Self-Organizing Code-Level Redundancy for Networking Protocols

Thomas Meyer and Christian Tschudin

Technical Report CS-2010-003
University of Basel
Aug 1st, 2010

Abstract

Fault tolerant systems are usually built around redundant elements controlled by a central observer that decides which of the elements provide the correct results and thus are healthy. Nature lacks such dedicated controllers; instead, proliferation of "good" results is an emergent phenomenon achieved through homeostasis – the intrinsic self-regulation in order to maintain a stable, healthy state.

We report about a methodology to design distributed homeostatic software systems that are dynamically stable and robust in execution. By continuously replicating its own code base, our software is able to thwart unreliable execution and even accidental code changes. The crucial part is to build the system such that it regulates its replication and thus organizes its own redundancy. This is achieved in an execution environment that mimics chemical kinetics in which we implement self-rewriting programs (Quines).

We analyze the robustness of these self-healing programs using phase-type distribution and apply other tools, developed for chemical reaction network analysis, which can now be applied to software execution and networking protocol analysis. We also demonstrate the practical use of our model with a link load balancing protocol.

Keywords:
code replication, robustness, phase-type distribution, artificial chemistry, unconventional computing.

Computer Science Department, Computer Networks Group, Bernoullistrasse 16, CH – 4056 Basel Switzerland. Phone: +41 61 267 05 55 Fax: +41 61 267 05 59 URL: http://cn.cs.unibas.ch/

1 Introduction

Natural systems are able to dynamically construct redundancy by assembling and reproducing their components. Often, components exist in several copies (flocks, but also blood or nerve cells), exploiting parallelism and minimizing the impact of the loss of a single item. For singular components (e.g. bones) and in order to fight the problem of aging, redundancy is achieved over time through procreation, yielding a new and possibly modified copy. In computer science however, software is considered to be static (and without wear). This view is recent: Back in the 1940's, von Neumann (1966) developed a theory of self-reproducing automata. He described a universal constructor, a machine able to produce a copy of any other machine whose soft- and hardware blueprint is provided as input. Being universal, the constructor is also able to generate a copy of itself.

Considerable research on self-replication was carried out on the framework of Cellular Automata (CA), in which remarkable results were achieved, also in terms of robustness and self-repair (Tempesti, Mange, & Stauffer, 1998). However, these results are hard to transfer from CA to the world of today's computer software. In the 1960s, with the desire to understand the fundamental information-processing principles and algorithms involved in self-replication, researchers started to focus on self-replicating code, how textual computer programs are able to replicate independent from their physical realization. The existence of self-replicating programs is a consequence of Kleene's second recursion theorem (1938), which states that for any program P there exists a program P', which generates its own encoding and passes it to P along with the original input. The simplest form of a self-replicating program is a "Quine", named after the philosopher and logician Willard van Orman Quine, and made popular by Hofstadter (1979): A Quine is a program that prints its own code. Quines exist for any programming language that is Turing complete and it is a common challenge for students to come up with a Quine in their language of choice. The *Quine Page* provides a comprehensive list of such programs in various languages (Thompson, 2010).

1.1 Contribution

In this work we put Quines in a parallel execution environment, permitting an ensemble of Quine copies to achieve surprising robustness with respect to code and packet loss and even execution errors. Our contribution consists in the demonstration of an operational system based on Quines that runs highly reliable network services with provable dynamic properties. More precisely, we will introduce an artificial chemistry embodied as interconnected "molecule vessels" in which we place carefully crafted self-replicating programs. Packets, or "molecules", react with each other and produce new packets, thus executing the program. Useful computations are piggybacked to the Quine structures in order to implement the network services. Due to the special scheduling of the reactions in the artificial chemistry according to the "law of mass action" in real chemistry, our system inherits the dynamic properties from chemistry such that we can apply the related analysis tools that were developed in the past two centuries. The law of mass action links the microscopic (scheduling) events with the observable behavior at macro scale. Using perturbation analysis we can then proceed in identifying equilibria and their stability.

1.2 Structure of This Report

We present our work along the following argumentation path: After a having highlighted the context of our approach and related work, we proceed in Section 3 with introducing the "style" of implementing network services that we call *chemical networking protocols*. Section 4 presents "chemical Quines" and we extensively study their long-term stability, both in a single node (Section 5) as well as in a distributed setting (Section 6). In Section 8 we show how the Quines can be used to design networking protocols: we present a link load balancing service for which we show its resilience to packet *and* code loss. Section 9 highlights future work before we conclude in Section 10.

2 Context and Related Work

In this section, we reference the relevant corner stones for our work where we could draw important insights, namely fault tolerance, autonomous computing, self-reproduction, artificial chemistries and their dynamics. The programming language "Fraglets", which we have used to implement our system, was described by Tschudin (2003) and is summarized in Section 3.2.

2.1 From Fault Tolerant to Self-Healing Systems

With the trend of increasing complexity, particularly accelerated by distributing computation, it became more intricate to control computer systems. They are equipped with human-programmed strategies to interact with the environment aiming at correct deterministic behavior. Such traditionally designed systems are brittle and vulnerable to errors, attacks, and even to small perturbation in the environmental conditions (Simon, 1996).

Autonomic Systems

The research branch of *Autonomic Systems* seeks for methods where human-made artifacts are nevertheless able to autonomously and without human assistance adapt themselves to different environments. One big driver of this research was IBM's *Autonomic Computing Manifesto* (IBM, n.d.; see also Kephart & Chess, 2003), in which an autonomic computer system is defined as a system which

- knows itself and comprise components that also posses a system identity,
- knows its environment and the context surrounding its activity, and acts accordingly,
- configures and reconfigures itself,
- optimize itself,
- perform something akin to healing,
- protect itself, and
- anticipate but hide its complexity.

The central life-like so called "self-$*$" properties were a challenge for many researches but were anticipated as necessary requirements to master the complexity of the future computer systems.

Self-Healing Systems

Self-healing is one the key requirements, whereby a system has "to be able to recover from routine and extraordinary events that might cause some of its parts to malfunction" (IBM, n.d.). Similarly, Ghosh, Sharman, Rao, and Upadhyaya (2007) define self-healing as a

> property that enables a system to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normalcy.

Today, fault tolerance is the method of choice to achieve robust and resilient services by building up redundancy in order to mask errors (Johnson, 1996; Pradhan, 1996; Wilfredo, 2000): Multiple identical or similar redundant systems are performing the same task. The result is compared, often by a centralized observer. The same architecture is used by most self-healing systems: a central entity monitors the system, detects deviations from normalcy and undertakes repair actions in order to restore the system's health (Dobson et al., 2006). This architecture inevitably leads to the problem that the central decision maker may also be error prone, requiring an observer of the observer, and so on, leading to infinite regression. Hence, we need a solution where the central observer that steers the redundancy is redundant too and is blended into the system.

Autonomic Communication

Researchers in the domain of networking and distributed systems seem to take up novel bio-inspired approaches much faster than the remaining computer science community. This is because the open distributed environment is much more error-prone than an isolated machine. Dobson et al. (2006) provide a survey of research carried out in the field of *Autonomic Communications*. Ideas range from swarm intelligence (White & Pagurek, 1998) or ant colony paradigms (Di Caro & Dorigo, 1998) to reinforcement learning as in the distributed neural network approach (Gelenbe, Lent, & Xu, 2001).

Unreliable Hardware Platforms

While autonomic communication masters network faults, there are little results how to design self-healing systems that cope with hardware faults caused by Single Event Upsets (SEU) such as spontaneous bit alterations due to electrical noise and cosmic radiation. Such faults will increase with the higher package density of future chips: systems become more susceptible to spurious execution errors. This not only lowers the reliability of calculations (soft errors), it also leads to disruptions of the program flow and unrecoverable program exceptions. The promise to reduce the production cost and energy consumption by *probabilistic chips* (Chakrapani, Korkmaz, Akgul, & Palem, 2007) leads to a further accentuation of this problem.

The current established approach is to add more hardware redundancy to mask this problem. Reis, Chang, and August (2007) proposes to duplicate code, which is executed in parallel, and to insert checkpoints where the two execution strands are compared. This method is able to catch about 90 % of all faults, but is still vulnerable to deadlocks.

Shaw (2002) argues that the reason for the vulnerability comes from static and deterministic requirements for the target systems and propagates to soften the precision to avoid brittleness.

Systems should rather exhibit *homeostatic* behavior, they should "maintain a stable internal environment despite external variations". Indeed, Wong and Horowitz (2006) showed that probabilistic inference algorithms are less vulnerable to soft errors.

We are aiming at a software that it able to run on unreliable hardware and that organizes its own health. In most programming paradigms available today it is hardly possible to monitor a program's health and recover from a deviation from the expected behavior. Moreover, in order to avoid having a central observer we require that the program monitors and repairs itself. Repairing implies that the program is able to modify its own code, for which a sound theoretical framework is still missing (Anckaert, Madou, & de Bosschere, 2007).

2.2 Self-Replication

The concept of self-replicating and self-modifying code exists since the dawn of computer science, but was seldom used for practical applications. Since the work of von Neumann (1966), many variants of universal constructors for self-reproduction have been proposed and elaborated. For an overview, see Freitas Jr. And Merkle (2004) or Sipper (1998). Langton (1984) argued that natural systems are lacking a universal constructor and relaxed the requirement that self-replicating structures must be equipped with a universal constructor. Instead, self-replication may arise from dynamic loops instead of static tapes; the information necessary to replicate the structure may be distributed in this loop and may not be present in explicit and distinct entities of a passive, un-interpreted blueprint and its active version of interpreted instructions. This allows information to be spread across different entities, which makes the system less vulnerable to errors. This observation lead to a new surge of research on such self-replicating structures (Perrier, Sipper, & Zahnd, 1996; Sipper, 1998).

2.3 Artificial Chemistry

Artificial chemical computing models (Banâtre, Fradet, & Radenac, 2006; Calude & Paŭn, 2001; Dittrich, 2005; Holland, 1992; Paŭn, 2000) express computations as chemical reactions that consume and produce objects (data or code). Objects are represented as elements in a multiset, which is an unordered set within which elements may occur more than once.

Dittrich, Ziegler, and Banzhaf (2001) classified chemical computing as applications of Artificial Chemistry, a branch of Artificial Life (ALife) dedicated to the study of the chemical processes related to life and organizations in general. In the same way as ALife seeks to understand life by building artificial systems with simplified life-like properties, Artificial Chemistry builds simplified abstract chemical models that nevertheless exhibit properties that may lead to emergent phenomena, such as the spontaneous organization of molecules into self-maintaining structures (Dittrich & Speroni di Fenizio, 2007; Fontana & Buss, 1994).

