Trail of Bits Blog

Formal Analysis of the CBC Casper Consensus Algorithm with TLA+

- **POST**
- OCTOBER 25, 2019
- LEAVE A COMMENT

by Anne Ouyang, Piedmont Hills High School, San Jose, CA

As a summer intern at Trail of Bits, I used the PlusCal and TLA+ formal specification languages to explore Ethereum's CBC Casper consensus protocol and its Byzantine fault tolerance. This work was motivated by the Medium.com article Peer Review: CBC Casper (https://medium.com/@muneeb/peer-review-cbc-casper-30840a98c89a) by Muneeb Ali, Jude Nelson, and Aaron Blankstein, which indicated that CBC Casper's liveness properties impose stricter Byzantine fault thresholds than those suggested by the safety proof. To explore this, I specified the Casper the Friendly Binary consensus protocol in TLA+ (https://github.com/crytic/whipstaff).

As expected, it was impossible to determine finality without a Byzantine fault threshold of less than one-third of the total weight of all validators, which is consistent with Lamport et al.'s original paper on the Byzantine Generals Problem (https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf). However, as long as that condition was satisfied, CBC Casper appeared to function as intended.

First, a Little Background

The CBC Casper (Correct By Construction) protocol is a PoS consensus protocol designed to one day be used for Ethereum. However, the current state of CBC Casper hasn't undergone much formal analysis, and its under-specification poses a challenge to the introduction of Ethereum 2.0.

In a distributed network, individual nodes, called validators, use the Minimal CBC Casper family of consensus protocols to make consistent decisions. The protocol's five parameters are the names and weights of validators, fault tolerant threshold, consensus values, and estimator. Validators make individual decisions based on their current state, which is defined in terms of the messages received, which have three components:

- **Sender:** the name of a validator sending the message
- **Estimate:** a subset of values in the consensus values set
- **Justification:** a set of messages (state) received to arrive at the estimate

As a result, the sending and receiving of messages can be defined as state transitions.

Equivocation occurs when a validator sends a pair of messages that do not have each other in their justifications. The *future states* are all reachable states, where the equivocation fault is less than the fault tolerant threshold. Finality is defined by *safety oracles*, which detect when a certain property holds for all future states.

