Holo Fuel Model

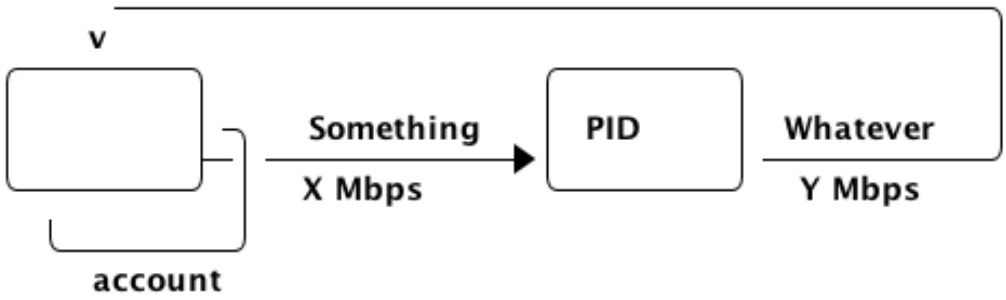
Perry Kundert

July 20, 2018

Contents

1 Holo Fuel

A value-stable wealth-backed cryptocurrency platform, where each unit is defined in terms of a basked of computational resources, operating in a powerful decentralized verification environment.



1.1 The Computational Resources Basket

One Holo Fuel (HOT) is defined as being able to purchase 1 month of Holo Hosting services for the front-end (ie. web API) portion of a typical dApp:

Amount	Units	Commodity	Description
1.00	Host	holo	Inclusion in the Holo system
0.10	TB	net	Internet bandwidth
0.25	GB	ram	Processor memory
0.10	TB	data	Persistent storage (DHT/DB/file)
0.25	Core	cpu	A processing core

This might be roughly equivalent to the 2018 price of (and actual utilization of) a small cloud hosting node (eg. a $5/month Droplet on Digital Ocean), with a premium for inclusion in the Holo system.

1.1.1 Holo Hosting Premium

A Holochain Distributed Application (dApp) hosted on Holo provides a valuable set of features, over and above simply hosting a typical web application on a set of cloud servers.

- ☐ Reliability. Few single points of failure.
- ☐ Backup. All DHT data is spread across many nodes.
- ☐ Scalability. Automatically scales to absorb increased load.

The value of Holo is substantial in terms of real costs to traditional app developers, and is a component of the basket of commodities defining the price of Holo Fuel. However, it's real monetary value will emerge over time, as the developer community comprehends it. Our pricing algoritm must be able to dig this Holo premium component out of the historical hosting prices, as a separate component.

1.1.2 Resource Price Stability

There are many detailed requirements for each of these commodities, which may be required for certain Holochain applications; CPU flags (eg. AVX-512, cache size, ...), RAM (GB/s bandwidth), HDD (time to first byte, random/sequential I/O bandwidth), Internet (bandwidth/latency to various Internet backbone routers).

The relative distribution of these features will change over time; RAM becomes faster, CPU cores more powerful. The definition of a typical unit of these commodities therefore changes; as Moore's law decreases the price, the specifications of the typical computer also improve, counterbalancing this inflationary trend.

For each metric, the price of service on the median Holo Host node will be used; 1/2 will be below (weaker, priced at a discount), 1/2 above (more powerful, priced at a premium). This will nullify the natural inflationary nature of Holo Fuel, if we simply defined it in terms of fixed 2018 computational resources.

1.2 Commodity Price Discovery

Value stabilization requires knowledge of the current prices of each commodity in the currency's valuation basket, ideally denominated in the currency itself. If these commodities are traded within the cryptocurrency implementation, then we can directly discover them on a distributed basis. If outside commodity prices are used, then each independent actor computing the control loop must either reach consensus on the price history (as collected from external sources, such as Distributed Oracles), or trust a separate module to do so. In Holo Fuel, we host the sale of Holo Host services to dApp owners, so we know the historical prices.

When a history of Holo Hosting service prices is available, Linear Regression can be used to discover the average fixed (Holo Hosting premium) and variable (CPU, ...) component costs included in the prices, and therefore the current commodity basket price.

