ChatGPT

Availability and Reliability Calculations using MTBF and MTTR

Steady-State Availability of a Component

Availability (A) is the fraction of time a component is operational (up) in the long run, considering it can be repaired after failures. For a repairable component with **Mean Time Between Failures (MTBF)** and **Mean Time To Repair (MTTR)**, the steady-state (inherent) availability is:

- **Formula:** $A = \frac{\text{MTBF}}{\text{MTBF}+\text{MTTR}}$. This comes from MTBF being the sum of the average **up time** (often called MTTF for mean time to failure) and the average **down time** (MTTR) [1] [2]. In other words, the proportion of each cycle that the component is operational is MTBF's up-time portion divided by the total cycle time (up + down).
- **Explanation:** If a device runs on average (MTTF) hours before failing and then takes MTTR hours to restore, over many cycles it will be operational for MTTF out of every MTBF = MTTF+MTTR hours. For example, if a server has MTBF = 1000 hours and MTTR = 10 hours, its inherent availability would be $1000/(1000+10) \approx 0.990$ (99%) uptime in steady state.
- This steady-state formula assumes a **constant failure rate and repair rate** (often modeled with an exponential time-to-failure and exponential repair). Under these assumptions, the system's availability stabilizes to a constant value as time goes to infinity [3] [2]. (Instantaneous availability may be lower early on, but approaches A as the system "settles" with repeated failures and repairs.)

Important: This calculation yields the **inherent availability** of a single component, excluding any logistic delays or preventive maintenance. It captures only the intrinsic design reliability and maintainability (failure and repair) characteristics [4]. For network components like routers or switches, this provides a baseline availability assuming whenever a failure occurs, repairs begin immediately and only corrective downtime is counted.

Deriving Reliability (R) from MTBF (and MTTR)

Reliability (R) is the probability that a system or component will perform its intended function **for a specified mission time t without failure**. In contrast to availability, reliability assumes no repair during the mission – the focus is on **survival probability** over time. To derive a reliability function from MTBF, we typically make an assumption about the failure distribution:

- **Exponential Failure Model (Constant Hazard Rate):** The most common model assumes a constant failure rate λ, which corresponds to an exponential time-to-failure distribution. Under this model, the reliability is:
  $$R(t) = e^{-\lambda t},$$
  with failure rate λ = 1/MTBF [5]. Substituting λ, we get:

$$Reliability formula$$

: $R(t) = \exp!\big(-t/\text{MTBF}\big)$ [6] [7].
This formula gives the probability that the component survives time t without failing, given its MTBF. For example, a component with MTBF = 100 hours has $R(100) = e^{-100/100} = e^{-1} \approx 0.3679$ (only ~36.8% chance to last 100 hours without failure) [8]. Longer mission times yield lower reliability (approaching zero as $t \to \infty$), since eventually a failure becomes almost certain if no repair occurs.

- **Additional Parameters:** If we **only** have MTBF (and no other distribution parameters), we are essentially assuming an exponential distribution (constant hazard). The exponential model is convenient because it needs just one parameter (MTBF or its reciprocal, λ) [9]. However, real components might not fail with a truly constant rate (think of "burn-in" early failures or wear-out later in life). To capture non-constant hazard rates, a more general distribution like **Weibull** is often used. The Weibull distribution has a shape parameter β that can model increasing or decreasing failure rates, whereas an exponential is a special case with β = 1 (constant rate) [10]. **If a non-exponential model is needed, additional parameters (like a Weibull shape factor) must be provided**, since MTBF alone isn't enough to define a reliability curve in those cases [11]. In summary: with only MTBF given, the **implied model is exponential**; to use any other reliability model, you'd need more data (e.g. a failure rate trend or distribution shape in addition to the mean).

- **Role of MTTR:** Note that MTTR does **not** directly figure into the reliability calculation for a single mission – reliability is about *no failure occurring* in time t, so if a failure did occur (even if quickly repaired) the mission is considered failed. Thus, for a non-repairable scenario or one mission without repairs, only the failure characteristics (MTTF/MTBF) matter for R(t). In practice, if you have MTBF and MTTR for a repairable component, you can derive the mean time to **first** failure (MTTF). Often **MTBF = MTTF + MTTR**, so **MTTF ≈ MTBF** if MTTR is negligible or **MTTF = MTBF – MTTR** in general. Using the MTTF (the average up-time before failure) in the exponential formula is more precise. However, for simplicity, many engineers use MTBF in $R(t) = e^{-t/MTBF}$, which is a reasonable approximation when MTTR << MTBF [1]. Just be mindful that if MTTR is a significant fraction of MTBF, the actual no-failure probability up to t might be slightly higher if you base it on MTTF (since the component is only operational during the up-time portion in each cycle).

