Evaluating BlockCypher's Microtransaction Confidence Level

By Dean Makovsky, Vignesh Kuppusamy, Joseph Tobin, Kevin Zhao

Motivation: BlockCypher offers a unique service that predicts the likelihood of a double spend for a given transaction by calculating a confidence interval of a transaction. BlockCypher claims that it is able accurately calculate the confidence interval of a microtransaction within an average of 8 seconds. This is significantly faster than the time to get a single confirmation from the bitcoin network (an expected time of 10 minutes). With the ability to confirm or deny transactions within ~8 seconds, the variety of the applications of bitcoin exponentially increases. For example, according to Josh Cincinnati, he once had to wait 10 minutes for his bitcoin transaction to confirm to get a coffee at a coffee shop accepting bitcoin. With this confidence interval, the merchant can be almost instantaneously informed about the reliability of a consumer's transaction, making bitcoin much more accessible and practical for brick and mortar shops.

Because of the great potential of the confidence interval to make bitcoin significantly more practical to businesses globally, the confidence interval must be thoroughly evaluated in order to ensure there are not ways to manipulate BlockCypher's network. Therefore, we are going to evaluate the strength of BlockCypher's confidence interval method by trying to double spending a transaction with a confidence interval of 98% or greater. If we can successfully double spend, we will have exposed a flaw in Blockypher's confidence interval and therefore can help them improve their algorithm. Otherwise, we will be able to serve as a validator of BlockCypher's confidence interval to improve vendor's confidence in BlockCypher's service.

Background:

- Bitcoin: "Bitcoin is a decentralized digital currency that enables instant payments….Bitcoin uses a peer-to-peer technology to operate with no central authority: transaction management and money issuance are carried out collectively by the network." (https://en.bitcoin.it/wiki/Main_Page).
- Confirmation: "After a transaction is broadcast to the Bitcoin network, it may be included in a block that is published to the network. When that happens it is said that the transaction has been mined at a depth of 1 block. With each subsequent block that is found, the number of blocks deep is increased by one. To be secure against double spending, a transaction should not be considered as **confirmed** until it is a certain number of blocks deep." (https://en.bitcoin.it/wiki/Confirmation).
- Double spending: According to the bitcoin wiki, double spending is defined as the "result of successfully spending some money more than once". There are multiple ways to execute a double spend. The five main methods of attack (according to the bitcoin wiki) are: Race attack, Finney attack, Vector76 attack, Brute force attack, and the >50 % attack. Because this project does not have the resources to run a miner, collude with a miner, or flood the bitcoin network with nodes, we focused on the race attack, as described in the classic paper "Two Bitcoins at the Price of One". (https://en.bitcoin.it/wiki/Double-spending).

- "Two Bitcoins at the Price of One" (Karame, Androulaki, Capkun) (https://eprint.iacr.org/2012/248.pdf) : academic paper written in 2011 revealing the weakness of the bitcoin network to double spends and several solutions to this problem. According to the paper, Bitcoin's solution to the issue as of 2011 was to encourage vendors to not wait for "transaction verification, as long as the transaction is not of high value". This solution was shown to be relatively ineffective as the paper calculated the overwhelmingly high probability of performing a successful double spend race attack with relatively little overhead.
    - Race attack: an attack where the attacker sends out multiple valid transactions to the network, one to a vendor and one back to himself/herself. The attack is successful in the transaction back to himself/herself is published while the vendor accepts the transaction upon zero confirmations.

  The authors produced three significantly more effective and novel solutions to the problem: "using a listening period", "inserting observers into the network", and "communicating double-spending alerts among peers".
    - "Using a listening period": a vendor receiving a zero-confirmation transaction should listen to the network for a listening period of a few seconds and monitor the network to ensure none of the received transactions are double-spends. The paper points out that there is still a chance this method can be circumvented if the attacker delays the transmission of the double spend such that the transaction to itself still has a chance of propagating through the network after the listening period closes. This can be minimized by extending the listening period. Additionally, the more connected the vendor is to the network, the less likely the double-spend transaction will arrive after the listening period.
    - "Inserting observers into the network": In addition to "using a listening period", a vendor can also prevent a double spend by inserting multiple nodes into the network. By having more nodes in the network, the vendor will be able to more quickly and easily detect double spends because it will be aware of more transactions. This also makes it significantly more difficult for the attacker to target specific nodes to propagate transactions to.
    - "Communicating double-spending alerts among peers": If a node currently receives a double-spend transaction, the node simply ignores the transaction. If a node receives a transaction with the same input as a transaction in its memory pool, that node can use the existing bitcoin client to send out an "alert" messaging notifying the network of the double spend attack. This would make it significantly harder for an attacker to strategically send double spend transactions to certain nodes because all nodes would make each other aware of the double spending.
- Confidence interval/level: A measure used by BlockCypher to accurately express their prediction of the likelihood of an attempted double spend succeeding against a transaction (using the same input as that transaction). With a confidence level of 98%, there is a 2% chance that an attempted double-spend will succeed. The level is based off of a model developed by BlockCypher based on their internal research and "Two

Bitcoins at the Price of One". According to BlockCypher, their model is based off of two factors: the "shape" and "behavior" of a transaction.

