SDO Blockchain Platform

Blockchain Protocol Analysis & Recommendation

Prepared for Smart Dubai

April 19th, 2017

Prepared by ConsenSys

Contents

SDO Blockchain Platform - Overview and Recommendation

Blockchains promise a new era of distributed and secure computing with innovations in trust to enable new forms of interaction, monetization, and governance. Blockchains present these capabilities by composing technologies such as cryptographic security and immutability, decentralization, and open protocols. Even now, millions of users disintermediate their interactions in novel and unique ways, having transacted trillions of US dollars on public blockchain networks including Bitcoin and Ethereum. So it should come as no surprise for governments and enterprises to advocate for the advent of blockchains in order to further develop their valuable relationships with their constituents and customers.

Governments and enterprises want the ability to harvest the rewards of blockchains with a platform that has the flexibility, fidelity, and robustness to execute and support their diverse and unique visions. However, such endeavors carry risk - a poorly chosen blockchain platform can incur significant cost, hinder innovation, and limit interoperability with existing and new systems. Selecting a suitable blockchain platform depends on having a clear understanding of the platform's security, scalability, and support, with concise insights into the direction and growth of the platform's ecosystem in a way that is independent of a vendor's control.

Ethereum distinguishes itself as a clear leader amongst blockchain platforms with its formally specified security and smart contract capabilities, and multi-billion dollars of value protected on the public network. The recently formed Enterprise Ethereum Alliance (EEA) involves more than 30 launch members from enterprises, startups, academics, and technology vendors, including ConsenSys, Bank of Santander, Accenture, and BNY Mellon. The EEA's stated goal is to "*define enterprise-grade software capable of handling the most complex, highly demanding applications at the speed of business".* These efforts drive home the point that Ethereum has proven capability to support private, permissioned blockchains for enterprise and government use cases. In fact, since Ethereum's launch in mid-2015, its clients have included support for private, permissioned blockchains.

Ethereum is an open platform standard that is supported and led by its developer community. An example of this bottom-up approach is that proposals to improve the core specification are submitted as github pull requests (the git protocol was originally created to manage Linux kernel development). During this ongoing process, enterprises have been integrating their existing systems with Ethereum protocols, such as identity management that can be then used for immigration connecting country visa systems.

The high demand for enterprise adoption of Ethereum is emphasized by top cloud vendors who already support Ethereum as a first class citizen: Microsoft

Azure, RedHat OpenShift, Pivotal CloudFoundry all feature Ethereum as one of their, if not the, primary blockchain offering. The promotion of such a bottom-up, shared, vendor-neutral process to define Ethereum specifications is one of the key reasons (literally) hundreds of enterprises are applying for participation in the next EEA membership cohort which will be announced in May, placing the EEA's growth on track to outpace all other blockchain consortiums combined.

Enterprises are already deploying Ethereum at scale and many implantations are in production or running live transactions. For example, ING recently announced it was opening its Ethereum-based oil trading platform to additional banks to participate in live transactions. Santander has announced multiple applications built on Ethereum. JP Morgan has released its previously internal project Quorum and is reported to be running the software in production. Even enterprises in typically more conservative industries, such as commodities and mining, are running Ethereum in production: BHP Billiton announced its supply chain tracking application in summer 2016 and is in live production. We are well-aware of many dozens, if not hundreds, of applications built on Ethereum entering production in 2017. Given its open source availability and the tens of thousands of Ethereum developers it is impossible to estimate the total number of applications but it is surely sizable.

Many, if not all, of these applications are running on permissioned, private Ethereum blockchains. This is achieved by leveraging Ethereum's configurable network capabilities to govern access. Certain implementations include data and transaction privacy. Even with current architectures, some enterprises have reported achieving >1,000 transactions per second in testing environments, and many are running in 10s or 100s of transactions per second in production. Even the public network has block completion times of sub-15 seconds.

While the public Ethereum blockchain may not be an immediate deployment platform for government and enterprise applications, it has certain important elements that are already valuable. First, given the monetary value of the public network, Ethereum is hardened by continual activity. Second, it is the dominant platform for the 'token ecosystem', in which innovative (and technologically risky) projects are funded and key developers are incentivized. Third, it encourages experimentation that is already trickling into government and enterprise applications, such as identity, governance and indeed virtual currencies themselves. Finally, as the Executive Director of the Hyperledger Foundation noted, private chains need public chains, initially to anchor cross chain transactions and over time as a “network of networks” and for interconnection.

That said, Ethereum continues to innovate at great pace and with a materially increased focus on enterprise requirements. Currently available, production Ethereum clients such as Parity and Geth support high performance consensus algorithms, while the core Ethereum roadmap includes adoption of proof-of-stake, sharding, state channels, and near-metal virtual machine.

In addition to creating a specification and reference implementation based on current best in class approaches, the EEA is also exploring next generation approaches to privacy, throughput and scalability including:

1. New privacy approaches (on-chain ZKP, elliptic rings, off-chain at-rest encryption, encrypted zones, etc)
2. Pluggable consensus (round-robin BFT with finality, non-BFT traditional consensus, proof-of-work, hybrid, non-fully-replicated)
3. Permissioning (within EVM per account, validator bonding, granularity of permissioning, interfaces for external ACLs)
4. Checkpointing (how to manage state 'at rest', archiving, recovery)
5. Cross-chain messaging (compatibility by communication rather than protocol-level compatibility, oracles, cross-validators, interledger, cosmos, etc)

It is important to note that these features already exist in a variety of forms within Ethereum clients and that these EEA discussions are for building a common *specification*. For example, JPMorgan's Quorum Ethereum client provides a prototypical approach to configurable privacy with its ability to send a private transaction over a permissioned network that updates the private state database of specified parties and storing a permanent hash record of the private transaction in everyone's non-private state database. Another example is the support for pluggable consensus by the two most popular open source Ethereum clients, Geth and Parity, giving users the option of selecting Proof of Work (suitable for public networks), Proof of Authority (suitable for private networks), or Tendermint (suitable for private networks having different levels of trust). Finally, since Oct 2015, BlockApps STRATO has had an enterprise offering of a private, permissioned Ethereum client implementation that leverages Kafka to scale the number of transactions per second. The EEA is a forum for key stakeholders of these enterprise features to communicate and work together towards standardization furthering the development of robust enterprise features in Ethereum.

In the following sections, we give details of the Ethereum protocol, including security, scalability, consensus algorithms, developer support, data layer, pluggability, and governance models. In each section, we discuss how the corresponding Ethereum protocol satisfies a Dubai Production Blockchain environment, describing current features that are available with an eye on short-term milestones. As well, we derive from our experiences recommended infrastructure to support load and performance of a Dubai Production Blockchain based on the Ethereum protocol.

Our Recommendation

Providing a specific recommendation for the Dubai Production Blockchain environment is nuanced. First, there are timing considerations, based on the POC, Pilot, and Production model for the program. Second, there is a trade-off between technology available today vs the roadmap. Third, is the availability of production infrastructure designed for blockchain deployments. And, finally there are the Dubai specific requirements that we have begun to discuss.

We therefore propose the following recommendation:

1. We believe it is highly unlikely that within Dubai or any other state we will see only a single protocol adopted because of risk mitigation, application availability, talent management and systems integration. We also believe that Ethereum is one of a very small number of protocols capable of delivering Dubai's requirements.
2. Immediate POCs and Pilots should be developed using 'stock', generally available software, hosted on 'cloud sandboxes'. In the case of Ethereum, we would recommend clients such as Parity and Geth, which are frequently used to host public nodes (and therefore heavily tested) and are also capable of running private, permissioned networks. The most recent releases of each support multiple consensus algorithms and privacy configurations.
3. For the purposes of near term production deployments, we would recommend leveraging the existing cloud infrastructure such as Microsoft Azure or IBM Bluemix, which already provide robust infrastructure for blockchain deployments, whether cloud, on-premise or hybrid. We would be happy to facilitate a demonstration of the Microsoft Azure blockchain infrastructure, including its multi-protocol support.
4. For the production system, a Dubai-specific, production blockchain infrastructure, which we will develop over 2017-2018. Given Dubai's requirements such as multi-protocol interoperability, scalability, specific permissioning and identity models, it is likely we will need to develop certain custom infrastructure, middleware and services as our team and Dr Sohail have discussed at length. Ideally, this platform would allow for *any* (or a set of) blockchain protocol to be deployed and elastically scaled, while retaining permissioning and cross-chain interoperability. Our aim would be to limit (approaching zero) any changes to the core protocol.

