Robustness Analysis of Networked Systems*

Roopsha Samanta[1], Jyotirmoy V. Deshmukh[2], and Swarat Chaudhuri[3]

[1] University of Texas at Austin roopsha@cs.utexas.edu
[2] University of Pennsylvania djy@cis.upenn.edu
[3] Rice University swarat@rice.edu

Abstract. Many software systems are naturally modeled as networks of interacting elements such as computing nodes, input devices, and output devices. In this paper, we present a notion of robustness for a networked system when the underlying network is prone to errors. We model such a system $\mathcal{N}$ as a set of processes that communicate with each other over a set of internal channels, and interact with the outside world through a fixed set of input and output channels. We focus on network errors that arise from channel perturbations, and assume that we are given a worst-case bound $\boldsymbol{\delta}$ on the number of errors that can occur in the internal channels of $\mathcal{N}$. We say that the system $\mathcal{N}$ is $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$-robust if the deviation of the output of the perturbed system from the output of the unperturbed system is bounded by $\boldsymbol{\epsilon}$.
We study a specific instance of this problem when each process is a Mealy machine, and the distance metric used to quantify the deviation from the desired output is either the L_1-norm or the Levenshtein distance (also known as the edit distance). We present efficient decision procedures for $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$-robustness for both distance metrics. Our solution draws upon techniques from automata theory, essentially reducing the problem of checking $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$-robustness to the problem of checking emptiness for a certain class of reversal-bounded counter automata.

1 Introduction

More than ever before, we live in an era where computation does not exist in a vacuum, but is tightly integrated with networked communication and, often, interactions with the physical world. The heterogeneous systems that result from such integration — medical devices, power plants, vehicles and aircrafts — are often safety-critical. Unsurprisingly, they have long been regarded as important targets for formal methods.

One aspect of such systems that has received relatively less attention in the formal methods literature is *uncertainty*. Uncertainty is pervasive in complex heterogeneous systems—for example, the data generated by sensors in the system can be inexact or corrupted, or the network channels implementing communication between the components of the system can be corrupt or lose data

* This research was partially supported by CCC-CRA Computing Innovation Fellows Project, NSF Award 1162076 and NSF CAREER award 1156059.

packets. Left unchecked, such uncertainty can wreak havoc. For a large class of systems, we often wish to determine to what extent the system behavior is predictable, when the system faces uncertainty. Traditional system correctness properties like safety and liveness are *qualitative* assertions about individual system traces, and proof techniques for such properties typically do not provide any *quantitative* measure on predictable system execution. In most engineering disciplines, the core property in reasoning about uncertain system behavior is *robustness*: "small perturbations to the operating environment or parameters of the system does not change the system's observable behavior substantially." This property is *differential*, in the sense that it relates a range of system traces possible under uncertainty. Furthermore, proof techniques to prove robustness demand a departure from traditional correctness checking algorithms as they require quantitative reasoning about the system behavior.

Given the above, formal reasoning about robustness of systems is a problem of practical as well as conceptual importance. In well-established areas such as control theory, robustness has always been a fundamental concern; in fact, there is an entire sub-area of control — *robust control* — that extensively studies this problem. However, as robust control typically involves reasoning about continuous state-spaces, the techniques and results therein are not directly applicable to cyber-physical systems which contain large amounts of discretized, discontinuous behavior.

In the context of cyber-physical systems, robustness analysis has only recently begun to gain attention. While several recent papers explore quantitative formal reasoning about robustness of software, the problem of reasoning about robustness with respect to errors in *networked communication* has been largely ignored. This is unfortunate as communication between different computation nodes is a fundamental feature of most modern systems. In particular, it is a key feature in emerging cyber-physical systems [20, 21] where runtime error-correction features for ruling out uncertainty may not be an option. In this paper, we focus on such networked systems, and characterize an efficiently verifiable notion of robustness for them. At a high level, our contributions in this paper are as follows:

1. We present a model for communicating processes that is representative of the complexity of real systems.
2. We present a model for perturbations in the communication channels in the network, and formulate a notion of robustness for a networked system in the presence of these unreliable channels.
3. We present efficient, automata-theoretic, decision procedures for analyzing the robustness of the networked system with respect to different metrics characterizing the deviation of the observed behavior of the system.

In this paper, we model a synchronous, networked, system $\mathcal{N}$ as a set of communicating Mealy machines (processes). Processes communicate over a set of internal channels, and interact with the outside world through a set of external input and output channels. We assume that processes communicate with each other using symbols from a finite alphabet, and perform computations over

strings of such symbols. Each such symbol can be treated as an abstraction of complex data used for computation and communication in a real-world networked system.

As observed in [9], a critical requirement in networked cyber-physical systems is for all components of the system to have a common sense of time. This is usually ensured by protocols that guarantee that the global clock remains consistent across components. Thus, bearing this observation in mind, and following in the footsteps of recent papers on wireless control networks [20, 21], classic models like Kahn process networks [16], and languages like Esterel [3], we assume our networks to have synchronous communication.

An input to the networked system $\mathcal{N}$ is a word capturing the sequence of symbols appearing on the input channels of $\mathcal{N}$; the externally observable behavior of $\mathcal{N}$ is a sequence of symbols appearing on its output channels. We model uncertainty by letting each internal channel perturb the data that is sent through it at any given point. Perturbations can include deletion of symbols and mutating symbols to other symbols. Deviations from the ideal, unperturbed, system's observable behavior are defined using suitable distance metrics on words. In this paper, we consider two such metrics: *Levenshtein distance* and the L_1*-norm*. We define the networked system $\mathcal{N}$ to be $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$*-robust* if for any given input, the maximum change to the observable behavior of $\mathcal{N}$ is bounded by $\boldsymbol{\epsilon}$ as long as the number of perturbations introduced by the internal channels of $\mathcal{N}$ is bounded by $\boldsymbol{\delta}$.

Our central technical result is a decision procedure for determining whether a given system $\mathcal{N}$ is $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$*-robust*. Our algorithm reduces this problem to the problem of checking the emptiness of a certain class of *reversal-bounded counter automata*. A key step in our algorithm is the construction of automata that accept pairs of strings (s, t) if and only if the distance between s and t (w.r.t. a chosen metric) exceeds a specified constant. We present constructions for such automata for Levenshtein distance and the L_1-norm metric. We remark we can check robustness of a networked system with respect to any metric for which such automata constructions are possible.

The rest of the paper is organized as follows. In Sec. 2 we define our model of robust networked systems. In Sec. 3 and Sec. 4, we present the automata constructions involved in our decision procedure for checking robustness of such systems. We discuss related work in Sec. 5, and conclude with a discussion of future work in Sec. 6.

2 Robust Networked Systems

In this section, we present a formal model for a synchronous networked system. We then introduce a notion of robustness for computations of such networked systems when the communication channels are prone to errors. In what follows, we use the following notation. Strings are typically denoted by lowercase letters s, t etc., with output strings sometimes denoted by primed lowercase letters s', t' etc. We denote the concatenation of strings s and t by $s.t$, the j^{th} character

of string s by $s[j]$, the substring $s[i].s[i+1].\ldots.s[j]$ by $s[i,j]$, the length of the string s by $|s|$, and the empty string by λ. We sometimes denote vectors of objects using bold letters such as $\mathbf{s}$ and $\boldsymbol{\epsilon}$, with the j^{th} object in the vector denoted s_j and ϵ_j respectively.

2.1 Synchronous Networked System

A networked system, denoted $\mathcal{N}$, can be described as a directed graph $(\mathcal{P}, \mathcal{C})$, with a set of processes $\mathcal{P} = \{P_1, \ldots, P_n\}$ and a set of communication channels $\mathcal{C}$. The set of channels consists of internal channels N, external input channels I, and external output channels O. An internal channel $C_{ij} \in N$ connects a source process P_i to a destination process P_j. An input channel has no source process, and an output channel has no destination process in $\mathcal{N}$.

Process Definition and Semantics. A process P_i in the networked system is defined as a tuple (In_i, Out_i, M_i), where $In_i \subseteq (I \cup N)$ is the set of P_i's input channels, $Out_i \subseteq (O \cup N)$ is the set of P_i's output channels, and M_i is a machine describing P_i's input/output behavior. We assume a synchronous model of computation: (1) at each tick of the system, each process consumes an input symbol from each of its input channels, and produces an output symbol on each its output channels, and (2) message delivery through the channels is instantaneous. We further assume that a networked system $\mathcal{N}$ has a computation alphabet Σ for describing the inputs and outputs of each process, and for describing communication over the channels. Please see Fig. 2.1 for an example networked system. Observe that a process may communicate with one, many or all processes in $\mathcal{N}$ using its output channels. Thus our network model subsumes unicast, multicast and broadcast communication schemes.

