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

Jake Read, Dougie Kogut, Nick Selby, Patrick Wahl

November 2017

1 Overview of Progress

Since the last report and meeting, we have:

- Clarified our Objectives, Cost Functions and Constraints
- Clarified the value of TinyNet's design against other multipath routing protocols
- Approached completion of a network simulation tool that will allow us to refine and measure TinyNet's performance
- Designed and Fabricated a new physical router with an order of magnitude increase communication bitrate and a sixfold increase in processing power, while staying under $20 in unit cost.

2 Objectives

Our goal is to develop and demonstrate a stateless multipath routing methodology for Networked Control Systems.

Networked Control Systems present unique constraints to network design. Most critically, total throughput is not a driving metric - messages tend to be very small, often only one packet in length. Rather, NCS must run with very low message delivery times, and most importantly these message delivery times must be highly deterministic. Control loops that run on multiple devices in the network must be closed within set time periods.

No current network architectures offer multipath routing without also including global network state-machines. When a node or a link is severed in stateful networks, or when the networks see bottlenecks at particular nodes, routing paths must re-converge. This process often takes seconds, with a lower bound of 200ms. Because these convergence algorithms add huge amounts of traffic to the network, they make big impacts on message delivery determinism.

With this, and Networked Control Systems in mind, we impose the following constraints:

- **Statelessness.** The network should perform multipath routing without any node containing information about the entire network graph.
- **Scalability in Traffic.** Unlike Ethernet, which has a packet delay that scales linearly with traffic at particular switches, the system should be able to take advantage of multipath routing to improve scalability to be logarithmic in traffic.
- **Short Decision Time.** Unlike state-of-the-art multipath routing algorithms that maintain knowledge of the entire graph and rebuild its spanning tree whenever a new node is advertised, networked control systems need to be able to respond quickly to changes in graph structure and local route busyness.
- **Minimal Overhead.** The system should be arbitrarily implemented on microcontrollers in software with system-designer defined hardware and network parameters. There should be no large jacks, proprietary IC's, or black box IC's. TinyNets will be the first open-source network for realtime control.

To accomplish our goal within the constraints, we present the following key contributions:

- A real-time cost function, using next-hop buffer size (i.e. a busyness metric) as well as historical hop-count for per-packet dynamic re-routing, that increases packet delivery-time determinism.
- Small packet sizing to reduce transmission time.
- An open-source routing protocol for arbitrary implementation in any embedded system, where computing, physical space, and time is limited.
- A multipath routing protocol that does not require global knowledge of the network topology.

Finally, to demonstrate the success of our project, our project consists of the following deliverables:

- An implementation of our network architecture on a real hardware system of twenty five nodes in variable configurations.
- A simulation of our network architecture in JavaScript on a network of 1000 to 10 000 nodes.
- Plot illustrating network utilization over time to ensure efficiency.
- Demonstration of dynamic packet re-routing in the case of link/router failure.
- Comparison of dynamic re-routing time of our system vs. network reconfiguration time in state-of-the-art techniques like ECMP and OSPF.
- Demonstration of multipath utilization around busy nodes.

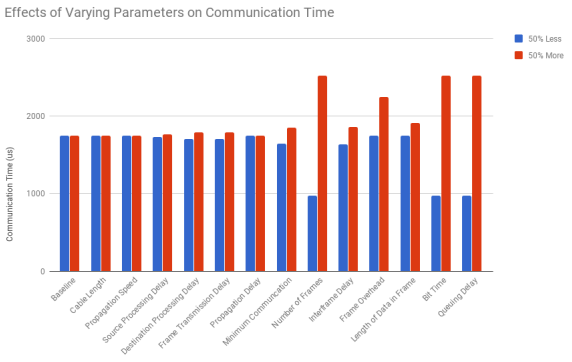

Figure 1: Effects of Varying Parameters on Communication Time.

- CDF of packet latency in real world traffic scenarios for our network compared to state-of-the-art techniques like Ethernet/IP.

- Demonstration of individual component effects, such as including the next-hop buffer size in the cost function or dynamic packet size, on performance metrics.

3 Proving Our Merit

Alternative methods to TinyNets for real-time control can be broadly classified into two distinct categories: stateless routing like Ethernet/IP and stateful routing like ECMP, OSPF, SPB, and TRILL.