In the below, we have responded to each of your specific questions with regard to Ethereum’s capabilities both today and the future as outlined by its roadmap. We have also provided a reference infrastructure and summary information to help guide you. We look forward to discussing this further in the coming days and working with you to develop this exciting platform.

1. Security

In this section, we describe how the Ethereum specification defines cryptographically secure accounts based on (elliptic curve) public/private keys, and how invocation of smart contracts can be restricted to specific sets of accounts. We also describe how the network communication protocol currently implemented by Ethereum clients is cryptographically secure and scalable to thousands of nodes. We further describe how commonly used enterprise infrastructure, such as docker containers, and network security, such as firewalls and VPN, supplements the described facets of the specification to enable complete, secure, permissioned environments restricted to registered participants.

Permission Model

How does the Blockchain restrict membership to registered participants?

Ethereum blockchain accounts can either be “external accounts” (controlled by a key pair), or “contract accounts” (a smart contract). Contract accounts cannot perform actions on their own, and their actions are always initiated by an external account, making private key management an integral aspect of blockchain security.

In the case of a permissioned network, there are a number of mechanisms to restrict membership to registered participants:

1. Only users sharing the genesis block descriptor file can connect to the network described by that genesis block. The network protocol relies on the encrypted form of the genesis block descriptor to establish connections between peers.
2. This can further enhanced by creating a virtual private network (VPN) between the blockchain nodes. This will ensure that only registered participants can view and participate in the blockchain and its associated contracts and transactions. Even without VPN, communication between peers is encrypted and cannot be inspected.
3. Ethereum allows different forms of peer membership, such as blockchain authorities that can construct blocks and peers who cannot create blocks. For example, in a network with Proof of Authority as the consensus mechanism (https://github.com/parityte...) the list of authority accounts that are permitted to construct blocks is decided upon at network genesis and can only be modified by authorities.
4. Quorum has added a --permissioned flag that can specify a list of nodes (identified by IP address and public key) which restricts blockchain-network participants to this list.
5. In some networks it is desirable to allow different forms of peer membership, such as authorities that can construct blocks and peers that

cannot create blocks. In such networks, Proof of Authority for example, the list of authorities that are permitted to construct blocks is decided upon network genesis and can be modified only by authorities.

Another option that is available is the use of a hardware wallet to maintain private keys and interact with the Ethereum blockchain:
https://github.com/paritytech/parity/wiki/Ledger-Nano-S

Privacy and Confidentiality

Does the Blockchain support Confidentiality, Anonymity, Unlinkability? What are the mechanisms?

Confidentiality support exists in the form of running distinct networks restricted to users with the same network configuration. Network level isolation in the form of VPN augments this isolation. In quorum-like functionality, only those transactions involving a specifc node that is indicated in the transaction are kept on the node.

Anonymity exists only as pseudonymity since the account serves as a pseudonym; however, by default no identifying information (human/machine/network/client identifiers) are used in account creation, and it is up to account owners to reveal themselves. Links between blockchain addresses and identifying information, such as KYC details, can be kept in a separate secured environment visible only to those authorized to view those details.

Unlinkability exists in the form of Hierarchical Deterministic wallets which use a seed key to populate new keys in a way that cannot identify the original seed. The benefit of HD keys are that they can be used to generate new keys independent of the machine on which earlier keys were generated, obviating the need for new key backups and this reducing the burden of key management to that of root key management.

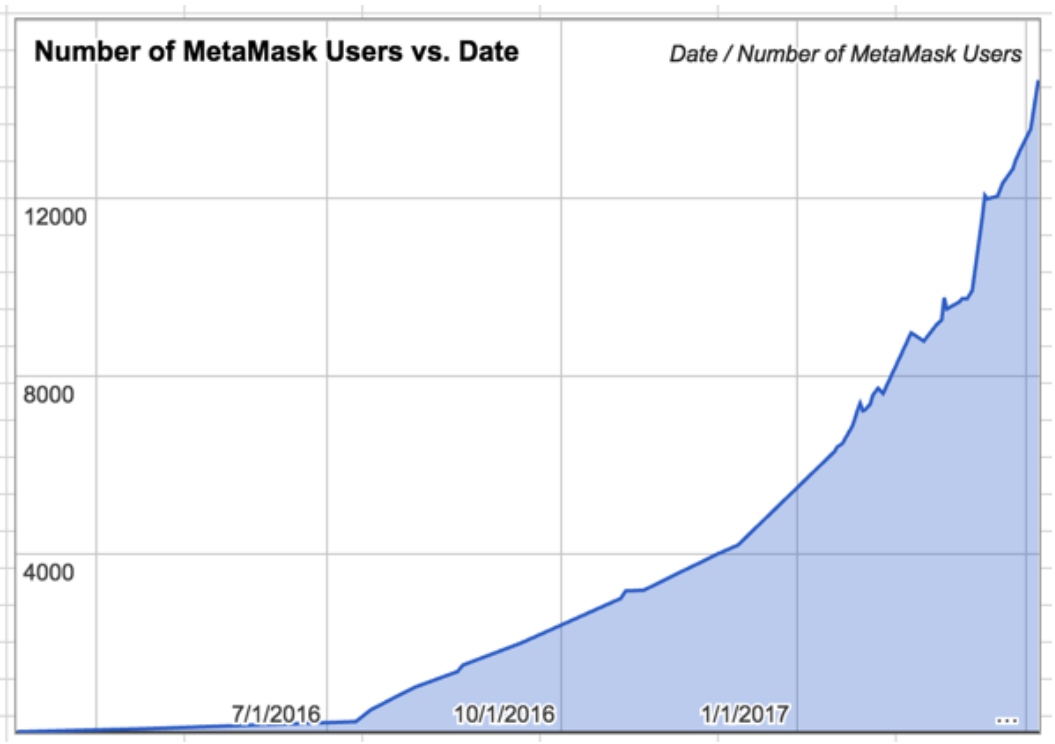
At a high level, “Users can use these accounts to organize the funds in the same fashion as bank accounts; for donation purposes (where all addresses are considered public), for saving purposes, for common expenses etc.”

https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

Standardization specific to Ethereum wallets:
https://github.com/ethereum/EIPs/issues/84

Metamask, a ConsenSys product, provides this functionality and is active in production for thousands of users, and supports multiple networks, for example users can use it on multiple private networks as well as the public network: https://github.com/MetaMask/eth-hd-keyring/blob/master/index.js



Metamask is the main traffic driver to ConsenSys's Infura high availability, high transaction throughput back-end infrastructure which services 120,000,000 requests per day on average from the main and test public Ethereum blockchains. Metamask is currently installed in 15k browsers and represents approximately 1% of public transaction traffic. MyEtherWallet is another HD wallet provider which serves approximately 3% of public Ethereum traffic, and which equates to approximately 2.5k transactions/day.

Privacy and confidentiality in the context of data sharing is covered in the section “Identity, Trust and Private Data”.

Runtime Isolation

What kind of runtime isolation is provided between contracts?

Smart contracts are executed by all block-creating peers on the network. If there is an issue with a client or peer, then other peers will continue in isolated and

distributed fashion. If a contract has an error then the error is recorded onto the blockchain and is not hidden by the operating system or container. Contracts are limited specific set of (Ethereum Virtual Machine) EVM **operands**. These operands are deterministic and operate identically across all platforms and clients (on those that are creating blocks). The set of permitted operands does not allow low level calls, and storage and input/output calls are restricted to those defined by the storage and memory model specified in the Ethereum Yellowpaper. There is no serialization/deserialization attack vector since there is no marshalling/unmarshalling of contracts/accounts types. Executing an Ethereum client within a Docker/VM, while leveraging Linux permissioned models such as AppArmor, provides security at the level of the operating system. Furthermore, since Ethereum smart contracts are not long-running processes, and the EVM clears memory between distinct transactions, there is intrinsic isolation and does not require garbage collection.

