Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network)

Harish Sukhwani
Duke University
Durham, NC, USA
hvs2@duke.edu

Nan Wang
Duke University
Durham, NC, USA
wang.nan@duke.edu

Kishor S. Trivedi
Duke University
Durham, NC, USA
ktrivedi@duke.edu

Andy Rindos
IBM Corporation
RTP, NC, USA
rindos@us.ibm.com

***Abstract*—Hyperledger Fabric (HLF) is an open-source implementation of a distributed ledger platform for running smart contracts in a modular architecture. In this paper, we present a performance model of Hyperledger Fabric v1.0+ using Stochastic Reward Nets (SRN). From our detailed model, we can compute the throughput, utilization and mean queue length at each peer and critical processing stages within a peer. To validate our model, we setup an HLF network in our lab and run workload using Hyperledger Caliper. From our analysis results, we find that time to complete the endorsement process is significantly affected by the number of peers and policies such as `AND()`. The performance bottleneck of the ordering service and ledger write can be mitigated using a larger block size, albeit with an increase in latency. For the committing peer, the transaction validation check (using Validation System Chaincode (VSCC)) is a time-consuming step, but its performance impact can be easily mitigated since it can be parallelized. However, its performance is critical, since it absorbs the shock of bursty block arrivals. We also analyze various what-if scenarios, such as peers processing transactions in a pipeline, and multiple endorsers per organization.**

***Index Terms*—blockchain; hyperledger fabric; model validation; performance modeling; Stochastic Reward Nets;**

I. Introduction

According to National Institute of Standards and Technology (NIST) [1], "Blockchains are immutable digital ledger systems implemented in a distributed fashion (i.e., without a central repository) and usually without a central authority." Blockchain network enables trusted parties to send transactions in a peer-to-peer fashion in a verifiable way, without the need of a trusted intermediary. It allows parties to settle transactions quicker, resulting in faster movement of goods and services [2]. Thus a cryptocurrency such as Bitcoin is an application running on the blockchain network.

Hyperledger project is an open source collaborative effort hosted by the Linux Foundation to advance blockchain technologies for business enterprises. In our research, we focus on Hyperledger Fabric release v1.0+ [3] (termed *Fabric V1* or simply *Fabric*). It is currently deployed in more than 400 proof-of-concept and production distributed ledger systems across different industries and use-cases [4]. As opposed to a public blockchain network such as Bitcoin or Ethereum where anyone can join the network, HLF is a permissioned blockchain network where participants know and identify each other but do not fully trust each other. Thus organizations can benefit from a distributed ledger technology (DLT) without a need of a cryptocurrency [5].

As the HLF project is evolving and maturing, it is imperative to model the complex interactions between peers performing different functions. Such models provide a quantitative framework that helps compare different configurations and make design trade-off decisions. In this paper, we present a performance model of Fabric V1 using Stochastic Reward Nets (SRN) to compute the throughput, utilization and mean queue length at various peers as a function of various system parameters. Just like multiple subsystems of the HLF are "pluggable", we ensure the corresponding submodels are pluggable as well. With this model, we can ask various what-if questions, such as

1) How does the throughput, utilization and mean queue length vary at each peer with an increasing transaction arrival rate?
2) If the peer validates transactions in a pipeline as opposed to batches, will the overall system throughput increase?
3) If there are multiple endorsing peers per organization, how much performance speed-up do we get?

The research contributions of this paper are as follows.

1) A comprehensive performance model of the Fabric V1 blockchain network. For Fabric's unique blockchain network architecture, it captures the key steps performed by each subsystem as well as interactions between them.
2) Analysis for critical what-if scenarios that system developers and practitioners care about.
3) Validate the model with a multi-node experimental setup.

II. System Description - Hyperledger Fabric V1

Fabric V1 introduces an *execute-order-validate* architecture [3], which is a fundamental shift from the traditional order-execute design followed by other blockchain platforms, including HLF's preview release (called v0.6) [6]. The goal is to separate transaction execution (via smart contract (SC)) from transaction ordering. Compared to the traditional state-machine replication approach [7], this architecture provides better scalability, new trust assumptions for transaction validation, support for non-deterministic SC, and modular consensus implementations [3], [5].

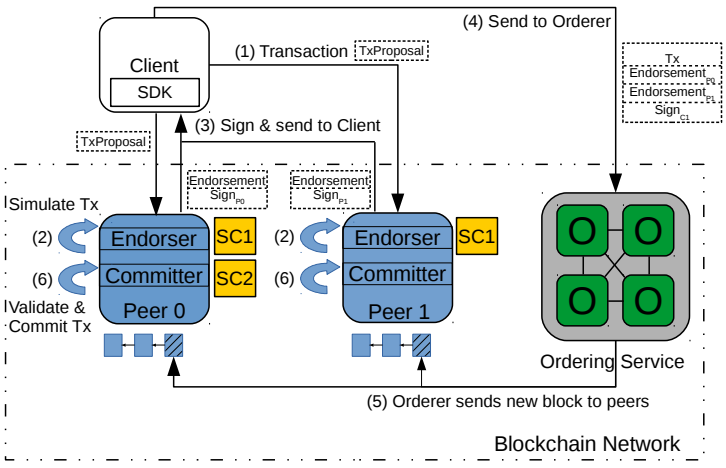

Fig. 1. Transaction flow on Hyperledger Fabric V1 with 2 peers and ordering service running a single channel

In Fabric V1, the *peers* execute transactions and maintain the distributed ledger. The *orderers* order all the transactions in the network, propose new blocks and seek consensus. The collection of orderers form ordering service. By default, all peers are *committers*, thereby receiving ordered state updates in the form of a block of transactions from the ordering service, and maintaining the ledger. Upon receiving a new block, the peer validates the transactions, commits the changes on the local copy of the ledger and appends on the block on the blockchain. Peers can take up an additional responsibility of endorsing transactions, thereby called *endorsers*. An endorser simulates the transaction by executing the smart contract (SC) (called *chaincode* in HLF) and appending the results with its cryptographic signature (called endorsement) before sending it back to the client. Note that a single peer node can be both an endorser and committer.

Fabric maintains a global state across all the peers using a versioned key-value store (KVS) and the ledger. The KVS shows the latest state for the transactions, which can be read by the chaincode using `get()` operation and state updates are proposed by transactions using `put()` operation. The version numbers increase monotonically. KVS is partitioned by chaincode. The ledger is an ordered hashchain of blocks of all transactions, thus providing a verifiable history of all state changes. Both the key-value store and ledger are updated privately by each peer.

In a Fabric network, a subset of peers that want to conduct business transactions privately can form a *channel*, thus maintaining a separate partition of key-value store and ledger. In our analysis, we consider a single-channel Fabric network.

