Parallel Simulation of Quantum Key Distribution Networks

Xiaoliang Wu
Illinois Institute of Technology
Chicago, United States
xwu64@hawk.iit.edu

Bo Zhang
Illinois Institute of Technology
Chicago, United States
bzhang43@hawk.iit.edu

Dong Jin
Illinois Institute of Technology
Chicago, United States
dong.jin@iit.edu

ABSTRACT

With the significantly growing investment in quantum communications, quantum key distribution (QKD), as a key application to share a security key between two remote parties, has been deployed in urban areas and even at a continental scale. To meet the design requirements of QKD on a quantum communication network, today researchers extensively conduct simulation-based evaluations in addition to physical experiments for cost efficiency. A practical QKD system must be implemented on a large scale via a network, not just between a few pairs of users. Existing discrete-event simulators offer models for QKD hardware and protocols based on sequential execution. In this work, we investigate the parallel simulation of QKD networks for scalability enhancement. Our contributions lay in the exploration of QKD network characteristics to be leveraged for parallel simulation. We also develop a parallel simulator for QKD networks with an optimized scheme for network partition. Experimental results show that to simulate a 64-node QKD network, our parallel simulator can complete the experiment 9 times faster than a sequential simulator running on the same machine. Our linear-regression-based network partition scheme can further accelerate the simulation experiments up to two times than using a randomized network partition scheme.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Computing methodologies** → **Discrete-event simulation**.

KEYWORDS

parallel discrete-event simulation, quantum key distribution, BB84

ACM Reference Format:
Xiaoliang Wu, Bo Zhang, and Dong Jin. 2020. Parallel Simulation of Quantum Key Distribution Networks. In *2020 SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS'20), June 15–17, 2020, Miami, FL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3384441.3395988

1 INTRODUCTION

The development of quantum physics leads to many emerging and rapidly growing applications, founded by quantum states, to transform the way we process information today. Quantum key distribution (QKD) is a critical application that offers solutions for secure key distribution through insecure communication channels. When an eavesdropper intercepts classical bits shared between Alice and Bob, the interception must disturb the associated quantum state, and thus be detected by Alice and Bob. The law of quantum physics helps to defend such attacks against adversaries. The unique property motivates people to build quantum networks for secure communication. Several photonic quantum networks have been established including a 30-mile optical fiber link connecting Argonne and Fermilab [20], a 2,000-kilometer fiber-optic link between Beijing and Shanghai [12], and other projects in the Netherlands [9], the United Kingdom [14], and South Korea [28].

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 [22], in which they do not simulate the transformation in the quantum channels, like transmission and transformation of photons. SQUANCH is an agent-based simulator for quantum networks. As an agent-based simulator [4], it 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 [13] and qkdSim [10] 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 [29, 30] takes around 120 seconds to simulate 200 ms of a 64-node QKD network (see Figure 7 in Section 6 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. To efficiently parallelize a QKD network simulation experiment, we develop a network partition scheme using a linear regression based prediction model. As searching for the optimal solution is NP-hard, we use simulated annealing to find the approximate solution. Finally, we conduct extensive experiments to evaluate the simulation scalability and speedup under various QKD network settings and perform an error analysis of the execution time prediction model. Results show that to simulate a 64-node QKD network using the same physical machine, our parallel simulator significantly reduces the execution

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.
SIGSIM-PADS '20, June 15–17, 2020, Miami, FL, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7592-4/20/06…$15.00
https://doi.org/10.1145/3384441.3395988

time from 122 seconds in the sequential mode to 13 seconds using 32 threads. Also, the optimized network partition scheme can further accelerate the simulation time up to 117% more than the random network partition scheme. The contributions of this work are listed as follows.

- 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 develop a network partition scheme to improve the simulation scalability.

The remainder of the paper is organized as follows, Section 2 introduces the background of QKD protocols and models of optics. Section 3 explores the characteristics of QKD networks from existing QKD network simulators and physical QKD networks. Section 4 presents the architecture of our QKD network parallel simulator. Section 5 describes the network partition scheme. In Section 6, we evaluate the performance of our simulator in terms of speedup, scalability, and error analysis. Section 7 concludes the paper with future work.

