QUANTAS-WACOFA

Quantas With A Couple Of Features Added

Table of Contents

Problem

Quantas (**Oglio et al. 2022**) was lacking functionality to implement Byzantine behaviors.
My goal was to add Byzantine behaviors that would:

- Barely increase code complexity of Peers
- Work with many different types of Peers
- Be easily extendable

Approach

Write code that modifies the Peer code at runtime.
I call this code an “infection.”
Assumption: Every language can modify its code at runtime. Even assembly language can do this.
No doubt C++ can do this too.

50 hours later…

It turns out C++ can **not** do this.
C++ is not reflective.

Infections as higher-order functors

```
class Infection {
public:
    Infection(function<void(Peer<type_msg>*,function<void()>)> fn) : _infection(fn) {}
    Infection(function<void(Peer<type_msg>*)> fn) : _fn(fn) {}
    /** An infection is a higher-order functor:
     *
     * @param the peer that we are performing computation on
     * @param the original performComputation function
     * @return a modified version of performComputation
     */
    void operator()(Peer<type_msg>* peer, function<void()> performComputation) {
        if (_infection != nullptr)
            _infection(peer, performComputation);
        else
            _fn(peer, performComputation);
    }
  private:
    function<void(Peer<type_msg>*,function<void()>)> _infection;
    function<void(Peer<type_msg>*)> _fn;
};
```

Solution

An infection is a function that modifies a peer.
Infections can be specified in the config file.
An “infected” node is simply a node with modified behavior. It is no longer *guaranteed* to be correct, but it *may* be correct.

Peer.hpp

Old:

```
virtual void performComputation() = 0;
```

New:

```
virtual void defaultComputation() = 0;

auto computationPerformer = [] (Peer* peer)
    { peer->defaultComputation(); };

void performComputation()
    { computationPerformer(this); };
```

infect.hpp

```
template<class type_msg>
map<string, function<void(Peer<type_msg>* peer)>> infection = {
    { "crash", [](Peer<type_msg>* peer) {
        peer->computationPerformer = [](Peer<type_msg>* peer) {};
    } },
    { "censor", [](Peer<type_msg>* peer) {
        peer->submitTransPerformer = [](Peer<type_msg>* peer, int tranID) {};
    } },
    { "equivocate", [](Peer<type_msg>* peer) {
        peer->sendMsg = [](Peer<type_msg>* peer, type_msg msg)
            { peer->NetworkInterface<type_msg>::randomMulticast(msg); };
    } }
}
```

