Tiny Nets: Project Report One
A Small Network with Industrial Dreams

Jake Read, Dougie Kogut, Nick Selby, Patrick Wahl

October 2017

1 Overview of Progress

Following our re-focus on Networked Control Systems, and in response to the challenge posed by [Dan+14] that "existing solutions [for Networked Control Systems] will not fulfill the future challenges in terms of reliability, scalability, and flexibility", we have designed a new switched-medium network. The network can be implemented on almost any micro-controller available to systems engineers using a small software include, has a configurable number of addresses, and uses back-pressure routing to maintain low delivery times in the face of switch congestion - a common problem in Switched Ethernet implementations.

In this project, we will be particularly focused on the network's ability to dynamically re-route packages along multiple routes in the face of different network congestion scenarios. As discussed in our project proposal, Switched Ethernet uses a Spanning Tree Protocol to cull connections in a graph, such that only one route to any given endpoint exists. In the case that a switch is occupied transmitting from one node, any other node below it must wait before its message can be passed along to the rest of the network.

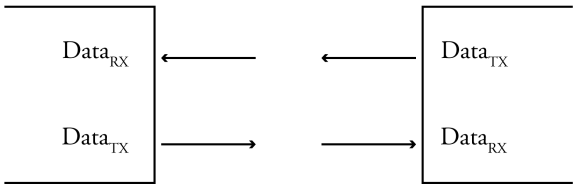
While the associated wait-times are negligible in regular internet traffic, their minimization is of critical importance in Networked Control Systems. We propose a solution where network switches maintain a list of possible routes to any given packet's final destination, and themselves use real-time information about neighbouring switches' current states (idle, busy, or current buffer length) in order to route packets intelligently.

2 TinyNet Implementation

Here we will briefly describe how TinyNet is implemented physically, how it structures packets, and how switches in TinyNet route messages.

2.1 Our PHY

We use UART Hardware to link nodes. UART is a ubiquitous micro-controller peripheral - at least one port can be found on almost any micro-controller on the market today, and often many more are present. The micro-controllers we have begun building with have five ports, and micro-controllers in the same family have up to nine ports, allowing for the development of radically interconnected graphs. By using UART Hardware, we can bring nearly any embedded system onto the network with only two micro-controller pins and a software library. UART is full-duplex, and requires little configuration.



UART Hardware:

Ubiquitous microcontroller peripheral. Uses 8-bit words on a full-duplex connection, typically modulated at the microcontroller's logic level.

2.2 Our Packet

As opposed to Ethernet's minimum size of 672 bits, we implement a packet with a 48 bit minimum. This is of primary importance for driving message delivery times down, and we expect to approach Ethernet's delivery time without any of the purpose built ICs and electronics that are found in Ethernet switches and endpoints.
For Ethernet, the Time to Transmit a message,

$$T_m = \frac{bits/packet}{bitrate}$$

$$T_{m_{ethernet}} = \frac{672}{100*10^6} = 6\mu s$$

We use UART Hardware with a 2Mbps delivery rate, that we may be able to increase to 4Mbps.

$$T_{m_{uart}} = \frac{48}{2*10^6} = 24\mu s$$

It is clear to see that if time were spent to engineer a higher bitrate into the physical layer, a new network architecture using radically smaller packets could deliver a major improvement in delivery time. We are interested in using FPGA technology in order to implement higher bitrates on our network.

