Introduction

- pvAccess is a high-performance network communication protocol for signal monitoring and scientific data services interconnect

- Designed to support the structured data types – pvData

- Combines CPU and wire data size considerations to optimize overall control network throughput and minimum latency
First full Java and C++ implementations available from Feb 2012 (1.0-BETA release)

After this Java implementation refactored
- “codec” based – abstract class that takes care of all the encoding/decoding
- transport implementations (e.g. TCP, UDP) simply use the codec and contain no protocol specific knowledge
- extensive JUnit tests implemented to verify codec implementation

C++ implementation refactoring in progress
Status (cntd.)

  - Lots of discussion, expert reviews…
  - Comments, reviews welcome and appreciated

- Last change: unsigned integer support!

- Currently no implementation (Java, C++) fully implement the specification
“Everybody” wants to see some performance figures…

A simple test was performed to give a feeling what pvAccess performance capabilities are

What was tested:

- C++ implementations tested
- Only protocols tested (aka their “portable server” implementations)

pvAccess C++ (1.0.1-BETA)
EPICS 3.14.12 Channel Access
Performance (cntd.)

- Clean isolated environment:
  - 2 MacBook Pro’s (~2GHz Intel Core Duo-s, 4GB memory)
  - 1GBit Ethernet point-to-point connection (no switch between)

- Fair test – avoid test-cases where pvAccess can optimize data transfer (i.e. transfer only changed fields in a structure)

- The test:
  - GET request on a double array value channel (aka DBR_DOUBLE)
  - variable number of channels
  - variable value array size
One channel, double array element count on x axis
1000 channels, double array element count on x axis

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvAccess</td>
<td>106000</td>
<td>111000</td>
<td>86600</td>
<td>11800</td>
<td>1183</td>
<td>118</td>
</tr>
<tr>
<td>CA</td>
<td>333000</td>
<td>228000</td>
<td>41570</td>
<td>4510</td>
<td>443</td>
<td>44</td>
</tr>
</tbody>
</table>
pvAccess handles large arrays better
- its design allows to send GByte arrays using only 16kB buffers, while CA requires entire array to fit the buffer

CA handles small messages better
- pvAccess does not yet implement user-controlled flushing, aka ca_flush_io()
- API design in progress
- new codec-based implementation is also expected to provide better performance figures
THANK YOU!