Debugging EPICS Channel Access

Web Links On Channel Access

Here are several weblinks to CA docs.

Channel Access Spec and Docs

http://www.aps.anl.gov/epics/docs/ca.php

EPICS Base Docs, CA Ref manual

http://www.aps.anl.gov/epics/base/R3-14/12.php

Channel Access In depth

http://www.aps.anl.gov/bcda/epicsgettingstarted/specialtopics/channeldepth.html

caSnooper and casw

http://www.aps.anl.gov/epics/extensions/caSnooper/index.php

How channel Access Works-

From

http://epics.cosylab.com/cosyjava/JCA-Common/Documentation/CAproto.html#toc

1.2. Basic Concepts [Send e-mail to document owner].

1.2.1. Process Variables [Send e-mail to document owner].

A Process Variable (PV) is representation of a single value within an EPICS host.

1.2.2. Channels [Send e-mail to document owner].

A channel is created whenever a PV is connected using CA.

From implementation point of view, a channel is a connection, established over virtual circuit, between server and client through which a single PV is accessed. Both the client and the server will provide a way of uniquely identifying channels. These identifiers are explained in section Message Identifiers (3.2.).

1.2.3. Monitors [Send e-mail to document owner].

A monitor is created on a channel as a means of registering for asynchronous change notifications. CA protocol defines the subscription mechanism through which clients register for notifications. These changes will then be provided through monitor object or callback, depending on implementation and environment. Monitors may be filtered to receive only a subset of events, such as value or alarm changes. Several different monitors may be created for each channel.

1.2.4. Channel Access Client Library [Send e-mail to document owner].

A client library implements channel access protocol and exposes it through a programmer-friendly API. How the protocol and its mechanisms are implemented in the library depends on the programming language and no specific requirements are made with respect to this.

1.2.5. Repeater [Send e-mail to document owner].

Repeater is a standalone process that allows several clients on one machine to share a predefined UDP port for recieving broadcast messages. Broadcast messages are sent without prior knowledge of which port the CA clients and hosts are listening at, so predefined port numbers are used. Since most network stack implementations do not allow multiple clients to listen on the same port, repeater is installed to listen on a single port, used for broadcast notifications. Repeater will fan-out unmodified incoming datagram messages to all the clients on the same host, that have previously registered with the repeater.

Typically, a client library will attempt to register with the repeater during startup. If repeater cannot be found or is not available, the library will attempt to spawn a new repeater process. Repeater and client communicate using a minimal set of messages over UDP.

1.2.6. Server Beacons [Send e-mail to document owner].

Server beacons are simple protocol messages that allow servers to broadcast their availability. These messages are sent out periodically to announce their presence. Additionally, these beacons are used to restore virtual circuits that have lost connections.

1.2.7. Virtual Circuit [Send e-mail to document owner].

Channel Access protocol is designed to minimize resources used on both client and server. Virtual circuits minimize number of TCP connections used between clients and servers. Each client will have exactly one active and open TCP connection to an individual server, regardless of how many channels it accesses over it. This helps to ensure that servers do not get overwhelmed by too many connections.

The life-cycle of a virtual circuit is defined in section Virtual Circuit Operation (10.).

1.2.8. Message Buffering [Send e-mail to document owner].

When sending packets over network, channel access allows huge savings to be made by grouping individual messages in a single TCP packet. This is performed by maintaining per virtual circuit buffers that collect all outgoing messages. These buffers are flushed upon application's explicit request. This mechanism is independent of the TCP/IP stack, which maintains its own send queue and defines a maximum frame size. Messages sent over virtual circuit may be larger than this and will not always be aligned on frame boundary. Implementation should also be aware that messages received may not yet be available entirely or will be split over several TCP frames.

1.2.9. Version compatibility [Send e-mail to document owner].

Certain aspects of Channel Access protocol have changed between releases. In this document, Channel Access versions are identified using CA_VXYY, where X represents single-digit major version number and YY represents a single- or double-digit minor version number. Stating that a feature is available in CA_VXYY implies that any client supporting version XYY must support the feature. Implementation must be backward compatible with all versions up to and including its declared supported minor version number.

Example: CA_V43, denotes version 4.3 (major version 4, minor version 3).

Currently, all protocol definitions are assumed to have major version 4. Minor version ranges from 1 through 11.

