A Scalable Quantum Key Distribution Network Testbed Using Parallel Discrete-Event Simulation

XIAOLIANG WU, BO ZHANG, GONG CHEN, and DONG JIN, Illinois Institute of Technology

Quantum key distribution (QKD) has been promoted as a means for secure communications. Although QKD has been widely implemented in many urban fiber networks, the large-scale deployment of QKD remains challenging. Today, researchers extensively conduct simulation-based evaluations for their designs and applications of large-scale QKD networks for cost efficiency. However, the existing discrete-event simulators offer models for QKD hardware and protocols based on sequential event execution, which limits the scale of the experiments. In this work, we explore parallel simulation of QKD networks to address this issue. Our contributions lay in the exploration of QKD network characteristics to be leveraged for parallel simulation as well as the development of a parallel simulation framework for QKD networks. We also investigate three techniques to improve the simulation performance including (1) a ladder queue based event list, (2) memoization for computationally intensive quantum state transformation information, and (3) optimization of the network partition scheme for workload balance. The experimental results show that our parallel simulator is 10 times faster than a sequential simulator when simulating a 128-node QKD network. Our linear-regression-based network partition scheme can further accelerate the simulation experiments up to two times over using a randomized network partition scheme.

CCS Concepts: • **Networks** → **Network simulations**; • **Computing methodologies** → **Discrete-event simulation**;

Additional Key Words and Phrases: Parallel discrete-event simulation, quantum key distribution, BB84

ACM Reference format:

Xiaoliang Wu, Bo Zhang, Gong Chen, and Dong Jin. 2022. A Scalable Quantum Key Distribution Network Testbed Using Parallel Discrete-Event Simulation. *ACM Trans. Model. Comput. Simul.* 32, 2, Article 11 (February 2022), 22 pages.

<https://doi.org/10.1145/3490029>

1 INTRODUCTION

Quantum key distribution (QKD) provides a promising solution for secure communications by establishing secure shared cryptographic keys between various parties through untrusted

D. Jin is also with the University of Arkansas.

This work was partly sponsored by the Air Force Office of Scientific Research (AFOSR) under grant YIP FA9550-17-1-0240 and the National Science Foundation (NSF) under grants CNS-1730488 and DMR-1747426. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR and NSF.

Authors' address: X. Wu, B. Zhang, G. Chen, and D. Jin, Illinois Institute of Technology, Chicago, United States; emails: xwu64@hawk.iit.edu, bzhang43@hawk.iit.edu, gchen31@hawk.iit.edu, dongjin@uark.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

1049-3301/2022/02-ART11 \$15.00

<https://doi.org/10.1145/3490029>

networks. Apart from the classical information represented by bits, QKD also utilizes quantum information encoded on photons to share secrets. The law of quantum physics protects QKD networks against eavesdropping. The non-cloning theorem forbids an eavesdropper from extracting information without disturbing the associated quantum state. Since the disturbance is noticeable for applications, the eavesdropper could not secretly retrieve the shared keys even with full control of all communication channels. This unique property motivates people to build quantum networks for secure communication. Several photonic quantum networks have been established. The first QKD network was established by DARPA [18], operating 10 nodes, across Boston and Cambridge, Massachusetts. Further, the long-distance QKD network, using the 2,000-km fiber-optic link, was established between Beijing and Shanghai [15]. The Netherlands [12], United Kingdom [17], and South Korea [38] also have their projects of establishing QKD networks.

The high cost of physical testbed limits quantum network research, and therefore, for cost efficiency, people develop simulators of quantum networks including simulating interactions with various protocols and tracing the quantum state of qubits. Researchers built modules on NS-3 to simulate QKD protocols [27], in which they do not simulate the transformation in the quantum channels, like transmission and transformation of photons. As an agent-based simulator, SQUANCH [4] can be parallelized by running every agent on its own process. The speed of the simulator can be largely affected as workloads are often unevenly distributed among agents. In addition, SQUANCH does not trace the simulation time. SimulaQron [16] and qkdSim [13] are two sequential discrete-event simulators for quantum internet and QKD systems. However, the sequential simulation approach significantly limits the scale of simulated quantum networks. For example, using sequential simulation [39, 40] takes around 184 seconds to simulate 20 ms of a 128-node QKD network (see Figure 10(a) in Section 7 for detailed experimental settings).

In this work, we explore the feasibility of using parallel simulation techniques for large-scale QKD network simulation as well as the means of making the parallel simulation efficient. Based on the careful analysis of a QKD network including events and channel properties, we develop a parallel discrete-event simulator for QKD networks. The simulator has two layers. The upper layer consists of models of components in the QKD network and represents the state of a QKD network. The lower layer is a parallel simulation kernel based on conservative synchronization. Additionally, we apply three techniques to speed up the parallel simulation: (1) using the ladder queue [35] instead of min-heap to enqueue and dequeue events, (2) caching results of the massive matrix multiplications caused by the transformation of the quantum state, and (3) optimizing the partition scheme. The ladder queue [35] improves the time complexity of enqueue and dequeue operations to $O(1)$, whereas the same operation costs $O(\log N)$ in the min-heap. The memoization technique significantly reduces the time on tracing a quantum state, which consumes 25.9% of the simulation time in our experiment. The generation of the optimal partition scheme consists of two parts: time prediction for a given partition scheme and searching the optimal partition scheme. To predict the total execution time, we develop linear regression models to predict three components of a QKD network simulation, such as computing, communication, and synchronization. As searching for the optimal solution is NP-hard, we use simulated annealing to find an approximate solution. Finally, we conduct extensive experiments to evaluate the simulation scalability and efficiency with various QKD network settings and perform an error analysis of the execution time prediction model. Results show that to simulate a 128-node QKD network using the same physical machine, our parallel simulator significantly reduces the execution time from 184 seconds in the sequential mode to 18 seconds using 32 threads. In addition, the optimized network partition scheme can further accelerate the simulation time up to 40% more than the random network partition scheme. The contributions of this work are listed as follows.

Fig. 1. Two-terminal QKD network.

- We study the characteristics of QKD networks and summarize five observations that are useful for designing parallel simulation of QKD networks.
- We develop a QKD network simulator using conservative parallel discrete event simulation.
- We explore three techniques (i.e., ladder queue, memoization, network partition) to speed up the parallel simulation of QKD networks.
- We conduct an extensive evaluation of the simulation framework in terms of speed, scalability, and error analysis.

The rest of the article is organized as follows. Section 2 overviews the background of QKD protocols and models of optics and the related work on QKD simulation. Section 3 explores the characteristics of QKD networks from existing QKD network simulators and physical QKD networks. Section 4 presents the architecture design of our QKD network parallel simulator. Section 5 introduces the ladder queue and memoization techniques to speed up the simulation with evaluation. Section 6 first describes models to predict computing, communication, and synchronization time using linear regression, then describes the algorithm of searching the optimal network partition scheme. Section 7 evaluates the performance of our simulator in terms of speedup, scalability, and error analysis. Section 8 concludes the article with future works.

2 BACKGROUND AND RELATED WORK

The advancement of quantum technologies redefines many well-founded fields with new applications, such as encryption crack algorithms, modeling complex molecular structures, and realization of strong artificial intelligence. New applications in these fields take quantum states as the foundation. Among quantum states, superposition can be realized in the experiment by encoding different physical properties of a quantum object. Among many quantum objects, photon stands out for carrying quantum information. The photonic state has been proved and used widely including the validity of quantum mechanics, the superdense encoding of classical information [1, 2], and security limits of QKD [2, 22, 25]. QKD, as one of the best-known applications, was used in creating a secret shared key for secure communication [5].