2 BACKGROUND

The advancement of quantum technologies re-defines 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 quantum key distribution (QKD) [2, 18, 21]. QKD, as the best-known application, 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 any 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 [5] proposed by Charles H. Bennett and Gilles Brassard in 1984. 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 polarized beam splitter; the clicked single-photon detector 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 [15]. These protocols only rely on the classical channel to achieve their functions, so they are not studied in this paper.

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. An detector is capable of accurately recording photon arrival times. A splitter is used for separating the photons based on polarization under either conjugate basis.

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.

Quantum channel (QC), 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 dB/km.

Single-Photon Detector (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.

Polarized Beam splitter (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.

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 conduct experiments using a sequential simulator to understand the characteristics of QKD networks in Section 3.1. We then analyze QKD networks from simulation models as well as existing QKD networks in Section 3.2. Five observations that one can leverage for designing parallel simulation of QKD networks are presented as follows.

3.1 Analysis of Sequential Simulation of QKD Networks

We empirically study the behaviors of a sequential QKD simulator using our prior work [29, 30] including the type of simulation events, the execution speed of different event types, and the distribution of events during the simulation experiments. We present the 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 consists of a polarized beam splitter (PBS) and two single-photon detectors (SPD). 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.

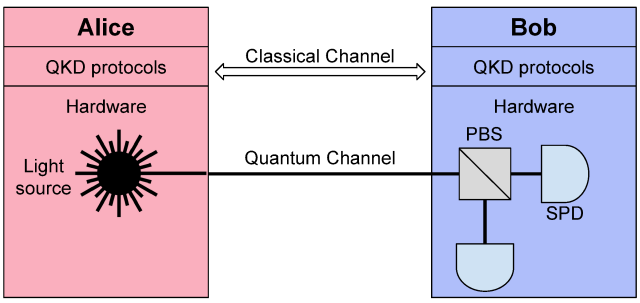
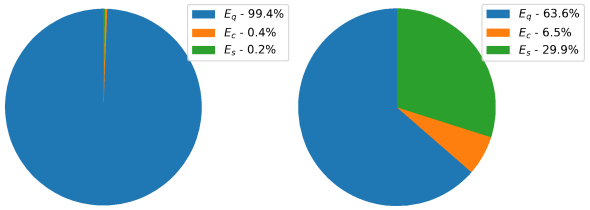


Figure 1: Simulation of a Two-Terminal QKD Network

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.



(a) Number of events **(b) Execution time**

Figure 2: Comparison of three simulation event types, E_q - events on a quantum channel, E_q - events on a classical channel, and E_s - events inside one QKD terminal

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 memory. The network successfully distributed 455 keys between Alice and Bob. Totally 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. Below 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 2a, 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 2b 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 5 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 μsec) is around 30 times smaller than the one of E_c (140 μsec) and around 6 times smaller than the one of E_s (947 μsec). E_q has the lowest variance among all three types, i.e., the standard deviation of E_c is more than 26 times of E_q. 95% of E_q events are executed within 9.6 μsec. 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 5.1 based on the prediction.

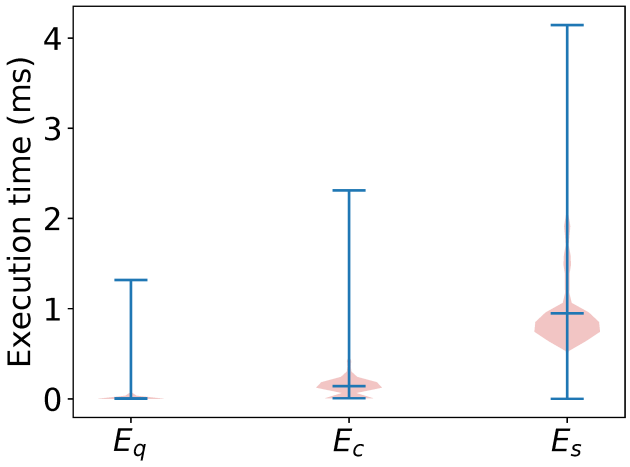


Figure 3: Distribution of individual simulation event execution time, E_q - events on a quantum channel, E_q - events on a classical channel, and E_s - events inside one QKD terminal

Observation 3. *E_q events between two QKD terminals are evenly distributed over 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 5.1.