Our Packet contains destination and source information,

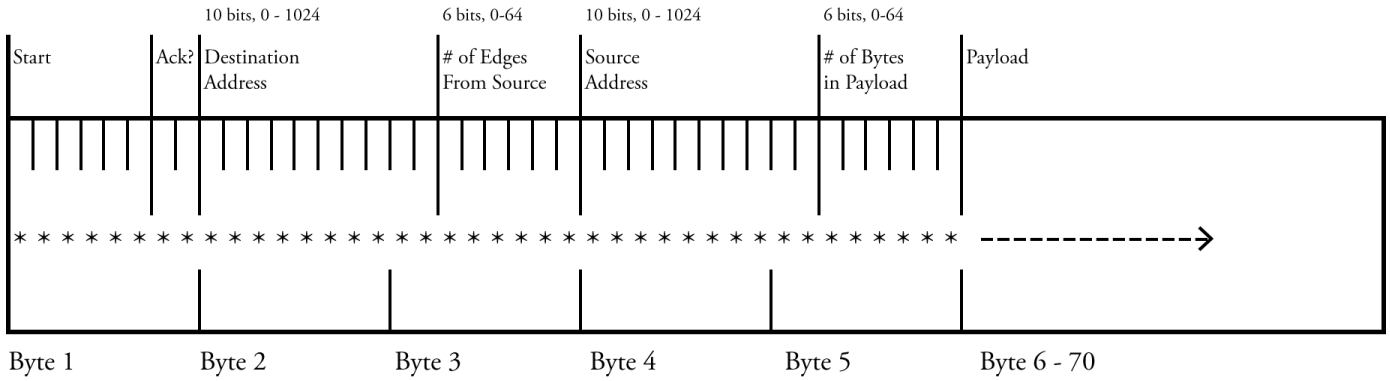


Figure 1: A TinyNet Packet. A Start Byte, Destination and Source Addresses, as well as the Number of Edges the Packet has crossed prior to arriving at the switch, the Number of Bytes in the Payload, and of course the Payload itself.

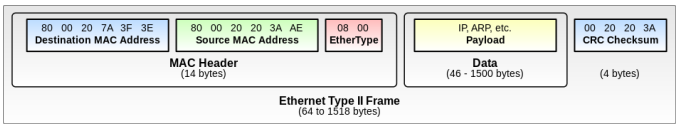


Figure 2: An Ethernet Packet.

a payload, and critically, 6 bits describing the number of edges it has crossed before arriving at the current switch. These additional 6 bits are critical in allowing the switch to make informed decisions about where to deliver the packet next, as will be discussed in the following subsection.

2.3 Our Routing Protocol

Switched Ethernet's Spanning Tree Protocol is designed to eliminate all but one possible route to any given endpoint. While this increases network simplicity, it does not allow for any dynamic routing, and promotes 'star' topologies rather than truly distributed networks. When an Ethernet switch is occupied with an incoming transmission, other nodes that share the switch as a gateway to the network must wait before they are passed along.

We see this as a critical limitation to deterministic message delivery. In order to switch packets in a dynamic manner, we push a small amount of routing information into the packet itself, in the form of a count of the edges that that packet has traversed. The Edge Count value begins at zero when the packet departs from it's source, and each switch in series increments the value by one.

To begin explaining how we expect to route packets, we offer this simple example network, and an overview of Routing, Address Discovery, and Acks.

1. **Routing Tables:**

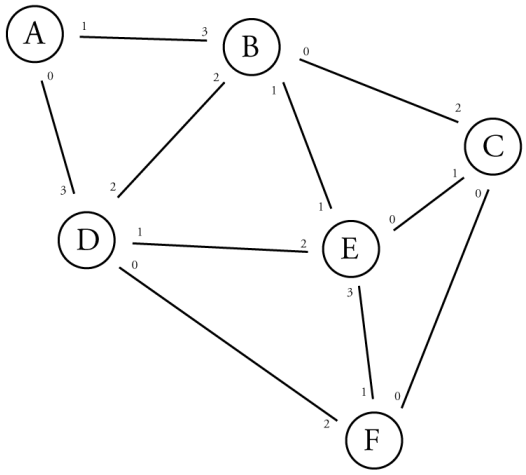


Figure 3: A example TinyNet graph. Switches have addresses (A, B, ... F) and Ports (E_0, ... E_4).

Much like Ethernet, switches in a TinyNet build routing tables. Each switch has a table with a list of addresses, a list of the ports that packets from those addresses have appeared on, and data about the particular connection on that port - namely, how many edges a packet crossed before it arrived on that port. Switches pull this data out of packets they see during operation. An example of an ideal routing table for our example graph is show in Figure 4

2. **Port Selection:**
Data in this table is used when a switch receives a packet. For example, if Switch E has previously received a packet from A on Port E_1 with an edge

An Ideal Address Table from Node E's Memory

Address	Port	Min. Edges	
0			
.			
.			
.			
A	0	3	
	1	2	
	2	2	
.	3	3	
.			
.			
B	0	2	
	1	1	
	2	2	
.	3	3	
.			
.			
C	0	1	
	1	2	
	2	3	
.	3	2	
.			
.			
.			
.			
.			

