PyRBD: An Open-Source Reliability Block Diagram Evaluation Tool

Shakthivelu Janardhanan*, Sareh Badnava*, Ritanshi Agarwal† and Carmen Mas-Machuca*†
*Chair of Communication Networks (LKN), Technical University of Munich (TUM), Germany
†Chair of Communication Networks, University of the Bundeswehr Munich (UniBw), Germany
{shakthivelu.janardhanan, sareh.badnava}@tum.de, {ritanshi.agarwal, cmas}@unibw.de

***Abstract*—Reliability Block Diagrams (RBDs) are pictorial representations that evaluate a system's availability by considering its components and their interconnections. In communication networks, a flow's availability is evaluated using the network's RBD. However, there are no open-source tools to evaluate complex networks' RBDs. We present an open-source tool, `PyRBD`, to (i) generate RBDs from topologies, (ii) decompose RBDs for faster processing, and (iii) calculate flow availabilities. `PyRBD`'s effectiveness was measured on five topologies based on simulation time and complexity. Additionally, single-core and multicore implementations were studied. Though the use-case in this work is networks, `PyRBD` is suitable for any system.**

***Index Terms*—Reliability Block Diagram (RBD), `PyRBD`, open-source, availability, reliability.**

I. Introduction

A system's availability is the probability that it delivers its expected service at a particular time instant [1]. Complex systems consist of several different interconnected components. The availability evaluation of such systems is critical to meeting dependability standards and the system's operational efficiency. One of the most widely used tools for achieving this goal is the Reliability Block Diagram (RBD). An RBD is a graphical description demonstrating how a component's failure may affect the overall system's availability.

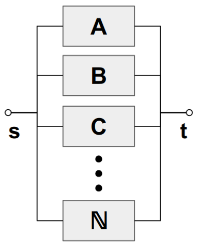
The most known RBD architectures are the parallel and series RBDs, which are depicted in Figs. 1a and 1b, respectively. The parallel system's availability (A_{par}) is given by,

$$A_{par} = 1 - \prod_{i=A}^{\mathbb{N}} (1 - A_i), \tag{1}$$

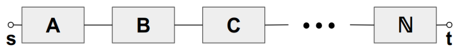
where A_i is component i's availability. The series system's availability (A_{ser}) is given by,

$$A_{ser} = \prod_{i=A}^{\mathbb{N}} A_i. \tag{2}$$

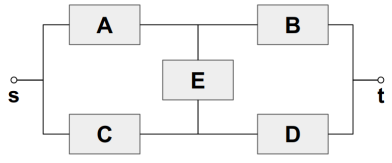
The bridge RBD in Fig. 1c is a more complicated RBD. Here, the component E is neither in series nor in parallel with the others. In such a case, the Conditional Decomposition Method (CDM) is employed to derive two different RBDs: one where E is available, as seen in Fig. 2a and another where E is unavailable, as seen in Fig. 2b. The availabilities of these



(a) Parallel RBD.

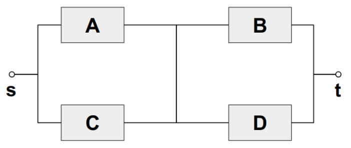


(b) Series RBD.

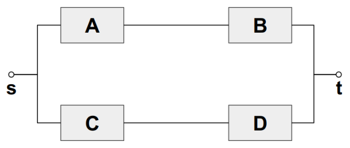


(c) Bridge RBD.

Fig. 1. Simple Reliability Block Diagrams (RBD).



(a) Bridge RBD with E available.



(b) Bridge RBD with E unavailable.

Fig. 2. Bridge Reliability Block Diagram (RBD).

RBDs are denoted as $\widehat{A_E}$ and $\widehat{A_e}$, respectively. The bridge system's availability A_s is calculated as,

$$A_{CDM} = A_E.\widehat{A_E} + (1 - A_E).\widehat{A_e}, \tag{3}$$

where, $\widehat{A_E}$ and $\widehat{A_e}$ are given by,

$$\widehat{A_E} = (1-(1-A_A)(1-A_C)) \times (1-(1-A_B)(1-A_D)), \tag{4}$$

$$\widehat{A_e} = 1 - (1 - A_A A_B)(1 - A_C A_D). \tag{5}$$

The same method can be used for RBDs of any size. However, as the RBD size and the number of elements requiring the CDM increases, the computational complexity increases. Several recent studies on network dependability focusing on working path availability [2]–[4], highly available spine substructures [5], disaster-resilient routing schemes [6], [7], data center sovereignty [8], [9], and core network sovereignty [10], [11] have tried to improve network robustness from different perspectives. However, the lack of an RBD evaluation tool to calculate the overall network availability has inhibited performing a full dependability analysis considering different parameters like availability, reliability, and sovereignty together. Therefore, there is a strong need for an open-source software tool to solve this problem. To address this need, we developed `PyRBD` [12], an open-source software tool for RBD evaluation. Although this work focuses on communication networks, the methods described and the Python library `PyRBD`