3.2 Analysis of Existing QKD Networks

Existing QKD networks, like DARPA [15] and SECOQC [24], 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 one hundred kilometers fiber can be as large as around 99%. Furthermore, QKD network does not route or process quantum information unlike what happens in a classical network. 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 one picosecond, 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 ten QKD terminals [15]. 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, 26, 32]. 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 the section 7 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 [23]. In this work, we take the barrier-based synchronization approach to develop a parallel simulator for QKD networks [19].

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 BB84 protocol, light source,

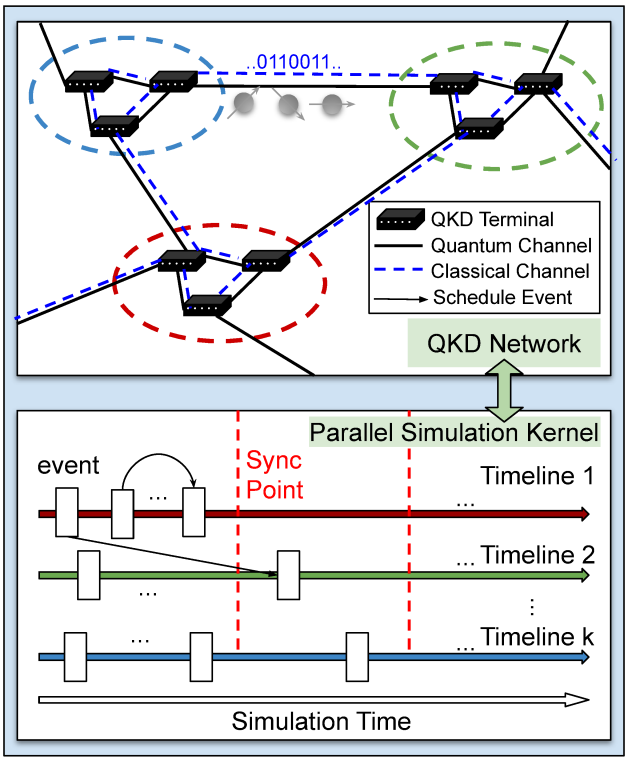


Figure 4: Architecture of Simulator

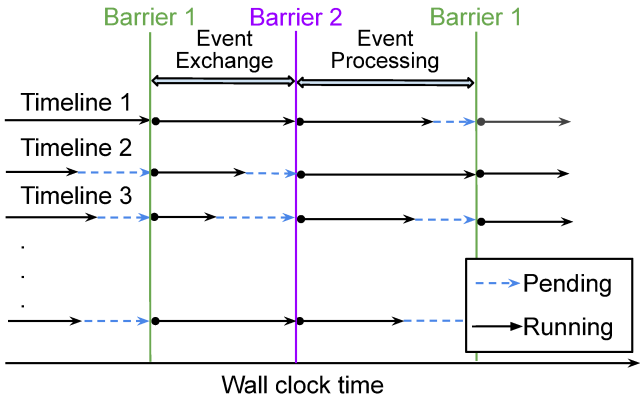


Figure 5: Barrier synchronization. "Pending" means that a timeline is blocked by a barrier and waiting for other timelines. "Running" means that a timeline advances its simulation time by executing events in its own queue.

single-photon detector, polarized beam splitter, 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 attached to a timeline, which hosts an event queue and is in charge of advancing all objects attached to it. Interactions between objectives within one timeline do not need synchronization other than executing the event queue. The event queue is implemented as a min-heap sorted by 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 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 NETWORK PARTITION

As shown in Figure 5, a simulation experiment is divided into multiple simulation cycles, and each cycle contains two stages, i.e., the event exchange and event processing, bounded by the two barriers. The performance of parallel simulation is highly dependent on (1) a balanced workload for each timeline in the event processing stage, (2) the number of events to exchange (i.e, merging to the target timelines) in the event exchange stage, and (3) the frequency of synchronization. Therefore, we need to investigate how to partition a QKD network to meet those requirements. An optimized network partition scheme groups objects (e.g., QKD terminals) and assigns them to timelines with the objective of balanced workload, small cross-timeline events, and low synchronization frequency, and thus achieve good scalability of parallel simulation.