The applications of artificial chemistries go beyond ALife, reaching biology, information processing (in the form of natural and artificial chemical computing models) and evolutionary algorithms for optimization, among other domains. Chemical models have also been used to express replication, reproduction and variation mechanisms (Dittrich & Banzhaf, 1998; Dittrich et al., 2001; Hutton, 2002; Teuscher, 2007; Yamamoto, Schreckling, & Meyer, 2007).

2.4 Chemical Kinetics

The dynamics of natural chemical reactions is governed by the law of mass action (Abrash, 1986) which states that the reaction rate is proportional to the reactant concentration. Several algorithms have been proposed to simulate chemical reactions on the microscopic level: Gillespie (1977) described an exact stochastic simulation algorithm that accurately mimics the randomness of reactions, which stems from the Brownian motion of the colliding molecules. Several variants and improvements of this algorithm have been proposed since then (Gibson & Bruck, 2000; Gillespie, 2007).

Originally, the aim of these algorithms was to simulate real chemical reactions. In this work, we use them as scheduling algorithms for our chemical programs. As a consequence, program execution is an inherently stochastic process; there is no guarantee, which reaction will be executed next and when. However, since on the macroscopic time scale these algorithms simulate the law of mass action, the average dynamic behavior can be described by the same Ordinary Differential Equations (ODEs) that are used to deterministically approximate real chemical reactions.

A novel element of our work is the use of hard limits to an artificial chemical vessel's capacity. Environments with limited resources that host replicating entities lead to natural selection, as was shown by research on population dynamics by Fernando and Rowe (2007), Stadler, Fontana, and Miller (1993), Szathmáry (1991). In our case, the population consists of software components: Healthy software survives whereas errors are displaced. This naturally leads to software homeostasis – the intrinsic self-regulation of code in order to maintain a stable, healthy state.

3 Chemical Networking Protocols

Traditionally, protocol execution is handled by a state machine that upon the reception of a packet synchronously changes its internal state and performs some communication activity. Here we introduce a "molecule metaphor" where each packet is treated as a virtual molecule. Virtual molecules react with other molecules in a reaction vessel (node). A reaction may produce other molecules being delivered to the application or being sent over the network. In such a chemical perspective, we obtain a web of reactions that together perform a distributed computation (called network service).

3.1 Modeling Chemical Communication

Instead of encoding a deterministic state machine, or having a sequential program that processes an incoming packet, each network node contains a finite multiset $\mathcal{M}(\mathcal{S})$ of molecules $\mathcal{S} = \{s_1, \ldots, s_n\}$ (=packets). In addition, each node defines a set of reaction rules $\mathcal{R} = \{r_1, \ldots, r_m\}$ expressing which reactant molecules can collide and which molecules are generated during this process. Such a reaction is typically represented as

$$\mathrm{C}_i + \mathrm{X}_i \longrightarrow \mathrm{C}_i + \mathrm{X}_j \tag{1}$$

The above reaction in node i consumes, if present, two molecules C and X from the local multiset, regenerates C and sends molecule X to neighbor node j. In a simple two-node network topology,

the above example spans the following reaction network, also depicted in Figure 1

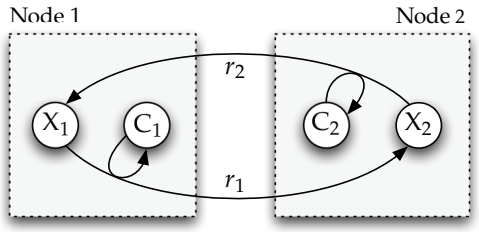


Figure 1: Distributed chemical reaction network: Each node provides a reaction that consumes two molecules C and X from the local multiset, regenerates C and sends molecule X to the neighbor node. The local reaction rules (2a) and (2b) span a global reaction network.

$$C_1 + X_1 \xrightarrow{r_1} C_1 + X_2 \tag{2a}$$

$$C_2 + X_2 \xrightarrow{r_2} C_2 + X_1 \tag{2b}$$

that works as follows:

A received molecule is in a first step passively placed into the multiset of the node. For example, rule (2b) is not executed immediately after node 2 receives a new X-molecule. It is rather scheduled for a later time determined by an exact stochastic reaction algorithm, for instance those proposed by Gillespie (1977) or Gibson and Bruck (2000). The reaction scheduler draws the delay between two occurrences of the same reaction from an exponential probability distribution. The role of this delayed execution is to enforce the law of mass action at the macroscopic level: Molecules C_1 and X_1 react with an average rate equal to the product of their quantity $r_1 = c_1 x_1$. The rate of packets sent from node 1 to node 2 is equal to r_1 while the packet stream in the opposite direction exhibits a rate of $r_2 = c_2 x_2$. It can be shown that the overall reaction system spanned by the two local reaction rules strives towards equilibrium where the number of X-molecules in either node is inversely proportional to the number of the corresponding C-molecules (Meyer & Tschudin, 2009).

3.2 Fraglets — a Chemical Programming Language

So far we demonstrated a static reaction networks where abstract reaction rules were "installed" permanently in each node. Here, we extend this model aiming at dynamically changing the set of reaction rules. We present the Fraglets language (Tschudin, 2003), an artificial chemistry according to the definition of Dittrich et al. (2001), whose corresponding chemical machine is executable and which serves as a simple platform to run chemical protocols.

Each molecule $s \in \{S\}$, or packet, is a string of symbols over a finite alphabet Σ. The first symbol of the string defines the string rewriting operation applied to this molecule by the virtual chemical machine; it can be thought of an assembler instruction. For example, the molecule `[fork a b c d]` transforms itself and splits into the two molecules `[a c d]` and `[b c d]`. The list below shows some essential instructions and their actions.

$$
\begin{aligned}
\texttt{[match } \alpha \ \Phi\texttt{]} + \texttt{[}\alpha \ \Omega\texttt{]} &\longrightarrow \texttt{[}\Phi \ \Omega\texttt{]} \\
\texttt{[fork } \alpha \ \beta \ \Omega\texttt{]} &\longrightarrow \texttt{[}\alpha \ \Omega\texttt{]} + \texttt{[}\beta \ \Omega\texttt{]} \\
\texttt{[nop } \Omega\texttt{]} &\longrightarrow \texttt{[}\Omega\texttt{]} \\
{}_i\texttt{[send } k \ \Omega\texttt{]} &\longrightarrow {}_k\texttt{[}\Omega\texttt{]} \quad \text{if } (i,k) \in \mathcal{E}
\end{aligned}
$$

$\alpha, \beta \in \Sigma$ are arbitrary symbols, $\Phi, \Omega \in \Sigma^*$ are symbol strings, $i, k \in \mathcal{V}$ denote network nodes and $(i,k) \in \mathcal{E}$ communication links of the network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Molecules starting with `match` or any non-instruction identifier are in their *normal form*. The `match`-instruction can be used to join two molecules by concatenating the second to the first after removing the processed headers. Subsequent instructions immediately reduce the product further until they again reach their normal form. For example, the two molecules `[match pkt send 2 pkt]` and `[pkt data]` in node 1 imply the reaction

$$
\begin{aligned}
{}_1\texttt{[match pkt send 2 pkt]} + {}_1\texttt{[pkt data]} & \\
\longrightarrow {}_1\texttt{[send 2 pkt data]} & \\
\text{(transmission to neighbor 2)} \quad \longrightarrow {}_2\texttt{[pkt data]} &
\end{aligned}
$$

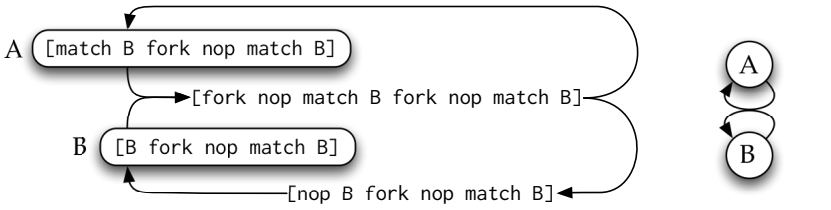
Such a chemical language allows us to "program" the reaction graph. Molecules now have a structure; they *contain* information such as piggybacked user data. However, the dynamics of the reaction network is still governed by the law of mass action and thus, the protocol's behavior is chemically controlled.

4 The Chemical Quine

In ordinary sequential programming languages, a "Quine" is a single piece of code outputting its own source code. In the parallel world of an artificial chemistry like Fraglets, a Quine becomes a set of molecules that is able to regenerate itself. An example that illustrates this concept is the combination of a blueprint molecule B = `[B fork nop match B]` and its active variant A = `[match B fork nop match B]` (Yamamoto et al., 2007). The two molecules react with each other and, according to the Fraglets rewriting rules, regenerate themselves as shown in Figure 2. The schematic illustration in Figure 2(b) shows the corresponding chemical reaction network that is dynamically equivalent to the Fraglets rewriting loop in Figure 2(a). Note that only bimolecular reactions are scheduled according to the law of mass actions; unimolecular rewriting rules such as `fork` are immediately executed, hence these intermediate steps (molecules) are omitted in the schematic notation.

4.1 Replicating Quine and Limited Resources

By repeating the `fork` instruction three times, the above Quine can be converted into a "replicating Quine" as shown in Figure 3. The replicating Quine generates two copies of itself in each round while consuming the original copy. Because the reactions are scheduled according to the law of mass action, the overall production rate increases with the growing number of Quine

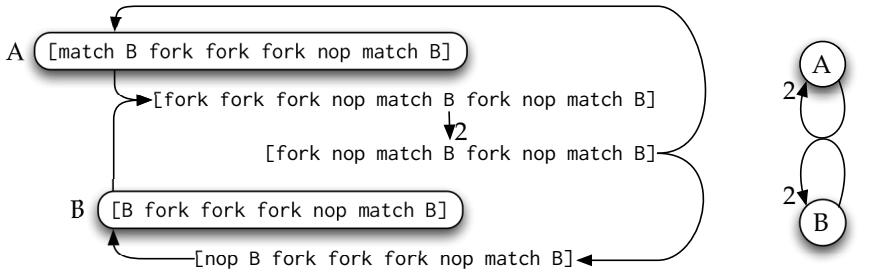


(a) Rewriting loop in Fraglets (b) Reactions

Figure 2: Chemical Quine in Fraglets: A set of molecules (here: Fraglets strings) that regenerates itself. The blueprint molecule B reacts with its active variant A. The consecutive rewriting steps regenerate the two molecules.

instances. Consequently, the population of Quines grows hyperbolically (Szathmáry, 1991), meaning that it theoretically reaches infinite quantity in finite time.