TLA+ and PlusCal

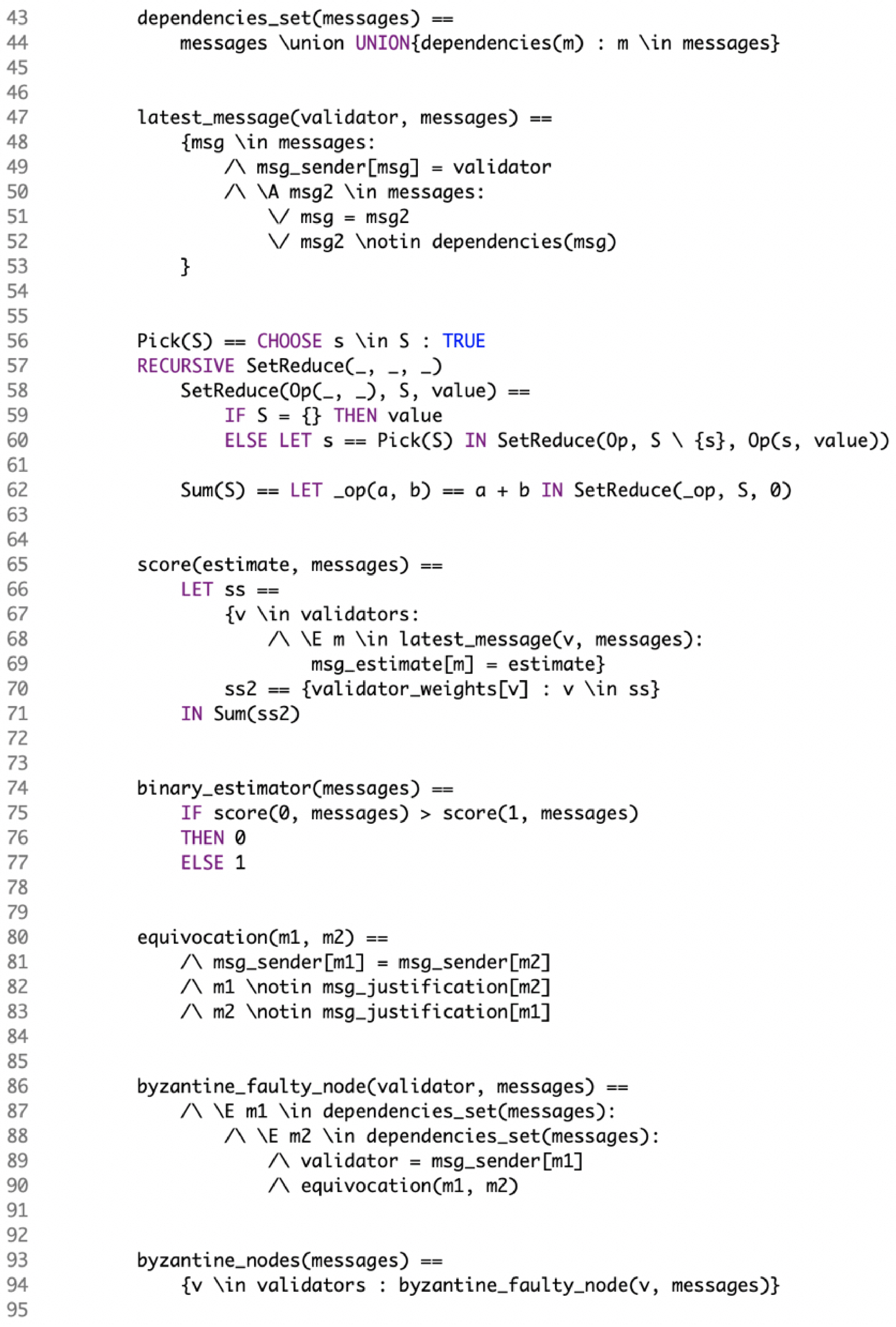
TLA+ is a formal specification language describing behaviors with states and state transitions. The specifications and state machine models can be checked with the TLC model checker (https://lamport.azurewebsites.net/tla/tools.html). TLC performs a breadth-first search over the defined state transitions and checks for the properties that need to be satisfied.

The Byzantine Generals Problem

For context, *The Byzantine Generals Problem (https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf)* is an analogy for decision-making in distributed systems in the presence of malicious individuals. The problem states that a commanding general must send an order to n-1 lieutenant generals such that **1)** all of them obey the same order and **2)** if the commanding general is loyal, then every loyal lieutenant obeys the order. A solution to the problem must ensure that all the loyal generals agree on the same plan, and a small number of traitors cannot lead the generals to a bad plan.

