```
--------------------------------------------------------------------------------
----
KingCobra - A Research Software for Distributed Request Service on Cloud with Arbiters

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

#--------------------------------------------------------------------------------
------------------
#Copyleft (Copyright+):
#Srinivasan Kannan
#(also known as: Ka.Shrinivaasan, Shrinivas Kannan)
#Ph: 9791499106, 9003082186
#Krishna iResearch Open Source Products Profiles:
#http://sourceforge.net/users/ka_shrinivaasan,
#https://github.com/shrinivaasanka,
#https://www.openhub.net/accounts/ka_shrinivaasan
#Personal website(research): https://sites.google.com/site/kuja27/
#emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
#kashrinivaasan@live.com
#--------------------------------------------------------------------------------
------------------
********************************************************************************
***/

Theoretical Interludes, Design Notes and ToDo (long term with no deadline)
--------------------------------------------------------------------------

[This is a major research oriented subsystem of NeuronRain and inspired by COBRA
project (done by the author in his BE (1995-1999) along with few other esteemed
classmates.
http://sourceforge.net/projects/acadpdrafts/files/Excerpts_Of_PSG_BE_FinalProject_COBR
A_done_in_1999.pdf/download.)]

[KingCobra though a misnomer is expanded as ClOud With ARBiters MimicKING containing
the anagram]

1. (THEORY) There is a cloud of nodes which execute a set of services from randomly
created clients.

2. (THEORY) This cloud could be on iCloud (AsFer+USBmd+VIRGO) platform or any other
opensource cloud platforms like Hadoop Cluster.

3. (THEORY) The Clients are publishers of Service requests which are of many types -
miscellaneous types of Service that could be dynamically added through other kernel
modules and invoked through a switch-case or embedded in function itself. Identified
by unique id(s) for different types of services (for example Problem reports,
Suggestions etc.,)

4. (THEORY) The Services on the Cloud are Subscribers to these requests of specific
```

type. Thus this is the conventional publisher-subscriber model.

5. (THEORY) The requests flow through cloud using a workqueue (which could be a lowlevel Linux workqueue or VIRGO queue or some other queuing middleware software like ActiveMQ). The publishers enqueue and Subscribers dequeue the requests.

6. (THEORY) The difference is that the Cloud has nodes that "deceive" or "corrupt".

7. (THEORY) Service requests - are published by the clients in the need of a service which could be defined by markup file. These requests are scheduled and routed by the middleware to competent authority which services it (with or without timeframe) and replies to the client.

8. (THEORY) Problem reports - are published by clients which are "dissatisfied" by the quality of service by the cloud. These are analyzed by "arbiters" in the cloud which find the faulting node(s) and take action. This allows manual intervention but minimizes it.

9. (THEORY) Suggestions - are enhancement requests sent by clients and require manual intervention.

10. (THEORY) Cloud nodes have a Quality of Service metric calculated by a model.

11. (THEORY) The cloud has a reporting structure of nodes - either as a graph or tree. The graph is dynamically reorganized by weighting the Quality of Service of each node.

12. (THEORY) The difficult part of the above is using Arbiters to find "faulty" nodes based on problem reports from clients.

13. (THEORY) Brewer's CAP conjecture proved by [GilbertLynch] as a theorem (still debated) states that only 2 of the 3 (Consistency of data, Avaliability of data and Partition tolerance when some nodes or messages are lost) can be guaranteed and not all 3 are simultaneously achievable.

14. (THEORY) CAP theorem does not seem to apply to the above faulty scenario with corrupt nodes under Consistency or Availablity or Partition Tolerance. This is because a corrupt node can have any 2 of the 3 - it can give consistent data, is available with success response or can make the cloud work with missing data in partition tolerance but yet can "corrupt" the cloud. Probably this needs to be defined as a new attribute called Integrity.

15. (THEORY) As "corruption" is more conspicuous with monetary element, if above services are "charged" with a logical currency (e.g. bitcoin), then corruption in cloud is defineable approximately as (but not limited to)- "Undue favour or harm meted out to a client not commensurate with the charge for the service (or) unreasonable extra logical currency demanded to execute the service of same quality (or) deliberate obstruction of justice to a client with malevolent and unholy collusion with other cloud nodes with feigned CAP".

16. (THEORY) Identifying criminal nodes as in (15) above seems to be beyond the ambit of CAP. Thus CAP with Integrity further places a theoretical limit on "pure" cloud. If Integrity is viewed as a Byzantine problem with faulty or corrupt processes in a distributed system, and if resilience factor is rf (expected number of faulty nodes), then most algorithms can ensure a "working" cloud only if resilience is ~30% or less (3*rf+1) of the total number of cloud nodes. Probably this could apply to Integrity also that places a limit of 30% on "corrupt nodes" for the Cloud to work with sanity. Translating this to a Governance problem, a corruption-free administration is achievable with a maximum limit of 30% "corrupt" elements.

17. (THEORY-ONGOING) Analytics on the Problem reports sent to the cloud queue give a pattern of corrupt nodes. Intrinsic Merit ranking with Citation graph maxflow

considering cloud as a flow network where a node positively or negatively cites or "opines" about a node, as mentioned in http://arxiv.org/abs/1006.4458 (author's Master's thesis) and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf(published by the author during PhD) give a p2p ranking of cloud nodes that can be used for analysis though may not be reliable fully. AsFer has bigdata analytics functionality that fits well to this point to analyse the problem reports with machine learning algorithms and set the key-value pairs that are read by VIRGO kernel_analytics module and exported kernelwide. The persisted REQUEST_REPLY.queue with the logged request-reply IPs and timestamps can be mined with AsFer bigdata capability (e.g. Spark)

18. (THEORY) Policing the cloud nodes with arbiters - This seems to be limited by CAP theorem and Integrity as above. Also this is reducible to perfect inference problem in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt and drafts in https://sites.google.com/site/kuja27/

19. (THEORY) Brooks-Iyengar algorithm for sensors in all cloud nodes is an improved Byazantine Fault Tolerant algorithm.