This work has been funded by the Bavarian Ministry of Economic Affairs, Regional Development, and Energy under the project '6G Future Lab Bavaria'.

are applicable to any system that can be represented as an RBD. The main contributions of this work are as follows.

(i) Different methods to study reliability, a comparison of RBD evaluation tools, and the theory on minimal cut sets (MCS) and CDM are discussed (Sec. II).
(ii) `PyRBD`'s methodology to apply the CDM to decompose and evaluate the complex RBDs based on the MCS approach is explained (Sec. III).
(iii) `PyRBD`'s performance in the single-core and multicore implementations are compared (Sec. IV).

II. Related Work and Background

A. Methods to study reliability

Several methods exist to study system reliability.

- Monte Carlo Simulation [13] is effective in handling non-linear and complex systems. This approach involves the simulation of several possible scenarios to accurately model the uncertainty in the system.
- Bayesian Reliability Analysis [14] is a dynamic and iterative framework to evaluate system availability accurately.
- Markov Chain Monte Carlo [15] method models the complex stochastic behavior of system components by generating probabilistic models that show the interaction between components over time.
- Fault Tree Analysis [16] identifies possible component failures that may result in the overall system failure. This is useful in identifying the critical weak spots and determining the chance of failure.
- Petri Nets [17] uses stochastic activity networks to calculate component availability based on several input parameters. These component availabilities are integrated into a hierarchical model to obtain the system availability.
- RBDs are simple but efficient tools to study system availability based on components' availabilities. Extended RBDs such as the Dynamic RBD (DRBD) [18] also exist for time-variant reliability modeling. However, we only consider time-invariant systems in this work. Time-variant DRBDs are marked for future work.

However, all these methods, except the RBDs, are computationally intensive for a large system. For a complex network topology with even a few nodes, the above methods grow infeasible. Therefore, RBDs are best suited for evaluating network availability based on node availabilities.

B. Other software tools to evaluate RBDs

Some previous software tools also perform RBD evaluation. However, they have certain disadvantages.

- *librbd* [19] is a published library for RBD evaluation. However, *librbd* is limited by its formalism. That is, *librbd* does not accept network topologies as input. The user has to manually describe the series, parallel, and bridge connections in the topology and generate the RBD first. This process is tedious for large topologies.
  For example, consider the sample Germany_17 topology [20] in Fig. 3. Such a large network topology is nearly impossible to describe with just the series, parallel, and bridge notations because several nodes are a part of multiple bridge connections. Additionally, the RBD for each flow changes with the source and destination. Therefore, *librbd* would require the user to generate separate RBDs manually for each flow. Therefore, *librbd* is infeasible and, hence, unsuitable for our application.
- Other python libraries like *PyReliability* [21], *Reliability˙Analyzer* [22], and *Fiabilipy* [23] exist. However, they also share the same drawbacks as *librbd*. Moreover, these tools only work with directed graphs. However, in a network, the links are most often bidirectional. In this case, the RBD evaluation tools are prone to forming loops inside the RBD, causing inaccurate results. Our goal is to avoid all the aforementioned issues.
- Commercial tools like ReliaSoft [24], Relyence [25], Isograph [26], and ExtendSim [27] also perform reliability studies. However, these expensive commercial tools have different formalisms and do not operate on a common programming paradigm. Moreover, their formalisms can not directly convert topologies to RBDs.

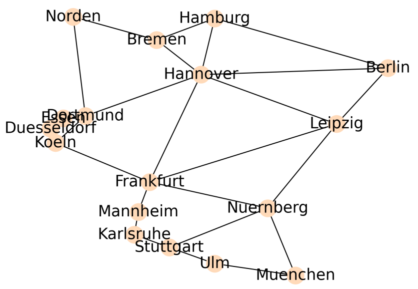


Fig. 3. Germany_17 topology [20]

Our goal in this work is to develop a scalable, open-source tool to evaluate RBDs. The methodology used to achieve this is discussed in the following section.