Now Let's Dive into the Process

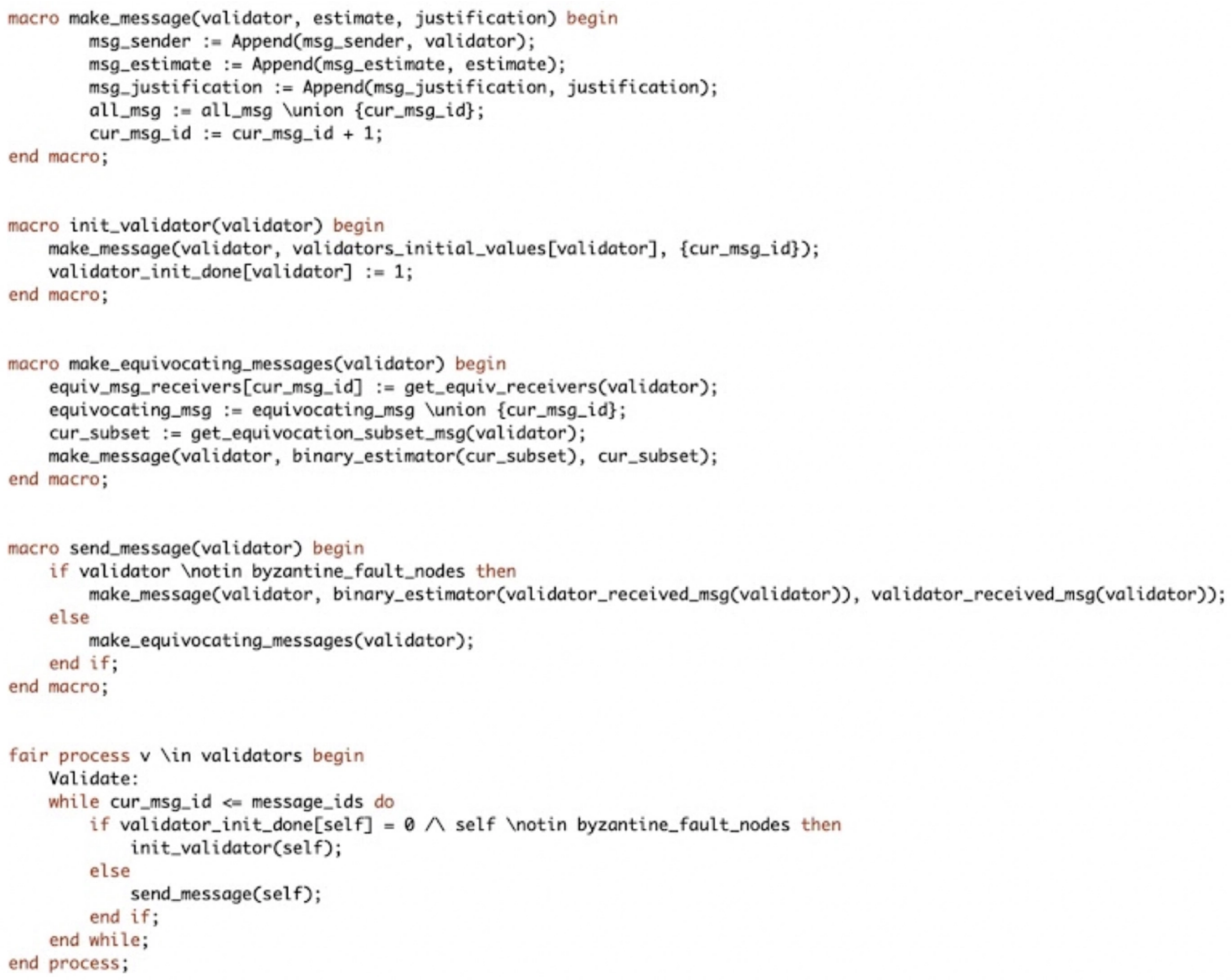
To start, I specified the definitions in the TLA+ language and defined the states and state transitions in terms of sets and set operations. Figure 1 shows a snippet of the specification.



```
dependencies_set(messages) ==
    messages \union UNION{dependencies(m) : m \in messages}


latest_message(validator, messages) ==
    {msg \in messages:
        /\ msg_sender[msg] = validator
        /\ \A msg2 \in messages:
            \/ msg = msg2
            \/ msg2 \notin dependencies(msg)
    }


Pick(S) == CHOOSE s \in S : TRUE
RECURSIVE SetReduce(_, _, _)
    SetReduce(Op(_, _), S, value) ==
        IF S = {} THEN value
        ELSE LET s == Pick(S) IN SetReduce(Op, S \ {s}, Op(s, value))

    Sum(S) == LET _op(a, b) == a + b IN SetReduce(_op, S, 0)


score(estimate, messages) ==
    LET ss ==
        {v \in validators:
            /\ \E m \in latest_message(v, messages):
                msg_estimate[m] = estimate}
        ss2 == {validator_weights[v] : v \in ss}
    IN Sum(ss2)


binary_estimator(messages) ==
    IF score(0, messages) > score(1, messages)
    THEN 0
    ELSE 1


equivocation(m1, m2) ==
    /\ msg_sender[m1] = msg_sender[m2]
    /\ m1 \notin msg_justification[m2]
    /\ m2 \notin msg_justification[m1]


byzantine_faulty_node(validator, messages) ==
    /\ \E m1 \in dependencies_set(messages):
        /\ \E m2 \in dependencies_set(messages):
            /\ validator = msg_sender[m1]
            /\ equivocation(m1, m2)


byzantine_nodes(messages) ==
    {v \in validators : byzantine_faulty_node(v, messages)}
```