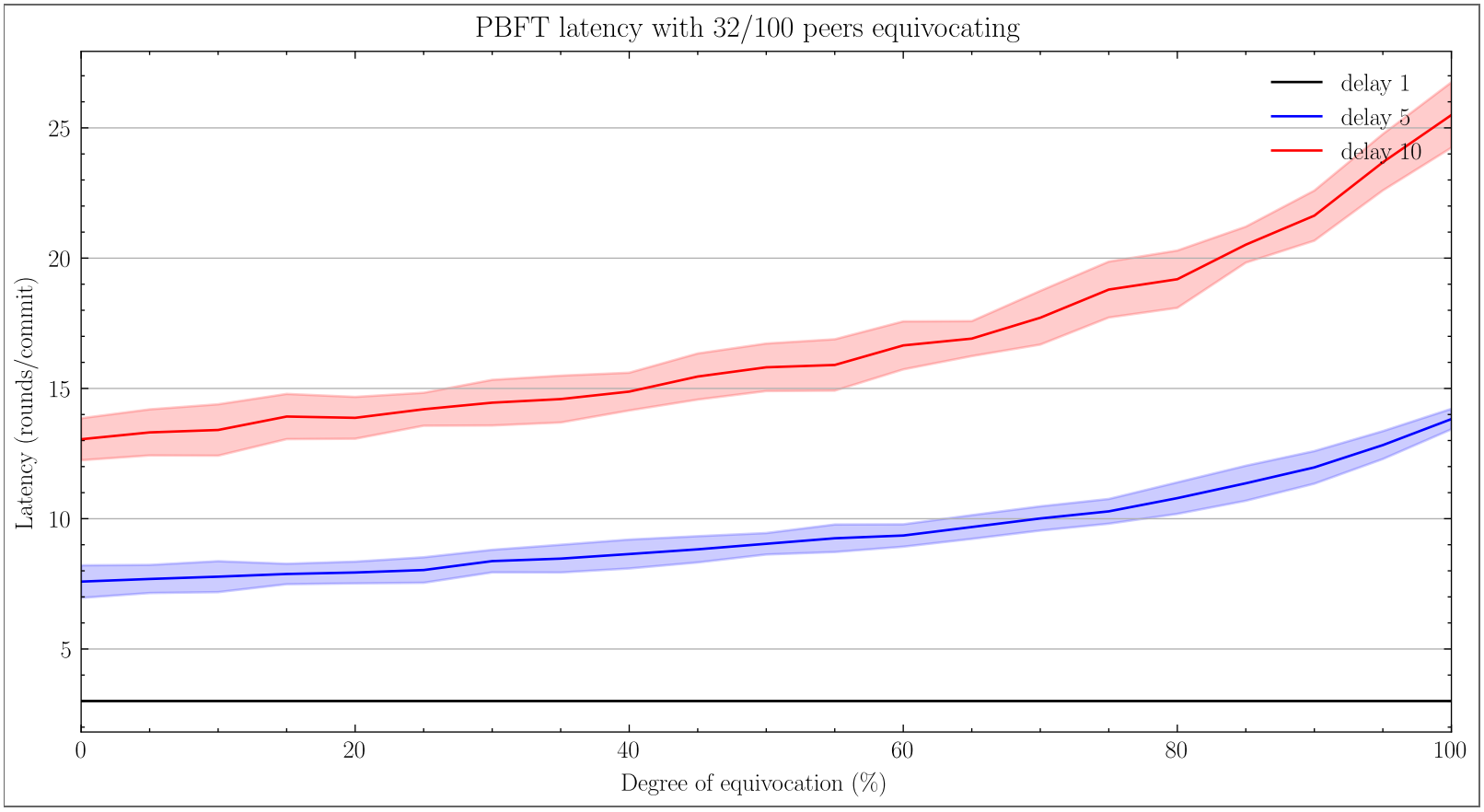
Input JSON

```
.
.
.
"infectPeersAtRound": 200,
"numberOfPeersToInfect": 32,
"infectionType": "equivocate",
.
.
.
```