C. Minimal cut set (MCS) approach in literature

A cut set of a system is a set of elements that, when removed, cause the system to fail. A minimal cut set (MCS) is a cut set that can not be further reduced while still being a cut set. For example, consider the example Germany_17 topology in Fig. 3. Consider a connection between Karlsruhe and Leipzig. (Frankfurt, Mannheim, Stuttgart) is a valid cut set. However, it can be further reduced to (Mannheim, Stuttgart) and still be a cut set. The new set can not be further reduced, and still be a cut set. Therefore, this new set is an MCS.

Our solution, CDM, with the MCS approach, evaluates the RBD by identifying the path sets with non-overlapping terms and performing boolean addition to obtain flow availability. Boolean expansion [28] in the probabilistic domain was proposed much earlier. However, the authors in [29] identified a critical issue that boolean expressions in the probabilistic domain might have overlapping terms being counted multiple times, leading to inaccurate results. Therefore, they propose a method to rewrite the boolean expression to remove overlapping terms. In our work, without using this method, the CDM with the MCS approach automatically removes all overlapping terms and produces an accurate result. The MCS approach has also been used previously for other applications like Binary Decision Diagrams in fault tree analysis [30]. However, to the best of our knowledge, we are the first to use the MCS approach with CDM to evaluate RBDs.

Moreover, the authors in [29] also identified programming complexity as a limiting factor in [28] and proposed other software implementations to evaluate RBDs. However, they assume only directed, acyclic, and simple (no self-loops) graphs. In reality, communication networks are more complex, with bidirectional links and loops. Therefore, our solution, the CDM with the MCS approach in `PyRBD`, is designed to efficiently handle these complex scenarios.

III. Methodology

The objective of this work is as follows. Given a network topology, a flow represented by a source-destination pair, and node availabilities as input, the CDM with the MCS approach

(i) iteratively decomposes the network RBD to identify all the path sets in the network,
(ii) produces the corresponding boolean expression, and
(iii) calculates the flow availability as output.

In this study, the links are considered to be fully available. Therefore, the RBD from a topology is the same as the topology itself. If the user wishes to use link availabilities, `PyRBD` works by considering the links as well as the nodes as blocks in the RBD.

A. Theoretical upper bound on the number of branches in the Conditional Decomposition Method (CDM)

The CDM considers the presence and absence of each node in the RBD to identify path sets in the RBD. However, if the nodes in the RBD are randomly decomposed, the number of branches increases exponentially with the total number of nodes in the RBD. That is, to consider all possible combinations in an RBD with n nodes, the number of branches would be 2^n. Therefore, a more efficient method must be adopted to account for scalability. In this work, this challenge is overcome by utilizing the properties of the MCSs. When the nodes in the MCSs are removed first, that branch leads to a cut set, and the branching ends faster. This prevents unnecessary branching. This is further explained in the following example.

B. Conditional Decomposition Method (CDM) + Minimal cut set (MCS) approach

Let us consider the working of the CDM in `PyRBD` with the example topology shown in Fig. 1c. The example follows the method described in Algorithm 1. First, for the source-destination pair (s, t), we check if there is a one-hop path. If a one-hop path exists, then the flow availability (A_F) is given by the product of the source A_s and the destination A_t availabilities. If not, then the MCSs for the flow are calculated. In this topology, the set of MCSs κ for the flow are $\{(A, C), (B, D), (B, E, C), (A, E, D)\}$. The nodes are then ordered based on the most repeated in the smallest MCS. Here, there are only 2 MCSs with cardinality two, with different nodes. Moreover, in this simple example, since there is no repetition of nodes in the first two MCSs, the four nodes can be taken in any order. In this example, we consider the order of removal μ as $\{A, C, B, D, E\}$. C is decomposed second because (A, C) is an MCS. The example branching shown in Fig. 4a corresponds to the topology in Fig. 1c.

Then, the branching procedure commences with node A chosen for decomposition. First, node A is considered fully available, as shown in Fig. 4b. Then, the new topology is tested for a one-hop path between s and t. In this case, there is no one-hop path, but a multi-hop path exists. Therefore, this branch is marked by -1. On the other hand, node A is considered fully unavailable, as shown in Fig. 4c. Once again, there are no one-hop paths. Therefore, this branch is also marked by -1. The capital and small letter boolean format used in Fig. 4a refers to the case where the node is fully available and unavailable, respectively.

