PVAccess Gateway Design

Michael Davidsaver

November 18, 2015

1 Terms

GW Gateway

CLI An arbitrary PVAccess client (end user or another gateway)

GWS Gateway server side (CLI communicates with this)

GWC Gateway client side (SRV communicates with this)

SRV An arbitrary PVAccess server (may be another gateway)

2 Goals/Features

Identified features and design goals.

De-duplication of sockets and data to reduce overall resource use.

Policies for access. This include access control (whether an operation is permitted) and administrative limits (bound per-client resource usage).

3 Theory of Operation

3.1 Name resolution/socket setup

Use of periodically re-sent UDP messages for name search permits an asynchronous mode of operation for name resolution and socket (circuit) setup.

```
struct ChannelCacheEntry {
        string name;  // key
        int priority; // key
        unsigned refcount; // implicily via. shared_ptr
        bool searched;
        shared_ptr<Channel> chanGWC;
};
```

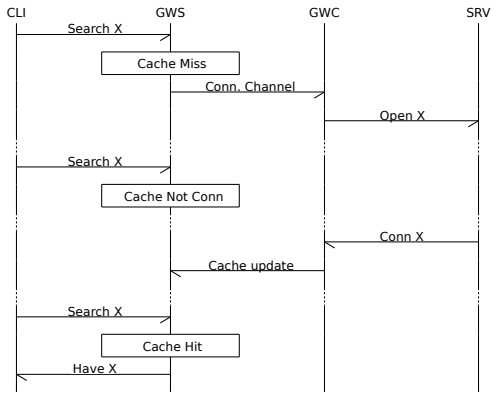

Figure 1: CLI name search outcomes

A cache of GWC Channels index by name and priority (the parameters of the createChannel message) is maintained. Entries in this ChannelCache have an associated GWC Channel, and a garbage cleanup flag.

When GWS receives a new CLI search request message a lookup is made to the ChannelCache which has three possible outcomes (see figure 1).

Miss There is no entry in the cache. A new entry is created. The new GWC Channel begins the search/connect process. No reply to CLI is made.

Not Conn An entry exists, but the associated GWC Channel is not yet connected.No reply to CLI is made.

Hit There is an entry with a connected GWC Channel. A positive search reply is sent to CLI.

Handling of TCP channelCreate messages by GWS is similar except that a negative reply is sent for Miss and Not Conn outcomes.

A reference count is maintained by each ChannelCache entry for the GWS channels using each GWC channel. However, an entry should not be immediately removed when the ref. count drops to zero. Instead it should be kept for some time to allow it to be found by for future CLI search requests.

Each ChannelCache entry should also have a boolean flag which is set on creation, and re-set whenever it is looked up. A periodic cleanup task should run which removes all entries with this flag cleared, and zero ref. count. Each time it run, the cleanup task should clear the flag of any entry not removed.

This should ensure that entries not used by a GWS channel are eventually removed (GWC Channel closed) when no client is searching for them.

Ownership of an active GWC Channel is shared by several active GWS channels, and a ChannelCache entry as shown in figure 2. Red arrows represent strong ownership and blue weak ownership.

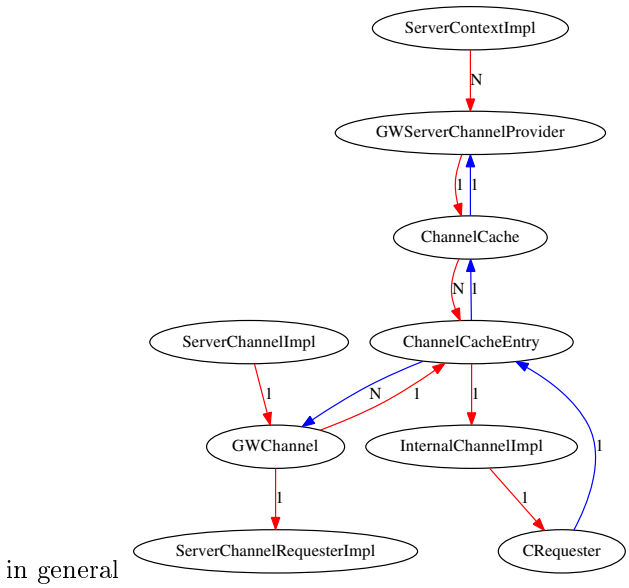

Figure 2: GWC Channel ownership

Notification of loss of a GWC Channel should result in the disconnection of any associated GWS Channels, and the immediate removal of the associated ChannelCache entry.

