MONTE CARLO ESTIMATION OF THE DIAMETER-CONSTRAINED RELIABILITY OF NETWORKS USING PARALLELISM[*]

MARTIN LAPINSKI[†], HELEN LIN[†], AND MYLES MCHUGH[‡]

ADVISOR: LOUIS PETINGI
Computer Science Department, College of Staten Island (CUNY), Staten Island, NY, USA

Abstract. In this paper we study an heuristic approach, based on parallelism, to estimate the Diameter-Constrained Reliability (DCR) of a given undirected probabilistic graph $G = (V, E)$, with two terminal nodes s and t, a given diameter bound D, and where edges are assigned independent probabilities of failure (node are assumed to be perfect). Since exact methods to evaluate the DCR are computationally expensive (i.e., NP-hard), we propose to implement a Monte Carlo (MC) method based upon MPI parallel processing to estimate the reliability. We conduct computational tests on several topologies while considering different factors such as number of cluster nodes utilized, number of trials performed by the cluster nodes, different ranges generated by random number functions, and then we determine how these factors affect the estimation of the DCR.

Key words. Network Reliability, Diameter-Constraint, Factoring Theorem, Monte Carlo, Parallelism.

1. Introduction.

The system under study is a communication network represented by a undirected graph $G = (V, E)$, where V and E, are the set of vertices and edges of G, respectively. We are also assuming that edges fail independently with known probabilities (vertices are always reliable). If we are interested in determining the probability that messages are successfully transmitted between a set of terminal nodes $K \subseteq V$, the *classical reliability*, $R_K(G)$, measures the probability that after deletion of the failed edges, there exists a path connecting each pair $x, y \in K$, of terminal vertices.

However, in many real-life situations, the quality of the communication depends on the existence of a path connecting each pair of terminals x and y, whose length (measured as the number of edges) is bounded by a given integer D. The *K-terminal Diameter Constrained Network Reliability* (DCR), denoted as $R_K(G, D)$, introduced by Petingi and Rodriguez in 2001 [9], is the probability that there exists a path of length D or less between each pair of terminal vertices x and y. The DCR can be applied to assess performance objectives, for example, of packet-oriented networks where links may fail and there is a "time-to-live" (TTL) limit, specified in number of hops that can be traversed by any given packet (for instance IPv6 packets include a hop limit field [6]). It is also the case of many overlay networks (such as peer-to-peer file sharing networks) that employ flooding protocols for peer discovery which specify a maximum number of hops to be visited by a request.

The DCR measure subsumes the classical reliability in the following sense; as the maximum path length in a network on n vertices is composed of at most $n - 1$ edges, then $R_K(G, D) = R_K(G)$, whenever $D = n - 1$. As calculation the classical reliability for arbitrary terminal set K is an NP-hard problem [10], then determination of the

[*]This work was done under the frame of the REU (Research Experience for Undergraduates) program "Computational Methods in High Performance Computing with Applications to Computer Science" held at the College of Staten Island (CUNY) in the Summer of 2015; the program was sponsored by the National Science Foundation (Grant 1359266) and the United States Department of Defense.

[†]College of Staten Island (CUNY)

[‡]Kean University

DCR is an NP-hard problem as well. For a fixed number of terminal vertices K, and for fixed diameter bound D, Cancela and Petingi [3] proved that to determine $R_K(G, D)$ is also NP-hard; thus in order to efficiently estimate the reliability, in this paper we propose an heuristic to evaluate the DCR, based on Monte Carlo (MC) using MPI parallel processing [12], which is run on a distributed-memory architecture system. Even though MC has been applied successfully to evaluate the classical reliability [7], as well as the DCR [2], to our knowledge it was never implemented using parallelism.

In this paper we specifically concentrate on the case when $K = \{s, t\}$, also called the two-terminal DCR or the source-to-terminal DCR [8], since many applications today require to measure the reliability of transmitting messages between two participating nodes (e.g., peer-to-peer networks). We evaluate the precision of our methodology by performing several tests on graphs whose exact DCR reliability measures are known [8]. We've also run some tests on graphs in order to measure how different factors such as the number of cluster nodes utilized, number of trials executed by these nodes, and different ranges generated by a random-number function, affects the accuracy of the reliability estimation.

This paper is structured as follows. In Section 2 we will give a background for an exact method (Factoring Theorem) of evaluation and MC techniques to estimate the DCR [4, 8]. In Section 3 we present the pseudo-code of our implementation using MPI parallel processing. Then in Section 4, we will evaluate the results of our tests and in Section 5 we present our conclusions.