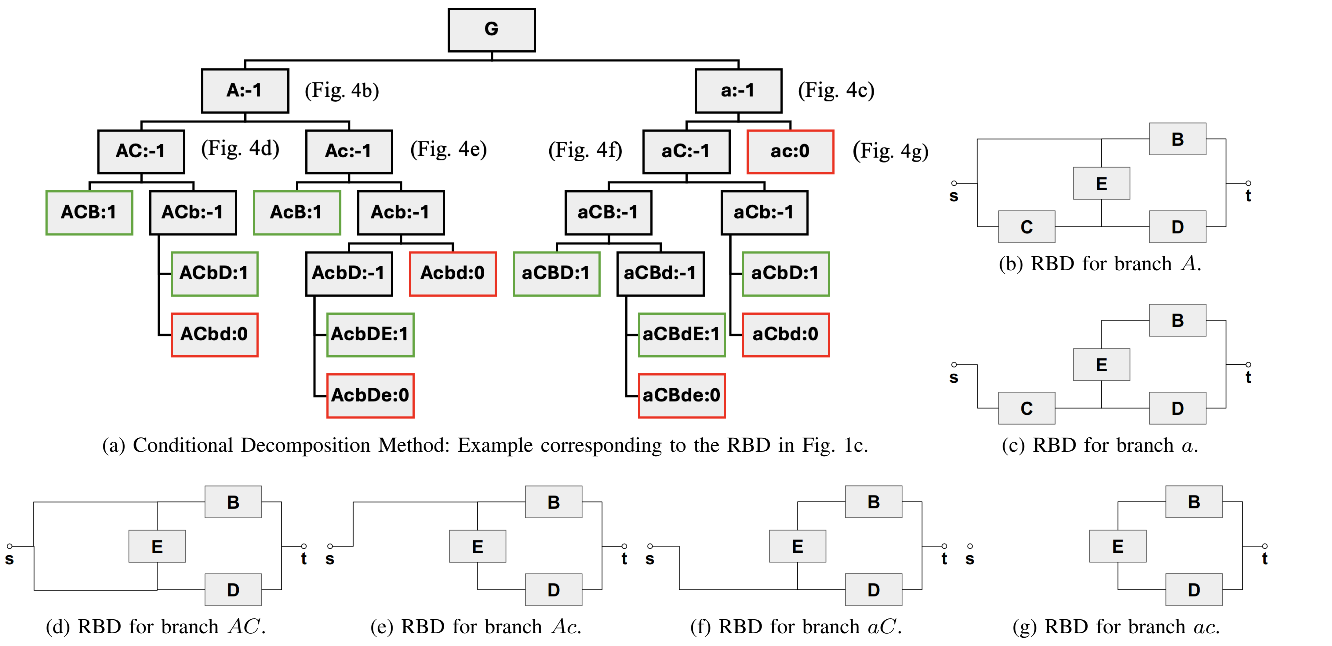
In the next iteration, node C is chosen for decomposition so as to remove the MCS (A, C). In each branch previously marked by -1, node C is considered to be fully available and fully unavailable sequentially. This gives rise to four different graphs and their respective branches. The new graphs considering the decomposition of C are (a) AC, where A and C are available as seen in Fig. 4d, (b) Ac, where A is available and C is unavailable as seen in Fig. 4e, (c) aC, where A is unavailable and C is available as seen in Fig. 4f, and (d) ac, where A and C are unavailable as seen in Fig. 4g. Here, all branches except ac have a multi-hop path between s and t, and hence they are marked by -1. However, branch ac marked by 0 is an MCS of the flow. This branch has no path between s and t, and hence, the branching stops.

Therefore, by adopting an approach with the MCSs, the branching in several cases can be terminated in the early stages. This termination of branches is very helpful because it makes the CDM method in `PyRBD` scalable and fast. The branching for the other three branches marked by -1 is continued until all branches either reach a 1 (one hop path exists) or a 0 (no path exists).

After the branching is completed, all the boolean expressions for the path set branches marked by 1 in Fig. 4a are added together to obtain the availability.

$$\begin{aligned} A_{CDM} &= ACB + ACbD + AcB + AcbDE + aCBD + \\ &\quad aCBdE + aCbD \\ &= A_A A_C A_B + A_A A_C (1 - A_B) A_D + \\ &\quad A_A (1 - A_C) A_B + A_A (1 - A_C)(1 - A_B) A_D A_E + \\ &\quad (1 - A_A) A_C A_B A_D + (1 - A_A) A_C A_B (1 - A_D) A_E \\ &\quad + (1 - A_A) A_C (1 - A_B) A_D \end{aligned} \tag{6}$$

On expanding Eqs. 3 and 6, the resultant expressions are the same. Note that adding all the boolean expressions for the branches marked by 0 gives the unavailability. Subtracting the unavailability from 1 also yields the availability. Finally,



(a) Conditional Decomposition Method: Example corresponding to the RBD in Fig. 1c.

