

GIPC -- A Real-Time Group IPC Communications Facility within CORDS

Franco Travostino

**The Open Group
Research Institute**

<http://www.osf.org/RI/PubProjPgs/CORDS.htm>

From our Fault Tolerance “Plan of Record”

OS Advisory Board Meeting, January ‘96

“Initially, an OS Single System Image is not essential.

The applications participate in Fault Management.

Real-time predictability under failure is important.

Build on the RI’s past base of distributed, real-time results.

Develop application demos hand-in-hand with systems work.

A Scalable, FT server for the Web.”

Real-Time requirements in Fault Tolerance

Throughput? No, thanks!

Emerging distributed, dependable real-time applications for combat systems, factory floors, hospital beds demand:

- **predictable timings (in the presence of failures too)**
- **intuitive and simple group delivery semantics (“all-or-none”)**
- **application’s full control over QoS contracts, joins, partitions, etc.**
- **privileged communication channels**
- **per channel resource budgets to limit cascading of failures**
- **robustness to complex failures and classes of threats from network**
- **adaptivity to different modus operandi and external stimuli**
- **configurability for different roles and failure models**

State of the art

Commercial and academic solutions for software fault tolerance are traditionally optimized for throughput

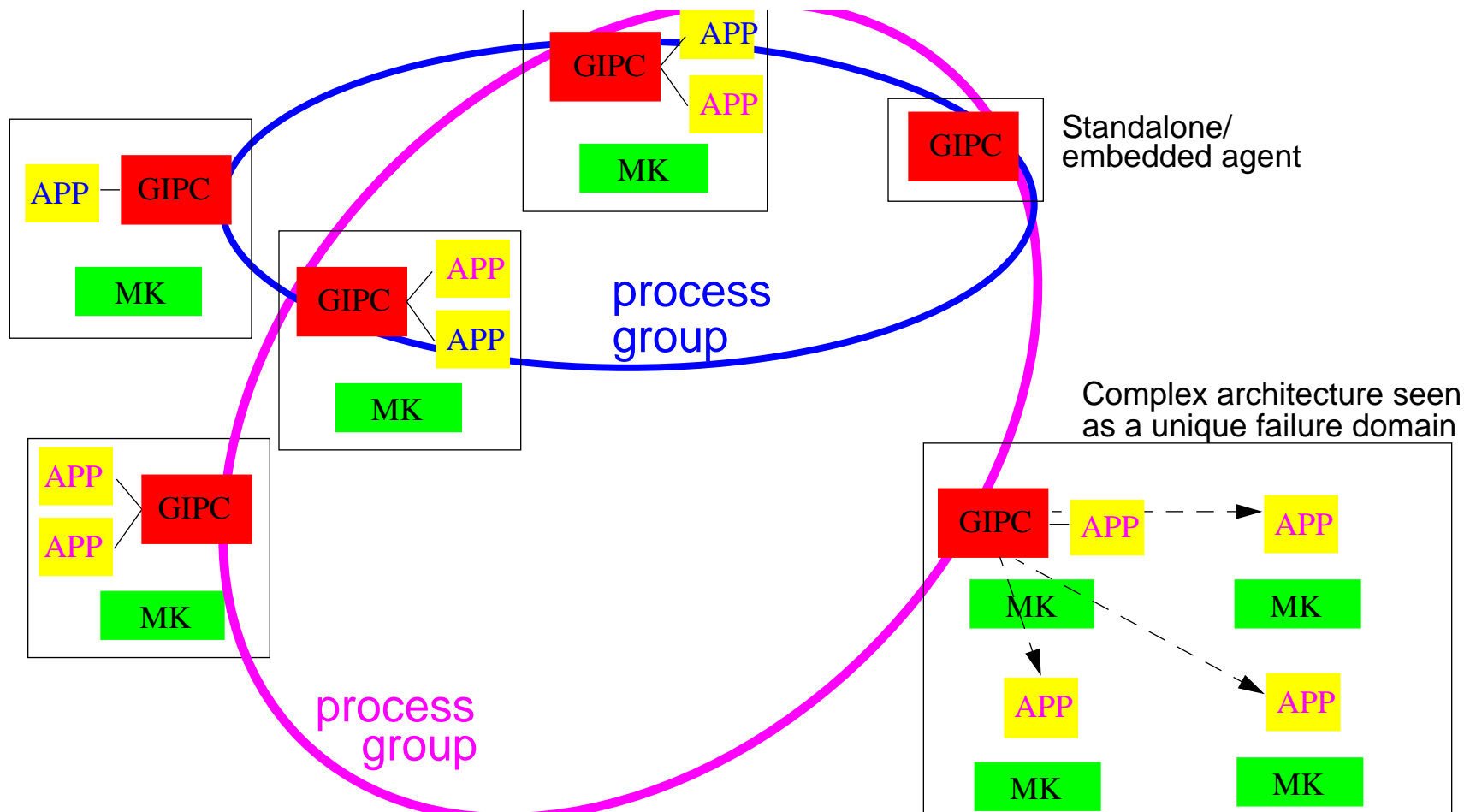
Failures are unlikely events with very troublesome handling