To execute a transaction, the client needs a signature from all the peers as defined by the *endorsement policy*. Endorsement policy is a reflection of the business logic, which can be described using an arbitrary boolean logic using AND, OR and k/n (more details in Section VII-A).

Let us consider a Fabric network in Figure 1. In this example, each peer represents an organization (say Org0 and Org1). The flow of transactions in Fabric V1 is described as follows

1) The client sends a *transaction proposal* (*TxProposal*) to the peers defined by the endorsement policy (all running the same smart contract, say SC1), consisting of the clientID, payload, and transactionID, along with a cryptographic signature of the client on the transaction header. The payload contains the chaincodeID, operation and input parameters.
2) Each peer simulates the transaction execution by invoking the corresponding SC (say SC1 in this case) with the user inputs against the local key-value state. The SC runs in a container isolated from the peer. After simulation, the endorser produces the *read-set* which represents the version numbers of keys read by SC, and *write-set* which represents the key-value pairs updated by the SC.
3) Each peer sends to the client the *endorsement* message containing the result, read-set, write-set, and metadata, where the metadata includes transactionID, endorserID, and endorser signature. This message is signed by peer's cryptographic signature.
4) The client waits for endorsements from peers until it satisfies the endorsement policy of the transaction and verify that the results received are consistent. The client then prepares the *transaction* containing payload and the set of endorsements received from the endorsing peers and sends it to the ordering service. This operation is asynchronous, and the client will be notified by its peer when the transaction is successfully committed.
5) After a few seconds (called *block timeout*) or after a set number of pending transactions (called *block size*), the ordering service creates a block of the pending transactions, maintaining order by timestamp. The block is appended with a cryptographic signature based on the block header and then broadcasted to all the peers on the same channel.
6) When the peer receives a block of transactions, it evaluates transaction endorsements against its endorsement policy in parallel (using Validation System Chaincode (VSCC)). The ones that fail are marked invalid. Next, for each valid transaction, it performs multi-version concurrency control (MVCC) [8], [9] (called read-write check in [3]), which means it serially verifies if the read-set version matches the current version on KVS (assuming the previous transactions are committed). The validity of transactions is captured as a bit mask and appended to the block before the block is appended on the local ledger. Finally, all the write-sets are written to the local KVS, and the state transition is thus completed. The peer notifies the client about the success or failure of the transaction.

Thus each transaction in Fabric V1 undergoes three phases: endorsement, ordering, and validation.

III. Performance Metrics

In this paper, we model Fabric's performance at a DLT system level. Hyperledger Performance and Scalability Working

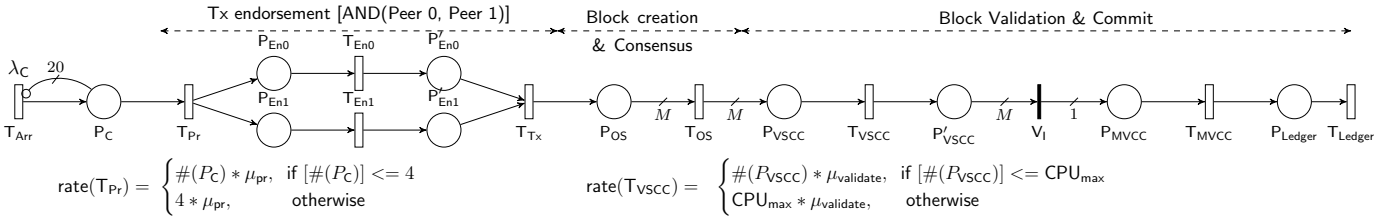

Fig. 2. SRN model of Hyperledger Fabric V1 network

Group[1] recently published the first release of the performance metrics document [10], that attempts to provide crisp and clear performance metrics applicable across different DLT platforms. We borrow and refine the definitions for our models and analysis.

Transaction throughput: Transaction throughput is the rate at which the blockchain network commits valid transactions in the defined period of time. It is presented in number of transactions per second (tps). For a single-channel Fabric network, in both our model and experimental analysis, we consider the measurement at a single peer. Note that a transaction here corresponds to the chaincode "invoke" and not "query." Since only *valid* transactions are considered, throughput is the same as goodput.

Transaction latency: Transaction latency is the time taken between when the transaction is submitted and when the transaction is confirmed committed across the network. Unlike lottery-based consensus protocols such as Bitcoin or Ethereum where the transaction finality is probabilistic, the consensus process for Fabric results in deterministic finality and hence this simple definition. In our model and analysis, we check the transaction confirmation at a single peer. The end-to-end latency consists of three latencies: endorsement latency, ordering latency and commit latency [9].

Queue length: Queue length of a node is the number of jobs waiting for service or in service at that node. In our model, we compute the mean queue length at the ordering service, at each endorsing peer, and each processing stage of a committing peer.

Utilization: Utilization of a node is a percentage of time the node is busy. In our model, we compute the utilization of each endorsing peer and each processing stage of a committing peer. For operations that multi-thread, the utilization corresponds to the average utilization across all logical processors.

IV. SRN MODEL OF HYPERLEDGER FABRIC V1

In this section, we present our performance model of Fabric using a Stochastic Petri Nets modeling formalism known as Stochastic Reward Nets (SRN) [11]. Such a formalism allows a concise specification and an automated generation/solution of the underlying (stochastic) process that captures the performance behavior of the blockchain network system. Moreover, using SRNs allows us also to study different scenarios, by easily adding or removing system details. SRNs have been successfully used to model different computer/communication systems from the performance perspective [12]. We solve the models using the Stochastic Petri Net Package (SPNP) [13].

The SRN model for a single-channel Fabric network with one client, two endorsing peers (`AND()`) and one peer running the validation logic is shown in Figure 2. Transaction requests follow a Poisson arrival process with rate λ_C. Max. pending requests are capped to the no. of threads in workload generator (20 in our case). Client prepares the endorsement request and sends it to the endorsing peers (say peer 0 and peer 1) (transition T_{Pr}, which includes transmission time). Peers endorse the transaction (transitions T_{En0} and T_{En1} respectively). When the client receives a response from both peers, the client sends the endorsed transaction to the ordering service (transition T_{Tx}), indicated by a token deposited in place P_{OS}. After *block size* pending transactions (denoted by M), a block of transactions is created and delivered to the committing peers (transition T_{OS}). The committing peer first performs a VSCC validation all the transactions in a block in parallel, limited by the number of logical processors (cores/vCPUs) in the peer (CPU_{max}). Then it performs MVCC validation for all transactions serially (transition T_{MVCC}). Finally, all transactions in the block are written to the local copy of the ledger (transition T_{Ledger}). Both transitions T_{MVCC} and T_{Ledger} are captured at a block level. For reasons discussed in Section VII-B, we do not consider *block timeout* in this model. Note that we implicitly assume that all transactions are of the same complexity and independent of each other. In summary, the three phases of a Fabric transaction can be seen in the SRN model.