(b) RBD for branch A.

(c) RBD for branch a.

(d) RBD for branch AC.

(e) RBD for branch Ac.

(f) RBD for branch aC.

(g) RBD for branch ac.

Fig. 4. Conditional Decomposition Method (CDM) - New RBDs corresponding to the branching of the example RBD in Fig. 1c and the branches in Fig. 4a.

this sum is also multiplied by the source and destination availabilities to obtain the overall flow availability. Using the MCS approach, the number of end branches reduces from the theoretical upper bound at $2^5 = 32$ to 17. This shows the scalability of the CDM with the MCS approach. For larger real-world networks, this improvement is more significant, as explained in Sec. IV-C.

IV. Results

A. Input and simulation setup

`PyRBD` was tested on a MacBook with an M1 chip (8-core GPU), 8GB memory, and 512GB storage. Five different topologies, such as Abilene [31] (11 vertices and 14 edges), Polska [20] (12 vertices and 18 edges), Germany_17 [20] (17 vertices and 26 edges), HiberniaUK (Ring topology) [31] (13 nodes and 13 edges), and dfn-bwin (Germany mesh) [20] (10 nodes and 45 edges) were evaluated to study the change in `PyRBD`'s performance for topologies of varying size and connectivity. `PyRBD` takes the topology inputs in the form of NetworkX graphs [32]. `PyRBD` was processed with three different techniques, such as single-core simulation, multiprocessing, and multithreading, to compare the computational efficiency of the techniques. The major difference between multiprocessing and multi-threading in Python is in the usage of the memory space. In multi-threading, the different threads use the same memory space concurrently, while the processes in multiprocessing use different memory spaces parallelly. Performance metrics, such as the number of branches in the CDM for each flow, simulation time per flow in the single-core implementation, and simulation time for different processing techniques, were measured. The obtained results show that `PyRBD` efficiently handles complex, large-scale, and real-life networks. Note that `PyRBD` can also be used for any system that can be represented as an RBD.

B. Comparison of single and multicore implementations

Fig. 5 compares the simulation times for the different implementations on the Y-axis for the corresponding topologies on the X-axis. First, there is a vast decrease in the simulation time when `PyRBD` runs on multiple threads or multiple cores. There are some exceptions here. For example, multiprocessing is usually the fastest solution. However, in the case of Abilene, multiprocessing is slower. This is because the overhead caused by the multiprocessing implementation is larger than that of the multi-threading and single-core implementations. The same is applicable for the dfn-bwin topology also. Therefore, if the topology is large and computation-intensive, multiprocessing is the best solution. If the topology is smaller with fewer nodes, multi-threading is recommended.

C. Comparison of number of decomposed branches

Fig. 6 compares the number of decomposed branches for each flow as a box plot along the Y-axis for each topology on the X-axis. The theoretical upper bound on the number of branches in CDM was discussed in Sec. III-A to be 2^n, for a topology with n nodes. This upper bound is also marked with a blue dot for each topology in Fig. 6. For example, in Germany_17, CDM with the MCS approach reduces the number of branches from the upper bound by over 10^3 in the median case. Note that in all topologies, there are flows with zero branches. These are the one-hop flows where there is no branching required as per lines 1, 29, and 30 in Algorithm 1. The number of branches for the dfn-bwin topology remains at 10^0 for all the flows because this is a full mesh network. All source-destination pairs have a one-hop path. This is also the reason for dfn-bwin to have very low simulation time, as seen