2.1 QKD Protocols

The key for cryptography algorithms, like Advanced Encryption Standard (AES), is distributed by QKD protocols, which allows two parties to detect eavesdroppers during the key distribution. This property, benefiting from a fundamental aspect of quantum mechanics, is important and unique. When a third party tries to eavesdrop on the key, the interception in classical channels do not expose any information about keys and the interception in quantum channels unavoidably introduces detectable noise. Therefore, the laws of quantum mechanics guarantee the security of QKD.

The most well-known QKD technique is BB84 proposed by Charles H. Bennett and Gilles Brassard in 1984 [5]. Figure 1 shows the hardware using the BB84 protocol. The sender (Alice) and

the receiver (Bob) are connected by a quantum communication channel and a classical channel. Quantum states are transmitted in a quantum channel. Classical bits are allowed to be transmitted in an insecure classical channel. The protocol is secure because it encodes information in non-orthogonal states or conjugate states. Any two pairs of conjugate states and two states within a pair that are orthogonal to each other can be used for the BB84 protocol. Quantum mechanics ensure that these states cannot, in general, be measured without disturbing the measurement result on the other conjugate states.

The BB84 protocol is modeled by performing the following steps:

- (1) Alice prepares a stream of random bits; these bits are encoded onto the polarization of photons under the conjugate basis that was randomly selected from two conjugate bases; the encoded photons are emitted from the light source and transmitted to Bob in the quantum channel.
- (2) Bob randomly chooses a conjugate basis of two for each received photon by adjusting the **polarized beam splitter (PBS)**; the clicked **single-photon detector (SPD)** determines the encoded bit to be 0 or 1.
- (3) Bob periodically sends his basis list of measurement to Alice.
- (4) Alice determines which basis matches and sends the indices of bits with matching encoding and measurement basis to Bob in the insecure classical channel.
- (5) Bob receives the list of indices of bits and uses these bits as the secret key shared with Alice.

QKD protocols also include error correction [8], privacy amplification [7], and authentication [18]. These protocols only rely on the classical channel to achieve their functions, so they are not studied in this article.

2.2 Models of Optics

The utilized optics are shown in Figure 1. The light source is used to generate photons with the desired quantum state. The optical fiber is used for quantum communication. A detector is capable of accurately recording photon arrival times. A splitter is used for separating the photons based on polarization under either conjugate basis.

The *light source*, as a pulsed laser, generates attenuated pulses with frequency f . Photons with the arbitrary quantum state can be generated in a pulse, and the number of emitted photons in a pulse follows a Poisson distribution with mean μ .

The *quantum channel*, as an optical fiber, is used for transmitting quantum information. The propagation time of the photons is modeled as $\frac{L}{c^*}$, where L denotes the length of the fiber and c^* denotes the speed of light in the fiber. The loss rate of a quantum channel is $10^{-\frac{L \cdot \alpha_o}{10}}$, where α_o is the attenuation measured in decibels per kilometer.

The *SPD* is able to detect a single photon and specify an arrival time. The arrival time is recorded by detectors. And the list of arrival times is reported to upper-layer protocols periodically. The index of a photon is calculated from its timestamp.

The *PBS* separates the photons based on polarization. The PBS can be adjusted to measure the quantum state of a photon on a different basis. In BB84, if a quantum state is encoded and measured on the same basis, the measured result presents the correct information from the encoded photon. Otherwise, the receiver only has a 50% probability to get the correct information.

2.3 Related Work

Quantum network simulators provide a flexible and cost-efficient platform for researchers to study quantum networks with various configurations. Several simulation tools have been recently introduced to the community. QKDNetSim [27] is an NS-3 simulation module of the QKD network.

This work simplifies models of QKD links and focuses on the network performance impact by the operations on the classical network. For example, the authors study how routing protocols affect the performance of QKD networks. Researchers could customize several parameters in the simulator including the number of nodes, packet size, type, and rate, as well as the type of encryption and authentication. `qkdSim` [13] is a simulation toolkit for QKD. Compared with `QKDNetSim`, `qkdSim` provides not only QKD protocols but also precise models of certain physical components (e.g., light source and detectors) to improve the simulation fidelity. Models implemented in `qkdSim` include the beam splitter, single-mode fiber, SPD, and free-space channel. The experimental imperfections of physical components are modeled to analyze any generic QKD protocol. Researchers use `qkdSim` to analyze the performance of the B92 protocol [10, 11].

Apart from QKD simulators, people also develop simulators for quantum internet [23]. `SeQUeNCe` [41], `Netsquid` [14], and `QuISP` [26] are three sequential discrete-event simulators for quantum networks that aim to transmit quantum information. `SQUANCH` [4] is a simulator for quantum internet that supports parallel simulation. By using the agent-based model, every node is modeled as an agent and executed by an independent process. Agents use inter-process communication to send and receive quantum or classical information. Although the parallel simulation may speed up experiments, the network partition scheme does not necessarily guarantee scalability. For example, an unbalanced workload or large volume and bursty traffic could make the overhead of parallelization overwhelm its benefit.

In this article, we consider the evolution of quantum systems and simulation scalability. Therefore, we develop models of physical components to trace changes on every photon and then parallelize the simulation with a partition scheme. To the best of our knowledge, this is the first work to investigate the discrete-event parallel simulation of QKD networks. We present how to use unique features of QKD networks to efficiently parallelize QKD simulation models. Our parallel simulation using the optimized network partition scheme is 40% faster than the simulation with a random partition scheme.

3 ANALYSIS OF QKD NETWORKS FOR PARALLEL SIMULATION

This work aims to study efficient methods to parallelize QKD network simulation to support large-scale QKD network research and development. The objective is to understand how a QKD network works and what features may bring opportunities and challenges for parallel simulation of a QKD network. We first present a motivation example using sequential simulation to demonstrate the characteristics of QKD networks in Section 3.1. We then analyze the simulated QKD networks and existing physical QKD networks in Section 3.2. We present five observations that one can leverage for designing parallel simulation of QKD networks.

3.1 Analysis of Sequential Simulation of QKD Networks

We first use the sequential QKD simulator developed in our prior work [39, 40] to study the behavior of QKD networks. The metrics include the type of simulation events, the execution speed of different event types, and the distribution of each type of event. We conduct extensive experiments and choose the following scenario as an illustrative example. We summarize three key observations that are useful for parallel simulation of QKD networks and also discuss the challenges.

We set up a QKD network consisting of two QKD terminals, Alice and Bob, as shown in Figure 1. Both terminals run the BB84 (i.e., the first QKD protocol) [5] and have access to a classical communication channel and a quantum communication channel. The hardware controlled by Alice consists of an 80-MHz light source producing a single photon with a specific quantum state. The photons are transmitted through the 10-km quantum channel to Bob. The hardware of Bob

Fig. 2. Comparison of three simulation event types: E_q , events on a quantum channel; E_c , events on a classical channel; and E_s , events inside one QKD terminal.

consists of a PBS and two SPDs. These devices measure the quantum state of received photons. The measurements are used to generate a 512-bit symmetric key shared between Alice and Bob.

Simulation events in the QKD network are executed either within one terminal or across two terminals. The cross-terminal events are generated and executed at different terminals, and both classical and quantum channels may process such events. Therefore, we categorize all simulation events into three types: (i) events on a quantum channel (E_q) (e.g., sending and receiving photons), (ii) events on a classical channel (E_c) (e.g., sending and receiving classical messages), and (iii) events inside one terminal (E_s) (e.g., detectors report click time to protocols periodically).

It took 10.3 seconds to simulate 100 ms of the QKD network on a server with a 2.6-GHz Dual Intel Xeon CPU and 64 GB of memory. The network successfully distributed 455 keys between Alice and Bob. A total of 1,259,964 events were scheduled, and 1,259,681 events were executed. Note that 283 events were scheduled beyond the simulation end time and thus were not executed. In the following, we present three observations from the simulation results that are critical for designing the parallel version of QKD network simulation.

