*

An Overview of ARTS

June, 2005

Data Flow Within *Antelope*

- Why should you care?
  - Antelope is the primary software for data acquisition, system monitoring, system control and configuration and initial data processing for the IRIS USArray/Earthscope project
  - Antelope also is available to IRIS member institutions and is being used as the primary software in a number of seismic networks in the US and around the world
  - For staff involved in the Earthscope projects, it is important to have some kind of understanding of the basic tools that are being used, such as Antelope
  - We have seen some problem situations which seem to us have arisen due to a misunderstanding about the Antelope tools, even sometimes by experienced Antelope users
  - *"Data Flow Within Antelope"* was written to address some of these problems and to clearly describe how and why Antelope was developed and how it works at its most fundamental level
  - We hope that all staffers and other interested parties will take the time to read the complete document

Data Flow Within ***Antelope***

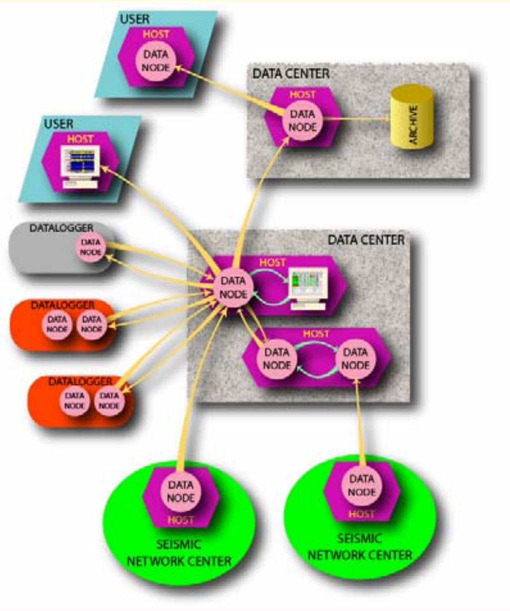
- Basic Design of the Antelope Real-time System (ARTS)
  - The Antelope software package
  - Brief history of development
  - Fundamental requirements
  - A quick review of the Antelope design
  - A list of basic Antelope capabilities
- ARTS Inner Workings
  - ORBs and **`orbserver`**s
  - ORB names and the ORB protocol
  - ORB – client communications
  - Pushes, pulls and state info
  - ORB packet ***srcname***s
  - Decoding and encoding ORB packets
  - Using **`orbstat`**
- An ARTS Example; the USARRAY Transportable Array Facility
  - The TA facility
  - A conceptual configuration for ARTS layout
  - Detailed ARTS configuration for the TA
  - A look at ORB packet data types in the TA ARTS

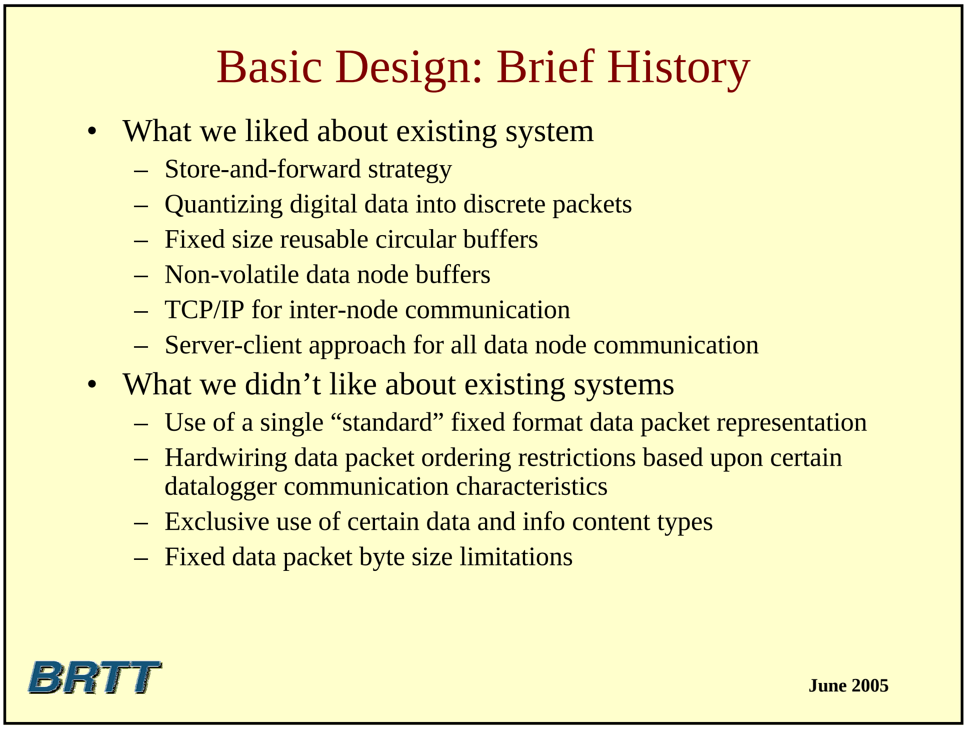
Basic Design: Antelope Software Package

- Antelope is a large software package
  - 480 programs and scripts
  - 70 software libraries
  - SUN/Solaris, x86/Linux, ARM-Xscale/Linux, Mac OS/X
- Supports both off-line batch processing using a database system and automated real-time processing (ARTS)
- Although Antelope can be configured to implement very complex and distributed systems, its basic design is simple

Basic Design: Brief History

- ARTS design started in mid-1990s
- Initial design based on existing systems and a set of initial requirements (derived from IRIS JSP and BBArray projects)
  - Needed a system that could work with a variety of dataloggers and acquisition systems
  - Needed a system that could accommodate non-time-series objects
  - Needed a system that could be easily distributed across many hosts in many geographic locations
  - Needed a system that was highly automated and robust to minimize human labor requirements