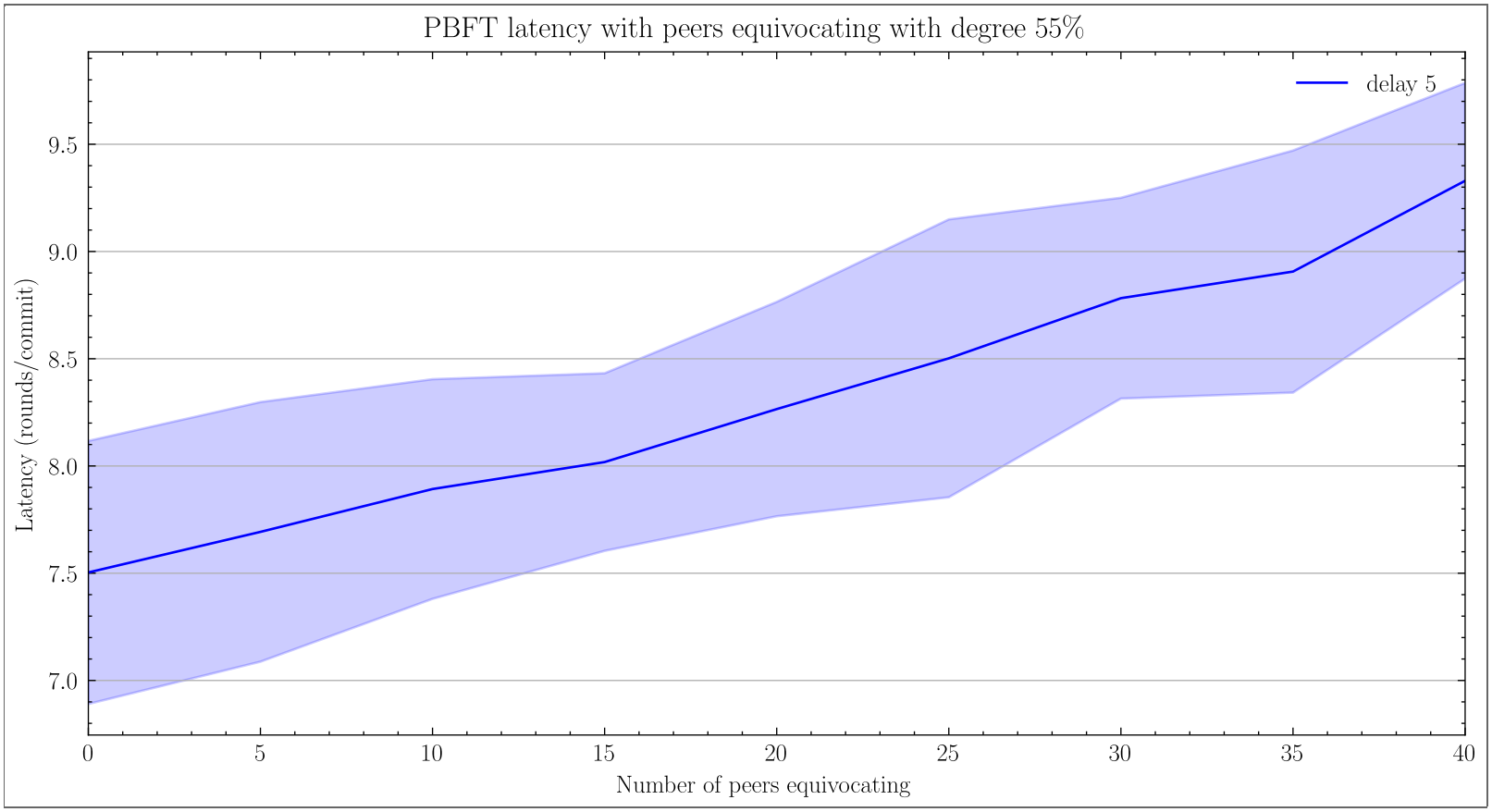
Results

Crashing, censoring, and equivocating, were all tested. Plots showing throughput over time are in **Proof of Work**.
Equivocating was tested in the trivial case that a peer sends a correct message to some nodes and no message to the rest.
(The details are moot because Quantas does not handle simulating DoS.)