OBSERVATION 1. *Quantum channel based events, E_q , are the dominant simulation events in a QKD network in terms of amount and execution time. We collected the number of events and their execution time for all three types of events (i.e., E_q , E_c , and E_s) and plotted the results in Figure 2. As shown in Figure 2(a), 99.4% events are E_q . This is because the successful transmission of quantum information is highly dependent on the high transmission frequency to compensate for the low transmission success rate. Figure 2(b) shows that E_q takes 63.6% of the total execution time and E_s takes 29.9%. E_q again consumes most of the execution time in total, and E_s is the heaviest event type. By analyzing individual events in E_s , we find that the heavy E_s events are generated from the QKD protocol for periodically calculating the index of photons by their arrival time. A linear relation exists between such events and the number of photons. The execution time of E_s is actually highly dependent on E_q . Therefore, the number of E_q events in a simulated QKD network is a dominant factor contributing to the simulation workload. We take advantage of this observation to design the network partition algorithm in Section 6 to efficiently balance the simulation workload.*

OBSERVATION 2. *The execution time of E_q events has little variance. The same type of events may have very different execution time in classic networks because of the dynamic network state. To understand the behavior in quantum networks, we measure the execution time of every individual event, and plot the distribution of event execution time for all three types in Figure 3. The average execution time of E_q (4.78 μ sec) is around 30 times smaller than the one of E_c (140 μ sec) and around 6 times smaller than the one of E_s (947 μ sec). E_q has the lowest variance among all three types—that is,*

Fig. 3. Distribution of individual simulation event execution time: E_q , events on a quantum channel; E_c , events on a classical channel; and E_s , events inside one QKD terminal.

the standard deviation of E_c is more than 26 times of E_q . Ninety-five percent of E_q events are executed within $9.6 \mu\text{sec}$. Therefore, it is reasonably accurate to estimate the execution time of E_q events by counting the number of such event. We take advantage of this observation to precisely predict the execution time and design our network partition algorithm in Section 6.2 based on the prediction.

OBSERVATION 3. E_q events between two QKD terminals are evenly distributed over simulated time. The sending rate of quantum information depends on the attributes of a light source. Although photons containing quantum information are not always generated by the light source, the light source emits photons with a specific frequency. Given a fixed light source frequency, the sending events are evenly distributed over time. A sending event may trigger a receiving event on the PBS. Although attributes like temperature may affect the delay between the QKD terminals, the variance of delay is very little because a fluctuating delay can destroy the time-based identification of a photon. Therefore, the receiving events on the PBS are also evenly distributed over time. We take advantage of this observation to further enhance the prediction of the execution time based on the amount of E_q events to enable an efficient network partition algorithm in Section 6.2.

3.2 Analysis of Existing QKD Networks

Existing QKD networks, like DARPA [18] and SECOQC [29], utilize optical fiber or free-space as quantum channels. Photons, a good medium to carry quantum information, are used to transmit quantum states. Despite the long transmission distance of a free-space-based QKD network, the high cost of satellite limits the scale of free-space QKD networks. Meanwhile, QKD over optical fibers could use dark fibers from existing infrastructures, which makes the optical fiber an economic solution. Therefore, this work only analyzes the optical-fiber-based QKD networks.

OBSERVATION 4. The delay of a quantum channel is dominated by the propagation delay and is significantly lower than the delay of a classical channel of the same distance. Unlike information carried over classical channels, quantum information cannot be cloned and extracted. Although such a feature enables perfect security [5], it significantly restricts the transmission distance. The loss rate in the optical fiber increases exponentially with the length of fiber due to the attenuation. One can use amplifiers to prevent the loss of classical information from the attenuation. However, such amplifiers are useless for quantum channels because the amplification process may alter the carried quantum information. To reduce the loss, the length of fiber is often restricted. For example, the loss rate of 100-km fiber can be as large as around 99%. Furthermore, the QKD network does not route or process quantum information. A quantum channel is composed of directly connected optical fibers. Once a photon arrives at the quantum terminal, it passes a series of optics and is measured by detectors. The time to pass the local optics can be as low as 1 ps, which is much lower than the packet processing time in a classical channel. Therefore, we only consider propagation delay in a quantum channel.

We will utilize the easy-to-obtained channel delay as the minimum lookahead value to design our conservative synchronization-based parallel simulation for QKD networks. On one hand, the large amount of E_q events occurred within a channel delay makes the approach promising; on the other hand, further performance improvement requires us to exact better lookahead values from other QKD network characteristics.

OBSERVATION 5. *Different QKD sessions have low interdependency. The DARPA QKD network consists of 10 QKD terminals [18]. The protocols assume that every terminal is a trusted terminal, and a trusted relay is used to establish a secure channel between terminals. The co-operation among terminals on the path relies on the information transmitted over the classical channels. This design allows every QKD session to distribute key independently. The low interdependency among different QKD sessions is a promising character for parallel simulation, and motivates us to investigate efficient network partition algorithms to balance simulation workload with low synchronization overhead.*

An alternative solution is called quantum teleportation [6] that uses a pair of entangled qubits to teleport quantum state. Because the target quantum state is not transmitted through the optical fiber, the attenuation of fiber does not cause the loss of target quantum state. However, how to efficiently distribute the entangled pairs between two terminals is a big challenge. Most entanglement distribution schemes still rely on photons and optical fibers to establish entanglement [3, 32, 43]. As a result, Observation 5 is still valid in the case of quantum teleportation. In this work, we focus on the BB84 protocol [5]. The discussion of quantum teleportation and entanglement is shown in Section 8 as future works.

4 SIMULATOR ARCHITECTURE

The sequential simulation in Section 3.1 took around 10.3 seconds to simulate 100 ms of a QKD network. We investigate parallel simulation with an objective to improve execution speed and preserve the accuracy of simulation experiments. An appropriate synchronization method for QKD network scenarios is the key to efficient parallel simulation. The propagation delay described by Observation 4 provides a reasonable minimum lookahead for conservative parallel simulation [28]. In this work, we take the barrier-based synchronization approach to develop a parallel simulator for QKD networks [24].

Figure 4 depicts the two-layer simulator architecture. The upper layer consists of models of hardware and protocols of QKD networks described in Section 2, including the BB84 protocol, light source, SPD, PBS, quantum channel, and classical channel. The lower layer is the kernel supporting parallel discrete event simulation.

The simulation enables interactions among a number of QKD objects, such as QKD terminals. Each object is assigned to a timeline, which hosts an event queue and is in charge of advancing all objects assigned to it. Interactions between objectives within one timeline do not need synchronization other than executing events from the event queue. The event queue is implemented as a min-heap sorted by the event timestamp. Each timeline keeps executing the top event in the heap and advances its simulation time to the timestamp of the event. Multiple timelines may run concurrently to exploit parallelism, but they have to be carefully synchronized to ensure global causality. The synchronization method explicitly expresses quantum channel delays across QKD terminals residing on different timelines. The synchronization algorithm creates synchronization windows that are guarded by two barriers. In one synchronization window, all timelines are safe to advance without being affected by other timelines.

New events targeting the same timeline are pushed into the event queue of this timeline. Cross-timeline events are stored in a temporal buffer of the timeline generated by the event. As shown in Figure 5, the barrier synchronization approach consists of two barriers. Barrier 1 ensures that

Fig. 4. Architecture of the simulator.

Fig. 5. Barrier synchronization. “Pending” means that a timeline is blocked by a barrier and waiting for other timelines. “Running” means that a timeline advances its simulation time by executing events in its own queue.

no cross-timeline event exchange occurs before the end of the current synchronization window. Timelines finished executing early wait at the barrier for other timelines. Barrier 2 sets another synchronization point for exchanging cross-timeline events in the temporal buffers and setting the next synchronization window for event parallel processing. Note that the simulation time does not advance between the two barriers. The procedure keeps repeating until every event queue is empty or the simulation end condition is met.