Algorithm 1 RBD Evaluation with CDM, given a topology G and a source-destination pair (s, t).

```
1:  if no one-hop path between $s$ and $t$ then
2:      Find all MCSs $\kappa$ except the MCSs with only the source
        and destination.
3:      Order the nodes in MCSs as per the most repeating
        nodes in the smallest MCSs to obtain the set of nodes
        in the order of removal $\mu$ for RBD decomposition.
4:      Start CDM branching method.
5:      for each node ‘$n$’ in $\mu$ do
6:          Consider that node ‘$n$’ is fully available $(N)$.
7:          if one-hop path exists between $s$ and $t$ then
8:              stop branching (marked by ‘:1’).
9:          else if multiple-hop path exists between $s$ and $t$ then
10:             continue branching (marked by ‘:-1’).
11:         else
12:             stop branching (marked by ‘:0’).
13:         end if
14:         Consider that node ‘$n$’ is fully unavailable $(n)$.
15:         if one-hop path exists between $s$ and $t$ then
16:             stop branching (marked by ‘:1’).
17:         else if multiple-hop path exists between $s$ and $t$ then
18:             continue branching (marked by ‘:-1’).
19:         else
20:             stop branching (marked by ‘:0’).
21:         end if
22:         Iteratively decompose branches that are marked ‘:-1’.
23:         if no more branches exist with ‘:-1’ then
24:             Exit For loop.
25:         end if
26:     end for
27:     Add the availability of all available branches (branches
        marked by A=1) to get $\widehat{A_F}$.
28:     Multiply $s$ and $t$ availabilities with $\widehat{A_F}$ to obtain the
        flow availability $A_{(s,t)}$ .
29: else
30:     Multiply $s$ and $t$ availabilities to get flow availability
        $A_{(s,t)}$.
31: end if
```

in Fig. 5. This analysis shows the massive improvement from the upper bound for PyRBD's CDM with the MCS approach.

Another interesting observation is that the median number of branches for Germany_17 is lesser than that of HiberniaUK and Polska. This is because the number of branches in CDM with the MCS approach relies on the cardinality of the MCS ($|MCS|$) and the number of MCSs for a flow (κ). Polska and Germany_17 have an average $|MCS|$ of 3.2 and 2.5, respectively. On the other hand, though HiberniaUK has a low average $|MCS|$ of 2, the average κ (excluding the source-destination MCSs) is 18.34. Germany_17 has an average κ of 7.14. Therefore, the size of the topology alone does not influence the number of decomposed branches. It is also dependent on $|MCS|$ and $|\kappa|$.

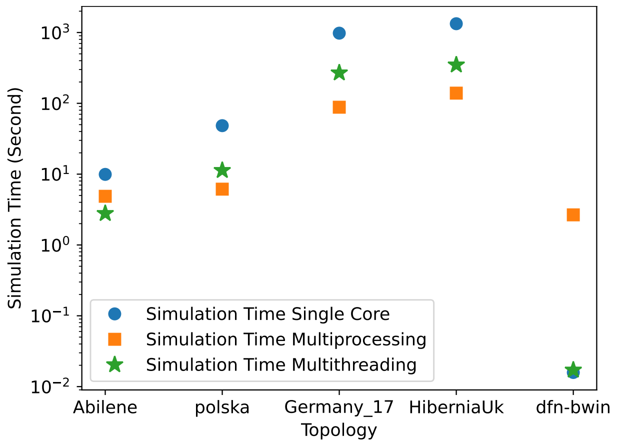


Fig. 5. Simulation Time (seconds) vs. Topology: Multiprocessing is better for large topologies, and multi-threading is better for smaller topologies in terms of number of nodes.

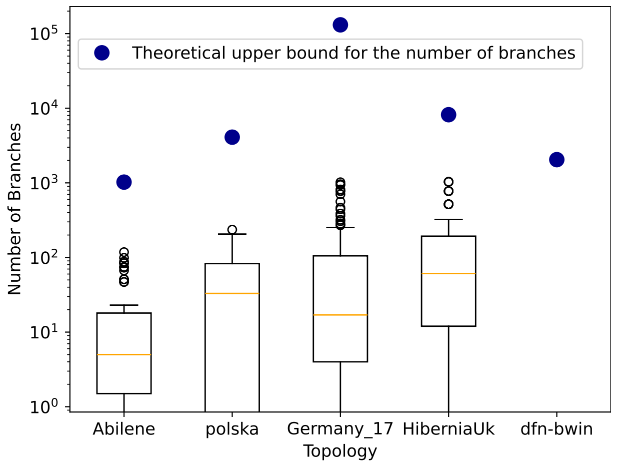


Fig. 6. Number of branches vs. Topology: Theoretical upper bound (2^n) for each topology is plotted to show the reduction in the number of decomposed branches when using PyRBD.

