Toward Applying Quantum Computing to Network Verification

Kahlil Dozier
Columbia University
New York, NY, USA

Justin Beltran
Columbia University
New York, NY, USA

Kylie Berg
Columbia University
New York, NY, USA

Hugo Matousek
Columbia University
New York, NY, USA

Loqman Salamatian
Columbia University
New York, NY, USA

Ethan Katz-Bassett
Columbia University
New York, NY, USA

Dan Rubenstein
Columbia University
New York, NY, USA

ABSTRACT

Network verification, broadly defined as proving the correctness of certain properties resulting from a network's configuration, cannot be efficiently solved on classical hardware via brute force. Prior work has developed a variety of methods that scale by observing a structure in the search space and then evaluating classes induced by that structure. However, even these classification mechanisms have their limitations. In this paper, we consider a radically different approach: applying quantum computing to more efficiently solve network verification problems. We provide an overview of how to map variants of verification problems into unstructured search problems that can be solved via quantum computing with quadratic speedup, making the approach feasible in theory to problems that twice as big in the size of the input. Emerging quantum systems cannot yet tackle problems of practical interest, but rapid advances in hardware and algorithm development make now a great time to start thinking about their application. With this in mind, we explore the limits of scale of the problem for which quantum computing can solve network verification problems as unstructured search.

CCS CONCEPTS

• **Hardware** → *Quantum computation*; • **Networks** → **Protocol testing and verification**; **Protocol testing and verification**.

KEYWORDS

Network Verification, Quantum Computing

ACM Reference Format:
Kahlil Dozier, Justin Beltran, Kylie Berg, Hugo Matousek, Loqman Salamatian, Ethan Katz-Bassett, and Dan Rubenstein. 2024. Toward Applying Quantum Computing to Network Verification. In *The 23rd ACM Workshop on Hot Topics in Networks (HOTNETS '24), November 18–19, 2024, Irvine, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3696348.3696891

1 INTRODUCTION

Networks are designed with the intention of meeting certain goals and requirements. Network verification refers to a class of techniques used to check that various properties of concern to network administrators are satisfied by a network's configuration. For instance, common concerns might be to ensure that routes between specific sources and destinations explicitly do (or do not) pass through specific intermediate routers, or that the number of hops is bounded below some threshold (precluding infinite loops). There is a significant body of prior art that explores verification in the context of properties of different networks using different techniques to efficiently perform the verification process [5, 14, 21]. Despite their differences, the focus of each verification technique can generally be split into two classes:

- Data plane: whether the forwarding table rules and other data plane elements (e.g., ACLs) achieve desired properties for how data flows through the network.
- Control plane: whether the configuration of routing protocols results in data planes that satisfy desired properties.

Substantive progress has been toward the design of approaches that address many concerns of network administrators, despite the problems' inherent complexity [15]. However, there remains a large set of properties for which existing approaches do not offer efficient solutions. For instance, for data plane verification, it remains elusive to determine whether routing paths through the network are guaranteed to be bounded by a fixed number of hops or fixed delay. For control plane, evaluating whether desired properties hold across flows in the face of link failures remains an open challenge.

Network verification differs from a traditional single-machine program verification in three important ways:

- The network "program" is implemented across a distributed set of components (e.g., routers). While what each component does is relatively simple when compared to a standard (centralized) program, the complexity in analyzing properties of a network system greatly depends on how these relatively simple parts interact as a whole.
- The network "program" is generally an operation that needs to be completed quickly (e.g., transit from a source to a point), and loops or significant recursion is highly undesirable. In contrast, traditional programs often rely heavily on long loops and recursive calls to effectively implement operations of significant complexity.
- When a property is violated, the verification system should (hopefully) return an *input* that caused the violation. Inputs in traditional programs are inputs to the program and are often themselves quite large, for example files, large arrays, or values input within iterations of a program loop. In contrast, network verification inputs for a given network tend to be small, on the order of tens of bits: e.g., a packet headers or a set of failed links that cause a property to be violated.

Because in network verification the internal state of the system may be highly complex but an input of the problem is small, quantum computing may offer a more efficient means at solving a variety of problems that remain elusive classically. While quantum computing technology is not yet at a point where it can be deployed to address reasonably-sized networks, the trajectory of advances in the field is moving so quickly that now is a good time to further our understanding of how to apply quantum technology so that we can rapidly build solutions when its applicability does reach fruition.