- **Repairable vs. Non-repairable Context:** The exponential reliability derived from MTBF strictly applies to the **first failure probability** (non-repairable case or before any repair). In a **repairable system**, you might still compute the reliability of *no failures in [0, t]* (which would also follow an exponential law for the *first* failure if constant λ). But if the system is quickly repaired, one could instead ask a different question: *"what is the probability the system is operational at time t?"* – that is an **availability** question rather than classic reliability. (This distinction is discussed more below.) Generally, use reliability when you consider any failure during the mission as unacceptable, and use availability when continuous operation with possible repairs is acceptable.

Key takeaway: To compute reliability from MTBF, assume a distribution (typically exponential) to get $R(t)$. No additional input is needed beyond MTBF if exponential is assumed [9], but other models require more info. The MTTR is not used in the standard reliability formula – it comes into play for availability and for modeling what happens after a failure.

Mixing Availability and Reliability Metrics (Context Matters)

Availability and reliability are related concepts, but **they are not interchangeable – each is appropriate in different contexts**. It is important to understand when to use one, the other, or both:

- **Different Questions Answered:** Reliability $R(t)$ answers *“What is the probability my component/system will run continuously without failing for the next t hours?”* (no repairs considered in that interval). Availability A answers *“What fraction of the time is my system operational on average (or at a specific time) given that failures* may *occur and be repaired?”*. Thus, reliability is a time-bound probability of **no failure**, whereas availability is an overall probability **to be in working state** at a given time (allowing that there could have been failures earlier, as long as they’re fixed by that time) [12].

- **Repairable vs. Non-Repairable Systems:** If a system **cannot be repaired during operation** (non-repairable or one-shot system, like a spacecraft during a mission), then availability over the mission is the same as reliability – if it fails at any point, it’s down for the rest of the mission. In fact, *for a system without repair, $A(t) = R(t)$* [13], since being “available at time t” is equivalent to “having never failed up to time t.” Conversely, in a **repairable system** running indefinitely, eventually failures will occur, so $R(t \to \infty) = 0$ while steady-state availability stays at some non-zero value. For example, a network router that can be rebooted or repaired after failures might have 99.9% availability long-term, even though given enough time it will certainly have experienced failures (zero reliability at infinite time). Thus, **reliability and availability are context-dependent metrics**: reliability is appropriate for short-term, no-repair scenarios (or the first-failure probability of a device), whereas availability is appropriate for long-term, sustained operations with maintenance.

- **Using Both Metrics Side by Side:** It can be valid to consider both metrics **for different purposes**. For instance, you might want to know that a firewall has 95% reliability for a 24-hour unmanned mission (chance it doesn’t crash in 24h) but also boasts 99.99% steady-state availability (because if it does crash, it reboots in 1 minute). However, **do not mix the two in the same calculation context** – a given analysis of a network should use either a reliability approach or an availability approach at one time. In other words, when modeling a network’s performance, you wouldn’t multiply a router’s availability by a switch’s 24h reliability; you would either compute the **system availability** from all component availabilities or the **system mission reliability** from all component reliabilities, depending on what you need to evaluate. Mixing them without care leads to confusion and incorrect results. Always align with the scenario: use reliability for mission success probability (if any failure during the mission = mission failure), and use availability for continuous service scenarios where components can fail and be restored.

- **Maintainability Influence:** Reliability as defined ignores repair, whereas availability inherently includes maintainability (MTTR). For a high-availability system, improving MTTR (faster repairs) boosts availability but doesn’t change single-mission reliability (since a faster repair doesn’t prevent the failure in the first place, it just shortens downtime). So, if you are analyzing **design improvements**, decide if you want to focus on reliability enhancements (reducing failure rate, increasing MTBF) or availability enhancements (also reducing downtime via lower MTTR). Often both are improved in tandem, but the metrics will reflect different aspects of those improvements.

Bottom line: You can present both metrics in an analysis, but keep them clearly delineated. They are **complementary** – reliability is a probability of uninterrupted operation for a duration, while availability is a proportion of uptime over the long run [12]. Use each in the appropriate context and time scale.

