Modeling Metastability

Ali Farahbakhsh
Cornell University

Andreas Haeberlen
University of Pennsylvania / Roblox

Qingjie Lu
University of Pennsylvania

Lorenzo Alvisi
Cornell University

Robbert van Renesse
Cornell University

Shir Cohen
Cornell University

Abstract

Recently, there has been increasing concern about a new failure mode in data-center systems: when there is an external shock, such as a sudden load spike or some machine failures, systems will sometimes respond with reduced throughput – but, in contrast to a traditional overload situation, the throughput does not recover once the external shock disappears, and remains permanently degraded. This phenomenon has been called a *metastable failure*.

In this paper, we sketch a simple model that could help to explain how and why metastability arises. We also show how our model can be used to predict the presence or absence of metastable states in a given system.

CCS Concepts

• **Computer systems organization → Reliability**.

Keywords

Metastability, fault tolerance

ACM Reference Format:
Ali Farahbakhsh, Andreas Haeberlen, Qingjie Lu, Lorenzo Alvisi, Robbert van Renesse, and Shir Cohen. 2025. Modeling Metastability. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3772356.3772426

1 Introduction

Metastable failures are a relatively new phenomenon. Instances have occasionally appeared in the literature, but more as a curiosity; they have not been studied systematically until a pair of recent papers [1, 4] identified them as part of a larger and previously unexplored category of failures. So far, pretty much everything about them is unknown or poorly understood: their causes, possible mitigation and countermeasures, etc. However, there is concern in industry about these failures because their prevalence seems to be increasing, and because they are so much harder to mitigate and even diagnose than classical failures: a single metastability event can take hours to fully mitigate, and this can mean losing millions of dollars in production. We have heard this concern repeatedly, from multiple sides – most recently in Marc Brooker's keynote at SoCC 2023.

In this paper, we take a first step by suggesting a precise definition of metastability and a general mechanism that can give rise to it. The rough shape of the mechanism is already known: Bronson et al. [1] suggested that metastability involves 1) a bad state of some kind; 2) a trigger that causes the system to enter it; and 3) a self-sustaining loop of events that prevents the system from leaving the bad state even when the trigger is removed. However, this is too vague to start looking for metastability in a given system. Subsequent papers have made things more concrete, but at the expense of generality: Huang et al. [4] specifically studies load spikes and capacity drops, while Qian et al. [6] and Habibi et al. [3] specifically focus on retry storms. Based on our own deployment experience with a massive-scale distributed system, this is at most a small part of the rather large and varied range of metastable behaviors.

In order to answer the "VP's plea" from [1] ("Can you predict the next metastable failure, rather than explain the last one?"), we need a fairly general pattern we can look for – general enough to fit not just the existing examples but also new ones we have not yet seen. For instance, we would prefer to avoid the assumption that triggers are necessarily load spikes, or that the bad state is necessarily, say, a 10% capacity drop. Our goal was to find something like the definition of deadlocks, whose presence can be detected from the presence of a cycle in the resource allocation graph [2]. This captures the essence of what a deadlock is, and it is general enough to apply to a wide variety of systems.

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772426

In this paper, we propose just a such definition. Our definition predicts the existence of a larger and more general set of metastable behaviors than what has been described in the literature so far. In addition, it reveals a duality between a certain type of system and a certain class of linear equations; the latter are somewhat easier to study and help with establishing a precise condition for the presence of metastable states, somewhat analogous to the one for deadlocks. In fact, this may be a sign of a connection to dynamical systems – a rich area with decades of potentially relevant existing work.

2 System model

We begin by describing a system model that abstracts away the functional behavior of a system and focuses exclusively on how work flows through it. Our first goal is to convince the reader that, when analyzing metastability, it makes sense to describe complex distributed systems in this way.

In our model, systems are represented by components that process streams of opaque *tokens*. The components can be connected by "wires"; in addition, each system has at least one input wire and one output wire. Execution proceeds in synchronous rounds; in each round t, the system receives some number of tokens on each input wire and can optionally produce some number of tokens on each output wire. For now, we will use the following four components:

- **emitter(k)**: Emits k fresh tokens in each round.
- **queue(c)**: A queue that can hold up to c tokens. At the beginning of each round, emits all queued tokens.
- **multiplier**: Produces two or more streams of tokens that each contain one token for every input token.
- **subtractor**: Receives two streams of tokens A and B and outputs $A - B$ tokens if $A > B$, or 0 otherwise.