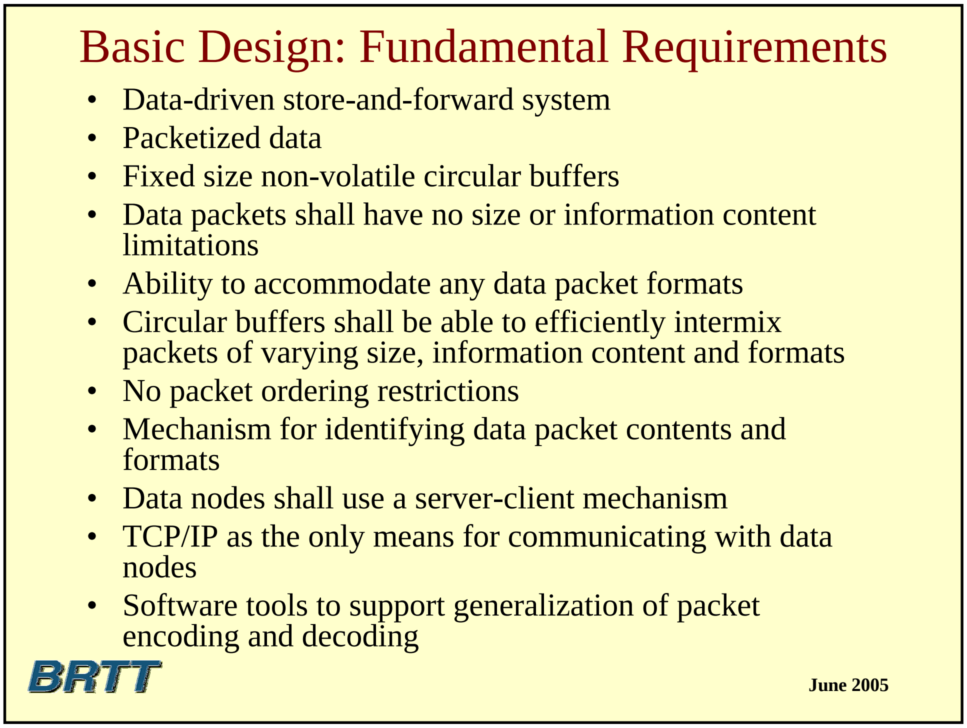


Basic Design: Brief History

- What we liked about existing system
  - Store-and-forward strategy
  - Quantizing digital data into discrete packets
  - Fixed size reusable circular buffers
  - Non-volatile data node buffers
  - TCP/IP for inter-node communication
  - Server-client approach for all data node communication
- What we didn’t like about existing systems
  - Use of a single “standard” fixed format data packet representation
  - Hardwiring data packet ordering restrictions based upon certain datalogger communication characteristics
  - Exclusive use of certain data and info content types
  - Fixed data packet byte size limitations

BRTT

June 2005



Basic Design: Fundamental Requirements

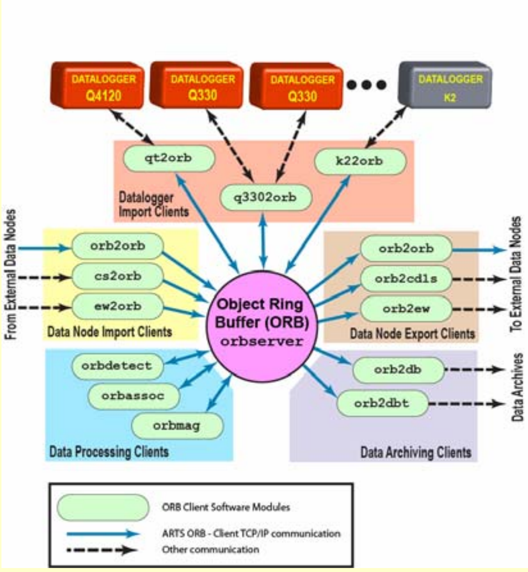
- Data-driven store-and-forward system
- Packetized data
- Fixed size non-volatile circular buffers
- Data packets shall have no size or information content limitations
- Ability to accommodate any data packet formats
- Circular buffers shall be able to efficiently intermix packets of varying size, information content and formats
- No packet ordering restrictions
- Mechanism for identifying data packet contents and formats
- Data nodes shall use a server-client mechanism
- TCP/IP as the only means for communicating with data nodes
- Software tools to support generalization of packet encoding and decoding

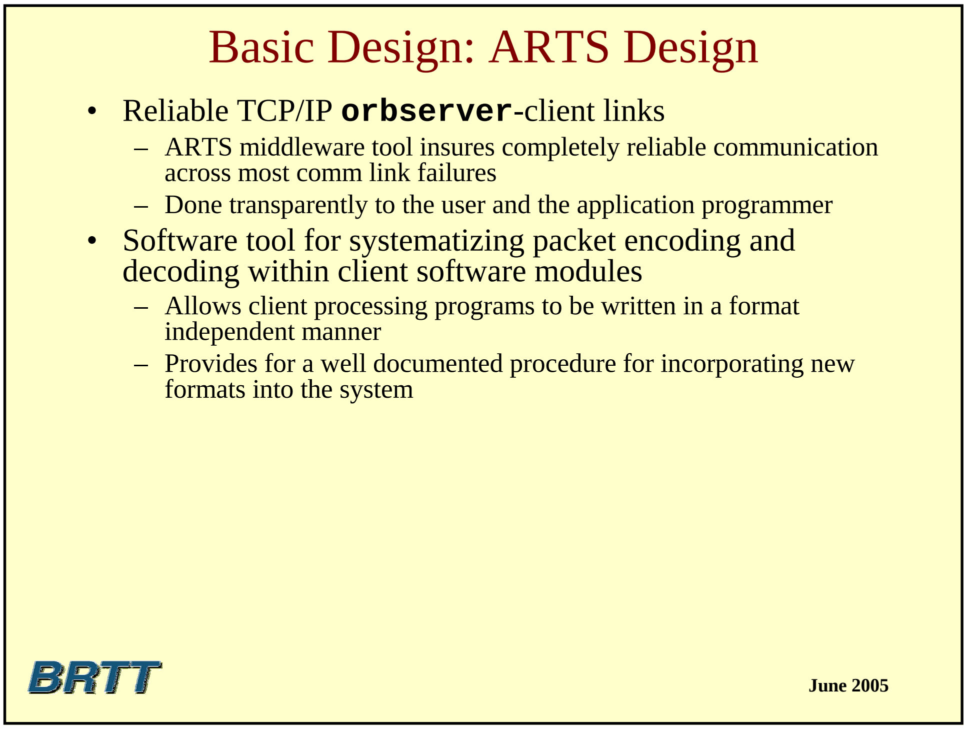
BRTT

June 2005

Basic Design: ARTS Design

- ORB and **`orbserver`**
  - ORB implemented as a circular non-volatile data store on disk
  - **`orbserver`** acts as a server for all ORB input and output
  - A single ORB can efficiently accommodate variable size, information content, format packets with no ordering restrictions
  - All client modules “talk” to ORBs only through its **`orbserver`** program using TCP/IP
- Packet subscription and control of read pointers
  - All ORB packets have a time tag and a ***srcname*** which is used to identify the packet origin, information content and format
  - The ***srcname*** is a simple ASCII string that is used as a means for client packet subscription to the **`orbserver`**
  - Clients can also cause the **`orbserver`** to position the read pointer according to time, `pktid` or relative location
- Note that **`orbserver`** never makes all attempt to decode packet payloads or use anything in the packet payloads