Given a partition scheme, the simulation execution time T is defined as

$$T = \sum_{0<i\leq C} (T_s^i + T_p^i) \quad (1)$$

where C denotes the number of synchronization, T_s^i denotes the event-exchange time and T_p^i denotes the event-processing time in the i^{th} cycle respectively. The event-processing time T_s and event-exchange time T_p in the i-th cycle equal to the maximum execution time among all timelines, which are described as follows.

$$T_s^i = max_{1\leq j\leq |TL|} t_s^{ij} \quad (2)$$

$$T_p^i = max_{1\leq j\leq |TL|} t_p^{ij} \quad (3)$$

where the t_s^{ij} denotes the event-exchange time and t_p^{ij} denotes the event-processing time in timeline j in i^{th} cycle respectively. |TL| denotes the total number of timelines. The objective function is to minimize the total execution time.

The first task is to precisely estimate t_s^{ij} and t_p^{ij}. As described in Section 3, there are three types of event, E_q, E_c, and E_s. E_c and E_s is 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, on the other hand, 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 t_s^{ij} and t_p^{ij}.

Let us define the number of cycles as $\frac{t_{sim}}{\delta t}$, where t_{sim} denotes the simulation time of the entire experiment and δt denotes the lookahead time (i.e., the quantum channel delay by default). Based on Observation 2 and 3 and the fact that QKD protocols greedily produce keys to upper applications, we assume that every cycle executes the same amount of E_q events. As a result, every cycle takes the same time for event-exchange (T_s) and event-processing (T_w). The updated formula of simulation execution time is presented as follows, which leads us to use a linear regression model to estimate time as described in Section 5.1.

$$T = (T_s + T_p) * (\frac{t_{sim}}{\delta t}) \quad (4)$$

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. This problem is NP-hard, and in this work we use simulated annealing to search for the optimal scheme as described in Section 5.2.

5.1 Model Construction Using Linear Regression

Observation 2 implies that the execution time is predictable given the number of E_q events. The number of E_q events can be predicted by the attributes of light sources and channels regardless if the 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, $\tau_s = f * \mu$, 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 number of E_q in one cycle is

$$N_e = \delta t \sum_{v\in S} \sum_{a\in A^-(v)} \tau_s(a) + \sum_{a\in A^+(v)} \tau_r(a) \quad (5)$$

where S denotes the set of QKD terminals in one timeline, $A^+(v)$ and $A^-(v)$ denote the incoming and out-going quantum channels of a QKD terminal v respectively. A linear relation between N_e and T_p exists under the assumption that every E_q event consumes the same time. The expected event-processing time is now modeled as

$$E(T_p) = a0 + a1 * k + a2 * N_e \quad (6)$$

where k denotes the number of timelines. The $a1 * k$ models the behavior that the overhead linearly increases as the number of thread grows. The evaluation results and further improvement are discussed in Section 6.

We apply linear regression to determine the coefficients. We collected 410,940 event processing times from various simulated QKD networks. The number of events was in the range of [15, 36375] and the number of threads was in the range of [2, 32]. 80% of the data were selected as the training set and the remaining data are used to evaluate the model. 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. Further, the R-square of the training set is 0.9262 and the R-square of predicting set is 0.9205. The high R-square values imply that our model accurately estimates the simulation execution time.

Table 1: Coefficients and p-values of the event-processing model

	a0	a1	a2
coefficient	288700	8977	25190
p-value	<0.01	<0.01	<0.01

Within event-exchange period, timelines exchange events and negotiate next synchronization time. The time complexity of exchanging event is O(N log(k)). The time complexity of negotiation is O(k). Then, the expected event-exchange time is

$$E(T_s) = a0 + a1 * N_e + a2 * N_e * log_2 k \quad (7)$$

We did not select k as an independent variable in our model because the p-value of k shows the weak correlation between k and $E(T_s)$. This low correlation is caused by the small number of threads. The negotiation time is too low to affect the event-exchange time. Although the total number of events in the queues may not equal to the total number of E_q events, we take an approximation to use the number of E_q to estimate the event-exchange time since it is not feasible to predict the size of a merged event queue because running the simulation.

Table 2: Coefficients and p-values of the event-exchange model

	a0	a1	a2
coefficient	-1472000	-232.4	101.5
p-value	<0.01	<0.01	<0.01

We collected 10,179 event exchange times from various simulated QKD networks. The number of events is in the range of [227, 174808] and the number of threads is in the range of [2,32]. We picked 80% as the training data set and the rest as the testing data set. The coefficients and p-values are shown in Table 2. The p-values show the high significance of variables for the model. Furthermore, the R-square in the training data set is 0.8038 and R-square in the testing data set is 0.7655. The high values of R-square indicate our model is still useful, but compared with $E(T_w)$, this model has worse performance. The reason is the exact numbers of elements in two queues significantly affect the time of merging two queues, and thus, the time of merging two event queues has a higher variance than the event-processing time.