20. (THEORY) BitCoin is a Byzantine Fault Tolerant protocol.

21. (THEORY) Byzantine Fault Tolerance in Clouds is described in http://www.computer.org/csdl/proceedings/cloud/2011/4460/00/4460a444-abs.html, http://www.eurecom.fr/~vukolic/ByzantineEmpire.pdf which is more on Cloud of Clouds - Intercloud with cloud nodes that have malicious or corrupt software. Most of the key-value(get/set) implementations do not have byzantine nodes (for example CAP without Byzantine nodes in Amazon Dynamo: http://www.eurecom.fr/~michiard/teaching/slides/clouds/cap-dynamo.pdf)

22. (THEORY) Related to point 18 - The problem of fact finding or fault finding using a cloud police has the same limitation as the "perfect inference" described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt. "Money trail" involving the suspect node in point 28 is important to conclude something about the corruption. In real world tracking money trail is a daunting task. In cloud that abides by CAP, missing messages on trail can prevent reaching a conclusion thereby creating benefit-of-doubt. Also fixing exact value for a transaction that involves MAC currency message is undecidable - a normative economics problem can never be solved by exact theoretical computer science.

23. (THEORY) Reference article on cloud BFT for Byzantine, Corrupt brokers - Byzantine Fault-Tolerant Publish/Subscribe: A Cloud Computing Infrastructure (www.ux.uis.no/~meling/papers/2012-bftps-srdsw.pdf)

---
24. KingCobra messaging request-response design - options

24a. Implementing a message subscription model in kernelspace where clients publish the message that is queued-in to subscribers' queue (Topic like implementation - use of ActiveMQ C implementation if available).

24b. (DONE-minimal implementation) At present a minimum kernelspace messaging system that queues remote request and handles through workqueue handler is in place. This responds to the client once the Kingcobra servicerequest function finishes processing the request(reply_to_publisher() in KingCobra driver). Unlike the usual messaging server, in which client publishes messages of a particular type that are listened to by interested clients, one option is to continue the status-quo of KingCobra as a peer-to-peer messaging system. Thus every VIRGO node is both a kernelspace messaging client and server that can both publish and listen. Every message in the cloud can

have a universally unique id assigned by a timestamp server - https://tools.ietf.org/html/rfc4122 (similar to bitcoin protocol) so that each message floating in the cloud is unique across the cloud (or) no two messages on the VIRGO cloud are same. The recipient node executing kingcobra_servicerequest_kernelspace() parses the unique-id (example naive unique-id is <ip-address:port>#localtimestampofmachine which is a simplified version of RFC4122) from the incoming remote request and responds to the remote client through kernel socket connection that gets queued-in the remote client and handled similar to incoming remote request. To differentiate request and response-for-request response messages are padded with a string "REPLY:<unique-id-of-message>" and requests are padded with "REQUEST:<unique-id-of-message>". This is more or less similar to TCP flow-control with SEQ numbers but state-less like UDP. Simple analogy is post-office protocol with reference numbers for each mail and its reply. Thus there are chronologically two queues: (1) queue at the remote VIRGO cloud service node for request (2) queue at the remote client for response to the request sent in (1). Thus any cloudnode can have two types of messages - REQUEST and REPLY. Following schematic diagram has been implemented so far.

24c.(DONE) KingCobra - VIRGO queue - VIRGO cpupooling , mempooling and queue service drivers interaction schematic diagram:

```
        KingCobraClient ==========><REQUEST:id>==================> VIRGO cpupooling service =====> VIRGO Queue ============> KingCobraService
                ||                                                                                                                                  ||
                ||                                                                                                                                  ||
                 <================= VIRGO Queue <====== VIRGO cpupooling service ==== <REPLY:id>================================== V

        KingCobraClient ==========><REQUEST:id>==================> VIRGO mempooling service =====> VIRGO Queue ============> KingCobraService
                ||                                                                                                                                  ||
                ||                                                                                                                                  ||
                 <================= VIRGO Queue <====== VIRGO mempooling service ==== <REPLY:id>================================== V

        KingCobraClient ==========><REQUEST:id>==================> VIRGO Queue service ================================> KingCobraService
                ||                                                                                                                                  ||
                ||                                                                                                                                  ||
                 <================= VIRGO Queue service ============================ <REPLY:id>================================== V
```

24d. (ONGOING) kingcobra_servicerequest_kernelspace() distinguishes the "REQUEST" and "REPLY" and optionally persists them to corresponding on-disk filesystem. Thus a disk persistence for the queued messages can either be implemented in 1) VIRGO queue driver 2) workqueue.c (kernel itself needs a rewrite (or) 3) KingCobra driver. Option (2) is difficult in the sense that it could impact the kernel as-a-whole whereas 1) and 3) are modularized. At present Option 3 persistence within KingCobra driver has been implemented.

24e.Above option 24b implements a simple p2p queue messaging in kernel. To get a Topic-like behaviour in VIRGO queue might be difficult as queue_work() kernel function has to be repeatedly invoked for the same work_struct on multiple queues which are subscribers of that message in AMQ protocol. Moreover creating a queue at runtime on need basis looks difficult in kernel which is usually done through some CLI or GUI interface in ActiveMQ and other messaging servers.