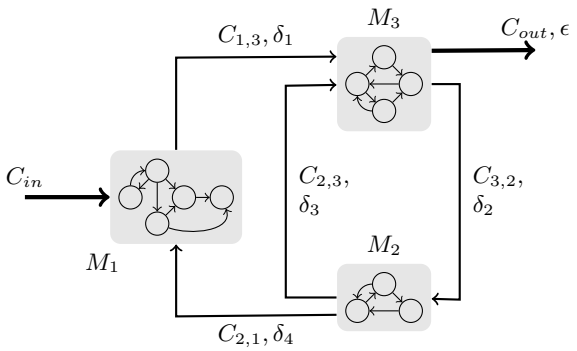

Fig. 2.1: Networked System

In this paper, we focus on processes described as Mealy machines. Recall that a Mealy machine [19] M is a deterministic finite-state transducer that in each

step, reads an input symbol, possibly changes state, and generates an output symbol. Formally, M is described as a tuple $(\Sigma_{in}, \Sigma_{out}, Q, q_0, R)$, where Σ_{in} and Σ_{out} are input and output alphabets respectively, Q is a finite, nonempty set of states, q_0 is an initial state, and $R \subseteq Q \times \Sigma_{in} \times \Sigma_{out} \times Q$ is the transition function.

The operational semantics of M is defined in terms of its run $\rho(s)$ on an input string $s = s[1] \dots s[m]$. A run is a sequence of the form (q_0, λ), $(q_1, s'[1])$, $\dots$, $(q_m, s'[m])$, where for each j, $1 \leq j \leq m$, $(q_{j-1}, s[j], s'[j], q'_j) \in R$. Such a run $\rho(s)$ of a Mealy machine defines the output function $[\![M]\!] : \Sigma^{\star}_{in} \to \Sigma^{\star}_{out}$, with $[\![M]\!](s[1].s[2] \dots s[m]) = s'[1].s'[2]. \dots .s'[m]$.

In each tick, a Mealy machine process in a networked system $\mathcal{N}$ consumes a composite symbol (the tuple of symbols on its input channels), and outputs a composite symbol (the tuple of symbols on its output channels). Thus, the input alphabet Σ_{in} for M_i is $\Sigma^{|In_i|}$, and the output alphabet Σ_{out} is $\Sigma^{|Out_i|}$. Let $(\Sigma^{|In_i|}, \Sigma^{|Out_i|}, Q_i, q_{0_i}, R_i)$ be the tuple describing the Mealy machine underlying process P_i.

Operational Semantics of a Network. We define a *network state* $\mathbf{q}$ as the tuple $(q_1, \dots, q_n, c_1, \dots, c_{|N|})$, where for each i, $q_i \in Q_i$ is the state of P_i, and for each k, c_k is the state of the k^{th} internal channel, i.e., the current symbol in the channel. A transition of $\mathcal{N}$ has the following form:

$$\begin{array}{l} (q_1, \dots, q_n, c_1, \dots, c_{|N|}) \\ \qquad\qquad \Big\downarrow (a_1, \dots, a_{|I|}), ({a'}_1, \dots, {a'}_{|O|}) \\ ({q'}_1, \dots, {q'}_n, {c'}_1, \dots, {c'}_{|N|}) \end{array}$$

Here $(a_1, \dots, a_{|I|})$ denote the symbols on the external input channels, and $({a'}_1, \dots, {a'}_{|O|})$ denote the symbols on the external output channels. During a transition of $\mathcal{N}$, each process P_i consumes a composite symbol (given by the states of all internal channels in In_i and the symbols in the external input channels in In_i), changes state from q_i to q'_i, and outputs a composite symbol. The generation of an output symbol by P_i causes an update to the states of all internal channels in Out_i and results in the output of a symbol on each output channel in Out_i.

Thus, we can view the networked system $\mathcal{N}$ itself as a machine that in each step, consumes an $|I|$-dimensional input symbol $\mathbf{a}$ from its external input channels, changes state according to the transition functions R_i of each process, and outputs an $|O|$-dimensional output symbol $\mathbf{a}'$ on its external output channels.

Formally, we define the semantics of a computation of $\mathcal{N}$ using the tuple $(\Sigma^{|I|}, \Sigma^{|O|}, Q, \mathbf{q}_0, R)$, where $Q = (Q_1 \times \dots \times Q_n \times \Sigma^{|N|})$ is the set of states and $R \subseteq (Q \times \Sigma^{|I|} \times \Sigma^{|O|} \times Q)$ is the network transition function. The initial state $\mathbf{q}_o = (q_{0_1}, \dots, q_{0_n}, c_{0_1}, \dots, c_{0_{|N|}})$ of $\mathcal{N}$ is given by the initial process states and internal channel states. An execution $\rho(\mathbf{s})$ of $\mathcal{N}$ on an input string $\mathbf{s} = \mathbf{s}[1]\mathbf{s}[2] \dots \mathbf{s}[m]$

is defined as a sequence of configurations of the form $(\mathbf{q}_0, \lambda)$, $(\mathbf{q}_1, \mathbf{s}'[1])$, ..., $(\mathbf{q}_m, \mathbf{s}'[m])$, where for each j, $1 \leq j \leq m$, $(\mathbf{q}_{j-1}, \mathbf{s}[j], \mathbf{s}'[j], \mathbf{q}_j) \in R$. The output function computed by the networked system $[\![\mathcal{N}]\!] : (\Sigma^{|I|})^\star \rightarrow (\Sigma^{|O|})^\star$ is then defined such that $[\![\mathcal{N}]\!](\mathbf{s}[1].\mathbf{s}[2] \ldots \mathbf{s}[m]) = \mathbf{s}'[1].\mathbf{s}'[2] \ldots \mathbf{s}'[m]$.

2.2 Channel Perturbations and Robustness

An execution of a networked system is said to be perturbed if one or more of the internal channels are perturbed one or more times during the execution. A channel perturbation can be modeled as a deletion or substitution of the current symbol in the channel. To model symbol deletions[4], we extend the alphabet of each internal channel to $\Sigma_\lambda = \Sigma \cup \lambda$. A perturbed execution includes transitions corresponding to channel perturbations, of the form:

$$\begin{array}{c} (q_1, \ldots, q_n, c_1, \ldots, c_{|N|}) \\ \Big\downarrow \boldsymbol{\lambda}, \boldsymbol{\lambda} \\ (q'_1, \ldots, q'_n, c'_1, \ldots, c'_{|N|}), \end{array}$$

Here, for each i, the states q'_i and q_i are identical, and for some k, $c_k \neq c'_k$. Such transitions, termed τ-transitions[5], do not consume any input symbol and model instantaneous channel errors. We say that the k^{th} internal channel is perturbed in a τ-transition if $c_k \neq c'_k$. We denote the perturbed version of the networked system $\mathcal{N}$ by $\mathcal{N}_\tau$. $\mathcal{N}_\tau$ is given by the tuple $(\Sigma^{|I|}, \Sigma^{|O|}, Q, \mathbf{q}_0, R_\tau)$, where R_τ includes R and all possible τ-transitions from each state. We require the set of states of $\mathcal{N}_\tau$ to be the same as that of $\mathcal{N}$, to enable $\mathcal{N}$ to proceed with a (unperturbed) transition from each network state resulting from a τ-transition. Thus, a perturbed network execution $\rho_\tau(\mathbf{s})$ on an input string $\mathbf{s} = \mathbf{s}[1]\mathbf{s}[2] \ldots \mathbf{s}[m]$ is a sequence of configurations $(\mathbf{q}_0, \boldsymbol{\lambda}), \ldots, (\mathbf{q}_\tau, \mathbf{s}'[m])$, where for any j either $(\mathbf{q}_{j-1}, \mathbf{s}[\ell], \mathbf{s}'[\ell], \mathbf{q}_j) \in R$ or $(\mathbf{q}_{j-1}, \boldsymbol{\lambda}, \boldsymbol{\lambda}, \mathbf{q}_j)$ is a τ-transition.