UI Design: Switching Between Availability and Reliability Modes

To integrate both metrics into an interactive network simulator, it's best to provide a clear **mode switch** or equivalent UI mechanism. This allows the user to choose whether they want to perform an availability analysis or a reliability analysis of the network. Key recommendations for the UI and inputs:

- **Toggle or Mode Selection:** The tool should have a switch (e.g. a toggle, radio buttons, or tabs) between **"Availability mode"** and **"Reliability mode."** This makes it explicit which metric is being computed for the network. All computations and outputs (both at component and system level) would then adapt to that mode. For example, in Availability mode the outputs might show % uptime for each component and the network, whereas in Reliability mode they might show the probability of mission success (no failure) for a given time. It should be evident to the user which context they're in, to avoid confusion.

- **Component Input Fields:** It's wise to allow users to input **MTBF and MTTR for each component** (since these are often the data they have from specs or field data) and let the software derive both availability and reliability from them as needed. However, the relevance of MTTR changes with the mode:

- In **Availability mode**, both MTBF and MTTR are used to compute the component's availability $A = \frac{MTBF}{MTBF+MTTR}$. You might display the computed A for feedback. Some UIs might even allow direct input of an availability percentage instead, but having MTBF/MTTR is useful because from those you can derive other info (like expected downtime per year, etc.). If a user enters MTBF and MTTR, you can auto-calculate and show A. For clarity, labeling the fields "Mean Time Between Failures" and "Mean Time To Repair" and perhaps also showing the resulting "Availability" can help the user understand the relationship.

- In **Reliability mode**, the MTTR field is not needed for the calculation (since no repairs are considered during the mission). The primary input is the failure rate or MTBF (or equivalently MTTF for first failure). In this mode, you could **grey out or hide the MTTR field** for each node to emphasize it's not used, or at least ignore MTTR in calculations. The component's reliability over the mission time t would be calculated from its MTBF (assuming exponential unless a different distribution is specified). For example, if MTBF = 200 hours and mission time t = 50 hours, the tool would compute $R(50) = e^{-50/200} = 0.7788$ (77.9% chance of no failure in 50h). If the user had entered an MTTR of say 10 hours for that component, Reliability mode would essentially disregard that input (or you could use it only to compute MTBF=MTTF+MTTR if you want to derive MTTF, but that's an extra nuance). To avoid user confusion, it's often best to visually indicate MTTR is not applicable in this mode.

- **Global Mission Time Setting:** In reliability calculations, you need a specified **mission time horizon** (since reliability is a function of time). It's recommended to have a **global mission time input** in the UI (for example: "Mission time = ___ hours" or a slider). This would apply to all components and to the system as a whole for computing reliability. All component reliabilities R(t) would be evaluated at that mission time. By making it a single global parameter, you ensure consistency (the whole network is assessed for the same duration). In Availability mode, this mission time input could be hidden or not required (since steady-state availability is time-independent), or if provided it might only be used for secondary info (like computing point availability or number of expected failures in that time, etc., but that's optional). Keeping mission time global also makes it easy for the user to, say, adjust the time and see how system reliability drops as mission time increases, without needing to change each component individually.

- **Auto-Derived Metrics Display:** The UI can use the MTBF/MTTR inputs to show both metrics to the user, but emphasize the one relevant to the current mode. For instance, if a user inputs MTBF=500h, MTTR=10h for a node, the tool might show "Availability = 98%" next to that node (since $500/(500+10) = 0.9804$) and also "Reliability = 85% for 100h mission" (if the global mission time is 100h, then $R(100) = e^{-100/500} = 0.8187$, ~81.9% – this could be shown if it doesn't clutter the UI). This dual display can be informative, but be careful: it should be clear that the reliability percentage is tied to the specific mission time currently set. A tooltip or sub-label like "Reliability (no failure in X hours)" can clarify this.

- **Input Flexibility:** Depending on user preference, you might allow alternative input modes as well. For example, some users might know the **availability target** directly (e.g. "five nines" 99.999% uptime) and not have clear MTBF/MTTR. In availability mode, you could let them enter availability and either MTBF or MTTR, and derive the other (since A = MTBF/(MTBF+MTTR) gives one equation with two unknowns – you'd need an assumption to split it, often MTBF is much larger than MTTR so MTTR could be derived if one chooses a downtime per year, etc.). However, this can complicate the UI, so at minimum providing MTBF & MTTR fields is straightforward. In reliability mode, if a user has a specific reliability requirement (like "90% chance to last 24 hours"), they could enter that and derive required MTBF, but that's more of a design calculation than an input – probably outside scope of the simulator UI except as a separate utility.