Notice that our system model intentionally contains no normal state – there are no "variables" or memory locations that could be used to remember things. The only state in this type of system is the number of tokens in each queue. Given some arbitrary ordering of the queues, we say that the *state* of a system with N queues is an N-dimensional vector $S = (S_1, \ldots, S_N)$ such that S_i is the number of tokens in the i^{th} queue. In the first round, the system starts in the *initial state* $I := (0, \ldots, 0)$, that is, all the queues are initially empty.

2.1 Example: Retry storm

Figure 1 shows a simple example of how a system would be represented using this approach. We intentionally start with the classic example – the retry storm – that has appeared in each of the earlier papers [1, 4].

This model consists of three queues, two subtractors, two emitters, and one multiplier. Tokens represent requests; they enter the model from the left and are initially stored in a `waiting` queue. In each round, up to S requests are moved to a `served` queue; the rest go back to the `waiting` queue. If the number of waiting requests exceeds a threshold T, the requests time out and copies are moved to the `rexmits` queue, which feeds back to the waiting queue. Both the number of waiting requests and the number of retries are limited by the queue capacities Q_{max} and R_{max}, respectively. The system outputs a token for each request that has been served.

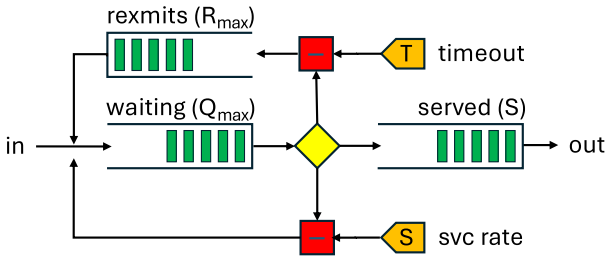


Figure 1: A system in which requests are retransmitted after a timeout. Multipliers are shown as diamonds, emitters as arrows, and subtractors as squares.

In Section 3.1, we will show that this model captures a well-known metastable behavior: once there are enough waiting requests that the number of retransmissions exceeds the service rate S, the number of requests explodes. The model focuses on the essence of this problem: we do *not* need to model what happens to individual requests, or what exactly the requests do, in order to capture metastable behavior.

2.2 Connection to real distributed systems

For our model to be useful more generally, its components should represent functionality that commonly occurs in real distributed systems. Here are some practical examples:

Emitter: An emitter describes any part of a system that is periodically active on its own, even without any input. For instance, a periodic heartbeat message, periodic replica maintenance, a periodic nightly backup, or periodic routing updates could all be represented as emitters.

Queue: Queues are ubiquitous in distributed systems. The variant in our model is a bit unusual in that it drains completely in each round. We made this choice to simplify the mathematical model (Section 3.2) and to use the same component to represent both queueing and delay: a sequence of k queues causes a delay of k rounds.

Multiplier: This component represents any kind of amplification: for instance, if a storage system maintains k replicas, incoming writes would typically cause k internal writes, one at each replica. Other examples include pub-sub systems, multicast systems, and route distribution.

Subtractor: The subtractor represents situations where multiple activities compete for the same fixed set of resources. For instance, a storage system might be handling high-priority