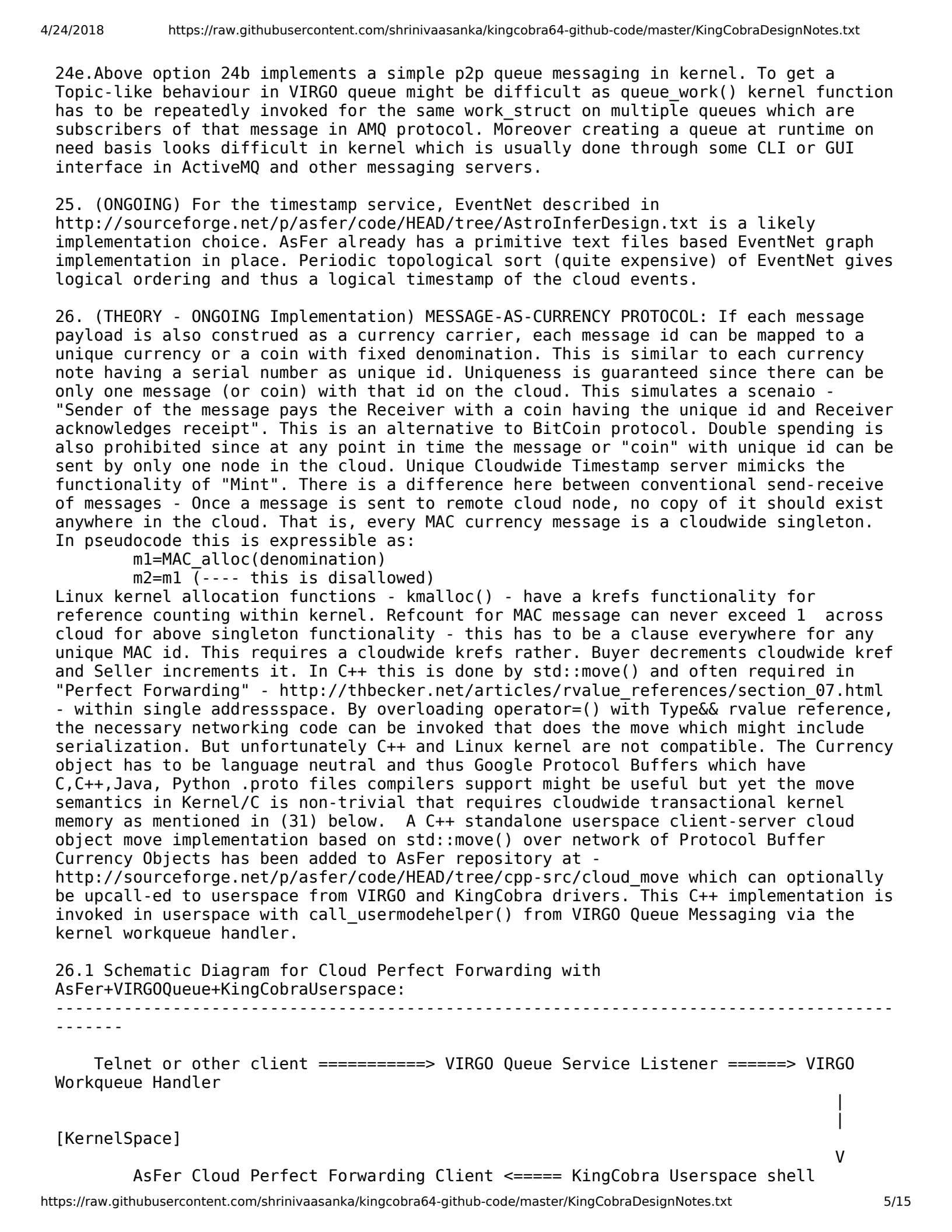
25. (ONGOING) For the timestamp service, EventNet described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt is a likely implementation choice. AsFer already has a primitive text files based EventNet graph implementation in place. Periodic topological sort (quite expensive) of EventNet gives logical ordering and thus a logical timestamp of the cloud events.

26. (THEORY - ONGOING Implementation) MESSAGE-AS-CURRENCY PROTOCOL: If each message payload is also construed as a currency carrier, each message id can be mapped to a unique currency or a coin with fixed denomination. This is similar to each currency note having a serial number as unique id. Uniqueness is guaranteed since there can be only one message (or coin) with that id on the cloud. This simulates a scenaio - "Sender of the message pays the Receiver with a coin having the unique id and Receiver acknowledges receipt". This is an alternative to BitCoin protocol. Double spending is also prohibited since at any point in time the message or "coin" with unique id can be sent by only one node in the cloud. Unique Cloudwide Timestamp server mimicks the functionality of "Mint". There is a difference here between conventional send-receive of messages - Once a message is sent to remote cloud node, no copy of it should exist anywhere in the cloud. That is, every MAC currency message is a cloudwide singleton. In pseudocode this is expressible as:

```
        m1=MAC_alloc(denomination)
        m2=m1 (---- this is disallowed)
```

Linux kernel allocation functions - kmalloc() - have a krefs functionality for reference counting within kernel. Refcount for MAC message can never exceed 1 across cloud for above singleton functionality - this has to be a clause everywhere for any unique MAC id. This requires a cloudwide krefs rather. Buyer decrements cloudwide kref and Seller increments it. In C++ this is done by std::move() and often required in "Perfect Forwarding" - http://thbecker.net/articles/rvalue_references/section_07.html - within single addressspace. By overloading operator=() with Type&& rvalue reference, the necessary networking code can be invoked that does the move which might include serialization. But unfortunately C++ and Linux kernel are not compatible. The Currency object has to be language neutral and thus Google Protocol Buffers which have C,C++,Java, Python .proto files compilers support might be useful but yet the move semantics in Kernel/C is non-trivial that requires cloudwide transactional kernel memory as mentioned in (31) below. A C++ standalone userspace client-server cloud object move implementation based on std::move() over network of Protocol Buffer Currency Objects has been added to AsFer repository at - http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/cloud_move which can optionally be upcall-ed to userspace from VIRGO and KingCobra drivers. This C++ implementation is invoked in userspace with call_usermodehelper() from VIRGO Queue Messaging via the kernel workqueue handler.