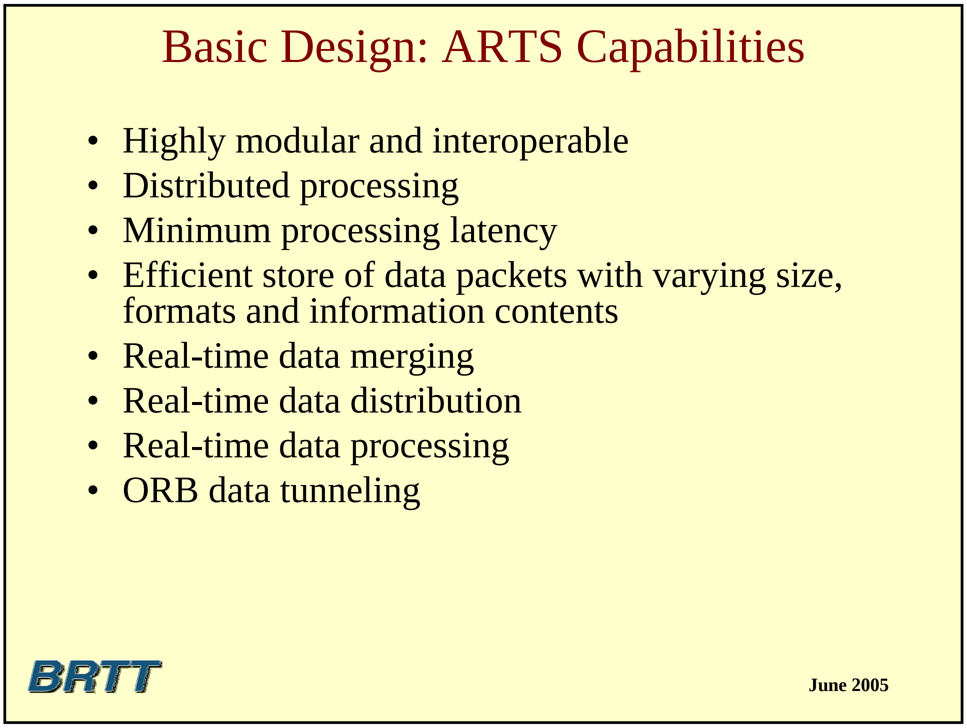


Basic Design: ARTS Design

- Reliable TCP/IP `orbserver`-client links
  - ARTS middleware tool insures completely reliable communication across most comm link failures
  - Done transparently to the user and the application programmer
- Software tool for systematizing packet encoding and decoding within client software modules
  - Allows client processing programs to be written in a format independent manner
  - Provides for a well documented procedure for incorporating new formats into the system

BRTT

June 2005

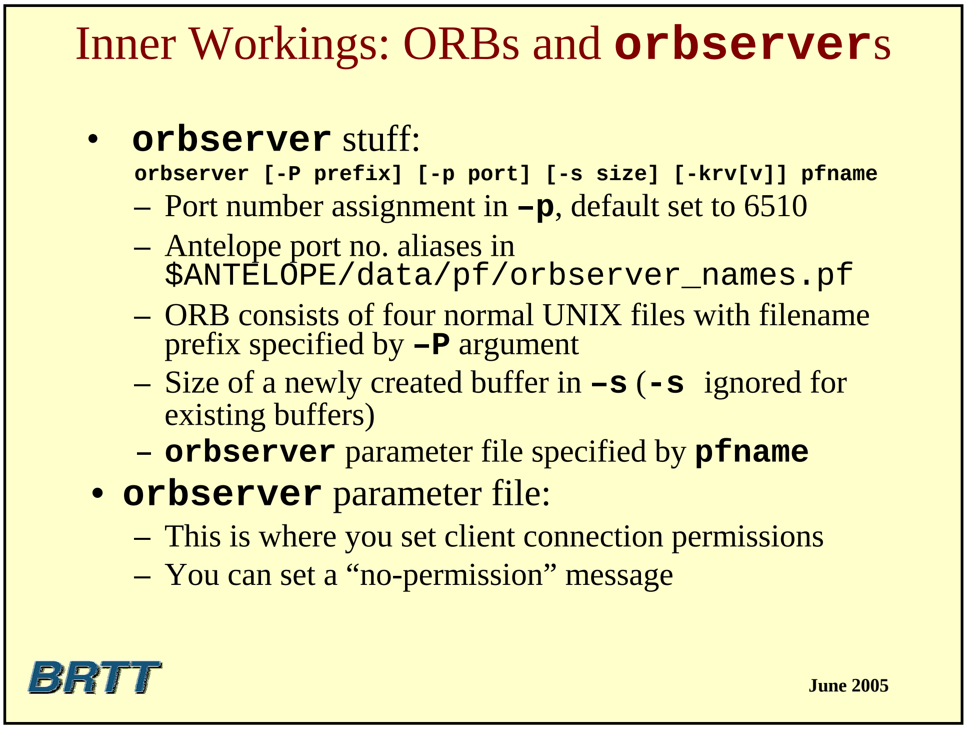


Basic Design: ARTS Capabilities

- Highly modular and interoperable
- Distributed processing
- Minimum processing latency
- Efficient store of data packets with varying size, formats and information contents
- Real-time data merging
- Real-time data distribution
- Real-time data processing
- ORB data tunneling

BRTT

June 2005

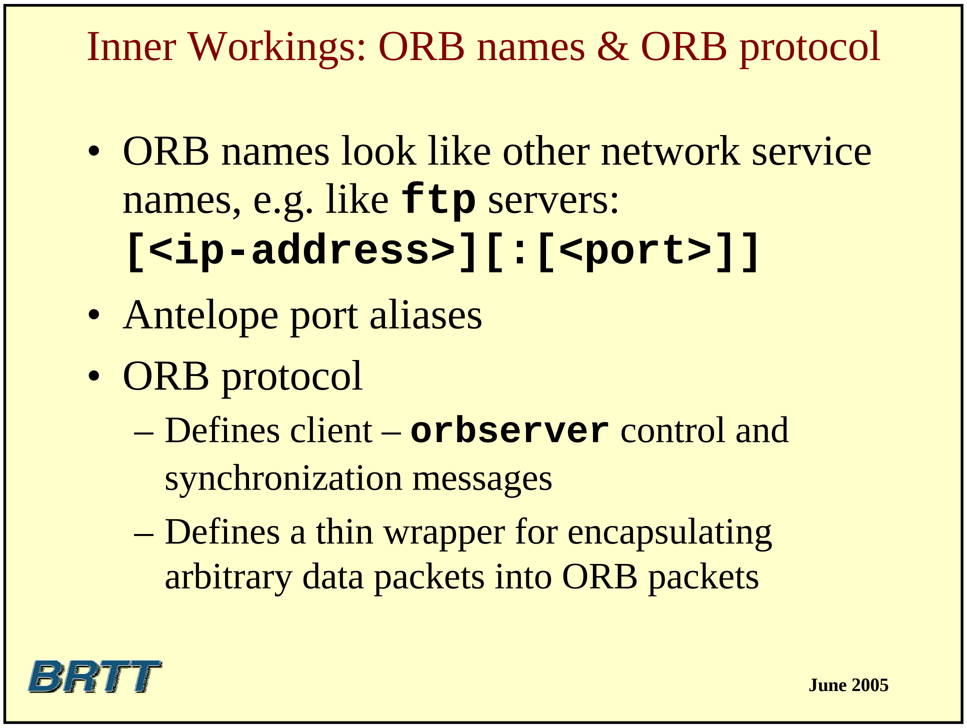


Inner Workings: ORBs and `orbservers`

- **`orbserver`** stuff:
  `orbserver [-P prefix] [-p port] [-s size] [-krv[v]] pfname`
  - Port number assignment in **`-p`**, default set to 6510
  - Antelope port no. aliases in `$ANTELOPE/data/pf/orbserver_names.pf`
  - ORB consists of four normal UNIX files with filename prefix specified by **`-P`** argument
  - Size of a newly created buffer in **`-s`** (**`-s`** ignored for existing buffers)
  - **`orbserver`** parameter file specified by **`pfname`**
- **`orbserver`** parameter file:
  - This is where you set client connection permissions
  - You can set a “no-permission” message