As a limit to this unbounded growth we introduce a non-selective dilution flux to the reaction vessel, which destroys arbitrary molecules as long as the total number of molecules exceeds a pre-defined vessel capacity. This leads to a selective pressure: Only molecules that are part of a self-replicating set have a chance to remain present – all other molecules will eventually be displaced.



(a) Rewriting loop in Fraglets (b) Reactions

Figure 3: Replicating Quine: The replicating Quines increases its population by generating two replicas while the original copy is consumed.

4.2 Dynamic Behavior and Deterministic Fixed Points

The dynamic behavior of the replicating Quine in a vessel of limited capacity N is described by the *Catalytic Network Equation* (Stadler et al., 1993), a deterministic approximation expressed by Ordinary Differential Equations (ODEs) where x_A is the number of A-molecules and where x_B denotes the number of blueprints B

$$\dot{x}_A = \overbrace{x_A x_B}^{\text{growth}} - \overbrace{\frac{x_A}{N} f(\vec{x})}^{\text{death}} \tag{3a}$$

$$\dot{x}_B = \underbrace{x_A x_B}_{\text{growth}} - \underbrace{\frac{x_B}{N} f(\vec{x})}_{\text{death}} \tag{3b}$$

subject to the conservation relation $x_A + x_B = N$, where N is the vessel capacity. The two molecules react, according to the law of mass action, with rate $r = x_A x_B$; each reaction event

leads to an additional pair of molecules. The dilution flux applied to the vessel is equal to the net production rate $f(\vec{x}) = 2x_A x_B$, thus satisfying the conservation relation. The dilution flux is non-selective, this is, each species is diluted with a rate proportional to its relative concentration.

Molecular quantities in a limited vessel are often expressed in (relative) concentrations. The "concentration" of molecule $s \in \mathcal{S}$, χ_s, denotes the frequency of s in the vessel multiset, hence $\chi_s = x_s / N$. Expressed in concentrations, the above equations become

$$\dot{\chi}_A = N\chi_A\chi_B - \chi_A\Phi(\vec{\chi}) \tag{4a}$$

$$\dot{\chi}_B = N\chi_A\chi_B - \chi_B\Phi(\vec{\chi}) \tag{4b}$$

where $\Phi(\vec{\chi}) = 2N\chi_A\chi_B$.

The system exhibits three dynamic fixed points, one at $\hat{\chi}_A = \hat{\chi}_B = 1/2$ and two pathological cases at $\hat{\chi}_A = 1$, $\hat{\chi}_B = 0$, and $\hat{\chi}_A = 0$, $\hat{\chi}_B = 1$. The first fixed point is locally stable according to a standard perturbation analysis (Strogatz, 1994) and is characterized by both molecules – the blueprint and its active variant – being present with the same number of instances.

Figure 4 demonstrates the stability of the replicating quine to perturbations: In a Fraglets vessel of capacity $N = 1000$ molecules both molecule types are present with 500 instances each. At time $t = 0.1$ s we forcefully removed 80 % of the active molecules A from the vessel. The remaining Quines continue to produce replicas, which quickly repopulate the vessel. Once the vessel reaches saturation, there are more blueprints than active molecules, which causes the dilution flux to remove the first more frequently than the latter, until equilibrium is reached. At time $t = 0.2$ s we performed the same attack to the active molecules, from which the system recovers likewise.

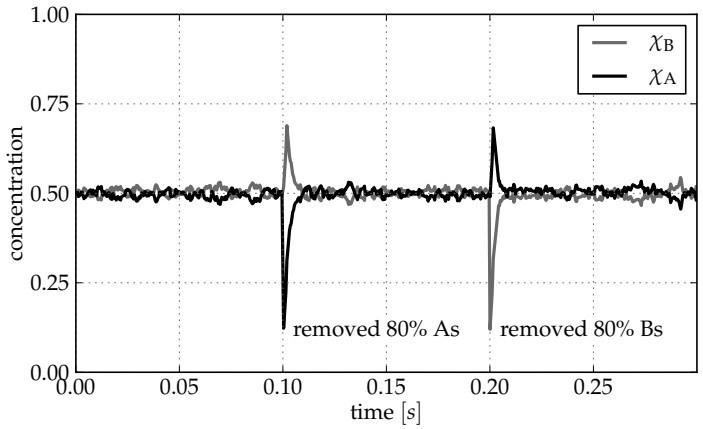


Figure 4: Replicating Quine under deletion attacks: Fraglets simulation of the replicating Quine in a vessel of capacity $N = 1000$ molecules. At time $t = 0.1$ s we removed 80 % of the active molecules A, whereas the same amount of B molecules is removed at time $t = 0.2$ s. Shortly after the attack, the missing molecules are replenished by the remaining Quine instances.

The stability property essentially means that the system returns to equilibrium condition: Even if we perturb the system by removing some instances of either species, the opponent forces of hyperbolic growth and non-selective dilution flux let the system autonomically find back to this fixed point. In other words, the system intrinsically maintains its own redundancy without an external controller!

5 Robustness of the Quine

In this section we quantify the robustness of the replicating Quine in the presence of faults: execution errors and random alteration of memory. But even when no faults occur, the lifetime of the Quine is finite due to stochastic fluctuations.

5.1 Baseline Robustness

The two pathological fixed points from the analysis above deserve some more attention. The deterministic ODE model predicts that they are not locally stable, which may lead to the conclusion that these states are not reachable or not persistent. However, in our stochastic execution environment, these fixed points will eventually be reached and represent states where one of the molecule species is completely absent such that the system becomes deadlocked and finds itself in a so called *absorbing state*. Hence, the lifetime of a chemical Quine is finite, even in the absence of faults.

5.1.1 Quantification of the Robustness Using Phase-Type Distributions

In order to quantify the baseline robustness of the replicating Quine, we now calculate the mean first-passage time to one of the two absorbing states.

Stochastic Model with a Markov Chain. A chemical reaction network governed by the law of mass action is stochastically modeled by a continuous time discrete space Markov jump process, whose dynamic behavior is described by the Chemical Master Equation (CME) (Gillespie, 1992). Our simple replicating Quine only consist of two species and the total number of molecules is fixed to N. This allows us to model the system as a finite birth-death Markov chain where the state $N_{\mathrm{A}}(t) \in [0, N]$ is a random variable denoting the number of A-molecules, whereas the number of blueprints (B) is given by $N_{\mathrm{B}}(t) = N - N_{\mathrm{A}}(t)$.

Figure 5 depicts the corresponding birth-death Markov chain, in which we labeled the states according to the number of A-molecules present. In saturation, a reaction yields two additional molecules and brings the system to a hypothetical state with $N+2$ molecules, such that the dilution flux has to remove two arbitrary molecules. The effective transitions move the system along the saturation line; their rate is calculated as product of the reaction rate times the sum of all dilution path probabilities to one of the neighbor states. In this context, the birth rate λ_i represents the average rate of gaining an A-molecule and losing a B-molecule in state i whereas the death rate μ_i describes a movement in the opposite direction:

$$\lambda_i = \overbrace{i(N-i)}^{\text{reaction rate}} \overbrace{\frac{N-i+1}{N+2}\frac{N-i}{N+1}}^{\text{dilution prob.}} \tag{5a}$$

$$\mu_i = \underbrace{i(N-i)}_{\text{reaction rate}} \underbrace{\frac{i+1}{N+2}\frac{i}{N+1}}_{\text{dilution prob.}} \tag{5b}$$

We are interested in the survival time of the replicating Quine, this is the time until the Markov process hits one of the two absorption states ($\{0, N\}$).

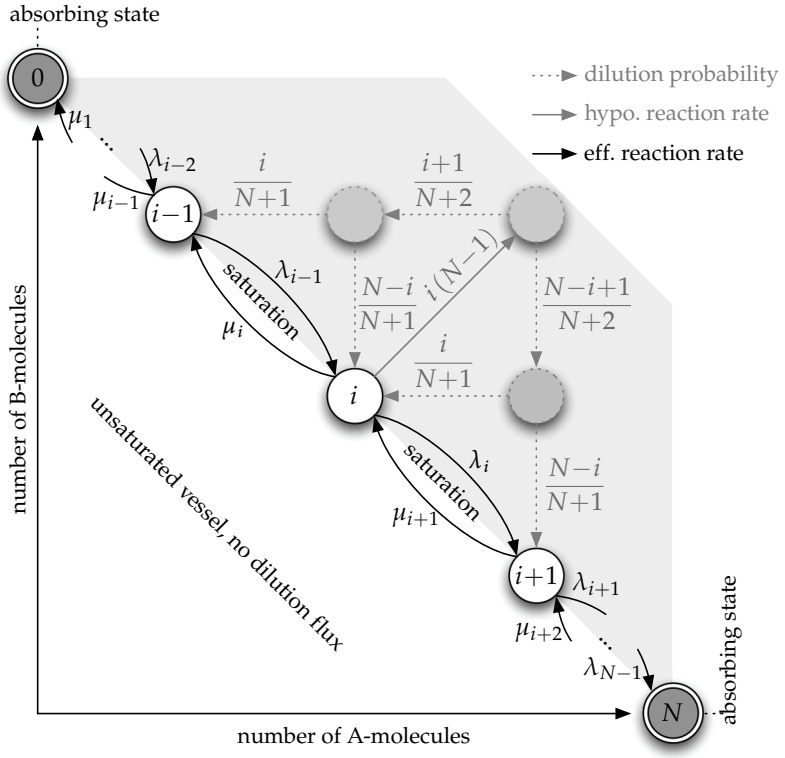


Figure 5: Markov chain of the replicating Quine: The replicating Quine in a saturated vessel is modeled as a birth-death Markov chain. The states are labeled according to the number of A-molecules present. Reactions from a saturated state leads to a hypothetical state (dashed circles). The dilution flux (dashed arrows) brings the system back to a valid state along the saturation line. The effective transition rates (black arrows) are calculated as product of the reaction rate times the sum of all dilution paths to the target state.

