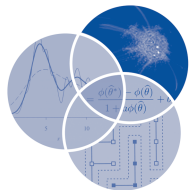
Advanced Review



Network reliability evaluation

Mehmet Sahinoglu[1]* and Benjamin Rice[2]

This article, beyond presenting a spectrum of network reliability methods studied in the past decades, describes a scalable innovative 'overlap technique' to tackle large complex networks' reliability evaluation difficulties, which cannot be handled by straightforward reliability block diagramming (RBD) techniques used for the simple parallel-series topologies. Examples are shown on how to apply the overlap algorithm to compute the ingress-egress reliability. Monte Carlo simulations demonstrate the methods discussed. (1) Static (time independent), (2) dynamic (time dependent) using a versatile Weibull distribution to represent the multiple stages of network components from infancy to useful life period and to wear-out, and (3) multistate versions to include derated behavior beyond conventional working and nonworking states, are illustrated for calculating the directional source-target (s-t) reliability of complex networks by using the Java software ERBDC: *Exact Reliability Block Diagramming Calculator*.
WIREs Comp Stat

Network reliability is the probability that a network with all its subnetworks and constituting components will successfully complete the task it is intended to perform under the conditions encountered for the specified period of time defined between a source and a target.[1–11] Reliability analysis is the process of quantifying a system's ingress–egress [or source–target (s–t) at will] serviceability by examining the dependency relationships between the components that comprise the system. Analysis is essential whenever the cost of failure is high.[12,13] Modeling and simulation allow analysts to determine weak spots in the systems so that the maintenance engineer can inventory a backup list of components. The reliability analysis focuses on the computer network components and the connections between them to determine the overall system reliability as well as the reliabilities between any two individual nodes in the network. Network reliability computations are similar to those developed for industrial applications, but there are a few exceptions. In industrial applications, all of the components in the system are usually considered critical to the overall function of the system. However, in network applications, the target communication between two nodes may select few components in the system due to redundancy.[11,14,15]

Currently, most published educational materials cover methods for determining system reliabilities in networks that can be expressed as pure parallel–series systems or reducing a complex topology to a parallel–series one using a conditional 'keystone' method.[10] However, as experience proves, these ready-to-cook networks with small sizes rarely occur outside textbooks and classrooms. These computations prove impossible or mathematically unwieldy when applied to real complex networks and are therefore useful only to teach basic reliability concepts.[11] The graphical screening ease and convenience of reliability block diagramming (RBD) algorithms[16] is advantageous for planners and designers trying to improve system reliability by allowing quick and efficient intervention that may be required at a dispatch center to observe routine operations and identify solution alternatives in case of a crisis. The Boolean decomposition and binary enumeration algorithms[17–19] are outside the practical scope of this article because of large networks we will work with. The algorithm through a user-friendly and graphical Java applet computes the reliability of any complex parallel–series network. Furthermore, the coded topology can be transmitted remotely and then reverse-engineered to reconstruct the original network diagram for purposes of securing classified information and saving space.[12,13,15,20–23,25] This, too, can be applied to security-related input for wired or wireless systems. All current exact

*Correspondence to: msahinog@aum.edu

[1]Informatics Institute, Auburn University, Montgomery, AL 36124, USA

[2]Computer Science Department, Troy University, Montgomery, AL 36104, USA

DOI: 10.1002/wics.81

computational algorithms for general networks are based on enumeration of states, minpaths, or mincuts.[2,3] Network reliability estimation has been used successfully for nontrivial-sized networks using neural networks and heuristic algorithms in Refs 7 and 12 as well as employing a concurrent error detection approach by the coauthor of earlier research.[23] Other researchers have used efficient Monte Carlo simulation.[4,5] Bounds such as Jan's upper bound, used to reduce the complexity of computations, are approximate.[3] A thorough analysis is by Colbourn.[1] The next sections demonstrate recent progress in s–t network reliability field with software tools.[14,15,24–27]

OVERLAP TECHNIQUE TO CALCULATE COMPLEX NETWORK RELIABILITY

With the emergence and maturity of reliability engineering over the past six decades since the space age started, there is a growing need to efficiently produce accurate reliability models for increasingly larger and more complex systems. For purely parallel–series networks, there are simple and widely available algorithms to mathematically determine exact reliabilities between a given ingress and egress node. For complex networks, however, there is very little in the way of freely available algorithms that are simple, accurate, efficient, and scalable. This article discusses the discovery and implementation of one such 'overlap' algorithm by treating nodes and links in (1) single-state static, (2) multistate static, and (3) single-state time-dependent dynamic with a Weibull failure probability density function.[11,14,15,27]

Every network can be represented, in its simplest form, as a binary state chart representing all of the possible states and their respective reliabilities, whose sum is always 1. The reliability from one node to another is the sum of all states that constitute at least one path of sequentially connected nodes that form an unbroken link between the source node and the target node. The source and target nodes can be interchanged with no impact on the final result.

Example 1. Consider the five-node sample network in Figure 1. Each node has a reliability of 0.9, meaning that 90% of the time it is up (operating, functional, working, etc.), whereas 10% of the time it is down (not operating, dysfunctional, not working, etc.). The state diagram would look like the following. The '1-5 probability' column is only populated in $2^5 = 32$ states that constitute a complete link between node 1 and node 5, where the rest are left out with zero results in Table 1.

TABLE 1 | Five-Node/Two-Path Static Network State Enumeration Table for '1 – 5' in Figure 1

Node						
1	2	3	4	5	Probability	1–5 Probability
1	0	1	0	1	0.00729	0.00729
1	0	1	1	1	0.06561	0.06561
1	1	0	1	1	0.06561	0.06561
1	1	1	0	1	0.06561	0.06561
1	1	1	1	1	0.59049	0.59049
Total reliability					1	0.79461

Note that this returns the same results as standard parallel–series equations that can be performed manually:

$$R_T = R_1 \times \{1 - [(1 - R_3) \times (1 - R_2 \times R_4)]\} \times R_5$$
$$R_T = 0.9 \times \{1 - [(1 - 0.9) \times (1 - 0.9 \times 0.9)]\} \times 0.9 = 0.79461. \quad (1)$$

Using a binary state chart works fine for very small networks, but the exponential nature of the underlying data render it impossible to use for larger networks. A relatively small 10-node network would require 10 column and 2^{10} (1025) rows (10,250 total cells), whereas a still small 20-node network would require 20 columns and 2^{20} (1,048,576) rows (20,971,520 total cells). This is too much data to handle efficiently in the calculation of small network reliabilities. These state charts can be, however, useful in analyzing how individual paths affect each other while generating reliabilities as a system.

For instance, consider the same five-node network in Figure 1. We can quickly deduce that the two unique paths through the system from node 1 to node 5 are {1→3→5} and {1→2→4→5}. More on algorithmically determining paths will follow. If all of the nodes in either path are up, there will be a continuous connection between the source and target nodes. If we map the two paths in the state diagram

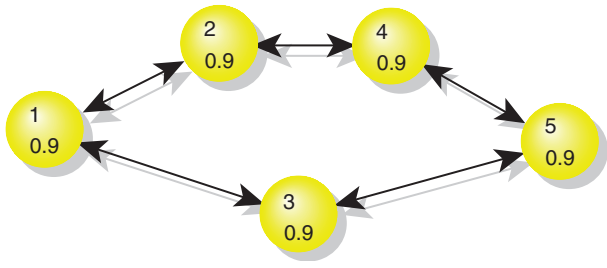


FIGURE 1 | *Example 1*: five-node/two-path static network.

TABLE 2 | Five-Node/Two-Path Static Network State Table with Useful Paths for Figure 1

Node							
1	2	3	4	5	Probability	1,3,5	1,2,4,5
1	0	1	0	1	0.00729	0.00729	
1	0	1	1	0	0.00729		
1	0	1	1	1	0.06561	0.06561	
1	1	0	1	1	0.06561		0.06561
1	1	1	0	1	0.06561	0.06561	
1	1	1	1	1	0.59049	0.59049	0.59049
			Total reliability		1	0.729	0.6561

below (all of the states where node 1 is down have been hidden to reduce the size), we find that the sum of two reliabilities (=1.385) is exactly 0.59049 greater than the desired result. This is the probability of overlap between the two paths. The overlap is found by taking a union of all the nodes in both paths. This union is simply a mathematical state overlap and does not need to represent a path across (see Table 2).

Unfortunately, reliability calculation for complex networks is not as simple as subtracting the overlaps between unique paths. When there are more than two paths, many states may overlap across multiple paths. When subtracting overlaps, any state that has been subtracted more than once will have to be re-added to the final result. Note: s(source), t(target).

Generating Minimal Paths

This practice into the binary nature of reliability networks and the overlapping characteristics of path reliabilities lends itself to the idea of a structured algorithm that could compute a finite set of minimal paths from ingress to egress. One considers the individual reliabilities of each path, and programmatically adjusts for overlaps. The first step, determining a set of minimal paths, is of utmost importance regarding efficiency and scalability. Theoretically, we could consider the same exact path multiple times, and the algorithm would determine that a duplicate path is a complete overlap. We could then systematically subtract all redundant state reliabilities. However, after brief analysis of the problem pattern, it is apparent that path overlap will have to be computed triangularly. It follows then that any duplication of initial paths can affect the total algorithm time exponentially.

Finding the set of minimal paths across a network from one node to another can be accomplished using a simple stack. As the algorithm will step through node connections in sequence, it is helpful to have the networked nodes uniquely ordered to prevent confusion as to which nodes have already been addressed and which node should come next at any given point along the algorithm. Ordering can either be accomplished by alphabetic sorting, if the nodes have been uniquely named beforehand, or simply assigning an abstract index $(1, \ldots, n)$ to each node before beginning. The order of the nodes is not important as the set of minimal paths generated will be the same regardless of order. Start by pushing the start node onto the initially empty stack and repeat the following algorithm on the top stack node until the stack is again empty.

1. If the top node connects directly to the target node:
   - **a.** Push the target node onto the stack.
   - **b.** Save the current stack as a new minimal path (the bottom node is the start node and represents the start of the minimal path, whereas the top node is now the target node representing the end of the path).
   - **c.** Pop the top two nodes (the target node and the node that links directly to it) off of the stack.
2. If the top node does not connect directly to the target node, push the next connecting node onto the stack. The next connecting node can be determined by the following logic:
   - **a.** No node that is already in the stack is a candidate. This would cause a recursion issue as paths could fall into a recurring loop that grows infinitely in size without ever reaching the target node.
   - **b.** If the previous action was a push action (i.e., the current node was newly pushed onto the stack) choose the lowest ordered node connected to the top stack node that is not already in the stack.
   - **c.** If the previous action was a pop action (i.e., one of the nodes connected to the top node had finished its iterations and was removed) choose the next ordered node connected to the top stack node after the previously popped node.
3. If the top node does not connect directly to the target node and no new connecting node was found to push onto the stack, pop the top node off the stack.

Using the simple network depicted below in Figure 2, it is possible to walk through the minimal

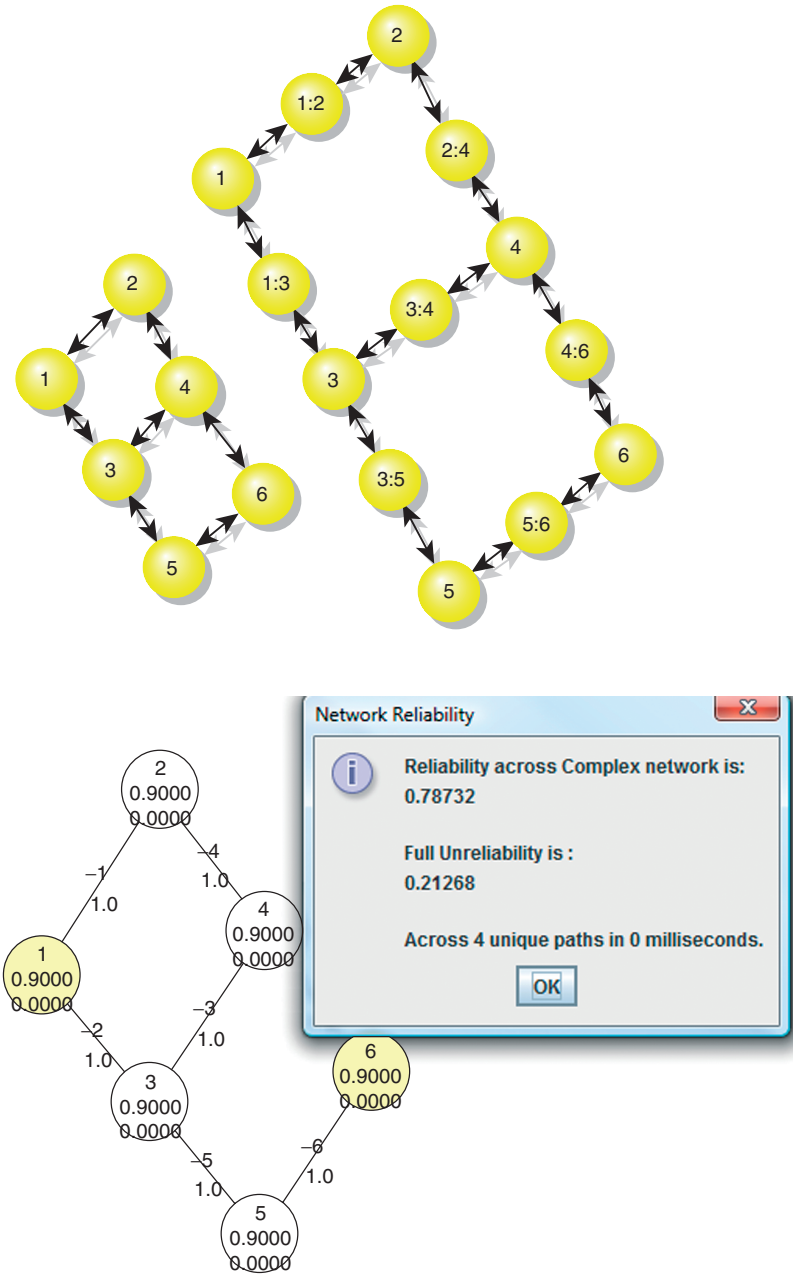


FIGURE 2 | *Example 2*: six-node network without including link reliabilities (= 1.0) and (s = 1, t = 6) solution.

path algorithm step by step to find the set of minimal paths across the network starting from node 1 and traversing to node 6.