BRTT

June 2005

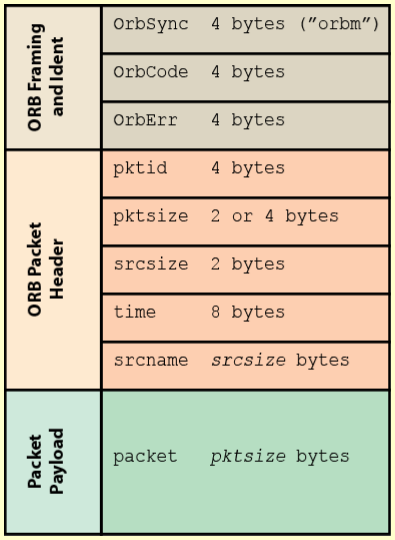


Inner Workings: ORB names & ORB protocol

- ORB names look like other network service names, e.g. like **`ftp`** servers: **`[<ip-address>][:[<port>]]`**
- Antelope port aliases
- ORB protocol
  - Defines client – **`orbserver`** control and synchronization messages
  - Defines a thin wrapper for encapsulating arbitrary data packets into ORB packets

BRTT

June 2005



ORB Framing and Ident	`OrbSync`	`4 bytes ("orbm")`
	`OrbCode`	`4 bytes`
	`OrbErr`	`4 bytes`
ORB Packet Header	`pktid`	`4 bytes`
	`pktsize`	`2 or 4 bytes`
	`srcsize`	`2 bytes`
	`time`	`8 bytes`
	`srcname`	*`srcsize`* `bytes`
Packet Payload	`packet`	*`pktsize`* `bytes`

BRTT

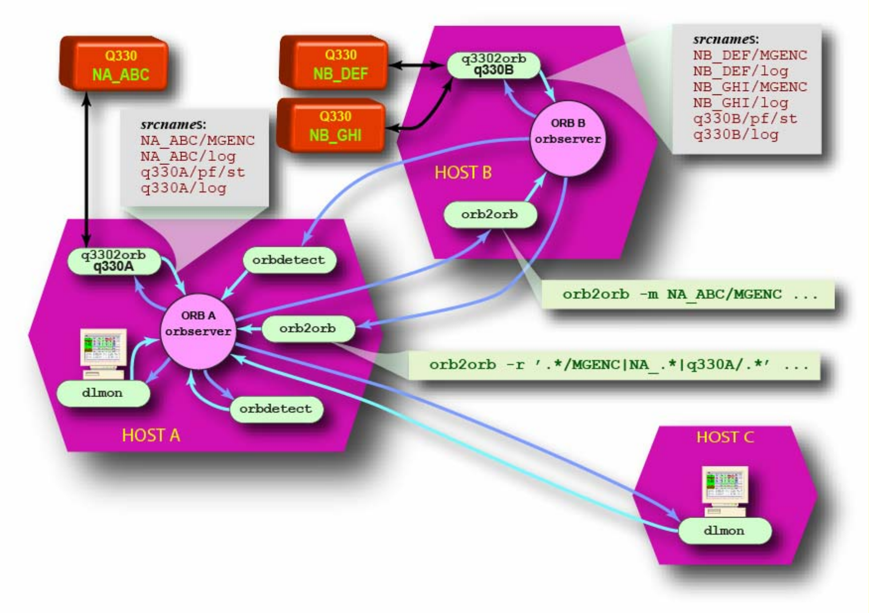
Inner Workings: ORB – client communications

- Most client software uses the ARTS **`liborb`** tool
- The ARTS **`liborb`** tool provides a very comprehensive and robust ORB interface:
  - Handles all client-**`orbserver`** TCP/IP connection setup, ORB protocol messages and data transmission
  - Packages binary packet payloads with ORB header and framing characters
  - Interfaces for packet subscription and setting ORB read pointers
  - Both blocking and non-blocking packet read interfaces
  - Both single packet and “reap” read interfaces
  - Both simplex and acknowledged write interfaces
  - Automatic seamless reconnects
  - Automatic byte ordering at ORB protocol level
  - Note that **`liborb`** makes no assumptions about packet payload contents

BRTT

Inner Workings: Pushes, pulls & state info

- ARTS has been designed to facilitate automatic transfers of real-time continuous data from one ORB to another: **`orb2orb`**
- Where to run ORB packet transfer clients, like **`orb2orb`**? At one ORB, at the other ORB, anywhere else with an ip connection
- Answer - usually, on the same host as the output ORB so that the pull is going across the long-haul link



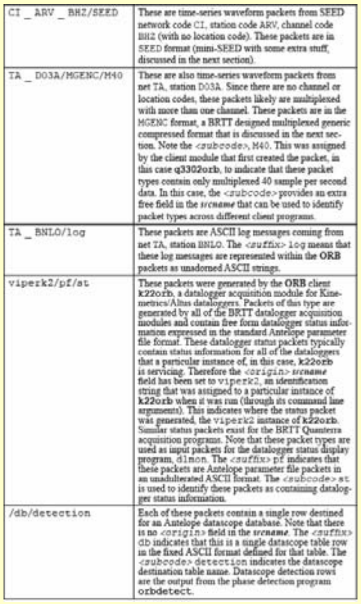
```
orb2orb -m NA_ABC/MGENC ...
orb2orb -r '.*/MGENC|NA_.*|q330A/.*' ...
```

Inner Workings: Pushes, pulls & state info

- Note that most of the inter-host data transfers are done with ORB client pulls
- Note the simplex ORB links
- Independency of ORB-client links; use of threading
- Note the potential feedback data loop between **`orb2orb`** insances on hosts A and B
- Client state processing with Antelope state files