Resistance to Attacks

What types of attacks is the Blockchain designed to mitigate?

With many avenues to target an Ethereum blockchain network, some of which are attacks similar to those described for other blockchain protocols (https://en.bitcoin.it/wiki/Weaknesses), the Ethereum network remains a robust and secure protocol.

To begin, the Ethereum network uses the concept of 'gas' as a measure of computation, whereby limiting the amount of gas ensures protection against code that may otherwise execute infinite loops.

In public networks, proof of work is the mechanism designed to prevent the 51% attack, which will make economically infeasible for an opposing party to take over the network, as they need to be able to produce the 51% of the current hashrate in order to control the canonical chain, thus, emitting doctored blocks.

The current setup of the bootnodes in the public chain (https://github.com/ethereum/go-ethereum/blob/master/params/bootnodes.go) which, as of 2017.04.18 distributes those machines around the world in Ireland, United States, Brazil, Australia, Singapore and Germany. These bootnodes provide the first layer of discovered nodes, making sure that the user will have a geographically rich distribution of nodes to start its own process of further discovery and synchronization.

Referencing both the Ethereum public blockchain as well as permissioned chains using PoA for example helps highlight the fact that permissioned chains get the benefits of what is learned on the public ethereum blockchain.

Apart from the consensus mechanism, private networks mostly use the same code base as the public network. This allows private networks to leverage the work done to secure the public network. Private networks do offer substantial benefits with respect to public networks having:

- control of the network topology (virtual networks, VPN, gateways) and
- control of the main entities of the nodes in charge of emitting blocks, whether those are generated through proof of work, proof of authority or a different consensus algorithm

High Security Deployment

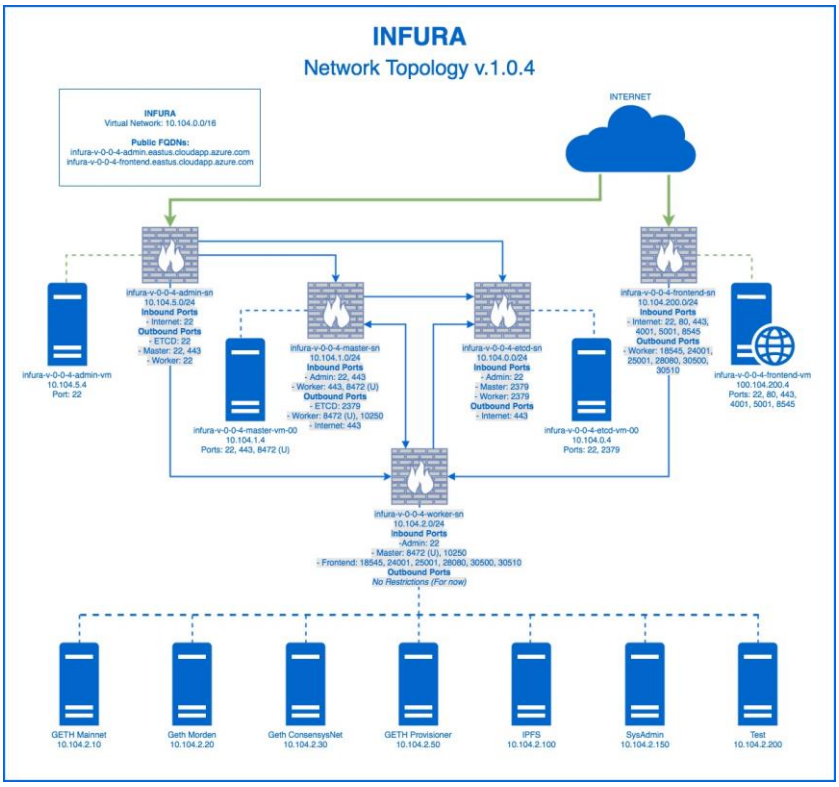
Is there an enhanced security deployment? What does it provide?

Peers can execute the Ethereum blockchain client in a Docker container, which restricts low-level calls. Additionally, Docker containers prevent a wide range of bugs in the Linux ecosystem by leveraging namespacing of processes and control groups to restrict low-level calls that may escape or escalate privileges, which will have negligible effect within a containerized instance.

The two most popular clients that can be used for private chains, Geth and Parity, both come with Docker support out-of-the-box:
https://github.com/ethereum/go-ethereum/tree/master/containers
https://github.com/paritytech/parity/tree/master/docker

As stated in the item "*Resistance to attacks*", for private networks the top measure so far is to design an adequate network topology able to control which machine from which network access, which services it access, and most importantly, which kind of messages it will let pass (i.e. Layer 7 firewalling).

In the specific case of INFURA, a scalable, standards-based cluster and API endpoint for Ethereum & IPFS, which **as of 2017.04.18 supports 120 millions of requests a day**, security focus was paramount in the design of the platform.



In this example we can see that just for a package to get into a Blockchain Node (ex: Mainnet), it needs to pass through an internet gateway, a reverse proxy, an internal firewall, and (not pictured) a L7 firewall, which will study the payload of the RPC request and redirect it properly. An `eth_blockNumber` request will be processed in a way different than an `eth_getLogs`, or a filter (callback) request, `eth_getFilterChanges`, which employs a more sophisticated internal logic to guarantee that a determined server will get it, not to mention our key service `eth_sendRawTransaction`.

To summarise the above paragraph, today's fast evolution of blockchain clients provide us with the best that research and implementation can provide us, in terms of being able to support a public network that already has more than 3.5 million blocks (not to mention its market capitalization of greater than 4 billion dollars) . Nevertheless, for a private blockchain architecture to be robust, an adequate design of **network topology** combined with versatile **middleware** are key requirements for production readiness.

2. Scalability

In this section, we give anecdotal results of benchmarks of Ethereum clients and describe our ongoing contributions to academic efforts to understand and benchmark networks of Ethereum clients under different types and numbers of transactions, leveraging the existing public network's dataset and enterprise transactional data.

Throughput

Provide throughput numbers achieved in lab testing? Describe test setup and workload used?

Improving Performance of the blockchain

Real evidence with the public network suggests that on average 12 transactions per second are achieved, amongst thousands of networked peers.
Anecdotal evidence suggests that on PoA with 1 authority and 20 peers, thousands of transactions per second can be achieved.
In order to benchmark scalable systems, a first step is to understand the types and number of transactions that are relevant for benchmarks. For instance, if the majority of transactions and time spent processing transactions on a blockchain are simple value transfer, then there is little demand to improve performance of complex contract transactions.

There is a set of academic efforts **between Fortune 500 companies and Universities in North America** that have been initiated jointly with the EEA membership.

The Initiative for CryptoCurrencies and Contracts (IC3) is a task force conformed of faculty members at Cornell University, Cornell Tech, UC Berkeley, UIUC and the Technion. It's based at the Jacobs Technion-Cornell Institute at Cornell Tech in NYC. The group focuses on advancing blockchain technologies such as PBFT algorithms. An example of the advances achieved in a one-week hackathon is described here:
http://www.initc3.org/events/2016-07-20-IC3-Ethereum-Crypto-Boot-Camp-and-Workshop-at-Cornell-University.html

At the **University of Toronto (U of T)**, Dr. Shahan Khatchadourian (ConsenSys) has initiated two efforts that are investigating performance and benchmarking of Ethereum clients. Beginning with analyses of the public Ethereum network's blockchain data, revealing the different types and number of transactions will provide insight into the components of the Ethereum stack that would benefit from improved performance.

The **first effort**, coordinated with Dr. Mariano Consens (U of T), involves graduate students in an engineering analytics course who are leveraging scientific notebooks and scalable data processing techniques while examining blockchain analytics that have been used on cryptocurrencies such as Bitcoin.

The **second effort**, coordinated with Dr. Cristiana Amza (U of T), involves a graduate computer science MSc student who is investigating caching mechanisms with popular data-parallel processing frameworks, such as Hadoop, that are effective in analyzing append-only log data and that is well-suited to blockchain data, i.e., an incremental state update with each new block.