Step 1. Add the start node to the stack.

1

Step 2. The top node (1) connect to both nodes 2 and 3 and neither one is already in the stack. As the last action was a push action, push the lowest order node, 2, to the stack.

1	2

Step 3. The top node (2) connects to nodes 1 and 4, but 1 is already in the stack, so push node 4 onto the stack.

1	2	4

Step 4. Because the top node (4) links directly to the target node (6), push 6 onto the stack and save the current stack as a minimal path. Then, pop the top two nodes off of the stack. *Note*: even though there are still possible paths through node 4 (e.g., 1,2,4,3,5,6), any such path will be short circuited by 1,2,4,6 and

thus will not need to be considered as a minimal path. Therefore, node 4 can be popped from the stack.

1	2	4	6

Add this to the list of minimal paths.

1	2

Pop the last two nodes off the stack.

Step 5. Because node 4 was just popped from the stack and the top node on the stack (2) does not link to any nodes higher than 4, pop node 2 off the stack.

1

Step 6. Node 2 was just popped off the stack; the next higher node than 2 connected to the top stack node (1) that is not already in the stack is node 3. Push node 3 onto the stack.

1	3

Step 7. The top node (3) connects to nodes 1, 4, and 5, but 1 is already in the stack, so push the lowest (4) remaining node onto the stack.

1	3	4

Step 8. The top node (4) connects directly to the target node (6), push 6 on the stack and save the current stack as a minimal path. Then, pop the top two nodes off the stack. *Note*: at this point, it is clear to see that if node 4 did not link directly to the target node, node 2 would have been the first node to be pushed onto the stack. However, from node 2, there would be no way to link to the target node (6) without passing through any nodes already in the stack. In this case, the process would still need to consider node 2 to insure total algorithm inclusion. In small networks such as this example, it is easy to visualize dead ends, but in a very large network, they may not be so clear and taking shortcuts could result in missed paths and thus incorrect results.

1	3	4	6

Add this to the list of minimal paths.

1	3

Pop the last two nodes off the stack.

Step 9. Because node 4 was just popped off the stack, the next highest node connected to the top node (3) that is not already in the stack is node 5. Push node 5 on to the stack.

1	3	5

Step 10. The top node (5) connects directly to the target node (6), push 6 on the stack and save the current stack as a minimal path. Then, pop the top two nodes off the stack.

1	3	5	6

Add this to the list of minimal paths.

1	3

Pop the last two nodes off the stack.

Step 11. Node 5 was just popped off the stack and there are no higher nodes connected to the top node (3), pop node 3 off of the stack.

1

Step 12. Node 3 was just popped off the stack and there are no higher nodes connected to the top node (1), pop node 1 off of the stack.

Step 13. The stack is now empty; the algorithm is complete. The minimum paths generated above are as follows:

1	2	4	6
1	3	4	6
1	3	5	6

The core algorithms presented here never consider link reliabilities. When link reliabilities are required, think of each link as a node itself. Of interesting note is, when considering link reliabilities, steps that would have resulted in an immediate link to the target node (i.e., steps 4, 8, and 10 in the minimal path example), now have opportunities to venture down additional paths. Networks with link reliability tend to have several more minimal paths than their counterparts without link reliabilities. See *Example 2* in Figure 2 to observe all above.

Overlap Method Algorithm

Now that the minimal set of paths has been generated, the overlap method can be executed. The overlap method can be implemented with a finite array of stacks. Each stack will be comprised of one or more sets $\{S1, \ldots, Sn\}$. Each set will be comprised of one or more nodes $\{N1, \ldots, Nn\}$. The initial stack will be the set of minimal paths generated across the network from the ingress to the egress nodes. Stacks will create additional stacks recursively per the logic below. A single reliability will be maintained (referred to as the global or working reliability), and each stack will increment or decrement the global reliability accordingly. The initial global reliability is 0.

1. Determine if this stack will be added to or subtracted from the final reliability. If this is the root stack (the set of minimal paths generated for the network), this stack will be added; otherwise

the operation will be the opposite of the calling stack.

2. Eliminate any complete overlaps from the stack.
   a. For each set Si in $S1, \ldots, S(n-1)$:
      i. For each set Sj in $S(i+1), \ldots, Sn$:
         1. If $Si\{\} \cap Sj\{\} = Si\{\}$, remove Sj from the stack.
         2. Else if $Si\{\} \cap Sj\{\} = Sj\{\}$, remove Si from the stack.
3. If there is only one set remaining in the stack:
   a. Add or subtract (see Step 1) the product of the sole set. The product of a set is the product of the reliabilities of the nodes in the set $(N1 \times N2 \times \cdots \times Nn)$. Return to the calling stack. If this is the root stack, the algorithm is complete.
4. For each remaining set in the stack {S1,..., Sn}, add or subtract (see set 1) the product of the set. The product of a set is the product of the reliabilities of the nodes in the set $(N1 \times N2 \times \cdots \times Nn)$.
5. If there are more than one set in the stack:
   a. For each set Si in $S2, \ldots, Sn$:
      i. Create a new empty stack.
      ii. For each set Sj in $S1, \ldots, S(i-1)$.
         1. Add a new set comprised of nodes $Si\{\} \cup Sj\{\}$ to the new stack. *Note*: this is a proper union, not a union all. If a given node occurs in both Si and Sj, it should occur only once in the new set.
      iii. Perform the overlap algorithm on the new stack.
6. Return to the calling stack. If this is the root stack, the algorithm is complete.

Example 3. For the complex network in Figure 3, if all the nodes are assumed to be 0.9; using 'overlap' algorithm:

$$\begin{aligned}\text{IN}-\text{OUT}(\text{s}=7,\ \text{t}=8) &= \{7\}[\{1-4\}+\{1-5\}\\ &+\{2-5\}+\{3-5\}+\{3-6\}-\{1-4-5\}\\ &-\{1-2-5\}-\{1-2-4-6\}-\{2-5-6\}\\ &-\{1-3-4-6\}-\{1-3-5-6\}\\ &-\{2-3-6\}+\{1-2-4-5-6\}\\ &+\{1-2-3-4-6\}\\ &-\{1-2-3-4-5-6\}]\{8\}. \qquad (2)\end{aligned}$$

$$\begin{aligned}\text{IN}-\text{OUT}(\text{s}=7,\ \text{t}=8) &= (0.9)\{(5)(0.9^2)-[(4)(0.9^3)\\ &+(3)(0.9^4)]+(4)(0.9^5)-(0.9^6)\}(0.9)\\ &= (0.9)\{(5)(0.81)-(4)(0.729)-3(0.6561)\\ &+4(0.59049)-0.531441\}(0.9)\\ &= (0.9)\{4.05-4.8843+2.36196\\ &-0.531441\}(0.9)\\ &= (0.996219)(0.81)=0.80694. \qquad (3)\end{aligned}$$

MONTE CARLO AND DISCRETE EVENT SIMULATION TO DEMONSTRATE THE OVERLAP METHOD

The component reliabilities will be simulated using a Bernoulli probability density function (pdf) given the static (time-independent) input data r_i for each component. That is, draw a random deviate, u_i, from the unity uniform (0,1). Lay out a network composed of these r_i. Announce it a passage (success) if $0 < u_i < r_i$. If not, component is a failure. Then tally one if there is a working connection between the ingress and egress components for this iteration. In iteration 2, apply the same method as in iteration 1. Tally if there is a connection between the source and target. Once you reiterate this routine, say with $n = 100{,}000$ runs for the network destination, and calculate the ratio of successful arrivals of service from source (ingress) to target (egress). Compute the overall simulation average by dividing the number of service connections by n. The links will also be simulated with respect to a Bernoulli pdf for P (=probability of being operative) taken as a constant. Generate a Bernoulli random deviate; that is, draw a uniform u_i, if $0 < u_i < P$, then it is a hit (success) for the link. Therefore, for example, you can advance through the link from 1 to 2. If link reliability is perfect then do not generate anything, simply advance to the neighboring component. The number of times out of n trials to advance from s to t with all successful hits will yield the output.[15,26,27]

Using Poisson counting process, for time-dependent discrete event simulation technique, assume the rates for each component, such as λ = (mean sojourn time)$^{-1}$. Let us assume that the probability of 'up' for a component, such as 0.9 denotes 9 time units out of 10 are operating and 1 time unit out of 10

not operating. The reciprocals of the means yield the rates: λ (failure rate) $= 1/9$, and μ (repair rate) $= 1/1$. Thus, FOR $= \lambda/(\lambda + \mu) = (1/9)/[(1/9) + (1/1)] = (1/9)/(10/9) = 0.1$. Using these rates of sojourn to generate times to failure with a negative exponential pdf, one travels from component to component. If both sending and receiving ends operate at the same time, it is a successful connection. How many successes out of how many 'n' trials, such as from $s = 1$ to $t = 19$ as in Figure 4 is the solution.

Slower (due to housekeeping records of clock time) dynamic simulation result is almost equal to that of the static Monte Carlo, which is easier to program. The overlap method has advantages of being 100% accurate and faster compared with simulation. However, when it comes to larger networks with more than 30 nodes and 50 links, the storage and time become a problem. *Examples 4* and *5*: see Figures 4 and 5 for details regarding a 19-node/32-link and a 52-node/72-link complex network.

MULTISTATE SYSTEM (MSS) RELIABILITY EVALUATION

When modeling real-life reliability systems, it does not always suffice to present an 'on or off only' paradigm. Most live systems such as electric power networks contain components that can be not only fully operational or completely nonfunctional but also a range of in-between states of marginal utility such as derated or degraded. It is sometimes inadequate to describe a node's states with only UP (fully operating) and DOWN (fully deficient) but with more states like DERATED (partially operating close to full) or even DEGRADED (less partially operating close to DOWN) or more downgraded states. Components in these states are functional but not operating at peak capacity. In computer networks, these conditions could be caused by network devices that are currently experiencing extraordinary computing loads. Routers sometimes are capable of trafficking data packets but only in limited counts. In electric power systems, they are the hydroelectric plants or turbines, etc. with less than full capacity because of lack of water or insufficient fuels such as coal, lignite, or gas.