Isis/Horus/Ensemble has not published any real-time claim

Transis and Totem cover some of the real-time spectrum

GIPC

We have built a Group IPC (GIPC) service for membership and reliable multicast within real-time process groups

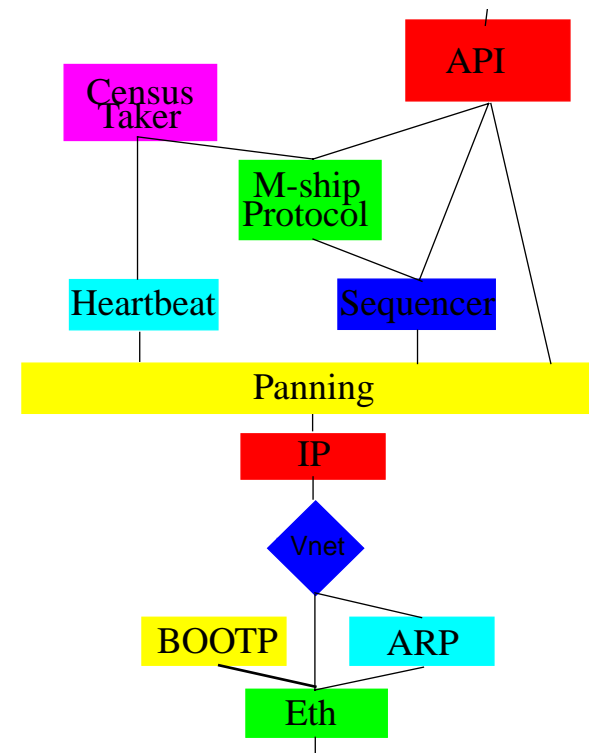


GIPC (cont.)

GIPC comes as a kit of client/server stubs, CORDS protocols, and sample demo clients

In its simplest incarnation - as delivered in MK7.3 - GIPC is an atomic broadcast engine, layered upon IP multicast

Model	Virtual synchrony
Link	Timed asynchronous
Failures	Node crashes and Ethernet's type of failures
Membership	Live but not accurate
Ordering	FIFO Atomic Broadcast
Delivery	Safe
Joins	Amnesia only
Addressing	Group identifier
Groups	Multiple logical groups



GIPC (cont.)

In spite of its simplicity, GIPC has already demonstrated behaviors that belong to synchronous, dedicated hardware

- **Example: < 100 msecs between a node crash and a new stable membership, on a private Ethernet**

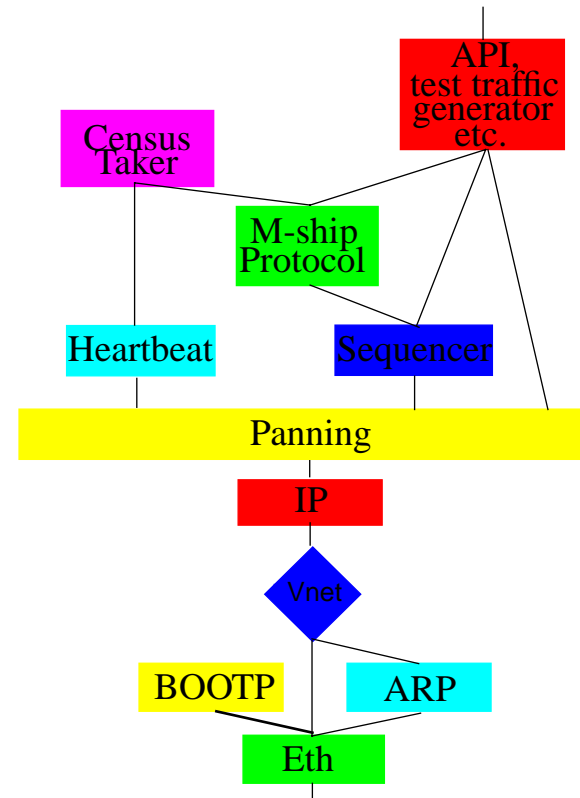
GIPC represents our evolutionary path through the real-time requirements