Figure 4: An ideal table for Node E from Figure **??**.

count of 2, as well as a packet from A on Port E_0 with an edge count of 3, it can select port E_1 as the ideal port to transmit a packet that is destined for A.

3. **Back Pressure Routing:**
   Channels are all full-duplex. When a switches sate updates (from idle to busy), it can deliver this data to neighbouring switches, even when receiving a frame. When a switch has a packet to deliver, it uses this data to further inform its routing decision. If the transmitter sees a busy switch upstream, it can loop through its lookup table to determine where the next-best switch to transmit on would be. In this way, messages always travel along a route which is most likely to take the minimum time.

4. **Flood Forwarding and Discovery:**
   In the case that a switch receives a packet with a destination address that is not yet contained in its table, it forwards the packet to all other neighbouring ports, so long as the packet has not already travelled over a certain number of edges (this prevents packets ringing through the network). The 'Maximum Number of Hops' can be defined fluidly in a network implementation, and can be designed such that any message will always find at least one route to its destination.

5. **Acks:**
   When packets contain an Ack Flag, they return to their sender (without a payload). Ack Requests and Flood Packets allow new nodes to discover network topology, by sending flood packets with ack-requests, and recording the responses.

3 Current Hardware Progress

In order to test the success of our network, we are building testbed hardware and firmware. We will use these circuits first to measure Packet Delivery Times in the following scenarios:

1. Between a pair of switches - a simple function of bitrate.

2. Across a chain of switches - a function of bitrate and switching time - i.e. overhead processing time that takes place within a switch.

3. Through a highly connected graph. Switches will have to collect route information using flood-discovery algorithms in order to route packets.

4. Through a highly connected graph, with increasing cross-traffic. This is the critical measurement; in existing Networked Control Systems, overall network congestion contributes strongly to increased wait times for individual messages.

So far, we have designed a switching circuit using an Atmel XMEGA processor running at 48MHz and with UART clocks of 2Mbps on each port. We have assembled five of these switches, and have written firmware that uses interrupt routines to buffer data on individual ports. We have successfully passed bytes through a daisy-chain of five nodes.

Going forwards, we will use a logic analyzer to time packet transitions between switches as we continue to develop firmware and switching protocols.