Note that there can be several possible perturbed executions of $\mathcal{N}$ on a string $\mathbf{s}$ which differ in their exact instances of τ-transitions and the channels perturbed in each instance. Each such perturbed execution generates a different perturbed output. For a specific perturbed execution $\rho_\tau(\mathbf{s})$ of the form $(\mathbf{q}_0, \boldsymbol{\lambda}), (\mathbf{q}_1, \mathbf{s}'[1]), \ldots, (\mathbf{q}_\tau, \mathbf{s}'[m])$, we denote the string $\mathbf{s}' = \mathbf{s}'[1].\mathbf{s}'[2] \ldots \mathbf{s}'[m]$ output by $\mathcal{N}$ along that execution by $[\![\rho_\tau]\!](\mathbf{s})$. We denote by $[\![\mathcal{N}_\tau]\!](\mathbf{s})$ the *set*

[4] Note that though a perturbation can cause a symbol on an internal channel to get deleted in a given step, we expect that the processes reading from this channel will output a nonempty symbol in that step. In this sense, we treat an empty input symbol simply as a special symbol, and assume that each process can handle such a symbol.

[5] Note that a network transition of the form $((q_1, \ldots, q_n, c_1, \ldots, c_{|N|}), \boldsymbol{\lambda}, \mathbf{a}', (q'_1, \ldots, q'_n, c'_1, \ldots, c'_{|N|}))$ where for some i, $q_i \neq q'_i$ is not considered a τ-transition: such a transition involves a state change by some process on an empty input symbol along with the generation of a nonempty output symbol.

of all possible perturbed outputs corresponding to the input string $\mathbf{s}$. Formally, $[\![\mathcal{N}_\tau]\!](\mathbf{s})$ is the set $\{\mathbf{s}' \mid \exists \rho_\tau(\mathbf{s}) \text{ s.t. } \mathbf{s}' = [\![\rho_\tau]\!](\mathbf{s})\}$.

Robustness. A distance metric $d : \Sigma^* \times \Sigma^* \to \mathbb{R}$ over a set Σ^* of strings is a function with the following properties: $\forall s, t, u \in \Sigma^*$: (1) $d(s,t) = 0$ iff $s = t$, (2) $d(s,t) = d(t,s)$, and (3) $d(s,u) \leq d(s,t) + d(t,u)$. Let d be such a distance metric over strings. We extend the metric to vectors of strings in the standard fashion. Let $\mathbf{w} = (w_1, \ldots, w_L)$ be a vector of strings; then $d(\mathbf{w}, \mathbf{v}) = (d(w_1, v_1), \ldots, d(w_L, v_L))$.

Let τ_k denote the number of perturbations in the k^{th} internal channel in $\rho_\tau(\mathbf{s})$. Then, the *channel-wise perturbation count* in $\rho_\tau(\mathbf{s})$, denoted $\|\rho_\tau(\mathbf{s})\|$ is given by the vector $(\tau_1, \ldots, \tau_{|N|})$. We define robustness of a networked system as follows.

Definition 2.1 (Robust networked system).
Given an upper bound $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_{|N|}\}$ on the number of possible perturbations in each internal channel, and an upper bound $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_{|O|})$ on the acceptable error in each external output channel of a networked system $\mathcal{N}$, we say that $\mathcal{N}$ is $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$-robust if:

$$\forall \mathbf{s} \in (\Sigma^{|I|})^\star, \forall \rho_\tau(\mathbf{s}) : \quad \|\rho_\tau(\mathbf{s})\| \leq \boldsymbol{\delta} \implies d(\ [\![\mathcal{N}]\!](\mathbf{s}), [\![\rho_\tau]\!](\mathbf{s})\ ) \leq \boldsymbol{\epsilon}$$

3 Distance Tracking Automata

The above formulation of the robustness problem is independent of the metric used to measure the distance between strings in the output channels. In this paper, we focus on distance metrics such as the Levenshtein distance and L_1-norm that are the most prevalent metrics used in practice to measure distances between strings. In Sec. 4, we show that the robustness problem with respect to each of these metrics is efficiently analyzable by reducing it to the problem of checking language emptiness of a suitably constructed *reversal-bounded counter machine*. But first, we briefly review reversal-bounded counter machines, as we use them extensively in the rest of the paper.

3.1 Review: Reversal-bounded Counter Machines [14, 15]

A (one-way, nondeterministic) h-counter machine $\mathcal{A}$ is a (one-way, nondeterministic) finite automaton, augmented with h integer counters. Let G be a finite set of integer constants (including 0). In each step, $\mathcal{A}$ may read an input symbol, perform a test on the counter values, change state, and increment each counter by some constant $g \in G$. A test on a set of integer counters $Z = \{z_1, \ldots, z_h\}$ is a Boolean combination of tests of the form $z\theta g$, where $z \in Z$, $\theta \in \{\leq, \geq, =, <, >\}$ and $g \in G$. Let $\mathcal{T}_Z$ be the set of all such tests on counters in Z.

Formally, $\mathcal{A}$ is defined as a tuple $(\Sigma_{in}, X, x_0, Z, G, E, F)$ where Σ_{in}, X, x_o, F, are the input alphabet, set of states, initial state, and final states respectively.

Z is a set of h integer counters, and $E \subseteq X \times (\Sigma_{in} \cup \lambda) \times \mathcal{T}_Z \times X \times G^{|Z|}$ is the transition relation. Each transition $(x, \sigma, t, x', g_1, \ldots, g_h)$ denotes a change of state from x to x' on symbol $\sigma \in \Sigma_{in} \cup \lambda$, with $t \in \mathcal{T}_Z$ being the enabling test on the counter values, and $g_k \in G$ being the amount by which the k^{th} counter is incremented.

A configuration of a one-way multi-counter machine is defined as the tuple $(x, \sigma, z_1, \ldots, z_h)$, where x is the state of the automaton, σ is a symbol of the input string being read by the automaton and $z_1, \ldots, z_h$ are the values of the counters. We define a move relation $\rightarrow_{\mathcal{A}}$ on the configurations: $(x, \sigma, z_1, \ldots, z_h) \rightarrow_{\mathcal{A}} (x', \sigma', z'_1, \ldots, z'_h)$ iff $(x, \sigma, t(z_1, \ldots, z_h), x', g_1, \ldots, g_h) \in E$, where, $t(z_1, \ldots, z_h)$ is *true*, $\forall k$: $z'_k = z_k + g_k$, and σ' is the next symbol in the input string being read. A path is a finite sequence of configurations $\mu_1 \ldots, \mu_m$ where for all $j : \mu_j \rightarrow_{\mathcal{A}} \mu_{j+1}$. A string $s \in \Sigma^\star_{in}$ is accepted by $\mathcal{A}$ if there exists a path from $(x_0, s_0, 0, \ldots 0)$ to $(x, s_j, z_1, \ldots, z_h)$ for some $x \in F$ and $j \leq |s|$. The set of strings (language) accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.

In general, multi-counter machines do not possess good algorithmic properties as they can simulate actions of Turing machines (even with just 2 counters). In [14], the author presents a class of counter machines that with certain restrictions on the counters possess efficiently decidable properties. We now briefly review these machines.

A counter is said to be in the increasing mode between two successive configurations if the value of the counter increases, and in the decreasing mode if the value of the counter decreases. We say that a counter *changes mode* if for (three) successive configurations, it goes from the increasing mode to the decreasing mode or vice versa. We say that a counter is *r-reversal bounded* if the maximum number of times it changes mode along *any* path is r. We say that a one-way multi-counter machine $\mathcal{A}$ is r-reversal bounded if each of its counters is at most r-reversal bounded. We denote the class of h-counter, r-reversal-bounded machines by NCM(h, r).

Lemma 3.1. *[12] The nonemptiness problem for a NCM(h, r) $\mathcal{A}$ can be solved in time polynomial in the size of $\mathcal{A}$.*

In Sec. 4, we show how we can algorithmically construct composite machines that can check robustness of networked systems. A key component of these constructions are machines that accept a pair of strings iff the two strings are more than ϵ distance apart according to the chosen metric. We now present the construction of a deterministic finite automaton (DFA) $\mathcal{D}^{\epsilon}_{Lev}$ that accepts a pair of strings iff their Levenshtein distance is greater than ϵ, followed by the construction of a reversal-bounded counter automaton $\mathcal{D}^{\epsilon}_{L_1}$ that accepts a pair of strings iff their L_1-norm is greater than ϵ. In what follows, we assume that for all $i > |s|$, $s_i = \texttt{\#}$, where \# is a special end-of-string symbol not in Σ. Let $\Sigma^{\texttt{\#}} = \Sigma \cup \{\texttt{\#}\}$.