Classical "structured" v. Quantum "unstructured": For classical solutions to be efficient, they generally require being able to make assumptions about a given network, effectively applying "structure" to inputs. Such a structure allows inputs to be grouped into (evolving) classes, where the classical approach analyzes the inputs on networks by each class, one at a time. Classical techniques become intractable when the inputs lack this structure necessary to separate them into a small number of classes. In contrast, quantum computing is known (in theory) to offer quadratic improvement with respect to classical computing when dealing with "unstructured" data: this allows a doubling of the size (number of bits) of the input space, where doubling can be beneficial in practice (e.g., handling headers twice the size).

In this paper, we show how to map variants of previously considered network verification problems into a quantum computing framework. The variants we consider are difficult to solve efficiently on classical infrastructure because of their inherent complexity [3] but can be solved with quadratic speedup in a quantum computing context as unstructured search problems by applying Grover's Algorithm [11]. These results are merely proof-of-concept, as hardware is not ready to implement these algorithms. Also, quantum computing may in fact offer significantly more computational power than what we demonstrate here *if* one can find some structure in the data whereby a solution beyond Grover's Algorithm may be applied with significantly greater speedup. We demonstrate one minor enhancement technique for our control plane example known as *amplified Grover* that can bias solutions toward those with a desired number of lossy links [6].

In the remainder of this paper, we present a formulation of the general network verification problem within the framework of an unstructured search scenario and illustrate how to use Grover's algorithm to address some of the challenges. Because it is hard to exactly assess practical costs (error correction and the stochastic nature of quantum measurement), it is difficult to precisely assess how long our proposed unstructured searches will take. For now, we limit our scalability analysis to the number of qubits (the analog of classical bits) needed to perform verification as unstructured search on emerging quantum devices.

2 NETWORK VERIFICATION FORMALISM

This section introduces basic nomenclature we use to define a verification problem, which is effectively a 4-tuple consisting of 1) the network components, 2) the protocols as run on those network components, 3) inputs of interest, and 4) properties to verify. A network verification system incorporates these four components by emulating the protocols on a software manifestation of the network and determining, across the set of inputs of interest, which inputs satisfy the properties to verify and which do not.

- *Components* are the network hardware, i.e., routers, switches, links, that must be emulated in the network system.
- *Protocols* are the software and their configurations run on the components.
- *Inputs* of interest describe the scenarios over which verification is performed, e.g., in the data plane, the set of possible packet headers, and in the control plane, the set of possible link failure combinations.
- The *property* is the specific condition that the verifier is to identify inputs that either succeed or fail to satisfy the property. In the data plane, the property might be to find the sets of headers that never reach an intended destination

(infinite loop) or whose path is beyond a reasonable length. In the control plane, it may be a path either proceeding or failing to proceed through some intermediate router.

In classical approaches, there are simply too many inputs (e.g., possible packet headers, combinations of network links that fail) to test for. Hence, prior art has found a variety of ways to find "structure" within the problem such that, rather than analyze individual inputs, inputs can be classified into what is hopefully a much smaller number of classes. Classes can be dynamically modified during the evaluation process, but, as long as the number of classes remains small, the problem can be solved in a practically reasonable time. Much of the innovation in network verification involves translating from the underlying protocol that acts on an individual input to a form that functions on the (structured) classes of inputs. For instance, in the data plane, a protocol run at a router may determine a packet's next hop router based on a packet header. HSA's network verification system uses a data structure to represent the set of possible packet headers and represents a router's forwarding rules by indicating, for each next hop router, the subset of these headers that would transition there [14]. The subsets of headers that are equivalent with respect to the checked property can be grouped together as a union, intersection, or difference of regular expressions of their bit patterns. As these classes move between routers and headers are modified, these modifications are captured by recomputing the corresponding resulting combinations of regular expressions that capture the change. In the control plane, a router's choice of a next hop will be dependent on the set of network links that have failed. In NetDice's network verification system, for a given source-destination pair, a link can a priori be identified as "cold" and omitted from the evaluation if that link will only be on the path from that source to that destination given extremely unlikely failure scenarios of links that are otherwise preferred [21].