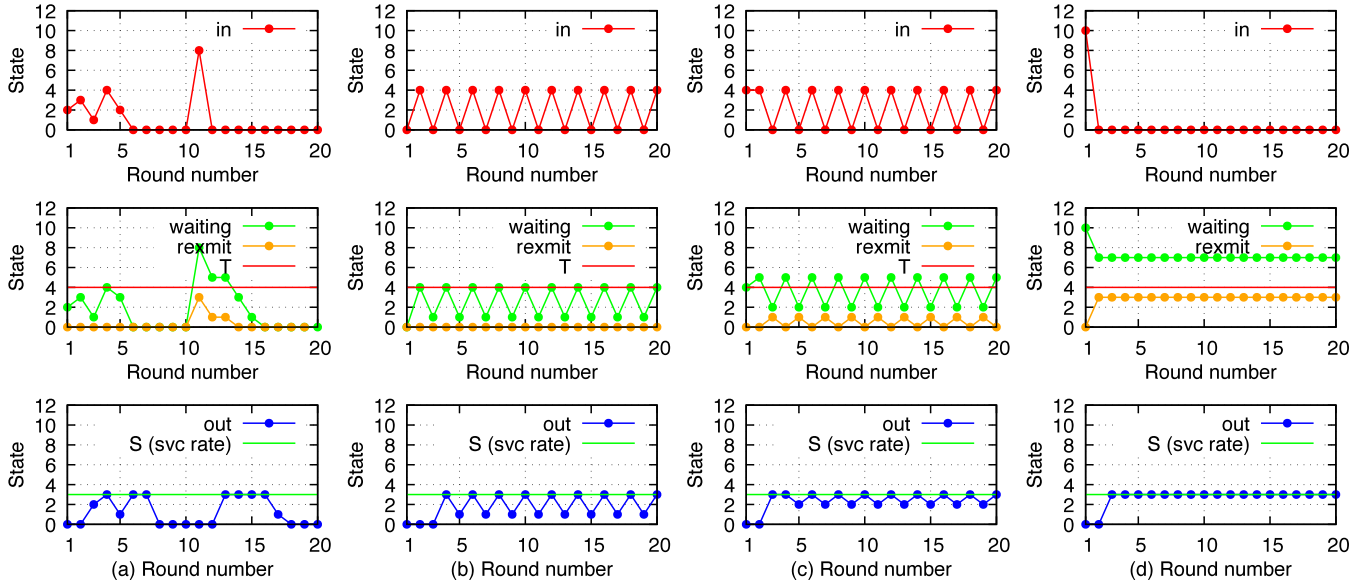


Figure 2: Behavior of the system from Figure 1 for (a) a single 8-token spike; (b) a $(\overline{4, 0})$ trace, starting from I; (c) a weakly metastable state $(4, 0, 0)$; and (d) a 10-token spike that enters strongly metastable state $(10, 0, 0)$. Each column consists of the input trace (top), the state of the `rexmits` and `waiting` queues (middle) and the output (bottom).

requests while a low-priority backup is active in the background. Both activities would compete for the same, limited I/O bandwidth; the subtractor can compute how much capacity is "left over" for the backup in each round.

3 Metastability

We now define metastability for systems that can be described with our system model. Informally, our definition captures that there is a trigger – represented by a finite prefix P of the input trace – that is "remembered" by the system and causes the subsequent behavior to be different indefinitely. We also distinguish between cases where the behavior change is permanent and cases where it can be reversed by some kind of "anti-trigger".

Definition 3.1. A state S is *weakly metastable* in a given system iff (1) S is reachable from the initial state I, that is, there is a finite input trace P that takes the system from I to S; and (2) there exists an infinite input trace R such that, if two instances of the system are started in the states S and I, respectively, and both receive the same trace R, their outputs differ infinitely often. S is *strongly metastable* if condition (2) also holds for any finite extension of P.

This kind of "memory" is interesting because we have intentionally defined our system model to contain no ordinary state (say, in-memory or on-disk state); recall that this has been abstracted away, and that the tokens merely represent units of work. Instead, the "memory" arises from feedback loops between its components, as we shall see shortly.

3.1 Examples of metastable states

We illustrate Definition 3.1 using the example system from Figure 1. Suppose the server can handle $S = 3$ requests per round, the input queue can hold $Q_{max} = 10$ requests, requests start timing out once there are more than $T = 4$ requests in the queue, and there can be at most $R_{max} = 3$ retransmissions per round. Figure 2(a) shows a "normal", execution without metastable behavior, in which this system receives a few smaller inputs followed by a larger spike in round 10 (top graph). This causes some queueing in the `waiting` queue which, after the spike, briefly exceeds the threshold T and triggers some retransmissions (middle graph). The output is steady and returns to zero each time (bottom graph).

However, the system does contain some metastable states. $S_1 := (4, 0, 0)$ is an example of a weakly metastable state: it can be reached with a single-element input trace $P_1 := (4)$, and if the trace then continues as $R_1 := (\overline{4, 0})$ ad infinitum – that is, alternates between 0 and 4 – the system's behavior (Figure 2(c)) is different than what it would be had the system started from the initial state I (Figure 2(b)). The reason is that R_1 by itself just barely stays under the threshold for retransmissions, but if the system already starts with enough queued requests when it receives R_1, retransmissions do occur and cause the output to be higher. This is visible in the middle graphs; notice how the `waiting` line exceeds the threshold T in Figure 2(c), but not in Figure 2(b). Somewhat counterintuitively, $R_1' := (\overline{4})$ does not satisfy the definition: in that scenario, the system is overloaded regardless of whether

it starts in state S or I. In R_1, the difference becomes visible in the rounds where the input is zero.