1.2.1 Recovering Commodity Basket Costs

To illustrate price recovery, lets begin with simulated prices of a basket of commodities:

```
# To simulate initial pricing, lets start with an estimate of proportion of basket value represented
# by each amount of the basket's commodities.  Keep it simple; all are roughly equally weighted.
price_mean          = 1.00                  # target price: 1.00 Holo Fuel == 1 basket
price_sigma         = price_mean / 10       #  difference allowed; about +/- 10% of target
prices              = { k: ( price_sigma * numpy.random.randn() + price_mean ) / len( basket ) / basket[k]
      for k in basket }
[ [ "Commodity", "HOT$", "Per" ],
  None ] \
+ [ [ k, "%5.2f" % ( prices[k] ), "%s / mo." % ( commodities[k].units ) ]
    for k in basket ]
```

Commodity	HOT$	Per
holo	0.17	Host / mo.
net	1.98	TB / mo.
ram	0.76	GB / mo.
data	2.24	TB / mo.
cpu	0.76	Core / mo.

From this set of current commodity prices, we can compute the current price of the HOT currency's basket:

HOT$ Basket Price
$ 0.98

Once we have the currency's underlying commodity basket, lets simulate a sequence of trades of various amounts of these commodities. We will not know the exact costs of each commodity used to compute the price, or how much is the baseline Holo system premium.

```
amounts_mean        = 1.00
amounts_sigma       = 0.50
error_sigma         = 0.10 # +/- 10% variance in bids (error) vs. price
trades              = []
number              = 10000
for _ in range( number ):
    amounts         = { k: 1 if k == 'holo' else max( 0, basket[k] * ( amounts_sigma * numpy.random.randn() + amounts_mean )) fo
    price           = sum( amounts[k] * prices[k] for k in amounts )
    error           = price * error_sigma * numpy.random.randn()
    bid             = price + error
    trades.append( dict( bid = bid, price = price, error = error, amounts = amounts ))

[ [ "HOT$","calc/err", "dApp Requirements" ], None ] \
+ [ [
      "%5.2f" % t['bid'],
      "%5.2f%+5.2f" % ( t['price'], t['error'] ),
       ", ".join( "%5.4f %s %s" % ( v * basket[k], k, commodities[k].units ) for k,v in t['amounts'].items() ),
    ]
    for t in trades[:5] ]
```

HOT$	calc/err	dApp Requirements
1.12	1.13-0.01	1.0000 holo Host, 0.0109 net TB, 0.0412 ram GB, 0.0227 data TB, 0.0360 cpu Core
1.30	1.13+0.17	1.0000 holo Host, 0.0182 net TB, 0.0365 ram GB, 0.0187 data TB, 0.0206 cpu Core
0.72	0.85-0.13	1.0000 holo Host, 0.0132 net TB, 0.0074 ram GB, 0.0044 data TB, 0.0962 cpu Core
0.76	0.89-0.13	1.0000 holo Host, 0.0148 net TB, 0.0269 ram GB, 0.0124 data TB, 0.0202 cpu Core
0.77	0.92-0.15	1.0000 holo Host, 0.0128 net TB, 0.0701 ram GB, 0.0048 data TB, 0.0562 cpu Core

Lets see if we can recover the approximate Holo baseline and per-commodity costs from a sequence of trades. Create some trades of 1 x Holo + random amounts of commodities around the requirements of a typical Holo dApp, adjusted by a random amount (ie. 'holo' always equals 1 unit, so that all non-varying remainder is ascribed to the "baseline" Holo Hosting premium).

Compute a linear regression over the trades, to try to recover an estimate of the prices.

```
items               = [ [ t['amounts'][k] for k in basket ] for t in trades ]
bids                = [ t['bid'] for t in trades ]
regression          = linear_model.LinearRegression( fit_intercept=False, normalize=False )
regression.fit( items, bids )
select              = { k: [ int( k == k2 ) for k2 in basket ] for k in basket }
predict             = { k: regression.predict( select[k] ) for k in basket }

#   + """\
#: Score(R^2): %r
#"""  % ( regression.score( items, bids ) )
[ [ "Commodity",  "Predicted", "Actual", "Error",
   # "selected"
  ],
  None ] \
+ [ [ k,
      "%5.2f" % ( predict[k] ),
      "%5.2f" % ( prices[k] ),
      "%+5.3f%%" % (( predict[k] - prices[k] ) * 100 / prices[k] ),
      # select[k]
    ]
    for k in basket ]
```

Commodity	Predicted	Actual	Error
holo	0.17	0.17	-0.370%
net	1.96	1.98	-0.979%
ram	0.77	0.76	+0.887%
data	2.25	2.24	+0.528%
cpu	0.76	0.76	-0.346%

Finally, we can estimate the current HOT$ basket price from the recovered commodity prices:

HOT$ Basket Price	Error
$ 0.98	-0.038%

We should be able to recover the underlying commodity prices, and hence the basket price within a high degree of certainty, even in the face of relatively large differences in the mix of prices paid for hosting.