3.1 TinyNets vs. Stateless Routing

The state-of-the-art in real-time control networks is Ethernet/IP [MT07], a stateless routing protocol which builds a spanning tree and calculates cost as the hop count between nodes. From [LLL06], we know that the worst case communication delay over Ethernet occurs when the number of frames attempting to pass through a single switch is the greatest. For example, when a spanning tree organizes itself such that 24 stations are connected to a single switching hub, a typical 144-bit message with a bit time of 0.1 us would take more than 1.5 ms to finish sending a single packet from all stations. If the packets could be interconnected without the tree structure required by Ethernet/IP, transmission time could be brought down to just over 300 us.

Figure 1 compares strategies for reducing communication time. The parameter which has the largest effect on message delivery time is the number of frames being communicated over a single switch. This is to be expected, since that parameter will have a multiplicative effect on the queuing delay. By sacrificing the spanning tree topology and leveraging multipath routing without the added processing delays and stateful nature of ECMP or other link-state routing methodologies, we will drastically reduce frame count and, therefore, communication time.

3.2 Convergence time in other routing protocols

Convergence refers to the consistency in topological information that routers have about their network. In our protocol, convergence will refer to the delay that occurs when a node is removed from the network and the subsequent dynamic packet flooding and re-routing occurs. Convergence is slightly different in our protocol since each node will not have a complete view of the network graph. However, convergence time from other routing protocols will serve as good benchmarks for ours, as convergence time measures the time it takes the network to operate normally in the event of a node failure or of node busyness. Convergence time when used in the paper will omit the time it takes to detect unresponsive nodes.

We analyze the failure scenarios in 3 protocols, namely OSPF, SPB, and TRILL, and conclude that our protocol will perform on par or better than them when measuring convergence time. In all 3 of these protocols, each node must know the entire network graph to calculate the shortest path between a source and destination. Thus, when a node goes down, the entire network will halt and update their view of the network graph, drastically decreasing message delivery time determinism. In our protocol when a node is withdrawn, only the neighbors of that node will need to update their network data as the other nodes with information on the downed-node will naturally update their data upon standard communication from other nodes. These protocols seem to imply that convergence times of around 200ms are performant[FA09][ELC08].

Heavily associated with convergence time is the ability for routing protocols to consider multiple paths. Each of the 3 protocols above offers strategies that allow this. TRILL uses Fabric Shortest Path First (FSPF) to find an alternate route upon topology change. OSPF and SPB can be configured to use ECMP to maintain multiple paths when there exists more than one path that has the same cost. These 3 protocols offer dynamic path finding only in certain scenarios whereas our protocol will naturally use a greedily found path so long as there exists any path at all — and it will do it without nearly as much message flooding as the aforementioned protocols.

To actually measure the performance of our protocol in a failure scenario, we propose several experiments to measure the convergence time among the 4 protocols:

- An experiment that tests the latency of corner node communication in a grid-structured network upon node failure. This can be a good control test since there are many shortest paths between corner nodes and each protocol should be expected to perform similarly since each handles the case of multiple shortest paths well.

- An experiment that tests the latency between communication of two nodes in a ring-structured network upon node failure. Since there are exactly two paths between any pair of nodes in such a network, this experiment will almost solely test the convergence time since the latency of the path chosen by each proto-

col will necessarily be the same and the timing of the communication will only be different due to dynamic path finding calculations.

- An experiment that tests the communication within a fully connected (or near fully connected) graph upon node failure. This experiment would serve to test the goodput within networks since we know that flooding occurs when nodes are removed and we want the amount of ringing of these messages to be minimal.

4 Simulation

In addition to building a small hardware network of 25 nodes, we wish to demonstrate the scalability of our protocol. To do this we are developing a simulation and visualization tool that allows us to model and artificially scale towards thousands of nodes.

As of this report, the simulation can successfully model:

- **Packet Transmission, Reception, Processing, and Forwarding.** The simulation can now handle the transmission and reception of standard packets and acknowledgements and their flood counterparts. Furthermore, each node can use the information in its routing table to decide along which port(s) to forward it. Nodes can also recognize if they are the intended packet recipient and acknowledge if necessary.

- **Distributed Learning.** Once packets begin flowing across the network, each node begins learning the fastest routes along which to send data. Nodes can do this without storing an adjacency matrix, spanning tree, or any other abstract graph model. Instead, the method in which packets are handled serves a double role of packet processing and continuously updating individual node's routing strategies.