To reduce system overhead from the frequent mutex operations, we design the following merge event mechanism. The cross-timeline events are not immediately pushed into the event queue of the target timeline; instead, these events are temporarily stored into a set of event queues. During the event exchange period (i.e., between barrier 1 and barrier 2), every timeline retrieves events from the temporary event queues of other timelines. Each timeline then merges the retrieved event queues into their main event queue. Since the event queue is implemented as a min-heap, the runtime complexity of merging two min-heap is $O(N)$, where N is the total size of events in the two min-heaps. To merge M min-heaps with total N elements, every two min-heaps are merged and thus produce the new set of min-heaps with the size of $M/2$. The time complexity of this step

is $O(N)$. This step repeats $\lceil \log_2(M) \rceil$ times before we get a min-heap that includes all N elements with a runtime complexity $O(N \log(M))$.

5 SIMULATION PERFORMANCE ENHANCEMENT

We investigate three methods to improve the performance of QKD network simulation. First, we change the data structure of the event queue from a min-heap list to a ladder queue [35], and thus greatly reduce the time on the enqueue and dequeue operations. Second, we memorize the states of all matrix multiplications and thus eliminate the massive repeated computational time to simulate a QKD network. Finally, we optimally partition the QKD network and thus minimize the parallelization overhead. Details about the network partition are introduced in Section 6.

We explore the efficient data structure of event queue for parallel discrete event simulation of QKD networks. The high failure rate in a QKD network is caused by factors like attenuation, noise, and unmatched measurement basis. A QKD network must transmit photons in a high frequency to distribute keys, and thus it generates massive enqueue and dequeue operations. The megahertz lasers emit photons encoded with specific quantum states to the network, which implies millions of emissions occur in 1 second. To trace every photon in the network, the simulator should also generate millions of events to mimic the procedures from generating photons to receiving photons. According to Observation 1, the quantum channel based event has much less execution time and much higher quantity than other types of events. The CPU profile of the simulation execution (see Section 3.1) shows that around 4.97% execution time is used for scheduling events. Researchers have explored various types of queue for parallel discrete-event simulation, such as ladder queue [35], splay-tree [33], and calendar queue [9]. Although the tree-based priority queue is a widely used data structure for scheduling events by timestamps, the operations on a tree-based priority queue is bounded by $O(\log(n))$, where n denotes the size of the queue [33]. On the other hand, the ladder queue [35], an implementation of priority queue, offers an amortized $O(1)$ complexity for both the enqueue and dequeue operations with widely varying priority increment distributions. Based on the good comparative performance evaluation of ladder queue with large simulation events [35], we chose to implement the ladder queue in Golang [19] to replace the min-heap event list. The cross-timeline events are now stored in an unsorted array instead of a min-heap. As a result, the simulation now consumes only 3.41% time for event scheduling as compared with the min-heap event list with the same experimental setting as shown in the Section 3.1. The ladder queue presents better performance as the size of the network grows. For example, the ladder queue reduces the sequential simulation execution time of a 128-node ring network by 4.7%.

We also explore the memoization method [34] to further speed up the simulation. Memoization enables the simulator to cache results of selected expensive functions. We conducted CPU profiling while running QKD network simulation experiments and observed that the transformation of quantum states involves matrix multiplication, which is a CPU-intensive task. Such transformation occurs on almost all photons and causes massive repetitive computation. Therefore, we decided to apply memoization for the quantum state transformation information to speed up the simulation. For example, we performed a CPU profile for the simulation of a two-terminal QKD network as shown in Figure 1 and the matrix calculation consumes around 25.9% time. In a QKD network, the matrix calculation is mainly used for the measurement of detectors with a small portion used for the noise model of quantum channels. For example, researchers measured the noise level at about 5.6% to 7.3% of the total measurement operations from a physical testbed [21]. The quantum state of photon and the basis of measurement are the two primary factors for computing the measurement of detectors. For example, the BB84 protocol requires four quantum states and two bases, so there are at most eight combinations of the quantum state and basis as long as the noise on the quantum channel does not modify the quantum state of photon. For the security of QKD protocol,

Fig. 6. Simulation execution time for a ring network with 128 identical nodes.

the noise on quantum channels cannot be arbitrarily large. Therefore, we assume the majority of transmitted photons are not affected by the noise on the channel. This assumption motivates us to use the **least recently used (LRU)** cache to memorize the results of the matrix calculation. We implemented a fixed-size LRU cache (capacity = 2,000 entries), and each entry of cache consumes 168 bytes. The memory overhead from the LRU cache is around 336 KB. The computational overhead of cache is also low. We measured the time for getting results from the cache and putting results to cache using the sequential simulation of the 128-node ring network. Experimental results indicate that the computational overhead introduced by the LRU cache is very low. It takes 1.59 seconds for the get operations and 5.2 ms for the put operations to complete. The same example in Section 3.1 now only takes 5.75% execution time on the measurement of detectors.

To demonstrate the overall improvement, we set up a ring network with 128 identical nodes and conducted experiments by varying the number of threads. Nodes are evenly distributed on threads to avoid the effect of an unbalanced network partition. Figure 6 depicts the simulation execution time with and without ladder queue and memoization, respectively. We observe that the methods can reduce the execution time for both sequential and parallel simulations. Experiments with different numbers of thread show performance improvement ranging from 33.9% to 39.1% with little variance. Using both methods, the average execution time reduction of all simulations is around 35.2%. We observe that with the increasing number of threads, the performance gain of using ladder queue decreases, whereas the performance gain of using memoization increases. The ladder queue improves the simulation speed by 4.7% in the sequential simulation experiments, and the improvement decreases to 1.1% in the 32-thread parallel simulation experiments. This is because the growing number of threads leads to the reduction of event queue length. The memoization improves the simulation speed by 29.9% in the sequential simulation experiments, and the improvement increases to 37.7% in the 32-thread parallel simulation experiments. Memoization not only reduces the computational time but also performs fewer allocation/release operations of the intermediate data structures used in the cached functions. As a result, the frequency of the garbage collection mechanism of Golang that periodically pauses all threads to free memory is greatly reduced, which leads to significant time savings.

6 NETWORK PARTITION

Our simulator divides a simulation run into multiple cycles, and each cycle contains the event exchange stage and the event processing stage which are bounded by two barriers (see Figure 5). The performance of parallel simulation is highly dependent on (1) a balanced workload for each timeline in the event processing stage, (2) the workload and balance of communication (i.e., moving events to the proper timelines) in the event exchange stage, and (3) the frequency of

synchronization. We investigate how to optimize the network partition scheme that groups objects (e.g., QKD terminals) and assigns them to timelines to achieve a balanced simulation workload, a small number of cross-timeline events, and low synchronization frequency.

A QKD network can be modeled as a weighted direct graph $G(V, A, W)$, where V is the set of QKD terminals, A is a set of quantum channels, and W is the function of workload in a channel. The network partition scheme aims to partition V into at most k groups, where k is the number of timelines, to minimize the simulation execution time. We first model the parallel QKD network simulation using the **bulk synchronous parallel (BSP)** model [37] in Section 6.1, then construct and evaluate the model using linear regression in Section 6.2. The network partition problem is NP-hard, and in this work, we use simulated annealing to search for the optimal scheme as described in Section 6.3.