3 QUANTUM UNSTRUCTURED SEARCH BACKGROUND

In this section, we present a very high-level overview of how *unstructured search* is implemented in a quantum computing context using Grover's algorithm, which has been shown to have quadratic speedup over classical analogs [12]. Our explanation provides a minimal description of Grover's algorithm sufficient to support our discussion in §4.

Grover's algorithm is most easily described in the context of a problem involving a *black box function* f whose internals are presumed unknown. The function f itself takes an n-bit input and, based on the input, deterministically outputs either a 0 or 1, where an output of 1 is uncommon. f itself is assumed to have no structure, i.e., knowing the outcome of a subset of the inputs provides no information about which as-of-yet unevaluated inputs might yield a 1. The goal is to find an n-bit input whose output evaluates to 1 (if such an output exists).

In a practical use, the function f in Grover is not an unknown oracle but is a reasonably computable (i.e., polynomial-time) *verifier* whose use is similar to that of a verifier in a nondeterministic Turing machine. It is assumed that building the verifier function is easy. However, to find a solution, the verifier must be run on a large (often exponential) number of inputs. Classical approaches require $O(2^n/k)$ time when k solutions exist; a (conceptual) nondeterministic Turing machine requires polynomial time since it checks all inputs in parallel; and a quantum computer applying Grover's algorithm will require time $O(\sqrt{2^n/k})$, a quadratic speedup over classical approaches. To illustrate the advantage of this quadratic speedup, consider a problem with an $n = 32$-bit input where a single solution exists ($k = 1$), and assume that classically computing $f()$ on an input takes 10 ms. Checking half the inputs (the expected number to find the solution) would require $2^{31}/100$ seconds, which is just under 250 days. In contrast, even if the corresponding overhead of computing $f()$ in the context of Grover's algorithm (high-level details below) took 1 second each (a relative slowdown of 100 per input), a solution could be found in slightly over 18 hours. Were $n = 48$, the respective times would be just under 450,000 years for classical vs. 194 days for quantum.

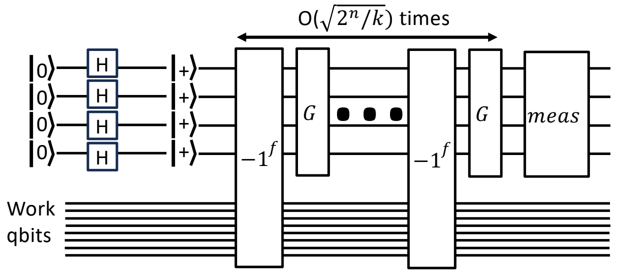


Figure 1: Grover overview

The general process of Grover's algorithm is depicted in Figure 1. Grover's algorithm starts much like a non-deterministic Turing machine, generating (in superposition) the set of all possible inputs (the time to do this is linear in the size of the input): the n qubits, initially set in a ground state (i.e., to 0 or more formally, $|0\rangle$), are each operated upon by a Hadamard gate that shifts each qubit to a state known as $|+\rangle$ which, if measured, returns 0 or 1 with equal probability of 0.5. This superimposed state is simultaneously run through the verifier circuit, f, which, in superposition, computes f on all 2^n inputs. The output of f is applied (in superposition) as the exponent to -1 (to produce 1 when $f = 0$ and -1 when $f = 1$), thereby generating (in superposition) the result of each input (success or failure of the desired result).

A final step is to choose one of the k inputs that was successful. Unfortunately this part is not easy, and extracting an appropriate solution is what adds the $O(\sqrt{2^n/k})$ additional complexity. This extraction process involves repeated application of a sequence of gate operations G, often termed a Grover iterate, that combines f with a *Diffuser* circuit, as shown in Figure 1, after which, the n qubits that represented the input are measured. With probability close to 1, these bits will collapse to an n-bit input that matches an input for which f evaluates to 1. In effect, this $O(\sqrt{2^n/k})$ extraction process finds a desired input [18]. In this sense, it differs from a nondeterministic Turing machine in two ways: first, it has an additional $O(\sqrt{2^n/k})$ overhead to find a solution, and it only returns a single solution, effectively selected at random, as opposed to conceivably returning all possible solutions by the end of the (nondeterministic) run.