Figure 1: Snippet of specification in TLA+

I specified the message relay in PlusCal, which is more pseudocode-like and can be automatically transpiled to TLA+ (https://lamport.azurewebsites.net/tla/toolbox.html).

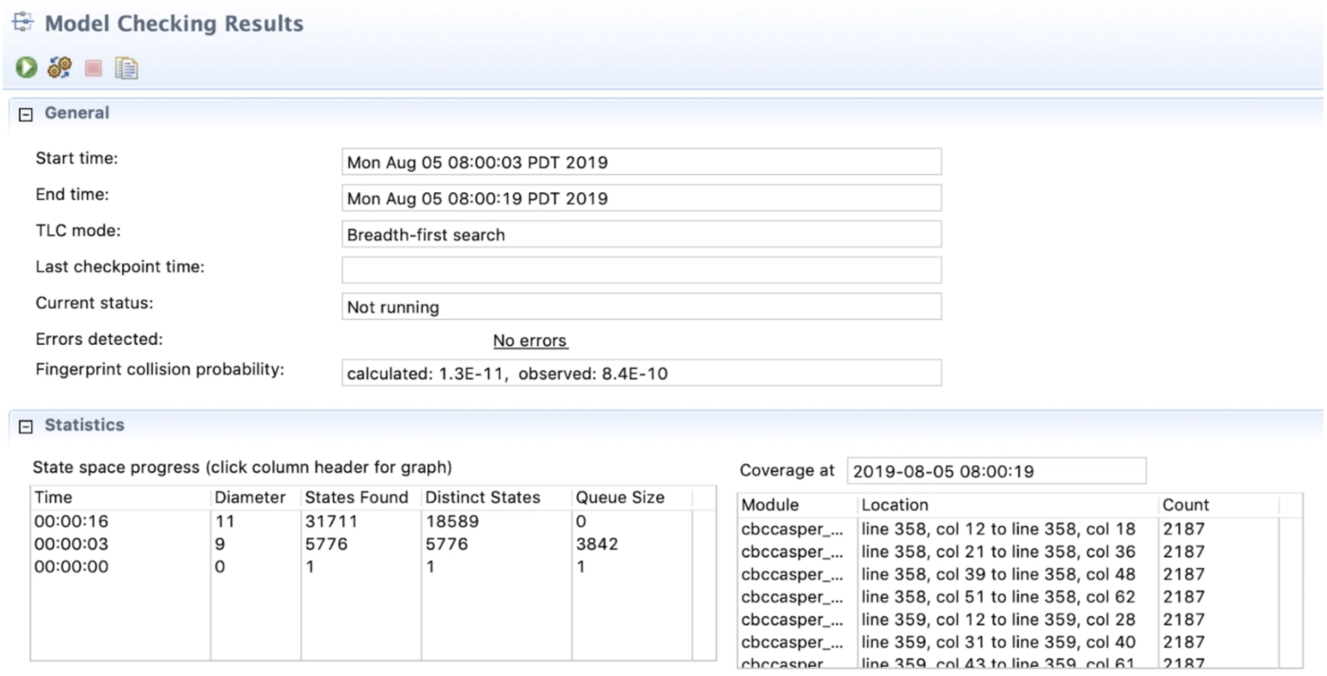


```
macro make_message(validator, estimate, justification) begin
        msg_sender := Append(msg_sender, validator);
        msg_estimate := Append(msg_estimate, estimate);
        msg_justification := Append(msg_justification, justification);
        all_msg := all_msg \union {cur_msg_id};
        cur_msg_id := cur_msg_id + 1;
end macro;


macro init_validator(validator) begin
    make_message(validator, validators_initial_values[validator], {cur_msg_id});
    validator_init_done[validator] := 1;
end macro;


macro make_equivocating_messages(validator) begin
    equiv_msg_receivers[cur_msg_id] := get_equiv_receivers(validator);
    equivocating_msg := equivocating_msg \union {cur_msg_id};
    cur_subset := get_equivocation_subset_msg(validator);
    make_message(validator, binary_estimator(cur_subset), cur_subset);
end macro;


macro send_message(validator) begin
    if validator \notin byzantine_fault_nodes then
        make_message(validator, binary_estimator(validator_received_msg(validator)), validator_received_msg(validator));
    else
        make_equivocating_messages(validator);
    end if;
end macro;


fair process v \in validators begin
    Validate:
    while cur_msg_id <= message_ids do
        if validator_init_done[self] = 0 /\ self \notin byzantine_fault_nodes then
            init_validator(self);
        else
            send_message(self);
        end if;
    end while;
end process;
```