The Time to Absorption is Phase-Type Distributed. The distribution of the first passage time from an initial state (here: $k = N/2$) to one of the absorption states is given by the phase-type distribution according to Neuts (1981): We first build the transition rate matrix $\mathbf{Q}$, the off-diagonal elements of which are the transition rates ($q_{i,i+1} = \lambda_i$, $q_{i,i-1} = \mu_i$) and the diagonal elements are given by the sum $q_{i,i} = -\sum_{i \neq j} q_{i,j}$ such that its row sum is zero. Next we separate the transient states from the absorbing states; thereby we summarize all absorbing states to one single state. Finally, we rewrite the transition matrix to the form

$$\mathbf{Q} = \begin{pmatrix} \mathbf{S} & \mathbf{S}_0 \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{6}$$

where $\mathbf{S}$ is the transition matrix among the transient states and $\mathbf{S}_0 = -\mathbf{S}\vec{1}$ is the vector of transition rates to the absorbing state. The mean time to absorption is now given by the expression

$$\mathbb{E}\left[T_{\{0,N\}}\right] = -\vec{p}\mathbf{S}^{-1}\vec{1} \tag{7}$$

where $\vec{p}$ is the initial probability distribution with $p_k = 1$ and $p_{i \neq k} = 0$.

Method and Results. We developed a tool that given the reaction network and the vessel capacity N automatically builds the corresponding Markov chain and its transition matrix.

We then use Scilab (Campbell, Chancelier, & Nikoukhah, 2006) to calculate the mean time to absorption based on the generated transition matrix. This method is feasible for vessel capacities $N \leq 30$. For larger vessels, the transition matrices become very large and Scilab produces erratic values due to the limited precision of the floating point representation. Therefore, for vessel capacities $N > 30$ we used Glaz' method. Glaz (1979) provided an explicit formula for the mean of the first absorption time for the special case of a finite birth-death processes with two absorbing states at the opposite ends of the Markov chain.

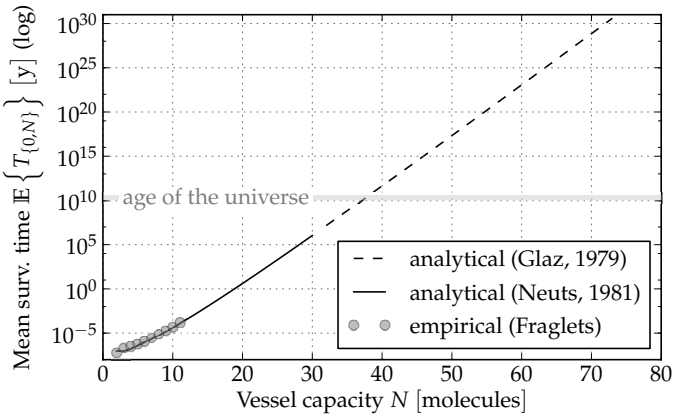


Figure 6: Baseline robustness of the replicating Quine: Based on calculation methods by Glaz and Neuts, the mean survival time exponentially increases with the vessel capacity N and reaches the age of the universe for about $N = 37$ molecules. These predictions match empirical measurements, i.e. survival times of Fraglets simulations, averaged over 1000 runs.

Figure 6 shows the mean time to absorption for the replicating Quine in vessels with different capacities. The figure illustrates that the mean time to absorption exponentially increases with the vessel capacity N. In fact, already for $N = 37$ molecules the mean survival time exceeds the current age of the universe. For small Ns we complemented these analytical calculations with 1000 simulation runs in Fraglets; the empirically measured average survival time accurately matches the predicted survival time.

5.1.2 Identification of Absorption States Using the Chemical Organization Theory

For large reaction networks, it becomes unfeasible to analyze the corresponding Markov chain, and thus a detailed dynamic analysis is not possible anymore. The *Chemical Organization Theory* (COT) (Dittrich & Speroni di Fenizio, 2007) analyzes a reaction network statically rather than analyzing the system's dynamics; it not even requires the reactions to follow the law of mass action. Instead of operating on multisets, the COT identifies those *sets* of species – *organizations* – "which are likely to be observed in a large reaction vessel on the long run" (Dittrich & Speroni di Fenizio, 2007). Such organizations are either desired quasi steady-states or unwanted absorption states. In the following, we demonstrate how to use the COT to at least identify organizations of large reaction systems, for which a detailed stochastic analysis would be too expensive or even impossible.

The COT models a dilution flux by explicit reactions: all molecules are consumed by first-

order reactions. The reaction network for the replicating Quine becomes

$$A + B \longrightarrow 2A + 2B \tag{8a}$$
$$A \longrightarrow \emptyset \tag{8b}$$
$$B \longrightarrow \emptyset \tag{8c}$$

for which the COT predicts two organizations: the quasi steady-state where both species are present ($\{A, B\}$) and the pathological empty set $\{\}$ as consequence of the dilution reactions.

Note however that the COT does not predict one of the absorption states we previously identified in our dynamical analysis: we know that two absorption states exist where either A- or B-molecules are absent. The reason for this roots in the way COT models the dilution flux. In Fraglets, dilution is always linked to an excessive production of molecules; the dilution rate is actually proportional to the net production rate. For the special case where no reaction occurs anymore, i.e. when the system becomes inert, the dilution rate drops to zero and the reactions (8b) and (8c) become void. Thus, in addition to finding the organization for the above reaction system, we have to determine the organizations of the reaction network without a dilution flux. Then, the union of both organization sets yields the organizations relevant for our execution model (P. Dittrich, personal communication, July 8, 2010).

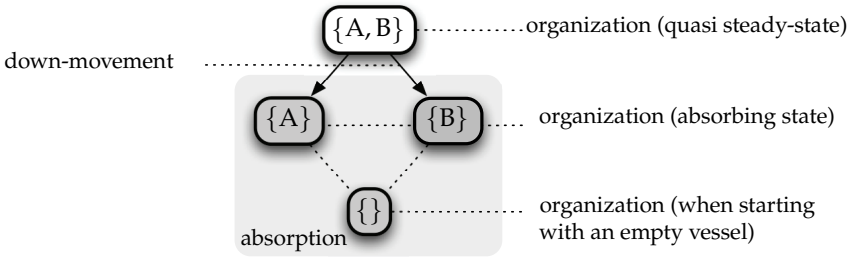


Figure 7: Lattice of organizations of the replicating Quine: The system may lose one of the species due to the stochastic dilution flux (down-movement). The system is dead for species sets in the shaded area (absorption).

The resulting lattice of organizations for the replicating Quine is depicted in Figure 7. Organization $\{A, B\}$ contains all species and represents the healthy quasi steady-state, in which the system stays for a long time.

A down-movement in the lattice of organizations to one of the absorption organizations $\{A\}$ or $\{B\}$ may happen because the reaction vessel loses one of the species due to the still active dilution flux. But from there, the system is never able to reach the pathological organization $\{\}$. This organization is only reachable if the vessel already starts empty. From a lower-level organization such as $\{A\}$ or $\{B\}$ the system is not able to intrinsically move up to a higher organization. Thus, all states within the two single-species organizations are stochastically closed and hence absorption states.

The COT provides us a qualitative picture of the possible fate of a complex reaction system. In this report, we will complement the robustness calculations of all reaction networks with the corresponding lattice of organizations.

5.2 Robustness to Execution Errors

We expect the robustness of a Quine to decrease with the presence of execution errors, which we model as follows: With probability p_e, each Fraglets instruction fails to produce the right result; instead, a neutral fraglet is generated, that is unable to replicate or to react with another molecule.

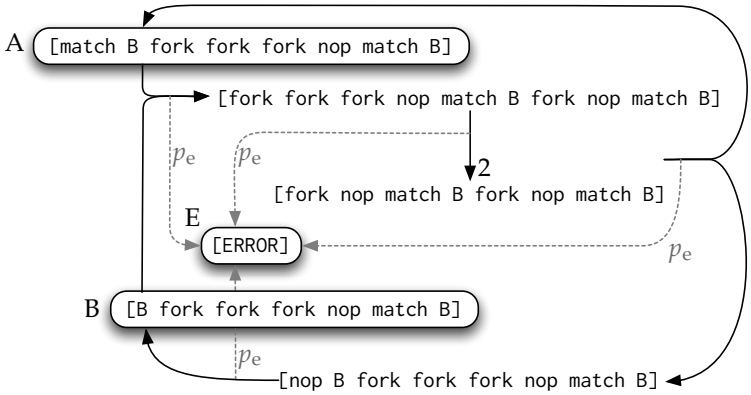


Figure 8: Robustness of the replicating Quine to execution errors: Each rewriting step may be subject to an execution error (dashed lines) with probability p_e. We assume that the error product E is not able to react with the other molecules.

Figure 8 shows the rewriting steps of the replicating Quine that is subject to execution errors. The dashed arrows indicate that instead of the intended rewriting result, a error product E is generated. The corresponding reaction network is

$$\mathrm{A} + \mathrm{B} \xrightarrow{p_e(2-p_e)} \mathrm{E} \tag{9a}$$

$$\mathrm{A} + \mathrm{B} \xrightarrow{(1-p_e)^2 p_e} 2\mathrm{E} \tag{9b}$$

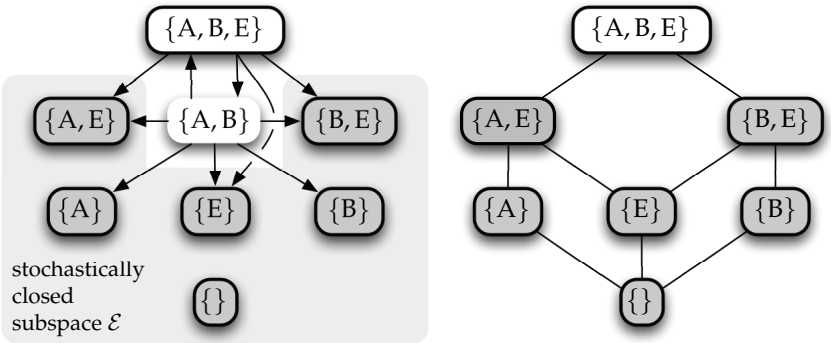
$$\mathrm{A} + \mathrm{B} \xrightarrow{(1-p_e)^3 p_e^2} 2\mathrm{E} + 2\mathrm{B} \tag{9c}$$

$$\mathrm{A} + \mathrm{B} \xrightarrow{(1-p_e)^3 p_e(2-p_e)} \mathrm{A} + \mathrm{E} + 2\mathrm{B} \tag{9d}$$

$$\mathrm{A} + \mathrm{B} \xrightarrow{(1-p_e)^5} 2\mathrm{A} + 2\mathrm{B} \tag{9e}$$

Before calculating the survival time, we analyze the reaction network statically. Figure 9(a) provides the set transition diagram, a simplified illustration of the underlying Markov process where we unified all states containing the same species. An even simpler view is provided by the lattice of organizations in Figure 9(b), which depicts the possible (quasi) steady-states of the system.