Let us examine briefly a simple multistate network. For this example, we will consider a three-state system; UP (fully operational), DER (derated or partially operational), and DN (down and fully nonoperational). Each state has a reliability value between 0 and 1 representing the probability that the component will be in that state at any given time. The

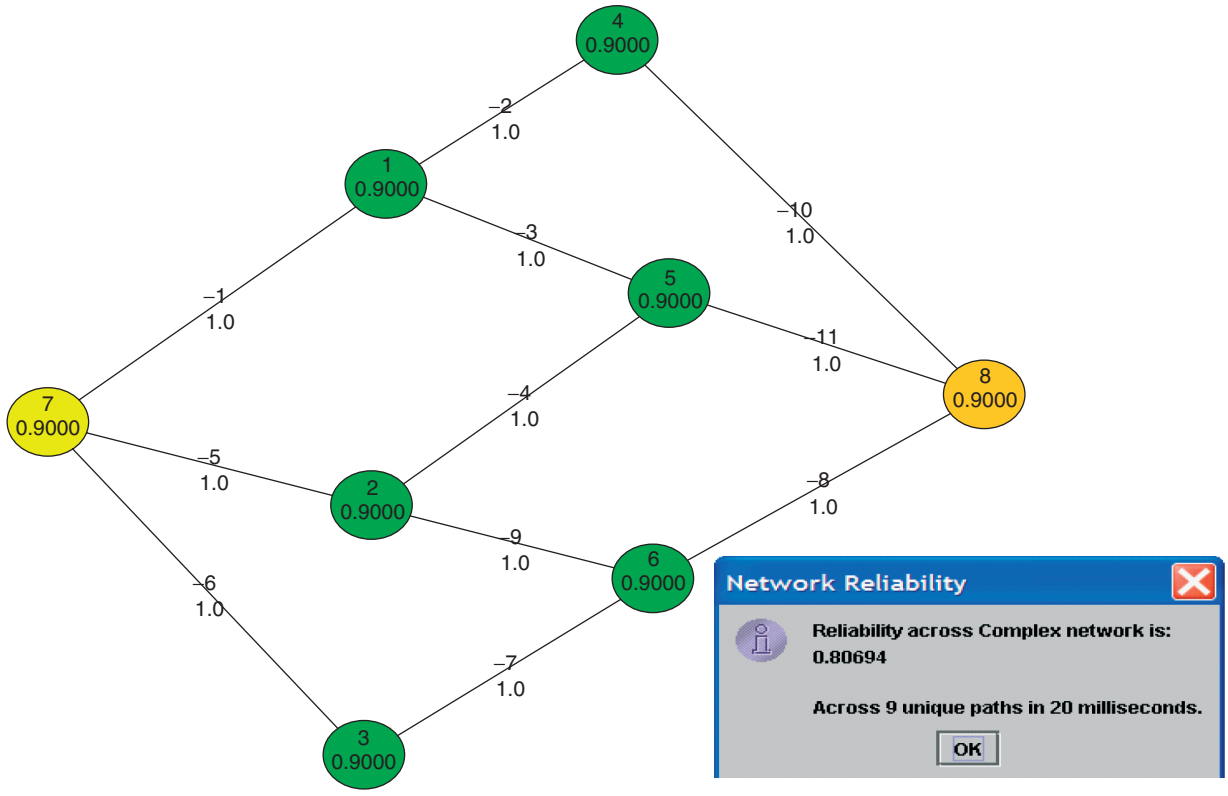
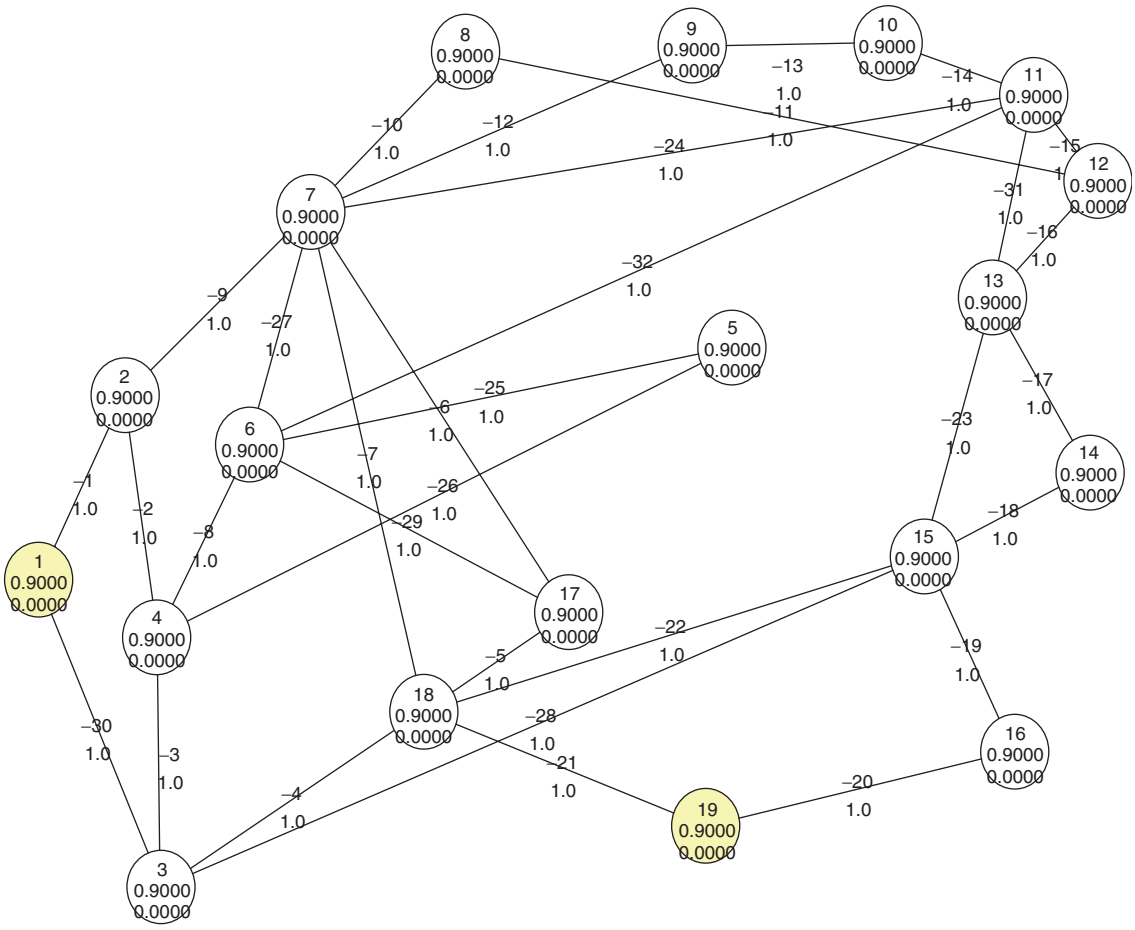
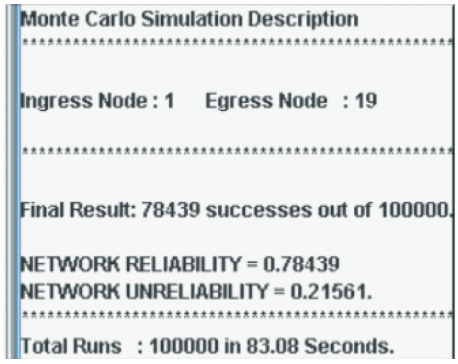


FIGURE 3 | *Example 3*: eight-node/nine-path static network with $s = 7$, $t = 8$.





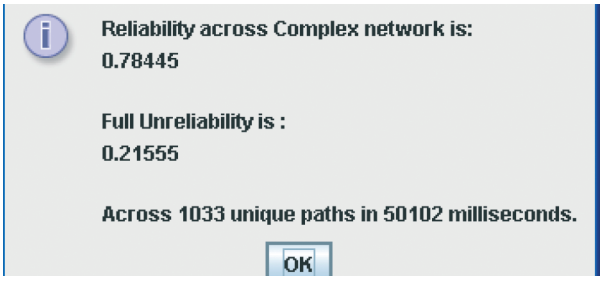


FIGURE 4 | *Example 4:* 19-node ($s = 1$, $t = 19$) complex network with Monte Carlo simulation (83 s) and analytical results (50 s), both 0.784 with simulation taking 33 s more.

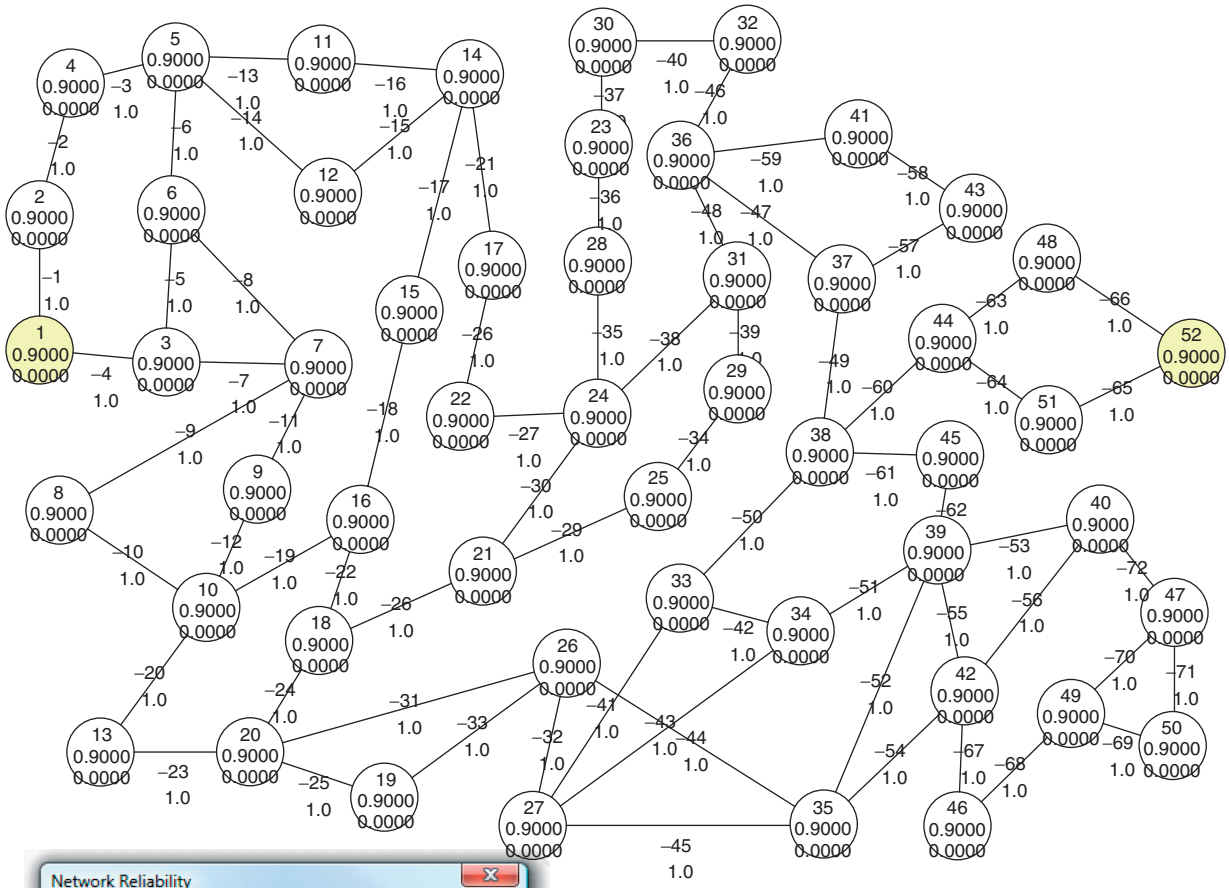


FIGURE 5 | *Example 5:* Simulation for ($s = 1, t = 52$) with 72 links is 0.558 in 17,119 s. The faster analytical result is 0.556 in 18.034 s in 0.1% of the much longer simulation time. The more simulation runs are conducted, the closer the analytical and simulation results will converge to each other.

sum of the reliabilities of the states must always equal 1. In this respect, what we referred to as static reliability in the previous sections is really two-state, UP and DN. The UP probability was stated while the DN probability was left unstated and assumed at {1—UP}. In the same fashion, when we depict multistate reliabilities we indicate the UP value first, followed by the remaining functional states in operational order. We omit the DN state, again leaving it assumed. For example, if a component is UP 70% of the time, DER 20% of the time, and DN the remaining 10%, we would write {0.7, 0.2} to indicate the component reliability summary. It is imperative that these states add up to unity (1.0). Let us take the situation where there are three states, UP, DER, and DN, and study for simple series and active parallel systems.

Simple Series System with Singly Derated States

A simple series system with fully operating and derated states is shown in Figure 6 in *Example 6*.

Our goal is to calculate the simplest series system reliability of a primitive example, where the node has three states, with probabilities, $P(\text{UP}) = 0.7$, $P(\text{DER}) = 0.2$, and $P(\text{DN}) = 0.1$ in Figure 1. We use two approaches:

Longer State Enumeration Approach

In this example, there can be $S^N = 3^2 = 9$ combinations, where S is the number of states and N is the number of nodes. $S = 3$ and $N = 2$ as follows:

$$\begin{aligned}
P(\text{UP and UP}) &= 0.7^2 = 0.49\\
P(\text{UP and DER}) &= (0.7)(0.2) = 0.14\\
P(\text{UP and DN}) &= (0.7)(0.1) = 0.07\\
P(\text{DER and UP}) &= (0.2)(0.7) = 0.14\\
P(\text{DER and DER}) &= 0.2^2 = 0.04\\
P(\text{DER and DN}) &= (0.2)(0.1) = 0.02\\
P(\text{DN and UP}) &= (0.1)(0.7) = 0.07\\
P(\text{DN and DER}) &= (0.1)(0.2) = 0.02\\
P(\text{DN and DN}) &= 0.1^2 = 0.01\\
\text{Sum of probabilities} &= 1.00. \qquad (4)
\end{aligned}$$

Of these nine combinations, the state that yields a fully UP combination is the first line with $P(\text{UP and UP}) = 0.7^2 = 0.49$. Then states that are indicative of the system being inoperative are those states on the third and sixth to ninth lines in Eq. (4) above, which contain at least one DOWN state, which sum to 0.19. The DERATED states on the second, fourth, and fifth lines add up to 0.32. $P_{\text{sys}}(\text{DER}) = 1 - P_{\text{sys}}(\text{UP}) - P_{\text{sys}}(\text{DN}) = 1 - 0.49 - 0.19 = 1 - 0.68 = 0.32$.