D. Comparison of single-core simulation times

Fig. 7 compares the simulation time per flow as a box plot along the Y-axis for each topology on the X-axis. The generally observed trend is that the simulation time per flow increases with the topology's size. Though the number of branches for Germany_17 is smaller than that for Polska and HiberniaUK, the simulation time is greater for Germany_17. This is because the PyRBD's computation time depends more on the topology's size because new graphs are generated with each branch. dfn-bwin has a low simulation time because it does not involve any branching due to all the one-hop paths.

E. Sample output from PyRBD

Fig. 8 shows the sample output obtained from PyRBD. The flow availabilities of all the flows are on the Y-axis as a boxplot for the respective topology on the X-axis. In this example, all the nodes in each topology have 0.99 availability. The user can specify the source-destination pairs separately and/or change the node availabilities as required. In this example, the flow

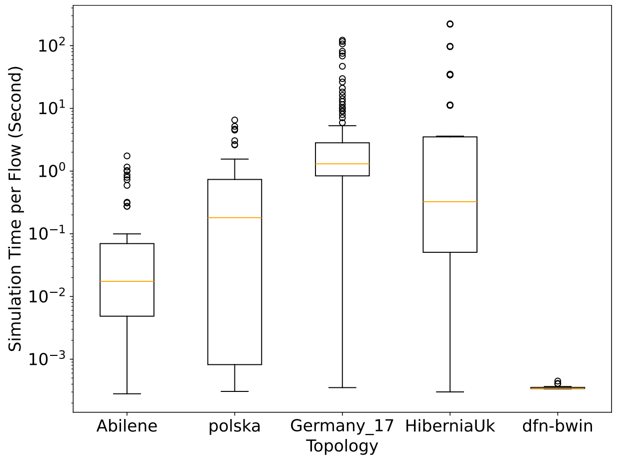


Fig. 7. Simulation Time per flow vs Topology: Simulation time per flow increases with the topology's size.

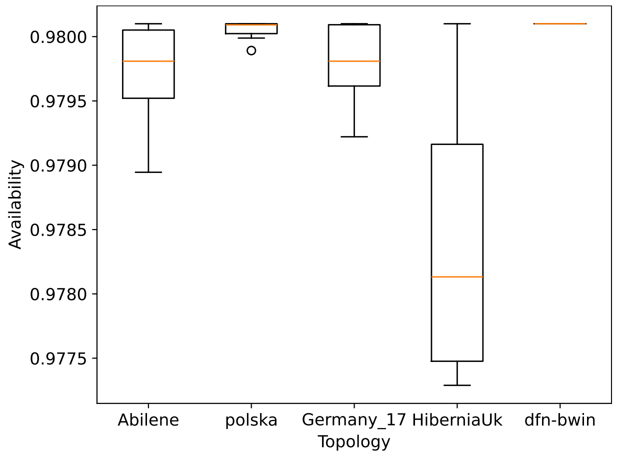


Fig. 8. Sample output from `PyRBD` for all source-destination pairs, for each node with 0.99 availability in each topology.

availabilities for dfn-bwin are very high because of all the one-hop paths. The availabilities for HiberniaUK vary across a large spectrum since, in a ring structure, all flows (except the ones with one-hop paths) have two paths in parallel with varying numbers of nodes in each path.

V. Conclusions and Outlook

This work presented `PyRBD` [12], an open-source Python tool for efficient Reliability Block Diagram (RBD) evaluation. Unlike other open-source tools, `PyRBD`'s Conditional Decomposition Method (CDM) with the minimal cut set (MCS) approach is capable of evaluating complex RBDs based on cyclic, bidirectional networks (and any other systems). We also proposed guidelines on using `PyRBD` in the multiprocessing mode for large topologies, and the multi-threading mode for smaller topologies. Through `PyRBD`, we hope to move towards studies based on overall network availability.

References

[1] A. Avizienis *et al.*, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[2] S. Janardhanan *et al.*, "Zohra: Joint Routing and Manufacturer Assignment Problem," in *IEEE Int. Commun. Qual. and Rel. Workshop (CQR)*, Oct. 2023.

[3] D. A. Schupke and F. Rambach, "A link-flow model for dedicated path protection with approximative availability constraints," *IEEE Commun. Lett.*, vol. 10, no. 9, pp. 679–681, 2006.

[4] S. Yang *et al.*, "Availability-based path selection and network vulnerability assessment," *Netw.*, vol. 66, no. 4, pp. 306–319, 2015.

[5] A. Alashaikh *et al.*, "The spine concept for improving network availability," *Comp. Netw.*, vol. 82, pp. 4–19, 2015.

