# cMsg - A Publish/Subscribe Interprocess Communication Package and Framework

Elliott Wolin, Carl Timmer, D Abbott, V Gyurjyan, G Heyes, E Jastrzembski

Thomas Jefferson National Accelerator Facility, Newport News, VA 23606

## Introduction

**cMsg is many things: an API under which one can unify interprocess communication (IPC); a framework within which one can deploy multiple underlying IPC packages; a proxy server than can communicate with underlying IPC packages on behalf of a remote client; and a full-featured message-based publish/subscribe IPC system.**

**We call the underlying IPC systems "domains", and the domain identifier (Universal Domain Locator, or UDL) is specified at runtime. Processes can have as many domain connections as needed, and can simultaneously communicate with a number of underlying IPC systems through a uniform API. Unification of IPC under a single API makes migration from one underlying package to another extremely simple, nothing more than changing a single runtime parameter.**

**The built-in full-featured publish/subscribe facility is fast, robust, and rivals existing public domain and commercial packages in functionality.**

## An API

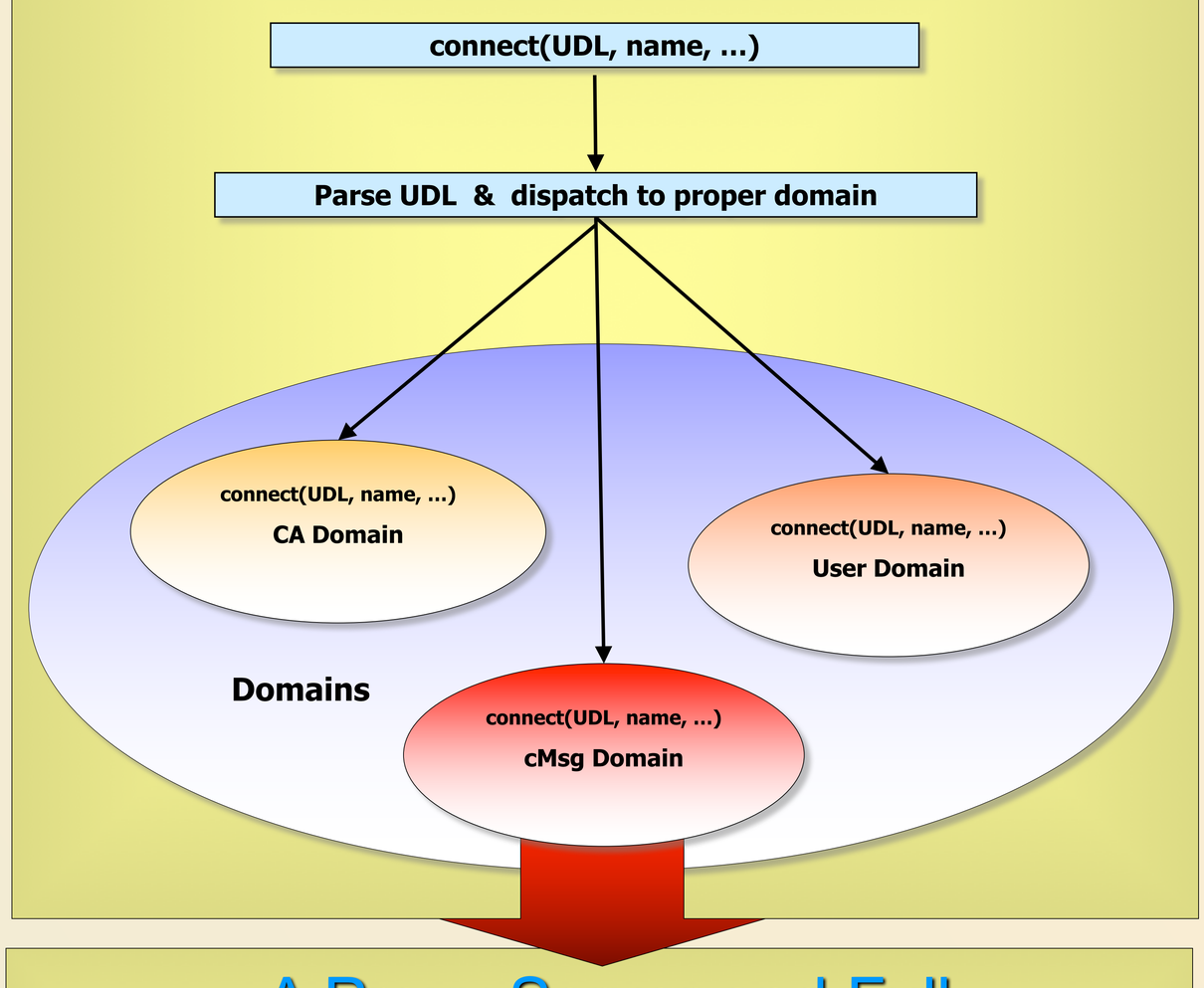
**cMsg aims to unify communication packages under a single, message-passing API (Java, C, C++). An underlying package (or domain) may not implement all functions. This table contains a simplified list of the major client functions. The messages contain several user-settable fields*.**