2. Background. We use the notation r_e to represent the probability that an edge e is operational, while $q_e = 1 - r_e$ is the edge probability of failure.

2.1. Exact Method of Evaluation. The exact DCR may be calculated using the Factoring Theorem, which is also known as Moskowitz's Decomposition Theorem [8]. We define an edge e undetermined if $0 < r_e < 1$. Factoring Theorem generates a binary tree where each node j represents a subgraph of G, G_j (originally $G_0 = G$), and whose children correspond to the two subgraphs of G_j that will have the selected edge e to be operational ($r_e = 1$) or failed ($r_e = 0$). When considering the source-to-terminal DCR of graph G, the reliability of each subgraph G_j is calculated as [8]:

$$R_{\{s,t\}}(G_j, D) = \begin{cases} 0 : \text{if there exist no path between } s \text{ and } t\text{, whose length is less} \\ \quad \text{than or equal to } D. \\ \\ 1 : \text{if } G_j \text{ contains an operational path between } s \text{ and } t\text{, whose} \\ \quad \text{length is less than or equal to } D. \\ \\ r_e R_{\{s,t\}}(G_j * e, D) + (1 - r_e) R_{\{s,t\}}(G_j - e, D) : \\ \quad \text{if there is an undetermined edge } e \text{ that would} \\ \quad \text{allow for the existence of at least one path between } s \text{ and } t\text{,} \\ \quad \text{whose length is less than or equal to } D. \\ \qquad \triangleright\ G_j * e \text{ is the subgraph of } G \text{ obtained from fixing} \\ \qquad\ \ r_e = 1 \text{ (operational).} \\ \qquad \triangleright\ G_j - e \text{ is the subgraph of } G \text{ obtained from fixing} \\ \qquad\ \ r_e = 0 \text{ (failed).} \end{cases}$$

The leaves of the generated binary tree correspond to the subgraphs of G whose reliabilities are either 0 or 1 as stated in the previous bracketed equation. This evaluation will yield a binary tree with at most $2^{|E|}$ nodes, where E is the edge-set, thus this method is of exponential complexity in the worst-case. In the next section we propose the implementation of a MC technique using parallelism.

2.2. Monte Carlo techniques to evaluate the DCR. Monte Carlo techniques are methods that perform repeated random sampling in order to obtain an approximate solution to a problem, and, they have been successfully applied to estimate both the classical reliability as well as the DCR. To estimate the DCR, two MC methods have been proposed [2]. One is the Crude Monte Carlo (*CMC*) method and the other is the Recursive Variance Reduction (*RVR*) method. The CMC is the method we applied for our tests, and simply consist in generating n independent copies of G (i.e. $G^1, \dots, G^n$), which are subgraphs of the original graph G where all the edges are randomly fixed as operational or failed, based on the edge reliability. Therefore, the estimated source-to-terminal DCR would be: $\sum_{j=1}^{n} R_{\{s,t\}}(G^j, D)/n$. Due to the law of large numbers, this approximation converges to the exact DCR, $R_{\{s,t\}}(G, D)$ as $n \to \infty$. The second MC method is the RVR which associates every output random variable with a variance, and limits the results in order to obtain a greater precision. Therefore RVR will give a more accurate DCR estimation than the CMC method when using the same number of random samples [4]. However as we will show in the next sections, the CMC approach will yield excellent results when parallelism is applied.

3. Proposed methodology to evaluate the DCR: Monte Carlo using MPI parallel processing. In this work we discuss the Crude Monte Carlo method and implement it on a distributed-memory architecture system utilizing MPI (Message Passing Interface) [12].

The original implementation code in C++ using MPI libraries can be found in [5].

The following is our MC method using MPI parallel processing:

1. Input: Probabilistic graph $G = (V, E)$, two terminal nodes s and t, reliability $r(e)$ for each edge $e \in E$, and a diameter bound D.
2. Each processor generates an assigned *number-of-trials*, each trial corresponding to a subgraph G^j of G (see previous section).
   - 2.1 For each trial, determine if each edge of G either succeeded (include in the subgraph G^j) or failed (excluded from G^j).
     - 2.1.1 Determine the edge succeeded or failed by generating a random number $0 < rand(e) < 1$. A random number $rand_{int}$ is generated within the range [1, Max] and $rand(e) = rand_{int}/Max$. If $rand(e)$ is less than or equal the reliability of the edge (r_e), then the edge succeeded, otherwise the edge failed.
   - 2.2 After all the edges are determined, use Dijkstra's algorithm [8] to determine whether there is a path with a distance less than or equal to the diameter bound D between s and t in G^j. If yes, then increment *success-count.*
3. Once each processor has finished, pass its *success-count* and *number-of-trials* to the master processor p_0.
4. The master processor p_0 will sum-up *success-count* gathered from the slave processors into a variable *Total-Successes* and *number-of-trials* into a variable *Total-Trials* and will display the approximated DCR, which is estimated as *Total-Successes* / *Total-Trials.*