Inner Workings: ORB packet ***srcname*s**

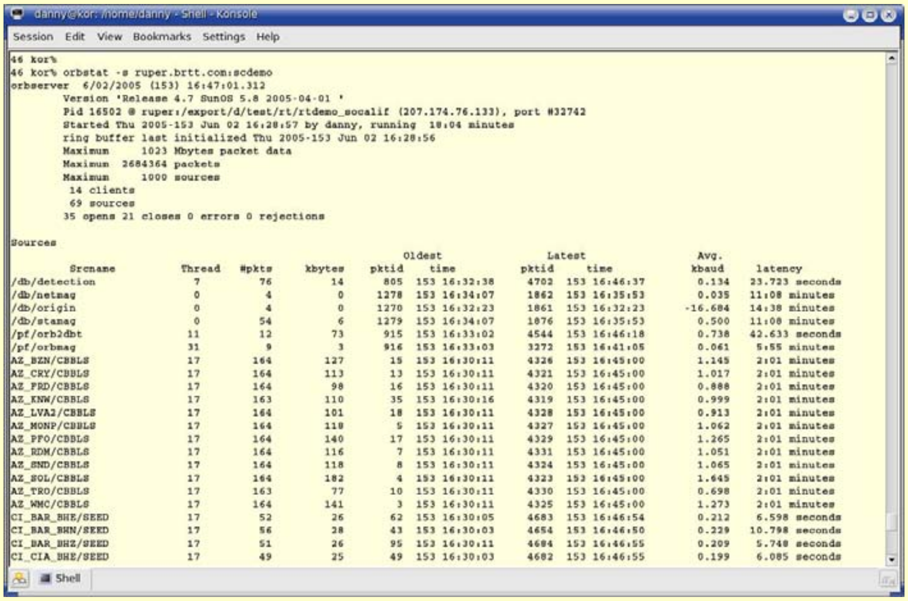
- Generic ASCII tag for identifying packet origin, information content and format
- Assigned by client programs that generate new packets, managed verbatim by the **`orbserver`** and reported with the packet payload when read by a client program
- There is an ARTS naming convention for ***srcname*s**; note this is strictly a client-level thing
  `[<origin>]/<suffix>[/<subcode>]`
- The **`<origin>`** part is, by ARTS convention, the SEED **`<net>[_<sta>[_<chan>[_<loc>]]]`** for time-series waveform data packets



CI _ ARV _ BHZ/SEED	These are time-series waveform packets from SEED network code CI, station code ARV, channel code BHZ (with no location code). These packets are in SEED format (mini-SEED with some extra stuff, discussed in the next section).
TA _ D03A/MGENC/M40	These are also time-series waveform packets from net TA, station D03A. Since there are no channel or location codes, these packets likely are multiplexed with more than one channel. These packets are in the MGENC format, a BRTT designed multiplexed generic compressed format that is discussed in the next section. Note the <subcode>, M40. This was assigned by the client module that first created the packet, in this case **q3302orb**, to indicate that these packet types contain only multiplexed 40 sample per second data. In this case, the <subcode> provides an extra free field in the ***srcname*** that can be used to identify packet types across different client programs.
TA _ BNLO/log	These packets are ASCII log messages coming from net TA, station BNLO. The <suffix> log means that these log messages are represented within the **ORB** packets as unadorned ASCII strings.
viperk2/pf/st	These packets were generated by the **ORB** client **k22orb**, a datalogger acquisition module for Kinemetrics/Altus dataloggers. Packets of this type are generated by all of the BRTT datalogger acquisition modules and contain free form datalogger status information expressed in the standard Antelope parameter file format. These datalogger status packets typically contain status information for all of the dataloggers that a particular instance of, in this case, **k22orb** is servicing. Therefore the <origin> ***srcname*** field has been set to viperk2, an identification string that was assigned to a particular instance of **k22orb** when it was run (through its command line arguments). This indicates where the status packet was generated, the viperk2 instance of **k22orb**. Similar status packets exist for the BRTT Quanterra acquisition programs. Note that these packet types are used as input packets for the datalogger status display program, dlmon. The <suffix> pf indicates that these packets are Antelope parameter file packets in an unadulterated ASCII format. The <subcode> st is used to identify these packets as containing datalogger status information.
/db/detection	Each of these packets contain a single row destined for an Antelope datascope database. Note that there is no <origin> field in the ***srcname***. The <suffix> db indicates that this is a single datascope table row in the fixed ASCII format defined for that table. The <subcode> detection indicates the datascope destination table name. Datascope detection rows are the output from the phase detection program **orbdetect**.

Inner Workings: ORB packet ***srcname***s

- ORB ***srcname***s provide a powerful and generalized mechanism for client packet subscription through the use of UNIX regular expression matching by the **`orbserver`**
- Understanding how ***srcname*** matching and rejection work is crucial to understanding how to configure ARTS
- Lets look back at the previous data flow example to see how this works


danny@kor: /home/danny - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
46 kor%
46 kor% orbstat -s ruper.brtt.com:scdemo
orbserver  6/02/2005 (153) 16:47:01.312
        Version 'Release 4.7 SunOS 5.8 2005-04-01 '
        Pid 16502 @ ruper:/export/d/test/rt/rtdemo_socalif (207.174.76.133), port #32742
        Started Thu 2005-153 Jun 02 16:28:57 by danny, running  18:04 minutes
        ring buffer last initialized Thu 2005-153 Jun 02 16:28:56
        Maximum     1023 Mbytes packet data
        Maximum  2684364 packets
        Maximum     1000 sources
         14 clients
         69 sources
        35 opens 21 closes 0 errors 0 rejections