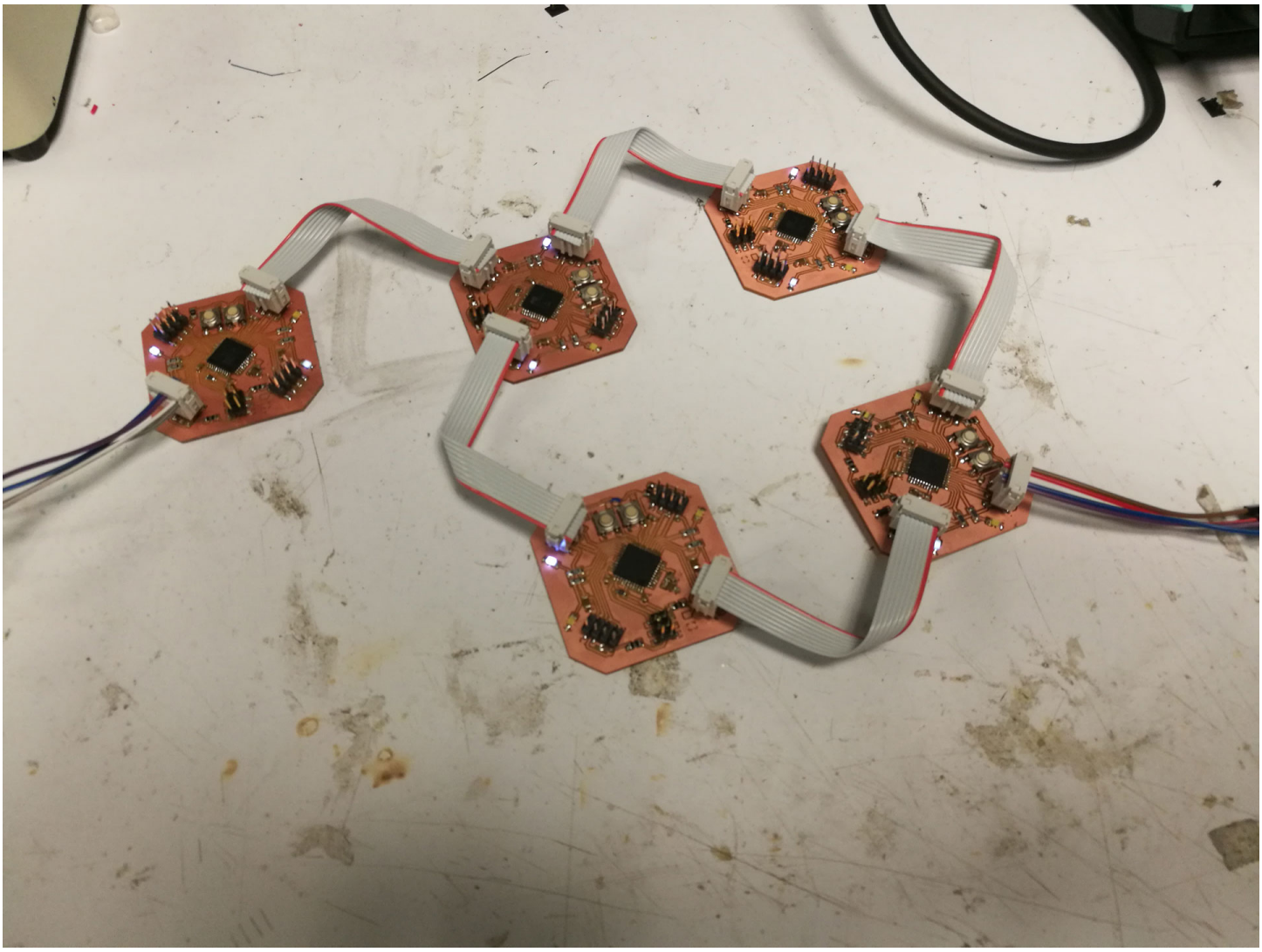


Figure 5: Five boards in our first switching testbed.

4 Simulation

In addition to building physical hardware, we have begun implementation of a computer simulated network. By coupling this simulation with physical experiments that characterize hardware performance, we expect to be able to extrapolate our results into networks with hundreds or thousands of nodes.

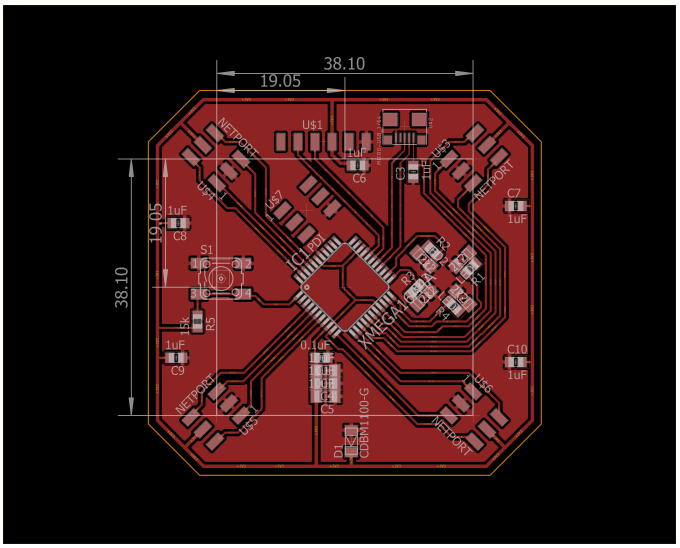


Figure 6: A network switch, implemented on an XMega Microprocessor, $5 total hardware cost.

References

[Dan+14] P. Danielis et al. “Survey on real-time communication via ethernet in industrial automation environments”. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. 2014, pp. 1–8. DOI: 10.1109/ETFA.2014.7005074.