We obtain the metrics from our model as follows. The throughput of a transaction phase corresponds to the rate of the corresponding transition, using function `rate()` in SPNP [14]. E.g., the rate of transition T_{Ledger} signifies the block throughput of the system (multiply by M to obtain transaction throughput). The utilization of a transaction phase is computed by the probability that the corresponding transition in SRN is enabled, using function `enabled()`. For transitions with function-dependent marking rate (such as T_{VSCC}), the average utilization across all logical processors is computed using reward functions. The mean queue length of a transaction phase can be obtained by the number of tokens in the corresponding phase, using function `mark()`. E.g., the mean number of tokens in place P_{OS} signifies the mean queue length at the ordering service.

[1]https://lists.hyperledger.org/g/perf-and-scale-wg

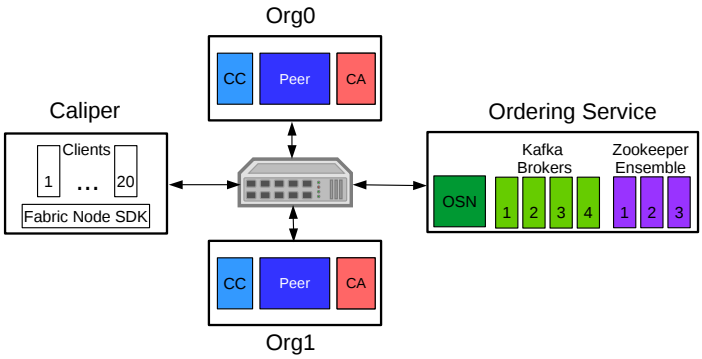

Fig. 3. Hyperledger Fabric V1 network setup

V. Empirical analysis for model parameterization

We parameterize our model using data collected from a Fabric network setup in our lab. Unlike the research summarized in Section IX-B, the goal of this work is not to benchmark the performance by stress loading the system; instead we take measurements from a Fabric network subjected to a realistic traffic pattern. In this section, we provide details of the tools used, network setup, and our methodology for data collection and analysis.

A. Network Setup

The deployed network is shown in Figure 3. Each node is launched as a Docker container and then connected in a network using the Docker Swarm[2]. Although peers and orderers can run natively on the physical/virtual machine, a network of Docker containers is a recommended approach for deploying Fabric networks [15]. Containers corresponding to each organization (peer and ca) are run on an independent physical node ('Org0', 'Org1'). Peers executes chaincode in a separate container (called CC). All containers corresponding to the ordering service run in a single physical node ('Ordering Service'). It includes one ordering service nodes (OSN), 4 Kafka brokers, 3 ZooKeeper nodes. Hyperledger Caliper is deployed on a separate physical node, with multiple client threads that interact with the locally installed Fabric Node.js SDK[3]. We provide the detailed steps for setting up the Fabric network here[4]. Note that we consider a network with one peer per organization, and hence we ignore the implication of the gossip protocol on the network performance.

Physical machines corresponding to Org0, Org1, and Caliper have 4 CPUs (1 socket, 4 core) (Intel Xeon 2.2 GHz) with 12GB RAM, and that running ordering service has 16 CPUs (2 sockets, 4 cores, 2 hyper-threads) Intel Xeon 2.4 GHz with 32GB RAM. Each machine is running Ubuntu 16.04 LTS on a 7200 rpm Hard Disk Drive with Fabric release v1.1[5] installed. All physical machines are connected with a 1 Gbps switch. All nodes are synchronized using Network Time Protocol (NTP) service so that transaction latency can be measured across nodes. Communication between all nodes is configured to use Transport Layer Security (TLS).

To execute workload, we use the Hyperledger Caliper[6], recently approved as a Hyperledger project. It is a benchmark execution platform that enables the user to measure the performance of different DLT platforms consistently. An advantage of using this tool is that it takes care of the complex workflow performed by the client (using Fabric SDK) including handling of event notifications from the peer. To generate traffic following a Poisson arrival process, we implemented a new rate-control function in Caliper. A current limitation of Caliper is that it supported interaction with only one OSN node.

B. Test application

For our performance testing, we leverage the *simple* chaincode provided by the Caliper tool and extend it for our needs. This application maintains account balances for users. It can perform two functions. Function 'open' checks if an account exists, and if not, create a new account and assigns it an account balance. Thus it performs one read, one write operation to the key-value store. Function 'transfer' allows transfer of money from one account to another. Thus it performs two read, two write operations. Before running 'transfer' transactions, we run 'open' transactions for a few mins. to load up the key-value store. Authors in [9] also followed a similar approach of differentiating workloads by the number of read-write operations. For both the functions, the input account number(s) are selected randomly; hence there is almost no dependency between consecutive transactions; thus the transactions never seem to fail the MVCC validation. This way, we can easily generate a workload of all valid transactions at a high rate[7].

C. Measurements

We measure the time taken to perform critical steps in the transaction life-cycle by analyzing the caliper output files, peer logs, and orderer logs. Figure 4 shows the transaction sequence diagram along with the venue where the timestamp is captured and a snapshot of the log entry in peer/orderer. We converted these vital log entries from DEBUG mode to INFO mode. To measure the mean queue length at each transaction life-cycle phase, we add additional log entries to capture the time when a new transaction enters and leaves that phase. The weighted time-average of the no. of transactions/blocks gives the mean queue length in that phase [16]. We also increased the timestamp resolution to microseconds. We measured the overhead of additional logging for a setup with block size 500 and $\lambda_C = 100$, and observe only a 0.64% increase in the average transaction latency, and 0.48% increase in 75%ile latency. Our Fabric source code changes can be found here[8].

