Holo Fuel Model

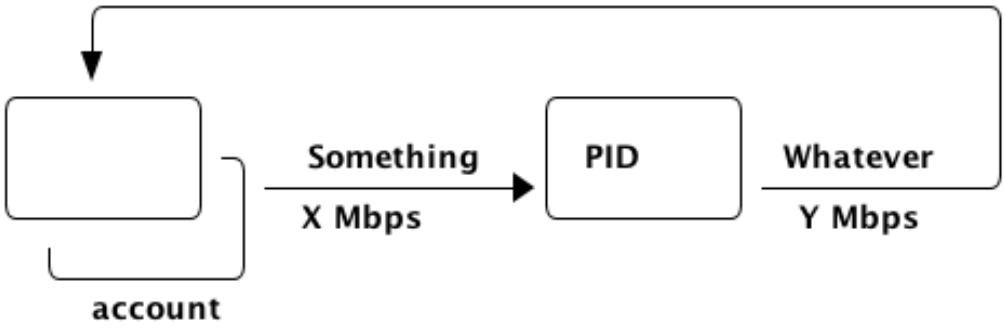
Perry Kundert

July 26, 2018

Contents

1 Holo Fuel Valuation

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, databases, etc.) portion of a typical dApp.

This might be roughly equivalent to the 2018 price of (and actual utilization of) a small cloud hosting setup (eg. several $5/month Droplet on Digital Ocean at partial utilization hosting front-ends, DBs, backups, etc.), plus ancillary hosting services (represented by a premium for inclusion in the Holo system).

If the minimal Hosting costs for a small Web app is estimated at USD$100/mo., and comprises 5 cloud hosting nodes and anciliary services for the various aspects of the system. An equivalent Holo Host based system would have similar CPU and storage requirements overall, but a greater redundancy (say, 5 x, so all DHTs spread across 25 Holo Hosts).

Thus, the basket of commodities defining the value of USD$100 ~ 100 Holo Fuel could be defined as:

Commodity	Amount	Units	Weight	Description
holo	25.00	Host	60.000%	Inclusion in the Holo system
net	0.50	TB	20.000%	Internet bandwidth
ram	1.00	GB	5.000%	Processor memory
data	0.25	TB	10.000%	Persistent storage (DHT/DB/file)
cpu	1.00	Core	5.000%	A processing core

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. These services must usually be either purchased, or architected by hand and distributed across multiple cloud hosting nodes for redundancy.

- ☐ 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. A prototypical minimal dApp owner could select 100 Holo Fuel worth of these resources, eg. 25x Holo Hosts, .05 TB data, 1.5 cpu, etc. as appropriate for their specific application's needs.

This Hosting selection wouldn't actually be a manual procedure; testing would indicate the kind of loads to expect for a given amount and type of user activity, and a calculator would estimate the various resource utilization and costs. At run time, the credit extended to the dApp owner (calculated from prior history of Hosting receipt payments) would set the maximum outstanding Hosting receipts allowed; the dApp deployment would auto-scale out to qualified Hosts in various tranches as required; candidate Hosts (hoping to generate Hosting receipts) would auto-install the application as it reached its limits of various resource utilization metrics across its current stable of Hosts.

```
def rnd_std_dst( sigma, mean=0 ):
    """ """
    return sigma * np.random.randn() + mean

# To simulate initial pricing, lets start with an estimate of proportion of basket value represented
# by each amount of the basket's commodities.  Prices of each of these commodities is free to float
# in a real market, but we'll start with some pre-determined "weights"; indicating that the amount
# of the specified commodity holds a greater or lesser proportion of the basket's value.
# Regardless, 100 Holo Fuel is guaranteed to buy the entire basket.
prices                  = {}
for k in basket:
    price_mean          = basket_target * weight[k] / basket[k] # target price: 1 Holo Fuel == 1 basket / basket_target
    price_sigma         = price_mean / 10 #  difference allowed; about +/- 10% of target
    prices[k]           = rnd_std_dst( price_sigma, price_mean )

[ [ "Commodity", "Price", "Per", "Per" ],
  None ] \
+ [ [ k, "%5.2f" % ( prices[k] ), commodities[k].units, 'mo.' ]
    for k in basket ]
```

Commodity	Price	Per	Per
holo	2.51	Host	mo.
net	39.36	TB	mo.
ram	5.21	GB	mo.
data	45.75	TB	mo.
cpu	4.95	Core	mo.