26.1 Schematic Diagram for Cloud Perfect Forwarding with AsFer+VIRGOQueue+KingCobraUserspace:



```
script(call_usermodehelper)
                        |                                                   |
                        |                                                   |
  Virtual Currency      |                                                   |
[UserSpace]
                        V                                                   V
        AsFer Cloud Perfect Forwarding Server <=================================/
```

References:

26.2 An example distributed transactional memory implementation in cloud - http://infinispan.org/tutorials/simple/tx/ and http://www.cloudtm.eu/ - these are in userspace cloud (C++ and Java) and may not have cloud move functionality - move has to be simulated in a transaction: replicate data, delete in one endpoint and create in other endpoint

27. (THEORY) SIMULATING A VIRTUAL ECONOMY with above MAC protocol (Message-as-currency): If each message sent is considered as "money" element and cloud nodes and clients are the consumers and producers of "electronic money", the timestamp "Mint" becomes a virtual Federal Reserve or Central Bank that controls the "electronic money" circulation in the cloud. Infact any REPLY messages could be mapped to a Service a client derives by s(p)ending the REQUEST "money message". Thus value(REQUEST) should equal value(REPLY) where value() is a function that measures the value of a money denomination and the value of goods and/or services for that money. For example Rs.10000 or $10000 has no meaning if it doesn't translate into a value (analogy: erstwhile Gold Standard).When the value() function gets skewed phenomena like Inflation arise. Thus above model could also have a notion of value() and "electronic money inflation". Thus any "message money" with a unique id assigned by the cloud unique id(or logical timestamp) server can exist at most in only one node in the cloud. Money trail can be implemented by prefixing a header to the incoming message money in each cloud node that receives the money which traces the "path" taken. Cloud has to implement some Byzantine Fault Tolerant protocol. The value() function to some extent can measure the "deceit" as above. When a Buyer and Seller's value() functions are at loggerheads then that is starting point of "cloud corruption" at either side and might be an undecidable problem.

28. (THEORY) TRADING WITH ABOVE KINGCOBRA MAC protocol - somewhat oversimplified:

```
                                --------------------
                                |Unique MAC id MINT|
                                --------------------
                                        ||
        -----money trail-----------------
        |
        V
        ....
        Buyer   ======= sends MAC message (REQUEST id) =======> Seller (stores the MAC
in local cash reserve and prepends money trail)
        ||                                                          ||
        <============ sends the goods and services (REPLY id) ===
```

In the above schematic, money with unique id in cloud reaches a buyer after many buyer-seller transitions called "money trail". The MAC currency is prefixed by each node to create a chain. Buyer then sends a request to the seller through MAC virtual currency and seller replies with goods and services. Seller prepends the money trail chain. When a transaction occurs the whole cloud need not be notified about it except only buyer and seller. MAC Mint could create a bulk of money denominations and circulate them in cloud economy.

References:

```
-----------
28.1 Price fixing for items in Buyer-Seller-Trader networks - Trading Networks -
Market Equilibrium and Walrasian Model of Price fixing -
http://www.cs.cornell.edu/~eva/traders.pdf
28.2 Algoithmic Game Theory - Market Equilibrium for Price - Equilibrium is a
strategic standoff - both players can't better their own present by changing
strategies e.g Buyer-Sellers are market players and equilibrium price is the one where
both buyer and seller can't gain by varying it -
http://www.cis.upenn.edu/~mkearns/nips02tutorial/nips.pdf. Buyer-Seller payoff matrix
picturises the bargaining problem.
28.3 Price-setting in Trading networks - Chapter 11 -
https://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf

29. (THEORY) VALUE FOR ELECTRONIC MONEY: How is above MAC money earned - This again
requires linking value to money (as money is not a value by itself and only a pointer
to valuable item or resource). Thus any buyer can "earn" MAC money by something
similar to a barter.

30. (THEORY and IMPLEMENTATION) FIXING VALUE FOR MAC MONEY: To delineate corruption as
discussed in 27 above with value() disparity between MAC money and REPLY goods and
services, an arbiter node in cloud has to "judge" the value of MAC sent by buyer and
goods and services from seller and proclaim if corruption exists. Thus value()
function itself has to be some kind of machine learning algorithm. This is related to
or same as points 12 to 23 above. For example, while buying an item for few million
bucks, value() has to take as input the description of the item and calculate the
value "ideally" which is difficult. Because there are no perfect references to
evaluate and only a weighted average of available market price range has to be taken
as a reference which is error-prone. value() function can be recursively defined
as("Reductionism"):
        ------------------------------------------------------------------------
-----------------------
        |value(i) = summation(value(ingredients of i)) + cost(integrating the
ingredients to create item i) |
        ------------------------------------------------------------------------
-----------------------
Obviously the above recursion combinatorially explodes into exponential number of
nodes in the recursion tree. Ideally recursion has to go deep upto quarks and leptons
that makeup the standard model. If for practical purposes, recursion depth is
restricted to t then size of value() tree is O(m^t) where m is average number of
ingredients per component. Hence any algorithm computing the value() recursion has to
be exponential in time. Computation of value() in the leaf nodes of the recursion is
most crucial as they percolate bottom-up. If leaf nodes of all possible items are same
(like quarks and leptons making up universe) then such atomic ingredient has to have
"same" value for all items. Only the integration cost varies in the levels of the
tree.For infinite case, value() function is conjectured to be undecidable - probably
invoking some halting problem reduction .But above value() function could be Fixed
Parameter Tractable in parameter recursion depth - t but yet could only be an
approximation. A Turing machine computing value() function exactly might loop forever
and thus Recursively Enumerable and not Recursive. A CVXPY implementation for Pricing
Market Equilibrium has been implemented in KingCobra.

31. (THEORY) Buyer-Seller and MAC electronic money transaction schematic:
---------------------------------------------------------------------------

        Buyer            A-------<id><refcnt:0>----------------------> Seller <id>
<refcnt:1> (increments refcnt)
        (<id><refcnt:1>  |
         <id><refcnt:0>  |
        after decrement  |
        refcnt           |
        )---------------->
```

Above has to be transactional (i.e atomic across cloud nodes)

32. (THEORY) MAC protocol reaper

Reaper thread in each cloud node harvests the zero refcounted allocations and invokes destructors on them. Same MAC id cannot have kref count of 1 or above in more than one cloud node due to the transaction mentioned previously.