Figure 2: The CBC Casper message relay specified in PlusCal.

The assumption is that all the messages are sent and received successfully without time delay. The specification does not include message authentication, because it is assumed that the validators can verify the authenticity and source of messages. In this implementation, equivocating validators behave such that they take different subsets of all received messages, use these subsets to obtain the estimates, and send different messages to different validators.

The TLC model checker checks that eventually a clique will be found where all the non-Byzantine nodes can mutually see each other agreeing with a certain estimate in messages, and cannot see each other disagreeing. When this condition is met, finality is achieved, and the model checker should terminate without giving errors, as shown in Figure 3.


Model Checking Results

General

Start time:	Mon Aug 05 08:00:03 PDT 2019
End time:	Mon Aug 05 08:00:19 PDT 2019
TLC mode:	Breadth-first search
Last checkpoint time:	
Current status:	Not running
Errors detected:	No errors
Fingerprint collision probability:	calculated: 1.3E-11, observed: 8.4E-10

Statistics

State space progress (click column header for graph)

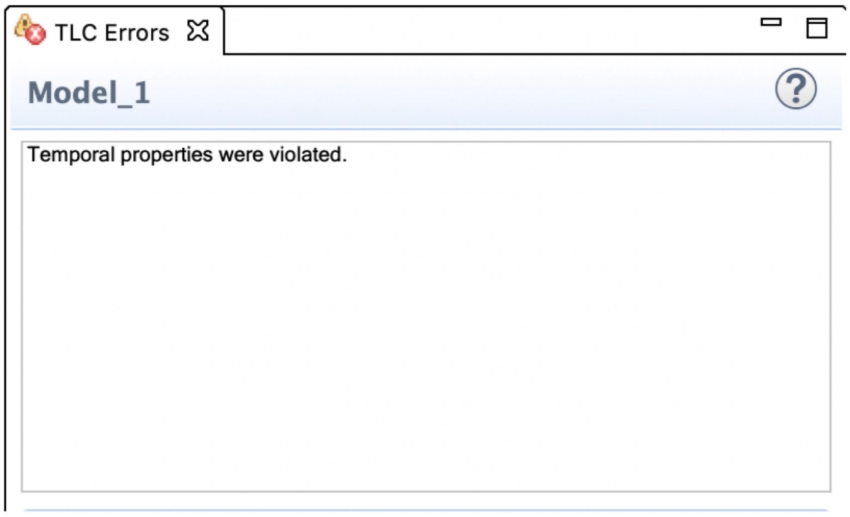
Time	Diameter	States Found	Distinct States	Queue Size
00:00:16	11	31711	18589	0
00:00:03	9	5776	5776	3842
00:00:00	0	1	1	1