Shortcut Formulation Approach

Working on the same two-node simple series system, let us calculate the system up, derated, and down probabilities in Figure 6.

$$\begin{aligned}
P_{\text{sys}}(\text{UP}) &= P_1(\text{UP})\\
P_2(\text{UP}) &= (0.7)(0.7) = 0.49. \qquad (5)\\
P_{\text{sys}}(\text{DER}) &= P_1(\text{UP} + \text{DER})P_2(\text{UP} + \text{DER})\\
&\quad - P_{\text{sys}}(\text{UP}) = (0.7 + 0.2)^2 - 0.7^2\\
&= 0.81 - 0.49 = 0.32. \qquad (6)\\
P_{\text{sys}}(\text{DN}) &= 1 - P_{\text{sys}}(\text{UP}) - P_{\text{sys}}(\text{DER})\\
&= 1 - 0.49 - 0.32 = 0.19. \qquad (7)
\end{aligned}$$

Active Parallel System with Singly Derated States

The parallel system with IN($s = 1$) and OUT($t = 4$) both 100% reliable, 2 and 3 DER are shown in Figure 7 in *Example 7*:

Longer State Enumeration Approach

The system-UP scenario is when at least one of the middle two states is UP. This is possible when UP–UP, UP–DER, DER–UP, UP–DN, and DN–UP combinations exist whose sum $= 0.49 + 0.14 + 0.14 + 0.07 + 0.07 = 0.91$. Then the system-DER is when at least one of the states is DER. This is when DER–DER, DER–DN, DN–DER combinations exist whose sum $= 0.04 + 0.02 + 0.02 = 0.08$. The only remaining combination is DN–DN, whose probability is $0.1^2 = 0.01$, or by subtraction.

Shortcut Formulation Approach

Working on the same four-node simple parallel–series system, let us calculate the system up, derated, and down probabilities in Figure 7.

$$\begin{aligned}
P_{\text{sys}}(\text{UP}) &= P_1(\text{UP})\{1 - [1 - P_2(\text{UP})][1 - P_3(\text{UP})]\}\\
P_4(\text{UP}) &= (1.0)[1 - (1 - 0.7)^2](1.0) = (1.0)(0.91)\\
&= 0.91. \qquad (8)\\
P_{\text{sys}}(\text{DER}) &= P_1(\text{UP})P_4(\text{UP})\\
&\quad - [1 - \{1 - P_2(\text{UP} + \text{DER})\}\\
&\quad \times \{1 - P_3(\text{UP} + \text{DER})\}]\\
&\quad - P_{\text{sys}}(\text{UP}) = 1^*(1 - 0.1^2) - 0.91\\
&= 0.99 - 0.91 = 0.08. \qquad (9)\\
P_{\text{sys}}(\text{DN}) &= 1 - P_{\text{sys}}(\text{UP}) - P_{\text{sys}}(\text{DER})\\
&= 1 - 0.91 - 0.08 = 0.01. \qquad (10)
\end{aligned}$$

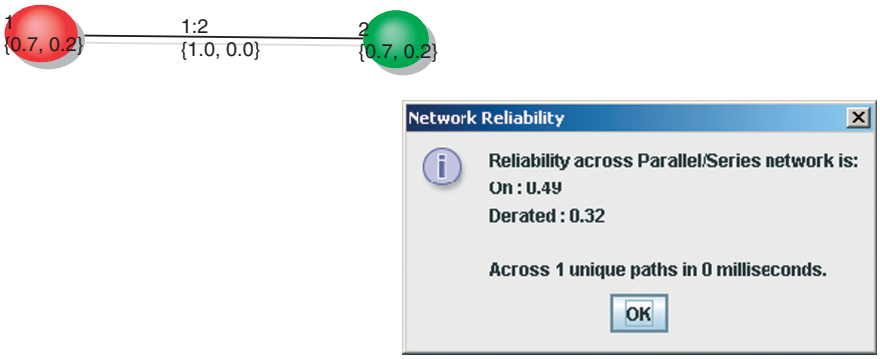


FIGURE 6 | *Example 6:* simple series system with a derated state.

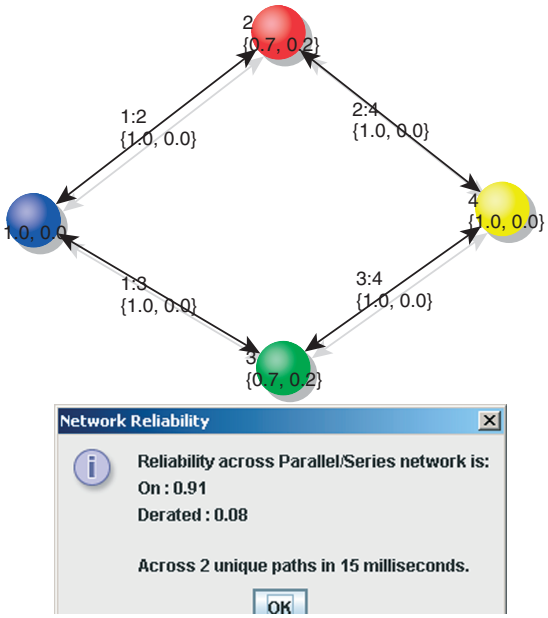


FIGURE 7 | *Example 7:* active parallel system with IN(1) and OUT(4) full reliability; others(2,3) are derated.

Simple Parallel–Series System with Singly Derated States

A simple parallel–series system with singly derated states is shown in Figure 8 in *Example 8*.

Longer State Enumeration Approach

The parallel and series topologies merged will contain $3^4 = 81$ combinations, such as UP–UP–UP–UP, DER–UP–UP–UP all the way to DN–DN–DN–DN. This method is cumbersome and time consuming to distinguish the desirable states by enumerating. The shortcut technique is faster.

Shortcut Formula Approach

$$\begin{aligned} P(\text{UP}) &= P_1(\text{UP})P_4(\text{UP})\{1-[1-P_2(\text{UP})][1-P_2(\text{UP})]\} \\ &= (0.7)(0.7)[1-(1-0.7)^2] \\ &= (0.49)(0.91) = 0.4459. \end{aligned} \tag{11}$$

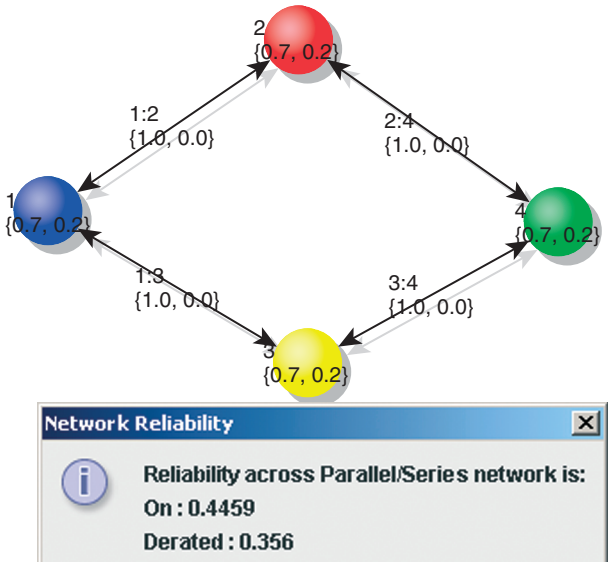


FIGURE 8 | *Example 8:* a simple parallel–series system with single derated states.

$$\begin{aligned} P_{\text{sys}}(\text{DER}) &= P_1(\text{UP}+\text{DER})P_4(\text{UP}+\text{DER}) \\ &\quad -\{1-[1-P_2(\text{UP})][1-P_3(\text{UP})]\} \\ &\quad -P_{\text{sys}}(\text{UP}) = (0.7+0.2)^2(1-0.1^2) \\ &\quad -0.4459 = (0.81)(0.99)-0.4459 \\ &= 0.356. \end{aligned} \tag{12}$$

$$\begin{aligned} P_{\text{sys}}(\text{DN}) &= 1-P_{\text{sys}}(\text{UP})-P_{\text{sys}}(\text{DER}) \\ &= 1-0.4459-0.356 = 0.19. \end{aligned} \tag{13}$$

An Active Parallel System with Doubly Derated States

A simple parallel system with derated and degraded states is shown in Figure 9 in *Example 9*.

Using the shortcut formulation approach (the state enumeration method requires $S^N = 4^4 = 256$ combinations in general; here, due to 1 and 4 being full states, $4^2 = 16$, and therefore cumbersome to work with), we get, using the same logic as we

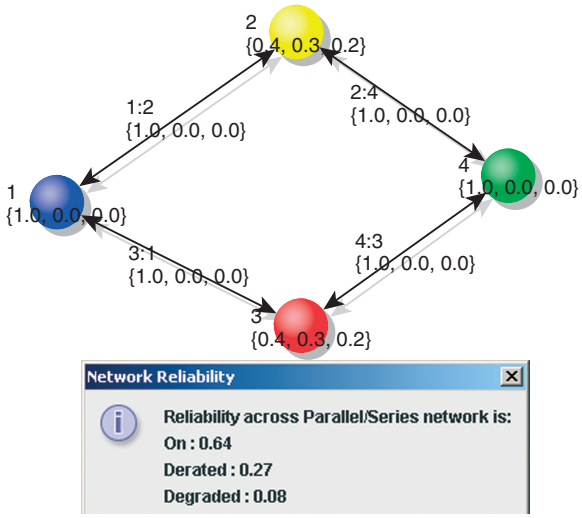


FIGURE 9 | *Example 9:* active parallel–series system with doubly derated states for 2 and 3.

used earlier.

$$\begin{aligned} P(\text{UP}) &= P_1(\text{UP})P_4(\text{UP})\{1-[1-P_2(\text{UP})] \\ &\quad \times [1-P_3(\text{UP})]\} = (1.0) \\ &\quad \times [1-(1-0.4)^2] \\ &= 1-0.36 = 0.64. \end{aligned} \tag{14}$$

$$\begin{aligned} P_{\text{sys}}(\text{DER}) &= (1.0)\{1-[1-P_2(\text{UP}+\text{DER})] \\ &\quad \times [1-P_3(\text{UP}+\text{DER})]\} - P_{\text{sys}}(\text{UP}) \\ &= (1.0)(1-0.3^2) - 0.64 \\ &= 0.91 - 0.64 = 0.27. \end{aligned} \tag{15}$$

$$\begin{aligned} P_{\text{sys}}(\text{DEGR}) &= (1.0)^*\{1-[1-P_2(\text{UP}+\text{DER} \\ &\quad +\text{DEGR})][1-P_3(\text{UP}+\text{DER} \\ &\quad +\text{DEGR})]\} - P_{\text{sys}}(\text{UP}) - P_{\text{sys}}(\text{DER}) \\ &= (1.0)(1-0.1^2) - 0.64 - 0.27 \\ &= 0.99 - 0.64 - 0.27 = 0.08. \end{aligned} \tag{16}$$

$$\begin{aligned} P_{\text{sys}}(\text{DN}) &= 1 - P_{\text{sys}}(\text{UP}) - P_{\text{sys}}(\text{DER}) \\ &\quad - P_{\text{sys}}(\text{DEGR}) \\ &= 1 - 0.64 - 0.27 - 0.08 = 0.01. \end{aligned} \tag{17}$$

A Combined System Example with Multiple Derated States

A hydroelectric power plant in Figure 10 can generate 100% (fully operating), 75% (derated 1), 50% (derated 2), 25% (derated 3) or 0% (fully down) of rated electric power capacity depending on the water storage level and thus the amount of steam flow reaching the turbine.[15,28] The corresponding system states are 1, 2, 3, 4, and 5. The power plant consists of four turbines in active parallel and an output transformer in series with the turbines. The available turbine with maximal power output is always used. For any demand level of $w = 1, 2, 3, 4, 5$, the combined system reliability function of states takes the following recursive form in *Example 10* of Figure 1.