- Shape: time-invariant characteristics of a transaction. More specifically, factors such as the amount of inputs and outputs addresses, the amount of bitcoin transferred, the transaction fee, and the signature of the transaction. Generally, addresses with higher transaction fees and lower amounts of bitcoin transacted (between 2,000 satoshi and 4,000,000 satoshi) are less likely to be double spent.
- Behavior: time variant characteristics of a transaction, or network propagation. More specifically, how quickly the transaction travels through the network and how many nodes receive a transaction. Generally, the more nodes that receive a transaction and the more quickly a transaction travels through the network, the less likely the transaction is double spent.

BlockCypher analyzes transactions by being connected to 10-20% (a statistically significant amount) of the bitcoin network at any time so that they are sufficiently knowledgeable about the network.

Methodology: Before attempting to attack BlockCypher's confidence interval, we thoroughly researched the methods of attack and the theoretical structure of BlockCypher's confidence interval system. After reading their website (http://dev.BlockCypher.com/) and "Two Bitcoins at the Price of One" and gaining the permission of Josh Cincinatti (Bitcoin Evangelist), we determined our method of attack. Because of the limited resources of our project, we could not execute a Finney attack, Vector76 attack, Brute force attack, and the >50 % attack. Additionally, Josh Cincinnati informed us that "maleated transactions don't count". Therefore, our main (and only) method of attack was a "race attack". Because BlockCypher bases their confidence level factor on the shape and behavior of a transaction, we determined that we would have to send out transactions that would appear trustworthy in these aspects to BlockCypher while actually double spending the input.

To determine the time-time invariant characteristics of an equation, we sent out small transactions and looked at BlockCypher's live feed (https://live.BlockCypher.com/btc/). We found that BlockCypher had confidence in transactions with significantly higher miner fees. After looking through their API documentation, BlockCypher stated that it takes between 2,000 and 4,000,000 satoshis around 8 seconds (http://dev.BlockCypher.com/#microtransaction-api) to get 98% confidence and any larger transactions take around 20 seconds (http://blog.BlockCypher.com/?p=51).

For the time-variant characteristics, we had to develop a more strategic plan of attack. Using the information that their system in based on "Two Bitcoins at the Price of One" and the model's description in the API, we determined that Blockcyper has a system of nodes listening to the network. Specifically, they are "connected to anywhere between 10 and 20% of the peers" … "and we prioritize diverse connections in different locations to guarantee that these nodes are a representative sample" (http://blog.BlockCypher.com/?p=51). Because BlockCypher's nodes are so ubiquitous, we deduced that we did not have a chance of double spending unless we were connected to more than 20% of the network or we knew where BlockCypher's nodes were located so that we could avoid them. Because we do not possess

the temporal and monetary resources to creates nodes in more than 20% of the network, we determined that the best route of attack was to locate BlockCypher's servers and strategically send transactions to them.

BlockCypher's nodes' IP addresses are not publicly available, and therefore we had to devise a strategy to determine their addresses. With the addresses of these nodes, we could develop a strategy to send our double spend transactions such that they arrive within in the time window where the transaction to the network is accepted by BlockCypher, but the double spend transaction (which sends bitcoin back to ourselves) is accepted by the network. By having more knowledge of the network topology, we could also be able to determine the behavioral characteristics that BlockCypher looks for, and use that information to make our transactions appear more trustworthy. We attempted to find BlockCypher's nodes' locations through several means.

First, we attempted to graph the connectivity of the network. Although https://bitnodes.21.co/ has a nice list of bitcoin nodes in the network and an API, the API did not provide us with connectivity. Therefore, we had to create our own programs to traverse the bitcoin network to determine the network topology. We designed a program to traverse the bitcoin network by using *getAddr* to get all of the nodes a given node is connected to (see previous versions of mini_node.py). With this list, we would then recursively traverse the network by calling *getAddr* on all of the returned IP addresses until we traversed the entire network. This design failed due to the inaccuracies involved with *getAddr.* We assumed *getAddr* would return all of the nodes a given node is currently connected to. Rather, *getAddr* returns all of the nodes a given node has ever been connected to, including the ones that are no longer online or exist. Additionally, *getAddr* returns a randomly selected subset of this list (and not the entire list), deeming this method impractical. If we had been able to develop a map of network connectivity we could determine the location of BlockCypher's nodes by sending out double spend transactions and observing how they propagate (elaborated upon below). Additionally, we would be able to determine the latency time between nodes to get better temporal calculations so we can better design our transaction path.