4 MAPPING NETWORK VERIFICATION TO QUANTUM UNSTRUCTURED SEARCH

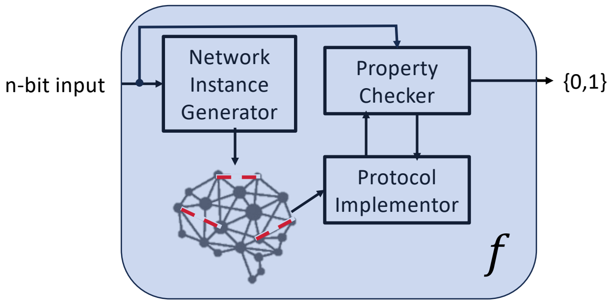


Figure 2: Components of the oracle function f

In this section, we describe how to map Grover's algorithm to a generalized instance of a network verification problem. In short, we must define an oracle function f as a *verifier* that takes an n-bit input to specify a specific input of the search space and emulates network operation on that input to validate whether the desired (or undesired) property holds for that input. The basic functionality that f must perform is depicted in Figure 2. Given an n-bit input, f

- generates an emulated network whose configuration can be influenced by the n-bit input, which we call a *network instance*.
- emulates the underlying routing/forwarding protocol upon the network input via a *Protocol Implementor*
- in parallel to the execution of the protocol, a *Property Checker* checks whether the property holds or is violated for the current network input.

By encoding f as a circuit that can be executed in a quantum computer, Grover's algorithm can be applied to simultaneously evaluate all possible 2^n inputs.

In the following two subsections, we separately describe application of the general mapping approach to verification problems on the data plane and control plane, each followed by a very preliminary, proof-of-concept mapping of the problem to existing NISQ quantum hardware.

4.1 Data Plane Example

Consider a data plane setting similar to that proposed in the HSA paper [14], viewing the underlying network as a graph. At each router, the next hop is determined by the n-bit packet header, which may also be modified by the router in a deterministic manner. We wish to determine whether, from a specific source node S, any packets will traverse more than 100 hops. This property can be hard to detect since short-lived cycles are permissible if packets break out of the loops due to packet header changes. We design f as follows:

- The *Network Instance* contains a hard-coded version of G, as well as a hard-coded set of forwarding rules and packet header modification routines for each node in the network. The n bit input indicates the packet header and is the only variable component to the network instance.
- This graph is passed into the *Protocol Implementer* that emulates the hard-coded forwarding rules upon the provided n-bit packet header for 100 hops.
- The *Property Checker* monitors the path taken by the packet as it traverses through the network, counting the number of hops as it proceeds along for at least 100 hops. If the packet reaches its destination prior to the 100th hop, a 0 is immediately returned. Otherwise, upon reaching the 100th hop, a 1 is returned.

4.2 Control Plane Example

Consider a control plane setting similar to that proposed in NetDice [21], in which a given underlying network can be viewed as a graph $G = \{V, E\}$ with $n = |E|$, the number of links in the network. A routing protocol emulating BGP/IGP determines routes through this graph. Selecting specific nodes $C, D, E \in V$ on this network, we wish to evaluate whether there exist combinations of up to 10 link failures for which C's path to D fails to proceed through node E. We design f as follows:

- The *Network Instance Generator* contains a hard-coded version of G and uses the n-bit input to encode which of the n links are up/failed, removing these links from the instance of the graph.
- This revised graph is passed to the *Protocol Implementer* that emulates a hard-coded implementation of BGP/IGP on the underlying network.

- The *Property Checker* then explores the route from C to D, returning 0 as soon as vertex E is reached (i.e., before D). Also, if the n-bit input itself initially has more than 10 bits encoded to 1 (i.e., more than 10 link failures), 0 is also returned. Otherwise, if D is reached without going through E, then 1 is returned.

In addition to implementing a "cutoff" that bounds the maximal number of failing links, a variant of Grover known as *Amplified Grover* can be used to increase the efficiency of finding solutions with a specific number of link failures. While the traditional implementation of Grover applies the Hadamard gate on each qubit to put its measurement outcome split in half between 0 and 1, an enhanced *Amplified Grover* can instead initialize the state of each qubit into any arbitrary probability p of evaluating to 0 (link failure). Doing so changes the relative likelihoods of Grover selecting a given outcome, maximizing those centered around the distribution where a fraction p of the links fail. Hence, in a large network where we are interested in only considering the cases where small numbers of links are expected to fail, we can utilize values of p that are far smaller than 0.5.