S_1 is *not* a strongly metastable state: if we send three zero inputs, the queues drain and the system returns back to I. However, the state $S_2 := (10, 0, 0)$ is strongly metastable (Figure 2(d)). It, too, can be reached via a single-element input trace $P_2 := (10)$, and once the system is in that state, it is "stuck": in each round, the server handles $S = 3$ requests but there are also $10 - S - T = 3$ retransmissions, so the queue never drains again, even if all future inputs are zero! Thus, $R_2 = (\overline{0})$ satisfies the definition: starting from I, the output will be $(\overline{0})$, but starting from S_2, it will be $(\overline{3})$.

3.2 Mathematical model

An important question is whether we can tell whether a given system has any metastable states. Before we can answer this question, we need to provide a denotational semantics for our otherwise operational model. In essence, each instance of our model corresponds to a finite-state transducer (FST) that transforms a sequence of inputs into a sequence of outputs using a finite state space, namely the cross product of all the queue lengths. (FSTs have limited expressivity, but recall that we are only modeling how work flows through the system, not the actual work itself, which can be substantially more complex.) We can write down the transformation the FST computes by associating a term with each wire as follows: for wires starting at an emitter(k), the term is k; for wires starting at a queue Q, it is Q(t-1); for the input, it is $in(t-1)$; for a multiplier(k), it is the term on the multiplier's input wire; for a subtractor(A,B), it is $(A-B)_{\perp 0}$ (where $(\cdot)_{\perp 0}$ clips negative arguments to zero); and for a wire with multiple inputs, it is the sum of the terms on the input wires. Then the equation for queue Q with capacity C is $q(t) = X^{\top C}_{\perp 0}$, where X is the term on the queue's input wire and $(\cdot)^{\top C}_{\perp 0}$ clips arguments larger than C to C, and the equation for the output is the term on the output wire. If we use this method for our example system in Figure 1, we get:

$$
\begin{aligned}
wait(t) &= (in(t-1)+(wait(t-1)-S)_{\perp 0}+rexmits(t-1))^{\top Q_{max}}_{\perp 0}\\
served(t) &= (wait(t-1))^{\top S}_{\perp 0}\\
rexmits(t) &= (wait(t-1)-T)^{\top R_{max}}_{\perp 0}\\
out(t) &= served(t-1).
\end{aligned}
$$

It is also possible to convert a given set of equations back into a system, provided that it has the above format.

3.3 More complex examples

The system from Figure 1 has a metastable behavior: after an input spike of sufficient magnitude, a permanent retry storm occurs. However, metastable behaviors can be substantially more complex. We provide a few examples here.

Oscillation: Consider the system in Figure 3a. It can be expressed with the following equations:

$$
\begin{aligned}
out(t) &= in(t-1) + a(t-1)\\
a(t) &= ((in(t-1)-5)_{\perp 0} + b(t-1))^{\top 10}_{\perp 0}\\
b(t) &= a(t-1)^{\top 10}_{\perp 0}.
\end{aligned}
$$

Once an input spike of magnitude six or larger occurs, this system will enter a strongly metastable state and permanently add a spike to its output every two rounds. In general, adding a k-round pattern will require a loop with k queues; in this case, $k = 2$. We have not seen oscillation discussed in the literature on metastability so far, but we have observed instances of this in our data centers.

Specific triggering pattern: In the existing case studies from the literature, the trigger tends to be a single input spike above a certain magnitude. However, metastability can also be triggered by specific *patterns*. For instance, the circuit in Figure 3b triggers a runaway effect once the input contains the specific sequence 5-4. The equations are:

$$
\begin{aligned}
prev(t) &= in(t-1)^{\top 10}_{\perp 0}\\
saw5(t) &= (in(t-1)-4)^{\top 1}_{\perp 0} - (in(t-1)-5)^{\top 1}_{\perp 0}\\
saw4(t) &= (prev(t-1)-3)^{\top 1}_{\perp 0} - (prev(t-1)-4)^{\top 1}_{\perp 0}\\
det(t) &= (saw5(t-1) + saw4(t-1) + 2\cdot det(t-1) - 1)^{\top 1}_{\perp 0}\\
out(t) &= in(t-1) + det(t-1).
\end{aligned}
$$