6.1 BSP-Based Parallel QKD Network Simulation Model

We model the parallel simulation of QKD networks using the BSP model [37]. The BSP model consists of three components: concurrent computation, communication, and barrier synchronization. The timeline in our simulator is equivalent to a processor in the BSP model. The event-processing and event-exchange periods can be represented by the concurrent computation and communication components, respectively. Barrier 2 in Figure 4 stands for the barrier synchronization in the BSP model. With respect to the cost definition in the BSP machine [37], we define the execution time T as

$$T = \sum_{s=1}^S w_s + g \sum_{s=1}^S h_s + Sl, \quad (1)$$

where w_s denotes the event-processing time, h_s denotes the number of exchanged events in the s -th cycle, g denotes the ability of delivering data, S denotes the number of synchronization, and l denotes synchronization overhead. The event-processing time w_s and event-exchange time h_s in the i -th cycle are equal to the maximum execution time of all timelines, which are described as follows:

$$w_s = \max_{i=1}^{|TL|} w_s^i, \quad (2)$$

$$h_s = \max_{i=1}^{|TL|} h_s^i, \quad (3)$$

where h_s^i denotes the event-exchange time and w_s^i denotes the event-processing time in timeline i in the s -th cycle. $|TL|$ denotes the total number of timelines. The objective function is to minimize the total execution time T by changing the scheme of network partition.

The first task is to precisely estimate the time on computing, communication, and barrier synchronization. $\sum_{s=1}^S w_s$, the first component of Formula (1), indicates the time on computing is related to the workload on processors. We use the number of events to model workloads on timelines with the assumption of a linear relationship between the two. As described in Section 3, there are three types of event in our simulator: E_q , E_c , and E_s . E_c and E_s are determined by the QKD network protocol. It is a challenging job to predict E_c and E_s before executing experiments as the QKD network states dynamically evolve over time. QKD terminals, however, have a much simpler state machine than the ones in protocols. The pattern of E_q events can be analyzed and predicted by reading the configuration file. Furthermore, Observations 1 and 2 indicate that E_q events occupy the most running time of a simulation experiment. Based on the analysis, we decide to simplify our optimization model with the focus on E_q as the primary indicator to estimate time on computing.

$(g \sum_{s=1}^S h_s)$, the second component of Formula (1), indicates that the communication time is related to the amount of transmitted information. Thus, we use the number of exchanged events to

model the amount of transmitted information with the assumption of a linear relationship between the two. As E_q events occupy most of the events, we estimate the number of the exchanged events on the quantum channel from the channel configuration.

(Sl), the third component of Formula (1), indicates that the time on the barrier synchronization is related to the number of synchronization. Thus, we use the simulation time of the entire experiment t_{sim} and the lookahead δt (i.e., the quantum channel delay by default) to estimate the number of synchronization. Here, we assume a linear relationship exists between the number of synchronization and the time on the barrier synchronization.

The preceding formulas about time on computing, communication, and synchronization motivate us to use a linear regression model to estimate the time of a specific network partition scheme as described in Section 6.2.

6.2 Model Construction Using Linear Regression

We synthesize a dataset by running 3,107 varying experiments to train (80% dataset) and test (20% dataset) our linear regression models. The number of threads varies from 2 to 32. The computing time, in theory, is only related to the executed events regardless of the number of threads. However, the Intel Turbo Boost Technology [30] and the garbage collection mechanism of Golang [19] degrade the performance of processors with the increasing scale of parallelization. Figure 7(a) shows the computing time of experiments with 2 and 32 threads. We observe that experiments with the same number of threads present a clear linear relationship between the number of executed events and the computing time. Meanwhile, the slope of the line for the 32-thread case is clearly larger than the one for the 2-thread case. The observation implies that the performance of processors decreases with the growing number of threads.

We now use the linear models only containing one feature (i.e., the number of executed events) to predict the time on computing. We define the number of executed events as the aggregation of maximum computing time of all timelines in all the synchronization rounds.

$$N_{\text{executed}} := \sum_{s=1}^S \max_{i=1}^{|TL|} w_s^i \quad (4)$$

The model consisting of just one feature is

$$E\left(\sum_{s=1}^S w_s\right) = a_0 + a_1 * N_{\text{executed}}, \quad (5)$$

where $E(\sum_{s=1}^S w_s)$ denotes the predicted time on computing and N_{executed} denotes the number of executed events by the processors.

The black line in Figure 7(a) represents the predicted time after training. We can observe that the prediction value is around the middle of red and blue dots after fitting data from the varying number of threads. This model presents a bad performance, as the **mean absolute percentage error (MAPE)** of the model is 41.2%, which also increases with the growing number of simulation threads.

To precisely predict the computing time, we introduce two new features: k and $k * N_{\text{executed}}$, where k denotes the number of threads. The feature $k * N_{\text{executed}}$ is used to adjust the slope of the model with the different number of threads. The new model is described as Formula (6).

$$E\left(\sum_{s=1}^S w_s\right) = a_0 + a_1 * k + a_2 * N_{\text{executed}} + a_3 * k * N_{\text{executed}} \quad (6)$$

Fig. 7. Performance of simple linear regression models on simulation experiments with 2 threads and 32 threads (dots are randomly sampled from the whole dataset).

Fig. 8. Predicted time versus actual time on computing, communication, and synchronization (dots are randomly sampled from the whole dataset).

Table 1. Coefficients and p -Values of the Computing Time Model

	a0	a1	a2	a3
Coefficient	2.31	3.75e-06	-8.93e-02	1.77e-07
p -Value	<0.01	<0.01	<0.01	<0.01

The coefficients and p -value of our model are presented in Table 1. The small p -values imply that all the variables are significantly related to the actual execution time. The R-square of the training set is 0.9977. The high R-square values imply that our model accurately estimates the simulation execution time. Figure 8(a) shows the performance of the updated linear regression model. The x-axis is the predicted time, and the y-axis is the actual time. The black diagonal represents the perfect result in theory. We observe all data points are located around the diagonal line. The MAPE of the updated model is reduced to 6.9%.

Formula (6) requires two features before running the simulation: the number of threads and the number of executed events. Unlike the number of threads, we cannot directly retrieve the number of executed events from the configuration files. Therefore, we need to predict N_{executed} based on the configuration of the simulated network. Observation 2 implies that the execution time is predictable given the number of E_q events. Therefore, it is reasonable to assume that all the quantum channels are used to generate QKD keys. As a result, the number of E_q events is

Table 2. Coefficients and p -Values of the Communication Time Model

	a0	a1	a2	a3
Coefficient	5.30e-04	2.54e-07	6.30e-05	7.17e-10
p -Value	<0.05	<0.01	<0.01	<0.01

static—that is, it can be predicted by the attributes of light sources and channels regardless of whether a quantum channel uses time-division multiplexing or wavelength-division multiplexing. A light source decides the sending rate τ_s of a quantum channel. In other words, $\tau_s = (1 - e^{-\mu}) * f$, where f is the frequency of a light source and μ is the mean of a Poisson distribution. The receiving rate, τ_r , is expressed as $\tau_r = \tau_s(1 - l)$, where l is the transmission loss rate.

To compute the number of E_q events in one timeline, we aggregate all E_q events generated by the QKD terminals aligned to the same timeline. The expected number of E_q is

$$E(N_{\text{executed}}) = t_{\text{sim}} \cdot \max_{i=1}^{|TL|} \left(\sum_{v \in G_i} \sum_{a \in A^-(v)} \tau_s(a) + \sum_{a \in A^+(v)} \tau_r(a) \right), \quad (7)$$

where G_i denotes the group of QKD terminals in the i -th timeline, and $A^+(v)$ and $A^-(v)$ denote the incoming and outgoing quantum channels of a QKD terminal v . The combination of Formula (6) and Formula (7) enables us to predict the computing time.

Similarly, the communication time also suffers from performance degradation as shown in Figure 7(b). The time on communication is defined as the aggregation of the maximum communication time in every communication stage as shown by the Formula (8).

$$N_{\text{exchanged}} := \sum_{s=1}^S \max_{i=1}^{|TL|} gh_s^i \quad (8)$$

We observe the divergence between the 2-thread data and 32-thread data even though the divergence is not as large as the time on computing. The black line is generated from a theoretical model consisting of one feature (i.e., the number of exchanged events), which is shown in Formula (9). The MAPE of this model is 11.0%.

$$E\left(g \sum_{s=1}^S h_s\right) = a0 + a1 * N_{\text{exchanged}} \quad (9)$$

We then update the model to include more features shown as Formula (10).

$$E\left(g \sum_{s=1}^S h_s\right) = a0 + a1 * N_{\text{exchanged}} + a2 * k + a3 * N_{\text{exchanged}} * k \quad (10)$$

We get the coefficients and p -values after training and list them in Table 2. The p -values show the high significance of variables for the model. The R-square in the training dataset is 0.9829. The high values of R-square indicate the effectiveness of our model. Figure 8(b) presents the performance of the updated model. We observe that all data points are located around the diagonal line. The MAPE of the updated model is 10.6%.