5 (VERY) PRELIMINARY IMPLEMENTATION

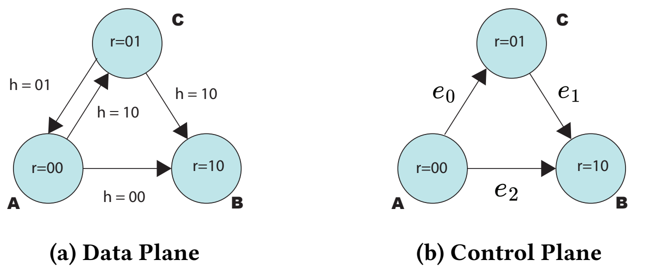


Figure 3: Example Networks for Experiments

For purposes of proof-of-concept and to evaluate initial efficacy of using quantum computation, we consider a very simple network that can be easily analyzed on today's QC platforms. Our examples illustrate how to perform network verification on top of QC, but the hardware available to us is nowhere near large enough to observe quantum advantage. In fact, our networks are so simplistic that validation can simply be done by hand. Figure 3 depict simple 3-node networks (with nodes A,B,C) with 2-bit router IDs r assigned. Figure 3a's data plane example depicts a network where routers forward packets with 2-bit headers in the directions indicated by arrows, while, in Figure 3b, we annotate 3 edges e_0, e_1, and e_2 that may fail, and evaluate whether flows from router node A can reach node B.

5.1 Data Plane Proof-of-concept

We demonstrate an example of mapping network verification to quantum search for the data plane setting. Figure 3a shows a toy network with 3 routers, A, B and C. We assume packets have two-bit headers. The arrows represent the network routing behavior based on header values labeling the edges, which can be described in the same way as the transfer functions of Header Space Analysis [14]. Headers with no corresponding arrows remain at the current router without being forwarded.

We consider checking the property that packets must route from A to B within 2 hops (i.e., avoiding loops). To construct the quantum circuit, the Network Instance Generator and Protocol Implementer, collectively referred to as the oracle f, are simultaneously integrated into one quantum circuit block (please see [8] for more details on the quantum circuit). We associate two bits to each router to represent a packet's current location. The input that f takes is four bits long: two for the packet header, which will be put into superposition for Grover's Algorithm, and two to represent a packet's current router location, which is reset in between iterations.

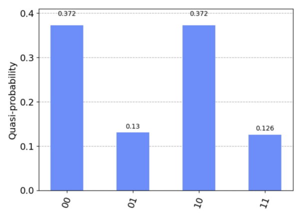
Our quantum circuit employs standard combinatorial logic to implement the network and its routing behavior. First, the header bits of the input are fed through a bit checker. Based on the value of the header bits and the network's forwarding logic, the bits encoding the packet's location are updated to reflect where it is forwarded.

The circuit consisting of bit-checking and altering router location represents one "hop" of the routing process. To simulate a second hop, this circuit is applied once more, with all inputs except the current router location reset to their original value.

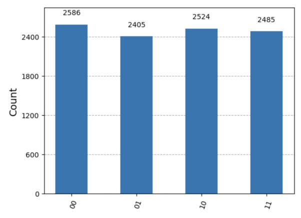
Property Checker: After applying the above circuit to simulate two "hops" of routing logic, the Property Checker is another logical circuit that checks the value of the current router location bits to see if they are set to 10 (representing router B).

The combined Network Instance Generator and Protocol Implementer represent the "oracle" of Grover's algorithm. For the full algorithm, the header bits of the input are initially put into equal superposition and fed into the Grover oracle. For this problem, a total of one Grover iterate (oracle + diffusion) are sufficient to extract the solution headers ("00" and "10") with high probability.

We implemented this circuit in IBM Qiskit and ran it on both IBM's AerSimulator and on IBM's Osaka, an actual 127-qubit Quantum Computer, for a total of 10,000 shots each. Figure 4 shows the results for both, with the measurement outcomes labeled on the x-axis and relative proportion of that outcome on the y-axis. When simulated (Figure 4a), Grover's algorithm returns a correct header roughly 75% of the time.