The next task is to incorporate the final message type, buffer update and heartbeat. After that, the simulation should be able to model hardware failures and disconnects statelessly.

4.1 Implementation

The simulation software is implemented using the open-source Simbit library (https://github.com/ebfull/simbit), which was originally designed for simulating peer-to-peer networks like those on which Bitcoin runs. The user can define the initial network topology as an array of arrays representing the connections each node has on its various ports. This topology can be created algorithmically making it easy to scale the network up to thousands of nodes.

The simulation models the software running on each processor in the network as a "manager" which handles incoming requests on each of its ports. Each manager keeps track of its own lookup and buffer depth tables, and will process incoming requests using the forwarding algorithm. The managers send their actions to an asynchronously-running network controller, which facilitates communication between them. Manual simulation actions can also be fed to the managers in order to test different types of communication on the network.

5 Hardware Development

Since the last report, we have developed a new router capable of 20 Mbps communication over a differential (RS-485) UART PHY, with a 300MHz core clock for message handling. Our first prototype ran on a 48MHz clock with a maximum bitrate of 3Mbps. We believe that this will mark a substantial objective decrease in TinyNet message delivery times.

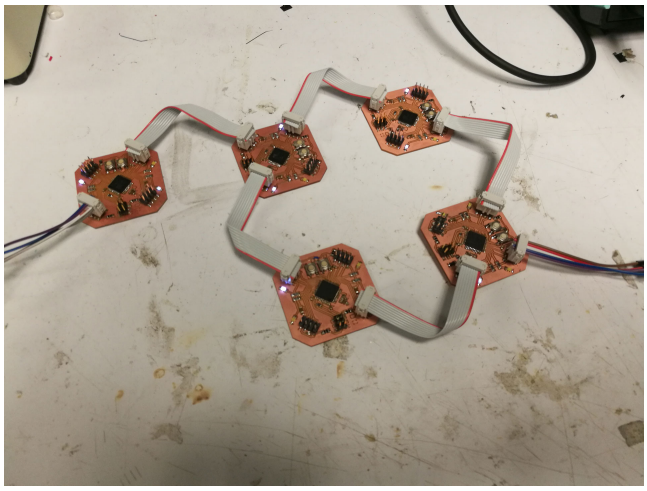

Figure 2: Our v0.1 Router.

Our new router also includes a pin header that breaks out many of the processor's other peripherals - GPIO, PWM, SPI, USART, I2C, ADC's etc - to allow development of endpoint hardware. One such piece of hardware, a brushless motor controller, is being developed.

References

[LLL06] K. C. Lee, S. Lee, and M. H. Lee. "Worst Case Communication Delay of Real-Time Industrial Switched Ethernet With Multiple Levels". In: *IEEE Transactions on Industrial Electronics* 53.5 (2006), pp. 1669–1676. ISSN: 0278-0046. DOI: 10.1109/TIE.2006.881986.

[MT07] J. R. Moyne and D. M. Tilbury. "The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data". In: *Proceedings of the IEEE* 95.1 (2007), pp. 29–47. ISSN: 0018-9219. DOI: 10.1109/JPROC.2006.887325.

[ELC08] V Eramo, M Listanti, and A Cianfrani. "Multipath OSPF performance of a software router in a link failure scenario". In: *Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008. IT-NEWS 2008. 4th International* (2008). DOI: 10.1109/ITNEWS.2008.4488153.

