Neuron: a Framework for Decentralized Functional Programming

Eric Griffis
dedbox@gmail.com

October 23, 2018

1 Introduction

There seems to be a gap in the programming literature: where are all the decentralized programming models? A wealth of concurrent and distributed programming models have been published, but decentralized programming has received relatively little attention. To illustrate the difference, consider the following Racket implementation of a basic TCP echo server[1]:

```
(define listener (tcp-listen 12321))
(let echo-server ()
  (define-values (I O) (tcp-accept listener))
  (thread
   (λ () (copy-port I O) (close-output-port O)))
  (echo-server))
```

It can handle many clients simultaneously, but no progress is made until clients are connected. It is not gainfully distributable, yet all clients are trusted implicitly. So, is it decentralized? The answer is not in the code. *It depends on who controls the client.* If client and server are controlled by the same person, the two processes may trust each other implicitly. Otherwise, some amount of discretion is warranted:

```
(define listener (tcp-listen 12321))
(let picky-echo-server ()
  (define-values (I O) (tcp-accept listener))
  (thread
   (λ ()
     (let message-handler ()
       (define message (read-line I))
       (unless (eof-object? message)
         (when (regexp-match? #px"^KEY" message)
           (displayln message O)
           (flush-output O)
           (message-handler))))
     (close-output-port O)))
  (picky-echo-server))
```

This version processes each line separately. Valid lines begin with a pre-shared key. Clients that send an invalid line are dropped immediately. The server now has a way to delegate control to trusted clients. Unfortunately, the additional control flow and byte shuffling obscures its meaning. Still, even trusted clients can misbehave by failing to send a newline. Clients that connect and never send anything still occupy space in the server's TCP stack, and clients spewing random non-newline data will eventually fill the server's memory. In either case, a denial-of-service attack is possible. As before, the solution is straight forward yet more awkward.

The trouble is that decentralized programming is cumbersome without support for message-level communications or decentralized control. This paper introduces *Neuron*, a programming model for robust decentralized software in a functional style with minimal boilerplate.

- I explain precisely what makes a program decentralized and show that the design priorities of decentralized programming are orthogonal to those of distributed programming (Section 2). (I don't think this has been done before.)
- I give the syntax and semantics of the communication-based concurrency model underpinning Neuron (Section 3). It provides a feature-rich API for asymmetric messaging with local neighbors that accomodates common patterns like client-server, publish-subscribe, and content-based dispatch.
- I provide a design for messaging endpoints that encapsulate serial communications in process networks (Section 4). Ordinary functions like **read-line** and **write-json** may be converted into these endpoints and used interchangeably.
- I have implemented several non-trivial messaging constructs with Neuron and used them to build multi-player video game servers (Section 5). The results are compact, robust, and reusable.
- I give detailed quantitative measurements (Section 6) that show programs developed with Neuron perform comparably to other high level networked software development tools.

My conclusion is simple: we need more decentralized programming models.

[1] Taken from `https://rosettacode.org/wiki/Echo_server`