$$\begin{aligned} R_{\text{sys}}(w) &= \left(\sum_{j=1}^{w} R_{5_j}\right)\left[1-\left(1-\sum_{j=1}^{w} R_{1_j}\right)\right. \\ &\quad \times \left(1-\sum_{j=1}^{w} R_{2_j}\right)\left(1-\sum_{j=1}^{w} R_{3_j}\right) \\ &\quad \left.\times \left(1-\sum_{j=1}^{w} R_{4_j}\right)\right] - \sum_{j=1}^{w} R_{\text{sys}}(j-1). \end{aligned} \tag{18}$$

where $w = 1, 2, 3, 4, 5$ and $R_{\text{sys}}(0) = 0.0$

MSS elements are statistically independent. If $R_{w1} = 0.4, R_{w2} = 0.3, R_{w3} = 0.15, R_{w4} = 0.1$, and $R_{w5} = 0.05$, $w(\text{state}) = 1,\ldots,5$ as shown in Figure 10, then the system reliabilities $R_{\text{sys}}(w)$ are:

$$R_{\text{sys}}(1) = 0.4[1-(1-0.4)^4] = 0.34816. \tag{19}$$

$$\begin{aligned} R_{\text{sys}}(2) &= (0.4+0.3)\{1-[1-(0.4+0.3)]^4\} - R(1) \\ &= 0.69433 - 0.34816 = 0.34617. \end{aligned} \tag{20}$$

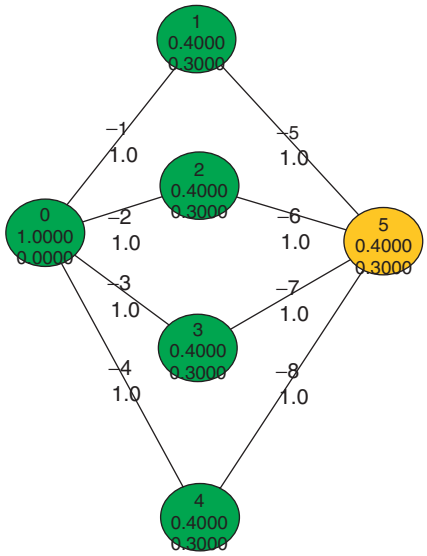
$$\begin{aligned} R_{\text{sys}}(3) &= (0.4+0.3+0.15) \\ &\quad \times \{1-[1-(0.4+0.3+0.15)]^4\} \\ &\quad - R(1) - R(2) = 0.84957 - 0.34617 \\ &\quad - 0.34816 = 0.15524. \end{aligned} \tag{21}$$

$$\begin{aligned} R_{\text{sys}}(4) &= (0.4+0.3+0.15+0.1) \\ &\quad \times \{1-[1-(0.4+0.3+0.15+0.1)]^4\} \\ &\quad - R(1) - R(2) - R(3) \\ &= 0.94999 - 0.15524 - 0.34617 - 0.34816 \\ &= 0.10042. \end{aligned} \tag{22}$$

$$\begin{aligned} R_{\text{sys}}(5) &= (0.4+0.3+0.15+0.1+0.05) \\ &\quad \times \{1-[1-(0.4+0.3+0.15+0.1 \\ &\quad +0.05)]^4\} - R(1) - R(2) - R(3) - R(4) \\ &= 1 - R(1) - R(2) - R(3) - R(4) \\ &= 1 - 0.10042 - 0.15524 - 0.34617 \\ &\quad - 0.34816 = 0.05001. \end{aligned} \tag{23}$$

Large Network Examples Using Multistate Overlap Technique

Using Figure 4 for the 19-node large cyber network, we can calculate the ingress–egress overlap reliability using multistate (up = 0.7, derated = 0.2, down = 0.1) as illustrated in Figure 11 for *Example 11*. Similarly



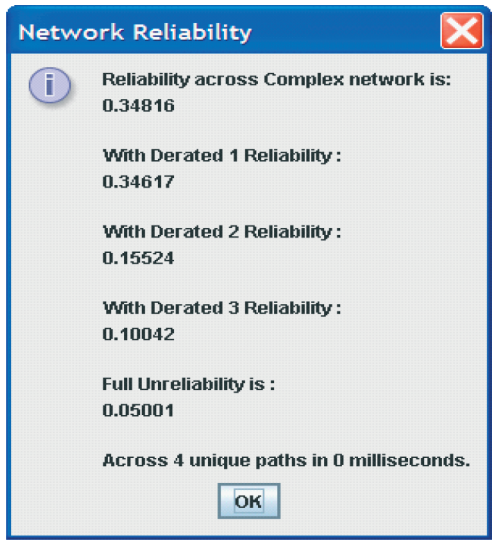
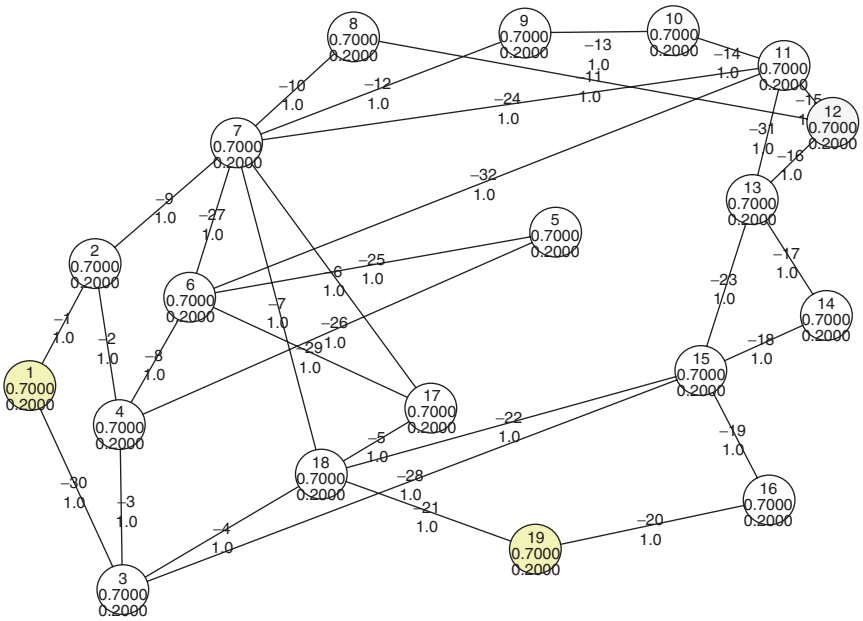


FIGURE 10 | *Example 10:* power plant with four multiple derated turbines (nodes 1 to 4) in parallel and a transformer (egress node 5) in series, and node 0 (as an ingress dummy node with full reliability).



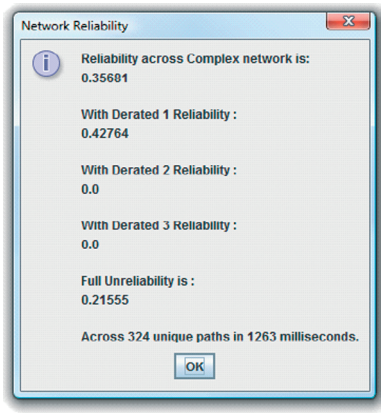


FIGURE 11 | *Example 11:* implementation of multistate overlap reliability to 19-node network in 1.26 s. Solution ($s = 1$, $t = 19$): full reliability = 0.36, derated reliability = 0.43, full unreliability = 0.21.

using Figure 5 data, we can calculate multistate reliability in Figure 12 for *Example 12*.

WEIBULL RELIABILITY EVALUATION

Motivation

When modeling real-life networks, industry engineers have quickly found that static reliabilities do not accurately depict component behaviors. For instance, a 10-year-old fatigued component is much more likely to fail than a brand new component recently commissioned. Therefore, it is necessary to implement a time-dependent reliability paradigm that can systematically represent a wide range of failure conduct. There are already several industry standard methods capable of modeling sample component heuristics or historical failure data including normal, lognormal, and Weibull.

We have chosen Weibull distribution due to the fact that it is currently one of the most widely utilized methods in practical applications and because of the versatile range of characteristics it can model from infancy to useful life and wear-out periods. The benefit of choosing Weibull distribution is that we can implement the generic mathematical behavior once and

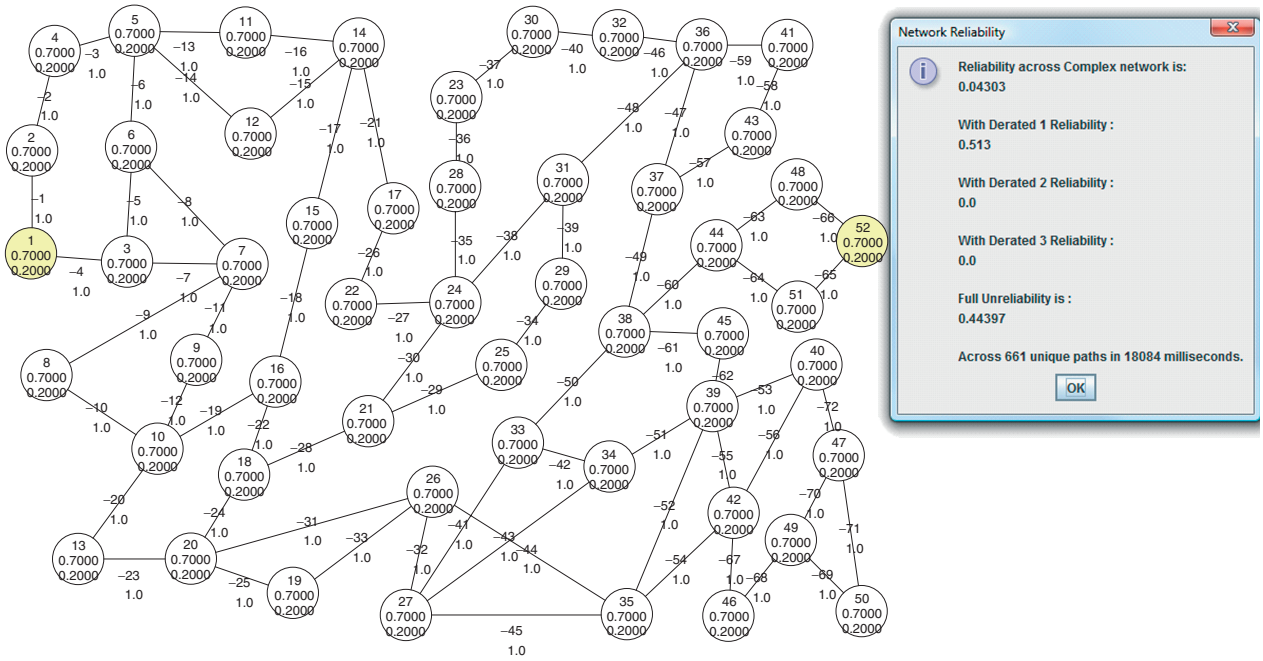


FIGURE 12 | *Example 12:* implementation of multistate overlap reliability to 52-node network in 18 s. Solution ($s = 1$, $t = 52$): full reliability = 0.05, derated reliability = 0.51, full unreliability = 0.44.

satisfy a wide range of component behavior by simply manipulating the shape parameter, β, at runtime. As shown in the subsequent graph of Figure 13, the entirety of component behavior is spanned by β.

In recent years, the Weibull distribution has become more popular as a reliability function. It is named after the Swedish scientist Waloddi Weibull, who used it in analysis of the breaking strength of solids. A chief advantage of the Weibull distribution is that as in the bathtub curve its hazard rate function may be decreasing for $\beta < 1$, constant for $\beta = 1$, or increasing for $\beta > 1$. When $\beta = 2$, the Weibull is called the Rayleigh pdf. The Weibull density and reliability functions are, respectively,

$$f(t) = \frac{\beta t^{\beta-1}}{\alpha^{\beta}} e^{-(t/\alpha)^{\beta}},\ t \geqslant 0,\ \alpha,\ \beta > 0. \qquad (24)$$

$$R(t) = e^{-(t/\alpha)^{\beta}},\ t \geqslant 0,\ \alpha,\ \beta > 0. \qquad (25)$$

The Weibull family of distributions is a member of the family of extreme value distributions. The Weibull distribution is probably the most widely used family of failure (e.g., electronic component, mechanical fatigue, etc.) distributions, mainly because by proper choice of its shape parameter β, it can be used as an IFR (increasing failure rate), DFR (decreasing failure rate), or CFR (constant failure rate, as in the negative exponential case). Often, a third parameter, known as the threshold or location parameter, t_0 is added to obtain a three-parameter Weibull, where

$$R(t) = e^{-(t-t_0/\alpha)^{\beta}},\ t \geqslant t_0 > 0,\ \alpha,\ \beta > 0. \qquad (26)$$

As the graph depicts, when $0 < \beta < 1$, the reliability distribution function represents a failure rate that declines over time. The range of β (shape value) depicts component status, where new units at infancy fail early when $\beta < 1$, and the reliability of the average unit steadies out at $\beta = 1$, and the defective components wear out of the sample when $\beta > 1$. That is, $\beta = 1$ denotes a constant failure rate. The failure of constant rate components is generally caused by random events. As β increases from 1, the component fails at an increasing rate over time. These are components whose useful lives are limited by environmental stress in the system. The longer a component is in use, the more likely it is to fail.