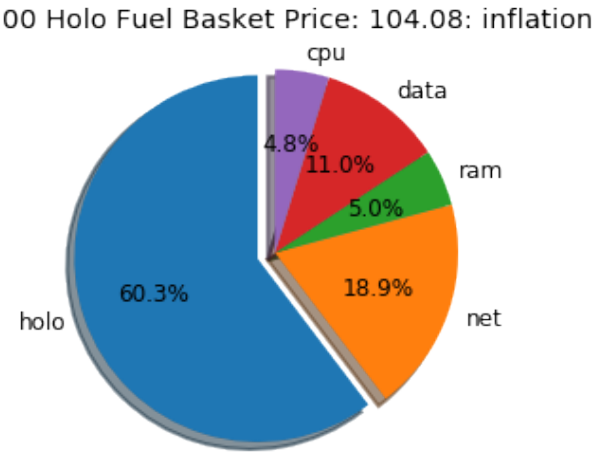
From this set of current assumed commodity prices, we can compute the current price of the Holo Fuel currency's basket:

```
basket_price            = sum( basket[k] * prices[k] for k in basket )
[ [ "Holo Fuel Basket Price" ],
  None,
  [ "$%5.2f / %.2f" % ( basket_price, basket_target ) ] ]
```

Holo Fuel Basket Price
$104.08 / 100.00

If the current price of this basket is >100, then we are experiencing commodity price inflation; if <100, price deflation. Feedback control loops will act to bring the price back to 100 Holo Fuel per basket.

```
labels                  = [ k for k in basket ]
sizes                   = [ basket[k] * prices[k] for k in basket ]
explode                 = [ .1 if k == 'holo' else 0 for k in basket ]
# with plt.xkcd():
fig1,ax1       = plt.subplots()
ax1.pie( sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90 )
ax1.axis( 'equal' ) # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title( "%6.2f Holo Fuel Basket Price: %6.2f: %sflation" % (
    basket_target, basket_price, "in" if basket_price > basket_target else "de" ))
plt.show()
```



1.2.2 Holo Hosting Receipts

Once we have the currency's underlying commodity basket, lets simulate a sequence of trades of various amounts of these commodities. In the Holo system, this is represented by Hosts issuing receipts for services to dApp owners.

Each Hosting receipt will be for a single Holo Host, not for the entire dApp; the sum of all Holo Hosting receipts issued to the dApp owner for our archetypical small dApp would sum to approximately 100 Holo Fuel per month.

We will not know the exact costs of each commodity used to compute the price, or how much is the baseline Holo system premium. However, it will be dependant on the capability of the Host

(stronger hosts can charge more, for hosting more specialized dApps), and the amount of various services used.

So, lets issue a bunch of small Holo Hosting receipts, each for approximately 1/25th of the total Holo Hosting load (since our small dApp is spread across 25 Holo Hosts).

```
amounts_mean            = 1.00
amounts_sigma           = 0.5
error_sigma = 0.10 # +/- 10% variance in bids (error) vs. price
trades                  = []
number                  = 10000
for _ in range( number ):
    # Each dApp consumes a random standard distribution of the target amount of each commodity
    amounts             = { k: 1 if k == 'holo'
       else max( 0, basket[k] * rnd_std_dst( amounts_sigma, amounts_mean ) / basket['holo'] )
    for k in basket }
    price               = sum( amounts[k] * prices[k] for k in amounts )
    error               = price * rnd_std_dst( error_sigma )
    bid                 = price + error
    trades.append( dict( bid = bid, price = price, error = error, amounts = amounts ))

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

Fuel	calc/err	dApp Requirements
4.92	4.46+0.45	1.0000 holo Host, 0.0261 net TB, 0.0656 ram GB, 0.0062 data TB, 0.0601 cpu Core
5.02	4.64+0.38	1.0000 holo Host, 0.0264 net TB, 0.0552 ram GB, 0.0125 data TB, 0.0456 cpu Core
3.04	3.40-0.36	1.0000 holo Host, 0.0078 net TB, 0.0534 ram GB, 0.0024 data TB, 0.0386 cpu Core
4.63	4.14+0.49	1.0000 holo Host, 0.0170 net TB, 0.0641 ram GB, 0.0074 data TB, 0.0582 cpu Core
3.58	4.32-0.74	1.0000 holo Host, 0.0230 net TB, 0.0464 ram GB, 0.0088 data TB, 0.0534 cpu Core
...		

1.2.3 Recovery of Commodity Valuations

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): ", "%.9r" % ( regression.score( items, bids )), '', '' ],
  None ] \
+ [ [ "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 ]
```

Score(R^2):	0.5548147		
Commodity	Predicted	Actual	Error
holo	2.50	2.51	-0.476%
net	39.48	39.36	+0.286%
ram	5.51	5.21	+5.663%
data	45.85	45.75	+0.233%
cpu	4.81	4.95	-3.015%

1.2.4 Commodity Basket Valuation

Finally, we can estimate the current Holo Fuel basket price from the recovered commodity prices.

```
basket_predict          = sum( basket[k] * predict[k]  for k in basket )
[ [ "Holo Fuel Price Recovered", "vs. Actual", "Error" ], None,
  [ "$%5.2f / %.2f" % ( basket_predict, basket_target ),
    "%5.2f" % ( basket_price ),
    "%+5.3f%%" % (( basket_predict - basket_price ) * 100 / basket_price ),
    ]]
```

Holo Fuel Price Recovered	vs. Actual	Error
$104.01 / 100.00	104.08	-0.067%

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

2 Holo Fuel Value Stabilization

Price discovery gives us the tools we need to detect {in,de}flation as it occurs. Control of liquid credit available in the marketplace gives us the levers we need to eliminate it.