In order to determine the absorption state we recall that the system if alive if the vital reaction among A and B molecules is still possible, and hence, those molecules must be present. Note that $\{\mathrm{A}, \mathrm{B}\}$ is not an organization because errors will eventually be produced while the two molecules react. Thus the set $\{\mathrm{A}, \mathrm{B}, \mathrm{E}\}$ hosts the healthy quasi steady-state of the system. Figure 9(a) helps us in identifying the stochastically closed subspace $\mathcal{E}$, from which no transition leads back to this organization. Absorption states are instances of any organization in this subspace.



(a) Molecule set transitions (b) Lattice of organizations

Figure 9: Lattice of organizations of the replicating Quine subject to execution errors: The system may lose one of the species due to the stochastic dilution flux. The system is dead for species sets in the stochastically closed subspace $\mathcal{E}$. Organizations have a black border; they form a lattice as shown in (b). Note that $\{A, B\}$ is not an organization because errors eventually appear.

Now that we identified the subspace $\mathcal{E}$, in which the system became inert, we calculate the mean first-passage time to this subspace from the healthy quasi steady-state. Glaz' method does not work, because the system is not modeled by a simple birth-death chain anymore, but Neuts' generic method based on phase-type distribution can still be applied. The survival time, i.e. the first-passage time to an absorption state, can be calculated from the transition matrix of the corresponding two-dimensional Markov state array.

Figure 10 shows the mean survival time of a replicating Quine in vessels of different capacities plotted with respect to different execution error probabilities. The robustness of the Quine barely decreases for realistic error probabilities and only sharply drops for probabilities above 1 %. The diagonal line in Figure 10 illustrates the survival rate of a single Quine instance that just rewrites itself but does not replicate, and thus, is not able to heal itself. This situation is comparable to traditional sequential software, which realistically fails at the first occurrence of an execution error. Although the self-healing Quines eventually die even in the absence of execution errors, they live much longer than un-instrumented code in the presence of execution errors, even in small reaction vessels.

5.3 Robustness to Memory Alterations

The second type of error we discuss in this section are spontaneous alterations of memory bits with rate δ $[1/\text{bit s}]$. Such Single Event Upsets (SEUs) can be caused, for example, by cosmic radiation (Normand, 1996). The rate at which a fraglet is hit depends on its length l in symbols and the bit-width of the symbol encoding scheme, b; thus a fraglet is hit $lb\delta$ times per second. We model the spontaneous mutations that turn a fraglet into a neutral molecule E with the

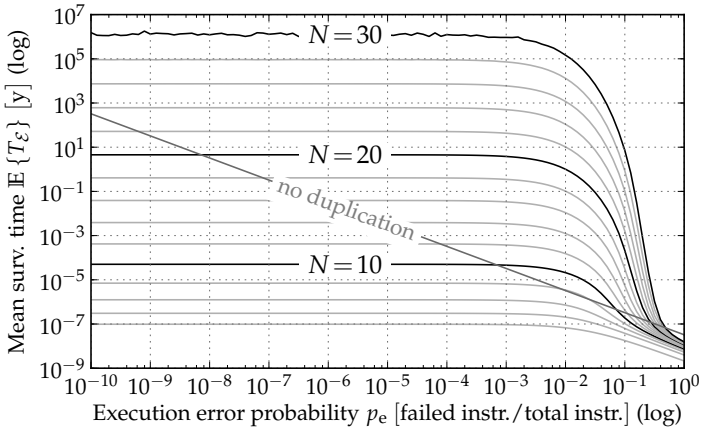


Figure 10: Robustness of the replicating Quine in presence of execution errors: Mean survival time in a vessel of capacity N, plotted with respect to the execution errors probability p_e. The fluctuations for $N=30$ molecules are due to floating point precision limits in Scilab. The diagonal line illustrates the survival rate of a single non-replicating and thus not self-healing Quine instance, representing a traditional piece of software, which realistically fails at the first occurrence of an execution error.

reaction network

$$\mathrm{A} + \mathrm{B} \xrightarrow{1} 2\mathrm{A} + 2\mathrm{B} \tag{10a}$$

$$\mathrm{A} \xrightarrow{8b\delta} \mathrm{E} \tag{10b}$$

$$\mathrm{B} \xrightarrow{7b\delta} \mathrm{E} \tag{10c}$$

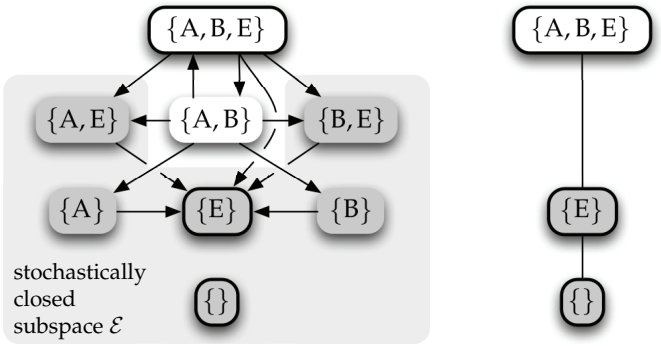
As for execution errors, the lattice of organizations for memory alterations depicted in Figure 11 indicates that the quasi steady-state includes mutation products. The stochastic dilution flux may bring the system to absorption, this is, to the species set $\{A, E\}$ or $\{B, E\}$. Note that the memory alteration "reactions" (10c) and (10c) are still active in those states. They eventually but very slowly turn all A- and B-molecules to error products E, which is the reason why this species set is the only non-pathological absorption organization.

For calculating the survival time of the replicating Quine subject to memory alterations, we are not interested in the time to absorption, i.e. in the first-passage time to the organization $\{\mathrm{E}\}$; we are rather interested in the much shorter time in which the system enters the stochastically closed subset $\mathcal{E}$, from which the quasi steady-state cannot be reached anymore.

Figure 12 shows the mean survival time of our replicating Quine, now plotted with respect to different symbol mutation rates (we assumed that each fraglets symbol is encoded by eight bits). To appraise these curves, we recall that, according to Normand (1996), the Single Event Upset (SEU) rate caused in microelectronic devices by radiation is around $5 \cdot 10^{-16}$ 1/bit s. In this region, the Quine is almost unaffected by mutations.

6 Distributed Quines

In this section we study whether the survival time of a Quine is higher in a distributed context. Intuitively, we expect that the system is able to survive for a longer time when backup molecules



(a) Molecule set transitions (b) Lattice of organizations

Figure 11: Lattice of organizations of the replicating Quine subject to memory alterations: The system may lose one of the species due to the stochastic dilution flux. The system is dead for species sets in the stochastically closed subspace, indicated by the shaded area in (a). Organizations have a black border; they form a lattice as shown in (b).

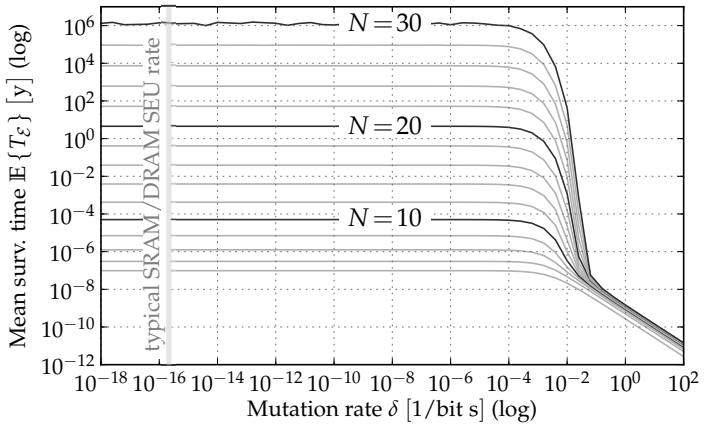


Figure 12: Robustness of the replicating Quine in presence of memory alterations: Mean survival time in a vessel of capacity N, plotted with respect to the bit mutation rate δ. A Fraglet symbol is encoded by 8 bits. The fluctuations for $N=30$ molecules are due to floating point precision limits in Scilab. The Single Event Upset (SEU) rate caused in microelectronic devices by radiation is around $5 \cdot 10^{-15}$ 1/bit s. In this region, the Quine is almost unaffected by mutations.

exist: When one vessel hits an absorption state, another vessel in the network could be still alive and would be able to heal the inert node by sending it the seed that can restart the Quine's loop. However, we found out that the strategy of spreading seeds is important for this scheme to work.

We analyzed the robustness of two different types of distributed Quines. Common to both types is that each node i contains a reaction between the Quine's active molecule A_i and the corresponding blueprint B_i. The difference is that in the first case we produce the seed replicas locally and send them to a random neighbor node (anycast), as is shown in Figure 13. In

Fraglets, this is realized by encoding the two species as

$$A_i = \texttt{[match B anycast fork fork fork nop match B]} \tag{11a}$$

$$B_i = \texttt{[B anycast fork fork fork nop match B]} \tag{11b}$$

Once arrived, the seeds unfold and generate two instances of either species.

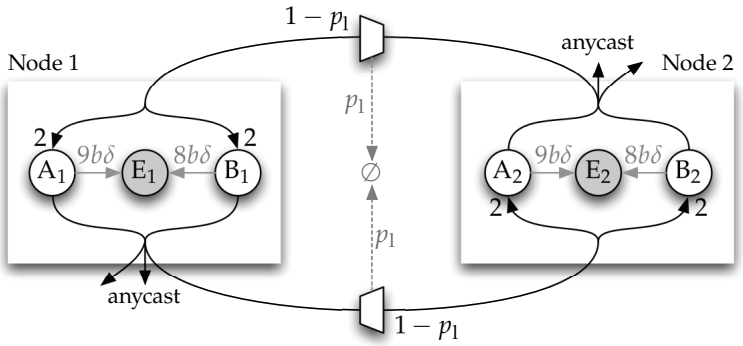


Figure 13: Distributed Quine type 1: The product of a reaction among A and B is sent to an arbitrary neighbor node where two replicas of those molecules are produced. We consider memory bit mutations at rate δ and packet loss during transmission with probability p_l.