Scaling with "near-chain" strategies

In addition to simply improving the throughput of a single Ethereum node, there are several strategies available for scaling the throughput of the total network by "checking in" data and tasks to the blockchain, for use and verification as needed.

For scaling transaction throughput between finite numbers of parties, state channel systems like The Raiden Network allow parties of users to place deposits into a smart contract that allows faster transactions off-chain, with final account closure and verification on-chain.

For scaling the computational capabilities of the blockchain, projects like Golem and TrueBit provide on-chain marketplaces for mostly-off-chain computation, with a sort of digital verification of contested computations in the case of alleged mis-computation.

For scaling the storage capabilities of the blockchain, a simple strategy is to record a merkle root of a large database into a smart contract that is capable of verifying some data's inclusion in that merkle tree later. This allows the speed benefits of a pure hash-based data structure, but with the smart-contract automation of a Turing-Complete blockchain.

Beyond these current strategies for scaling computation and memory, the team building the next version of the Ethereum Virtual Machine, including Martin Becze, is exploring strategies of "sharding" the blockchain along contract boundaries, allowing seamless scaling of individual contracts as needed by their workloads. This strategy is much younger, but is designed to be backwards-compatible with the current Ethereum VM.

Number of Nodes

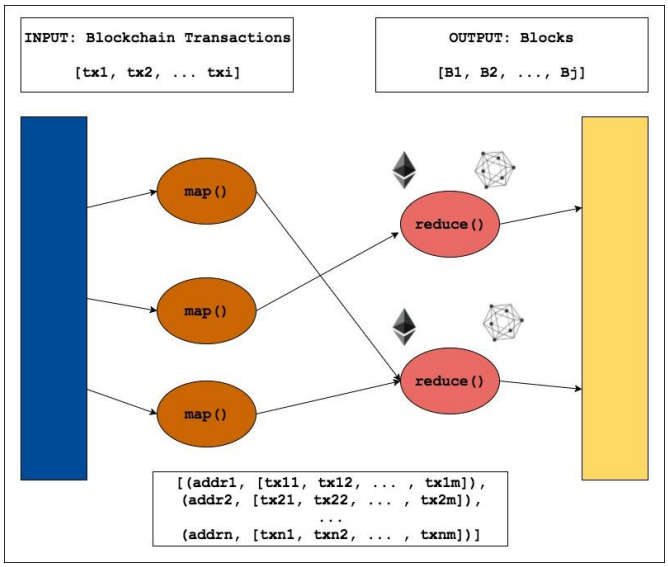
What is the number of full nodes beyond which the network does not scale well?

The current public network has around 6000 nodes with the Proof of Work (PoW) consensus engine. The goal of the PoW engine is that it provides the latency needed to execute smart contracts, create blocks, share blocks with the network, and add newly created blocks to the canonical chain. PoW, specifically ethash, is intended to provide a scalable and resilient form of block creation that enables any network participant to contribute to the construction of the canonical chain. This also means that the network requires a massive investment in the form of electricity, computational power, and time, to disrupt the canonical chain.
In networks where layers of trust between parties is more reasonable, different consensus algorithms can be used. For instance, the Proof of Authority consensus mechanism (used for one of the public Ethereum test nets at a large scale) enables massive networks and scale (theoretically) to hundreds of thousands of peers, leveraging a only a few peers to extend the canonical chain. To support environments with low levels of trust at massive scale involving millions of block-creating peers, there is active development on BLS signature schemes. This mechanism leverages random formation of groups with group signatures and enables all peers the ability to participate in extending the canonical chain.

Load Distribution

Does the protocol allow distribution of workload between participant nodes. How?

In a Proof of Authority chain, authoritative (block-creating) peers can act in a distributed fashion in a deterministic and provably secure way. For instance, an authoritative peer can be restricted to creating blocks limited to a specific range of accounts; this can be done by mapping the mod-ed value of a cryptographically hashed transaction's “from” field, where the resulting modded result value is mapped to a specific authoritative peer which will then construct blocks for ALL transactions having that “from” account. This is mod-value approach is the same as that of Hadoop, the massively parallel and scalable data processing framework created by Yahoo and used by companies like Google, Facebook, and thousands of organizations with hundreds of millions of users. This was shown at a high level in the SDO Ethereum presentation of April 13.



This solution currently does not exist but is proposed as a potential “low-hanging fruit” solution to implement due to the simplicity of the approach. Such hash-mod-based distribution functions are fewer than 10 lines long. https://github.com/shahankhatch/ScalableGraphSummaries/blob/master/SGBHadoop/src/main/java/edu/toronto/cs/sgbhadoop/hadoop220/MyCompoundKeyPartitioner.java

Number of Networks

Can a node participate in multiple networks?

A node can participate in multiple networks simply by launching multiple clients. These clients can also be launched in different Docker/VM containers for isolation.

The key to answering this particular question is agreeing on the definition of a node: A node as a process of a blockchain client, will represent only one instance of a particular blockchain. This node will be capable of synchronize its database to the canonical chain produced by consensus by the other nodes (and itself), relay transactions into the network, and, if the configuration is set up, it will process using its EVM the simple transactions and smart contracts to create blocks.

Now, if we consider a node as a server of an entity, it may, of course, need to connect to not only a single network, but support several blockchains for diverse reasons. In that case, only two things must be considered: The first one, run enough number of processes as per blockchains are required, and Second,

provide the required network permissions and accesses such that this node becomes a member on each permissioned blockchain.

Managing processes can be easily accomplished with coordinated bash tools per entity datacenter. Quorum examples (https://github.com/jpmorganchase/quorum-examples/tree/master/examples/7nodes) provide a good view on how simple is to start, stop a coordinated mesh of blockchain processes in a single virtual machine.
Scripts have been developed whose aim is to perform all setup and deployment activities from a single console, avoiding the need for the administrator to log into each machine. The setup and deployment scripts below have been developed by ConsenSys with the Quorum Ethereum client implementation for use in a production environment, and exemplify how machines can be configured to host one or more clients, each having a different set of roles, i.e., block maker, validator, and observer roles.
https://github.com/INFURA/quorum-tools

3. Consensus

In this section, we describe the consensus algorithms that are supported in a diverse array of conditions ranging from networks where all participants are untrusted, to networks where all participants are trusted. We also describe the pluggability of Ethereum clients and that they currently support the full range of trust models.

Failure Modes

Which failure modes are supported: e.g. Byzantine, crash fault tolerance?

The PoW consensus supports byzantine failures in that blocks created are validated by nodes. In PoA consensus, every member of the validation set is considered an authoritative and valid block maker which means that created blocks do not undergo validation. In both PoW and PoA, there is crash fault tolerance: PoW allows for any mining party to propose a block, and PoA allows a different validator to be considered authoritative depending on time step.

The guaranteed immutability of an ethereum blockchain is one of the key elements for fault tolerance / catastrophe recovery. As in a blockchain in any moment, all nodes will contain an exact replicated copy of the blockchain, several strategies to backup and restore this database could be employed, such as, keeping observer nodes, which increments the robustness of the network; or, perform timely backups of the chaindata information into a separate medium.

Other important factor to mention is that a block, as an element belonging to a canonical blockchain, possesses the feature of being replicable in any node running an EVM. That is, given the set of transactions belonging to the block, and its parent block as an initial state, the transaction results applied over the state whose merkle root is stored in the parent block will be equivalent regardless the installation where the stateless EVM will run. That is, there is no state code, no hidden configurations. Just state and transactions.

Since Parity and Geth/Quorum provides the option to choose different consensus mechanisms they offer different fault tolerances. Here we look at several failure modes and how the different consensus mechanisms deal with them.

Byzantine faults
A PBFT derived consensus mechanism has a Byzantine fault tolerance of ~33% where PoW has a fault tolerance of ~49% and with PoA it is assumed that Authorities will be held accountable for their actions since they would be de-anonymized. In the event that an Authority goes rogue, the longest chain will still win, so although the fault

tolerance (https://github.com/paritytech/parity/wiki/Aura) is 0% when it becomes the turn of a byzantine authority to create a block, the asymptotic long-term fault tolerance reaches around f/n where f is the number of byzantine authorities and n is the number of non-byzantine authorities. This value is obtained because the block maker turn moves in round-robin fashion between authorities at each time step.