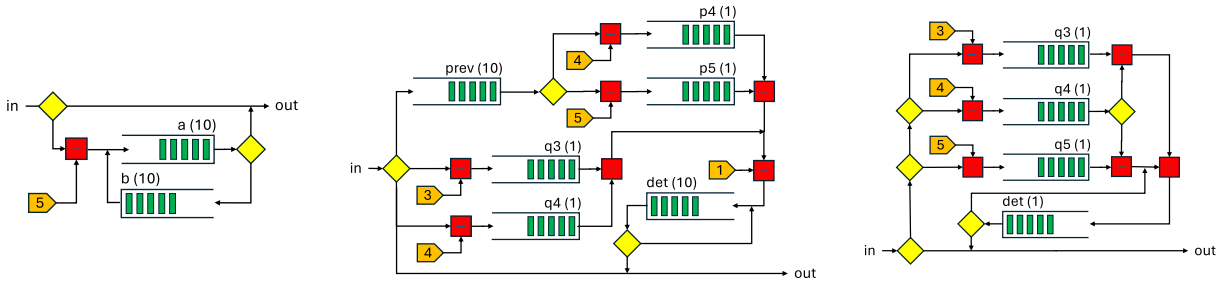
In Figure 3b, $saw5$ and $saw4$ are implemented by groups of components around $p5/p4$ and $q4/q3$, respectively. Notice that we can essentially implement Boolean logic here: the terms in $saw5$ are 1 if the previous input was at least 5 and 6, respectively, and 0 otherwise; det is the equivalent of an AND gate. So, with enough queues, we can look for any finite subsequence in the input!

Bistable system: The system in Figure 3c is similar to a flip-flop: Its equations are:

$$
\begin{aligned}
saw5(t) &= (in(t-1)-4)^{\top 1}_{\perp 0} - (in(t-1)-5)^{\top 1}_{\perp 0}\\
saw4(t) &= (in(t-1)-3)^{\top 1}_{\perp 0} - (in(t-1)-4)^{\top 1}_{\perp 0}\\
det(t) &= (saw4(t) + (det(t-1) - saw5(t))_{\perp 0})^{\top 1}_{\perp 0}\\
out(t) &= (in(t-1) + det(t-1))^{\top 10}_{\perp 0}
\end{aligned}
$$

It outputs its input until exactly three tokens are observed in any single round; after that, the outputs increase by one token. This behavior continues until exactly five tokens are input in any single round; then it returns to normal. This is an example of a weakly metastable state.

Multiple metastable states: So far, we have only seen systems that have a single metastable behavior and a single



(a) Oscillation **(b) Specific trigger** **(c) Bistable**

Figure 3: Example system models with more complex metastable behaviors

trigger. But in fact, the same system can have multiple different triggers with different effects. Consider the following:

$$3to6(t) = (in(t-1) - 2)_{\perp 0}^{\top 1} - (in(t-1) - 6)_{\perp 0}^{\top 1}$$
$$feed(t) = (3to6(t-1) + 2 \cdot feed(t-1))_{\perp 0}^{\top 100}$$
$$mem(t) = ((in(t-1) - 9)_{\perp 0}^{\top 5} + mem(t-1))_{\perp 0}^{\top 5}$$
$$out(t) = (in(t-1) + feed(t-1) - mem(t-1))_{\perp 0}.$$

Here, inputs between 4 and 6 will trigger a feedback loop to output 100, whereas inputs greater than 9 will trigger a permanent throughput drop.

3.4 Metastability requires a feedback cycle

Our model also allows us to formalize the hypothesis from [1] – that metastability requires a feedback cycle – as follows:

Theorem 3.2. *If a system has a metastable state, it contains at least one variable whose equation depends, directly or indirectly, on an earlier value of the same variable.*

Proof. Assume the opposite, that is, the system has at least one metastable state S but no variable depends, directly or indirectly, on its own earlier values. Then, whenever a term $V(t-x)$ for some variable V appears on the right side of an equation, we can substitute in its place the right side of V's own equation, with the $t-x$ substituted in for the argument. For instance, if $a(t) = b(t-1) + 6$ and $b(t) = in(t-1)_{\perp 0}^{\top 10}$, then we get $a(t) = in(t-2)_{\perp 0}^{\top 10} + 6$. Since we have assumed that there are no cyclic dependencies, a finite number of substitutions will transform each equation to one that has in as its only variable, i.e., that depends only on input values at various past points in time. Let k be the smallest number such that, whenever $in(t-x)$ appears in the equation for $out(t)$, $x \leq k$. Then the output cannot depend on any input that was received more than k steps ago. But now consider what would happen if we run two copies of the system, one from the initial state I and one from the metastable state S we have assumed exists. Although the outputs of the two systems can differ for the first k inputs, they will inevitably be the same after that. But if S truly were a metastable state, then the output would have to differ infinitely often. So S cannot be a metastable state, and the claim follows. □

Note that cyclic dependencies are not enough to cause metastable states. For instance, if we take the three components at the top of Figure 1 away, we get a simple queue:

$$wait(t) = (in(t-1) + (wait(t-1) - S)_{\perp 0})_{\perp 0}^{\top Q_{max}}$$
$$served(t) = (wait(t-1))_{\perp 0}^{\top S}$$
$$out(t) = served(t-1)$$

This system has a feedback cycle for the queue $wait$, but there are no metastable states.