3.2 Automaton for Tracking Levenshtein Distance

Levenshtein distance. The Levenshtein distance $d_{Lev}(s,t)$ between strings s and t is the minimum number of symbol insertions, deletions and substitutions required to transform one string into another. The Levenshtein distance, or edit distance, is also defined by the following recurrence relations, for $i, j \geq 1$, and $s[0] = t[0] = \lambda$:

$$\begin{array}{lll} d_{Lev}(s[0],t[0]) = 0, & d_{Lev}(s[0,i],t[0]) = i, & d_{Lev}(s[0],t[0,j]) = j \\ d_{Lev}(s[0,i],\ t[0,j]) = \min(& d_{Lev}\,(s[0,i\text{-}1],\ t[0,j\text{-}1]) + \texttt{diff}(s[i],\ t[j]), & \\ & d_{Lev}\,(s[0,i\text{-}1],\ t[0,j]) + 1, & \\ & d_{Lev}\,(s[0,i],\ t[0,j\text{-}1]) + 1) & \end{array} \tag{1}$$

Here, $\texttt{diff}(a,b)$ is defined to be 0 if $a = b$ and 1 otherwise. The first three relations, that involve empty strings, are obvious. The edit distance between the nonempty prefixes, $s[0,i]$ and $t[0,j]$, is the minimum of three distances: (1) the distance corresponding to editing $s[0,i\text{-}1]$ into $t[0,j\text{-}1]$ and substituting $s[i]$ for $t[j]$ if they are different symbols, (2) the distance corresponding to editing $s[0,i\text{-}1]$ into $t[0,j]$ and deleting $s[i]$, and, (3) the distance corresponds to editing $s[0,i]$ into $t[0,j\text{-}1]$ and inserting $t[j]$.

In [11], the authors show that for a given integer k, a relation $R \subseteq \Sigma^\star \times \Sigma^\star$ is rational if and only if for every $(s,t) \in R$, $|s| - |t| < k$. It is known from [10], that a subset is rational iff it is the behavior of a finite automaton. Thus, it follows from the above results that there exists a DFA that accepts the set of pairs of strings that are within bounded edit distance from each other. However, these theorems do not provide a constructive procedure for such an automaton. In what follows, we present a novel construction for a DFA $\mathcal{D}^{\epsilon}_{Lev}$ that accepts a pair of strings (s,t) iff $d_{Lev}(s,t) > \epsilon$.

The standard algorithm for computing the Levenshtein distance $d_{Lev}(s,t)$ uses a dynamic programming-based approach that uses the above recurrence relations. This algorithm organizes the bottom-up computation of the Levenshtein distance with the help of a table **tab** of height $|s|$ and width $|t|$. The 0^{th} row and column of **tab** account for the base case of the recursion. The $\mathbf{tab}(i,j)$ entry stores the Levenshtein distance of the strings $s[0,i]$ and $t[0,j]$. In general, the entire table has to be populated in order to compute $d_{Lev}(s,t)$. However, when one is only interested in some bounded distance ϵ, then for every i, the algorithm only needs to compute values for the cells from $\mathbf{tab}(i, i-\epsilon)$ to $\mathbf{tab}(i, i+\epsilon)$ [13]. We call this region the ϵ-*diagonal* of **tab**, and use this observation to construct the finite-state automaton $\mathcal{D}^{\epsilon}_{Lev}$.

The DFA $\mathcal{D}^{\epsilon}_{Lev}$ is defined to run on a pair of strings (s,t), and accept iff $d_{Lev}(s,t) > \epsilon$. In each step, $\mathcal{D}^{\epsilon}_{Lev}$ reads a pair of input symbols and changes state to mimic the bottom-up edit distance computation by the dynamic programming algorithm. We illustrate the operation of $\mathcal{D}^{\epsilon}_{Lev}$ with an example.

Example Run. A run of $\mathcal{D}^{\epsilon}_{Lev}$ on the string pair (s,t) that checks if $d_{Lev}(s,t) > \epsilon$, for $\epsilon = 2$ is shown in Fig. 3.1. After reading the i^{th} input symbol pair, $\mathcal{D}^{\epsilon}_{Lev}$ uses

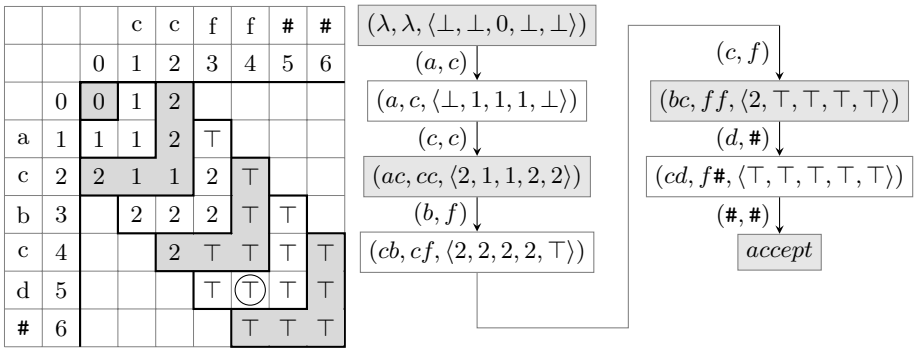

Fig. 3.1: Dynamic programming table emulated by $\mathcal{D}^{\epsilon}_{Lev}$. The table **tab** filled by the dynamic programming algorithm is shown to the left, and a computation of $\mathcal{D}^{\epsilon}_{Lev}$ on the strings $s = acbcd$ and $t = ccff$ is shown to the right. Here, $\epsilon = 2$.

its state to remember the last $\epsilon = 2$ symbols of s and t that it has read, and transitions to a state that contains the values of $\mathbf{tab}(i, i)$ and the cells within the ϵ-diagonal, above and to the left of $\mathbf{tab}(i, i)$.

Formally, $\mathcal{D}^{\epsilon}_{Lev}$ is defined as a tuple $(\Sigma^{\#} \times \Sigma^{\#}, Q_{Lev}, \mathbf{q}_{0\,Lev}, R_{Lev}, F_{Lev})$, where $(\Sigma^{\#} \times \Sigma^{\#})$, Q_{Lev}, $\mathbf{q}_{0\,Lev}$, R_{Lev}, F_{Lev} are the input alphabet, the set of states, the initial state, the transition relation and the set of final states respectively. $F_{Lev} = \{acc_{Lev}\}$ is a singleton set. In what follows, we define the other components.

We first note that as indicated earlier, $\mathcal{D}^{\epsilon}_{Lev}$ synchronously runs on a pair of strings, i.e., in each step it reads a symbol from $(\Sigma^{\#} \times \Sigma^{\#})$. We assume that each string is well-formed, i.e., each string is an element of $\Sigma^{*}.\#^{*}$. A state of $\mathcal{D}^{\epsilon}_{Lev}$ is defined as the tuple $(x, y, \mathbf{e})$, where x and y are strings of length at most ϵ and $\mathbf{e}$ is a vector containing $2\epsilon + 1$ entries, with at most $\epsilon + 3$ possible values for each entry. A state of $\mathcal{D}^{\epsilon}_{Lev}$ maintains the invariant that if i symbol pairs have been read, then $x = s[i\text{-}\epsilon\text{+}1, i]$, $y = t[i\text{-}\epsilon\text{+}1, i]$ and the entries in $\mathbf{e}$ correspond to the values $\{\mathbf{tab}(i, j) \mid j \in [i\text{-}\epsilon, i\text{-}1]\}$, $\{\mathbf{tab}(j, i) \mid j \in [i\text{-}\epsilon, i\text{-}1]\}$, and $\mathbf{tab}(i, i)$. The values in these cells greater than ϵ are replaced by $\top$. The initial state of $\mathcal{D}^{\epsilon}_{Lev}$ is $\mathbf{q}_{0\,Lev} = (\lambda, \lambda, \langle \bot, \ldots, \bot, 0, \bot, \ldots, \bot \rangle)$, where λ denotes the empty string, $\bot$ is a special symbol denoting an undefined value, and the value 0 corresponds to entry $\mathbf{tab}(0, 0)$.