Weibull Methodology

Inside the overlap algorithm, there are three types of arithmetic performed on node reliabilities; multiplication, addition, and subtraction. In order to incorporate Weibull reliability directly into the algorithm, we must first find a way to define and implement Weibull math. Unfortunately, because of the nature of the survival (reliability) distribution function, this is not possible. Multiplication is only possible when both components being multiplied share the same

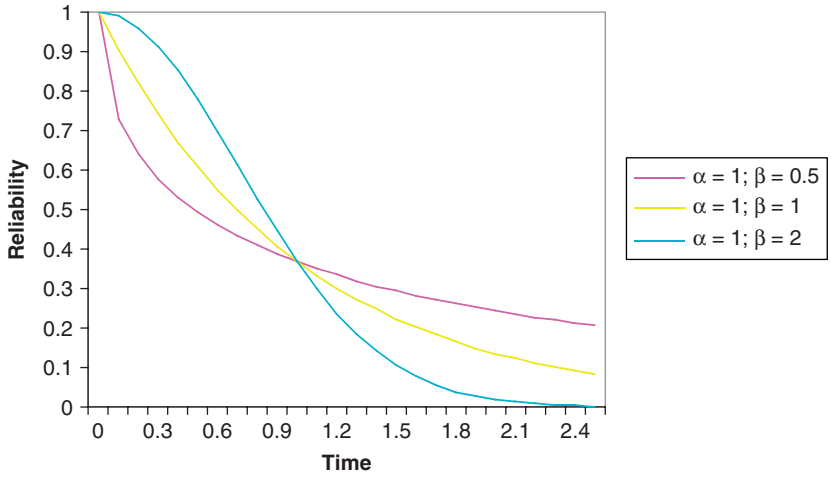


FIGURE 13 | Weibull reliability distributions.

shape parameter. When multiplying heterogeneous components, the resultant will not simplify back into cumulative distribution function (cdf) form such as that of a reliability distribution. To compound matters, addition and subtraction will definitely not result in a standard cdf regardless of whether the nodes are homogenous or heterogeneous. Consider the fact that at $t = 0$ a two-parameter Weibull reliability should always be 1; if we were to add two components, the value of $t = 0$ would be $1 + 1$ or 2, directly conflicting with the reliability bounds.

Unfortunately, as we have seen in the past few examples, there is no way of supporting Weibull math in systems containing hybrid shape parameters without incurring a quantitatively large margin of error. The solution lies in breaking the reliability of each component into an array of static reliabilities at evenly distributed points in time. The overlap algorithm is run generating a multitude of reliabilities. Finally, the reliability plot is used to generate a final Weibull goodness of fit.

Overlap Algorithm Applied to Weibull Distributed Components

Before delving into how to estimate a final pair of shape and scale parameters from the solution generated by the overlap algorithm, let us first walk through an example of a slightly more complex network up to this point. Consider the following Figure 14 network[15] from node 1 to node 19 in which each node has both β and α equal to 1 assumed. See *Example 13* in Figure 14 tabulated in Table 3.

Estimating Weibul Parameters for Nineteen-Node Example

Now that we have generated a graphical result, we can use basic axis scale regression and linear least squares to approximate the values for α and β. The goal is to adjust the scale of the x- and y-axis until the result is a straight line with a slope of m. Linear least squares will then be used to determine the best fit line through the linear plot points to return the most correct value for m.

TABLE 3 | Nineteen-Node Weibull Overlap Results

Time	Reliability	Time	Reliability	Time	Reliability	Time	Reliability
0.0	1	1	0.02736	2.0	0.00041	3	0.00001
0.1	0.79467	1.1	0.01801	2.1	0.00027	3.1	0
0.2	0.59842	1.2	0.01184	2.2	0.00018		
0.3	0.43235	1.3	0.00778	2.3	0.00012		
0.4	0.30315	1.4	0.00511	2.4	0.00008		
0.5	0.20812	1.5	0.00336	2.5	0.00005		
0.6	0.14079	1.6	0.00221	2.6	0.00003		
0.7	0.09427	1.7	0.00145	2.7	0.00002		
0.8	0.06268	1.8	0.00096	2.8	0.00002		
0.9	0.04148	1.9	0.00063	2.9	0.00001		

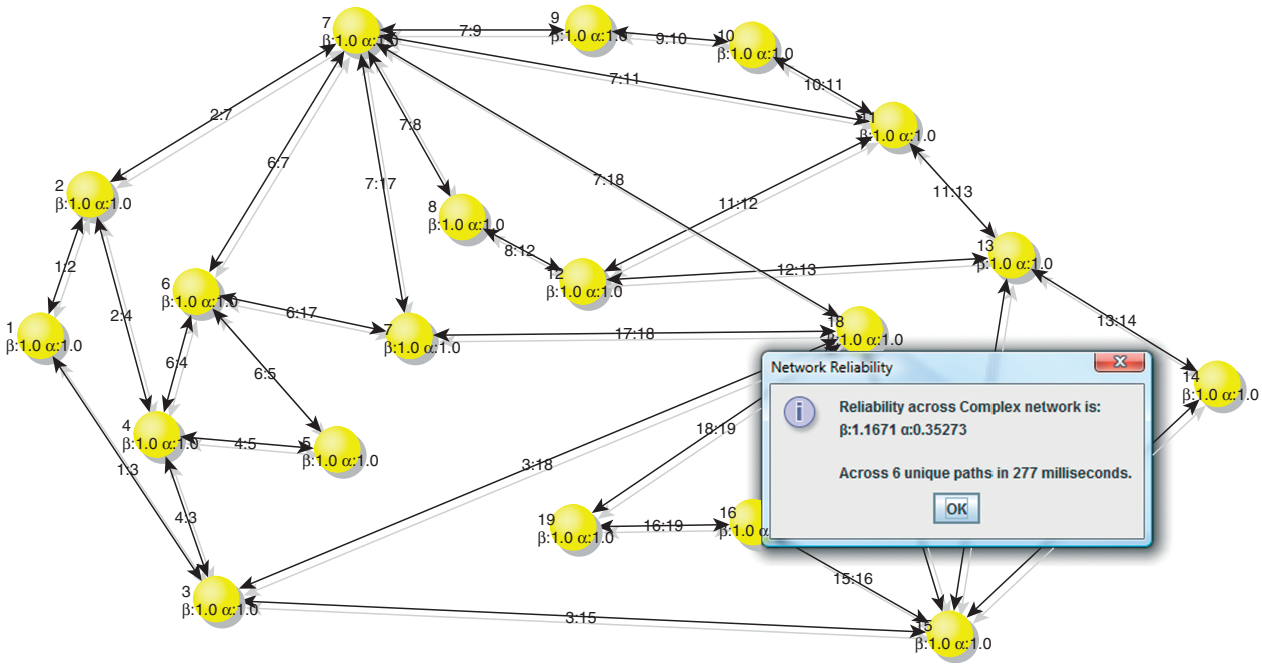


FIGURE 14 | *Example 13:* 19-node Weibull network.

Recall that the natural logarithm of the reliability function for a Weibull plot by using Eq. (25) on page 14 is:

$$-\ln(r) = (t/\alpha)^{\beta}. \tag{27}$$

Graphing the above equation depicts an exponential curve with a power of β (=slope) as in Figure 15.

To determine the value of β, take the log of both axis and use linear least squares to determine the slope. See Figure 16.

The x-coordinates are given by $\log(t)$, whereas the y-coordinates are $\log[-\ln(r)]$ in Table 4 as depicted in Figure 16.

The best line slope function for calculating first-degree polynomials for n number of x and y plot points are given in Eqs. (28) and (29), respectively, as follow:

$$m = \frac{(\Sigma y)(\Sigma x) - n(\Sigma xy)}{(\Sigma x)^2 - n(\Sigma x^2)}. \tag{28}$$

Substituting the x and y plot functions from above yields the slope to attain the least square estimation (LSE):

$$\beta = \frac{\{\Sigma \log[-\ln(r)]\}[\Sigma \log(t)] - n\{\Sigma \log(t) \log[-\ln(r)]\}}{[\Sigma \log(t)]^2 - n[\Sigma \log(t)^2]}. \tag{29}$$

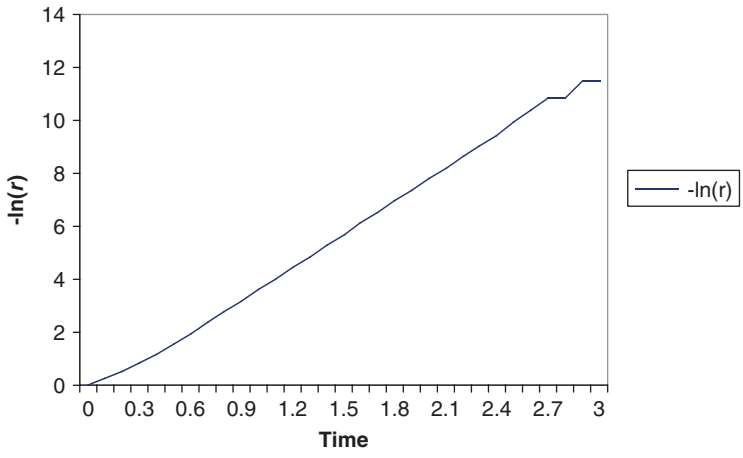


FIGURE 15 | Exponential graph for β.

TABLE 4 | Nineteen-Node Linear Data for β with Least Squares Calculations

t	r	$\log(t)$	$\log[-\ln(r)]$	$\log(t)^2$	$\log(t)\log[-\ln(r)]$
0	1	–	–	–	–
0.1	0.79467	–1	−0.638596411	1	0.638596411
0.2	0.59842	−0.698970004	−0.289491328	0.488559067	0.202345755
0.3	0.43235	−0.522878745	−0.076486661	0.273402182	0.039993249
0.4	0.30315	−0.397940009	0.076832447	0.158356251	−0.030574705
0.5	0.20812	−0.301029996	0.19580018	0.090619058	−0.058941727
0.6	0.14079	−0.22184875	0.292363714	0.049216868	−0.064860524
0.7	0.09427	−0.15490196	0.373204919	0.023994617	−0.057810173
0.8	0.06268	−0.096910013	0.442434748	0.009391551	−0.042876357
0.9	0.04148	−0.045757491	0.502774402	0.002093748	−0.023005695
1	0.02736	0	0.556142408	0	0
1.1	0.01801	0.041392685	0.603883249	0.001713354	0.024996349
1.2	0.01184	0.079181246	0.647018132	0.00626967	0.051231702
1.3	0.00778	0.113943352	0.68629647	0.012983088	0.07819892
1.4	0.00511	0.146128036	0.722350541	0.021353403	0.105555666
1.5	0.00336	0.176091259	0.755555822	0.031008132	0.133046776
1.6	0.00221	0.204119983	0.786379612	0.041664967	0.160515793
1.7	0.00145	0.230448921	0.815324783	0.053106705	0.187890717
1.8	0.00096	0.255272505	0.841895892	0.065164052	0.214912873
1.9	0.00063	0.278753601	0.867455156	0.07770357	0.241806249
2	0.00041	0.301029996	0.892058599	0.090619058	0.268536396
2.1	0.00027	0.322219295	0.91471797	0.103825274	0.294739779
2.2	0.00018	0.342422681	0.935635908	0.117253292	0.320382956
2.3	0.00012	0.361727836	0.955592456	0.130847027	0.345664391
2.4	0.00008	0.380211242	0.974672114	0.144560588	0.370581295
2.5	0.00005	0.397940009	0.99578816	0.158356251	0.396263949
2.6	0.00003	0.414973348	1.017630634	0.17220288	0.422289591
2.7	0.00002	0.431363764	1.034218361	0.186074697	0.446124325
2.8	0.00002	0.447158031	1.034218361	0.199950305	0.462459046
2.9	0.00001	0.462397998	1.061185693	0.213811908	0.49069014
3	0.00001	0.477121255	1.061185693	0.227644692	0.506314249
	$\sum$	2.423660075	19.03804202	4.151746254	6.125067395

Extrapolate the values in the plot table to quickly find the various sums in Table 4.