33. (THEORY) Cloud Policing With Arbiters - Revisited:

When a suspect node is analyzed when a complaint problem is filed on it, (1) it is of foremost importance on how flawless is the arbiter who investigates on that and is there a perfect way to choose a perfect arbiter. In the absence of the previous credibility of entire cloud judiciary is blown to smithereens and falls apart. (2) Assuming a perfect arbiter which is questionable, next thing is to analyze the credibility of the node who sulked. This is nothing but the Citation problem in http://arxiv.org/abs/1106.4102 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf where a node can positively or negatively cite another node a "Societal Norm" which can be faulted and citations/opinions could be concocted with malafide intent(perjury). This is rather a generalization of PageRank algorithm with negative citations. Thus Perfect Cloud Arbitration could be an unsolvable problem. (3) Even if both arbiter and complainant are perfect which is again questionable, there are still loopholes - lack of evidences or implicated witnesses might portray a negative impression of a positive node. Thus there are 3 tiers of weakenings in cloud arbitration and there could be more. P(Good) series in https://sites.google.com/site/kuja27/ precisely addresses this problem.

34. (THEORY) MAC Money Flow as MaxFlow problem:

Transactions happening in a cloud are edges between the nodes involved (buyer and seller). Thus it creates a huge directed graph. Flow of money in this graph can be modelled as Flow network. Minimum Cut of this graph shows crucial nodes in the graph which play vital role in cloud economy removal of which paralyzes the cloud (could be Central bank and other financial institutions). This graph has bidirectional edges where one direction is for money and the opposite direction is for Goods and Services. In the Flow network sum of flows is zero. But in the Money Flow Network each node is having a cash reserve ratio (CRR) due to commercial transactions which is confidential and privy to that node only and thus sum of flows can not be zero. Hub nodes in the Money Flow graph which can be obtained by getting k-core or D-core of the graph by some graph peeling algorithms are crucial nodes to the economy that contribute to Money circulation.

35. (THEORY) Cycles and components in above MAC Money Flow Graph:

Above graph of money transactions could be cyclic which implies a supply chain. Strongly connected components of this graph are most related nodes that are in same industry.

36. (THEORY) STOCK TRADING:

One of the component in above MAC Money Flow Graph of cloud could be a virtual Stock Exchange. Based on the financial and securities transactions of constituent organizations in the graph, index of the exchange varies.

37. (THEORY) Analysis of Poverty and Alleviation through above money flow graph:

Weights of the edges of money flow graph are the denominations of the transaction. Thus high value edges and low value edges divide the Graph logically into Rich and Poor strata(Bourgeoisie and Proletariat subgraphs). Equitable graph is the one which does not have too much of value difference between Rich and Poor sets of edges - a

utopian to achieve. Mathematically, it is an optimization LP problem that seeks to minimize sum(RichEdges)-sum(PoorEdges) or Sum(RichVertices) - sum(PoorVertices) - without harming either - to be precise. This requires money flow to be programmed to find a feasible solution to this LP subject to constraints like work-pay parity etc., (there could be more variables and constraints to this LP) . Due to CRR above Vertices also can be Rich and Poor in addition to Rich Edges and Poor Edges.

38.(THEORY) Demand and Supply and Value() function:

Alternative to the recursive definition of value() function above can be done through Demand and Supply - more the demand and less the supply, price increases and vice-versa. This is quite subjective compared to absolute recursive definition above. To simulate demand and supply, the weights of the money edges (-> direction) in the bidirectional graph change and fluctuate dynamically over time for unchanging weights of the Goods and Services edges (<- direction) between any pair of Buyer-Seller vertices. This makes Money and G&S Flow graph a Dynamic Graph with edge weight update primitive.

39.(THEORY) Hidden or Colored Money:

In an ideal Cloud with only MAC currencies, colored money can co-exist if (not limited to) some money trails are missing, due to "cloud corruption", systemic failure, hardware and network issues etc.,. Probably this is the direct consequence of CAP theorem and can be conjectured to be undecidable. Hidden money is to some extent dependent on quantity of net flow (if non-zero) and how much of this net flow is contributed by Rich vertices and Edges. Money Circulation with Colored money can be formulated as Network Flow Problem with Time horizon and storage at nodes. Time horizon implies a certain flow has to happen before a stop time. Flow conservation is affected by this Dynamic Flow because of storage at nodes and money entering a node need not be equal to money leaving. Thus not all Hidden/Colored money is illegal in theoretical terms. As mentioned in 39.1, storage is simulated with a closed loop at nodes so that flow conservation is not seemingly violated. Profiteering is achieved by money flows over time in financial markets by assigining a multiplicative factor at each edge which accrues through a cycle and comes back to start node in cycle with a magnification in value. BlockChain techniques maintain ledgers which record all transactions globally thus decimating hidden unaccounted wealth if any. KingCobra experimental MAC currency relies on unique global identifier and global refcounts with atomic cloud transactions in linux kernel. Money Flow Graph mentioned in 34-39 has striking resemblance to Money Flow Markets already studied in Algorithmic Game Theory(AGT). 39.3 primarily devotes to Pricing and Equilibrium of Edges in Money Flow Graphs and not much on Colored Money Flow. An algorithm to find colored money flow could be a major advance in AGT. Prima facie there exists no zero-knowledge blackbox proof algorithm to find colored money because all storage data is a prerequisite which is impossible to know. Money trail for MAC currency described in 22,27,28 requires tracking of currencies. There are recent dollar and euro bills issued with Radio Frequency ID tags (RFID).