Chain fork
In the event of a chain fork/split, the longest chain will be chosen as the valid chain by nodes in the network. This will resolve the fork/split, as long as there is a majority of “good” or “honest” nodes.

Crash faults
All considered consensus mechanisms handle crash faults in a similar manner. Nodes which are stopped and started again will handle this fault by simply re-syncing from other peers (nodes) until they are up to date. Nodes that are removed and replaced will need to sync from scratch, however a copy of another node's blockchain files will speed this process up significantly.

In consensus mechanisms where newly created blocks need to be voted on, taking a voting node offline will have no effect on the network as long as a threshold of voters are online. For example, for a consensus mechanism configured with 10 voters and a threshold of 2, as long as 2 voters are online the network will not be affected by the other 8 voters being offline. Aura has this threshold set to 50%, that is, 50% of validators need to be online.

In consensus mechanisms where a few selected nodes are responsible for building the chain (let's call these chain building nodes), a crash fault of one of these chain building nodes will cause the network to suffer some loss in transaction throughput. The network will still continue and upon detection a (pre-selected, failover) node could be configured to start building on the chain.

Random network delay
There are three forms of network delay that are dealt with, delaying a single random transaction, delaying voting on a newly created block and delaying a newly created block.

Delaying a random transaction will not cause any effect on the network's operation, unless there are transactions building on the delayed transaction. The transactions building on to the delayed transaction will be kept back. Once the delayed transaction is propagated, the transactions building on it will be included into the blockchain.

In consensus mechanisms where newly created blocks need to be voted on, delaying a vote on a newly created block will have no effect on the network as long as a threshold of votes is received. For example, for a consensus

mechanism configured with 10 voters and a threshold of 2, as long as 2 voters are online the network will not be affected by the other 8 voters being delayed. Aura has this threshold set to 50%, that is, 50% of validators need to be online.

Delaying a newly created block will result in a similar situation as with a crash fault of a chain building node where the network will suffer some loss in throughput.

Pluggable Consensus

Is there support for pluggable consensus?

Yes, the two popular clients Parity and Geth support pluggable consensus.

Parity (RUST Client by Gavin Wood, writer of the Ethereum Yellow Paper)
Starting version 1.5.0 released on 20170119

https://github.com/paritytech/parity/releases/tag/v1.5.0

https://github.com/paritytech/parity/wiki/Pluggable-Consensus

- Proof of Work (PoW)
    - Ethash (Original Ethereum PoW)
- Proof of Authority (PoA)
    - Aura
- Voting-based
    - Tendermint

Go-Ethereum (Golang client supported by the Ethereum Foundation)

https://github.com/ethereum/go-ethereum/releases/tag/v1.6.0

- Proof of Work
    - Ethash: Traditional Consensus Algorithm

Proof of Authority

- Clique
    - https://github.com/ethereum/EIPs/issues/225
    - https://blog.ethereum.org/2017/04/14/geth-1-6-puppeth-master/
    - https://github.com/ethereum/go-ethereum/pull/3817
    - https://github.com/ethereum/go-ethereum/pull/3753

Algorithms Available

Which consensus algorithms are available?

Please see above for other details.

The two most popular Ethereum clients, Parity and Geth, support pluggable consensus.
The pluggable consensus algorithms available are implementations of Proof of Work, Proof of Authority, and an experimental implementation of Tendermint.
Proof of Work is a consensus algorithm ideal for public networks that can scale to thousands of machines.
Proof of Authority is a consensus algorithm ideal for networks with trusted block maker authorities. The consensus can be ran with sealing such that blocks are produced even if there are no transactions. This is necessary for fault tolerance. By considering time stepped validator selection, transaction verification and block signing is distributed evenly across validators.

Tendermint is a consensus algorithm that is experimental nature intended for use in networks with somewhat trusted parties where blocks are made based on achieving minimum number of votes on a block's validity. The Tendermint algorithm is not intended to be used in large networks due to the immense communication overhead implied in determining consensus; the communication it uses is akin to the overhead required by PBFT algorithms which limits machines involved in consensus to around 30.

It is becoming common practice to record validators and consensus algorithms on the blockchain, thus the blockchain maintains a record of the protocols and their use, further solidifying the immutability and openness of the blockchain.

Parity:
Aura
https://github.com/paritytech/parity/wiki/Aura

Geth:
https://godoc.org/github.com/ethereum/go-ethereum/consensus

Hyperledger Monax Burrow, based on Tendermint:
https://github.com/hyperledger/burrow
https://github.com/tendermint/tendermint
https://tendermint.com/

Proof of Authority standardization discussion:
https://github.com/ethereum/EIPs/issues/225

Trust Models

Does it make use of existing trust in business networks to improve performance and security?

The public Ethereum network solves the hard problems in the most challenging context: when up to half of the nodes can be malicious and can permissionlessly connect to and arbitrarily disconnect from the network. In a strong governance context in which node owners are known and subject to certain constraints, many assumptions can be relaxed enabling simplifications and optimizations to gain orders of magnitude increases in scalability.

Participation in Consensus

Does it allow limiting consensus to few participants? Give examples of kinds of policies supported?

The Proof of Authority (PoA) consensus algorithm restricts block creation to a set of authorities, which are configured in a client's configuration or in a smart contract. In general, Proof of Authority policies imply that any one of the set is permitted to create blocks. A variation to this general policy is the 'Aura' consensus PoA algorithm, which chooses a new block creator from the set of authorities in round-robin fashion based on a time step. An important property of PoA policies is to perform 'auto-sealing' which is to create blocks even when there are no transactions to include in a block. The reason for auto-sealing is to ensure that no specific authority becomes a leader and thus does not stop other block makers from assuming a block creation while awaiting the previous block maker just in case it has incurred a failure.

4. Developer Support

In this section, we give an overview of the wide range of language support for developing for the Ethereum blockchain, including imperative languages such as Solidity and mostly-functional languages like LLL. Ethereum apps consist of on-chain code as well as off-chain code (such as javascript libraries to interact with the blockchain) and we highlight frameworks used by tens of thousands of developers which simplify end-to-end development and include support for industry standard approaches, such as test-driven development. Included in this section are mentions of research-driven smart contract domain-specific languages.

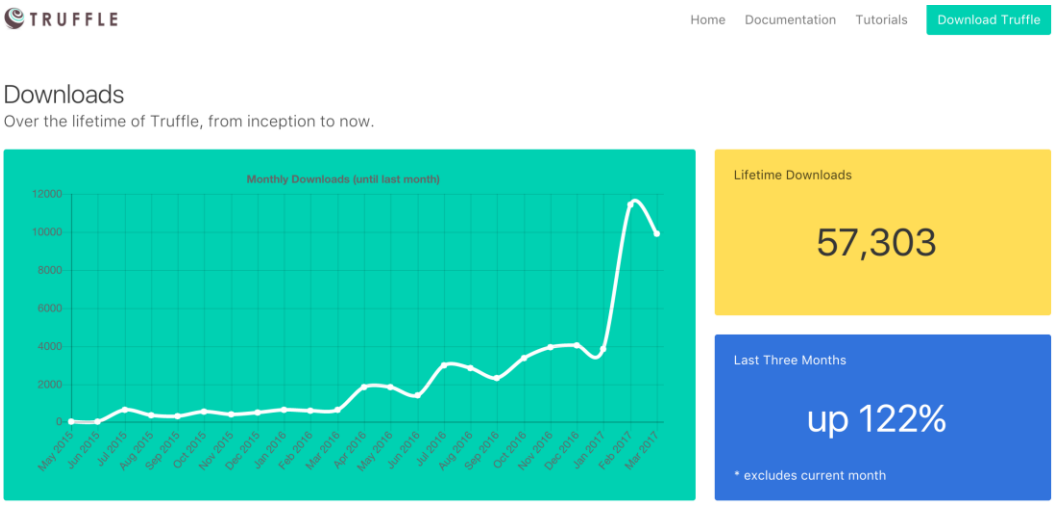
Smart Contract Development

Languages, IDE, Testing, Tools

As part of the process of developing secure smart contracts, ConsenSys has developed a significant collection of smart contract best practices:
https://github.com/ConsenSys/smart-contract-best-practices