D. Measurement Campaign

We run Caliper with 20 client threads and test duration of 240 sec., ensuring that the transaction arrivals follow a Poisson

[2]https://docs.docker.com/engine/swarm/
[3]https://github.com/hyperledger/fabric-sdk-node
[4]https://bitbucket.org/hvs2/fabric-perf-model/src/master/network_setup
[5]commit id 523f644
[6]https://www.hyperledger.org/projects/caliper
[7]https://lists.hyperledger.org/g/perf-and-scale-wg/topic/17550808
[8]https://bitbucket.org/hvs2/fabric-perf-model/src/master/diff_files

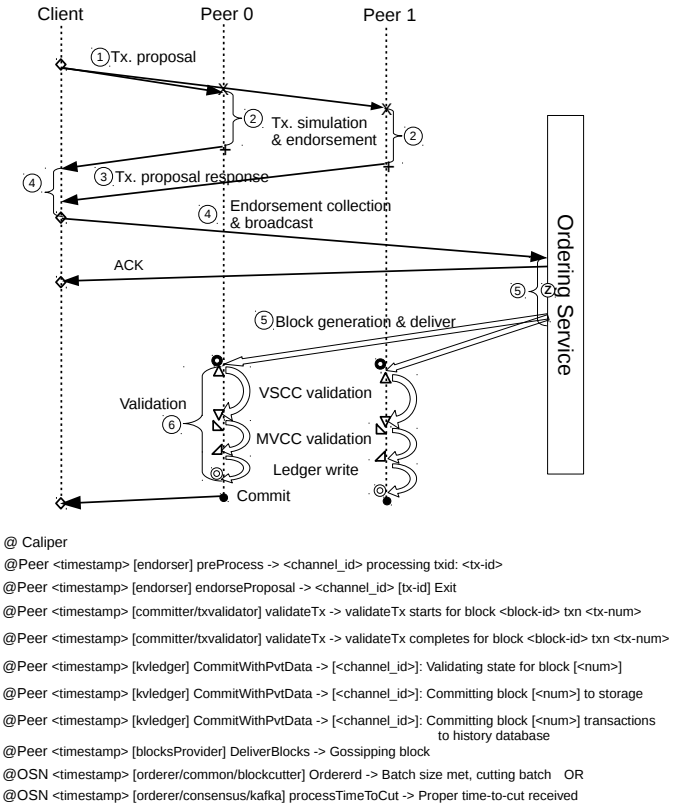

Fig. 4. Transaction life-cycle on Hyperledger Fabric V1 with relevant measurement details

arrival process. We trim the first and last 20 sec. of a test run as a ramp-up and ramp-down phase. To validate our model for different configuration settings, we vary three parameters: a) client tx. arrival rate (λ_C), b) block size, c) Transaction type ('open' and 'transfer'). Due to our hardware limitations, we could not vary the number of CPUs for validating peer. We plan to pursue it in our future work. For a given block size, we keep λ_C sufficiently high such that most blocks are created due to block size rather than timeout (ref. Section VII-B). All docker instances were restarted between each test run.

E. Model Parameters

We perform test runs for each set of configuration parameter values described above and collect the log files, from which we derive the required output metrics. Depending on λ_C, each test run consists of 7k to 30k transactions, resulting in 7k to 30k samples for transaction level parameter values and 150 to 900 samples for block-level parameter values. To ensure that the parameter values are consistent across various groups, we perform Analysis of Variance (ANOVA) F-test [17] to see if there is a statistically significant difference in means, followed by multiple comparisons procedure using Tukey's Honest Significance Test [17], [18]. Thus, we derive our parameter values from a large dataset.

In our current work, we assume the firing time for all transitions is exponentially distributed. In our future work, we plan to do a distributional analysis and choose the best-fit distribution for each transition. Since our validation results are good, we feel confident about our choice. With this choice, the underlying stochastic process is a continuous-time Markov chain (CTMC), and hence we can analyze our models using analytic-numeric solutions (Section VII).

The parameter values for 'open' transactions are summarized in Table I. Due to space constraints, we show results only for 'open' transactions in this paper. Detailed empirical analysis is shared in [19]. Transitions T_{MVCC} and T_{Ledger} are measured at a block level since the measurements are too small at a transaction level. Time to process messages at the client (T_{Pr}), and time to prepare a block (T_{OS}) also includes the transmission time to the peer. Regarding T_{OS}, although the current Kafka based ordering service does not have a notion of consensus, in future when Byzantine fault tolerance (BFT) consensus protocols are implemented, this transition will also include consensus time.

TABLE I
PARAMETER VALUES FOR SRN MODEL TRANSITIONS FOR 'OPEN' TRANSACTION

Parameter (SRN transition)	Block Size	Mean time(ms)	Rate (ms^{-1})
Client processing (T_{Pr})	–	6.47	0.155
Endorsement (T_{En0}, T_{En1})	–	3.25	0.308
Transmit to Ordering service (T_{Tx})	–	5.22	0.192
Block creation and delivery (T_{OS})	40	75.74	0.013
	80	81.60	0.012
	120	93.56	0.011
VSCC validation (T_{VSCC})	–	2.52	0.397
MVCC validation (T_{MVCC})	40	2.56	0.391
	80	5.10	0.196
	120	7.20	0.139
Ledger write (T_{Ledger})	40	207.80	0.0048
	80	208.30	0.0048
	120	188.40	0.0053

VI. OVERALL SYSTEM ANALYSIS & MODEL VALIDATION

Let us analyze the overall system model (Figure 2) for a network using `AND()` endorsement policy with $CPU_{max} = 4$ for VSCC validation and various block sizes. Unfortunately, the underlying Markov model has a huge space-space, and we could not solve the full system model using analytical-numeric solution. Hence we take the simulation approach using SPNP.

To validate our model, we compute the mean queue length at the ordering service (OS) and critical processing stage within a peer (VSCC validation, MVCC validation and Ledger Write (LWrite)) and compare it with the empirical results. We chose 'mean queue length' since it was reliable to measure and provides intuitive insights into the system performance. We validated our results for different client arrival rates and different block sizes.

From our results in Figure 5, we observe that all measurement results (marked as (m)) are comparable to the model results at different client arrival rates. The results are similar for other block sizes as well. Hence we consider our model validated. Unfortunately, we were not able to reliably measure

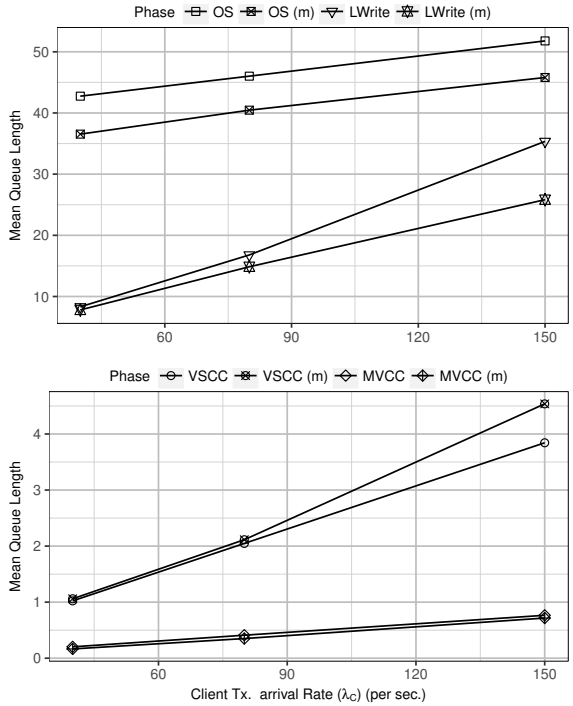

Fig. 5. Model validation comparing mean queue length at various transaction phases with empirical measurements (m) (block-size = 80)