Total storage of money in Flow Market Graph = | Incoming money flow at Source - Incoming money flow at Sink | i.e Flow conservation is no longer obeyed.

There is a special vertex in Money Flow Market designated as Direct Taxation Hub which has incoming direct tax money flow edges from all other vertices in Flow market.Colored money is then approximately the taxed storage money estimated above minus the net flow of money received at Taxation Hub.

Colored Money = | Total storage money * Direct Taxation rate - Incoming Flow at Direct Taxation Hub Node|
Colored Money = | Incoming money flow at Source * Direct Taxation rate - Incoming money flow at Sink * Direct Taxation rate - Incoming Flow at Direct Taxation Hub Node |

Previous is an approximate naive zero-knowledge estimation of Colored money in Money Flow Market. This assumes that there is always a sink in Money Flow Market which is not necessarily valid unless notes are returned to mint. Satellite RFID Tracking technologies though invasive and intrusive like previous can present a rough figure of total money circulation in Source and Sink.

Above estimate is for direct taxation and resultant evasion. For indirect taxes (on Goods and Services etc.,), there is no direct impact on the money component and only G&S are affected. Combining Goods,Services and Income into one (direct+indirect) taxation could have a negative effect on Colored money because incentive to hide money no longer exists. For example, if only Goods and Services consumed by high-income brackets are preferentially taxed at high percentage replacing tax on income, income is also indirectly taxed in addition to G&S. Thus there is no necessity to tax income and any advantage of such direct taxation is indirectly usurped by preferential G&S taxation. Previous zero-knowledge estimate then becomes:

Colored Money = | Total Goods and Services * Taxation rate - Incoming Flow at Indirect Taxation Hub Node|
Colored Money = | GDP^2 * Tax-to-GDP ratio - Incoming Flow at Indirect Taxation Hub Node| where GDP is assumed to be equal to Total Goods and Services.

References:

39.1 Network Flows over time over Storage Area Networks (SAN) - https://hal.inria.fr/inria-00071643/document
39.2 Network Flow - [GoldbergTardosTarjan] - http://www.cs.cornell.edu/~eva/network.flow.algorithms.pdf
39.3 Algorithmic Game Theory - Flow Markets - [TimRoughGarden] - http://theory.stanford.edu/~tim/books.html
39.4 RFID tagged currencies - $100 bill - http://www.businessinsider.in/New-Smart-Paper-Could-Put-An-End-To-Dark-Money/articleshow/21134569.cms
39.5 Cons of RFID currencies - http://www.prisonplanet.com/022904rfidtagsexplode.html
39.6 Mechanism Design and Machine Learning - https://www.cs.cmu.edu/~mblum/search/AGTML35.pdf - Design of algorithms for maximizing gain in auctions involving sellers and buyers
39.7 Financial and Economic Networks - https://supernet.isenberg.umass.edu/bookser/innov-ch1.pdf

---
Commits as on 1 March 2014

Example java Publisher and Listeners that use ActiveMQ as the messaging middleware have been committed to repository for an ActiveMQ queue instance created for KingCobra. For multiple clients this might have to be a Topic rather than Queue instance. Request types above and a workflow framework can be added on this. This will be a JMS compliant implementation which might slow down compared to a linux workqueue or queue implementation being done in VIRGO.

Commits as on 17 March 2014

KingCobra userspace library and kernelspace driver module have been implemented that are invoked 1) either in usermode by call_usermodehelper()
2) or through intermodule invocation through exported symbols in KingCobra kernel module, by the workqueue handler in VIRGO workqueue implementation.

Commits as on 22 March 2014

Minimalistic Kernelspace messaging server framework with kernel workqueue,handler and remote cloud client has been completed - For this VIRGO clone cpupooling driver has

been added a clause based on a boolean flag, to direct incoming request from remote client to VIRGO linux workqueue which is popped by workqueue handler that invokes a servicerequest function on the KingCobra kernel module. (Build notes: To remove any build or symbol errors, Module.symvers from VIRGO queue has to be copied to VIRGO cloudexec and built to get a unified VIRGO cloudexec Module.symvers that has exported symbol definitions for push_request()). End-to-end test with telnet path client sending a request to VIRGO cloudexec service, that gets queued in kernel workqueue, handled by workqueue handler that finally invokes KingCobra service request function has been done and the kern.log has been added to repository at drivers/virgo/queuing/test_logs/

Commits as on 29 March 2014

Initial commits for KingCobra Request Response done by adding 2 new functions parse_ip_address() and reply_to_publisher() in kingcobra_servicerequest_kernelspace()

Commits as on 30 March 2014

Both VIRGO cpupooling and mempooling drivers have been modified with use_as_kingcobra_service boolean flag for sending incoming remote cloud node requests to VIRGO queue which is serviced by workqueue handler and KingCobra service as per the above schematic diagram and replied to.

Commits as on 6 April 2014

Fixes for REQUEST and REPLY headers for KingCobra has been made in virgo_cloudexec_mempool recvfrom() if clause and in request parser in KingCobra with strsep(). This has been implemented only in VIRGO mempool codepath and not in VIRGO clone.