The number of plot points, n, is 30. The first and last points are omitted as the values of 0 for time and reliability would cause the log functions to return $-\infty$. Solve for β and α[11,27] in Eqs. (30) and (31) as follows, respectively:

$$\beta = \frac{\{\Sigma \log[-\ln(r)]\}[\Sigma \log(t)] - n\{\Sigma \log(t) \log[-\ln(r)]\}}{[\Sigma \log(t)]^2 - n[\Sigma \log(t)^2]}$$

$$\beta = \frac{19.03804202 \times 2.423660075 - 30 \times 6.125067395}{2.423660075^2 - 30 \times 4.151746254}$$

$$\beta = 1.16. \tag{30}$$

Use the value of β to estimate for α. Recall the exponential equation for—$\ln(r)$:

$$-\ln(r) = (t/\alpha)^{\beta}.$$

Solving for α produces:

$$-\ln(r)^{1/\beta} = t/\alpha$$

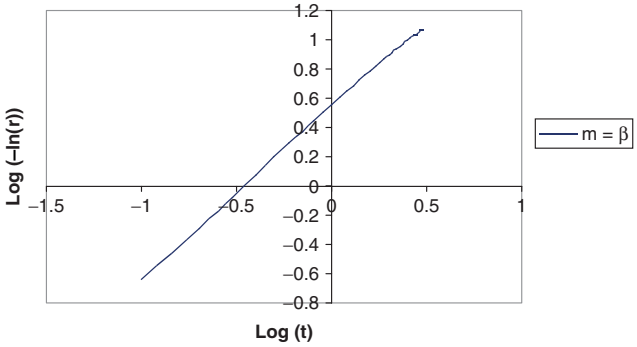


FIGURE 16 | Linear graph for β.

$$\alpha = t/-\ln(r)^{1/\beta}$$
$$\alpha = t/-\ln(r)^{1/1.159524}$$
$$\alpha = t/-\ln(r)^{0.862423}. \quad (31)$$

Graphing t versus $-\ln(r)^{0.862423}$ produces a regression line, the slope of which is $1/\alpha$. Again, we use the linear LSE principle to approximate the value for α. As the slope of the line is the inverse of the scale parameter, the inverse of the least squares slope formula is used for result interpretation as in Table 5.

$$1/m = \frac{(\Sigma x)^2 - n(\Sigma x^2)}{(\Sigma y)(\Sigma x) - n(\Sigma xy)}.$$

Use the equation obtained above to solve for α:

$$\alpha = \frac{(\Sigma t)^2 - n(\Sigma t^2)}{[\Sigma - \ln(r)^{1/\beta}](\Sigma t) - n\{\Sigma t[-\ln(r)^{1/\beta}]\}}$$
$$\alpha = \frac{(46.5)^2 - 30 \times 94.55}{135.4755627 \times 46.5 - 30 \times 273.4606493}$$
$$\alpha = 0.354085. \quad (32)$$

To verify our results and determine the margin of error, we can chart and graph our original plot points versus the evaluation of the determinate function at each plot point in time. The difference between the generated LSE results and the observed is placed in the final column titled Δ (delta) as in Table 6.

The sum of the difference is 0.124193; a small margin of error as can be seen in the plots of the two graphs. The sum of the error squares will be even more negligible and miniscule (=0.00168). Note the x-axis (time) has been reduced to a maximum of 1.6. This is the point at which the plotted reliability function $f(t)$ becomes close enough to the axis to appear as zero and help to better illustrate the similarity of the two lines. See Figure 17.

To sum up the past series of calculations, once a set of plot points has been generated by the overlap algorithm for a Weibull network, the two equations for α and β are used in succession to estimate the shape and scale parameters, respectively, of the resulting Weibull solution. See Figure 18 for Table 3.

Fifty-Two Node Weibull Example for Estimating Weibull Parameters

Let us try the overlap algorithm for Weibull networks again, as in Figure 19, this time for a substantially more complicated network. In this 52-node 72-link network,[15] we will again use unity for both the shape and scale parameters as we generate the reliability block diagram from node 1 to node 52. See *Example 14* in Figure 19.

For each plot point, calculate the values for $\log(t)$, $\log[-\ln(r)]$, $\log(t)^2$, and $\log(t)\log[-\ln(r)]$ as in Table 7.

Note that the more points, the more accurate the plot will be, but more microprocessor runtime will be observed.

Find the sums of each of the calculated columns and apply the values to the equation for the shape parameter in Table 8. Given the new-found value for β in Eq. (33), compute the values required to estimate the scale parameter α in Eq. (34). Calculate $-\ln(r)^{1/\beta}$, t^2, and $t[-\ln(r)^{1/\beta}]$ and their corresponding sums across the plot as shown in Tables 9 and 10.

$$\beta = \frac{\{\Sigma \log[-\ln(r)]\}[\Sigma \log(t)] - n\{\Sigma \log(t) \log[-\ln(r)]\}}{[\Sigma \log(t)]^2 - n[\Sigma \log(t)^2]}$$
$$\beta = \frac{6.299380256 \times -3.440236967 - 10 \times -0.939435447}{-3.440236967^2 - 10 \times 2.095633341}$$
$$\beta = 1.370000185. \quad (33)$$
$$\alpha = \frac{(\Sigma t)^2 - n(\Sigma t^2)}{[\Sigma - \ln(r)^{1/\beta}](\Sigma t) - n\{\Sigma t[-\ln(r)^{1/\beta}]\}}$$
$$\alpha = \frac{5.5^2 - 10 \times 3.85}{24.79955109 \times 5.5 - 10 \times 24.79955109}$$
$$\alpha = 0.1558046056966797. \quad (34)$$

TABLE 5 | Nineteen-Node Linear Data for α with Least Squares Calculations

r	t	$-\ln(r)^{1/\beta}$	t^2	$t[-\ln(r)^{1/\beta}]$
1	0	0	0	0
0.79467	0.1	0.281358356	0.01	0.028135836
0.59842	0.2	0.562776615	0.04	0.112555323
0.43235	0.3	0.859085029	0.09	0.257725509
0.30315	0.4	1.164828702	0.16	0.465931481
0.20812	0.5	1.475239606	0.25	0.737619803
0.14079	0.6	1.787068374	0.36	1.072241025
0.09427	0.7	2.098265681	0.49	1.468785977
0.06268	0.8	2.407497631	0.64	1.925998105
0.04148	0.9	2.713964545	0.81	2.442568091
0.02736	1	3.017379678	1	3.017379678
0.01801	1.1	3.317437553	1.21	3.649181308
0.01184	1.2	3.614125826	1.44	4.336950991
0.00778	1.3	3.907309449	1.69	5.079502284
0.00511	1.4	4.197315986	1.96	5.87624238
0.00336	1.5	4.48341211	2.25	6.725118165
0.00221	1.6	4.766414464	2.56	7.626263143
0.00145	1.7	5.048411906	2.89	8.58230024
0.00096	1.8	5.321944188	3.24	9.579499538
0.00063	1.9	5.599035378	3.61	10.63816722
0.00041	2	5.879383438	4	11.75876688
0.00027	2.1	6.149980591	4.41	12.91495924
0.00018	2.2	6.41082401	4.84	14.10381282
0.00012	2.3	6.669984716	5.29	15.34096485
0.00008	2.4	6.927548473	5.76	16.62611633
0.00005	2.5	7.224212891	6.25	18.06053223
0.00003	2.6	7.544456964	6.76	19.61558811
0.00002	2.7	7.797109302	7.29	21.05219512
0.00002	2.8	7.797109302	7.84	21.83190605
0.00001	2.9	8.226040953	8.41	23.85551876
0.00001	3	8.226040953	9	24.67812286
$\sum$	46.5	135.4755627	94.55	273.4606493

The final result for the 52-node Weibull system as in Figure 19 and Table 11 will be given in Eq. (35) and plotted in Figure 20.

$$R(t) = e^{-(t/0.155)^{1.37}},\ t \geqslant 0,\ \alpha,\ \beta > 0. \qquad (35)$$

The sum of Δ (difference errors) is 0.02849, which is satisfactorily negligible for a large complex network of 52 nodes and 72 links which would normally take years if calculated manually. The squares sum of errors is even smaller to be 0.00085 as in Table 11.

TABLE 6 | Nineteen-Node Weibull Results Comparison

t	r (observed)	$r = e^{-(t/0.35)^{1.16}}$	Δ (difference)
0	1	1	0
0.1	0.79467	0.793872314	0.000797686
0.2	0.59842	0.597116825	0.001303175
0.3	0.43235	0.438171073	0.005821073
0.4	0.30315	0.316047967	0.012897967
0.5	0.20812	0.224921716	0.016801716
0.6	0.14079	0.158303476	0.017513476
0.7	0.09427	0.1103625	0.0160925
0.8	0.06268	0.076300916	0.013620916
0.9	0.04148	0.052360252	0.010880252
1	0.02736	0.03568977	0.00832977
1.1	0.01801	0.024177139	0.006167139
1.2	0.01184	0.016285165	0.004445165
1.3	0.00778	0.010911397	0.003131397
1.4	0.00511	0.007274771	0.002164771
1.5	0.00336	0.004827687	0.001467687
1.6	0.00221	0.003189728	0.000979728
1.7	0.00145	0.002098764	0.000648764
1.8	0.00096	0.001375494	0.000415494
1.9	0.00063	0.000898089	0.000268089
2	0.00041	0.000584277	0.000174277
2.1	0.00027	0.000378811	0.000108811
2.2	0.00018	0.000244788	6.4788E-05
2.3	0.00012	0.00015768	3.76803E-05
2.4	0.00008	0.000101259	2.12594E-05
2.5	0.00005	6.4835E-05	1.4835E-05
2.6	0.00003	4.13947E-05	1.13947E-05
2.7	0.00002	2.6356E-05	6.35602E-06
2.8	0.00002	1.6736E-05	3.26395E-06
2.9	0.00001	1.05998E-05	5.99833E-07
3	0.00001	6.69654E-06	3.30346E-06

DISCUSSION AND CONCLUSIONS

This network reliability evaluation algorithm may open many doors in the vast reliability engineering field. Of the currently available free resources, most if not all impose strict component or topology limits such as pure series–parallel. The rest of the methods are only commercially available and no one knows, unless a client purchases it, about what needs to be done, and how and why?[29] As we have seen in the performance characteristics of complex cyber network reliability generation, because of the hardware and time resources required to perform the algorithmic

TABLE 7 | Fifty-Two Node Linear Data for β with Least Squares Calculations

t	r	log(t)	log[–ln(r)]	log(t)^2	log(t)log[–ln(r)]
0.0	1	—	—	—	—
0.1	0.577760	−1	−0.260746792	1	0.260746792
0.2	0.247430	−0.698970004	0.145080609	0.488559067	−0.101406994
0.3	0.084700	−0.522878745	0.392457705	0.273402182	−0.205207792
0.4	0.025000	−0.397940009	0.566894464	0.158356251	−0.225589988
0.5	0.006710	−0.301029996	0.699330868	0.090619058	−0.210519568
0.6	0.001700	−0.22184875	0.804625068	0.049216868	−0.178505065
0.7	0.000410	−0.15490196	0.892058599	0.023994617	−0.138181625
0.8	0.000100	−0.096910013	0.96427568	0.009391551	−0.093447969
0.9	0.000020	−0.045757491	1.034218361	0.002093748	−0.047323237
1.0	0.000010	0	1.061185693	0	0
1.1	0				

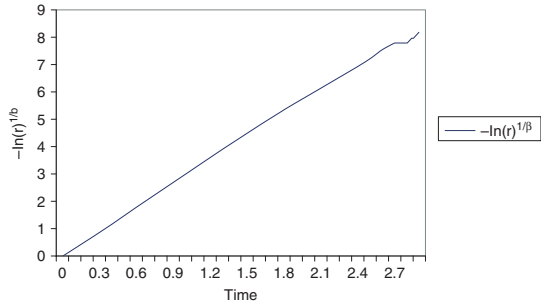


FIGURE 17 | Linear graph for 1/α.

TABLE 8 | Fifty-Two Node Least Squares Sum for β

Linear least squares sums for β	
∑log(t)	−3.440236967
∑log[–ln(r)]	6.299380256
∑log(t)^2	2.095633341
∑log(t)log[–ln(r)]	−0.939435447
Number of plot points	10

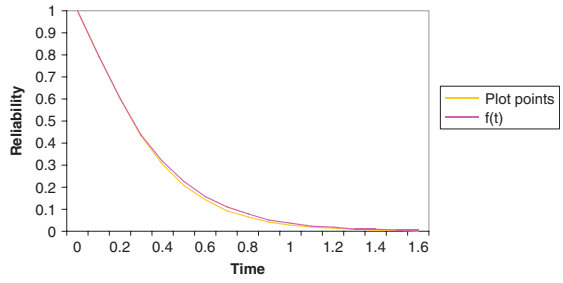


FIGURE 18 | Nineteen-node Weibull results for graphical comparison.