| Function | Description |
|---|---|
| **connect**(UDL, myName) | Connect to a cMsg system specified by the UDL for client myName |
| **disconnect**() | Disconnect from the cMsg system |
| **send**(msg) | Send a message asynchronously |
| **flush**(timeout) | Flush messages to send from client |
| **syncSend**(msg, timeout) | Send a message and wait for server response |
| **sendAndGet**(msg, timeout) | Send a message and wait for receiving client to send a response |
| **subscribe**(subject, type, callback) | Subscribe to messages of a given subject & type, registering a callback for incoming messages |
| **unsubscribe**() | Remove a subscription |
| **subscribeAndGet**(subject, type, timeout) | Subscribe to a subject & type and wait for one response |
| **start**() | Start receiving messages |
| **stop**() | Stop receiving messages |
| **monitor**(command) | Synchronous call to request monitoring information |

***User-settable fields include strings, an integer, a time, and a byte array. The next release of cMsg will allow any number of strings, primitive types, messages, binary and arrays of each.**

## A Framework

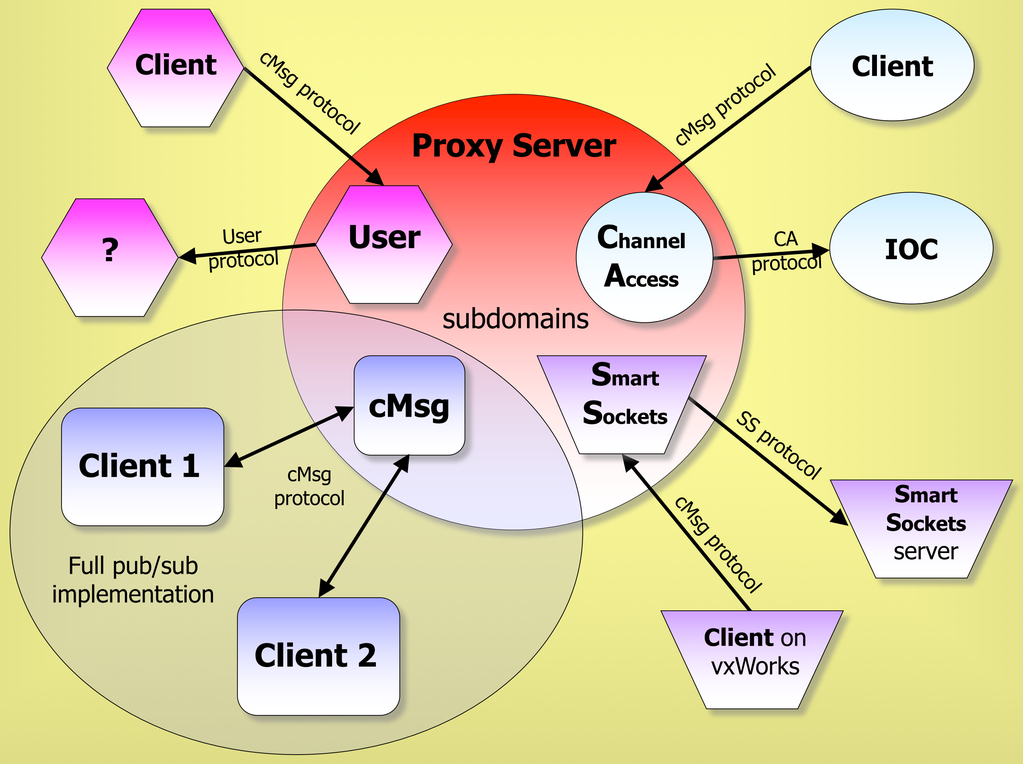
**The cMsg client API is implemented as a thin dispatching layer to the underlying domains.**