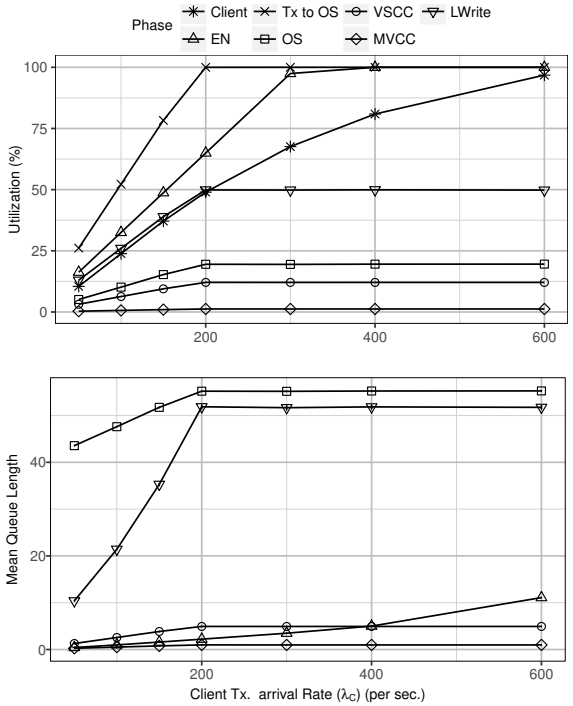

Fig. 6. Utilization, mean queue length at various transaction phases (block-size = 80)

the queue length at the endorsing peer from the peer logs. However, since the results match at four other measurement points, we leverage its results from our model.

From our model results, let us visualize the utilization and mean queue length (Figure 6) at each transaction phase with increasing λ_C. We compute the utilization for client processing and transmission to ordering service as well. We find that the transmission from client to the ordering service is a performance bottleneck with a sharply rising utilization, followed by the endorsing peer. The queue length at the ordering service/ledger write is expected to be large since it is waiting for *block size* transactions before generating/committing a block.

In our lab setup, at high client arrival rate, transactions tend to timeout (presumably due to queuing delays) and fail, although peers or ordering service is not particularly busy. It also explains that the transmission time of endorsed transactions to the ordering service is a performance bottleneck. From our measurements, it is hard to say whether the delay is due to network transmission time or due to internal queuing at the ordering service. Assuming that this delay is not a problem in another setup (replace T_{Tx} with an immediate transition), let us compute the max. throughput possible in this network. In our model, we keep increasing the client transaction arrival rate until the utilization at any transaction phase reaches 90%. We also consider a scenario where there are multiple endorsers per org. (En_{max}), using marking-dependent firing rate for transitions T_{En0}, T_{En0}. From our results in Figure 7, we see that the max. throughput increases significantly as the block size increases, albeit with an increase in mean latency

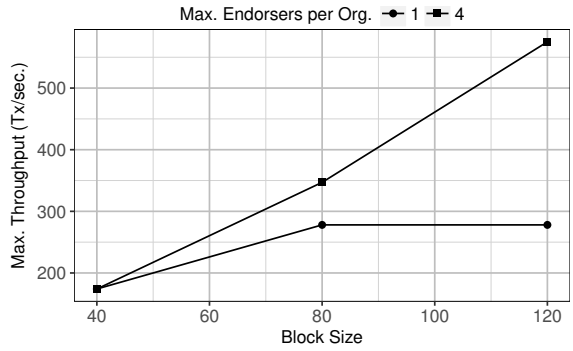

Fig. 7. Impact of block size, multiple endorsers on max. throughput

(not shown). When $En_{max} = 1$, the bottleneck is the endorsing peer. When $En_{max} = 4$, the bottleneck is the ledger write. Thus the max. system throughput can tremendously increase (especially for larger block sizes like 120) if there are multiple endorsers per org.

VII. Model Analysis

In the following three subsections, we analyze the subsystem corresponding to each transaction phase. All analysis in this Section is done using analytic-numeric solution (SPNP).

A. Endorsement Policies

The client node is responsible for seeking endorsements on the transaction it is proposing such that it satisfies the endorsement policy. The endorsement policies are monotone logical expressions that evaluate to TRUE or FALSE. E.g.,

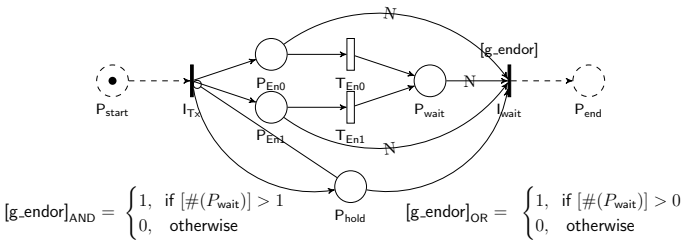

Fig. 8. Generalized SRN model to capture AND/OR endorsement policy between two peers

endorsement policy `OR(Org1, Org2)` means that endorsement from any peer from Org1 or Org2 would suffice. An endorsement policy can be expressed as an arbitrary combination of AND, OR, and k/n expressions, such as `OR(Org1, Org2) AND (2/3 of Org3, Org4, Org5)`.

The endorsement part of the model in Figure 2 represents two peers in an `AND()` policy; thus the client waits for a response from both before forwarding the transaction to ordering service. This model can be easily extended to represent `AND()` policy for more peers. To model the `OR()` and `k/n()` policies, we flush tokens from places that did not fire, using variable-cardinality arcs (called `viarc()`) in SRNs, shown by arcs with Z sign in Figure 8. Also, a guard function is used for the immediate transition I_{wait}, which is written similar to the endorsement logical expression. Thus, complex endorsement policies can be captured easily by extending the net as shown in Figure 8 (without the dotted part) and plugged in the overall system net in Figure 2.

The endorsement process adds significant latency to a transaction. First, the chaincode executes in a separate Docker container at each peer, adding a reasonable performance overhead. Second, the client needs to wait for endorsement response from multiple peers to satisfy the endorsement policy. Let us analyze the mean time to complete endorsement for different endorsement policies in Table II. We assume the time to endorsement at each peer is exponentially distributed with rate 0.308 per ms.

TABLE II
MEAN TIME TO ENDORSEMENT (MTTE) FOR DIFFERENT POLICIES

Endorsement policy	MTTE (ms)
OR (2 peers)	1.623
OR (3 peers)	1.082
AND (2 peers)	4.870
AND (3 peers)	5.952
2/3 peers	2.706
OR (2 peers) AND (2/3 of 3 peers)	3.193

In the current release of Fabric Node SDK, the client waits for replies (or timeout) from all endorsing peers before the client prepares the endorsement message for the ordering service. Thus we cannot validate our full-system model with different endorsing policies. A new feature[9] in the upcoming release will address this limitation.