5.2 Graph Partition Using Simulated Annealing

Another task of the network partition scheme 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 [25]. 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 [27] revealed that the simulated annealing is more powerful than the Kernighan-Lin approach [17] to solve the graph partition problem. Therefore, we utilize simulated annealing to find our network partition scheme.

We use simulated annealing to partition the graph as shown in Algorithm 1, where E(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 INIT-STATE() function randomly assigns QKD terminals to timelines. The energy of the state is calculated by Formula 4. The NEIGHBOR() function randomly re-assign a QKD terminal to a different timeline. The acceptance probability is described as follows.

$$P(\text{energy}, \text{energy}_N, T) = e^{(\text{energy}_N - \text{energy})/T} \tag{8}$$

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

Algorithm 1 Graph Partition Using Simulated Annealing

```
state ← INIT-STATE(), energy ← E(state)
state_B ← state, energy_B ← energy
T ← n
while T > 0 do
    state_N ← NEIGHBOR(state), energy_N ← E(energy_N)
    if energy_N < energy_B then
        state_B ← state_N, energy_B ← energy_N
    end if
    if P(energy, energy_N, T) > RAND() then
        state ← state_N, energy ← energy_N
    end if
    T ← T − 1
end while
return state_B
```

6 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 [16]. 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 $n \in \{48...80\}$ 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 frequency of each light source was randomly selected from 10^6 to 10^8 Hz. The distance of each quantum channel was randomly selected from 5 km to 15 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 $t_{sim} = 20$ ms using 2 to 32 threads. The maximum number of effective threads is limited by the number of CPU cores of the server, i.e., 40 cores of 2.6 GHz Dual Intel Xeon and 64 GB RAM.

6.1 Simulation Speedup Evaluation with Different Network Partitions

We repeated each simulation experiment ten times using a random partition scheme and the optimized partition scheme discussed in Section 5. The optimized partition was produced by the simulated annealing method with the temperature $T = 10^5$. 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 6.

We observe that the optimized partition scheme always outperforms the random partition scheme for all network scenarios. Also, ρ keeps increasing as the size of the network grows. For example, for the 32-thread simulation experiments, ρ is about 10% larger for a network with 80 QKD terminals than a network with 48 QKD terminals. Since we generated QKD networks from 45 random directed graphs and thus 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 0.08 to 0.56 when the number of threads is increased from 2 to 16. We notice that the 32-thread experiments have less improvement than the 16-thread experiments due to synchronization overhead, i.e., 32-thread experiments generate more cross-timeline events than 16-thread experiments.

We now fix the network scenario to $G(64, 2, 2)$ 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 7. First, the execution time is improved from 122 seconds (sequential simulation with one thread) to 13 seconds (optimized network partition with 32 threads), in other words, 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 6. However, with ten repeated runs, the standard deviation of the execution time is as little as up to 0.25 seconds.

6.2 Scalability Evaluation with Different Network Partitions

To evaluate the scalability of our simulator, we fixed the network scenario as $G(64, 2, 2)$, repeated the experiments only by increasing the number of thread, and then measured the experiment execution time with and without the optimized network partition. We also ran the phold experiments, a well-known parallel simulation benchmark, for comparison. In the phold experiment, we initialized 10^7 jobs that are evenly distributed over the threads (i.e., perfect simulation workload balance) and the experiments all finished after the 30^{th} 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 8.

We observe that as the number of thread grows over two, 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 0.04. 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.

6.3 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 random partition and optimized partition schemes using simulated annealing. These estimated execution times are then compared with the real execution time. The mean absolute errors (MAE) of the model are plotted in Figure 9.

The MAE of our model is lower than 2.5 seconds except for the 2-thread case. The results indicate that it is reliable to use outputs of our objective function to assess and compare the relative quality between different solutions. The large MAE of the 2-thread case is caused by its long execution time. In fact, the MAE of the 2-thread case is around five times more than the 4-thread case, the mean absolute percentage error (MAPE) of the 2-thread case (15%) is only three times more than the 4-thread case (5%).