Upon reading the i^{th} input symbol pair, the transition of $\mathcal{D}^{\epsilon}_{Lev}$ from state $q_{i\text{-}1}$ to q_i is as shown in Fig. 3.2. Note that to compute values in $\mathbf{e}$ corresponding to the i^{th} row, we need the substring $t[i\text{-}\epsilon, i\text{-}1]$, the values $\mathbf{tab}(i\text{-}1\text{-}\epsilon, i\text{-}1)$ to $\mathbf{tab}(i\text{-}1, i\text{-}1)$, and the symbol s_i. From the invariant on the state, it follows that the values of the required cells from **tab** and the required substring $t[i\text{-}\epsilon, i\text{-}1]$ are present in $q_{i\text{-}1}$ and the input symbol. Similarly, to compute $\mathbf{tab}(j, i)$, where $j \in [i\text{-}1\text{-}\epsilon, i]$ the string in y, values in $\mathbf{e}$ of q_{i-1} and the input symbol suffice. Thus, given any state of $\mathcal{D}^{\epsilon}_{Lev}$ and an input symbol pair, we can construct the unique next state that satisfies the state-invariant.

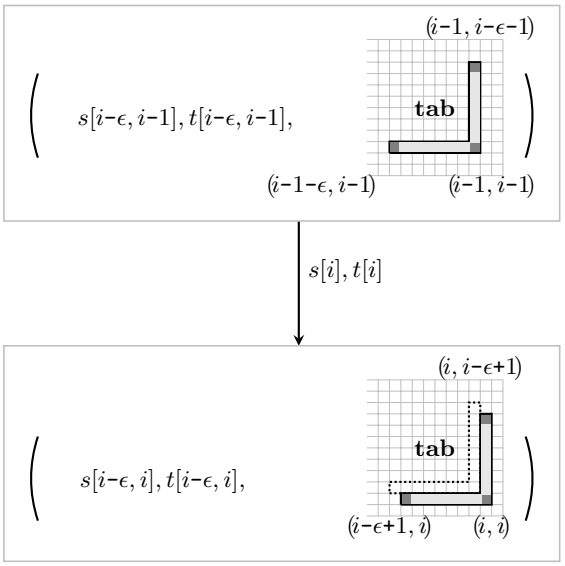

Fig. 3.2: A transition of $\mathcal{D}^{\epsilon}_{Lev}$

Recall that for strings s,t, the value of $d_{Lev}(s,t)$ is stored in the entry $\mathbf{tab}(|s|,|t|)$ of **tab**. Keeping this in mind, upon reading the symbol $(\#,\#)$, we add transitions to the accepting state acc_{Lev} iff:

- $|s| = |t|$, i.e., x and y do not contain #, and the $(\epsilon+1)^{th}$ entry in **e** is $\top$, or,
- $|s| = |t| + \ell$, i.e., y contains ℓ #'s, x contains no #, and the $(\epsilon+1-\ell)^{th}$ entry in **e** is $\top$, or,
- $|t| = |s| + \ell$, i.e., x contains ℓ #'s, y contains no #, and the $(\epsilon+1+\ell)^{th}$ entry in **e** is $\top$.

This shows how we can construct a $\mathcal{D}^{\epsilon}_{Lev}$ that exactly mimics the dynamic programming algorithm. The following lemma states the correctness of this construction. The proof follows from the state-invariant maintained by $\mathcal{D}^{\epsilon}_{Lev}$ and its acceptance condition.

Lemma 3.2. *$\mathcal{D}^{\epsilon}_{Lev}$ accepts a pair of strings (s,t) iff $d_{Lev}(s,t) > \epsilon$.*

3.3 Automaton for Tracking L_1-norm

The L_1-norm measures the number of positions in which two strings differ. As before, let $s[0] = t[0] = \lambda$. Formally, we define $d_{L_1}(s,t)$ using the following recurrence relations:

$$d_{L_1}(s[0],t[0]) = 0 \qquad d_{L_1}(s[0,j],t[0,j]) = d_{L_1}(s[0,j\text{-}1],t[0,j\text{-}1]) + \mathbf{diff}(s[j],t[j])$$

We now define the automaton $\mathcal{D}^{\epsilon}_{L_1}$ that accepts pairs of strings (s,t) such that $d_{L_1}(s,t) > \epsilon$. The automaton $\mathcal{D}^{\epsilon}_{L_1}$ is a 1-reversal-bounded 1-counter machine

(i.e., in NCM(1,1)), defined as a tuple $(\Sigma^{\#} \times \Sigma^{\#}, X_{L_1}, x_{0\,L_1}, Z, G_{L_1}, E_{L_1}, F_{L_1})$, where $(\Sigma^{\#} \times \Sigma^{\#})$ is its input alphabet, $X_{L_1} = \{x_{0\,L_1}, x_{L_1}, acc_{L_1}\}$, is a set of three states, $x_{0\,L_1}$ is the initial state, $Z = \{z\}$ is a single 1-reversal-bounded counter, $G_{L_1} = \{\epsilon, 0, -1\}$ is a set of integers, and $F_{L_1} = \{acc_{L_1}\}$ is the singleton set of final states. The transition relation contains the following types of transitions:

1. The transition $(x_{0\,L_1}, (\lambda, \lambda), \mathit{true}, x_{L_1}, \epsilon)$ is an initialization transition that sets the counter to ϵ.
2. The transition $(x_{L_1}, (a, a), z \geq 0, x_{L_1}, 0)$ keeps the state and counter of $\mathcal{D}^{\epsilon}_{L_1}$ unchanged upon reading a pair of the same symbols.
3. Transitions of the form $(x_{L_1}, (a, b), z > 0, x_{L_1}, -1)$, for $a \neq b$, decrement the counter by 1 upon reading a pair of distinct symbols. These transitions essentially count the number of differing positions of the two strings.
4. The transition $(x_{L_1}, (a, b), z = 0, acc_{L_1}, 0)$, for $a \neq b$, moves $\mathcal{D}^{\epsilon}_{L_1}$ to an accepting state when it finds the $(\epsilon + 1)^{th}$ differing position. This indicates that the L_1-norm between the strings being read is greater than ϵ.

Lemma 3.3. *$\mathcal{D}^{\epsilon}_{L_1}$ accepts a pair of strings (s, t) iff $d_{L_1}(s, t) > \epsilon$.*

Remark: The construction of $\mathcal{D}^{\epsilon}_{Lev}$ is significantly more involved than that of $\mathcal{D}^{\epsilon}_{L_1}$. This is perhaps clear from the difference in the complexity of the respective recurrence relations. Unlike the L_1-norm, for edit distance computation, it is not sufficient to focus on the positions of edits in each string. One must also obtain the optimal *alignment* or *matching* between strings s and t. For instance, the L_1-norm between the strings *shin* and *hind* is 4, while the edit distance is only 2 (delete s, align/match *hin*, insert d).

4 Analyzing Robustness of a Networked System

In this section, we present an automata-theoretic framework for checking robustness of a networked system in the presence of bounded channel perturbations. Checking if a networked system $\mathcal{N}$ is $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$-robust is equivalent to checking if, for each output channel $o_\ell \in O$ (with an error bound of ϵ_ℓ), $\mathcal{N}$ is $(\boldsymbol{\delta}, \epsilon_\ell)$-robust. Thus, in what follows, we focus on the problem of checking robustness of the networked system $\mathcal{N}$ for a single output channel. Rephrasing the robustness definition from before, we need to check if for all input strings $\mathbf{s} \in (\Sigma^{|I|})^{\star}.(\#^{|I|})^{\star}$, and all runs $\rho_\tau(\mathbf{s})$ of $\mathcal{N}$, $\|\rho_\tau(\mathbf{s})\| \leq \boldsymbol{\delta}$ implies that $d([\![\mathcal{N}]\!]|_\ell(\mathbf{s}), [\![\rho_\tau]\!]|_\ell(\mathbf{s})) \leq \epsilon_\ell$. Here, $[\![\mathcal{N}]\!]|_\ell(\mathbf{s})$, $[\![\rho_\tau]\!]|_\ell(\mathbf{s})$ respectively denote the projections of $[\![\mathcal{N}]\!](\mathbf{s})$ and $[\![\rho_\tau]\!](\mathbf{s})$ on the ℓ^{th} output channel. For simplicity in notation, henceforth, we drop the ℓ in the error bound on the channel, and denote it simply by ϵ.