Similar to the prediction of N_{executed} , we utilize the configurations of quantum channels to predict $N_{\text{exchanged}}$. We first predict the number of exchanged events for every timeline using the in-channel parameters of the timeline. Then, we multiply the maximum number of exchanged events

Table 3. Coefficients and p -Values of the Synchronization Time Model

	a0	a1	a2
Coefficient	2.50e-03	8.61e-05	6.06e-05
p -Value	<0.01	<0.01	<0.01

with the simulation time to predict $N_{\text{exchanged}}$. The prediction formula is shown as Formula (11).

$$E(N_{\text{exchanged}}) = t_{\text{sim}} \cdot \max_{i=1}^{|TL|} \left(\sum_{v \in G_i} \sum_{u \notin G_i} \tau_u^v \right) \quad (11)$$

Note that $\tau_u^v := \tau_r(a)$, where u and v are the source and destination of channel a .

Figure 7(c) shows the synchronization time of experiments executed using 2 threads and 32 threads. We observe a clear divergence between the 2-thread and 32-thread results. This observation indicates that the linear regression model with only one attribute is insufficient for predicting synchronization time accurately. Even the theoretical model shown in Formula (12) has 37.1% MAPE.

$$E(SI) = a0 + a1 * S \quad (12)$$

To address this issue, we update the model with a new feature, $k * S$, to adjust the slope of the line.

$$E(SI) = a0 + a1 * S + a2 * S * k \quad (13)$$

Table 3 shows the coefficients and p -values of the features after training. The test dataset shows that the MAPE of Formula (13) is improved to 12.2%. Figure 8(c) shows the performance of our prediction model of synchronization time. We observe that all dots are closely surrounded by the line of the perfect model.

We also need to predict S before running the simulation. The lookahead, δt , can be obtained by finding the minimum propagation delay among the cross-timeline channels. We use Formula (14) to predict the number of synchronization.

$$E(S) := \frac{t_{\text{sim}}}{\delta t} \quad (14)$$

Finally, we get the predicted total execution time by aggregating the predicted time on computing, communication, and synchronization.

$$E(T) := E\left(\sum_{s=1}^S w_s\right) + E\left(g \sum_{s=1}^S h_s\right) + E(SI) \quad (15)$$

We synthesize 2,624 random QKD networks and vary the number of threads from 2 to 32 with a random partition scheme to evaluate the model performance. The MAPE of the model is 5.53%, and the standard deviation of the absolute percentage error is 4.62%.

We also attempted to apply Formula (15) for each synchronization round. However, our experimental results show that the measured time has a large variance (e.g., the maximum time can be twice larger than the minimum time for the same amount of workload). This is because of the unpredictable behavior of the garbage collection in Golang [19] that could pause all threads for the stop-the-world mechanism. Therefore, we decide to use the aggregated time on computing, communication, and synchronization for our models.

6.3 Graph Partition Using Simulated Annealing

Another task is to generate a balanced graph partition. We want to divide a graph G into two equal-size sets and minimize the number of edges going from one set to the other, which is an NP-hard problem [31]. The optimal solution of the graph partition problem is the optimal solution of the network partition scheme if we preserve the same τ_s of channels and the same event-exchange time for different solutions. Since the graph partition problem is NP-hard, the network partition problem is also NP-hard. Existing work [36] revealed that the simulated annealing is more powerful than the Kernighan-Lin approach [20] to solve the graph partition problem. In this work, we utilize simulated annealing to find our network partition scheme.

Algorithm 1 describes how to partition the graph using simulated annealing, where $E(\text{state})$ denotes the energy of a particular state, state_B and energy_B denote the optimal state and its energy, state_N and energy_N denote the neighbor state and its energy, and T denotes the temperature. The $\text{INIT-STATE}()$ function randomly assigns QKD terminals to timelines. The energy of the state is calculated by Formula (15). The $\text{NEIGHBOR}()$ function randomly reassign a QKD terminal to a different timeline. The acceptance probability is described as follows.

$$P(\text{energy}, \text{energy}_N, T) = e^{(\text{energy}_N - \text{energy})/T} \quad (16)$$

The output of state_B describes the optimized scheme of the network partition for efficient parallel simulation.

ALGORITHM 1: Graph Partition Using Simulated Annealing

```

1: state  $\leftarrow$  INIT-STATE(), energy  $\leftarrow$  E(state)
2: stateB  $\leftarrow$  state, energyB  $\leftarrow$  energy
3: T  $\leftarrow$  n
4: while T > 0 do
5:   stateN  $\leftarrow$  NEIGHBOR(state), energyN  $\leftarrow$  E(energyN)
6:   if energyN < energyB then
7:     stateB  $\leftarrow$  stateN, energyB  $\leftarrow$  energyN
8:   end if
9:   if P(energy, energyN, T) > RAND() then
10:    state  $\leftarrow$  stateN, energy  $\leftarrow$  energyN
11:   end if
12:   T  $\leftarrow$  T - 1
13: end while
14: return stateB

```

In this work, the optimization of partition is based on the static traffic flows in the simulated network. For dynamic traffic flows, our partition algorithm may not always ensure the best performance (e.g., some hardware can be “lazy” at a certain time). One approach to enhance the optimization model in our future work is to monitor the simulated traffic flows and then adaptively generate the partition schemes on the fly.

7 EVALUATION

We evaluate the performance of our parallel simulator of QKD networks in terms of execution speed and scalability with various network scenarios. We also compare the performance between our network partition scheme and a random network partition scheme. Furthermore, we present the error analysis results of the network partition scheme.

We have implemented the parallel simulator using Golang [19]. To set up the experiments, we generated multiple QKD networks from 45 random directed graphs $G(n, d, seed)$. Here, n denotes the number of vertices (i.e., QKD terminals) where $n \in \{80 \dots 128\}$ with an increase of 8, d denotes the expected degree of vertex where $d \in \{1.5, 2, 2.5\}$, and $seed$ denotes the random seed where $seed \in \{0, 1, 2\}$. Each quantum channel was attached to a light source of one QKD terminal and measurement devices of the other QKD terminal. The light source generated attenuated pulses with frequency f . The number of emitted photons in the pulses followed a Poisson distribution with mean μ . The frequency f was randomly selected from 10^6 to 10^8 Hz, and the mean μ was set to 0.1. The propagation time of the photons was $\frac{L}{c^*}$. L is the distance of each quantum channel, and the value was randomly selected from 5 to 15 km. c^* is the speed of light in the fiber, and the value was set to $2 * 10^8$ m/s. The loss rate of the quantum channel was $10^{-\frac{L \cdot \alpha_0}{10}}$, where α_0 was the attenuation measured in 0.2 dB/km. The classical channels were created between two QKD terminals connected by quantum channels. The network delay on the classical channels is set to 1 ms. The simulation experiments run for 20 ms using 2 to 32 threads, which is sufficient to demonstrate all the stages of the QKD protocol. Each simulation experiment generates and executes millions of events. The maximum number of effective threads is limited by the number of CPU cores of the server—that is, 40 cores of a 2.6-GHz Dual Intel Xeon and 64 GB of RAM.