3.5 Finding metastable states

How do we decide whether a given system has metastable states – and if it does, how can we find them? In our model, this can be done with the following procedure.

Let X be the original FST. First, we compute the set of reachable states in X, along with a trace T_S that can be used to reach each state S from I. Then, for each reachable state $S \neq I$ of X, we build a new FST X_S. The states of X_S are concatenations (S_1S_2) of two states S_1 and S_2 of the original FST X, plus a single terminating state T, and the starting state is (IS). For the transition from a state (S_1S_2) with input i, there are two cases. Suppose the original FST X produces the outputs o_1 and o_2 when it receives i in states S_1 and S_2, and transitions to S_1' and S_2', respectively. Then, if $S_1' \neq S_2'$, the transition goes to $(S_1'S_2')$ and produces output o_1o_2, otherwise it goes to T and produces an arbitrary output (say, o_1o_1).

Intuitively, X_S simulates two instances of the original FST "in parallel" and produces the outputs the two instances would produce, as long as they are in different states; if and when the two instances reach the same state, the FST terminates because the two instances will always have the same output after that point, as our systems are deterministic. We can then look for cycles in this FST such that (1) the cycle can be reached from (IS), and (2) the outputs along the cycle contain at least one o_1o_2 with $o_1 \neq o_2$.

It is easy to see that, if such a cycle is found, S is a weakly metastable state: T_S satisfies the first condition of the definition (S is reachable from I if T_S is input), and the inputs along the path from (IS) to the cycle, plus the inputs along the cycle itself repeated ad infinitum, satisfy the second condition: both instances go through a cycle of states indefinitely, without ever reaching the same state, and produce at least one different output along the way. Conversely, if S is a weakly metastable state, such a cycle must exist in the FSM.

4 Conclusion and ongoing work

We admit that we are still far from our goal of fully understanding metastability. However, our model could be an important step: rather than describing specific instances of metastable behavior, like the earlier work [1, 3, 4, 6] has done, it offers a possible mechanism that could *explain* how metastability arises. Our model also hints at a potentially much larger space of metastable behaviors: triggers can be arbitrary patterns and not just spikes, and the sustaining effects are not limited to work amplification or decreased efficiency and can include, e.g., oscillation. Finally, our model can be used to *predict* the presence or absence of metastable states in the system it describes, which is an exciting prospect and could a first step towards answering the "VP's plea" from [1].

An important question is how one would generate a model for a given system. We speculate that system identification [5] could be used to build an initial model from diagnostic data, which many large-scale systems capture anyway. Another question is whether our set of four building blocks is sufficient, or whether there are other behaviors that need to be captured in the model. And finally, a practical solution would need to go far beyond the toy examples we have presented here, and be able to operate at datacenter scales. We hope to answer these questions in our ongoing work.

Acknowledgments:

We thank our shepherd Scott Shenker and the anonymous reviewers for their comments and suggestions. This work was supported in part by NSF grant CNS-1955670.

References

[1] N. Bronson, A. Aghayev, A. Charapko, and T. Zhu. Metastable failures in distributed systems. In *Proc. HotOS*, 2021.

[2] E. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *ACM CSUR*, 3(2), 1971.

[3] F. Habibi, T. Lorido-Botran, A. Showail, D. C. Sturman, and F. Nawab. MSF-Model: Modeling metastable failures in replicated storage systems. *CoRR*, abs/2309.16181, 2023.

[4] L. Huang, M. Magnusson, A. B. Muralikrishna, S. Estyak, R. Isaacs, A. Aghayev, T. Zhu, and A. Charapko. Metastable failures in the wild. In *Proc. OSDI*, 2022.

[5] L. Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010.

[6] S. Qian, W. Fan, L. Tan, and Y. Zhang. Vicious cycles in distributed software systems. In *Proc. 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2014.