GIPC builds upon our investment on a highly modular, real-time communication framework, CORDS

GIPC protocol components

Protocols for process groups and group communication:

- API *(interface)*.
- Membership *(who's out there?)*.
- Sequencer *(order messages)*.
- Heartbeat *(send "I'm alive")*.
- Census-taker *(get "I'm alive")*.
- Panning *(toss stale messages)*.



CORDS properties

CORDS is an evolution of the x-kernel work from University of Arizona

Standard and custom protocols in a library with 40+ protocols

Protocols can operate over Ethernet, FDDI, and Myrinet

Protocols are OS independent, hence extractable technologies

In principle, CORDS uses a thread per message

There is the notion of Path, an end-to-end communication flow

A Path is given a budget of CPU, memory, and bandwidth

With threads and Paths, control extends from top to bottom of the communication stack

GIPC implementation choices

GIPC uses positive acks to maintain source/receivers in lockstep and thus minimize/bound recovery actions

Problem: pos. acks imply knowledge of subscribers

- **Solution: as we track membership, this info comes at no extra cost**

Problem: with pos. acks we may insist in re-sending stale data

- **Solution: control messages (i.e., membership views) can preempt and suppress/postpone messages that are unable to become stable**

Problem: pos. acks scale poorly

- **Solution: whenever necessary, limit ack implosion using n Designated Ack-ers ($n = 0$ being negative acks)**

GIPC implementation choices (cont.)

GIPC uses a master/slave approach to enforce total message ordering and group consensus

This has important advantages and one disadvantage:

- + High concurrency, low latency increase group's responsiveness**
- + Centralized entity can use relative merit, enforce group policies**
- + Most failures are easy to deal with; master failures are trickier**
- Message implosion at the master with large group population**

A rotating token shows opposite pros/cons:

- + Message volume and flow control independent from group size**
- Token smashes concurrency; high latency with large groups**
- Lack of an observer makes group maintenance and repair tougher; it's hard to deploy policies for the group**

There are interesting opportunities in hybrid solutions

GIPC real-time properties

GIPC distinguishes between endpoint liveness contracts (node local) and physical node liveness contracts

Performance failures are treated as crash failures; skeptics provide a configurable filter to control status changes

Some messages (e.g., membership) can preempt others

Message stability is eagerly evaluated; bound recovery time

GIPC's protocols can use CORDS' Paths to shield communication channels from system and network traffic

A GIPC server is a real-time executive on its own and its threads can be programmed to fit into any priority schema

GIPC has a wealth of knobs to control tunable parameters

Ongoing GIPC work

Ongoing and future work:

- **Multiple Logical Groups (with updated demo clients)**
- **Extensive use of paths**
- **Characterization (i.e., real-time rather than real-fast)**
- **Exploit synergisms among protocols and improve piggybacking**
- **Compound messages**
- ***n* Designated Ackers**
- **Make clients supply policies for joins and partitions**
- **Out-of-group senders and group composition**
- **Estimated physical clock ordering**

Demos continue to tick our progress

References

- [1]S. Dowlati, “GIPC Interface”, OSF TR, to be published.
- [2]E. F. Menze, F. Travostino, “The x-kernel-based Communication Subsystem for Mach/OSF: changes to the x-kernel 3.2 API”, OSF TR, available in DRAFT form (Version 2.0), OSF, Cambridge, MA, September, 1996.
- [3]F. Travostino, “Elementary Groupware for Fault Management”, OSF TR, OSF, Cambridge, MA, October, 1995.
- [4]F.Travostino, F. Reynolds, “An O-O Communication Subsystem for Real-Time Distributed Mach”, IEEE Proceedings of the 1st Workshop on Object-Oriented Real-Time Dependable Systems, Dana Point, Ca, October 1994.
- [5]F. Travostino, E. Menze and F.Reynolds, “Paths: Programming with System Resources in Support of Real-Time Distributed Applications”, IEEE Proceedings of the 2nd Workshop on Object-Oriented Real-Time Dependable Systems, Laguna Beach, CA, February 1996.