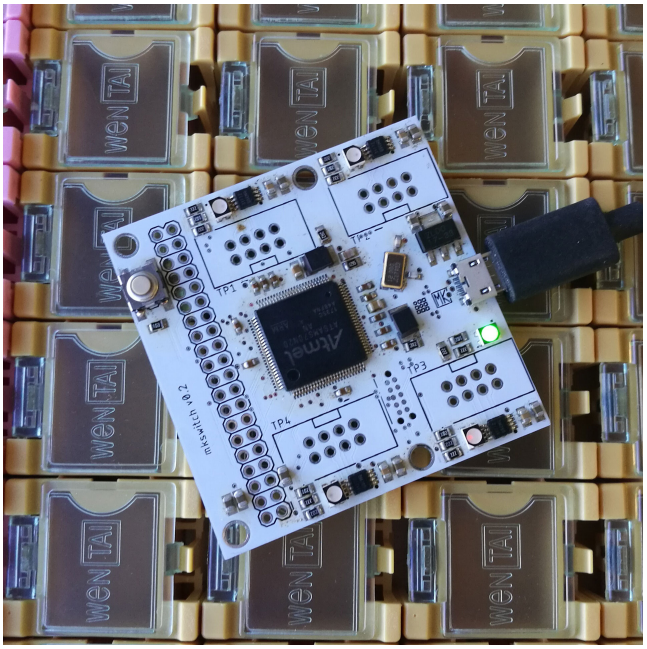

Figure 3: Our v0.2 Router: 300MHz core clock with 20Mbps RS-485 Driven UART.

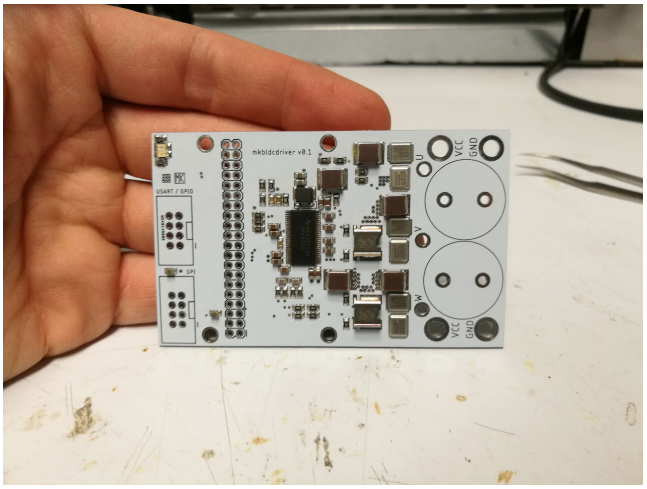

Figure 4: An add-on endpoint, a Brushless Motor Driver, to integrate TinyNet into robotics applications.

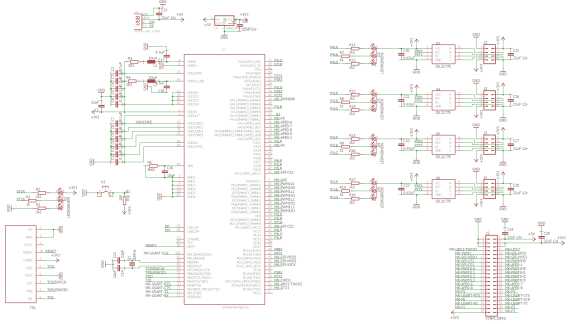

Figure 5: The router schematic.

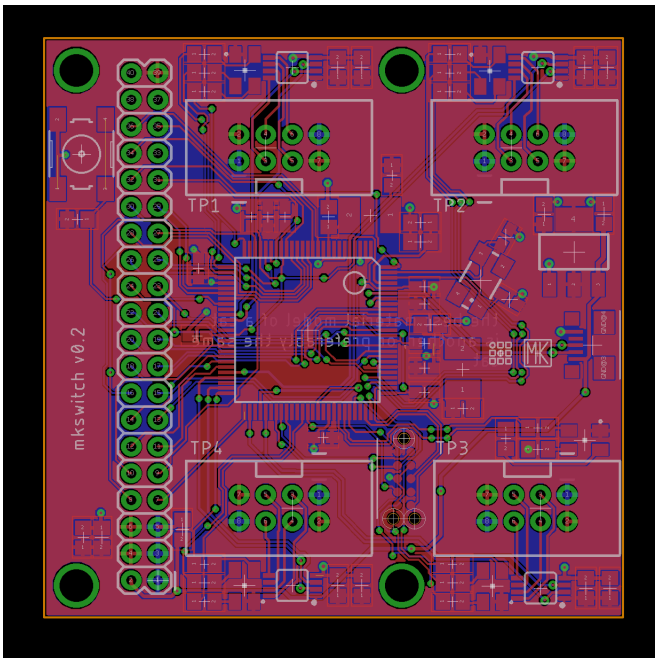

Figure 6: The router board file.

[FA09] J Farkas and Z Arato. "Performance Analysis of Shortest Path Bridging Control Protocols". In: *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE* (2009). ISSN: 1930-529X. DOI: 10.1109/GLOCOM.2009.5425776.