Traditional Fiat currencies control the issuance of liquidity by influencing the commercial banks to create more or less money through lending, and to increase/reduce liquidity through the net issuance/retirement of debt (which creates/destroys the principal money).

2.1 Wealth Monetization

In a wealth-backed currency, credit is created by the attachment of wealth to the monetary system, and credit lines of varying proportions being extended against the value of that wealth.

Depending on savings rates, monetary velocity, public sentiment etc., the amount of credit available to actually be spent varies. Since this available liquid credit is split between possible expenditures in priority order, the amount available to spend on each specific commodity therefore varies, driving the market price up and down.

If reliable indicators of both the liquid credit supply within, and the quality and amount of wealth attached, exist within the system itself then control systems can be executed within the system to automatically control the monetization of wealth to achieve credit unit value equilibrium – value-stability.

Each reserve of wealth provided different flows and indicators, and can support value-stability in different ways.

2.1.1 Reserve Accounts

The Reserve Accounts provide the interface between external currencies (eg. USD$, HOT ERC20 Tokens) and Holo Fuel.

Deposits to the reserve creates Holo Fuel credit limit (debt) at a current rate of exchange (TBD; eg. market rate + premium/discount). The corresponding Holo Fuel credits created are deposited to the recipient's account.

If Holo Fuel inflation occurs within the system, credit must be withdrawn. One way to accomplish this is to discourage creation of Holo Fuel (and encourage the redemption of Holo Fuel), by increasing the exchange rate. The inverse (lowering exchange rate) would result in more Holo Fuel creation (less redemption), reducing the Holo Fuel available, and thus reduce deflation.

The Reserve Accounts can respond very quickly, inducing Holo Hosts with Holo Fuel balances to quickly convert them out to other currencies when exchange rates rise. Inversely, reducing rates would release waiting dApp owners to purchase more Holo Fuel for hosting their dApps, deploying it into the economy to address deflation (increasing computational commodity prices).

A PD (Proportional Differential) control might be ideal for this. This type of control responds quickly both to direct errors (things being the wrong price), but most importantly to changes in the 2nd derivative (changes in rate of rate of change); eg. things getting more/less expensive at an increasing rate.

By eliminating the I (Integral) component of the PID loop, it does **not** slowly build up a systematic output bias; it simply adjusts the instantaneous premium/discount added to the current market exchange rate (eg. the HOT ERC20 market), to arrive at the Reserve Account exchange rate. When inflation/deflation disappears, then the Reserve Account will have the same exchange rate as the market.

Beginning with a set of reserves:

```
reserve_t               = collections.namedtuple(
    'Reserve', [
'rate',     # Exchange rate used for these funds
'amount',   # The total value of the amount executed at .rate
     ] )            #   and the resultant credit in Holo Fuel == amount * rate

reserve                 = {
    'EUR':          [],     # LIFO stack of reserves available
    'USD':          [ reserve_t( .0004, 200 ), reserve_t( .0005, 250 ) ], # 1,000,000 Holo Fuel
    'HOT ERC20':    [ reserve_t( 1, 1000000 ) ], # 1,000,000 Holo Fuel
}

def reserves( reserve ):
    return [ [ "Currency", "Rate avg.", "Reserves", "Holo Fuel Credits", ], None, ] \
   + [ [ c, "%8.6f" % ( sum( r.amount * r.rate for r in reserve[c] )
       / ( sum( r.amount for r in reserve[c] ) or 1 ) ),
 "%8.2f" % sum( r.amount for r in reserve[c] ),
 "%8.2f" % sum( r.amount / r.rate for r in reserve[c] ) ]
       for c in reserve ] \
   + [ None,
       [ '', '', '', sum( sum( r.amount / r.rate for r in reserve[c] ) for c in reserve ) ]]

summary                 = reserves( reserve )
summary # summary[-1][-1] is the total amount of reserves credit available, in Holo Fuel
```

Currency	Rate avg.	Reserves	Holo Fuel Credits
HOT ERC20	1.000000	1000000.00	1000000.00
USD	0.000456	450.00	1000000.00
EUR	0.000000	0.00	0.00
			2000000.0

As a simple proxy for price stability, lets assume that we strive to maintain a certain stock of Holo Fuel credits in the system for it to be at equilibrium. We'll randomly do exchanges of Holo Fuel out through exchanges at a randomly varying rate (also varied by the rate premium/discount), and purchases of Holo Fuel through exchanges at a rate proportional to the premium/discount.

```
t_last                  = -1
for t in range( 1000 ):
    dt                  = t - t_last
```