Coverage at 2019-08-05 08:00:19

Module	Location	Count
cbccasper_...	line 358, col 12 to line 358, col 18	2187
cbccasper_...	line 358, col 21 to line 358, col 36	2187
cbccasper_...	line 358, col 39 to line 358, col 48	2187
cbccasper_...	line 358, col 51 to line 358, col 62	2187
cbccasper_...	line 359, col 12 to line 359, col 28	2187
cbccasper_...	line 359, col 31 to line 359, col 40	2187
cbccasper	line 359, col 43 to line 359, col 61	2187

Figure 3: TLC model checking results.

When finality cannot be reached, the model checker will detect that a temporal property has been violated, as shown in Figure 4.


TLC Errors

Model_1

Temporal properties were violated.

Figure 4: TLC errors.

The temporal property checks for the existence of a clique of validators with a total weight greater than:

$$\frac{\sum validator\ weights}{2}+\frac{fault\ tolerant\ threshold}{2}-(fault\ weight\ seen\ by\ current\ validator)$$

All the validators in the clique satisfy the following:

- **None of the validators are Byzantine-faulty.** They do not send equivocating messages.
- **All the validators can mutually see each other agreeing.** Each validator has exactly one latest message, and they have the same estimate.
- **None of the validators can mutually see each other disagreeing.** A validator has a latest message in the other's justification, but has a new latest message that doesn't have the same estimate as the latest message of the other validator.

In practice, the failure to achieve finality would mean the blockchain stops producing new blocks, since there is no guarantee that a transaction (i.e. an estimate) will not be reversed in the future.

Conclusion

We find that when the fault-tolerant threshold is set to greater than one-third of the total weight of all the validators, e-cliques cannot be formed. This means finality can be reached when the fault-tolerant threshold is less than one-third of the total weight. This is consistent with "The Byzantine Generals Problem," which states that the problem "is solvable if and only if more than two-thirds of the generals are loyal." Intuitively, for the protocol using the clique oracle, a higher fault-tolerant threshold would cause there to be too few validators to form a clique.

While the CBC Casper specification provides a proof of safety, it does not address liveness properties. For example, the CBC Casper blockchain protocol may encounter the problem in which no more blocks can be finalized. Further work is needed in specifying liveness, because finality is a liveness problem and is necessary for a switch to a PoS method. I found no liveness faults, but only tested binary consensus with a very small number of validators. Liveness faults may exist in more sophisticated instantiations of CBC Casper.

Some Thoughts on Formal Verification and TLA+

Developing an abstract mathematical model and systematically exploring the possible states is an interesting and important way to check the correctness of algorithms. However, I encountered the following challenges:

- **Few examples of implementation details of the TLA+ language and good practices for formal verification.** Therefore, writing specifications can involve a lot of trial and error.
- **Unhelpful error messages generated by the TLC model checker when something fails to compile.** The error messages are vague and do not pinpoint a specific location where the error occurs. In addition, the TLA+ toolbox is a Java application, so the error messages are often Java exceptions. Figuring out what's wrong with the TLA+ specification, given the Java exceptions, is difficult.
- **Limited documentation of formal verification methods.** Googling a question specific to TLA+ yields very few results. As of [date], there were only 39 questions on Stack Overflow with "TLA+" as a tag.

Thanks

Working at Trail of Bits as an intern this summer was an amazing experience for me. I learned a lot about distributed systems and formal verification and greatly enjoyed the topics. I am glad to have experienced working in security research, and I am motivated to explore more when I go to college.

By Trent Brunson Posted in Blockchain (https://blog.trailofbits.com/category/blockchain/), Internship Projects (https://blog.trailofbits.com/category/internship-projects/)