(a) simulation



(b) experimentation

Figure 4: 10,000 shots of Grover's Algorithm

Ironically, Grover's algorithm increases in accuracy as the size of the input grows.[1] However, for small inputs, with enough repeated runs of the entire circuit, we can ensure we obtain a correct answer with high probability. On the real quantum machine (Figure 4b), this correct proportion is reduced to 51%, additional demonstration that quantum computation is not yet ready for conventional use.

5.2 Control Plane Proof-of-Concept

Using the topology shown in Figure 3b, we consider all 8 possible combinations of link outages.

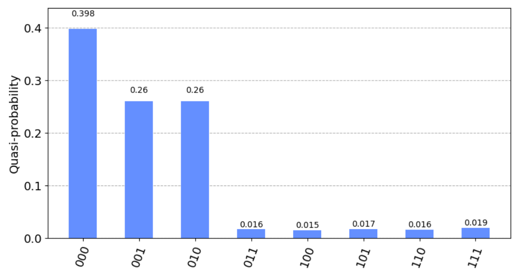


Figure 5: Results of control plane analysis

Our quantum circuit implements f to return 1 when the destination at B is unreachable from the source at A. We also implemented this circuit in IBM Qiskit and ran it on IBM's AerSimulator. Figure 5 shows histogram results of the output of 10,000 shots when applying Grover, where the label $x_2x_1x_0$ indicates the status of respective links e_2, e_1, and e_0 where 1 indicates the link is operational (0 indicates failure). The most frequently returned results represent all failure scenarios.

6 QUANTUM CIRCUIT COMPLEXITY ANALYSIS

The time required to perform verification depends on factors including error correction codes and their corresponding rates, the general speed of the circuits, and how well these circuits can be optimized. It is difficult at this time to presuppose the time computation will take. We can, however, speculate on the number of qubits, including scratch qubits (i.e., temporary variables) that a computation will require as a function of the size of the problem being considered. Our results demonstrate a linear scaling of qubits, which demonstrates the potential viability of the approach as quantum hardware continues to scale the number of available qubits.

6.1 Data Plane Verification

For data plane verification with Header Space Analysis, we consider two ways to quantify the size of the problem: the number of possible header addresses and the size of the network (in terms of the number of rules and routers).

For simplicity, we consider a network that consists only of if/then rules matching a header wildcard expression and forwarding to a particular port number. Define n to be the total number of headers, R the total number of routers, ℓ the number of unique wildcard expressions that appear in the rules of the network, P the number of unique port numbers that appear in the rules, k the max number of hops, and G the optimal number of Grover iterates. Our Quantum Circuit requires $(1+\ell)\lceil\log(n)\rceil + (P+k+G(2k-1))\lceil\log(P)\rceil + 2\max(\ell,P)+P+\ell$ qubits. If we allow the use of mid-circuit reset gates (instead of using extra ancilla qubits), we require only $(1+\ell)\lceil\log(n)\rceil + (1+P)\lceil\log(P)\rceil + 2\max(\ell,P)+P+\ell$.

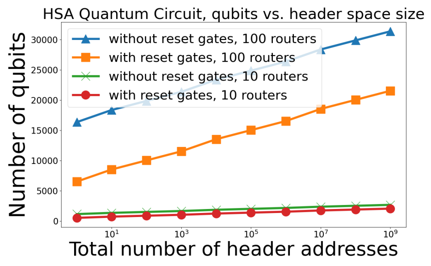
Figure 6a shows how the number of required qubits varies with n (varied logarithmically on the x-axis), for network sizes of 10 and 100 routers. For ease of analysis, we assume every router has the same number of rules r, and $\ell = P = R\cdot r$. We also set $k = R$ and $G = 5$. Unsurprisingly, the number of qubits required scales linearly with the log of the number of headers (i.e., linearly with the number of input bits).

Figure 6b shows how the required number of qubits varies with R, for an assumed 32-bit header space, with the number of rules per router r set to either 5 or 50. Again, we set $\ell = P = R\cdot r, k = R$ and $G = 5$. We see the scaling is linear in R, which is to be expected as we have set R to be directly proportional to both ℓ and P. The number of qubits needed for computation scales linearly in the size of the network.