- **Consistent Network Calculations:** Ensure that when the mode is switched, the **network-wide results** update using the correct metric. For example, in Availability mode, each node's MTBF/MTTR gives an availability, and the system availability is computed (using series/parallel formulas discussed next). In Reliability mode, each node's MTBF yields a reliability for the mission time, and system reliability is computed accordingly. The UI should update labels, units, and perhaps the formula shown to the user to reinforce this. For instance, the output could say "**System Availability:** 99.5%" in one mode versus "**System Reliability (t = 72h):** 88%" in the other. If the user switches mode, it might be helpful to remind them to verify inputs (e.g., if someone entered a very large MTTR to see its effect on availability, that doesn't affect reliability but might confuse the user until they recall the difference).

- **Help and Defaults:** Because this can be conceptually tricky for users, consider adding a small help tip or an example in the UI. For example: "*Availability mode: uses MTBF & MTTR to compute percent uptime. Reliability mode: uses MTBF (failure rate) to compute probability of zero failures over the mission time.*" Also, you might set sensible defaults (like a default mission time of 100 hours or 1 year, depending on context) so that if the user doesn't change it, they still see a meaningful reliability number.

By structuring the UI in this way, users can **seamlessly switch** between evaluating their network in terms of availability and reliability. This dual-mode approach essentially gives two lenses on the system: one for long-term operational uptime, and one for short-term success probability. Just ensure the switch is explicit to prevent any confusion, and recalculate all relevant values when the mode changes.

Best Practices for Network-Wide Availability vs. Reliability Modeling

When modeling an entire network (routers, switches, firewalls, links, etc. in series/parallel configurations), the approach to **combining component metrics** differs slightly between availability and reliability, although the structure (series vs parallel logic) is analogous. Here are best practices for modeling at the system level:

- **Use Reliability Block Diagrams (RBDs) or Similar Logic:** Represent the network as a reliability block diagram, where blocks (nodes) correspond to components and their interconnection (series or parallel) reflects the network design. In an RBD, “series” means all components must work for the system to work (a single point of failure chain), and “parallel” means redundant components where at least one needs to work for the system to function [14] [15]. This diagrammatic approach helps visualize how component reliabilities or availabilities combine.

- **System Reliability Calculation:** In reliability mode (no repairs during mission):

- For **series** components (all must function): the system reliability is the **product** of the component reliabilities. For example, if a router and a firewall are in series (both needed), and their individual reliabilities for 10-hour mission are 0.99 and 0.97, then system reliability is $0.99 \times 0.97 \approx 0.9603$ (96.03% chance both survive 10 hours). In general, $R_{\text{series}}(t) = \prod_{i=1}^{n} R_i(t)$. (This assumes independent failure behavior for components.)
- For **parallel** components (redundant, either can keep system up): the system reliability is **higher** – it’s the probability that **at least one** component survives the mission. If components fail independently, this is $R_{\text{parallel}}(t) = 1 - \prod_{i=1}^{n} \left(1 - R_i(t)\right)$ 【16†L229-L238】 . For example, two redundant servers each with 0.90 reliability for 1 day: system reliability = $1 - (1-0.90)*(1-0.90) = 1 - 0.1*0.1 = 0.99$ (99% chance that at least one is still up after 1 day). Parallel redundancy can dramatically improve mission reliability if components are not likely to fail simultaneously in that short period.

- For more complex networks (combinations of series and parallel blocks), break the network into segments and apply these rules hierarchically (or use the RBD tool’s computation engine). The key is consistency: **all inputs should be reliability values (at the given mission time)** when calculating system reliability.