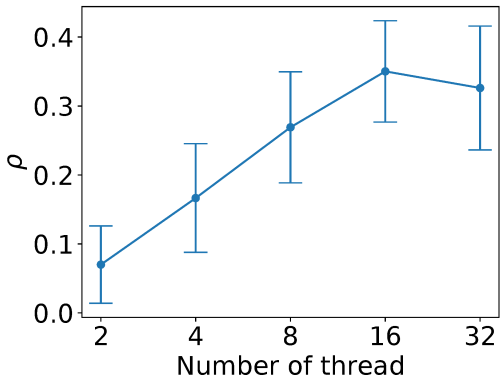


Figure 6: Improvement of execution time ρ for the optimized network partition scheme T_{opt} over the randomized network partition scheme T_{rand}, and $\rho = \frac{T_{rand} - T_{opt}}{T_{rand}}$.

Table 3: Runtime overhead of garbage collection in Golang

# of threads	2	4	8	16	32
overhead	6.91%	8.21%	10.34%	11.73%	12.34%

The prediction error is caused by two reasons: (1) overheads of multi-thread programming, and (2) limitations of the linear regression model. We first collect CPU profiles of the running program with *pprof*, as part of Golang's measurement toolkit. The runtime overheads are shown in Table 3. The overhead, mainly resulting from the garbage collection mechanism of Golang, increases as the number of threads grows, and thus affects the prediction accuracy. Second, although the evaluation results show that our linear regression model works effectively for predicting QKD network simulation workload based on Observation 1, 2 and 3, the linear regression 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. In a real-world computer system, data is rarely linearly separable (e.g., considering E_c and E_s events in a QKD network) and thus reducing the prediction accuracy due to the model deficiency. Besides the computational time, the data movement in the memory hierarchy also contributes a non-negligible part of the overall time [11]. We plan to investigate learning-based models to further improve accuracy as future works.

7 CONCLUSIONS AND FUTURE WORK

The increasing size of QKD networks calls for scalable simulation tools. In this paper, we study 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 develop a QKD network parallel simulator using conservative synchronization. A key component is an optimized network partition

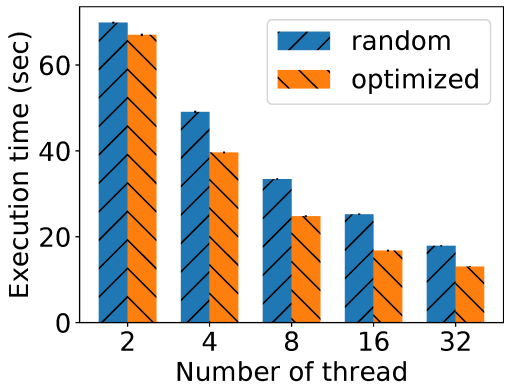


Figure 7: Simulation execution time for $G(64, 2, 2)$, (1) random network partition and (2) optimized network partition

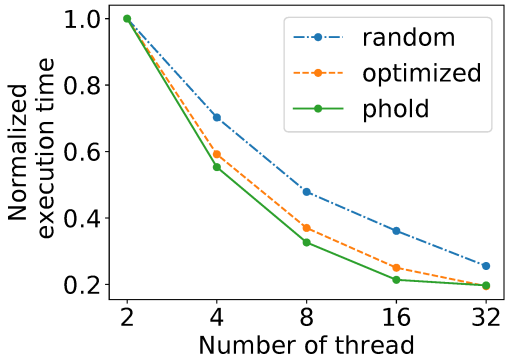


Figure 8: Normalized execution time with (1) random network partition, (2) optimized network partition, and (3) phold (benchmark)

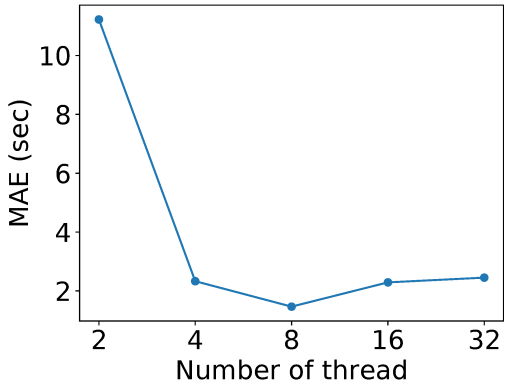


Figure 9: Mean absolute error (MAE) of execution time estimation

scheme based on a linear regression model and simulated annealing for achieving a well-balanced simulation workload for parallel simulation. We also conducted extensive evaluation experiments on simulation scalability, speedup, and predication error analysis.