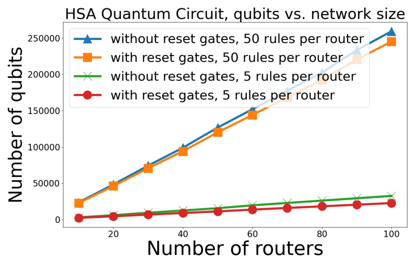
6.2 Control Plane Verification

For control plane verification with NetDice, we quantify the size of the problem in terms of the input space n, the total number of edges in the network. We define R to be the total number of routers, D to be the diameter of the network, and G the optimal number of Grover iterates. In these terms, this quantum circuit requires $\log(r) + e((r-1)*d) + g$ total

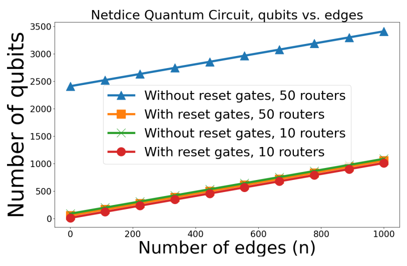
[1]Each iteration can be viewed as a rotation, where the size of the rotation angle is inversely proportional to the ratio of solution inputs to the space of all inputs. Hence, when the space of all inputs is small, one is forced to either significantly over- or under-rotate.



(a) Data plane varying size of input n



(b) Data plane varying number of routers



(c) Control plane varying size of input n

Figure 6: Required number of qubits for implementation

qubits. Allowing mid-circuit reset gates reduces this number to $\log(r) + e + r$ qubits.

Figure 6c shows how the number of required qubits varies with n, for network sizes of 10 and 50 routers. We assume $d = r - 1$ and $g = r$. Again, we see a linear scaling in the size of the input.

7 RELATED WORK

Quantum Computing for Program Verification: To our knowledge, our work is the first to propose the use of quantum computing for network verification. Recent work explores quantum computing for program verification [13]. Due to complexities in verification of general programs (§1), the program cannot be analyzed directly (i.e,. applying Grover directly to a verifier) as in our case but must first be converted to a corresponding SAT instance.

Network Verification: There are a wide range of classical approaches and techniques to verifying network properties have been developed [16]. In general, approaches to verification can be divided into two classes. *Control plane verification* ([1, 5, 10, 17, 21]) analyzes the network at the level of protocol and device configurations, derives the subsequent routing behavior, and then checks the relevant properties. Holistic approaches intend to holistically represent *all* possible data planes induced by a given control plane configuration via symbolic representation but at the expense of computational tractability [4, 5, 19, 20]. On the other hand, instance-based approaches emulate the convergence process from a given control plane to a fixed data plane, but they face the challenge of accurately modeling every network component and risk missing critical corner cases [1, 7, 9]. *Data plane verification* ([2, 14, 17, 22]) assumes the network routes to have already been established and checks relevant properties by analyzing the network forwarding tables. These methods generally aim to group packet headers with identical routing behavior to optimize the search of inputs that might violate the property of interest. However, the number of groups can increase exponentially with the number of network components, packet headers, and middleboxes. To maintain efficient verification, these techniques make assumptions about a given network's underlying structure. When violated, the claims to efficiency may no longer hold.

8 DISCUSSION / FUTURE PLANS

This paper's contribution is intended to introduce to the networking community an application of quantum computing to a computationally challenging problem in networks: that of network verification. Our preliminary work is meant to demonstrate that there will be certain challenges in network verification which remains open and impractical to solve using classical computing. Instead, by phrasing the verification problem as a functional verifier, quantum's ability to speed up unstructured search may itself provide sufficient benefit to make these challenges verifiable in reasonable time.

Our initial foray indicates that there is still a significant amount of work to be done. First, building the verifier f for large networks and sophisticated properties remains a challenging endeavor. For now, it must be designed at the circuit level. Second, our experience with Grover's algorithm thus far shows that its performance in practice is noisy (too often returning inputs where the property was not violated). This overhead must also be taken into account or somehow approved upon. Last, the speedup gained by utilizing unstructured search, while significant, does not take full advantage of the potential power of quantum computing. The community should try to understand how existing structure in the underlying problem (in this case, network verification) can be exploited in approaches that can potentially have exponential speedup when performed upon a quantum system.