Solidity:
"Solidity is a contract-oriented, high-level language whose syntax is similar to that of JavaScript and it is designed to target the Ethereum Virtual Machine (EVM). Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.
As you will see, it is possible to create contracts for voting, crowdfunding, blind auctions, multi-signature wallets and more."
http://solidity.readthedocs.io/en/latest/
https://github.com/ethereum/solidity/

Truffle (ConsenSys):
http://truffleframework.com/
https://github.com/ConsenSys/truffle
http://truffleframework.com/dashboard/



Hive, an Ethereum client end-to-end RPC API test harness:
https://github.com/karalabe/hive/

Dapple (deprecated, moved to Dapp see below):
https://github.com/dapphub/dapple

Dapp (replacement of Dapple):
https://github.com/dapphub/dapp
https://dapp.readthedocs.io/en/latest/

EthereumJS:
https://github.com/ethereumjs
https://blog.ethereum.org/2017/03/21/ethereum-js-ecosystem-updates/

LLL:
https://github.com/zigguratt
https://github.com/ethereum/solidity/tree/develop/liblll

Viper:
https://github.com/ethereum/viper

Serpent:
https://github.com/ethereum/wiki/wiki/Serpent
https://github.com/ethereum/serpent

eWASM (Ethereum flavored WebAssembly):
https://github.com/ewasm

Standards, approaches, and systems being developed for contract and cross-chain development:

Upgrade capabilities:
Smart contracts are stored on the blockchain and since blockchain data is immutable, contracts cannot unergo in-place updates. Instead, smart contracts need to be engineered to integrate references to the latest version of a smart contract.

Versioning capabilities:
Smart contracts do not have any inherent notion of versioning. Instead, if a smart contract that has been deployed needs to be upgraded, a software engineering solution that references the new (updated) contract should be used. At the same time, the older smart contract can still be referenced, such as to support backwards compatibility.

Roadmap can be inferred from the discussions surrounding the core protocol:
https://github.com/ethereum/EIPs
https://github.com/ethereum/EIPs/issues?q=is%3Aissue%20

For example, a universal spec for JSON-RPC API methods:
https://github.com/ethereum/EIPs/issues/217

Inter-protocol capabilities:

(1) BTCRelay, a robust example of inter-protocol interaction between the largest public blockchains. The approach can be leveraged for other forms of interaction.
http://btcrelay.org/
https://github.com/ethereum/btcrelay

(2) https://cosmos.network/

“The Cosmos fundraiser raised
$16.8 million USD and finished on April 6, 2017 6:28AM PDT.”

(3) http://polkadot.io/
https://github.com/polkadot-io/
https://github.com/polkadot-io/polkadotpaper/raw/master/PolkaDotPaper.pdf

App Development Support

Which platforms are supported for developing Blockchain based apps?

Windows, Mac, *nix, browser-based dev supported as well.
No Docker/VM containers are required but they can be leveraged for isolation.

Browser-Only Solidity IDE and Runtime Environment

https://github.com/ethereum/browser-solidity

Model Driven App Development

Does tooling include a model driven/declarative approach for programming?

We are in the process of initiating a Business Process Modeling model-driven solution based on a business process engine and designer that has been proven within production environments using Activiti.
https://www.activiti.org/

PaaS Support

Which commercial PaaS platforms provide support for the protocol?

Many popular and robust cloud providers support the blockchain protocol, with proven support by Microsoft Azure, Amazon AWS, Digital Ocean, Red Hat Open Shift and Pivotal cloud foundry.

Community / Ecosystem

Size of Dev community/Membership of Consortium

Enterprise Ethereum Alliance launch members:

There are many client implementations, implying a lot of client developers: https://entethalliance.atlassian.net/wiki/display/EEA/Ethereum+Client+Implementations

For further information regarding the support within different Ethereum clients: http://cdetr.io/eth-compat-table/

Live public network stats (covers a subset of nodes on the public network): https://ethstats.net/

Websites

- Ethereum StackExchange (5k+ questions, 6.6k users)
    - https://ethereum.stackexchange.com/questions
- Reddit Ethereum (34k+ subscriptors)
    - https://www.reddit.com/r/ethereum/
- Github (3k+ repositories mentioning ethereum)
    - https://github.com/search?utf8=%E2%9C%93&q=ethereum&type

5. Data Layer

In this section, we describe how Ethereum blockchain data is managed at the smart contract level, how it is managed within the Ethereum client by a local key-value store, and how blockchain data can be emitted and ingested into commonly used relational/key-value for analytics.

Supported Databases

What is the support for popular/mainstream database technologies?

Currently, the best practices approach to delivering data into the blockchain is via push messages that update the state of data/code, controlled by the blockchain security best practices.

Event logs are generated either during the creation of a block, or during the full synchronization of a node (that is, replaying every single transaction for verification purposes). While these event logs are stored into the selected Key-Value store of the node, there are not any optimization aiming to improve lookup times of there logs.

Currently, developers have two work arounds for this matter:
a) An off-chain process running parallel to the ethereum node will update on every block its relational database, which will be heavily indexed on topics and payloads for its querying afterwards.
b) This very database mentioned above can be fed by a daemon, which will perform RPC queries per block, transaction and transaction receipt to get the contains of the logs.

Alternatively there are projects, like **ConsenSys' aleth.io** tasked with providing analytics for the blockchain. Even if the developed ontologies are created, stored and consumed in big data friendly databases. Its core contents are dump from the native key value store into a relational database for ease of querying.

Types of Queries

What are the type of queries supported on the ledger?

The Ethereum specification defines two types of data that can be recorded within a smart contract: normal storage, and logs. Normal storage is available synchronously to smart contracts, and logs are more easily queried externally by API.

Normal storage is used for queries within a smart contract, and includes a variety of types including maps and arrays, which enable key and index lookups. Currently the common pattern to look up smart contract storage is to define getter methods on the contract, and query them via calls to the RPC API.

Logs can have a variety of fields, and are easily queried by the default RPC API, including querying by value, ranges, inclusion within sets of values, and more.

http://solidity.readthedocs.io/en/develop/contracts.html#events
https://media.consensys.net/technical-introduction-to-events-and-logs-in-ethereum-a074d65dd61e

For applications that require highly advanced query methods, there are several other approaches to providing the API that the application developer requires.

The most basic way of extending the default RPC API is to add a caching layer that runs parallel to a blockchain node, and keeps relevant info up to date in a relational or similar database.

There are also several projects under active development that are creating new methods of querying the Ethereum blockchain in different ways.

The IPLD project (used by IPFS) defines a method for defining string paths for querying hash-linked data, like a blockchain, and has a basic set of data bindings working, allowing lookups of all blockchain primitives, and soon will be extended to support querying contract storage as well.

Consensys also has a project under development called Ethon, which formally defines an Ontology for Ethereum entities, and will allow graph-database querying of Ethereum entities and contract data, using graph query languages like SPARQL.
http://ethon.consensys.net/EthOn_spec.html
https://github.com/ConsenSys/EthOn

Support for Custom Events

Does the ledger support notifications to client apps? Does it allow custom events do be defined?

https://github.com/paritytech/parity/wiki/Aura
https://github.com/paritytech/parity/wiki/Tutorial-Part-VIII

As discussed above, the Ethereum blockchain specifies a data type called an Log or Event, which is not stored on-chain, but is designed for triggering external

events, like client notifications and state updates. By using the event listening API, an external application can easily respond to any blockchain event.

Analytics Support

How does the ledger support running on-chain, off-chain analytics?

Blockchain ledger technology with transparent operations (including Ethereum) can be consumed and analyzed by any network participant. Typically analytics of (non-finalized) state requires nodes geographically distributed to gain topographical knowledge of the ledger's transient states. Analytics can be performed in real-time off-chain and combined with real-world data in a manner that influences on-chain activity.
http://blockchainers.org/index.php/2017/02/13/blockchain-streaming-analytics-smart-distributed-applications/

Incorporating analytics on-chain may utilize a mechanism such as TrueBit. TrueBit (https://medium.com/@chriseth/truebit-c8b6a129d580) offers a framework for performing and validating computationally-intensive tasks on behalf of on-chain contracts.