1.2.10. Exceptions [Send e-mail to document owner].

This document uses the concept of exceptions to refer to the mechanism of reporting errors that occur during command request execution. An exception defines reporting of error condition on the server. This can occur either due to client problem or due to some unexpected condition on the server (such as running out of resources). Exceptions are reported either within the command's response or using a specialized message. Actual form and method (response or asynhronously received message) depends on circumstances where the exception occurs. Individual commands and messages determine which method is used.

1.3. Overall Operation [Send e-mail to document owner].

This section is intended as a quick overview of channel access functionality and behaviour. For more information, please refer to Channel Access Reference Manual[1].

The goal of channel access (CA) is to provide remote access to records and fields managed by IOC, including search and discovery of hosts and minimal flow contol. Protocol itself is designed to provide minimal overhead and maximize network throughput for transfer of large number of small data packets. Additionaly, implementation overhead of the protocol can be kept very small, to allow operation with limited resources.

All commands and events in CA are encapsulated in predefined messages, which can be sent in one of three forms:

As a beacon, which requires no confirmation. Used for host discovery and keep-alive notification.

As a request/response pair. Most commands use this method.

As a subscriber notification, where the client registers with the host and receives updates. Event notification uses this method to report value changes.

Communication between server and client is performed by sending command messages over UDP and TCP. Client will use UDP to search for hosts and PVs, server will use them to notify its startup and shutdown. Once client requests a specific PV (by specifying its name), UDP message will be broadcast to either a subnet or a list of predefined addresses, and the server which hosts requested PV will respond.

Data exchange between client and server is performed over TCP. After locating the PV, the client will establish a TCP connection to the server. If more that one PV is found to be on the same server, client will use existing TCP connection. Reusable TCP connection between client and server is called Virtual circuit.

Once a virtual circuit is established or already available, channels can be created to PVs.

Let's assume the following setup of PVs and hosts:

PV "A" is located on host IOC1.

PV "B" is located on host IOC2.

PV "C" does not exist on either host.

PV "D" is located both hosts, IOC1 and IOC2.

A typical scenario would be similar to this:

User application starts and initializes channel access client library. During initialization, client library might spawn a new repeater process or register with an existing one. If successful, the library is ready to start using the channel access.

Application will start searching for PVs named "A", "B", "C" and "D". Client will send UDP search packets with "A", "B", "C" and "D" PV names, and wait for an application-specified amount of time. To which network hosts (broadcast or specific IP) these packets are actually sent depends on configuration of the client.

All hosts that receive this message will check their database and if they know about any of these PV names, they will respond using UDP search response message. Thus, IOC1 will respond with "A" and "D", whereas IOC2 will respond with "B" and "D".

For each search packet sent, the client will wait for a predefined ammount of time or until response is received.

In this case (assuming no packet loss in the network), the client will establish a virtual circuit (TCP connection) to IOC1 and IOC2. PV "C" will be assumed to be unavailable at this point and will be ignored. Since "D" is located on both hosts, client will be unable to distinguish between them. First response received will be considered to be proper answer and any additional responses will be considered erroneous and reported to user.

Once virtual circuits are established, client can access values of "A", "B" and "D". From now on, all messages for either of the PVs will be sent via a corresponding virtual circuit.

If virtual circuit looses TCP connection or the host disconnects, client will notify application appropriately. Client will also listen for server beacon messages (sent whenever IOC host starts), to be able to restore any lost connections. Beacons are also used to detect when a host stops responding.

Once the application is done using the channel access library, it will gracefully close any open connections.

Some useful linux debugging tools

Important EPICS Env Vars

EPICS_CA_ADDR_LIST

EPICS_CA_AUTO_ADDR_LIST

Things Linux sysadmins should know:

http://www.cyberciti.biz/tips/top-linux-monitoring-tools.html

Channel Access- Monitor CA Beacons.

casw –i 2

To put, get, monitor pvs

caput <pvname>

caget <pvname>

camonitor <pvname>

To get pv info

cainfo <pvname>

Probe

MEDM tool, CA client. Also a JProbe.

http://www.aps.anl.gov/epics/extensions/probe/index.php

Control System Studio

On dgs1 run

/home/dgs/tmadden/CSS_EPICS_3.1.2/css

CSS has a probe.

csSnooper

Tool for snooping channel access. Needs to be built on the machines. Not done yet.

Weblink above.

On SS,

From http://www.cyberciti.biz/tips/linux-investigate-sockets-network-connections.html

List Sockets

ss

Examples for ss

ss –t –a   to list all tcp sockets

ss –u –a to list all udp sockets

ss –l display all open net ports

On netstat

http://www.cyberciti.biz/tips/netstat-command-tutorial-examples.html

You can ping the IOCs

ping ioc2 or IP addr.