[9]https://jira.hyperledger.org/browse/FAB-10672

B. Ordering Service

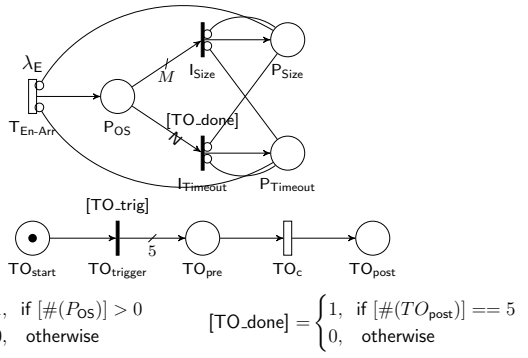

Fig. 9. SRN model of the ordering service considering block timeout and block size constraints

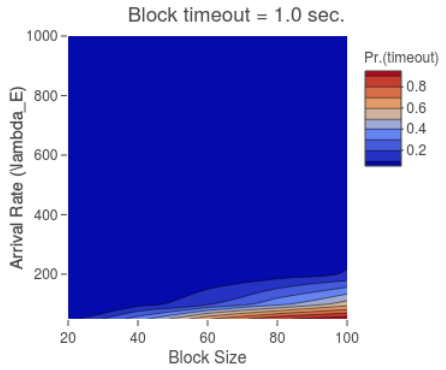

Fig. 10. Probability of block generated due to timeout as a function of endorsed transaction arrival rate and block size

The ordering service receives endorsed transactions from the client, orders them and creates a block of transactions based on block timeout or block size. We would like to assess the probability that a block was generated due to block size or block timeout, given an arrival rate of endorsed transactions (λ_E), block size, and block timeout.

Let us consider the SRN model in Figure 9. The number of tokens in place P_{OS} represents the pending transactions. A token is deposited in place P_{Size} or $P_{Timeout}$ depending on whether the block is created due to size limit or timeout. Since block timeout is deterministic, we approximate this using Erlang distribution [20], [21], which is shown as a separate net. The immediate transition $TO_{trigger}$ is enabled when there is at least one token in place P_{OS}. We consider a 5-stage Erlang distribution where each stage fires with rate 5/(block timeout).

Figure 10 presents the contour plot of the probability that a block created due to timeout condition for timeout = 1.0 sec. For smaller block size, the blocks are generated due to size limits even at low arrival rates (λ_E). As block size increases, a higher endorsed transaction arrival rate (λ_E) is required to skip the timeout condition from our model. Thus, we can skip the additional logic to account for the timeout condition, thereby significantly reducing the size and complexity of the full system model (Figure 2).

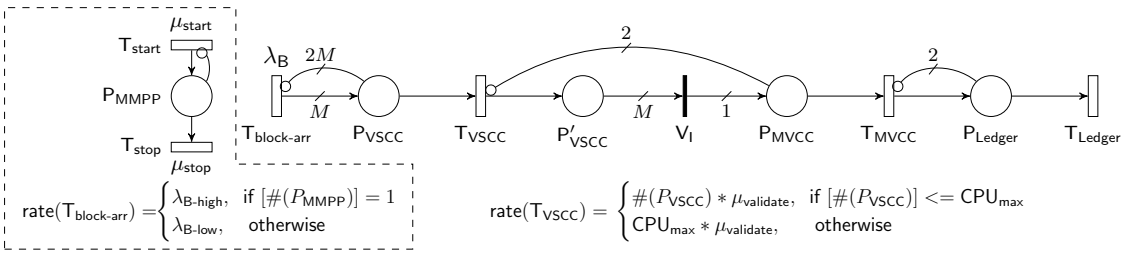

Fig. 11. SRN model of a *committer* peer, validating and committing blocks of transations

C. Block Validation & Commit

Let us analyze the performance of the committing peer using the SRN model in Figure 11. Let us assume that blocks arrivals (from the ordering service) follow a Poisson arrival process with rate λ_B, each of size M. To limit the size of the underlying Markov model, we consider a max. queue length of two blocks in each phase.

Let us consider two scenarios of interest to Fabric designers.

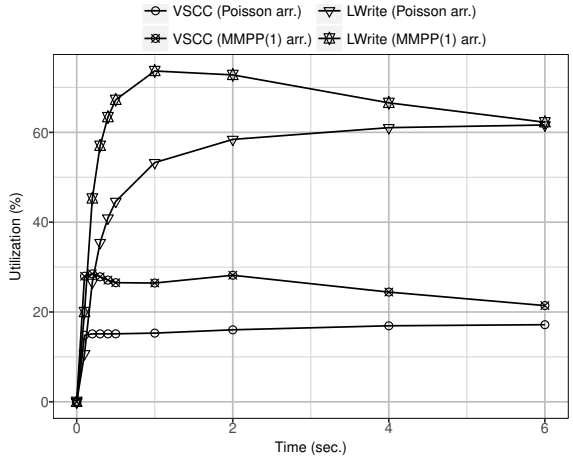

Fig. 12. Transient analysis for utilization of various stages in a committer with block-size = 80, VSCC validation $CPU_{max} = 4$

1) Bursty arrival of blocks: Let us consider a scenario in which the committing peer faces a bursty stream of blocks, consisting of periods of high and low mean block arrival rates. To capture this, we model the input arrival process using Markov modulated Poisson process (MMPP) [20], [21]. Let us consider the extended SRN model (shown in dashed box) in Figure 11 with block size M. When there is a token in place P_{MMPP}, blocks arrive at a high rate (say 6 blocks per sec.) for the average fraction of time $\frac{\mu_{start}}{\mu_{start}+\mu_{stop}}$ and low rate otherwise (say 2.25 blocks per sec.). The mean time to start of burst mode is eight sec. ($\frac{1}{\mu_{start}}$) and it lasts for a mean time of 2 sec. ($\frac{1}{\mu_{stop}}$). To compare the results from the model with non-bursty arrival rates, we consider the same average arrival rate. Thus, $\lambda_B = \lambda_{B\text{-high}} * \frac{\mu_{start}}{\mu_{start}+\mu_{stop}} + \lambda_{B\text{-low}} * \frac{\mu_{stop}}{\mu_{start}+\mu_{stop}}$.