The most common type of Ethereum analytics tools are block explorers, and there are many out-of-the-box products for this task, including multiple being developed within Consensys.

6. Pluggability

In this section, we give an overview of the modularity and pluggability of different components of the Ethereum client, such as the consensus algorithm, data storage/database, crytography, user management, and network protocols.

Consensus Algorithm

Is consensus pluggable?

Yes, see above.

Ledger Database

Is the storage layer abstracted?

Ethereum clients rely on the LevelDB key/value store, which has been abstracted into a library which enables getting and putting a value corresponding to a key. The result is that the storage layer has a simple abstraction.
While go-ethereum has not provided yet the community with the option of pluggable database, it is worth noting that it has been built with certain modularity in mind regarding key-value stores, giving the possibility to a developer team to replace the current LevelDB implementation with a K-V store of their convenience.

ETHDB Module
https://github.com/ethereum/go-ethereum/tree/master/ethdb

Interface (currently supports levelDB and a for-test memory DB)
https://github.com/ethereum/go-ethereum/blob/master/ethdb/interface.go

https://github.com/ethereum/go-ethereum/blob/master/ethdb/database.go

Cryptographic Libraries

Is it possible to use different different crypto implementations?

Currently this is tied to the Keccak256 family of algorithms but can be refactored to support more general key signing and verification methods.
https://github.com/ethereum/go-ethereum/tree/master/crypto

Membership Service

Is the membership service pluggable?

Yes, there is implicit support for different members and their roles. These are managed via smart contracts, thereby leveraging the benefits of distributed, secure, and consensus-layered blockchains.

Currently, membership can be set up in two ways:

The first one involves network topology / middleware, which is related with the access a node has to participate in the network. This condition has not necessarily to be binary (as is able or disabled to participate), but can have different degrees of involvement based on the filters that can be configured at the layer 7 in the middleware (i.e. Restricting transactions to pass, unless they are from a specific address. Enable the node to synchronize and ask via RPC for the last block, but disable the ability to send raw transactions with certain payloads).

The second one involves encoding such accesses inside smart contracts deployed in the very blockchain, this approach allows for a more rich configuration, as, for example, enables the existence of ranks among the peers. Also, legacy permission systems, such as LDAP can be integrated using this approach in a non-cumbersome way.

The following libraries are examples of such a membership service:

Simple access control pattern for Ethereum
https://github.com/dapphub/ds-auth

Whitelist authority for use with DSAuth
https://github.com/dapphub/ds-guard

Your dapp can have 256 roles. A user can have multiple roles. Roles define access to capabilities.
https://github.com/dapphub/ds-roles

P2P Protocols

Is it designed to use different mechanisms for information dissemination between nodes?

The Ethereum network currently uses RLPx, a cryptographic peer-to-peer network and protocol suite which provides a general-purpose transport and interface for applications to communicate via a p2p network. RLPx is designed to meet the requirements of decentralized applications and is used by Ethereum. https://github.com/ethereum/devp2p/blob/master/rlpx.md

In the current Ethereum client implementations, the RLPx network protocol is intrinsic to the way information is disseminated between nodes. The fact is that the protocol demonstrates robustness and security at public network scale and connects thousands of nodes on the public network. This same robustness and security is ideal for private networks and chains.

Enterprises have stated that configurable privacy, pluggable consensus, and scalability are key short term goals that enterprises need. Even with these stated goals, the dependency on the RLPx protocol has been discussed amongst members of the Enterprise Ethereum Alliance. The outcome of the discussions has been that due to the robustness and security of this protocol, and considering the stated priorities, the RLPx already satisfies these properties since the protocol is encrypted (thus private), can be used to communicate with any consensus protocol (as is evidenced by the pluggable consensus protocols that Ethereum clients support including PoW and PoA), and the protocol is scalable to thousands of networked nodes (and can also support thousands more nodes on the network). Furthermore, if a discussion of the modularization of this networking component is to be raised, the discussion must be entered with an understanding of what alternative capabilities are required, and which needs further feedback and suggestions from enterprises.

Some Ethereum clients that only support private networks implement p2p protocols with looser properties, such as not including encryption. The reason for this is that p2p protocol needs on a private network, which rely on a unique consensus algorithm, will make different assumptions about the nodes on the network. For example, in such private networks the assumption of trust is made intrinsic between peers. However, such implementations are also tied directly to the consensus algorithm and the p2p protocol is not pluggable.

7. Governance / Business

In this section, we describe at a high level how the governance of the Ethereum is managed by the core developers and the enterprises interested in creating specifications that standardize technologies such as for privacy and scalability.

Enterprise Adoption

Provide some public references of the Blockchain's use in Enterprises.

Amongst the use cases present in media, we highlight the South African Financial Blockchain Working Group has been building an Ethereum Network since August 2016 and have completed various tests on it. There are 9 different financial organisations running 15 nodes on the network.

http://www.coindesk.com/south-africas-biggest-financial-power-players-just-went-blockchain/

Ethereum has been available for use to Enterprise-level customers via Microsoft Azure beginning in 2015 (https://azure.microsoft.com/en-us/blog/ethereum-blockchain-as-a-service-now-on-azure/).

These efforts are generally in the partner-driven proof-of-concept phase (https://azure.microsoft.com/en-us/blog/blockchain-basics-partner-strategy-azure/).

Open Platform and Governance

Is it an open platform? What is the governance model?

EntEth1.0 will be licensed under and Apache2 / MIT licensing family.

From http://spectrum.ieee.org/tech-talk/computing/networks/enterprise-ethereum-alliance-launches:

“Since it arrived, Ethereum has been run as an open-source project, with no formal governance structure.”

The recently-formed Enterprise Ethereum Alliance “will seek to add some formal governance to the development process”.

All Core Devs Meetings discussing Core Ethereum Specification Roadmap https://github.com/ethereum/pm/tree/master/All%20Core%20Devs%20Meetings

Identity, Trust, and Private Data

uPort is a secure, easy-to-use system for self-sovereign identity, built on Ethereum. The uPort technology consists of three main components: smart contracts, developer libraries, and a mobile app.

uPort identities can take many forms: individuals, devices, entities, or institutions. Uport identities are self-sovereign, meaning they are fully owned and controlled by the creator, and don't rely on centralized third-parties for creation or validation. A core function of a uPort identity is that it can digitally sign and verify a claim, action, or transaction - which covers a wide range of use cases.

An identity can be cryptographically linked to off-chain data stores. Each identity is capable of storing the hash of an attributed data blob, whether on IPFS, Azure, AWS, Dropbox, etc., which is where all data associated with that identity is securely stored. Identities are capable of updating this file themselves, such as adding a profile photo or a friend, or they can also grant others temporary permission to read or write specific files.

Since they can interact with blockchains, uPort identities can also control digital bearer assets such as cryptocurrencies or other tokenized assets.

More details are available in the uPort whitepaper (http://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf).

Governance as a Service

Blockchain innovations have the potential to completely re-write governance, as reflected in https://corpgov.law.harvard.edu/2016/01/06/corporate-governance-and-blockchains/.

BoardRoom is a governance mechanism built on the blockchain in smart contracts. BoardRoom is structured to accommodate next-generation collusion-resistant technology including secret ballots, delegate voting and stake voting. BoardRoom provides a secure apparatus for organizational decision making on the blockchain incorporating the benefits to governance afforded by the Ethereum consensus system.

Additional Items not included in questionnaire:

Comment from Dr. Sohail in email:

“In addition I would like to get your detailed input into what Protocol (including version) you would implement in the Dubai Production Blockchain environment, if that deployment with be from a standard distribution or if certain modifications would be made and if so what would be the plan for making those modifications.