ACKNOWLEDGMENTS

We wish to thank our colleague Henry Yuen and the anonymous HotNets reviewers for their insightful feedback on earlier drafts. This material is based upon work supported by the National Science Foundation under Grant No. CNS-2148275. 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 the National Science Foundation.

REFERENCES

[1] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Tiramisu: fast multilayer network verification. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'20)*. USENIX Association, USA, 201–220.

[2] Ehab Al-Shaer and Saeed Al-Haj. 2010. FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*. ACM, New York, NY, 37–44.

[3] Kalev Alpernas, Aurojit Panda, Alexander Rabinovich, Mooly Sagiv, Scott Shenker, Sharon Shoham, and Yaron Velner. 2019. Some Complexity Results for Stateful Network Verification, In Formal Methods in System Design. *Formal methods in system design* 54, 191–231. Issue 2.

[4] Mina Tahmasbi Arashloo, Ryan Beckett, and Rachit Agarwal. 2023. Formal methods for network performance analysis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, USA, 645–661.

[5] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 155–168. https://doi.org/10.1145/3098822.3098834

[6] G. Brassard and P. Hoyer. 1997. An exact quantum polynomial-time algorithm for Simon's problem. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems (ISTCS-97)*. IEEE Comput. Soc, USA, 12. https://doi.org/10.1109/istcs.1997.595153

[7] Matt Brown, Ari Fogel, Daniel Halperin, Victor Heorhiadi, Ratul Mahajan, and Todd Millstein. 2023. Lessons from the evolution of the Batfish configuration analysis tool. In *Proceedings of the ACM SIGCOMM 2023 Conference*. Association for Computing Machinery, New York, NY, United States, 122–135.

[8] Kahlil Dozier, Justin Beltran, Kylie Berg, Hugo Matousek, Loqman Salamatian, Ethan Katz-Bassett, and Dan Rubenstein. 2024. Technical Report: Toward Applying Quantum Computing to Network Verification. *arXiv preprint* arXiv:5946629 (2024).

[9] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient network reachability analysis using a succinct control plane representation. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 217–232.

[10] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 300–313. https://doi.org/10.1145/2934872.2934876

[11] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM press, Philadelphia, PA, 212–219.

[12] Lov K. Grover. 1998. A framework for fast quantum mechanical algorithms. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 53–62. https://doi.org/10.1145/276698.276712

[13] Sebastian Issel, Kilian Tscharke, and Pascal Debus. 2024. *Towards Classical Software Verification using Quantum Computers*. Technical Report. Fraunhofer AISEC.

[14] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, United States, 113–126.

[15] Yahui Li, Xia Yin, Zhiliang Wang, Jiangyuan Yao, Xingang Shi, Jianping Wu, Han Zhang, and Qing Wang. 2019. A Survey on Network Verification and Testing With Formal Methods: Approaches and Challenges. *IEEE Communications Surveys and Tutorials* 21, 1 (2019), 940–969. https://doi.org/10.1109/COMST.2018.2868050

[16] Yahui Li, Xia Yin, Zhiliang Wang, Jiangyuan Yao, Xingang Shi, Jianping Wu, Han Zhang, and Qing Wang. 2019. A Survey on Network Verification and Testing With Formal Methods: Approaches and Challenges. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 940–969. https://doi.org/10.1109/COMST.2018.2868050

[17] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. 2011. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 290–301. https://doi.org/10.1145/2018436.2018470

[18] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, UK.

[19] Santhosh Prabhu, Kuan Yen Chou, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. 2020. Plankton: Scalable network configuration verification through model checking. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, United States, 953–967.

[20] Rob Sherwood, Jinghao Shi, Ying Zhang, Neil Spring, Srikanth Sundaresan, Jasmeet Bagga, Prathyusha Peddi, Vineela Kukkadapu, Rashmi Shrivastava, KR Manikantan, et al. 2024. Netcastle: Network Infrastructure Testing At Scale. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, United States, 993–10008.

[21] Samuel Steffen, Timon Gehr, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2020. Probabilistic verification of network configurations. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. Association for Computing Machinery, New York, NY, United States, 750–764.

[22] Shuyuan Zhang and Sharad Malik. 2013. SAT based verification of network data planes. In *Automated Technology for Verification and Analysis: 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*. Springer, Springer, United States, 496–505.