Sources
                                                            Oldest                Latest                 Avg.
         Srcname          Thread   #pkts     kbytes    pktid    time          pktid     time             kbaud    latency
/db/detection                  7      76         14      805  153 16:32:38     4702  153 16:46:37        0.134  23.723 seconds
/db/netmag                     0       4          0     1278  153 16:34:07     1862  153 16:35:53        0.035  11:08 minutes
/db/origin                     0       4          0     1270  153 16:32:23     1861  153 16:32:23      -16.684  14:38 minutes
/db/stamag                     0      54          6     1279  153 16:34:07     1876  153 16:35:53        0.500  11:08 minutes
/pf/orb2dbt                   11      12         73      915  153 16:33:02     4544  153 16:46:18        0.738  42.633 seconds
/pf/orbmag                    31       9          3      916  153 16:33:03     3272  153 16:41:05        0.061   5:55 minutes
AZ_BZN/CBBLS                  17     164        127       15  153 16:30:11     4326  153 16:45:00        1.145   2:01 minutes
AZ_CRY/CBBLS                  17     164        113       13  153 16:30:11     4321  153 16:45:00        1.017   2:01 minutes
AZ_FRD/CBBLS                  17     164         98       16  153 16:30:11     4320  153 16:45:00        0.888   2:01 minutes
AZ_KNW/CBBLS                  17     163        110       35  153 16:30:16     4319  153 16:45:00        0.999   2:01 minutes
AZ_LVA2/CBBLS                 17     164        101       18  153 16:30:11     4328  153 16:45:00        0.913   2:01 minutes
AZ_MONP/CBBLS                 17     164        118        5  153 16:30:11     4327  153 16:45:00        1.062   2:01 minutes
AZ_PFO/CBBLS                  17     164        140       17  153 16:30:11     4329  153 16:45:00        1.265   2:01 minutes
AZ_RDM/CBBLS                  17     164        116        7  153 16:30:11     4331  153 16:45:00        1.051   2:01 minutes
AZ_SND/CBBLS                  17     164        118        8  153 16:30:11     4324  153 16:45:00        1.065   2:01 minutes
AZ_SOL/CBBLS                  17     164        182        4  153 16:30:11     4323  153 16:45:00        1.645   2:01 minutes
AZ_TRO/CBBLS                  17     163         77       10  153 16:30:11     4330  153 16:45:00        0.698   2:01 minutes
AZ_WMC/CBBLS                  17     164        141        3  153 16:30:11     4325  153 16:45:00        1.273   2:01 minutes
CI_BAR_BHE/SEED               17      52         26       62  153 16:30:05     4683  153 16:46:54        0.212   6.598 seconds
CI_BAR_BHN/SEED               17      56         28       43  153 16:30:03     4654  153 16:46:50        0.229  10.798 seconds
CI_BAR_BHZ/SEED               17      51         26       95  153 16:30:11     4684  153 16:46:55        0.209   5.740 seconds
CI_CIA_BHE/SEED               17      49         25       49  153 16:30:03     4682  153 16:46:55        0.199   6.085 seconds
```

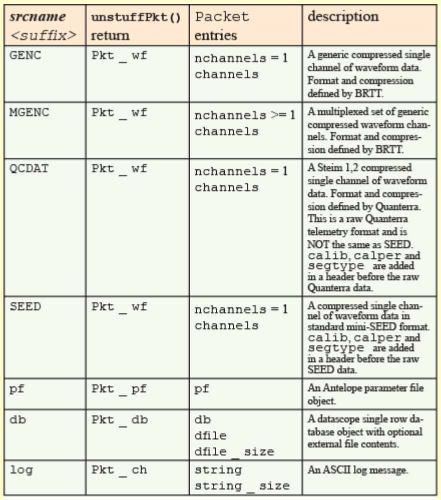
Inner Workings: Decoding and encoding

- ARTS is format neutral at its most basic working level (i.e. `orbserver` – `orb2orb` transport)
- How to design client programs to use packets of varying formats without having to write different versions for each format?
- The ARTS solution to this is the **libPkt** middleware tool
- Client programs that use **libPkt** can be written so that they will work with different packet formats without any application-level format dependent coding

Inner Workings: Decoding and encoding

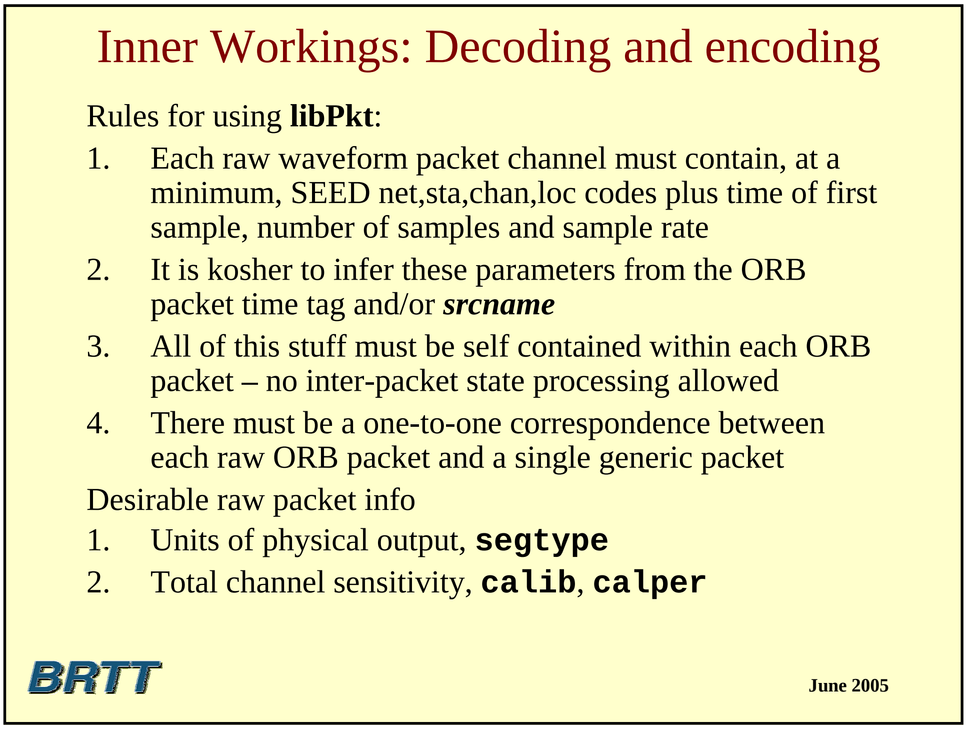
How do we do this?

1. Define “generic” application-level packet structures with the packet information in the most convenient form for the application programmer
2. Develop and use “universal” stuff and unstuff transcoders that convert any binary packet representations in and out of the generic structures
3. The ***srcname*** **`<suffix>`** field is used as a key by **libPkt**’s **`unstuffPkt()`** routine.



srcname
<suffix>
	unstuffPkt()
	return
		Packet
		entries
			description

GENC	Pkt_wf	nchannels = 1 channels	A generic compressed single channel of waveform data. Format and compression defined by BRTT.