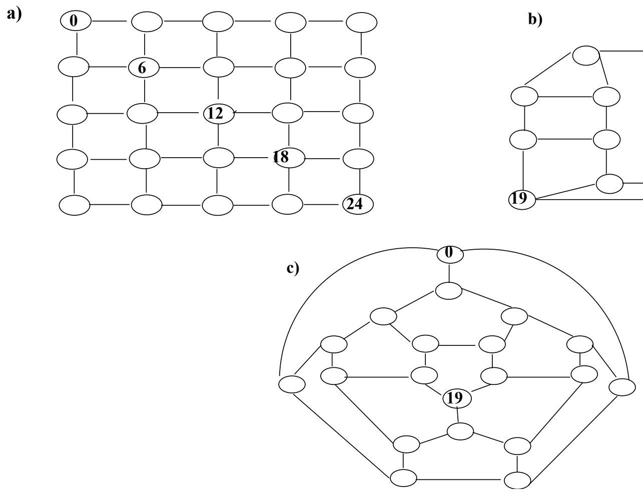

Fig. 4.1. *Types of topologies: a) 5X5-Grid, b) Arpanet, and c) Dodecahedron.*

Table 4.1
Comparison of the Factoring method (exact reliability) and parallelized Monte Carlo method (approximate reliability) on the 5x5-Grid, Arpanet, and Dodecahedron graphs.

G	(s,t)	D	Factoring		Monte Carlo		% Difference in
			$R_{s,t}(G,D)$	CPU-t (s)	$R_{s,t}(G,D)$	CPU-t (s)	Reliability
5X5Grid	(0,6)	8	0.498971	883	0.499057	3.96	.017
5X5Grid	(0,12)	8	0.339515	2706	0.339696	3.97	.053
5X5Grid	(0,18)	8	0.234521	> 24 hours	0.234588	4.12	.029
5X5Grid	(0,20)	8	0.176359	> 24 hours	0.176372	3.99	.008
5X5Grid	(0,24)	8	0.123324	> 24 hours	0.123232	3.97	.074
Dodeca	(0,19)	5	0.168441	95	0.168394	3.20	.028
Dodeca		7	0.268820	170	0.268790	2.99	.011
Dodeca		9	0.285391	201	0.285597	3.18	.072
Arpanet	(0,19)	4	0.162109	19	0.162146	3.00	.023
Arpanet		6	0.237618	103	0.237368	3.00	.105
Arpanet		9	0.295711	373	0.295696	3.08	.005
Arpanet		19	0.302415	475	0.302339	3.00	.025

The longest execution time taken by a processor will yield the execution time (in seconds) of the algorithm.

4. Evaluation of the accuracy of the proposed implementation.

In this section we perform several algorithmic tests on graphs to evaluate the accuracy of the proposed parallel processing implementation of MC when the number of processors, number of trials, and random number ranges are incremented. These tests were performed on the 5x5-Grid, Arpanet, and Dodecahedron topologies (see Fig. 4.1). Algorithms were implemented in C++, and ran on the cluster *Penzias* of the City University of New York High Performance Computing Center (CUNY-HPCC). Penzias is a cluster with a total of 1,152 Intel Sandy Bridge cores. Each Sandy Bridge core is separated into 2 virtual nodes: one with 12 cores and no GPUs and one with 4 cores and 2 GPUs [1]. To perform our tests, we used up to 50 non-GPUs virtual nodes.