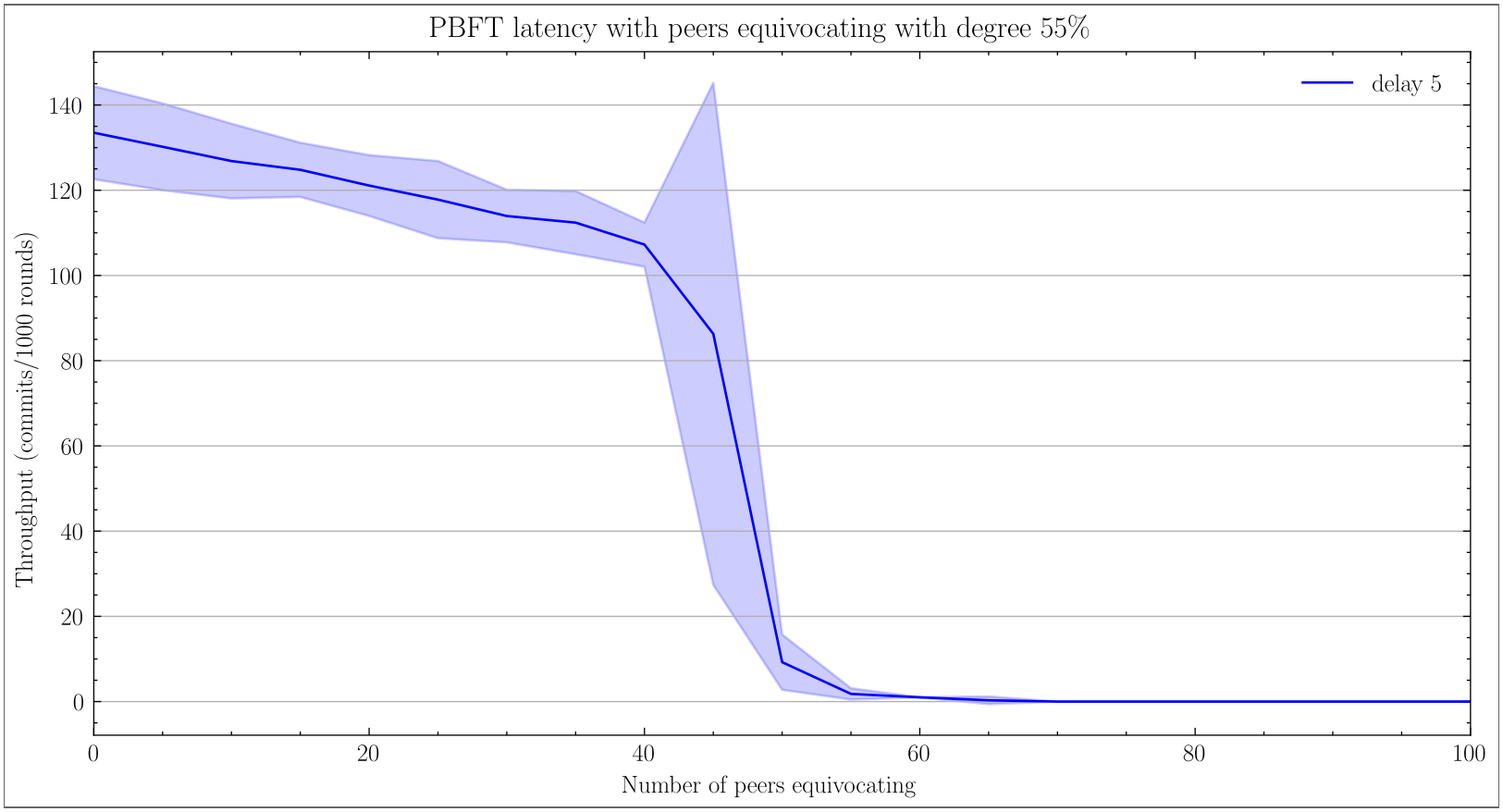
Varying Degree of Equivocation



Varying Number of Nodes Equivocating



Same, but with Throughput



Interpretation

Symbol	**Meaning**
Latency	The number of rounds to confirm a commit
Degree of equivocation	% of peers that are not sent a correct message
Black line	Latency averaged over 1000 rounds × 20 simulations
Red shaded region	95% confidence interval

Additional Problems Solved

- Developed a visualization solution with NumPy and Matplotlib.
- Developed a git post-receive hook that enables automated and reproducible simulations.
  - Output is stored as a git commit that contains a hash of the code that generated it.
  - (**Vallet, Michonneau, and Tournier 2022**)
- Produced recommendations for open source community engagement

References

astro, Miguel, and Barbara Liskov. 1999. “Practical Byzantine Fault Tolerance.” In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 173–86. Osdi ’99. New Orleans, Louisiana, USA: USENIX Association.

amblin, G, M Couplet, and B Iooss. 2013. “Numerical Studies of Space-Filling Designs: Optimization of Latin Hypercube Samples and Subprojection Properties.” *Journal of Simulation* 7 (4): 276–89. **https://doi.org/10.1057/jos.2013.16**.

glio, Joseph, Kendric Hood, Mikhail Nesterenko, and Sebastien Tixeuil. 2022. “Quantas: Quantitative User-Friendly Adaptable Networked Things Abstract Simulator.” *Proceedings of the 2022 Workshop on Advanced Tools, Programming Languages, and Platforms for Implementing and Evaluating Algorithms for Distributed Systems*, July. **https://doi.org/10.1145/3524053.3542744**.

allet, Nicolas, David Michonneau, and Simon Tournier. 2022. “Toward Practical Transparent Verifiable and Long-Term Reproducible Research Using Guix.” *Scientific Data* 9 (1). **https://doi.org/10.1038/s41597-022-01720-9**.