We compare the results for the MMPP arrival starting in burst mode (MMPP(1)) with that of a model with Poisson arrival rate λ_B= 3 blocks per sec. From Figure 12, we find that the utilization of Ledger write and VSCC validation stages jumps for MMPP(1) arrival in the first few seconds. There is a significant jump in the mean queue length of both MVCC check and Ledger write stages. Overall, the VSCC validation stage seems to absorb the shock of bursty arrivals quite well. However, the utilization and mean queue length of ledger write stage is affected the most, and hence it is critical from a performance perspective. Keep in mind that the blocks are delivered from the ordering service in a sequence. Thus we need to ensure that the input queue is sufficiently large and that the committing peer can process the blocks fast enough. Rerequesting for a block again can significantly slow down the peer.

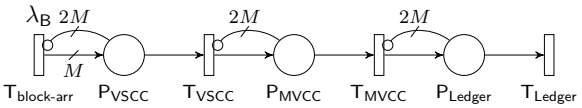

Fig. 13. SRN model of a *committer* peer in pipeline order

2) Pipeline model: To improve the performance of a committing peer, the authors in [3], [9] proposed a pipeline architecture, where each transaction passes through various stages in a pipeline, as opposed to the current architecture where transactions pass through each stage in blocks. We assume such a system would have only one logical processor for the VSCC validation. We assume the same parameter value for VSCC validation and parameter value divided by block size for MVCC, LWrite. We compute the max. throughput

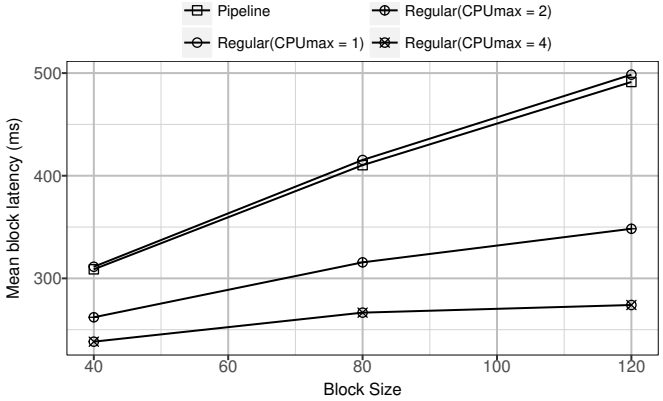

Fig. 14. Mean latency to complete block validation & commit for pipeline model vs regular model

for this pipeline model (Figure 13) for various block sizes. Surprisingly, the max. throughput is comparable to that from the regular model with $\text{CPU}_{\text{max}} = 1$. However, mean queue length at MVCC validation is slightly larger, and that at ledger write is smaller (not shown). The rest of the metrics are comparable.

Let us compute the latency for a block of transactions using a modified version of models in Figures 11 and 13, by removing the transition $\text{T}_{\text{block-arr}}$ and considering M initial tokens in place P_{VSCC} and an output place Done from transition T_{Ledger}. When M tokens are deposited in place Done, it signifies completion of a block. We compute the mean block latency for the regular model and pipeline model for different block sizes (Figure 14). As expected, the mean block latency improves slightly in the pipeline model compared to regular model with $\text{CPU}_{\text{max}} = 1$. As we add more CPUs in the regular model, the latency reduces further.

We perform sensitivity analysis for the pipeline model for mean block latency to the VSCC validation rate. We find that the max. throughput results are comparable with that of the regular model with VSCC validation $\text{CPU}_{\text{max}} = 2$, 4 and so on. Overall, pipeline architecture would slightly improve the block latency, but would not improve the max. throughput and other performance metrics compared to the conventional architecture.

VIII. Discussion

A. Largeness of stochastic model

We used a high-level formalism, i.e., SRN to model the complex interactions of a Fabric network. A drawback of this approach is that the underlying stochastic model generated is huge. We are unable to generate the underlying state-space for the full-system model since it is infinite. Hence we took the simulation approach (ref. Section VI). For the committing peer model (ref. Section VII-C), we considered a max. queue length of two blocks at each stage. This truncates the state-space of the underlying model, which we solved using analytic-numeric solution. Interestingly, we find that the model state-space size increases linearly with the block size.

The largeness problem limits our ability to answer questions regarding the scalability of Fabric networks. In our future work, we would consider a hierarchical approach and fixed-point iterative solution techniques to mitigate this issue [12].

B. Limitations of our model

A limitation of our model is that we cannot compute the latency at a specific transaction throughput. From our model, we can estimate the mean latency of a block of transactions (ref. Section VII-C2). However, this does not account for any queuing delays at various nodes when the system is "loaded." Along those lines, this model also cannot capture transaction failures due to timeout (mainly due to queuing delays). In our future work, we plan to compute the response time distribution of a transaction using a state-space modeling approach, in lines of work presented in [22].

C. Threats to validity

One threat to the validity of our results is that we did not use custom VSCC. Custom VSCC lets the user define additional (business) logic to validate the transactions. Since this is in addition to the endorsement policy signature validation done during VSCC validation, we feel a higher mean parameter would capture this easily. Note that authors in [3] used custom VSCC but not the authors in [9]. Another threat is that our nodes were connected in a LAN setting, as opposed to a wide-area network (WAN) that would be expected in the real world. In our future work, we plan to replicate our results by running the Fabric on a cloud service like Amazon Web Services (AWS).

IX. Related Work

A. Performance modeling of Blockchain Networks

Decker and Wattenhofer [23] presented a simple model to compute the stale block generation rate in the Bitcoin network, which takes into account the block generation and the block propagation process of bitcoin. Gervais et al. [24] model and analyze the security and performance aspects of Proof-of-Work (PoW) based blockchain networks. For performance modeling, they develop a simulator that mimics the block mining and propagation process as a function of block interval, block size and block propagation mechanism. Its output metric is stale block rate. Papadis et al. [25] developed stochastic models for PoW based blockchain networks to compute the block growth rate and stale block rate as a function of the propagation delay and the hashing power of the compute nodes. In our previous work, we modeled and analyzed the practical Byzantine fault tolerance (PBFT) consensus process of Fabric v0.6 [26]. Since the architecture of Fabric V1 has significantly evolved compared to release v0.6 [5], the models discussed in this paper are not applicable to release v0.6.

B. Performance evaluation of Hyperledger Fabric

Performance of Hyperledger Fabric v1.0 was studied extensively by Thakkar et al. in [9], where they vary five tunable parameters: block size, endorsement policy, number of channels, number of vCPUs for peers, and key-value database (GoLevelDB vs. CouchDB). Their research resulted in three optimizations for Fabric: parallelization of VSCC validation, cache for Membership Service Provider (MSP), and bulk read/write for CouchDB, all of which were incorporated in release v1.1, that was studied in [3] and our work. [3] presents extensive details of Fabric V1. They tested Fabric v1.1 using an application called *Fabcoin* using the data model is similar to the Bitcoin-style UTXO[10]. In contrast to [3], [9] where they run nodes on virtual machines in a cloud datacenter, we run our nodes on physical machines. Sousa et al. [27] integrated BFT-SMaRt [28] as an ordering service for Fabric v1.0, and present performance analysis only for the ordering service from their deployment in a geo-distributed setting over AWS data centers.