The second distributed Quine type is depicted in Figure 14 and the Fraglets code is shown below.

$$A_i = \texttt{[match B fork nop anycast fork nop match B]} \tag{12a}$$

$$B_i = \texttt{[B fork nop anycast fork nop match B]} \tag{12b}$$

Their product only sends one seed to a random neighbor node whereas the other seed is used to regenerate the Quine locally.

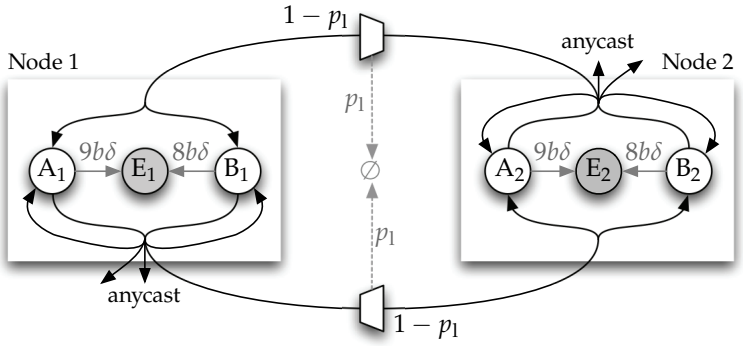


Figure 14: Distributed Quine type 2: The product of a reaction among A and B is sent to an arbitrary neighbor node and is delivered locally. In total, one additional instance of both molecules is produced. We consider memory bit mutations at rate δ and packet loss during transmission with probability p_l.

Figure 15 shows the mean survival time against different mutation rates of the two distributed Quine types in a two-node network topology, as well as the curve of a one-node replicating Quine for comparison purpose. The total capacity of the reaction vessel(s) is the

same for all three cases, namely $N = 14$ molecules: This is, for the one-node Quine, the single vessel provides the whole capacity, whereas for the two distributed cases, the capacity is evenly distributed between the two participating nodes.

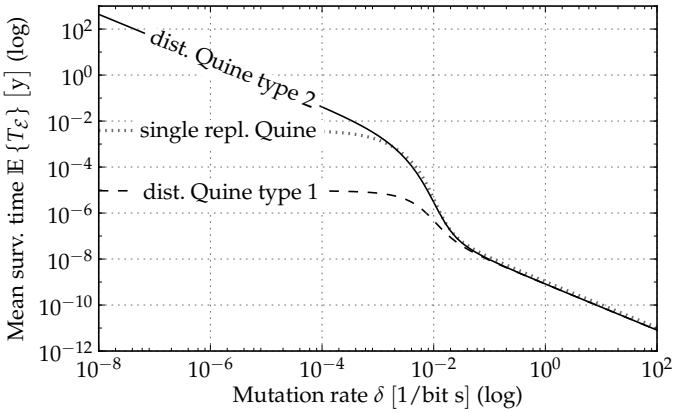


Figure 15: Robustness of the distributed Quines in presence of memory alterations: Mean survival time in two vessels with a total capacity of $N = 14$ molecules in comparison to a single replicating Quine with the same total capacity of $N = 14$ molecules, plotted with respect to the bit mutation rate δ. A Fraglet symbol is encoded by 8 bits. An absorption state is reached when both vessels either lack A_i or B_i molecules.

We observe that the single Quine always outperforms the distributed Quine of type 1, which sends both seed copies to the neighbor. The distributed Quine of type 2, however, is the most robust variant for moderate bit alteration rates. This stems from the fact that the second type has to digest only two received molecules at once, whereas the first type receives four instances at a time. The more excessive molecules the dilution flux has to remove, the higher is the fluctuation around the quasi steady-state and, consequently, the more likely one of the absorption states is reached. Thus, the good recipe for robust distributed Quines is to replicate locally in order to maintain the population and to send only one copy to a neighbor node for the case a neighbor deviates from the fixed point or even reached a local absorption state.

Figure 16 shows the robustness surface of the two distributed Quine types for different symbol alteration rates and packet loss probabilities. Packet loss only has a marginal effect on the robustness, especially for the second type, which replicates locally. The second type only needs the received Quine seeds to bootstrap the local Quine when accidentally hitting a local absorption state.

Our calculations show that a carefully designed distributed Quine exhibits a better robustness than an isolated Quine even when comparing them with the same *total* vessel capacity. In a typical networking scenario, the total memory resource available grows with the number of nodes in the network. In this case, where additional memory comes for free, the distributed Quine is always by far more robust than a network of isolated Quines.

7 A Generic Building Block for Self-Healing Software

The Quine studied so far just spends CPU cycles replicating itself. In this section, we demonstrate how our self-healing Quines can be enriched to perform an actual and useful computation. The resulting data-processing Quine can be used as a software building blocks for various tasks,

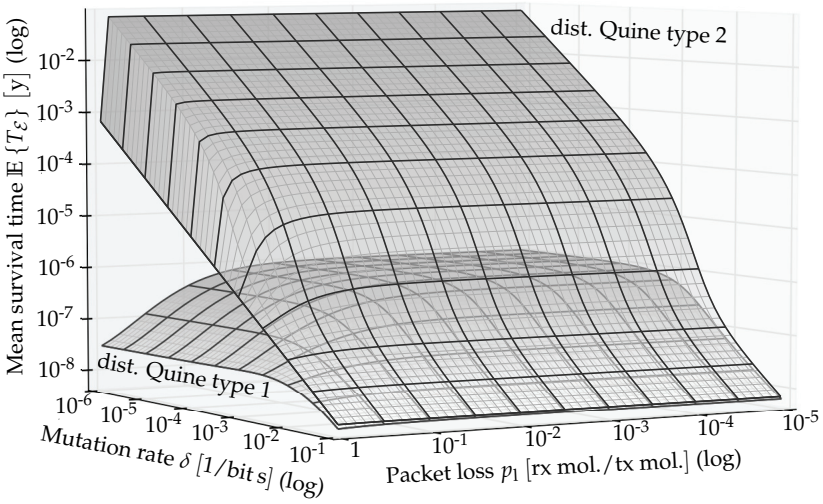


Figure 16: Robustness of the distributed Quines in presence of memory alterations and packet loss: Mean survival time in two vessels with a total capacity of $N = 10$ molecules, plotted with respect to the bit mutation rate δ and the inter-vessel packet loss probability p_1. A Fraglet symbol is encoded by 8 bits. The absorption state is an instance of the species set $\mathcal{E}$ where both vessels lack A_i or B_i.

serving as a design pattern to engraft the self-healing property to an arbitrary piece of code.

7.1 The Data-Processing Quine

A small modification to the replicating Quine's structure leads to a data-processing Quine, as is depicted in Figure 17. With the new structure, the set of A- and B-molecules does not directly replicate anymore. Instead, the active molecule A reacts with a data molecule (or "data packet") D, computes some product and also generates an additional reward molecule R. The reward molecule reacts with and consumes a blueprint molecule B, which contains the necessary information to re-create the active molecule and its blueprint. As usual with replicating Quines, the reaction between R and B results in two instances of A and B.

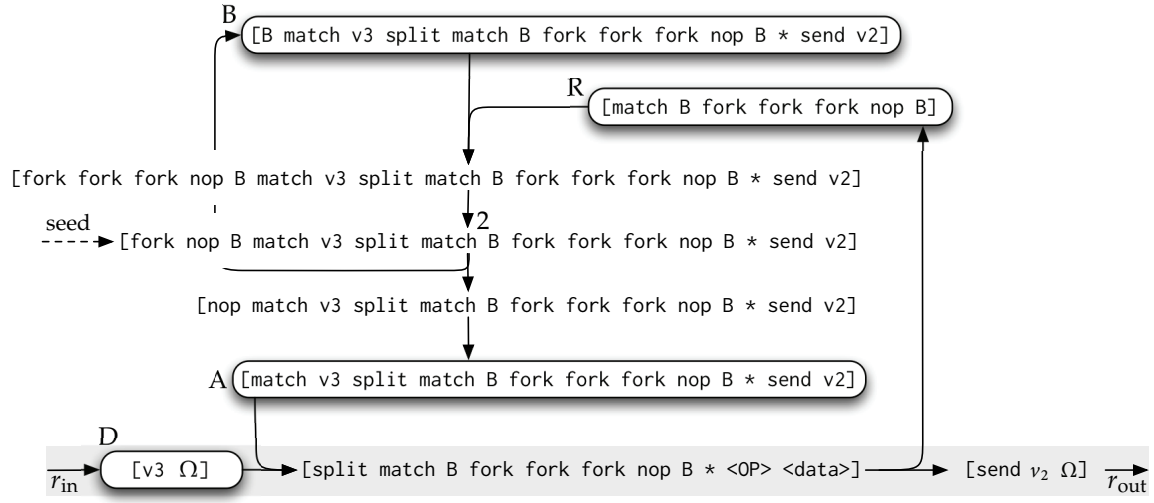
7.2 A Generic Template to Quinify Code

Every data-processing rule in Fraglets can be converted to a data-processing Quine, which is intrinsically robust to the discussed faults. In the following we introduce a simple recipe to "quinify" code.

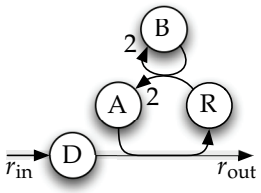
The data-processing Quine depicted in Figure 17 is the self-healing implementation of the following packet forwarding reaction:

$$\begin{aligned} & {}_{v_1}\overbrace{\texttt{[matchp v3 send v2 v3]}}^{f} + {}_{v_1}\texttt{[v3 }\Omega\texttt{]} \\ \longrightarrow\ & {}_{v_1}\texttt{[matchp v3 send v2 v3]} + {}_{v_2}\texttt{[v3 }\Omega\texttt{]}, \end{aligned} \tag{13}$$

The `matchp`-fraglet residing in node v_1 consumes local data packets destined to node v_3 and sends them to the next hop v_2. In fact, any persistent Fraglets rule f like the one in (13) can be



(a) Rewriting loop in Fraglets



(b) Reaction network

Figure 17: Data-processing Quine: The active molecule A processes a data molecule D resulting in a reward R. The blueprint B contains all information necessary to generate two copies of A and B when reacting with R.

converted to a data-processing Quine by using the template

$$[\Psi_{\text{spawn}}\ \Psi_{\text{consume}}(f)\ \ \Psi_{\text{replicate}}\ \Psi_{\text{produce}}(f)\ ] \tag{14a}$$

where generally the symbol strings Ψ_{spawn} and $\Psi_{\text{replicate}}$ are defined as

$$\Psi_{\text{spawn}} := \texttt{fork nop B} \tag{14b}$$

$$\Psi_{\text{replicate}} := \texttt{split match B fork fork } \Psi_{\text{spawn}}\ \texttt{*} \tag{14c}$$

To generate the Quine replacement for $f = $ `[matchp v3 send v2 v3]`, we define the consumption and production part of of the data-processing Quine as

$$\Psi_{\text{consume}}(f) := \texttt{match v3} \tag{14d}$$

$$\Psi_{\text{produce}}(f) := \texttt{send v2 v3} \tag{14e}$$

which results in the seed

```
[fork nop match v3 split match Bfork fork fork nop B * send v2 v3]
```
(15)

that bootstraps the data-processing Quine (compare Figure 17). Complex software consisting of many `matchp`-rules can be quinified by following this recipe. There are, however, some

circumstances in which those Quines compete against each other for the limited memory rather than cooperate to accomplish a common task. We studied this behavior separately in more detail (Meyer & Tschudin, 2010).

