Interprocess Communication for Robotics Applications

Edward M. Roderick*

***Abstract*— Modern robotic systems are developed as a collection of independent processes. For these processes to function together, an interprocess communication system (IPC) must be implemented. Many prebuilt systems are available and this paper presents four popular options (Sockets, ZeroMQ, ACH, and ROS) as candiates for robotics. An analysis is presented for each based on communication latency and design metrics as applied to robotics.**

I. INTRODUCTION

As modern robotic systems grow in complexity, it becomes increasingly beneficial to implement a multi-process control system. This provides a modular system architecture. A modular architecture protects the overall system from individual components failing. Critical system processes can be allocated additional resources to ensure continued operation during a partial system failure. By predefining the inputs and outputs for each subsystem, multiple teams of designers can use differing languages, allowing for optimization of each task. Additionally, the overall system can be distrubted across a variety of hardware (full computers, single board computers, microcontrollers).

Many IPC options are available to designers and each must be evaluated on a per system basis to determine the best candidate. This paper will focus on evaluating several mainstream options and how they apply to design metrics of robotic systems. As robots operate in real time, the latency and data integrity of each communication step is of critical importance. For complex systems (humanoids, robots operating in proximity with humans, etc) with multiple processes running concurrently, communication delay can results in actuators responding to obsolete data. This paper serves to provide a guide to evaluating the capabilites of Sockets, ACH, ZeroMQ and ROS.

II. BACKGROUND

A. ACH (Shared Memory)

Shared memory offers the fastest read/write speeds for communication between processes within a single computer. As part of the POSIX[1] IPC library, shared memory is available to virtually every robotic system. Real time robotics require that the IPC system be able to prioiritize the most recent data over data received sequentially (non head of line blocking). Basic shared memory provides this by overwriting variables but is susceptable to synchronization issues and all previous data is now lost[1].

ACH builds on the basic shared memory and provides a message bus (channel) between mutliple writers and multiple readers. An ACH channel consists of a data buffer and an index buffer that are stored in a shared memory file. The channel provides synchronized access via a mutex and a condition variable. This allows for priority inhereitence and prevents starvation[1]. The ACH library is open source and formally verified [1] making it suitable for widespread use.

ACH channels provide high levels of flexibilty in communication and programming. They can be configured to read the most recent data or in order it was received. A read command can block until data is received or it can poll the channel as needed. Figure 1 below is an example of an ACH based control system for the Hubo robot.

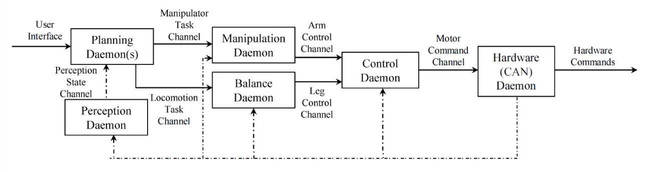

Fig. 1. HUBO ACH Communication Flow Chart

ACH provides network based communications by incorporating TCP and UDP messages. Each system running the ACH network daemon (ACHD) can write directly to shared memory resources connected to the network[2]. This allows for CPU intensive processes to be offloaded from the robot to a local high performance computer.

B. Sockets: TCP and UDP

Sockets provides interprocess communication over a network via the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). The implementations evaluated for this paper are the client/server architecture, which provides one to one communication between processes. Both TCP and UDP operate over a network interface and are bound to a specific port number and IP address. Wireless networks and wireless connected devices have become ubiquitous in modern society. With the growing number of single board computers with full ethernet/wireless network interfaces, it is possible to quickly and cheaply set up a robotic system that can interact with sensors, high power computers, and external controllers. Sockets is a well seasoned IPC and can be found as part of the POSIX library, allowing for many different operating systems and programming languages to communicate with no extra software and little programming effort. With modern newtork routers offering gigabit connec-

*Author is affilited with George Mason University, Fairfax VA, 22030, USA. `eroderi2@gmu.edu`.

[1]POSIX: Portable operating system interface. A set of standards for maintaining compatability between opearting systems.

tions, sockets can provide a method of fast communication between processes distrubted across a network.

Streams suffers from head of line blocking and is thus not a desirable method for robotics. Datagrams utilize a data buffer that when full, loses any new data. This is particularly problematic as the robot would routinely lose important data.