7.1 Simulation Speedup Evaluation with Different Network Partitions

We repeated each simulation experiment 10 times using a random partition scheme and the optimized partition scheme discussed in Section 6. The optimized partition was produced by the simulated annealing method with the temperature of $T = 10^5$. For each network scenario, we recorded the execution time with both random partition and optimized partition. We then calculated the improvement of execution time ρ for the optimized partition scheme T_{opt} over the random partition scheme T_{rand} , in particular, $\rho = \frac{T_{rand} - T_{opt}}{T_{rand}}$. We plotted the execution time improvement with respect to the number of threads in Figure 9.

We observe that the optimized partition scheme always outperforms the random partition scheme for all network scenarios. Since we generated QKD networks from 45 random directed graphs, the size of the network largely varies. ρ is computed based on different network scenarios with respect to the number of threads. Therefore, it is not surprising to see a large standard deviation of ρ . Furthermore, the performance of our network partition scheme increases as the number of threads grows. The average of ρ is increased from 0.04 to 0.38 when the number of threads is increased from 2 to 32.

We now fix the network scenario to $G(128, 2.5, 2)$ as an example and demonstrate the performance gain with our optimized network partition scheme. For both randomized partition and optimized partition schemes, we plot the mean and standard deviation of the execution time in Figure 10(a). First, the execution time is improved from 184 seconds (sequential simulation with 1 thread) to 18 seconds (optimized network partition with 32 threads)—that is, 9.38 times faster. Second, the optimized partition scheme always outperforms the randomized partition. Third, the parallel simulation performance increases as the number of threads grows, and the results match well with the ones in Figure 9. However, with 10 repeated runs, the standard deviation of the execution time is as little as up to 0.81 seconds.

7.2 Scalability Evaluation with Different Network Partitions

To evaluate the scalability of our simulator, we fixed the network scenario as $G(128, 2.5, 2)$, repeated the experiments only by increasing the number of threads, and then measured the experiment execution time with and without the optimized network partition. We also ran the phold

Fig. 9. Improvement of execution time ρ for the optimized network partition scheme T_{opt} over the randomized network partition scheme T_{rand} , and $\rho = \frac{T_{rand} - T_{opt}}{T_{rand}}$.

(a) Simulation execution time for (1) random network partition and (2) optimized network partition

(b) Normalized execution time with (1) random network partition, (2) optimized network partition, and (3) phold (benchmark)

Fig. 10. Execution time of simulating the QKD network $G(128, 2.5, 2)$.

experiments, a well-known parallel simulation benchmark, for comparison. In the phold experiment, we initialized 1.024×10^6 jobs that are evenly distributed over the threads (i.e., perfect simulation workload balance), and the experiments all finished after the 10th event-exchange period. We normalized the execution time based on the execution time using two threads, and plotted the execution time for phold, randomized partition, and optimal partition in Figure 10(b).

We observe that as the number of threads grows over 2, the optimized partition scheme always scales better than the random partition scheme. The normalized execution time of the optimized partition is very close to the performance of phold, and the maximum difference is just around 0.02. The result implies that our network partition model manages to generate balanced workloads (close to the optimal solution) to enhance the scalability of our parallel QKD simulator.

7.3 Error Analysis of Execution Time Estimation

We perform an error analysis of the execution time predicted by the linear regression model. We first collected the energy of the random partition scheme and the optimized partition scheme using simulated annealing. The estimated execution times are then compared with the real execution time. The **mean absolute errors (MAEs)** of the model are plotted in Figure 11(a).

We observe that the MAE of our model is always lower than 4 seconds. Therefore, it is reliable to use the outputs of our objective function to compare the relative quality between various solutions and then schedule jobs on the shared computational resources. Additionally, the MAEs computed from the 32-thread cases are larger than the 2-thread cases for both random and optimized partition schemes because of the shorter execution time on 32-thread cases. Figure 11(b) is a scatter plot that shows the relationship between the predicted time and the actual time for simulating 45 QKD networks using multi-threads ranging from 2 to 32. We observe that the predicted time and the actual time are very close, and the performance is even better with a longer execution time.

(a) Mean absolute error (MAE) of execution time estimation

(b) Predicted time versus actual time

Fig. 11. Performance of the linear regression prediction model for random partitions and optimized partitions.

Although the evaluation results show that the linear regression model works effectively for predicting QKD network simulation workload based on Observations 1, 2, and 3, the model has limitations. The main limitation is the assumption of linearity—in this case, the mean of the response variable is a linear combination of the parameters (regression coefficients) and the predictor variables. Although we observe the performance of the processor degrades with the number of threads, it is still uncertain such degradation exhibits a linear relation. In addition, the inputs to our model may contain errors because the number of executed events and exchange events are directly estimated from the network configuration. Our model inputs include the predicted workloads on computing (Formula (7)), communication (Formula (11)), and synchronization (Formula (14)). The errors in each component can be aggregated when predicting the total execution time (Formula (15)). We plan to investigate learning-based models to further improve accuracy as future works.

8 CONCLUSION AND FUTURE WORK

The increasing size of QKD networks calls for scalable simulation tools. In this article, we studied the feasibility of parallel simulation of QKD networks based on the analysis of QKD network properties in terms of event types, quantity, workload, and interaction. We then developed a QKD network parallel simulator using conservative synchronization. A key component is an optimized network partition scheme based on linear regression models and simulated annealing to achieve a well-balanced simulation workload for parallel simulation. We also conducted extensive evaluation experiments on simulation scalability, efficiency, and prediction error analysis.

In the future, we plan to apply other methods to explore solutions to the network partition problem, such as replacing simulated annealing with deep reinforcement learning or a dynamically balancing workload. We also plan to investigate quantum entanglement and develop the corresponding scalable simulation tools. Last, we will study hardware-in-the-loop simulation [42] to further improve the simulation testbed accuracy.

REFERENCES

- [1] Julio T. Barreiro, Nathan K. Langford, Nicholas A. Peters, and Paul G. Kwiat. 2005. Generation of hyperentangled photon pairs. *Physical Review Letters* 95, 26 (2005), 260501.
- [2] Julio T. Barreiro, Tzu-Chieh Wei, and Paul G. Kwiat. 2008. Beating the channel capacity limit for linear photonic superdense coding. *Nature Physics* 4, 4 (2008), 282.
- [3] Sean D. Barrett and Pieter Kok. 2005. Efficient high-fidelity quantum computation using matter qubits and linear optics. *Physical Review A* 71, 6 (2005), 060310.
- [4] Ben Bartlett. 2018. A distributed simulation framework for quantum networks and channels. *arXiv preprint arXiv:1808.07047* (2018).