7.3 Intrinsic Packet Loss

Unlike the original Quine, the data-processing Quine only replicates upon processing a data packet. The replication rate is given by the production rate of R which is approximately equal to the packet injection rate r_{in}. Thus, in order to exhibit self-healing properties, the data-processing Quine has to be continuously fed with input molecules.

Ideally, the rate at which the Quine produces results, r_{out}, is equal to the data molecule injection rate r_{in}. In this case the system does not intrinsically lose packets. However, we have to expect some of the packets D being diluted. In fact, the more buffered packets are present in the vessel, the higher is the probability for intrinsic packet loss due to the dilution flux. Here, we determine the packet loss probability p_{l}, given as the fraction of injected packets that does not arrive at the output:

$$p_{\text{l}} = 1 - \frac{r_{\text{out}}}{r_{\text{in}}} \tag{16}$$

In order to quantify this packet loss, we present a closed-form expression for the deterministic fixed point of the data-processing quine. Therefore, we write down the corresponding ODE system in terms of relative concentrations

$$\dot{\chi}_{\text{D}} = \frac{r_{\text{in}}}{N} - N\chi_{\text{A}}\chi_{\text{D}} - \chi_{\text{D}}\Phi(\vec{\chi}) \tag{17a}$$

$$\dot{\chi}_{\text{A}} = -N\chi_{\text{A}}\chi_{\text{D}} + 2N\chi_{\text{B}}\chi_{\text{R}} - \chi_{\text{A}}\Phi(\vec{\chi}) \tag{17b}$$

$$\dot{\chi}_{\text{R}} = N\chi_{\text{A}}\chi_{\text{D}} - N\chi_{\text{B}}\chi_{\text{R}} - \chi_{\text{R}}\Phi(\vec{\chi}) \tag{17c}$$

$$\dot{\chi}_{\text{B}} = N\chi_{\text{B}}\chi_{\text{R}} - \chi_{\text{B}}\Phi(\vec{\chi}) \tag{17d}$$

with the total dilution flux being

$$\Phi(\vec{\chi}) = \frac{r_{\text{in}}}{N} - N\chi_{\text{A}}\chi_{\text{D}} + 2N\chi_{\text{B}}\chi_{\text{R}} \tag{17e}$$

subject to the constraint that all concentrations denote proper molecule frequencies, χ_{D}, χ_{A}, χ_{R}, $\chi_{\text{B}} \in [0, 1]$, and that the total number of molecules is conserved, $\chi_{\text{D}} + \chi_{\text{A}} + \chi_{\text{R}} + \chi_{\text{B}} = 1$. This ODE system depends on two parameters, the vessel capacity N and the packet injection rate r_{in}. In order to eliminate one parameter, we normalize the packet injection rate: For the normalized injection rate $\rho(N) = 8r_{\text{in}}/N^2$ the above equation system yields the following parametric fixed

point.

$$\hat{\chi}_{\mathrm{D}}(\rho) = \frac{1}{2}\left(1 - \sqrt{1-\rho}\right) \tag{18a}$$

$$\hat{\chi}_{\mathrm{A}}(\rho) = \frac{1}{4}\left(3 - \hat{\chi}_{\mathrm{D}} - \sqrt{1 + 2\rho + 2\hat{\chi}_{\mathrm{D}} + \hat{\chi}_{\mathrm{D}}^2}\right) \tag{18b}$$

$$\hat{\chi}_{\mathrm{B}}(\rho) = \frac{1}{2}\left(1 - \hat{\chi}_{\mathrm{D}}\right) \tag{18c}$$

$$\hat{\chi}_{\mathrm{R}}(\rho) = \frac{1}{2}\left(1 - 2\hat{\chi}_{\mathrm{A}} - \hat{\chi}_{\mathrm{D}}\right) \tag{18d}$$

Figure 18 shows the steady-state concentrations with respect to the normalized packet injection rate ρ. If packets are injected at a slow pace, they are immediately processed by the Quine's active molecules while the vessel's capacity is equally shared among the active molecule A and its blueprint B. Consequently, the vast majority of injected packets are converted to products. For increasing injection rates, data molecules start to accumulate, building-up a non-zero steady-state concentration $\hat{\chi}_{\mathrm{D}}$. Since the vessel's capacity is shared among all molecules, this partly squeezes out active molecules and blueprints. On the other hand, the replicating quine also displaces packets. In order to calculate the probability of packet loss we recall that the output rate follows the Law of Mass Action (LOMA): $r_{\mathrm{out}} = x_{\mathrm{A}} x_{\mathrm{D}}$. Hence, the steady-state data loss probability $\hat{p}_{\mathrm{l}}$, shown as dashed line in Figure 18, is given as

$$\hat{p}_{\mathrm{l}}(\rho) = 1 - \frac{r_{\mathrm{out}}}{r_{\mathrm{in}}} = 1 - \frac{\overbrace{\hat{x}_{\mathrm{A}}\hat{x}_{\mathrm{D}}}^{\text{LOMA}}}{r_{\mathrm{in}}} = 1 - 8\frac{\hat{\chi}_{\mathrm{A}}\hat{\chi}_{\mathrm{D}}}{\rho} \tag{19}$$

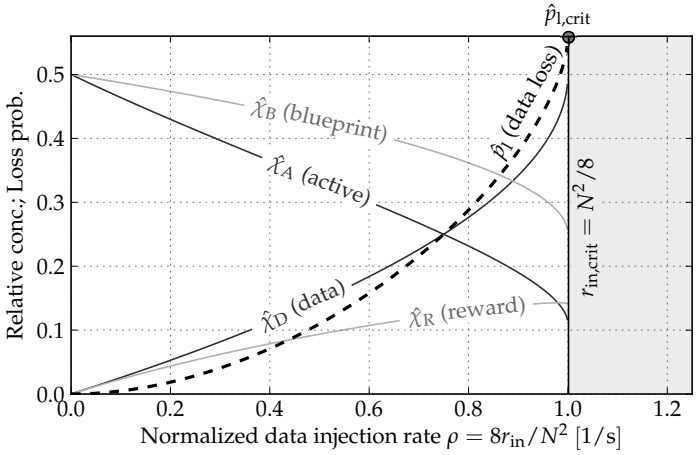


Figure 18: Deterministic fixed point of the data processing Quine: Plotted with respect to ρ, the data injection rate normalized to the vessel capacity. For increasing data injection rates, attended by a growing steady-state concentration of the data molecule $\hat{\chi}_{\mathrm{D}}$, the packet loss probability $\hat{p}_{\mathrm{l}}$ increases due to the dilution flux.

The steady-state packet loss rises for increasing data injection rates and reaches $\hat{p}_{l,\mathrm{crit}} \approx 0.56$ for $\rho = 1$, this is, for a critical packet injection rate of $r_{\mathrm{in,crit}} = N^2/8$.

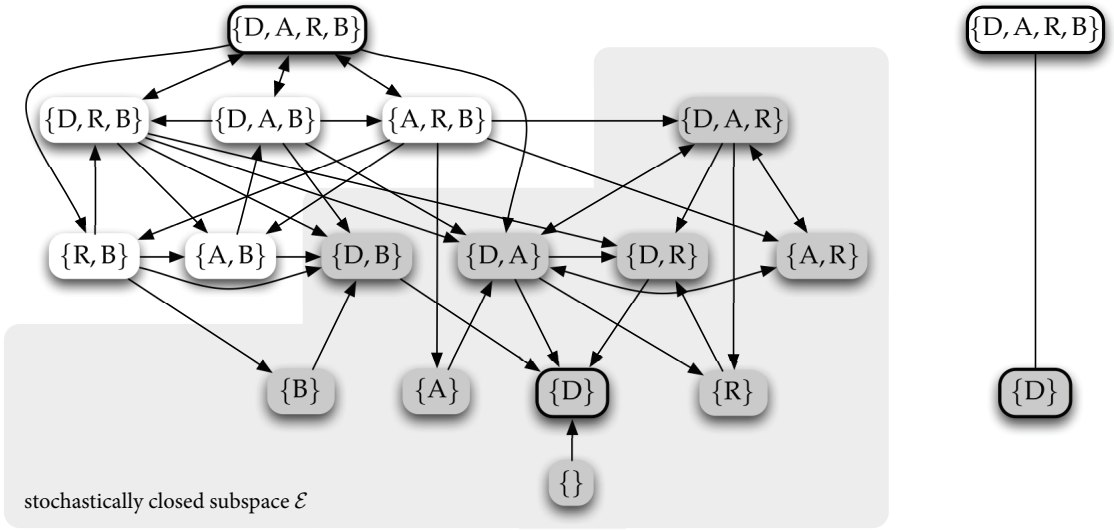
7.4 Critical Data Injection Rate

If the data-processing Quine has to digest data molecules that are injected faster than with the critical rate of $r_{\text{in,crit}} = N^2/8$ there is no stable fixed point where all species are present anymore. For higher rates, the active molecules and blueprints are completely squeezed out by the packet stream. As a result, the Quine is not able to replicate anymore and the system ends in the pathological fixed point $\hat{\chi}_D = 1$, $\hat{\chi}_A = \hat{\chi}_R = \hat{\chi}_B = 0$.