In what follows, we define composite machines $\mathcal{A}$ that accept input strings certifying the non-robust behavior of a given networked system $\mathcal{N}$. In other words, $\mathcal{A}$ accepts a string $\mathbf{s} \in (\Sigma^{|I|})^{\star}.(\#^{|I|})^{\star}$ iff there exists a perturbed execution $\rho_\tau(\mathbf{s})$ of $\mathcal{N}_\tau$ such that: $\|\rho_\tau(\mathbf{s})\| \leq \boldsymbol{\delta}$ and $d([\![\mathcal{N}]\!]|_\ell(\mathbf{s}), [\![\rho_\tau]\!]|_\ell(\mathbf{s})) > \epsilon$. Thus, the networked system $\mathcal{N}$ is $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$-robust iff $\mathcal{L}(\mathcal{A})$ is empty.

4.1 Robustness Analysis for the Levenshtein Distance Metric

The composite machine $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$, certifying non-robustness with respect to the Levenshtein distance metric, is a nondeterministic 1-reversal-bounded $|N|$-counter machine, i.e., in the class NCM($|N|$,1). In each run on an input string $\mathbf{s}$, $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$ simultaneously does the following: (a) it simulates an unperturbed execution $\rho(\mathbf{s})$ of $\mathcal{N}$ and a perturbed execution $\rho_\tau(\mathbf{s})$ of $\mathcal{N}_\tau$, (b) keeps track of all the internal channel perturbations along $\rho_\tau(\mathbf{s})$, and (c) tracks the Levenshtein distance between the outputs generated along $\rho(\mathbf{s})$ and $\rho_\tau(\mathbf{s})$.

Similar to the semantics of a networked system $\mathcal{N}$ with multiple output channels, we can define the semantics of $\mathcal{N}$ for the ℓ^{th} output channel using the tuple $(\Sigma^{|I|}, \Sigma, Q, \mathbf{q}_0, R|_\ell)$. Here, $R|_\ell$ denotes the projection of the transition relation R of $\mathcal{N}$ onto the ℓ^{th} output channel. To incorporate the addition of # symbols at the end of strings, the semantics of $\mathcal{N}$ is further modified to the tuple $(\Sigma^{|I|} \cup \{\#^{|I|}\}, \Sigma^{\#}, Q, \mathbf{q}_0, R^{\#})$, where $R^{\#} = R|_\ell \cup \{(\mathbf{q}, ((\#, \ldots, \#), \#), \mathbf{q}) : \mathbf{q} \in Q\}$. Similarly, the tuple defining $\mathcal{N}_\tau$ is modified to $(\Sigma^{|I|} \cup \{\#^{|I|}\}, \Sigma^{\#}, Q, \mathbf{q}_0, R_\tau^{\#})$, where $R_\tau^{\#}$ includes $R^{\#}$ and all the τ-transitions from each state as before. Also recall from Sec. 3, that the automaton $\mathcal{D}_{Lev}^{\epsilon}$, accepting pairs of strings with edit distance greater than ϵ from each other, is defined by the tuple $((\Sigma^{\#} \times \Sigma^{\#}), Q_{Lev}, \mathbf{q}_{0\,Lev}, R_{Lev}, F_{Lev})$. Formally, $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$, in the class NCM($|N|$,1), is defined as the tuple $(\Sigma^{|I|} \cup \{\#^{|I|}\}, X, \mathbf{x}_0, Z, G, E, F)$, where $X, \mathbf{x}_0, Z, G, E, F$ are respectively the set of states, initial state, set of counters, a finite set of integers, the transition relation and the final states of $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$. We define these below.

The set of states $X = Y \cup \{\mathbf{acc}, \mathbf{rej}\}$, where $Y \subseteq (Q \times Q \times Q_{Lev})$. Each state $\mathbf{x} \in Y$ of $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$ is a tuple $(\mathbf{q}, \mathbf{r}, \mathbf{q}_{Lev})$, where the component labeled $\mathbf{q}$ tracks the state of the unperturbed network $\mathcal{N}$, the component $\mathbf{r}$ tracks the state of the perturbed network $\mathcal{N}_\tau$, and $\mathbf{q}_{Lev}$ is a state in $\mathcal{D}_{Lev}^{\epsilon}$.

The initial state of $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$, $\mathbf{x}_0$, is given by the tuple $(\mathbf{q}_0, \mathbf{q}_0, \mathbf{q}_{0\,Lev})$. The set of counters $Z = \{z_1, \ldots, z_{|N|}\}$ tracks the number of perturbations in each internal channel of $\mathcal{N}$. The initial value of each counter is 0. $G = \{0, -1, \delta_1, \delta_2, \ldots, \delta_{|N|}\}$ is the set of all integers that can be used in tests on counter values, or by which any counter in Z can be incremented. The set of final states is the singleton set $\{\mathbf{acc}\}$.

The transition relation E of $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$ is constructed using the following steps:

1. *Initialization transition*:
   From the initial state $\mathbf{x}_0$, we add a single transition of the form:

$$\left((\mathbf{q}_0, \mathbf{q}_0, \mathbf{q}_{0\,Lev}),\quad \boldsymbol{\lambda},\quad \bigwedge_k z_k = 0,\quad (\mathbf{q}_0, \mathbf{q}_0, \mathbf{q}_{0\,Lev}),\quad (+\delta_1, \ldots, +\delta_{|N|})\right)$$

   In this transition, $\mathcal{A}_{Lev}^{\boldsymbol{\delta},\epsilon}$ sets each counter z_k to the error bound δ_k on the k^{th} internal channel, without consuming an input symbol or changing state. Note that the counter test ensures that this transition can be taken only once from $\mathbf{x}_0$.

2. *Unperturbed network transitions*:
For each pair of transitions in $R^{\#}$ and $R^{\#}_{\tau}$ on the same input symbol from the same state, i.e., $(\mathbf{q}, \mathbf{a}, b, \mathbf{q}') \in R^{\#}$ and $(\mathbf{r}, \mathbf{a}, b', \mathbf{r}') \in R^{\#}_{\tau}$, and transitions of the form $(\mathbf{q}_{Lev}, (b, b'), \mathbf{q}'_{Lev}) \in R_{Lev}$, we add a transition of the following form to $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$:

$$\left((\mathbf{q}, \mathbf{r}, \mathbf{q}_{Lev}), \quad \mathbf{a}, \quad \bigwedge_k z_k \geq 0, \quad (\mathbf{q}', \mathbf{r}', \mathbf{q}'_{Lev}), \quad \mathbf{0} \right)$$

In each such transition, $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ consumes an input symbol $\mathbf{a} \in \Sigma^{|I|} \cup \{\#^{|I|}\}$ and simulates a pair of unperturbed transitions on $\mathbf{a}$ in the first two components of its state. The distance between the corresponding outputs of $\mathcal{N}$ (b and b' above) is tracked by the third component. Note that in such transitions, all counter values are required to be non-negative in the source state and are not modified.

3. *Perturbed network transitions*:
From each state $\mathbf{x} \in Y$, we add transitions of the form:

$$\left((\mathbf{q}, \mathbf{r}, \mathbf{q}_{Lev}), \quad \boldsymbol{\lambda}, \quad \bigwedge_k z_k \geq 0, \quad (\mathbf{q}, \mathbf{r}_{\tau}, \mathbf{q}_{Lev}), \quad \mathbf{g} \right)$$

In each such transition, $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ simulates a τ-transition of the form $(\mathbf{r}, \boldsymbol{\lambda}, \boldsymbol{\lambda}, \mathbf{r}_{\tau}) \in R^{\#}_{\tau}$. In the transition, $\mathbf{g}$ denotes a vector with entries in $\{0, -1\}$, where $g_k = -1$ iff the k^{th} internal channel is perturbed in $(\mathbf{r}, \boldsymbol{\lambda}, \boldsymbol{\lambda}, \mathbf{r}_{\tau})$. Thus, we model a perturbation on the k^{th} internal channel by decrementing the (non-negative) z_k counter of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$. Note that in these transitions, no input symbol is consumed, and the first and third components, i.e. $\mathbf{q}$ and $\mathbf{q}_{Lev}$ remain unchanged.

4. *Rejecting transitions*:
From each state $\mathbf{x} \in Y$, we add transitions of the form:

$$\left((\mathbf{q}, \mathbf{r}, \mathbf{q}_{Lev}), \quad \boldsymbol{\lambda}, \quad \bigvee_k z_k < 0, \quad \mathbf{rej}, \quad \mathbf{0} \right)$$

From the state $\mathbf{rej}$, for all $\mathbf{a} \in \Sigma^{|I|}$, we add a transition: $(\mathbf{rej}, \mathbf{a}, \mathit{true}, \mathbf{rej}, \mathbf{0})$.