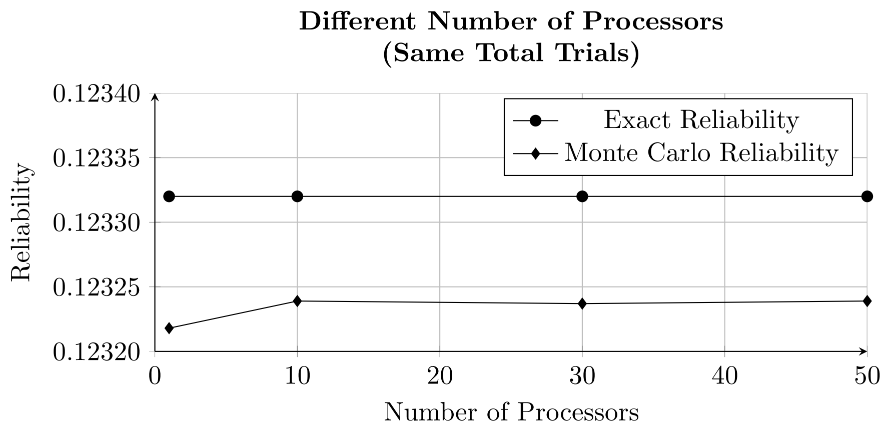

Fig. 4.2. *Graph of the Monte Carlo reliability and of the exact reliability obtained from using different number of processors.*

Table 4.2

Results from the Monte Carlo method using MPI parallelism on a 5X5-Grid using different number of processors with the total number of trials fixed to a billion.

Processors	$R_{s,t}(G,D)$	CPU-t (s)	% Difference in Reliability
1	0.123218	4094.94	.086
10	0.123239	394.38	.069
30	0.123237	131.33	.070
50	0.123239	78.39	.069

4.1. Verify the proposed parallelized Monte Carlo method. To first verify that the proposed parallelized Monte Carlo method will provide an accurate estimation of the exact reliability, we performed our tests on a 5x5-Grid, Arpanet, and Dodecahedron undirected graphs. For all the graphs, each edge $e \in E$ was assigned the probability of failure $q_e = 0.5$. We ran our tests using ten processors and a million trials per processor, given a total of ten million trials per graph. Table 4.1 shows the results obtained from the MC method compared to the exact evaluation method using Factoring method [8]. Column 1 shows the type of graph we are testing, the labels of the terminal nodes (s, t), and the diameter bound D. Columns 2 and 3 show the reliability and CPU-time from using the exact Factoring method [8]. Columns 4 and 5 show the reliability and CPU-time of the MC method. The percentage difference between the exact evaluation and the estimated one is shown in column 6. It's important to note since the Factoring method and MC method were executed on different computers, the execution time between the two methods can't be compared. However, we can compare the execution time relative to themselves. We note that the time grows exponentially using the Factoring method while the execution time for the MC method barely changes.

As expected, the differences between the reliability obtained using MC and the exact reliability are insignificant. Thus, this verifies that our MC implementation gives a good approximation to the actual reliability.

4.2. Investigate the effects of increasing the number processors while the total number of trials remains constant. To investigate whether the number of processors alone will impact the accuracy of the network reliability obtained using

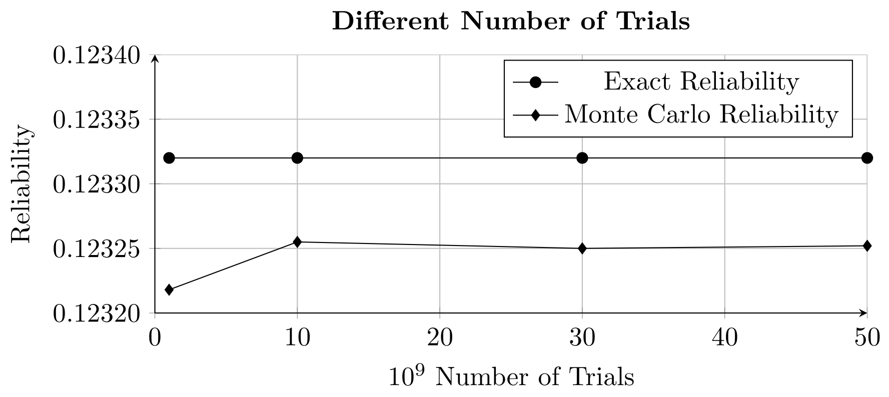

Fig. 4.3. *Graph of the Monte Carlo reliability and of the exact reliability obtained from using different number of trials.*

Table 4.3
Results from the Monte Carlo method using MPI parallelism on a 5X5-Grid using different number of trials.