- **System Availability Calculation:** In availability mode (steady-state, with repairs happening):

- You can use a similar combination approach, treating each component’s **steady-state availability (a probability)** as analogous to a reliability probability. In fact, the formulas mirror those for reliability:
    - **Series:** The system is up only if *all* components are up. So the system availability is the **product of all component availabilities** [17] [18]. For example, if three switches are in series (all must be operational for network connectivity) with availabilities 0.995, 0.999, 0.98, then system availability = 0.995 × 0.999 × 0.98 $\approx$ 0.974 (97.4%). Just like reliability, adding more

series elements multiplies probabilities and thus lowers overall success rate – **a chain is as strong as its weakest link** (and indeed usually weaker, since the multiplication by <1 numbers compounds the downtime) [19].
    - **Parallel:** The system is down only if *all* parallel components are down at the same time. Thus, system availability = $1 - \prod_i (1 - A_i)$ [16] [20]. For two identical components with availability 0.99 in parallel, $A_{\text{system}} = 1 - (1-0.99)^2 = 1 - 0.0001 = 0.9999$ (99.99%). This reflects the chance that both would happen to be down simultaneously. Parallel redundancy **greatly boosts availability** because even though each component might have downtime occasionally, the odds that two (or N) fail at the exact same time is very small, especially if repairs are quick [21] [22].
- These formulas assume statistical independence of failures and repairs. In practical terms, that means one device's failure doesn't directly cause the redundant device to fail, and their downtime overlaps are rare or random. With exponential failure/repair assumptions, treating availabilities as independent probabilities works well for these calculations. If there are common-cause failures or dependencies (e.g., redundant routers both share a power source that can fail), the model must incorporate that (availability would be lower than the simple formula predicts in that case). Best practice is to document such assumptions and, if needed, adjust the model or add "common mode" failure nodes, etc., in an advanced tool.

- **Tip:** Ensure that the notion of "parallel" in the model truly represents redundant functionality in the network. For example, two firewalls in parallel means either can handle the traffic if the other is down. If that's the case, use the parallel formula. If instead both are actively needed (load-sharing scenario for capacity), that's actually a partial redundancy scenario (which can be modeled as a k-out-of-n requirement – more complex but some tools support it) [23] [24]. In straightforward cases, series and full parallel cover most network designs (e.g., a single path vs. a fully redundant path).

- **Time Scale and Results Interpretation:** Availability is typically reported as a percentage (uptime fraction) or in "nines" per year (e.g., 99.99% = 4 nines). Reliability is reported as a probability for a given mission time or as a function $R(t)$. When presenting results:

- For availability, it can be helpful to also translate that into expected **downtime** over a reference period. For instance, 99.9% availability means ~0.1% downtime, which is ~8.76 hours per year of downtime. This gives users an intuitive sense of what that availability means in operational terms (days/hours of network outage per year). Many availability modeling tools show this as in the EventHelix example (where they listed downtime per year alongside availability) [19] [25].

- For reliability, especially if the mission time is shorter (hours or days), giving the raw probability (like "System reliability for 24-hour mission = 98%") is usually clear. If the mission time is variable or the user might want to see how reliability changes over time, consider allowing the output of a **reliability curve** or the ability to compute R at different t. But at least, make it clear that reliability is tied to that specific mission duration.

- **Repairable Systems and Mission Availability:** Sometimes you might be interested in the probability the system is operational at time t **with repairs allowed in between** (point availability at t). This is different from reliability, and in steady-state it equals the steady-state availability A. If the mission time is not long enough to reach steady state, computing time-dependent availability is more complex (it involves solving differential equations or performing simulations) [12] [26]. For most edge network simulations, you can assume either a long-running scenario (so steady-state is applicable) or a no-repair short mission (for reliability). Best practice is **not to attempt to mix** these by, say, allowing some repairs in a short mission – that requires Markov or simulation analysis beyond simple block diagrams. Instead, pick the scenario: *either* the mission is long enough that steady-state availability holds, *or* the mission is short and no repair (reliability) is the metric. This keeps the calculations clean and understandable. If needed, you can always run a Monte Carlo simulation in the backend to estimate availability over time, but that's an advanced feature.

- **User Education and Documentation:** Make sure to provide guidance (in-app help or documentation) so users of the tool know when to use availability vs. reliability. Some best practices to convey:

- **Use Availability** for evaluating **continuous services** – e.g., "What percentage of time will my network be up over a typical year? How many hours of downtime can I expect?" This is ideal for service level agreements (SLAs), maintenance planning, etc., in systems where components can be fixed or restarted quickly and the operation goes on indefinitely.
- **Use Reliability** for **one-time or short missions** – e.g., "What is the chance my network stays up for the 8-hour event without any outage?" This is critical for scenarios like a live broadcast, a space mission, or any fixed-time operation where even a brief failure is mission-ending. Reliability is also useful during design to set requirements (e.g., "we need 95% probability of no failures in a 72-hour test period").