We add a transition to a designated rejecting state whenever the value of some counter z_k goes below 0, i.e., whenever the perturbation count in some k^{th} internal channel exceeds the error bound $\boldsymbol{\delta}_k$. Once in the state $\mathbf{rej}$, $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ ignores any further input read, and remains in that state.

5. *Accepting transitions*:
Finally, from each state $(\mathbf{q}, \mathbf{r}, \mathit{acc}_{Lev}) \in Y$, we add transitions of the form:

$$\left((\mathbf{q}, \mathbf{r}, \mathbf{q}_{Lev}), \quad \boldsymbol{\lambda}, \quad \bigwedge_k z_k \geq 0, \quad \mathbf{acc}, \quad \mathbf{0} \right)$$

We add a transition to the unique accepting state whenever $\mathbf{q}_{Lev} = acc_{Lev}$ and $\bigwedge_k z_k \geq 0$. The first criterion ensures that $d([\![\mathcal{N}]\!]|_\ell(\mathbf{s}), [\![\rho_\tau]\!]|_\ell(\mathbf{s})) > \epsilon$ (as indicated by reaching the accepting state in $\mathcal{D}^{\epsilon}_{Lev}$). The second criterion ensures that $\|\rho_\tau(\mathbf{s})\| \leq \boldsymbol{\delta}$, i.e., the run $\rho_\tau(\mathbf{s})$ of $\mathcal{N}$ on $\mathbf{s}$ models perturbations on the network that respect the internal channel error bounds.

Theorem 4.1. *Given an upper bound $\boldsymbol{\delta}$ on the number of perturbations in the internal channels, and an upper bound ϵ on the acceptable error for a particular output channel, the problem of checking if the networked system $\mathcal{N}$ is $(\boldsymbol{\delta}, \epsilon)$-robust with respect to the Levenshtein distance is polynomial in the network states $|Q|$, perturbed network transitions $|R^{\#}_\tau|$ and $\boldsymbol{\delta}$, and is $\mathcal{O}(\epsilon^\epsilon)$.*

Proof. We first note that the construction of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ reduces the problem of checking $(\boldsymbol{\delta}, \epsilon)$-robustness of $\mathcal{N}$ (w.r.t. the Levenshtein distance) to checking emptiness of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$. As $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ belongs to the class NCM($|N|$, 1) from Lemma 3.1, we know that checking emptiness of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ is polynomial in the size of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$, which includes the states, transitions, counters and the set G of integer constants of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$. The complexity then follows from the constructions of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ and $\mathcal{D}^{\epsilon}_{Lev}$.

4.2 Robustness Analysis for the L_1-norm Distance Metric

The composite machine $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$ certifying non-robustness with respect to the L_1-norm metric, is a nondeterministic, 1-reversal-bounded $(|N|+1)$-counter machine, i.e., in the class NCM($|N|+1$, 1). Similar to $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$, the machine $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$ also simultaneously simulates an unperturbed execution of $\mathcal{N}$ and perturbed execution of $\mathcal{N}_\tau$, while tracking the L_1-norm between the outputs generated along both the runs.

Recall from Sec. 3, that the automaton $\mathcal{D}^{\epsilon}_{L_1}$, accepting pairs of strings with L_1-norm greater than ϵ from each other, is in the class NCM(1,1), and is defined by the tuple $(\Sigma^{\#} \times \Sigma^{\#}, X_{L_1}, x_{0\,L_1}, Z, G_{L_1}, E_{L_1}, F_{L_1})$. Formally, the machine $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$, in the class NCM($|N|+1$, is defined as the tuple $(\Sigma^{|I|} \cup \{\#^{|I|}\}, X, \mathbf{x}_0, Z, G, E, F)$, where all components have their usual meaning. The set of states $X = Y \cup \{\mathbf{acc}, \mathbf{rej}\}$, where $Y \subseteq (Q \times Q \times X_{L_1})$. The initial state $\mathbf{x}_0$ is $(\mathbf{q}_0, \mathbf{q}_0, x_{0\,L_1})$. The set of counters $Z = \{z_1, \ldots, z_{|N|}\} \cup \{z\}$, where z is an additional counter used to track the L_1-norm for the output strings. The set $G = \{0, -1, \delta_1, \delta_2, \ldots, \delta_{|N|}, \epsilon\}$, the set F of final states is the singleton set $\{\mathbf{acc}\}$.

We add transitions to E in a step-wise fashion similar to that for $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$:

1. *Initialization transition*:
   We add a single transition of the form:

$$\left((\mathbf{q}_0, \mathbf{q}_0, x_{0\,L_1}),\ \boldsymbol{\lambda},\ \bigwedge_k z_k = 0 \wedge z = 0,\ (\mathbf{q}_0, \mathbf{q}_0, x_{0\,L_1}),\ (+\delta_1, \ldots, +\delta_{|N|},\ +\epsilon) \right)$$

   In addition to initializing the counters for tracking the internal channel error bounds, this transition also initializes the counter for tracking the L_1-norm of the output strings.

2. *Unperturbed network transitions*:
For each pair of transitions in $R^{\#}$ and $R^{\#}_{\tau}$ from the same state, with the same input symbol and output symbol, i.e., $(\mathbf{q}, \mathbf{a}, b, \mathbf{q}') \in R^{\#}$ and $(\mathbf{r}, \mathbf{a}, b, \mathbf{r}') \in R^{\#}_{\tau}$, and transitions of the form $(x_{L_1}, (b, b), z \geq 0, x_{L_1}, 0)$ *in* E_{L_1}, we add a transition of the following form to $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$:

$$\Big((\mathbf{q}, \mathbf{r}, x_{L_1}), \quad \mathbf{a}, \quad \bigwedge z_k \geq 0 \wedge z \geq 0, \quad (\mathbf{q}', \mathbf{r}', x_{L_1}), \quad (0, \ldots, 0, 0)\Big).$$

For each pair of transitions in $R^{\#}$ and $R^{\#}_{\tau}$ from the same state, with the same input symbol and different output symbols, i.e., $(\mathbf{q}, \mathbf{a}, b, \mathbf{q}') \in R^{\#}$ and $(\mathbf{r}, \mathbf{a}, b', \mathbf{r}') \in R^{\#}_{\tau}$, and transitions of the form $(x_{L_1}, (b, b'), z > 0, x_{L_1}, -1)$ *in* E_{L_1}, we add a transition of the following form to $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$:

$$\Big((\mathbf{q}, \mathbf{r}, x_{L_1}), \quad \mathbf{a}, \quad \bigwedge z_k \geq 0 \wedge z > 0, \quad (\mathbf{q}', \mathbf{r}', x_{L_1}), \quad (0, \ldots, 0, -1)\Big).$$

For each pair of transitions in $R^{\#}$ and $R^{\#}_{\tau}$ from the same state, with the same input symbol and different output symbols, i.e., $(\mathbf{q}, \mathbf{a}, b, \mathbf{q}') \in R^{\#}$ and $(\mathbf{r}, \mathbf{a}, b', \mathbf{r}') \in R^{\#}_{\tau}$, and transitions of the form $(x_{L_1}, (b, b'), z = 0, acc_{L_1}, 0)$ *in* E_{L_1}, we add transitions of the following form to $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$:

$$\Big((\mathbf{q}, \mathbf{r}, x_{L_1}), \quad \mathbf{a}, \quad \bigwedge z_k \geq 0 \wedge z = 0, \quad (\mathbf{q}', \mathbf{r}', acc_{L_1}), \quad (0, \ldots, 0, 0)\Big).$$

The perturbed network transitions, rejecting transitions, and accepting transitions are added in a similar fashion to $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$, (substitute $\mathbf{q}_{Lev}$ in all transitions for $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{Lev}$ by x_{L_1}).

Theorem 4.2. *Given an upper bound $\boldsymbol{\delta}$ on the number of perturbations in the internal channels, and an upper bound ϵ on the acceptable error for a particular output channel, the problem of checking if the networked system $\mathcal{N}$ is $(\boldsymbol{\delta}, \epsilon)$-robust with respect to the L_1-norm is polynomial in the network states $|Q|$, perturbed network transitions $|R^{\#}_{\tau}|$, $\boldsymbol{\delta}$ and ϵ.*

Proof. We note that the construction of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$ reduces the problem of checking $(\boldsymbol{\delta}, \epsilon)$-robustness for $\mathcal{N}$ (w.r.t. the L_1-norm) to checking emptiness of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$. As $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$ belongs to the class NCM($|N|$+1,1), from Lemma 3.1, we know that checking emptiness of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$ is polynomial in the size of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$, which includes the states, transitions, counters and the set G of integer constants of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$. The complexity then follows from the constructions of $\mathcal{A}^{\boldsymbol{\delta},\epsilon}_{L_1}$ and $\mathcal{D}^{\epsilon}_{L_1}$.