Processors	$R_{s,t}(G,D)$	CPU-t (s)	% Difference in Reliability
1	0.123218	4094.94	.086
10	0.123255	4126.91	.056
30	0.123250	4151.21	.060
50	0.123252	4205.73	.061

MC, we ran tests using the same 5x5-Grid undirected graph, where s and t correspond to vertices 0 and 24, respectively, and whose exact reliability is 0.123324 (see Fig. 4.1-a). The total number of trials was fixed to one billion. We ran our tests using 1, 10, 30, and 50 processors as seen in table 4.2 column 1. Column 2 shows the reliability obtained from the fixed trials using MC. Column 3 shows the execution time to complete the trials. Column 4 shows the percentage difference in reliability between our MC implementation and the exact evaluation. Figure 4.2 corresponds to the graphical representation of the table.

Since our algorithm assigns each processor a unique seed for the random number generator, we expect with more processors, thus more seeds, the reliability estimation may become more accurate. However, the tests show that as the number of processors grows, the accuracy of the reliability barely changes. Therefore, using a large number of processors alone would not significantly affect the accuracy of the MC estimation.

4.3. Investigating the effects of increasing the total number of trials. To investigate the effects of increasing the total number of trials, we have run tests using the same 5x5-Grid graph (edge reliabilities are set to 0.5), and as in the previous subsection, s and t correspond to vertices 0 and 24, respectively, and whose exact reliability is 0.123324 (see Fig. 4.1-a). In order to test increasing the number of trials, we've increased the number of processors and assigned one billion trials per processor, which scales up the total number of trials. By using 1, 10, 30, and 50 processors as depicted in table 4.3 (corresponding graphical representation is illustrated by Fig. 4.3) column 1, we were able to test the effects of using 1, 10, 30, and 50 billion trials. Column 2 shows the reliability obtained from the increased trials using MC. Column 3 shows the execution time to complete the trials. Column 4 shows the percentage

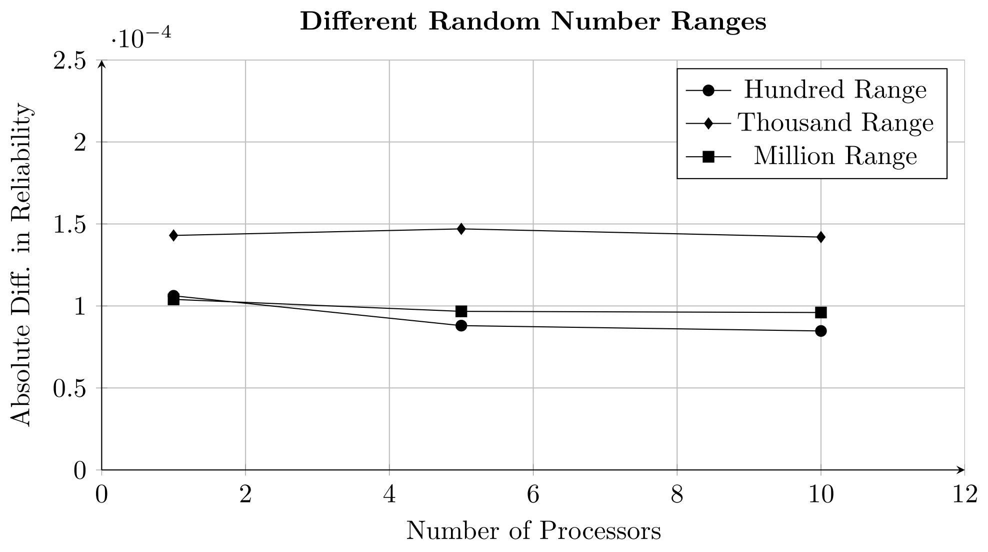

FIG. 4.4. *Graph of the absolute difference between the exact reliability and the Monte Carlo reliability obtained from using different random number ranges: hundred, thousand, and million as the number of processors increase (the total trials are fixed to a billion).*

TABLE 4.4

Results from the Monte Carlo method using MPI parallelism obtained from using different random number ranges: hundred, thousand, and million on a 5X5-Grid with the number of processors increasing (the total trials are fixed to a billion).

Ranges	Processors	$R_{s,t}(G,D)$	CPU-t (s)	% Difference in Reliability
Hundred	1	0.123218	4094.94	.086
	5	0.123236	776.73	.071
	10	0.123239	394.38	.069
Thousand	1	0.123467	3837.62	.116
	5	0.123471	792.24	.119
	10	0.123466	393.19	.115
Million	1	0.123428	3871.91	.084
	5	0.123421	779.11	.078
	10	0.123420	396.27	.078

difference in reliability between our MC implementation and the exact evaluation.