3.2 Get/Put/RPC/... operations

For operations other than Monitor, the timing of the client request can effect the results. No caching or de-duplication can be done without special knowedge about the intended behavior of CLI and SRV. Therefore, by default these operations a pass through the GW without de-duplication.

3.3 Monitor operations

Each ChannelCache entry will also include a MonitorCache. This cache is indexed by the pvrequest given with the corresponding monitorCreate operation. As a pvrequest may contain arbitrary data, two pvrequests may not be compared for anything other than exact equality without special knowledge. Therefore, by default MonitorCache hits are only generated when the CLI provides a pvrequest which exactly matches the MonitorCache entry.

Each MonitorCache entry should also include the most recent (last) value received by the GWC so that this may be returned immediately for new GWS subscriptions.

A list of GWS consumers (interested in event data) will also be maintained. Each new event data value is passed into the MonitorRequester of each con-

sumer.

```
struct MonitorCacheEntry {
        PVField request; // key
        weak_ptr<ChannelCacheEntry> chan;
        unsigned refcount; // implicily via. shared_ptr
        shared_ptr<Monitor> mon;
        PVField lastval;
        list<MonitorConsumer*> consumers;
};
struct MonitorConsumer {
        shared_ptr<MonitorRequester> queueGWS;
        shared_ptr<MonitorCacheEntry> entry;
        bool GC;
};
```

As with ChannelCache, a MonitorCache entry should not be removed immediately when its ref. count reaches zero. Instead unreferenced entries should be periodically closed in the same fashion (perhaps as part of) the periodic ChannelCache cleanup.

4 Channel Transmit Queueing

A major potential pitfall of de-duplication (and connection sharing in general) is the handling and prioritization (or lack thereof) of traffic on different channels/operations. For example, monitoring a single high data rate PV can cause other operations to experience larger latency. One way to mitigate this, in part, is to introduce some "fairness" to the circuit/Transport transmit message queue.

Instead of a simple FIFO fed by all Channels, give each channel a FIFO. The task which dequeues would do by taking from each FIFO in turn in a round robin. This should prevent the overall latency through the queue from being dominated by one fast PV.

However, this is only a partial solution as large PVs will still introduce latency in proportion to the individual data size.

```
queue = [A, A, B, A, B, C, C, A, A, B]
sort_fair() # [[A, A, A, A, A], [B, B, B], [C, C]]
queue == [A, B, C, A, B, C, A, B, A, A]
```

5 Loop avoidance

Another potential pitfall inherent in using UDP broadcasts for name resolution is the possibility of loops should GWS receive search requests from GWC. This can be avoided provided that GWS is aware of the set of endpoints that GWC uses to send requests, and ignores an requests from them.

6 Policies

Along with de-duplication, enforcement of administrative policies is the major function of a GW. Areas of policy include: access control, resource limits, and queuing behavior.

6.1 Access Control

In order to share GWC channels, the GW will make all access control decisions. Authentication information provided by CLI is **never** forwarded to SRV. Instead, the GWs own authentication information is sent to SRV.

Access control needs to be configurable on a per-PV, per-operation basis. This typically takes the form of an Access Control List, where rules are traversed in some order. Each rule makes some decision to Allow, Deny, or Pass.

To allow the flexibility, a rule may subscribe to several PVs and use the values obtained, in addition to client provided information and static configuration, to make decision.

6.2 Administrative Limits

A number of administrative limits should also be provided to limit the resource usage of potentially misbehaving clients including:

- Max # of clients
- Max # of channels per client
- Max # of concurrent operations, for each operation type
- Max # monitor queue depth

Such limits could be made hard (fail further requests) or soft (log and allow).

6.3 Queuing

At a minimum, the default and max queue sizes should be settable by policy. Additionally, a choice of algorithms could be provided to decide which entries to drop when the queue overflows.