5 Related Work

There is a growing interest in the study of robustness in the formal methods and software engineering communities. The initial papers by Majumdar and Saha [17] and by Chaudhuri et al [5–7] consider robustness of infinite-state programs. The

programs considered in these papers are essentially functional; their scope does not extend to concurrent systems with channel errors like ours.

More recent papers have aimed to develop a notion of robustness for reactive systems. In [22], the authors propose a comprehensive notion of input-output stability of finite-state transducers that bounds both the deviation of the output from disturbance-free behaviour under bounded disturbance, as well as the persistence of the effect on the output of a sporadic disturbance. The deviations are measured using cost functions that map strings to nonnegative integers. The authors present polynomial-time algorithms for the analysis and synthesis of robust transducers. Exploring extensions of techniques presented in our paper to address persistence of a sporadic disturbance would be interesting.

In [18, 4, 2], the authors develop different notions of robustness for reactive systems, with ω-regular specifications, interacting with uncertain environments. In [18], the authors present metric automata, which are automata equipped with a metric on states. The authors assume that at any step, the environment can perturb any state q to a state at most $\gamma(q)$ distance away, where γ is some function mapping states to real numbers. A winning strategy for a finite-state or Büchi automaton $\mathcal{A}$ is a strategy that satisfies the corresponding acceptance condition (stated as reachability of states in F or as infinitely often visiting states in F respectively). Such a winning strategy is defined to be σ-robust if it is a winning strategy for $\mathcal{A}$ where the set F' characterizing the acceptance condition includes all states at most $\sigma.\sup_{q \in F}\gamma(q)$ distance away from the F. We note that while there are some similarities in how a disturbance is modeled, our approach is quite different, as we quantify and analyze the effect of errors over time, and do not associate metrics with individual states.

In [8], the authors study robustness of sequential circuits w.r.t. a *common suffix distance* metric. Their notion of robustness essentially bounds the persistence of the effect of a sporadic disturbance in the input of a sequential circuit. To be precise, a circuit is said to be robust iff the position of the last mismatch in any pair of output sequences is a bounded number of positions from the last mismatch position in the corresponding pair of input sequences. The authors present a polynomial-time algorithm to decide robustness of sequential circuits modeled as (single) Mealy machines. The metric and its subsequent treatment developed in this paper is useful for analyzing circuits; however, for networked systems communicating via strings, metrics such as edit distance and the L_1-norm provide a more standard way to measure the effect of errors.

In [9], the authors present modeling techniques for cyber-physical systems. Further, the authors also discuss the challenges of including a network in a cyber-physical system. A key observation is that to maintain discrete-event semantics of components in such a system, it is important to have a common sense of time across all components. A critical requirement in such systems is that the communication remain synchronized, which is typically fulfilled by using protocols that bound the allowed drift in the value of the global clock. In our model, we do not analyze such details, and abstract them away, assuming that some underlying protocol ensures synchronous communication.

Work in the area of robust control seeks to analyze and design networked control systems where communication between sensors, the controller, and actuators occurs over unreliable networks such as wireless networks [1]. On the other hand, work on wireless control networks [20, 21] focuses on design of distributed controllers where the components of the controller communicate over unreliable wireless networks. In such applications, robustness typically means desirable properties of the control loop such as stability. We note that these papers typically assume a synchronous communication schedule as supported by wireless industrial control protocols such as ISA 100 and WirelessHART.

6 Discussion

We have presented a framework for the analysis of robustness of networked systems in the presence of bounded channel perturbations. There are a few directions in which this framework can be developed further. The first is a more extensive treatment of distance metrics. We observe that the symbol sequences (in Σ^*) in a networked cyber-physical system could represent a wide range of digital signals. To accurately model the deviation of such signals in an error-prone network from their error-free counterparts, one must track the *magnitude* of the signals. This necessitates defining and computing distances that are based on mapping individual symbols or symbol sequences to numbers [22].

The second direction is a generalization of the error model and subsequently, the robustness definition. In this work, we only focus on internal channel errors in a network, and assume that the input and output channels are error-free. However, in a real-world scenario, there can be multiple sources of uncertainty such as sensor and actuator noise, modeling errors and process failures. A comprehensive robustness analysis should thus check if small disturbances in the inputs or internal channels or processes result in small deviations in the system behaviour.

Finally, we also wish to investigate the extension of our current techniques to the *design* of robust networks.

References

1. Alur, R., D'Innocenzo, A., Johansson, K.H., Pappas, G.J., Weiss, G.: Compositional Modeling and Analysis of Multi-Hop Control Networks. IEEE Transactions on Automatic Control 56(10), 2345–2357 (2011)
2. Bloem, R., Greimel, K., Henzinger, T., Jobstmann, B.: Synthesizing Robust Systems. In: Proceedings of Formal Methods in Computer Aided Design (FMCAD). pp. 85–92 (2009)
3. Boussinot, F., De Simone, R.: The ESTEREL language. Proceedings of the IEEE 79(9), 1293–1304 (1991)
4. Cerny, P., Henzinger, T., Radhakrishna, A.: Simulation Distances. In: Proceedings of CONCUR. pp. 253–268 (2010)
5. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity Analysis of Programs. In: Proceedings of Principles of Programming Languages (POPL). pp. 57–70 (2010)

6. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity and Robustness of Programs. Communications of the ACM (2012)
7. Chaudhuri, S., Gulwani, S., Lublinerman, R., Navidpour, S.: Proving Programs Robust. In: Proceedings of Foundations of Software Engineering. pp. 102–112 (2011)
8. Doyen, L., Henzinger, T.A., Legay, A., Ničković, D.: Robustness of Sequential Circuits. In: Proceedings of Application of Concurrency to System Design (ACSD). pp. 77–84 (2010)
9. Eidson, J.C., Lee, E.A., Matic, S., Seshia, S.A., Zou, J.: Distributed Real-Time Software for Cyber-Physical Systems. Proceedings of the IEEE (special issue on CPS) 100(1), 45–59 (2012)
10. Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press New York (1974)
11. Frougny, C., Sakarovitch, J.: Rational Relations with Bounded Delay. In: Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS). pp. 50–63 (1991)
12. Gurari, E.M., Ibarra, O.H.: The Complexity of Decision Problems for Finite-Turn Multicounter Machines. In: Proceedings of the International Colloquium on Automata Languages and Programming (ICALP). pp. 495–505 (1981)
13. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
14. Ibarra, O.H.: Reversal-Bounded Multicounter Machines and Their Decision Problems. Journal of the ACM 25(1), 116–133 (1978)
15. Ibarra, O.H., Su, J., Dang, Z., Bultan, T., Kemmerer, R.A.: Counter Machines: Decidable Properties and Applications to Verification Problems. In: Proceedings of Mathematical Foundations of Computer Science (MFCS). pp. 426–435 (2000)
16. Kahn, G.: The Semantics of Simple Language for Parallel Programming. In: IFIP Congress. pp. 471–475 (1974)
17. Majumdar, R., Saha, I.: Symbolic Robustness Analysis. In: 30th IEEE Real-Time Systems Symposium. pp. 355–363 (2009)
18. Majumdar, R., Render, E., Tabuada, P.: A Theory of Robust Software Synthesis. CoRR abs/1108.3540 (2011)
19. Mealy, G.H.: A Method for Synthesizing Sequential Circuits. Bell Systems Technical Journal pp. 1045–1079 (1955)
20. Pajic, M., Sundaram, S., Pappas, G.J., Mangharam, R.: The Wireless Control Network : A New Approach for Control Over Networks. IEEE Transactions on Automatic Control 56(10), 2305–2318 (2011)
21. Pappas, G.J.: Wireless Control Networks: Modeling, Synthesis, Robustness, Security. In: Proceedings of Hybrid Systems: Computation and Control (HSCC). pp. 1–2 (2011)
22. Tabuada, P., Balkan, A., Caliskan, S.Y., Shoukry, Y., Majumdar, R.: Input Output Stability for Discrete Systems. In: Proceedings of International Conference on Embedded Software (EMSOFT) (2012)