MGENC	Pkt_wf	nchannels >= 1 channels	A multiplexed set of generic compressed waveform channels. Format and compression defined by BRTT.
QCDAT	Pkt_wf	nchannels = 1 channels	A Steim 1,2 compressed single channel of waveform data. Format and compression defined by Quanterra. This is a raw Quanterra telemetry format and is NOT the same as SEED. calib, calper and segtype are added in a header before the raw Quanterra data.
SEED	Pkt_wf	nchannels = 1 channels	A compressed single channel of waveform data in standard mini-SEED format. calib, calper and segtype are added in a header before the raw SEED data.
pf	Pkt_pf	pf	An Antelope parameter file object.
db	Pkt_db	db dfile dfile_size	A datascope single row database object with optional external file contents.
log	Pkt_ch	string string_size	An ASCII log message.



Inner Workings: Decoding and encoding

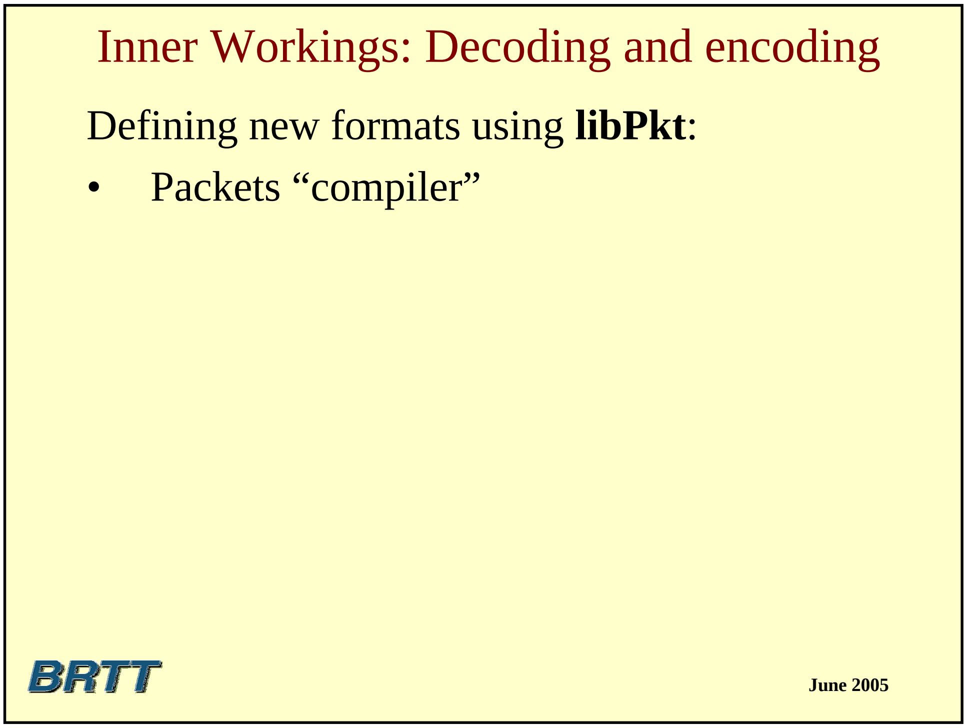
Rules for using **libPkt**:

1. Each raw waveform packet channel must contain, at a minimum, SEED net,sta,chan,loc codes plus time of first sample, number of samples and sample rate
2. It is kosher to infer these parameters from the ORB packet time tag and/or ***srcname***
3. All of this stuff must be self contained within each ORB packet – no inter-packet state processing allowed
4. There must be a one-to-one correspondence between each raw ORB packet and a single generic packet

Desirable raw packet info

1. Units of physical output, **`segtype`**
2. Total channel sensitivity, **`calib`**, **`calper`**

June 2005



Inner Workings: Decoding and encoding

Defining new formats using **libPkt**:

- Packets “compiler”

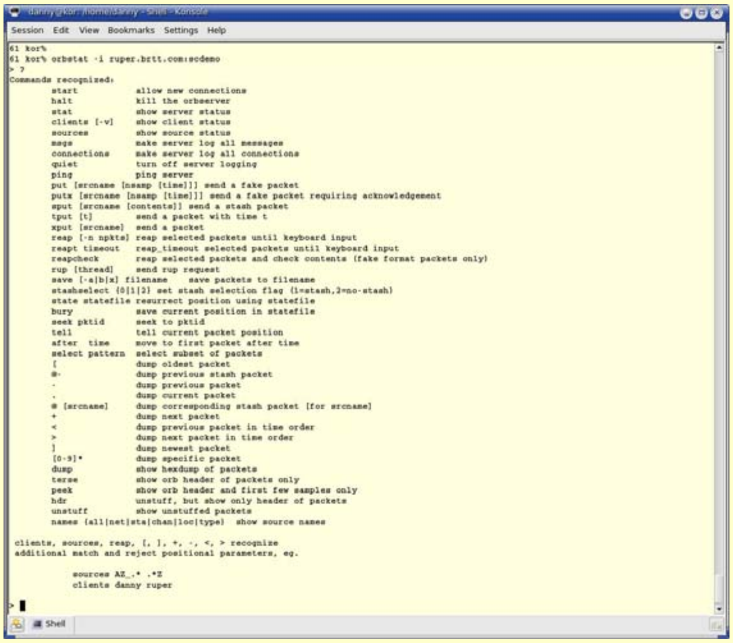
June 2005

Inner Workings: Using **`orbstat`**

- Please spend some time with this very useful tool
- Can be used for 1) debugging, 2) training 3) general exploration
- Strictly text base which means it should run in just about any environment

BRTT

June 2005

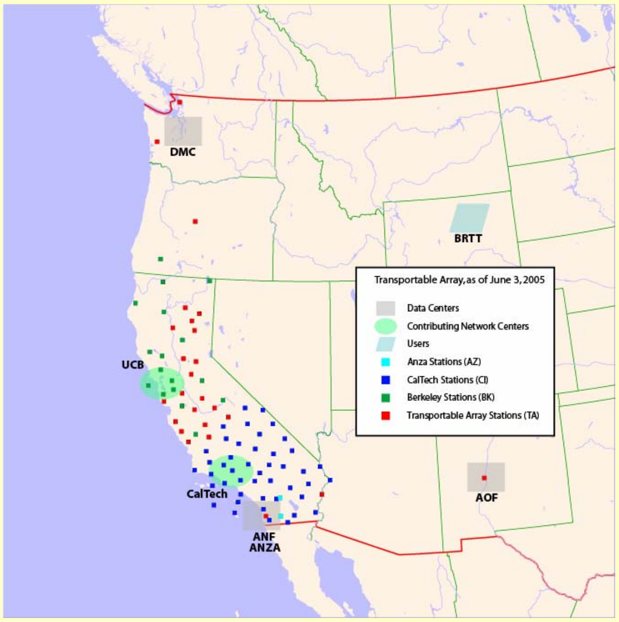


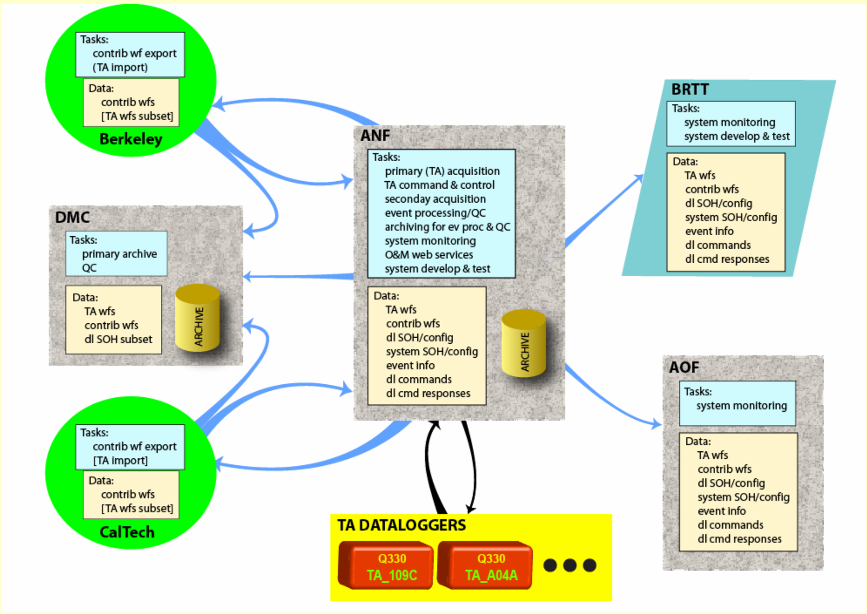
```
61 kor%
61 kor% orbstat -i ruper.brtt.com:scdemo
> ?
Commands recognized:
        start           allow new connections
        halt            kill the orbserver
        stat            show server status
        clients [-v]    show client status
        sources         show source status
        msgs            make server log all messages
        connections     make server log all connections
        quiet           turn off server logging
        ping            ping server
        put [srcname [nsamp [time]]] send a fake packet
        putx [srcname [nsamp [time]]] send a fake packet requiring acknowledgement
        sput [srcname [contents]] send a stash packet
        tput [t]        send a packet with time t
        xput [srcname]  send a packet
        reap [-n npkts] reap selected packets until keyboard input
        reapt timeout   reap_timeout selected packets until keyboard input
        reapcheck       reap selected packets and check contents (fake format packets only)
        rup [thread]    send rup request
        save [-a|b|x] filename    save packets to filename
        stashselect {0|1|2} set stash selection flag (1=stash,2=no-stash)
        state statefile resurrect position using statefile
        bury            save current position in statefile
        seek pktid      seek to pktid
        tell            tell current packet position
        after  time     move to first packet after time
        select pattern  select subset of packets
        [               dump oldest packet
        @-              dump previous stash packet
        -               dump previous packet
        .               dump current packet
        @ [srcname]     dump corresponding stash packet [for srcname]
        +               dump next packet
        <               dump previous packet in time order
        >               dump next packet in time order
        ]               dump newest packet
        [0-9]*          dump specific packet
        dump            show hexdump of packets
        terse           show orb header of packets only
        peek            show orb header and first few samples only
        hdr             unstuff, but show only header of packets
        unstuff         show unstuffed packets
        names {all|net|sta|chan|loc|type}  show source names