Commits as on 7 April 2014

New function parse_timestamp() has been added to retrieve the timestamp set by the VIRGO mempool driver before pushing the request to VIRGO queue driver

---
Commits as on 29 April 2014

Intial commits for disk persistence of KingCobra request-reply queue messages have been done with addition of new boolean flag kingcobra_disk_persistence. VFS calls are used to open and write to the queue.

Commits as on 26 August 2014

KingCobra driver has been ported to 3.15.5 kernel and bugs related to a kernel_recvmsg() crash, timestamp parsing etc., have been fixed. The random crashes were most likely due to incorrect parameters to filp_open() of disk persistence file and filesystem being mounted as read-only.

Version 14.9.9 release tagged on 9 September 2014

Version 15.1.8 release tagged on 8 January 2015

```
-----------------------------------------
Commits as on 17 August 2015
-----------------------------------------
KingCobra + VIRGO Queuing port of Linux Kernel 4.1.5 :
 - changed the REQUEST_REPLY.queue disk persisted queue path to
/var/log/kingcobra/REQUEST_REPLY.queue
 - kernel built sources, object files
 - kern.log with logs for telnet request sent to VIRGO queue driver, queued in kernel
work queue and handler invocation for the
KingCobra service request kernel function for the popped request; disk persisted
/var/log/kingcobra/REQUEST_REPLY.queue

-----------------------------------------
Commits as on 14 October 2015
-----------------------------------------
AsFer Cloud Perfect Forwarding binaries are invoked through call_usermodehelper() in
VIRGO queue. KingCobra commands has been updated with a clause for cloud perfect
forwarding.

-----------------------------------------
Commits as on 15 October 2015
-----------------------------------------
- Updated KingCobra module binaries and build generated sources
- kingcobra_usermode_log.txt with "not found" error from output redirection
(kingcobra_commands.c). This error is due to need for absolute path. But there are
"alloc_fd: slot 1 not NULL!" after fd_install() is uncommented in virgo_queue.h
call_usermodehelper() code. The kern.log with these errors has been added to testlogs
- kingcobra_commands.c has been changed to invoke absolute path executable. With
uncommenting of fd_install and set_ds code in virgo_queue the return code of
call_usermodehelper() is 0 indicating successful invocation

-------------------------------------------------------------------------
Commits as on 10 January 2016
-------------------------------------------------------------------------
NeuronRain KingCobra research version 2016.1.10 released.

-------------------------------------------------------------------------------------
------------
NEURONRAIN VIRGO Commits for virgo_clone()/telnet -> VIRGO cpupooling -> VIRGO Queue -
> KingCobra
- as on 12 February 2016
-------------------------------------------------------------------------------------
------------
VIRGO commit:
https://github.com/shrinivaasanka/virgo-linux-github-
code/commit/72d9cfc90855719542cdb62ce40b798cc7431b3d

Commit comments:
-------------------------------------------------------------------------------------
---
Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue ->
KingCobra
-------------------------------------------------------------------------------------
---
*) This was commented earlier for the past few years due to a serious kernel panic in
previous kernel versions - <= 3.15.5
*) In 4.1.5 a deadlock between VIRGO CPUPooling and VIRGO queue driver init was
causing following error in "use_as_kingcobra_service" clause :
  - "gave up waiting for virgo_queue init, unknown symbol push_request()"
*) To address this a new boolean flag to selectively enable and disable VIRGO Queue
kernel service mode "virgo_queue_reactor_service_mode" has been added.
```

*) With this flag VIRGO Queue is both a kernel service driver and a standalone exporter of function symbols - push_request/pop_request
*) Incoming request data from telnet/virgo_clone() system call into cpupooling kernel service reactor pattern (virgo cpupooling listener loop) is treated as generic string and handed over to VIRGO queue and KingCobra which publishes it.
*) This resolves a long standing deadlock above between VIRGO cpupooling "use_as_kingcobra_service" clause and VIRGO queue init.
*) This makes virgo_clone() systemcall/telnet both synchronous and asynchronous - requests from telnet client/virgo_clone() system call can be
either synchronous RPC functions executed on a remote cloud node in kernelspace (or) an asynchronous invocation through "use_as_kingcobra_service"
 clause path to VIRGO Queue driver which enqueues the data in kernel workqueue and subsequently popped by KingCobra.
*) Above saves an additional code implementation for virgo_queue syscall paths - virgo_clone() handles, based on config selected, incoming
data passed to it either as a remote procedure call or as a data that is pushed to VIRGO Queue/KingCobra pub-sub kernelspace
*) Kernel Logs and REQUEST_REPLY.queue for above commits have been added to kingcobra c-src/testlogs/

Commits - KingCobra 64 bit and VIRGO Queue + KingCobra telnet requests - 17 April 2017
---
*) Rebuilt KingCobra 64bit kernel module
*) telnet requests to VIRGO64 Queueing module listener driver are serviced by KingCobra servicerequest
*) Request_Reply queue persisted for this VIRGO Queue + KingCobra routing has been committed to c-src/testlogs.
*) kern.log for this routing has been committed in VIRGO64 queueing directory
) Similar to other drivers struct socket reinterpret cast to int has been removed and has been made const in queuesvc kernel thread

---
(FEATURE-DONE) Commits - CVXPY implementation for Eisenberg-Gale Convex Program - 18 August 2017

