Tiny Nets

Adaptable Networking for Distributed Embedded Computing

Jake Read, Dougie Kogut, Nick Selby, Patrick Wahl

October 2017

1 Introduction

1.1 What/ What For

What about short timescale distributed computing? Most networks are optimized for total throughput - the average number of bits that can be transmitted in a second. Throughput is desirable in streaming applications, browsing, and nearly any human-timescale based interaction, but ping time (or latency) is often good-enough (say, tenths of a second being noticeable).

Where multiple nodes need to coordinate in near real-time, e.g. in robotics and automation or in augmented reality and physical interface applications, packet size and throughput is not so critical, but latency is. Robots are not streaming video to each other, but nodes need to collectively close control loops in sub-millisecond timescales. Data rates can be smaller, but ping times must be minimized.

1.2 Why Now?

Embedded computing has changed the way we develop hardware. Today, a 32-bit 120MHz CPU with hardware floating point operations and a WiFi radio can be had for $5. A chip like this is a few grams x a few millimeters and runs on milliamps of current. However, coordinating and organizing computation between more than one of these processors is non-trivial. In order to truly leverage this dropping cost curve, we need inter-processor communication that is fast, reliable, and decentralized. We need better tiny networks in order to enable massively parallel embedded computing.

1.3 The State of the Art

The state-of-the-art for inter-processor communications can be surveyed with a few established technologies:

UART, SPI, I2C, EtherCAT and CANBus

All of these technologies assert a static system architecture: UART has Baud Rates, SPI has clock speeds and highly regulated master/slave hierarchies, EtherCAT is essentially a faster, proprietary SPI, CANBus is slow and cumbersome, etc. In any of these architectures it is non-trivial to add processors (extra data lines are needed, each node must be configured to match the system), processors must operate at similar speeds and follow the same protocols, and often have the same exact hardware. This means that developing complex embedded systems is cumbersome, and relegated to experts. Robots are not democratized! Automation is in the hands of the very well trained.

1.4 The Internet's Insights

The Internet became successful when performance standards were abandoned and interoperability was expanded. It's an interconnect of different networks, not a singular holistic system. This is what allowed it to grow and change and become the sprawling, wonderful mess it is today. The same systems ethos does not exist in embedded computing, where systems have to be hand-engineered one at a time, to work in very particular regimes, for very particular tasks.

2 Proposal

The notion here is to explore all layers of network architecture - the PHY, MAC, Network and Transport - where assumptions about throughput are set aside in favour of minimizing latency, and maximizing interoperability for physical systems - while minimizing size and overhead complexity. I.E. how fast can a motor controller request and get information from an encoder, or the position of a human interface device, etc. How quickly can systems which operate in 'real-time' be modified, extended, adapted, etc.

2.1 Hardware

The nature of the project is such that it allows explorations with relatively low-cost, easy to boot and acquire hardware. Namely Atmel AVR chips, ARM chips, or some variant (typ. $<$ \$10/IC). Developing circuits and writing code for these hardwares is low-tech, and easy to get into - but the insights and explorations are potentially of relatively high value.

2.2 Key Ideas and Questions

1. Interoperability
    - No clock, no bitrate: per-bit (or per byte?) handshaking is used to regulate speed.
    - 8MHz Processor can communicate with 2GHz Processor without configuring clock speeds.
    - Packet length is fluid.
2. Packet Automata / Node Automata
    - To consider networking under an assumption that global states or routing tables do not or cannot exist.
    - A packet must carry its own route, nodes only know who their neighbours are.
    - How does this reconcile with an idea of global addressing?
    - How does this reconcile with an idea of global state-machine systems architecture? w/r/t robotics state-of-the-art?
    - Use port-flooding to discover routes.
3. Local and Global States
    - Fluidity across timescales.
    - Fluidity across computing scales.
    - Small-timescale and large-timescale state machines sharing a network. I.E. a motor-encoder-loadcell 'branch' of the network operates a 100kHz control loop. It receives inputs from and delivers outputs to a 10kHz path-planning control loop. This 10kHz path-planning loop receives inputs from and delivers outputs to a 1kHz environment sensing system. All three loops participate in human / computer interaction (wherein a human's response time to touch is 0.15 seconds) at 6Hz.
4. Writing Programs on the Network
    - Many use-cases for the proposed networks are in Robotics and Control. In some sense the proposed network enables massively parallel computing in these fields.
    - Is the network engineered such that it interfaces (to a systems designer) as if it were a distributed computing architecture? Local / Remote Memory addresses, function calls, etc ?
    - Can we speculate on how these systems might be developed, debugged, and designed?
5. Path Planning
    - Classical Path Planning Algorithms (e.g. DFS, BFS, Greedy, Dijkstra) are powerful tools, but are often optimized for graphs where all topography information is readily accessible.
    - Heuristic Methods (e.g. RRT, RRT-Connect, Hierarchical Approximate Cell Decomposition) have similar problems, but can offer insights into dealing with uncertainty.
    - Distributed Path Planning Considerations. The additional cost of node discovery and practical limits on memory make previous "optimized" methods impractical and require a new framework to think about efficiency in information paths.