TABLE 9 | Fifty-Two Node Linear Data for α with Least Squares Calculations

T	r	–ln(r)$^{1/\beta}$	t^2	t[–ln(r)$^{1/\beta}$]
0.0	1	—	—	—
0.1	0.577760	0.640147819	0.01	0.064014782
0.2	0.247430	1.281700255	0.04	0.256340051
0.3	0.084700	1.956920992	0.09	0.587076298
0.4	0.025000	2.637361592	0.16	1.054944637
0.5	0.006710	3.307975147	0.25	1.653987573
0.6	0.001700	3.960862039	0.36	2.376517223
0.7	0.000410	4.599893257	0.49	3.21992528
0.8	0.000100	5.204761729	0.64	4.163809383
0.9	0.000020	5.866299131	0.81	5.279669218
1.0	0.000010	6.14326665	1	6.14326665

operations; newer algorithms other than for those commercially marketed have not been studied at large. The overlap algorithm presented in this article, along with the associated performance tuning activities, greatly increases the size and complexity of the networks that are possible.[14,15,27] This expansion of capability moves the presented set of algorithms from a tool that can only be used in demonstrations and educational activities to an enterprise-level solution than can be used to capably solve real-world scenarios. In addition to static (or constant) node reliability, the inclusion of both multistate and Weibull time-dependent paradigms in this research paper increase the algorithm's practical applicability. These two additional reliability features build on the power

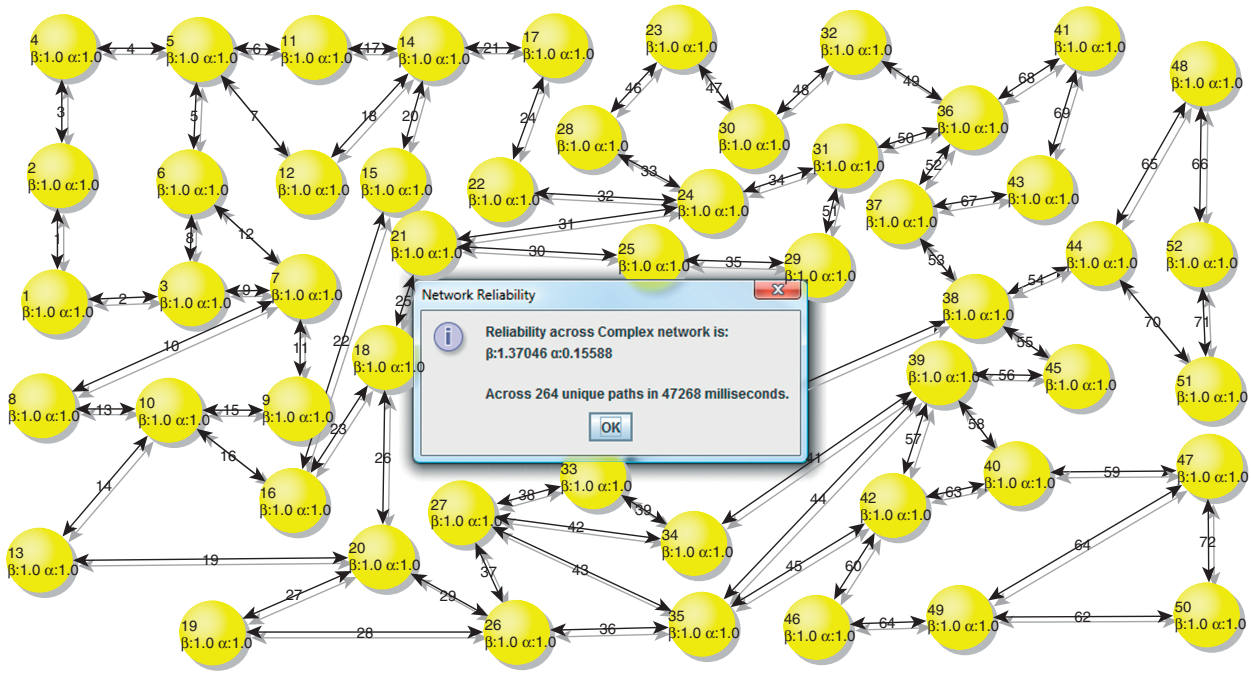


FIGURE 19 | *Example 14:* 52-node Weibull network, $s = 1, t = 52$.

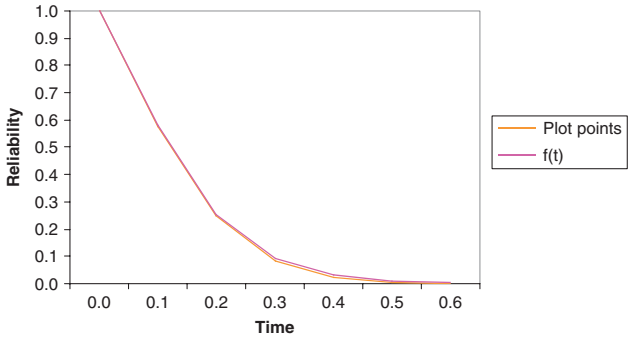


FIGURE 20 | Fifty-two node Weibull results graphical comparison.

TABLE 10 | Fifty-Two Node Least Squares Sum for the Intercept α

Linear least squares for α	
$\sum t$	5.5
$\sum -\ln(r)^{1/\beta}$	35.59918861
$\sum t^2$	3.85
$\sum t[-\ln(r)^{1/\beta}]$	24.79955109
Number of plot points	10
α	0.1558046057

TABLE 11 | Fifty-Two Node Weibull Results Comparison

t	r (observed)	$r = e^{-(t/0.155)^{1.37}}$	Δ(difference)
0.0	1	1	0
0.1	0.577760	0.58272	0.004956812
0.2	0.247430	0.25338	0.005953392
0.3	0.084700	0.09353	0.008834758
0.4	0.025000	0.0305	0.005504651
0.5	0.006710	0.00898	0.002272338
0.6	0.001700	0.00242	0.000721854
0.7	0.000410	0.0006	0.000193922
0.8	0.000100	0.00014	0.0000403337
0.9	0.000020	0.000031	0.0000010568
1.0	0.000010	0.0000063	0.0000037283
1.1	0	0	0

provided by the overlap algorithms to allow modeling of real-world complex network behavior. Multistate networks can be used by engineers to analyze networks in which components can exhibit varying levels of operational effectiveness. In applications such as monitoring a packet-switching network,

communications professionals can forecast and plan for situations where traffic controlling hubs may experience heavy system loads and are operated in a derated or degraded state.

The work illustrated in this article on Weibull reliability distributions allows technicians to model and analyze full life-cycle systems inspired from the famous bath–tub curve.[11,15,27,30] Using the various behaviors provided by the reliability function, one could model component wear-out or infancy periods and preemptively replace system components before they fail to act wisely and proactively rather than retroactively. The benefits listed above prove that we have documented an algorithm with a true industry application, whereas the shortcomings really only detail opportunities for growth rather than steadfast barriers. The innovative algorithms documented in this article provide a much needed and practically versatile function for both academic and industrial applications. Nondirectional reliability concerns such as in cloud computing with independent units in an additive rather than directional scenario are becoming recently popular and are out of scope in this work suitable in another setting.[31]

REFERENCES

1. Colbourn CJ. Combinatorial aspects of network reliability. *Ann Oper Res* 1981, R-30:32–35.
2. Aggarwal KK, Rai S. Reliability evaluation in computer communication networks. *IEEE Trans Rel* 1981, 30(1):32–35.
3. Jan RH. Design of reliable networks. *Comput Oper Res* 1993, 20(1):25–34.
4. Yeh MS, Lin JS, Yeh WC. *New Monte Carlo method for estimating network reliability*. *Proceedings of 16th International Conference on Computers and Industrial Engineering*, 1994, 723–726.
5. Fishman GS. A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness. *IEEE Trans Rel* 1986, 35(2):145–155.
6. Sahinoglu M, Libby D. *Sahinoglu-Libby (SL) probability density function- component reliability applications in integrated networks*. *Proceedings Annual Reliability and Maintainability Symposium*, RAMS03, Tampa, FL, January 27–30, 2004, 280–287.
7. Dengiz B, Altiparmak F, Smith AE. Efficient optimization of all-terminal reliable networks using evolutionary approach. *IEEE Trans Rel* 1997, 46(1):18–26.
8. Sahinoglu M, Libby D, Das SR. Measuring availability indices with small samples for component and network reliability using the Sahinoglu-Libby probability model. *IEEE Trans Instr Meas* 2005, 54(3):1283–1295.
9. Sahinoglu M. *Reliability index evaluations of an integrated software system for insufficient software failure and recovery data*. *Proceedings ADVIS-2000*, Springer-Verlag, Izmir, Turkey, October 2000, 25–27.
10. Woltenshome LC. *Reliability Modelling—A Statistical Approach*. Boca Raton, FL: Chapman and Hall/CRC; 1999, 106–107.
11. Trivedi KS. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. 2nd ed. New York: John Wiley and Sons, Inc.; 2002, 42–60, ISBN 0-471-33341-7.
12. Sahinoglu M, Smith A, Dengiz B. *Improved network design method when considering reliability and cost using an exact reliability block diagram calculation (ERBDC) tool in complex systems*. *ANNIE Smart Engineering Systems. Proceedings of the Intelligent Engineering Systems Through Artificial Neural Networks*, 13, St. Louis, MO, November 1–4, 2003, 849–855.
13. Sahinoglu M, Ramamoorthy C. RBD tools using compression and hybrid techniques to code, decode and compute s-t reliability in simple and complex Networks. *IEEE Trans Instr Meas, Special Guest Edition on Testing* 2005, 54(3):1789–1799.
14. Rice B. *Scalable complex cyber network reliability algorithm*. MS Thesis. Supervisor: M. Sahinoglu (AUM), Troy University Computer Science, Montgomery, May 2009.
15. Sahinoglu M. *Trustworthy Computing: Analytical and Quantitative Engineering Evaluation*. Hoboken, NJ: John Wiley and Sons; 2007.
16. Murphy KE, Carter CM. *Reliability block diagram construction techniques: Secrets to real-life diagramming woes*. *Proceedings Annual Reliability and Maintainability Symposium: Tutorial Notes RAMS03*, Tampa, FL, January 2003.
17. Luo T, Trivedi KS. An improved algorithm for coherent system reliability. *IEEE Trans Rel* 1998, 47(1):73–78.
18. Rai S, Veeraraghavan M, Trivedi KS. A survey on efficient computation of reliability using disjoint products approach. *Networks* 1995, 25(3):174–163.
19. Zang X, Sun HR, Trivedi KS. A BDD approach to dependable analysis of distributed computer systems with imperfect coverage. In: Avresky D, ed. *Dependable Network Computing*. Amsterdam: Kluwer Publications; 1999, 167–190.

20. Sahinoglu M, Larson J, Rice B. *An exact reliability calculation tool to improve large safety-critical computer networks*. *Proceedings DSN2003, IEEE Computer Society*, San Francisco, CA, B-38-39, June 22–25, 2003.

21. Sahinoglu M, Chow E. *Empirical bayesian availability index of safety and time critical software systems with corrective maintenance*. *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing (PRDC1999)*, Hong Kong, December 16–17, 1999, 84–91.

22. Sahinoglu M. *An exact RBD calculation tool to design very complex systems*. Invited Talk. *Proceedings of the First ACIS International Conference on Software Engineering Research and Applications*, San Francisco, CA, June 25–27, 2003.

23. Ramamoorthy C, Han YW. Reliability analysis of systems with concurrent error detection. *IEEE Trans Comput* 1975, 24:868–878.

24. Sahinoglu M, Munns W. *Availability indices of a software network*. *Proceedings of IX Brazilian Symposium on Fault Tolerant Computing*, Florianopolis, Brazil, March 2001, 123–131.

25. Sahinoglu M. *An algorithm to code and decode complex systems, and to compute s-t reliability*. *Proceedings Annual Reliability and Maintainability Symposium, RAMS05*, Alexandria, VA, January 24–27, 2005.

26. Sahinoglu M, Rice B, Tyson D. An analytical exact RBD method to calculate s-t reliability in complex networks. *Int J Comput Inf Technol Eng (IJCITE)* 2008, 2(2):95–104.

27. Sahinoglu M. *Solution manual to trustworthy computing: analytical and quantitative engineering evaluation*. Available at: http://www.areslimited.com. January 2008, 1–231.

28. Lisnianski A, Levitin G. *Multi-State System Reliability*. Singapore: World Scientific Pub. Co. Ltd.; 2003, ISBN 981-238-306-9.

29. Reliasoft. *BlockSim, Version 6, User's Guide*. ReliaSoft Corporation, ReliaSoft Publishing, 1–704, 1992–2004.

30. Lewis EE. *Introduction to Reliability Engineering*. 2nd ed. New York: John Wiley and Sons, Inc.; 1996.

31. Sahinoglu M, Cueva-Parra L, Tyson D, Das S. *Statistical inference and simulation on security metrics in cloud computing for large cyber systems*. *Invited Session #204120: Quantitative Security and Cybersytems, Joint Stat Meetings*, Washington DC, August 2009.