(*) First commits for Convex Optimized Market Equilibrium Prices
(*) Imports CVXPY Convex Program solver
(*) Objective function is a logistic variant of Eisenberg-Gale Convex Program i.e uses money * log(1+e^utility) instead of
 money * log(utility) because of curvature error (log is error flagged as concave and logistic is convex per:
 http://www.cvxpy.org/en/latest/tutorial/functions/index.html#vector-matrix-functions)
(*) Formulates constraints and objective functions based on http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf - Page 106 and Equation 5.1
(*) But, For all installed solvers ECOS, ECOS_BB, SCS, LS solved convex program prints value as None despite all constraints and objective functions being convex. Also is_dcp() prints "not a disciplined convex program". Logs in testlogs/.
(*) Obviously it should have worked. Therefore this is only a partial implementation commit.
(*) This implementation uses numpy randomly initialized arrays for Money each buyer has and per-good utility(happiness) each buyer has.
(*) Replacing money with perceived merit values translates this Market Equilibrium - Intrinsic Value versus Market Price - to Merit
 Equilibrium - Intrinsic Merit versus Perceived Merit. This has been already described in NeuronRain AsFer Design Documents:

```
        - http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-
docs/AstroInferDesign.txt and
        - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-
docs/AstroInferDesign.txt

------------------------------------------------------------------------------
--------------------------
(FEATURE-DONE) Commits - Convex Optimization - DCCP - 21 August 2017
------------------------------------------------------------------------------
--------------------------
(*) import dccp has been added
(*) DCCP is the recent advancement and generalization of DCP for convex-concave
programs
(*) method='dccp' has been added as parameter to solve()
(*) Objective function has been changed to log() from logistic() - curvature is
concave which is in conflict with definition of
eisenberg-gale convex program in textbooks. Reason for this contradiction is unknown.
(*) But DCCP overcomes the DCP limitation and solve() prints converged solutions for
objective functions
(*) logs have been committed to testlogs/
(*) CVXOPT solver has been installed but it does not solve the Eisenberg-Gale
objective function. Only SCS solver works - by default applies
KKT conditions indirectly.

------------------------------------------------------------------------------
--------------------------
(FEATURE-DONE) Commits - Convex Optimization - DCCP - 22 August 2017
------------------------------------------------------------------------------
--------------------------
(*) Verbose set to True for printing Splitting Conic Solver progress information
(*) logs committed to testlogs/

------------------------------------------------------------------------------
-------------------------------------------------
(FEATURE-DONE) Commits - Convex Optimization update - 29 August 2017
------------------------------------------------------------------------------
-------------------------------------------------
(*) Removed hardcoded variable values in objective and constraints
(*) In the context of pricing, ECOS Error Metrics print the matrices of market
clearing prices for goods
(Reference - pages 3072 and 3073 of
https://web.stanford.edu/~boyd/papers/pdf/ecos_ecc.pdf - KKT conditions in ECOS
solver)

------------------------------------------------------------------------------
-----------------------------------------
(FEATURE-DONE) Convex Optimization - Pricing Computation - 30 August 2017
------------------------------------------------------------------------------
-----------------------------------------
(*) Prices of Goods/Services have been computed explicitly from Karush-Kuhn-Tucker
Conditions (1,2,3 and especially 4)
(*) References:
        - Pages 106-108 of http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-
game-theory.pdf)
        - KKT conditions and Conic Optimization- https://arxiv.org/pdf/1312.3039.pdf
(*) logs committed to testlogs/

------------------------------------------------------------------------------
-------------------------
(FEATURE-DONE) KingCobra Kernelspace Messaging Driver for 4.13.3 64-bit kernel - 24
September 2017
```

```
------------------------------------------------------------------------------------------
-------------------------
(*) KingCobra driver in GitHub and SourceForge at present are 32-bit based on mainline
4.1.5 kernel
(*) Both USB-md and KingCobra kernel modules are subsidiaries of VIRGO kernel
(*) There is a necessity for 64-bit version of KingCobra for interoperability to
VIRGO64 64-bit kernel on mainline version 4.13.3
(*) This requires separate repository for KingCobra because of significant kernel
function changes between 4.1.5 and 4.13.3 and
idiosyncrasies of 64-bit
(*) KingCobra driver has been rebuilt on 4.13.3 64-bit kernel after some changes to
function prototypes and new kingcobra64 repository is
initialized with these commits
(*) KingCobra kernel sockets have been TLS-ed by kernel_setsockopt(TX_TLS) newly
introduced in 4.13 kernel.
(*) After this complete request-reply traffic from VIRGO64 system calls to VIRGO64
queueing and KingCobra is encrypted.

------------------------------------------------------------------------------------------
-------------------------
(FEATURE-DONE) Commits - telnet - VIRGO64Queue - KingCobra64 -  25 September 2017
------------------------------------------------------------------------------------------
-------------------------
(*) Disk persisted KingCobra64 REQUEST-REPLY Queue written by VIRGO64 Queue to
KingCobra64 telnet invocation after 4.13.3 64-bit KTLS upgrade
has been committed

------------------------------------------------------------------------------------------
---------------------------------------------
(FEATURE-DONE) VIRGO64 Queueing Kernel Module Listener - KingCobra64 - 4.13.3 - 6
October 2017
------------------------------------------------------------------------------------------
---------------------------------------------
(*) telnet client connection to VIRGO64 Queue and a subsequent workqueue routing
(pub/sub) to KingCobra64 has been tested on 4.13.3
(*) TX_TLS socket option has not been disabled and is a no-op because it has no effect
on the socket.
(*) REQUEST_REPLY.queue for this routing from VIRGO64 queue and persisted by
KingCobra64 has been committed to KingCobra64 repositories in GitHub and SourceForge

------------------------------------------------------------------------------------------
-------------------------------------------------
(FEATURE-DONE) KingCobra64 Neuro Electronic Currency transactional cloud move -
Perfect Forward - 17 January 2018
------------------------------------------------------------------------------------------
-------------------------------------------------
(#) Neuro Currency cloud perfect forward has been made transactional by wrapping it by
Python Transaction Manager (widely used in Zope
Python Application Server)
(#) imports transaction python package and invokes begin() and commit() on subprocess
call to neuro cloud move client and server
```