1) TCP: TCP offers a faily robust communication link between a single client and a single server process. When created, the two processes will perform some handshaking to complete the connection. During each transmission from a client, there is a response from the server indicating if the message was received. This allows the system to identify if a message was not sent. For many applications this would be an advantage, but in robotics data can be streaming in from sensors at a high rate and the missed message would be obsolete by the time the system realizes the missed transmission[3]. Any retransmission would add additional delay on top of the delay incured by waiting for a response from the server.

A TCP server receives messages as they arrive in the buffer and must process each message in the order it is sent. This first in first out system (FIFO) allows for all messages to be saved, but again suffers from the disadvantage of older messages beingn prioritized over new. If message are being received at a higher frequency than a process is running, the process must consume all old data received while alseep before accessing the data that is in real time.

2) UDP: UDP offers one quick directional communication from a client process to a server process. Each process is bound to a target IP address and port number. There is no handshaking or direct link between the processes. This allows mutltiple clients to send data to a single server. As there is no response indicating a successful transmission of a message, UDP inherently has less latency than TCP. The disadvantage to this method is that if all data is of critical importance, message can be missed and the system has no way of identifying that data was lost. Depending on the application, the requirement of high speed data transmission can overcome the tradeoff for message transmission assurance.

C. ZeroMQ

ZeroMQ (MQ) is a messaging system that extends on the foundations of sockets. It provides an additional abstraction layer over TCP by sending messages to a system topology instead of specific IP addresses. MQ prevents sending to specific IP addresses by design. The available messaging patterns is open ended, popular systems include publish/subscribe and request/reply. Only one messaging pattern is allowed for a topology and they cannot be interconnected. This is essential to providing guarantees that data will arrive at its intended location. MQ separates its stack into end to end and hop by hop layers. Unlike TCP/IP, each MQ end to end protocol has its own hop by hop protocol[4]. This allows each specific messaging pattern to have their own routing functionality. This allows bidirectional messages to be sent to specific nodes in the topology. Each intermediary node can determine if downstream sections of the network are unreachable and can signal the original client to resend now or wait until connectivity has been reestablished.

With multicore processors being commonplace in todays market and the growing availability of multicore processors in commercially available embedded systems there is a need for efficiently written multithreaded code. Current methods of dealing with the issues that arise from multithreading (synchronization, starvation, priority inversion) are difficult to learn and do not scale well as the number of available cores increases. By creating code with too many locks, threads can block one another and the system can perform inconsistently depending on what system the program is running on. Writing lock free code can become susceptible to processors reordering instructions and the code again performing inconsistently. These multithreading approaches are API based and incorrect usage can often result in multiple threads blocking each other and the program essentially running as a single threaded process. ZeroMQ provides a solution to these issues by abstracting the more difficult lock free multithreading code behind its native message passing techniques and achieves simple multithreading by sending messages between threads. Being based on BSD Sockets, the code implementation is simple and intuitive for existing programmers, as well as being available for a multitude of programming languages. ZeroMQ is infinitely scalable as the number of available cores increases without any rewriting of existing code.

D. ROS

ROS (Robot Operating System) is a flexible framwork for developing robotic systems. Despite its namesake ROS is not an operating system, and it offers more features than just an IPC. ROS utilizes TCP as its core transport mechanism for communication between its processes, but the architiecture differs than the TCP presented earlier as part of sockets. In addition to the IPC, ROS offers a wide library of prebuilt tools and a vast user base from which to draw on modular solutions that can plug into any ROS system. This reduces the development time and amount of low level code that needs too be written for each piece of hardware.

insert image here

With the growing availability of cheap ARM processor systems, versions of ROS are being developed to provide a lower cost design than the traditional full-fledged Linux pc. ROS subsystems can be distributed over multiple ARM systems, allowing for more design flexibility, lower power consumption, and improved scalability. ROS allows for higher level systems to be separated from real time subsystems and other system critical functions. This allows for high priority systems to be simplified to reduce risk of failure. ROS was designed to support higher level robot functions instead of the control of individual motors/sensors/etc. Much of the integration of subsystems is left to the design as ROS does not support fieldbuses. b Three architecture styles exist for the designing to embed the ROS system: Embedded PC, Proprietary system with interface, or use of ROS messaging and APIs. The embedded PC would require all control hardware of subsystems to be fitted to the PC itself. This provides smooth integration with ROS, but fails to offer real hard time support. Entire existing robots can be managed from ROS by configuring a translation function between the proprietary system and the embedded system. This abstracts the lower level programming and allows for the existing systems to operate in real time. The third architecture is focused on the IPC system for ROS, which operates on remote procedure calls and publish/subscribe support.