In the future, we plan to apply other methods to explore solutions for the network partition problem, such as replacing the linear regression model with learning-based models and replacing simulated annealing with tabu search. We also plan to investigate quantum entanglement and develop the corresponding scalable simulation tools. Lastly, we will study hardware-in-the-loop simulation [31] to further improve the simulation testbed accuracy.

ACKNOWLEDGMENTS

This work is 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 Grant 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.

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. *Theor. Comput. Sci.* 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 *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 410–423.

[9] 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

[10] 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).

[11] Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. 2019. Scalable performance prediction of codes with memory hierarchy and pipelines. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 13–24.

[12] Rachel Courtland. 2016. China's 2,000-km quantum link is almost complete [News]. , 11–12 pages.

[13] Axel Dahlberg and Stephanie Wehner. 2018. SimulaQron—a simulator for developing quantum internet software. *Quantum Science and Technology* 4, 1 (2018), 015001.

[14] JF Dynes, Adrian Wonfor, WW-S Tam, AW Sharpe, R Takahashi, M Lucamarini, A Plews, ZL Yuan, AR Dixon, J Cho, et al. 2019. Cambridge quantum network. *npj Quantum Information* 5, 1 (2019), 1–8.

[15] 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.

[16] Google. 2020. The Go programming language. https://golang.org/

[17] Brian W Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal* 49, 2 (1970), 291–307.

[18] Taehyun Kim, Ingo Stork genannt Wersborg, Franco NC 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.

[19] Ulana Legedza and William E Weihl. 1996. Reducing synchronization overhead in parallel simulation. In *Proceedings of the tenth workshop on Parallel and distributed simulation*. 86–95.

[20] Louise Lerner. 2018. Quantum network to test unhackable communications. https://www.anl.gov/article/quantum-network-to-test-unhackable-communications.

[21] 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.

[22] 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.

[23] David M Nicol. 1996. Principles of conservative parallel simulation. In *Proceedings of the 28th conference on Winter simulation*. 128–135.

[24] Momtchil Peev, Christoph Pacher, Romain Alléaume, Claudio Barreiro, Jan Bouda, W Boxleitner, Thierry Debuisschert, Eleni Diamanti, M Dianati, JF Dynes, et al. 2009. The SECOQC quantum key distribution network in Vienna. *New Journal of Physics* 11, 7 (2009), 075001.

[25] Theresa Rose. 2018. Heuristic research. In *What is Psychotherapeutic Research?* Routledge, 133–143.

[26] 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.

[27] L Tao, YC Zhao, Krishnaiyan Thulasiraman, and MNS Swamy. 1992. Simulated annealing and tabu search algorithms for multiway graph partition. *Journal of Circuits, Systems, and Computers* 2, 02 (1992), 159–185.

[28] Nino Walenta and Lee Oesterling. 2019. Quantum Networks: Photons Hold Key to Data Security. Photonics Media, https://www.photonics.com/Articles/Quantum_Networks_Photons_Hold_Key_to_Data/a60541.

[29] Xiaoliang Wu, Joaquin Chung, Alexander Kolar, Eugene Wang, Tian Zhong, Rajkumar Kettimuthu, and Martin Suchara. [n.d.]. Photon-Level Simulation of Quantum Key Distribution with Picosecond Accuracy. ([n. d.]).

[30] 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 *2019 IEEE/ACM Workshop on Photonics-Optics Technology Oriented Networking, Information and Computing Systems (PHOTONICS)*. IEEE, 28–35.

[31] 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 *2017 Winter Simulation Conference (WSC)*. IEEE, 408–418.

[32] Yong Yu, Fei Ma, Xi-Yu Luo, Bo Jing, Peng-Fei Sun, Ren-Zhou Fang, Chao-Wei Yang, Hui Liu, Ming-Yang Zheng, Xiu-Ping Xie, et al. 2020. Entanglement of two quantum memories via fibres over dozens of kilometres. *Nature* 578, 7794 (2020), 240–245.