Second, we tried to find the IP addresses of BlockCypher node's through DNS lookups, geolocation, and version numbers. Before a Bitcoin node interacts with another node, it exchanges the version of the code it's running on and a version string. This is useful because there are several different valid versions of code to run, and some nodes interact differently with other nodes with certain versions. According to Josh Cincinnati's presentation in class, BlockCypher overhauled the bitcoin core code to develop its own version tailored to BlockCypher's needs. For example, BlockCypher's nodes may have code that propagates double spends to other BlockCypher nodes so that the other nodes do not accept the transaction (while standard nodes simply reject double spends). Therefore, if we can find a certain amount of bitcoin nodes running a version that would indicate a version BlockCypher created, then we can determine the IP addresses of BlockCypher's nodes.

According to "Two Bitcoins at the Price of One", 3 nodes with high connectivity are recommended to connect with the network. Because the amount of nodes in the bitcoin network has remained relatively constant over time, we deduced that BlockCypher would use 3 or greater (for even more connectivity than deemed necessary) nodes to ensure they are well

connected to the network. We found the version strings of all bitcoin nodes in the network (see "set_nodes.py" and the actual list at "latest.json") and examined all of the sets of nodes with the same version with cardinalities between 3 and 7. We also looked at the geographic distribution of the nodes. We assumed that because BlockCypher wanted to be geographically well distributed, they would have the most nodes in the United States, but would also contain nodes in Europe and China (or a country near China). After examining all of the nodes and looking at their version number, location, and network, we deduced that potentially nodes running "Satoshi:0.10.4" or "Satoshi:1.1.1" were BlockCypher nodes because they fit the geographic distribution requirements and number of nodes to be expected. Because we had no confirming evidence, we did not test these nodes.

Additionally, we looked through the version of all of the other bitcoin nodes to determine if any would hint that BlockCypher was running them. Unfortunately, no nodes had some combination of "block", "cypher", or other exposing information in their version string. In our final effort to determine the address of BlockCypher nodes, we looked at the IP address of BlockCypher's website to try and determine if a BlockCypher node was located in the same servers as its website's servers. We found the servers were located on an EC2 instance in Ashburn, VA, but there was no revealing information about the rest of BlockCypher's servers.

Finally, we tried to find the nodes through programmatically sending transactions to determine which nodes are BlockCypher nodes. First, we had to rigorously develop code that could create custom transactions and send them to specific nodes (createTransaction.py and OPsPerLink.py). Second, we created a program to create and broadcast a legitimate transaction and then send a double spend transaction individually to every node in the bitcoin network (mini_node.py). Because BlockCypher nodes record double spend transactions and regular nodes do not, we can determine if each individual node is a BlockCypher node or not by determining if it propagated the double spend. More specifically, nothing will happen if the node that receives the transaction is a standard node, but BlockCypher nodes will store the transaction in their API. Consequently, we should be able to make an API requst to determine if the node is a Blockcypher node or not. This intensive program involved multithreading in order to decrease runtime (main.py).

Results: Ultimately, we were not able to double spend while getting a 98% confidence interval from BlockCypher despite our attempts. Therefore, BlockCypher still has not had a single failed microtransaction.

First, we unsuccessfully attempted to graph the bitcoin network. This failed due to difficulties in determining the connectivity of the network. The APIs available and the data given by bitcoin nodes was not satisfactory for our efforts

Second, we unsuccessfully attempted to determine the nodes of BlockCypher through looking at Bitcoin node information such as the version and geographic location. Despite extensive research, we were not able to determine which nodes were running BlockCypher's code. This attack was based on the assumption that the version of BlockCypher's nodes was unique to BlockCypher. Although most nodes usually state the code version they are running, nodes are not required to state the version they are using. Therefore, BlockCypher nodes could

be stating that they are running a version of the bitcoin code that would not be suspicious, such as Satoshi:0.11.2 (currently used by ~1,400 nodes).

Third, we unsuccessfully attempted to determine the nodes through programmatically sending transactions throughout the BlockCypher API. This was executed under the assumption that double spend transactions would be available in BlockCypher's API, but we were not able to find double spend transactions in their API. But seeing as this is a way to identify BlockCypher nodes and therefore potentially exploit the system, we do not think this information is available.

Consequently, BlockCypher's claims can be accepted with greater confidence and more vendors can have confidence in BlockCypher's services. With greater confidence in the quality of BlockCypher's services, vendors can use Bitcoin for a variety of new applications and take Bitcoin to the next level.

Future research could involve attempting to double spend using the other attacks listed above, which would require running a mining operation/colluding with a miner, or running multiple bitcoin nodes. Additionally, we could ask BlockCypher for their list of IP addresses and see if our attacks would work given the location of their nodes. This could be useful if there was a way to find the nodes that we did not discover or if the IP addresses were leaked somehow (for example, in a security leak by the network provider). Another idea for future work would be measuring the latency between nodes and generating a latency graph between all the nodes. This would then help us orchestrate race attacks if we knew the locations of the BlockCypher nodes.