As the results show, the reliability obtained from using MC becomes more accurate as the number of trials increases. However, we see that pass 10 billion trials the change in reliability becomes less significant. With 10 billion trials, the reliability obtained using Monte Carlo only differs from the exact reliability by .056 percent.

4.4. Testing the effects of generating a larger range of random numbers. To investigate the effects of generating larger random number ranges, to either include or delete an edge in a trial (see Section 3), we ran tests using the same 5x5-Grid graph (edge reliabilities are set to 0.5), under the assumption that $s = 0$ and $t = 24$ (i.e., exact reliability is 0.123324). The random number ranges are hundred (i.e., range [1,100], see Section 3), thousand, and million as depicted in table 4.4 column 1 (see also Fig. 4.4). The tests were performed on 1, 5, and 10 processors (column 2) while keeping the total number of trials fixed to a billion. Column 3 shows the reliability obtained from the different ranges generated as the number of processors increased using MC. Column 4 shows the execution time to complete the trials, and column 5 shows the difference in reliability between our MC implementation and the exact evaluation.

Since we are interested in how the reliability is affected by generating different random number ranges, the chart in figure 4.4 illustrates the difference in reliability estimated by the MC approach, compared to the exact evaluation as the number of processors increase. As the results show, there isn't a noticeable improvement in reliability by increasing the range of random numbers generated.

5. Conclusions. The purpose of this work is to study an alternative way to efficiently estimate the Diameter-Constrained Reliability (DCR) of a probabilistic graph, and the method we chose was the Crude Monte Carlo method using MPI parallel processing. We were interested in studying how the DCR estimation is affected by changing different factors when applying parallelism such as number of cluster nodes, number of trials, and the possible range of numbers generated to either include or exclude an edge. We thought perhaps a large number of processors might improve the reliability estimation, while maintaining the total number of trials constant, because this would allow for more seeds for the random number generator. Similarly, we consider the case of increasing the range of random numbers generated. However, we found that neither the number of processors nor the random number ranges generated (while keeping the other parameters fixed) played a significant role in the reliability approximation. The total number of trials was shown to affect the reliability approximation, but the changes in the reliability were insignificant after a sufficiently large number of trials (10 billion in our case).

Acknowledgments. This REU (Research Experience for Undergraduates) was sponsored by the National Science Foundation and United States Department of Defense. We would like to thank the referees for their important suggestions of how to improve the quality of the paper. Special thanks to our mentor Dr. Louis Petingi for his support and guidance.

REFERENCES

[1] CUNY Performance Computing Center: HPC Systems. *The City University of New York College of Staten Island.* http://www.csi.cuny.edu/cunyhpc/HPC_Systems.html

[2] H. Cancela, F. Robledo, G. Rubino, and P. Sartor, Monte Carlo estimation of diameter-constrained network reliability conditioned by pathsets and cutsets, *Computer Communications*, 2012, on-line publication.

[3] H. Cancela and L. Petingi, Reliability of Communication Networks with Delay Constraints: Computational Complexity and Complete Topologies, *International Journal of Mathematics and Mathematical Sciences* (2004)-29, 2004, pp. 1551–1562.

[4] H. Cancela and M. El Khadiri, A Recursive Variance-Reduction Algorithm For Estimating Communication-Network Reliability, *IEEE Transactions on Reliability.* 44(4), 1995, pp. 595–602.

[5] H. Lin and L. Petingi, CSI-REU Reliability Project: MC to estimate the DCR using MPI libraries. https://github.com/HLin212/dcrmc. 2016.

[6] S. Deering and R. Hinden. *RFC 2460, Internet Protocol, Version 6 (IPv6) Specification.* December 1998.

[7] H. Cancela and M. El Khadiri, On the RVR simulation algorithm for network reliability evaluation, *IEEE Transactions on Reliability*, 52(2):207–212, 2003.

[8] L. Petingi, Diameter-related Properties of Graphs and Applications to Network Reliability Theory, *WSEAS Transactions on Math.* 12(9), 2013, pp. 884–894.

[9] L. Petingi and J. Rodriguez, Reliability of networks with delay constraints, *Congressus Numerantium* 152, 2001, pp. 117–123.

[10] M. Ball, Computational complexity of network reliability analysis: An overview, *IEEE Trans. Reliab.* R-35(3), 1986, pp. 230–239.

[11] M. Bell, Computational complexity of network reliability analysis: An overview, *IEEE Trans. Reliab.* R-35(3), 1986, pp. 230–239.

[12] M. J. Quinn, Parallel Programming in C with MPI and OpenMP, *McGraw-Hill*, 2003.