- [5] Charles H. Bennett and Gilles Brassard. 2014. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science* 560, 12 (2014), 7–11.
- [6] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. 1993. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters* 70, 13 (1993), 1895.
- [7] Charles H. Bennett, Gilles Brassard, Claude Crépeau, and Ueli M. Maurer. 1995. Generalized privacy amplification. *IEEE Transactions on Information Theory* 41, 6 (1995), 1915–1923.
- [8] Gilles Brassard and Louis Salvail. 1993. Secret-key reconciliation by public discussion. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*. 410–423.
- [9] Randy Brown. 1988. Calendar queues: A fast $O(1)$ priority queue implementation for the simulation event set problem. *Communications of the ACM* 31, 10 (1988), 1220–1227.
- [10] W. T. Buttler, R. J. Hughes, P. G. Kwiat, S. K. Lamoreaux, G. G. Luther, G. L. Morgan, J. E. Nordholt, C. G. Peterson, and C. M. Simmons. 1998. Practical free-space quantum key distribution over 1 km. *Physical Review Letters* 81, 15 (Oct. 1998), 3283–3286. <https://doi.org/10.1103/PhysRevLett.81.3283>
- [11] W. T. Buttler, R. J. Hughes, P. G. Kwiat, G. G. Luther, G. L. Morgan, J. E. Nordholt, C. G. Peterson, and C. M. Simmons. 1998. Free-space quantum-key distribution. *Physical Review A* 57, 4 (April 1998), 2379–2382. <https://doi.org/10.1103/PhysRevA.57.2379>
- [12] D. Castelvecchi. 2018. The quantum internet has arrived (and it hasn't). *Nature* 554 (Feb. 2018), 289–292. <https://doi.org/10.1038/d41586-018-01835-3>
- [13] Rishab Chatterjee, Kaushik Joarder, Sourav Chatterjee, Barry C. Sanders, and Urbasi Sinha. 2019. qkdSim: An experimenter's simulation toolkit for QKD with imperfections, and its performance analysis with a demonstration of the B92 protocol using heralded photon. *arXiv preprint arXiv:1912.10061* (2019).
- [14] Tim Coopmans, Robert Knegjens, Axel Dahlberg, David Maier, Loek Nijsten, Julio Oliveira, Martijn Papendrecht, et al. 2020. NetSquid, a discrete-event simulation platform for quantum networks. *arXiv preprint arXiv:2010.12535* (2020).
- [15] Rachel Courtland. 2016. China's 2,000-km quantum link is almost complete. *IEEE Spectrum* 53, 11 (Nov. 2016), 11–12.
- [16] Axel Dahlberg and Stephanie Wehner. 2018. SimulaQron—A simulator for developing quantum internet software. *Quantum Science and Technology* 4, 1 (2018), 015001.
- [17] J. F. Dynes, Adrian Wonfor, W. W.-S. Tam, A. W. Sharpe, R. Takahashi, M. Lucamarini, A. Plews, et al. 2019. Cambridge quantum network. *npj Quantum Information* 5, 1 (2019), 1–8.
- [18] Chip Elliott, Alexander Colvin, David Pearson, Oleksiy Pikalo, John Schlafer, and Henry Yeh. 2005. Current status of the DARPA quantum network. In *Quantum Information and Computation III*, Vol. 5815. International Society for Optics and Photonics, 138–149.
- [19] Google. 2020. The Go Programming Language. Retrieved November 19, 2021 from <https://golang.org/>.
- [20] Brian W. Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* 49, 2 (1970), 291–307.
- [21] M. Khodr. 2017. Evaluations of quantum bit error rate using the three stage multiphoton protocol. In *Proceedings of the 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA'17)*. 1–4. <https://doi.org/10.1109/ICECTA.2017.8251929>
- [22] Taehyun Kim, Ingo Stork Genannt Wersborg, Franco N. C. Wong, and Jeffrey H. Shapiro. 2007. Complete physical simulation of the entangling-probe attack on the Bennett-Brassard 1984 protocol. *Physical Review A* 75, 4 (2007), 042327.
- [23] H. Jeff Kimble. 2008. The quantum internet. *Nature* 453, 7198 (2008), 1023–1030.
- [24] Ulana Legedza and William E. Weihl. 1996. Reducing synchronization overhead in parallel simulation. In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*. 86–95.
- [25] Ivan Marcikic, Hugues De Riedmatten, Wolfgang Tittel, Hugo Zbinden, Matthieu Legré, and Nicolas Gisin. 2004. Distribution of time-bin entangled qubits over 50 km of optical fiber. *Physical Review Letters* 93, 18 (2004), 180502.
- [26] Takaaki Matsuo, Clément Durand, and Rodney Van Meter. 2019. Quantum link bootstrapping using a RuleSet-based communication protocol. *Physical Review A* 100, 5 (2019), 052320.
- [27] Miralem Mehic, Oliver Maurhart, Stefan Rass, and Miroslav Voznak. 2017. Implementation of quantum key distribution network simulation module in the network simulator NS-3. *Quantum Information Processing* 16, 10 (2017), 253.
- [28] David M. Nicol. 1996. Principles of conservative parallel simulation. In *Proceedings of the 28th Winter Simulation Conference*. 128–135.
- [29] Momtchil Peev, Christoph Pacher, Romain Alléaume, Claudio Barreiro, Jan Bouda, W. Boxleitner, Thierry Debuisschert, et al. 2009. The SECOQC quantum key distribution network in Vienna. *New Journal of Physics* 11, 7 (2009), 075001.
- [30] Dhananjai M. Rao. 2018. Performance comparison of cross memory attach capable MPI vs. multithreaded optimistic parallel simulations. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 37–48.

- [31] Theresa Rose. 2018. Heuristic research. In *What Is Psychotherapeutic Research?* Routledge, 133–143.
- [32] Nicolas Sangouard, Christoph Simon, Hugues De Riedmatten, and Nicolas Gisin. 2011. Quantum repeaters based on atomic ensembles and linear optics. *Reviews of Modern Physics* 83, 1 (2011), 33.
- [33] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Self-adjusting binary search trees. *Journal of the ACM* 32, 3 (1985), 652–686.
- [34] Mirko Stoffers, Daniel Schemmel, Oscar Soria Dustmann, and Klaus Wehrle. 2018. On automated memoization in the field of simulation parameter studies. *ACM Transactions on Modeling and Computer Simulation* 28, 4 (2018), 1–25.
- [35] Wai Teng Tang, Rick Siow Mong Goh, and Ian Li-Jin Thng. 2005. Ladder queue: An $O(1)$ priority queue structure for large-scale discrete event simulation. *ACM Transactions on Modeling and Computer Simulation* 15, 3 (2005), 175–204.
- [36] L. Tao, Y. C. Zhao, K. Thulasiraman, and M. N. S. Swamy. 1992. Simulated annealing and tabu search algorithms for multiway graph partition. *Journal of Circuits, Systems, and Computers* 2, 02 (1992), 159–185.
- [37] Leslie G. Valiant. 1990. A bridging model for parallel computation. *Communications of the ACM* 33, 8 (1990), 103–111.
- [38] Nino Walenta and Lee Oesterling. 2019. Quantum networks: Photons hold key to data security. *Photonics Media*. Retrieved November 19, 2021 from https://www.photonics.com/Articles/Quantum_Networks_Photons_Hold_Key_to_Data/a60541.
- [39] Xiaoliang Wu, Joaquin Chung, Alexander Kolar, Eugene Wang, Tian Zhong, Rajkumar Kettimuthu, and Martin Suchara. 2019. Photon-level simulation of quantum key distribution with picosecond accuracy. In *Proceedings of the 2019 Single Photon Workshop*.
- [40] Xiaoliang Wu, Joaquin Chung, Alexander Kolar, Eugene Wang, Tian Zhong, Rajkumar Kettimuthu, and Martin Suchara. 2019. Simulations of photonic quantum networks for performance analysis and experiment design. In *Proceedings of the 2019 IEEE/ACM Workshop on Photonics-Optics Technology Oriented Networking, Information and Computing Systems (PHOTONICS'19)*. IEEE, Los Alamitos, CA, 28–35.
- [41] Xiaoliang Wu, Alexander Kolar, Joaquin Chung, Dong Jin, Tian Zhong, Rajkumar Kettimuthu, and Martin Suchara. 2020. SeQUnCe: A customizable discrete-event simulator of quantum networks. *arXiv preprint arXiv:2009.12000* (2020).
- [42] Xiaoliang Wu, Qi Yang, Xin Liu, Dong Jin, and Cheol Won Lee. 2017. A hardware-in-the-loop emulation testbed for high fidelity and reproducible network experiments. In *Proceedings of the 2017 Winter Simulation Conference (WSC'17)*. IEEE, Los Alamitos, CA, 408–418.
- [43] Yong Yu, Fei Ma, Xi-Yu Luo, Bo Jing, Peng-Fei Sun, Ren-Zhou Fang, Chao-Wei Yang, et al. 2020. Entanglement of two quantum memories via fibres over dozens of kilometres. *Nature* 578, 7794 (2020), 240–245.

Received December 2020; revised September 2021; accepted September 2021