## A Proxy Server and Full Publish/Subscribe Implementation

**This is a view of the client and server sides of the cMsg domain, which implements pluggable subdomains, including a full publish/subscribe messaging subdomain. The proxy server is written in Java.**




## Advantages

**Unification of all IPC under a single API allows for simple integration of new IPC packages and migration away from legacy packages. A process can simultaneously connect to multiple underlying IPC packages, so all IPC can be performed via a single API. Switching from one underlying package to another involves changing a single runtime parameter.**

**The proxy server allows processes to communicate via an IPC package that is not supported on the system the process is running on. Thus a process on VxWorks can communicate with systems for which no VxWorks library exists. Incorporation of new underlying packages (sub-domains) in the proxy server is quite simple (n.b. Java only).**

**The built-in pub/sub message-based IPC domain or facility supports asynchronous and synchronous communications, and messages can contain an arbitrary number of data fields (primitives, arrays of primitives, messages). Subscriptions and message routing is base on a message subject and type (arbitrary strings, wildcards supported).**

**Multiple IPC servers can inter-communicate, allowing for client load-balancing, and server hot-failover protects against of loss of one or more servers.**

## Conclusions

**The cMsg system is a simple, powerful, and flexible open-source framework within which one can deploy multiple underlying IPC systems. It includes a built-in, full-featured, asynchronous publish/subscribe component, support for a number of commonly used IPC systems, as well as a number of useful utilities. It supports C, C++ and Java clients, and runs on Unix and VxWorks.**

**cMsg performance approaches network bandwidth limits, and generally is only limited by the networking ability of the server machine. Indeed it exceeds our requirements by two orders of magnitude.**

**The use of Java in cMsg greatly reduced our development time compared to C, and Java performance has proven to be excellent, generally exceeding C performance (although this may change with further tuning of the C components). Our results clearly demonstrate that Java is a serious contender for almost any DAQ or online requirement.**

## Downloads

Download and give cMsg a try! You can get your free copy today at ftp://ftp.jlab.org/pub/coda/cMsg:

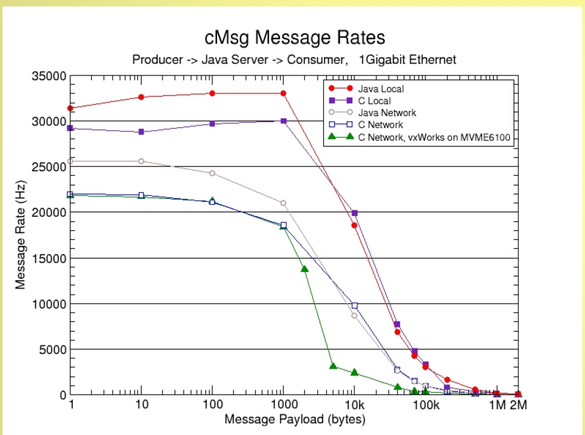
Carl Timmer, (757) 269-5130, timmer@jlab.org, or

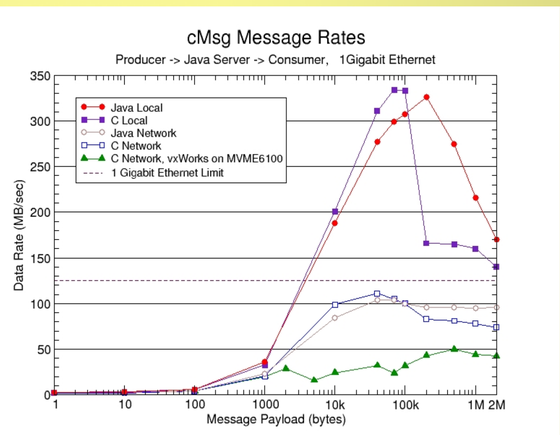
Elliott Wolin, (757) 269-7365, wolin@jlab.org

## Performance

**For small messages, rates over the network between one producer and one consumer are over 20kHz (100x above our design spec).**

**For large messages (>10K bytes), data rates over the network between one producer and one consumer reach 80% of the network bandwidth.**




cMsg server running on a 2.4 GHZ quad-Opteron, RHEL3, Gbit Ethernet, Cisco Catalyst 4000 switch