Tools on IOC Console

On VxWorks console:

To disp processes

i

netHelp

To show mem usage of TCP

mbufShow

To show IP address

```
version
```

To show boot params (Aps only)

```
bootShow
```

To measure system process usage. Run these three in order.

```
spyClkStart
spyClkStop
spyReport
```

To list info on CA servers

```
casr
casr 2
```

To list PVs

```
dbl
```

To Find a PV

```
dbgrep(“*partofPVname*”)
```

Get/Put PVs

```
dbgf “pvname”

dbpf “pvname”,”value”
```

Print info about a PV

dbpr “pvname”,6

To debug PVs and watch forward links, making sure pvs connect.

dbpf “pvname.TPRO”,”1”

On Virtual Circuits

From

http://www.aps.anl.gov/bcda/epicsgettingstarted/specialtopics/ChannelAccessInDepth1up.pdf

Beacons

• A Beacon is a UDP broadcast packet sent by a Server

• When it is healthy, each Server broadcasts a UDP beacon at

regular intervals (like a heartbeat)

- EPICS_CA_BEACON_PERIOD, 15 s by default

• When it is coming up, each Server broadcasts a startup

sequence of UDP beacons

- Starts with a small interval (25 ms, 75 ms for VxWorks)

- Interval doubles each time

- Until it gets larger than 15 s, then it stays at 15 s

- Takes about 10 beacons and 40 s to get to steady state

• Clients monitor the beacons

- Determine connection status, whether to reissue searches

Virtual Circuit Disconnect

• 3.13 and early 3.14

- Hang-up message or no response from server for 30 sec.

- If not a hang-up, then client sends “Are you there” query

- If no response for 5 sec, TCP connection is closed

- MEDM screens go white

- Clients reissue search requests

• 3.14.5 and later

- Hang-up message from server

- TCP connection is closed

- MEDM screens go white

- Clients reissue search requests

Virtual Circuit Unresponsive

• 3.14.5 and later

- No response from server for 30 sec.

- Client then sends “Are you there” query

- If no response for 5 sec, TCP connection is not closed

- For several hours, at least

- MEDM screens go white

- Clients do not reissue search requests

- Helps with network storms

- Clients that do not call ca_poll frequently get a virtual circuit

disconnect even though the server may be OK

- Clients written for 3.13 but using 3.14 may have a problem

- May be changed in future versions

Beacon Anomaly

• A Beacon Anomaly is any change from the normal beacon

interval (15 s)

• No beacons:

- After 30 sec the client sends message over TCP connection

- If no beacons and no reply, connection is down

- That is when MEDM screens go white

• Abnormal interval:

- Short: IOC has come up

- Long: IOC was disconnected

• May cause clients to reissue outstanding search requests

• Network problems can look like beacon anomalies

Connection Process

• A client (e.g. MEDM) wanting a PV sends a UDP search request

- Sent to EPICS_CA_ADDR_LIST

- (Or its default -- broadcast to all interfaces on the host machine)

- Sent on EPICS_CA_SERVER_PORT [5064]

- Do you have this PV?

• Each Server that gets a packet does an exist test

- Do I have this PV?

• Server with the PV sends a directed UDP reply to the Client

- I have this PV.

• A TCP connection is established between the Server and the

Client (or an existing one is used)

- One per Client-Server pair, no matter how many PVs

- Referred to as a Circuit

- Let’s talk.

Search Request

• A client makes a search request when it wants to find out what

server has the PV

- Happens when a PV is first created in the client

- On a beacon anomaly (unresolved PVs only)

- When another PV is created (unresolved PVs only)

• A search request consists of a sequence of UDP packets

- Starts with a small interval (30 ms), that doubles each time

- Until it gets larger than 5 s, then it stays at 5 s

- Stops after 100 packets or when it gets a response

- Used to never try again until it sees a beacon anomaly or

creates a new PV

- As of 3.14.7 retries at a slow rate

- Total time is about 8 minutes to do all 100

- The sequence may be different owing to fine tuning

• Usually connects on the first packet or the first few

Exist Test

• Every time a Server receives a search request packet, its

pvExistTest routine is called

• The Server has to check if it has the PV

- Returns ExistsHere or DoesNotExistHere

• Normally a search request sequence ends after a few packets

- Because one Server soon returns ExistsHere

• For PVs that do not exist

- There are 100 tests per search request sequence for that PV

- This happens every time a Client initiates a search request

sequence

- Each time the Client searches for a new PV

- At each beacon anomaly, perceived or real