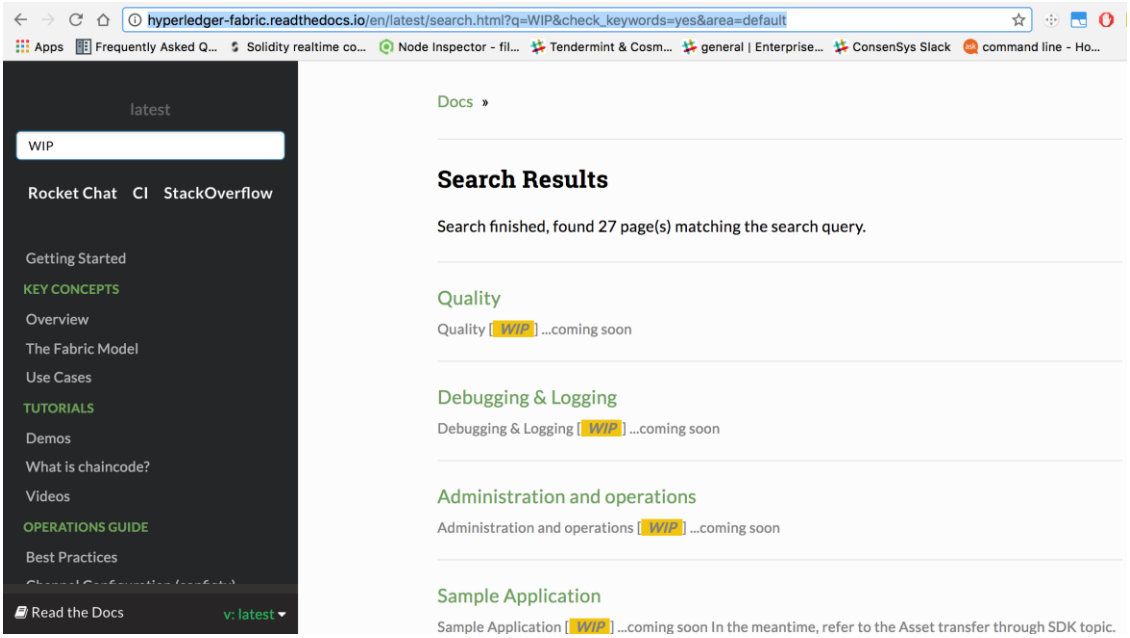
I would also like to know the technology roadmap as to what technical features are available in the current distributions and what features and functionality is already planned and in roadmap for the near future.

I would also like your input in the Infrastructure required for production implementation and what impact will the load and performance have on these recommendations.”

Notes on Hyperledger Fabric

Good insight into HL Fabric roadmap (and thus current limitations) that describes in detail that Fabric is still in alpha stage development and has not yet reached v1.0:
https://www.altoros.com/blog/general-availability-of-hyperledger-fabric-v1-0-what-to-expect-in-2017-and-when/

Hyperledger Fabric documentation is inadequate, missing many key concepts, and incomplete in general:
http://hyperledger-fabric.readthedocs.io/en/latest/search.html?q=WIP&check_keywords=yes&area=default



Chaincode

Chaincode [*WIP*] The widely-used term, smart contract, is referred to as “chaincode” in Hyperledger

Consensus

Consensus [*WIP*] Not to be conflated with the ordering process. Consensus in v1 architecture is a

Multichannel

Multichannel [*WIP*] The fabric will allow for multiple channels with a designated ledger per

Ordering Service

Ordering Service [*WIP*] ...coming soon This topic will outline the role and functionalities of the

Security Model

Security Model [*WIP*] Hyperledger Fabric allows for different organizations and participants in a

What is chaincode?

What is chaincode? [*WIP*] coming soon ... end-to-end examples of chaincode demonstrating the

Deployment:
Challenges when developing for HL Fabric:
Using the simple inter-protocol approach exemplified in:
https://github.com/hyperledger/fabric/blob/master/examples/chaincode/go/utxo/README.md

Need to build a docker container to host the code that is to be deployed, and this has to be repeated upon every code update.
Need to build a new peer container to host the code, and this has to be repeated upon every restart in order to start from a clean slate.
Need to start docker container and wait for each code and peer container to finish booting.
Need to wait for code to be deployed, HL Fabric suggests 30 seconds.
Need to get the appropriate container identifier from Docker via command line after boot is complete.
Need to issue a deploy command in order to deploy code from code container to peer container, and requires careful use of command line options to specify the container identifer; this container identifier is not guaranteed to be static.
Tedious repetition is needed if private key changes, or to have a clean slate between testing iterations.

Fabric Consensus (support not complete, currently only relies on Kafka ordering service, which is not consensus at all):
https://jira.hyperledger.org/browse/FAB-37

Ver 0.60 or earlier is not compatible with version 1.0alpha which was released a few weeks ago and is still not fully developed. There are lots of bugs and 'return nil' throughout

it is deployed as a 'chaincode' and interacting between chaincodes requires knowing the chaincode identifier. Each chaincode is deployed in a docker container and only the state is kept in the ledger, the code itself may or may not exist and may even change/be deleted.

Chaincodes communicate over channels (using gRPC), and each channel has a unique identifier. The data sent between the ledger and chaincode is marshalled and unmarshalled through that channel. This implies that there is a huge performance cost due to the serialization and deserialization of data. Since there are not costs associated with contract/data size, this can cause a major performance detriment, increase computational, storage, and network requirements.

The chaincodes timeout after a fixed period. There is no gas costing, and a chaincode is not analyzed at all before deployment, so permissions on the

docker container need to be enforced by platform admin to ensure "no funky business". Each chaincode has a version number, and that is used to interact with the state that is maintained on the ledger. There is no enforcement however that a chaincode use the state that is "get"-ed from the ledger when a chaincode it is invoked, essentially becoming a "hidden" business logic. Similarly, there is no enforcement that state is "put" back into the ledger in a correct way. When interacting with a chaincode, only the latest version is used by default, using an earlier version is disabled in the code. Interacting with an earlier version wouldn't be guaranteed to work since it depends on the code existing. In any case, each new deployment requires its own docker container afaik so good luck doing development with that. You have to install the chaincode to a docker container, you have to launch a peer docker container, then deploy the chaincode into the peer, waiting in between each of these steps to get the correct docker-image id, only then can you invoke a transaction. And if you want to reset your app, repeat above said steps!

If a peer is reliant on X509 certificates, and it has changed, then the container needs to be redeployed.
https://github.com/hyperledger/fabric/blob/599a7fa00927e526ad117a569f1afe7d7e0b2819/core/chaincode/chaincode_support.go#L348

Access Control Lists are not supported in the current implementation. This means that the only way to ensure membership and control of chaincode lifecycle such as deployment, invocation, upgrade actions are left to platform admins:
https://github.com/hyperledger/fabric/blob/f477ccc7e74c44f1ab0d217a1b06df072d68d76a/core/chaincode/handler.go#L247

Confidential transactions are not supported:
https://github.com/hyperledger/fabric/blob/f477ccc7e74c44f1ab0d217a1b06df072d68d76a/core/chaincode/handler.go#L263

Chaincode timeout handler (since gas is not used):
https://github.com/hyperledger/fabric/blob/f477ccc7e74c44f1ab0d217a1b06df072d68d76a/core/chaincode/handler.go#L1346

In addition to Hyperledger Fabric's lack of control on the type/size/libraries used in deployed code, it further does not prevent deployed code from deleted completely, and does not maintain any history of the deployed code - this is a key flaw in their approach that completely breaks one of core tenets of the blockchain ethos.
https://github.com/hyperledger/fabric/blob/master/core/scc/lccc/lccc.go#L526

Immutability capabilities:

Due to the Hyperledger Fabric approach of deploying arbitrary code, without any specific limitations such as code and libraries, there is an immense amount of trust placed on the developer to ensure that the intended security model is adhered to. Its current approach is to check the version number and naming convention, and does not limit at all the operands that the developer chooses to use. This means that the platform must doubly ensure that the security is appropriate for the use case. It further means that if permissions are somewhat loosened, there can be leaks in the data/network and may allow a developer within a network

It also increases the demand on security maintenance so that policy controls are always needed over loose and undefined developer needs. In contrast, the Ethereum blockchain has a well-defined approach with deterministic contract execution which does not make callouts to untrusted parties/code/data.