 clients, sources, reap, [, ], +, -, <, > recognize
 additional match and reject positional parameters, eg.

           sources AZ_.* .*Z
           clients danny ruper

> ▮
```

BRTT

June 2005

ARTS example: The TA facility





Berkeley

Tasks:
contrib wf export
(TA import)

Data:
contrib wfs
[TA wfs subset]

DMC

Tasks:
primary archive
QC

Data:
TA wfs
contrib wfs
dl SOH subset

ANF

Tasks:
primary (TA) acquisition
TA command & control
seconday acquisition
event processing/QC
archiving for ev proc & QC
system monitoring
O&M web services
system develop & test

Data:
TA wfs
contrib wfs
dl SOH/config
system SOH/config
event info
dl commands
dl cmd responses

CalTech

Tasks:
contrib wf export
[TA import]

Data:
contrib wfs
[TA wfs subset]

BRTT

Tasks:
system monitoring
system develop & test

Data:
TA wfs
contrib wfs
dl SOH/config
system SOH/config
event info
dl commands
dl cmd responses

AOF

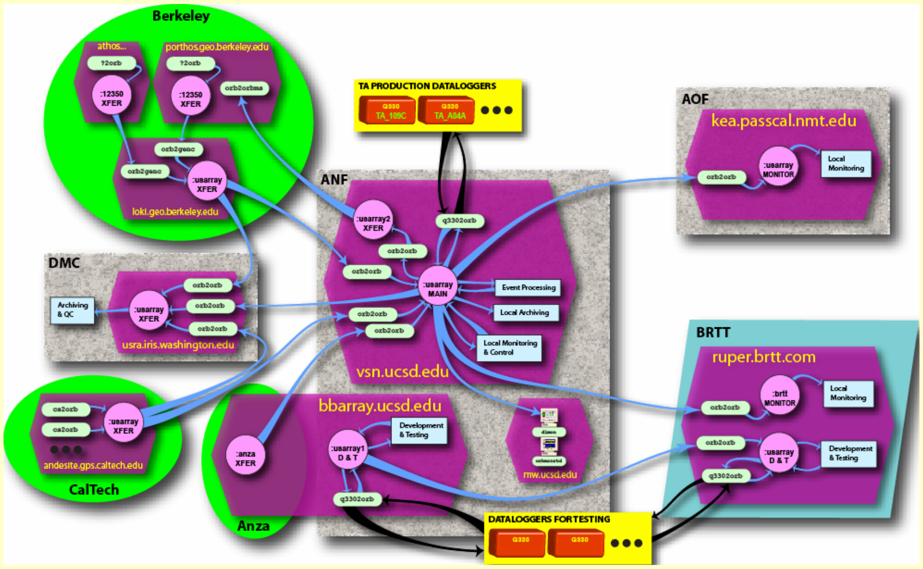
Tasks:
system monitoring

Data:
TA wfs
contrib wfs
dl SOH/config
system SOH/config
event info
dl commands
dl cmd responses

TA DATALOGGERS

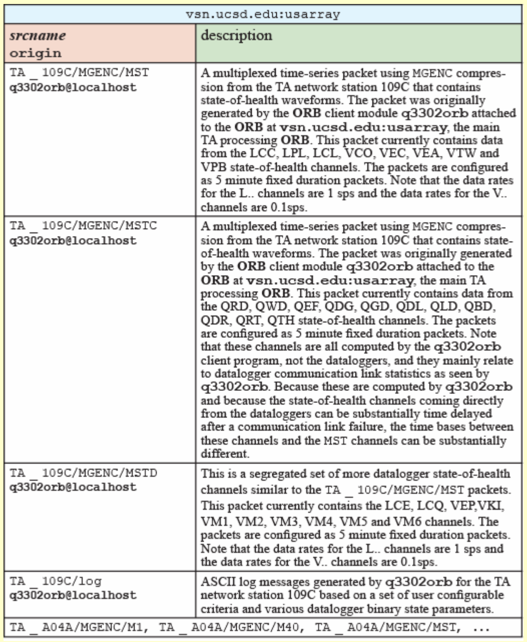
BRTT

June 2005



BRTT

June 2005



vsn.ucsd.edu:usarray	
srcname **origin**	description
TA _ 109C/MGENC/MST **q3302orb@localhost**	A multiplexed time-series packet using MGENC compression from the TA network station 109C that contains state-of-health waveforms. The packet was originally generated by the **ORB** client module q3302orb attached to the **ORB** at vsn.ucsd.edu:usarray, the main TA processing **ORB**. This packet currently contains data from the LCC, LPL, LCL, VCO, VEC, VEA, VTW and VPB state-of-health channels. The packets are configured as 5 minute fixed duration packets. Note that the data rates for the L.. channels are 1 sps and the data rates for the V.. channels are 0.1sps.
TA _ 109C/MGENC/MSTC **q3302orb@localhost**	A multiplexed time-series packet using MGENC compression from the TA network station 109C that contains state-of-health waveforms. The packet was originally generated by the **ORB** client module q3302orb attached to the **ORB** at vsn.ucsd.edu:usarray, the main TA processing **ORB**. This packet currently contains data from the QRD, QWD, QEF, QDG, QGD, QDL, QLD, QBD, QDR, QRT, QTH state-of-health channels. The packets are configured as 5 minute fixed duration packets. Note that these channels are all computed by the q3302orb client program, not the dataloggers, and they mainly relate to datalogger communication link statistics as seen by q3302orb. Because these are computed by q3302orb and because the state-of-health channels coming directly from the dataloggers can be substantially time delayed after a communication link failure, the time bases between these channels and the MST channels can be substantially different.
TA _ 109C/MGENC/MSTD **q3302orb@localhost**	This is a segregated set of more datalogger state-of-health channels similar to the TA _ 109C/MGENC/MST packets. This packet currently contains the LCE, LCQ, VEP,VKI, VM1, VM2, VM3, VM4, VM5 and VM6 channels. The packets are configured as 5 minute fixed duration packets. Note that the data rates for the L.. channels are 1 sps and the data rates for the V.. channels are 0.1sps.
TA _ 109C/log **q3302orb@localhost**	ASCII log messages generated by q3302orb for the TA network station 109C based on a set of user configurable criteria and various datalogger binary state parameters.
TA _ A04A/MGENC/M1, TA _ A04A/MGENC/M40, TA _ A04A/MGENC/MST, ...	