[6] J. Rak and D. Hutchison, *Guide to disaster-resilient communication networks*. Springer Nature, 2020.

[7] A. de Sousa *et al.*, "Determination of the minimum cost pair of D-geodiverse paths," in *13th Int. Conf. on the Des. of Rel. Commun. Netw. (DRCN)*. VDE, 2017, pp. 1–8.

[8] S. Janardhanan and C. Mas-Machuca, "Modeling and evaluation of a data center sovereignty," in *18th Int. Conf. on the Des. of Rel. Commun. Netw. (DRCN)*, 2022, pp. 1–8.

[9] ——, "Modeling and evaluation of a data center sovereignty with software failures," in *6th Int. Conf. on Syst. Rel. and Saf. (ICSRS)*, 2022, pp. 233–242.

[10] S. Janardhanan *et al.*, "Improving network sovereignty - a minimal cut set approach," in *24th Int. Conf. on Transparent Opt. Netw. (ICTON) Demo Zone*, 2024.

[11] ——, "Network sovereignty: A novel metric and its application on network design," 2024. [Online]. Available: https://arxiv.org/abs/2407.03814

[12] S. Janardhanan and S. Badnava, "PyRBD," GitHub repository. [Online]. Available: https://github.com/shakthij98/PyRBD/

[13] Y. Jiang *et al.*, "The method of network reliability and availability simulation based on monte carlo," in *2012 Int. Conf. on Qual., Rel., Risk, Maintenance, and Saf. Eng.*, 2012, pp. 245–250.

[14] J. Torres-Toledano and L. Sucar, "Bayesian networks for reliability analysis of complex systems," vol. 1484, 01 1998, pp. 465–465.

[15] Y. Zhang *et al.*, "Probabilistic analysis of network availability," in *IEEE 30th Int. Conf. on Netw. Protocols (ICNP)*, 2022, pp. 1–11.

[16] R. Patil, "An overview of fault tree analysis (FTA) method for reliability analysis," *J. of Eng. Res. and Stud.*, vol. 4, pp. 6–8, 03 2013.

[17] S. Janardhanan and C. Mas-Machuca, "Availability modeling and evaluation of switches and data centers," in *Int. Conf. on Dependable Syst. and their Appl. (DSA)*, Aug. 2023.

[18] S. Distefano and A. Puliafito, "Dependability evaluation with dynamic reliability block diagrams and dynamic fault trees," *IEEE Trans. Depend. Sec. Comput.*, vol. 6, no. 1, pp. 4–17, 2009.

[19] L. Carnevali *et al.*, "An efficient library for reliability block diagram evaluation," *Appl. Sci.*, vol. 11, no. 9, p. 4026, 2021.

[20] S. Orlowski *et al.*, "SNDlib 1.0-survivable network design library," vol. 55, no. 3. Wiley Online Library, 2010, pp. 276–286.

[21] X. Zhang, "PyReliability," GitHub repository. [Online]. Available: https://github.com/THUzxj/reliability-models

[22] Asad1287, "Reliability Analyzer," GitHub repository. [Online]. Available: https://github.com/Asad1287/Reliability˙Analyzer/

[23] V. Lecrubier, "Fiabilipy," GitHub repository. [Online]. Available: https://github.com/crubier/Fiabilipy

[24] ReliaSoft, "ReliaSoft BlockSim." [Online]. Available: https://www.hbkworld.com/en/products/software

[25] Relyence, "Relyence RBD." [Online]. Available: https://relyence.com/products/rbd/

[26] Isograph, "Reliability workbench." [Online]. Available: https://www.isograph.com/

[27] ExtendSim, "Reliability Block Diagramming in ExtendSim." [Online]. Available: https://extendsim.com/products/line/rbd

[28] R. B. Hurley, "Probability maps," *IEEE Trans. Rel.*, vol. R-12, no. 3, pp. 39–44, 1963.

[29] R. Bennetts, "Analysis of reliability block diagrams by boolean techniques," *IEEE Trans. Rel.*, vol. R-31, no. 2, pp. 159–166, 1982.

[30] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Comput. Sci. Rev.*, vol. 15, pp. 29–62, 2015.

[31] S. Knight *et al.*, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765 –1775, Oct. 2011.

[32] A. Hagberg *et al.*, "Exploring network structure, dynamics, and function using NetworkX," 1 2008. [Online]. Available: https://www.osti.gov/biblio/960616