The ROS IPC system allows for custom messages to be used and provides flexibility in configuring the embedded systems. Rosserial, rosc, and Rosbridge are common methods of passing these messages. Rosserial provides a proxy over a C++ client that can be ported easily to any system that supports the language, not required an OS. Rosserial provides a ROS-like API to publish, subscribe, offer and consume RPC services. Potential issues arise with the proxy becoming a bottleneck. Rosc allows for direct connection to the Ethernet and handles native ROS connections and messages. However, for embedded systems, the TCP/IP overhead may prove too overwhelming for low powered systems. Rosbridge provides dynamic socket and websocket access to the full capabilities of ROS

The emphasis of robotics research has shifted from larger platforms to smaller low cost platforms. This paper evaluates two Arduino based robotic systems: TraxBot and StingBot for use in an educational environment. Arduino provides a simple programming interfaces, highly available expansion/interfaces boards, and large online resources for sample code. ROS provides libraries and tools to be used across wide variety of hardware, allow for greater code reuse across different projects. ROS goals are to provide hardware abstraction, low level device control, easy implementation of common functionality, and message passing between processes. There are many commercially available robotic platforms (Roomba, lego mindstorms, etc) that can be cheaply obtained and have ROS drivers publically available. Existing commercial platforms offer little hardware expandability, but the growing Arduino community would solve this issue. The normal rossearial interface used for Arduino projects has high overhead and result in an excessive workload for the processor evaluated (Atmel 328P). For this Arduino system, the SRAM limitations provided a maximum of only 15 standard ROS topics to be used in parallel and the message buffer was limited to 70 standard messages. To circumvent this problem, the researchers developed a custom ROS driver that utilizes serial communications where ROS messages sent from the arduino are represented by character arrays. Translation can take place on the more powerful PC running the ROS core. There is a separate ROS node running to interface with the custom serial packet developed. By using this node/driver combination to integrate any Arduino platform, the benefits of existing Arduino hardware can be easily implemented in any existing ROS system. Regardless if the project is small or spans multiple teams in multiple locations.

III. Test Procedures

In order to evaluate each of the discussed IPC methods, a test system was developed that incorporated a laptop, 100mbps wired network connection, a raspberry pi SBC, and an OpenCM9.0 microcontroller board designed for command and control of dynamixel brand smart servomotors. This system incorporates communication links over ethernet, serial, and USB. For consitentcy the serial link was set to the same baud rate (9600) as the USB link. figure below shows the overall system diagram of the test setup.

Appendix

Placeholder for appendicies

Acknowledgment

placeholder for acknowledgements

References

[1] N. Dantam and M. Stilman, "Robust and efficient communication for real-time multi-process robot software," in *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, Nov 2012, pp. 316–322.

[2] M. Grey, N. Dantam, D. Lofaro, A. Bobick, M. Egerstedt, P. Oh, and M. Stilman, "Multi-process control software for hubo2 plus robot," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, April 2013, pp. 1–6.

[3] D. K. Cho, S. H. Chun, J. G. Ahn, Y. S. Kwon, J. H. Eom, and Y. I. Kim, "A udp-based protocol for mobile robot control over wireless internet," in *Robot Communication and Coordination, 2009. ROBOCOMM '09. Second International Conference on*, March 2009, pp. 1–4.

[4] M. Sstrik, "mq: The theoretical foundation," 250bpm, Tech. Rep., 09 2011.

[5] P. Bouchier, "Embedded ros [ros topics]," *Robotics Automation Magazine, IEEE*, vol. 20, no. 2, pp. 17–19, June 2013.

[6] P. Hintjens and M. Sustrik, "Multithreaded magic with mq," iMatrix Corporation, Tech. Rep., 09 2010.