[10]https://bitcoin.org/en/glossary/unspent-transaction-output

Auh Dinh et al. [29] conducted performance testing of Fabric v0.6 (along with Ethereum and Parity) on a 48-node cluster, using macro benchmarks based on the YCSB and Smallbank benchmarks. They also developed separate micro-benchmarks to gain insights into performance characteristics of various layers such as consensus, data model, and execution engine.

X. Conclusion & Future Work

In this paper, we developed stochastic models for a popular distributed ledger platform called Hyperledger Fabric. Each transaction in a Fabric network undergoes three phases: endorsement, ordering, and validation. We analyze the full-system as well as the subsystems corresponding to each transaction phase in details. We collect data from a Fabric setup running realistic workload to parameterize and validate our models. Our models provide a quantitative framework that helps a system architect estimate performance as a function of different system configurations and makes design trade-offs decisions.

For an incoming block, since the committing peer validates transactions (VSCC) in parallel, there is a significant performance improvement if the committing peer is deployed on a system with a large number of CPUs. It can significantly reduce the queue length at the VSCC validation as well as let system absorb the shock of a burst arrival of blocks. However, it is not the performance bottleneck for the committing peer, since ledger write has the highest utilization. We also analyzed two scenarios: multiple endorsing peers per org. and pipeline architecture for the validator. Transaction endorsement parallelization can significantly reduce the endorser queue length and increase the max. throughput of the system if the endorsement process is the performance bottleneck. The pipeline architecture provides around 1% improvement in mean block validation latency but offers no other performance benefit. In the future work, we would consider a hierarchical modeling approach to overcome the largeness in model generation and solution.

Acknowledgments

This research was supported by IBM under a faculty award and an IBM Ph.D. fellowship. This research was also supported in part by US NSF Grant number CNS-1523994. The authors would like to thank Thomas Napoles and Pratt IT dept. for lab infrastructure support, Praveen Jayachandran and Senthil Nathan N at IBM Research - India for discussions in the early phase of this research.

References

[1] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "Blockchain Technology Overview (Draft NISTIR 82022)," https://csrc.nist.gov/CSRC/media/Publications/nistir/8202/draft/documents/nistir8202-draft.pdf.

[2] D. Tapscott and A. Tapscott, *Blockchain Revolution : How the Technology behind Bitcoin is Changing Money, Business, and the World*. New York: Portfolio Penguin, 2016.

[3] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *EuroSys*, 2018, pp. 30:1–30:15.

[4] IBM, "IBM Blockchain Platform," https://www.ibm.com/blogs/blockchain/2017/08/your-guide-to-the-ibm-blockchain-platform-announcement/.

[5] M. Vukolić, "Rethinking Permissioned Blockchains," in *ACM Workshop on Blockchain, Cryptocurrencies and Contracts (BCC)*, 2017, pp. 3–7.

[6] C. Cachin, "Architecture of the Hyperledger Blockchain Fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL)*, 2016.

[7] F. B. Schneider, "Implementing Fault-tolerant Services using the State Machine Approach: A Tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, Dec. 1990.

[8] C. H. Papadimitriou and P. C. Kanellakis, "On Concurrency Control by Multiple Versions," *ACM Trans. Database Syst.*, vol. 9, no. 1, pp. 89–99, Mar. 1984.

[9] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," in *IEEE MASCOTS*, 2018.

[10] *Hyperledger Blockchain Performance Metrics White Paper*, v1.0 ed. The Linux Foundation, Oct. 2018. [Online]. Available: https://www.hyperledger.org/resources/publications/blockchain-performance-metrics

[11] J. K. Muppala, G. Ciardo, and K. S. Trivedi, "Stochastic Reward Nets for Reliability Prediction," in *Communications in Reliability, Maintainability and Serviceability*, 1994, pp. 9–20.

[12] K. Trivedi and A. Bobbio, *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press, 2017.

[13] G. Ciardo, J. K. Muppala, and K. S. Trivedi, "SPNP: Stochastic Petri Net Package," in *Petri Nets and Performance Models*, 1989, pp. 142–151.

[14] K. S. Trivedi, "SPNP User's Manual – Version 6.0," 1999.

[15] "Does Hyperledger Fabric need Docker?" https://stackoverflow.com/questions/48070380/does-hyperledger-fabric-need-docker.

[16] J. P. Buzen and P. J. Denning, "Measuring and Calculating Queue Length Distributions," *IEEE Computer*, vol. 13, no. 4, pp. 33–44, April 1980.

[17] T. Hothorn and B. S. Everitt, *A Handbook of Statistical Analyses using R*. Boca Raton, FL: CRC Press/Taylor & Francis Group, 2014.

[18] Y. Hochberg and A. C. Tamhane, *Multiple Comparison Procedures*. John Wiley & Sons, Inc., 1987.

[19] H. Sukhwani, "Performance Modeling & Analysis of Hyperledger Fabric (Permissioned Blockchain Network)," Ph.D. dissertation, Duke University, Dec. 2018, (in preparation).

[20] K. Trivedi, *Probability & Statistics with Reliability, Queueing & Computer Science applications*, 2nd ed. Wiley, 2001.

[21] S. Ramani, K. S. Trivedi, and B. Dasarathy, "Performance Analysis of the CORBA Event Service using Stochastic Reward Nets," in *IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2000, pp. 238–247.

[22] ——, "Performance Analysis of the CORBA Notification Service," in *IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2001, pp. 227–236.

[23] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013*, 2013, pp. 1–10.

[24] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the Security and Performance of Proof of Work Blockchains," in *ACM SIGSAC CCS*, 2016, pp. 3–16.

[25] N. Papadis, S. Borst, A. Walid, M. Grissa, and L. Tassiulas, "Stochastic Models and Wide-Area Network Measurements for Blockchain Design and Analysis," in *IEEE INFOCOMM*, 2018.

[26] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)," in *IEEE SRDS*, Sept. 2017, pp. 253–255.

[27] J. Sousa, A. Bessani, and M. Vukolic, "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform," in *IEEE/IFIP DSN*, 2018, pp. 51–58.

[28] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State Machine Replication for the Masses with BFT-SMaRt," in *IEEE/IFIP DSN*, 2014, pp. 355–362.

[29] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "BLOCKBENCH: A Framework for Analyzing Private Blockchains," in *ACM International Conference on Management of Data*, ser. SIGMOD, 2017, pp. 1085–1100.