- It is acceptable (and often wise) to examine both for critical systems: high reliability for the short term and high availability for the long term. For example, a system could have high availability (quickly repaired failures) but if those failures are too frequent it might have low short-term reliability. Both metrics together give a fuller picture [27]. But always interpret each in the right frame of reference.

- **Validation with Real Data:** If possible, validate the modeled calculations with empirical data or known industry benchmarks. For instance, if a certain network design is known to achieve "five nines" availability, the model's inputs (MTBF/MTTR for components) should reflect that. Similarly, if field data shows that during a 12-hour operation typically 0 failures occur 95% of the time, then your reliability inputs should align. This helps ensure the MTBF/MTTR numbers you plug in are realistic.

- **Consider Maintenance Strategies:** In availability modeling, MTTR can be influenced by repair strategies (on-site spares, failover protocols, etc.). Two components with the same MTBF but different MTTR (e.g., one has hot standby failover, effectively near-zero downtime, another requires 2 hours to replace) will have the same reliability for short missions (since the chance of failure in, say, 1 hour is related to MTBF only), but *very different availabilities*. Best practice is to capture these differences in MTTR and ensure your network model accounts for them. For example, a redundant firewall pair might have a slightly higher MTTR for a failure (because failover takes a few seconds but service isn't really interrupted – one might treat that as near 0 MTTR from an availability perspective for the service, if failover is seamless). Adjust inputs to reflect the architecture (some tools might allow modeling failover as part of the block diagram logic with standby spares, etc., but that's more advanced).

- **Common Pitfalls:** Avoid the mistake of using steady-state availability in situations where it doesn't apply (e.g., for a brand-new system over a short initial period). Steady-state formulas like $A =$

$\frac{MTBF}{MTBF+MTTR}$ assume the system has been operating long enough to reach equilibrium. If you are interested in the first day of operation of a system, instantaneous availability is initially just reliability (since no repair has happened yet) [13]. However, for most practical purposes in network modeling, using the steady-state availability is fine as long as MTBF is much larger than MTTR and you're looking at long-run performance [28]. Just keep in mind the "availability paradox": a system with high availability can still have frequent small outages. If your use case can't tolerate even a brief outage, then reliability (no failures allowed in interval) is the right metric.

By adhering to these practices, you can implement a **clean dual-mode calculation** in your network simulation tool. Users can toggle between seeing, say, that their network of routers/switches has *99.9% availability* (meaning ~9 hours downtime per year), versus *90% 30-day reliability* (meaning a 10% chance that at least one outage occurs in a given month-long mission). Both views are valuable. The tool should help users input consistent data and interpret the results correctly, whether they are designing for continuous uptime or single-mission success.

References:

- Reliability vs. Availability concepts [12] [13]
- Steady-state availability formula $A = MTBF/(MTBF + MTTR)$ [1] [2]
- Exponential reliability model and relation to MTBF [6] [7]
- Use of constant failure rate (exponential) vs. other distributions [9] [11]
- Series and parallel combination formulas for availability and reliability [29] [20]

---

[1] [4] [16] [17] [21] [29] Availability - Wikipedia
https://en.wikipedia.org/wiki/Availability

[2] [3] [12] [26] [28] Availability and the Different Ways to Calculate It
https://help.reliasoft.com/articles/content/hotwire/issue79/relbasics79.htm

[5] [7] [8] Reliability and MTBF Overview
https://www.vicorpower.com/documents/quality/Rel_MTBF.pdf

[6] Mean time between failures - Wikipedia
https://en.wikipedia.org/wiki/Mean_time_between_failures

[9] [10] Exponential Distribution
https://support.ptc.com/help/wrr/r12.0.4.0/en/wrr/ReferenceGuide/appendix/convert_prior_team_files_wrr_team_files.html

[11] How About Weibull Instead of MTBF? - Accendo Reliability
https://accendoreliability.com/weibull-instead-mtbf/

[13] Microsoft PowerPoint - 5214v14
https://people.cs.vt.edu/~irchen/5214/pdf/p24-43.pdf

[14] [15] [18] [19] [20] [22] [23] [24] [25] System Reliability and Availability
https://www.eventhelix.com/fault-handling/system-reliability-availability/

[27] What Is MTBF? Understanding Reliability Metrics
https://reliabilityqualitysolutions.com/what-is-mtbf/