2.3 Implementation

- Use low-cost, ubiquitous hardware and extremely simple wiring in order to rapidly prototype from the bit-transfer level up.
- Of interest as a carry-along question is whether, or to what extent, we can use Programmable Systems-On-A-Chip or FPGA's to extend the speed of our PHY layer. I.E. if we are handshaking on every bit in order to auto-regulate speed, can we develop a simple gate-based logic circuit for handshaking, interfaced with a bit-shift register to push data out?

3 Goals

3.1 Radical Interoperability

Demonstrate large processors communicating successfully with small processors (i.e. a Raspberry Pi can

perform packet transfers with an 8MHz processor, without configuring bitrates)

3.2 Route Discovery Algorithms

Develop and benchmark methods for the discovery of global routes through a distributed network. Analyze the performance of classical path planning algorithms and modern heuristic methods. For example, in classical graph theory approaches, a k+1 Greedy algorithm will usually outperform a k Greedy algorithm. However, this assumes that determining the cost of each of the paths in the extra step is negligible. In a distributed system, because communication is required to determine information about a possible path, there will be a k-threshold after which it becomes impractical to investigate the cost of continuing an extra step. This dynamic will significantly affect the information path planning paradigm.

3.3 Route Optimization Algorithms

Develop and benchmark methods for optimizing route discovery our route selection based on (1) minimum latency or (2) maximum throughput or (3) minimum network-computing-cost (i.e. leaving as much of the network as possible free to do other, more important work).

3.4 Ping Testing

We hope to produce some tests which demonstrate the network's ability to reliably transmit data at low latency, with increasing (1) # of nodes in the transmit route, (2) # of nodes simultaneously transmitting, (3) # of nodes transmitting at routes whose traffic intersects test-traffic.

3.5 Bulletproofness

Develop methods for continued network operation in the event of lost nodes.

3.6 The Demonstrator

A demonstrator 10x10 array of network nodes with 'arrows' wherein network computing and location-based addressing is used to point at nodes which are 'interacted with'. I.E. the user pushes a button on one node, all other nodes point to this node. This demonstration uses or implements much of the aforementioned work: location-based addressing, path discovery, flood routes, and interaction with physical systems.

4 Milestones / Schedule Outline

1. **October 12th**
   Hardware v0.1
   - Bring hardware and IDEs online.
2. **October 23rd**
   Project Report 1
   - Hardware v0.2, Bit-level handshaking / protocol
   - Measuring Bitrates, Handshakes between different hardwares
3. **November 7th**
   Project Meetings
   - Demonstrate a small graph network with some version of addressing, packet protocol, and routing.
4. **November 21st**
   Project Report 2
   - More Nodes, Bigger Graphs
   - Graph Discovery
   - Route Optimizations
   - Question assumptions about classical path planning w/r/t 'truly distributed' systems
5. **November 30th**
   Work on hardware for demos.
6. **December 7th**
   Demo development and project documentation.
7. **December 12th**
   Project Presentations

5 Reading / Resources

- Ring Test (https://pub.pages.cba.mit.edu/ring/)
  - where Ring MHz is proportional to MB/s upper limit per processor
- Conway's Game of Life
  - Classic touch-point to consider computing as small rule-sets and minimum local states 'emerging' with higher order complexities.
- Internet Routing Architecture (?)

  - What is the internet's routing strategy, and where do we stand to challenge the assumptions made during it's design? Do Routing Tables need to exist?

- Internet Zero (http://cba.mit.edu/docs/papers/06.09.i0.pdf)