We expect (and will show in the next section) that the robustness of the data-processing Quine drops for increasing injection rates. This essentially implies that the data-processing quine must be fed with a low-rate packet stream ($r_{\text{in}} \ll N^2/8$) or, said differently, the vessel capacity must be dimensioned such that the critical rate never occurs, on which we will focus later in Section 7.6. Since the critical rate grows with order N^2, the system scales well, and it is feasible to find a reasonable capacity for arbitrary packet rates.

7.5 Baseline Robustness

The deterministic fixed point calculated above coincides with the quasi steady-state of the stochastic model. As we showed for the replicating Quine, the stochastic process of the data-processing Quine also contains an absorbing state, in which the system eventually ends due to the stochastic dilution flux. In the following we compare the baseline robustness of the data-processing Quine to that of the replicating Quine.



(a) Reaction network (b) Short notation

Figure 19: Lattice of organizations of the data-processing Quine: The system may lose one of the species due to the stochastic dilution flux. The system is dead for species sets in the stochastically closed subspace, indicated by the red background in (a). Organizations have a black border; they form a lattice as shown in (b). Note that {} is not an organization because data molecules are continuously injected from outside the vessel.

Figure 19(a) depicts the set transition diagram, the simplified illustration of the underlying Markov process. An even simpler view is provided by the Chemical Organization Theory (COT), which attributes the system two organizations, shown in Figure 19(b): one is the quasi steady-state where all species are present ($\{D, A, R, B\}$) whereas the other is the single absorption

steady-state in which the whole vessel only contains data packets ({D}). Note that the empty set is no organization since packets are continuously injected from outside the vessel.

The states in the shaded area of Figure 19(a) span a stochastically closed subset $\mathcal{E}$, from which the quasi steady-state is unreachable. Instead, the reaction system eventually ends in the absorption organization {D}. In $\mathcal{E}$ the system became inert, meaning that the Quine is not able to replicate anymore and the remaining molecules are eventually displaced by the continuously arriving packets.

Once more, we use Neuts' method (1981) to calculate the baseline robustness, i.e., the survival time, of the data-processing Quine. More explicitly, we calculate the mean first-passage time to the stochastically closed set $\mathcal{E}$ when starting in the quasi steady-state.

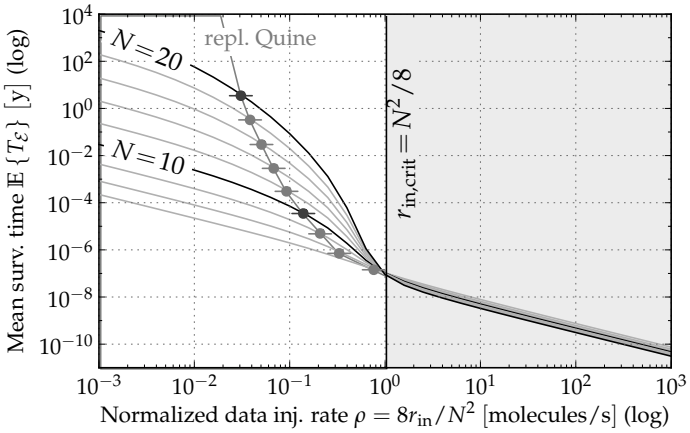


Figure 20: Robustness of the data-processing Quine: Mean survival time in a vessel of capacity N, plotted with respect to the normalized data injection rate $\rho = 8r_{\text{in}}/N^2$. The blue dots indicate the corresponding mean survival time of the idle-looping replicating Quine for comparison. The lifetime drastically drops around and above the critical injection rate (red area). The blue area denotes the packet injection rate for which the lifetime of the data-processing Quine is longer than the lifetime of the replicating Quine in a vessel with the same capacity.

Figure 20 depicts the mean survival time of the data-processing Quine for different vessel capacities N with respect to the normalized data injection rate ρ. The lifetime of the Quine without injected packets and without execution errors or memory alterations is infinite, but its expected lifetime continuously decreases with an increasing packet injection rate. As suspected before based on the deterministic ODE description, the critical injection rate $r_{\text{in}} = N^2/8$, above which there is no deterministic fixed point, sharply limits the survival time for any vessel capacity. This is, for a practical usage the system has to artificially limit the injection rate or the vessel capacity has to be dimensioned for the fastest injection rate possible.

7.6 Dimensioning of the Vessel

In a short thought experiment, we demonstrate how to dimension the vessel capacity for the worst case scenario, this is for the maximal expected packet injection rate, such that the packet loss is small and the lifetime of the data-processing Quine is long enough.

Consider a packet forwarding engine in which a data-processing Quine's job is to send incoming packets to a dedicated next hop j such as the Quine in Figure 17. We aim at dimensioning the vessel capacity N for the case that packets arrive at wire-speed from a Gigabit Ethernet

link.

A Gigabit Ethernet is able to transfer 81275 packets/s if all of them have the maximal size of 1500 bytes (not considering jumbo frames). The vessel capacity for which this is the critical rate is $N = \sqrt{8 r_{\text{in,crit}}} = \sqrt{8 \cdot 81275!} \approx 800$ molecules. We definitely need more capacity because, according to Figure 20, the survival time of the Quine for the critical rate is only a few seconds (10^{-7} years).

Let us consider a ten-fold capacity of $N = 8000$ molecules, resulting in a normalized injection rate of $\rho \approx 0.01$ at wire-speed. We cannot calculate the baseline robustness of such large vessels with reasonable effort anymore as the state space roughly contains 3^{8000} states. Instead, we approximate the survival time: Figure 21 depicts the survival time of the data-processing Quine for different packet injection rates. Already for small Ns we observe that the survival time increases exponentially with the vessel capacity. A rough linear regression on the lin-log plot (dashed lines) indicates that the survival time for the normalized injection rate $\rho = 0.01$ reaches the current age of the universe already for $N \approx 40$ molecules. For the same injection rate the packet loss probability is about 0.5 % according to the parametric fixed point calculation in the previous section. This loss probability is tolerable for most applications.

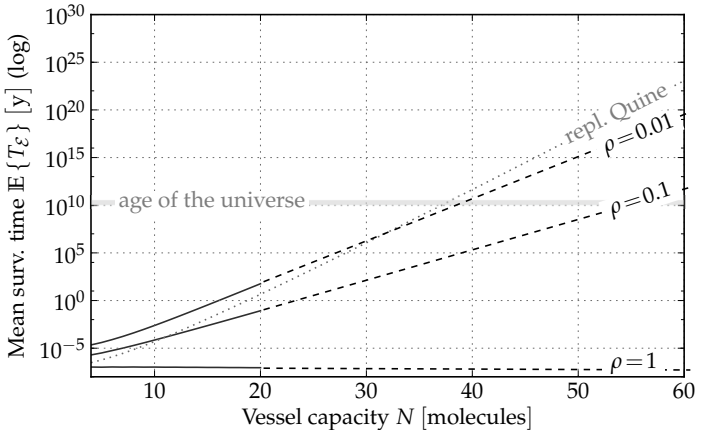


Figure 21: Robustness of the data-processing Quine: Mean survival time for different normalized data injection rates $\rho = 8 r_{\text{in}} / N^2$, plotted with respect to the vessel capacity N in comparison to the mean survival time of the replicating Quine.

Such a worst case calculation is helpful to optimize the system with respect to maximum lifetime, minimum packet loss, and minimum capacity needed. Usually we can carry out this optimization only on the packet loss vs. capacity axis and ignore the system's lifetime, because the expected lifetime increases in order of $\exp(N)$ and reaches a sufficiently high value soon, whereas the packet loss only decreases in the order of N^2.

8 Application Case: Robust Link Load Balancing with a Chemical Protocol

We now make use of the data-processing Quine motif in order to implement a *self-healing protocol* that balances a packet stream over two different network paths such that packet loss is minimized.

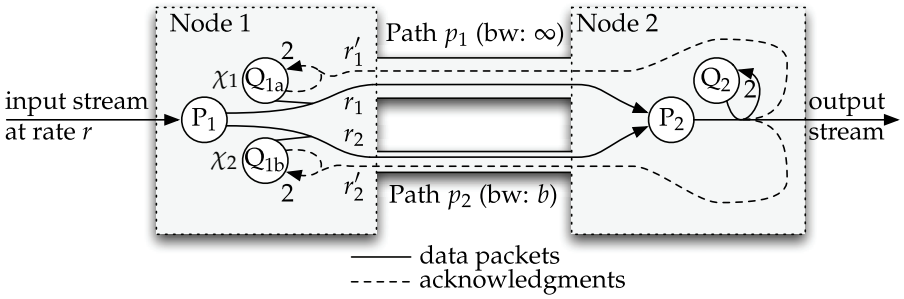


Figure 22: Robust Link Load Balancing Protocol: Data packets to node 2 are injected at rate r into node 1. Quines 1 and 2 (concentration χ_1 and χ_2, respectively) compete for and forward packets at rate r_1 and r_2 over paths p_1 and p_2, respectively. The Quine's replication is controlled by acknowledgments received at rate r'_1 and r'_2, respectively.

As depicted in Figure 22, we inject packets at rate r into node 1 where two Quines, one for each path, compete for them and send them over the corresponding path. Instead of replicating as fast as possible by generating their own reward, the Quines wait for and react with acknowledgment packets that serve as duplication rewards. The third Quine in node 2 sends these acknowledgments back over the reverse path and delivers the packet to the application.

This scheme leads to a perfect packet balance among the two paths. When the reaction vessel in node 1 is saturated, its molecules either belong to Quine Q_{1a} or Q_{1b}, as other molecules have been squeezed out. Let's denote the relative concentrations by χ_1 and χ_2, respectively, satisfying $\chi_1 + \chi_2 = 1$. Since replication is triggered by the received acknowledgments, these concentrations are

$$\chi_i = \frac{r'_i}{\sum_j r'_j} \tag{20}$$

where r'_j is the rate of acknowledgments received over path p_j.

Let's assume that the bandwidth of path p_1 is infinite whereas p_2 drops packets exceeding a rate of b packets/s. We examine the overload situation where the total rate $r > 2b$. Consequently, the rate of acknowledgments is $r'_1 = r_1$ and $r'_2 = \min(r_2, b)$. Due to the law of mass action, the fraction of packets sent over p_1 is proportional to the concentration of Quine Q_{1a}:

$$r_1 = x_1 r = \frac{r_1}{r_1 + \min(r_2, b)} r \tag{21}$$

Hence,

$$r_1 = r - b \quad \text{and} \quad r_2 = r - r_1 = b \tag{22}$$

Quine Q_{1b} reduced its concentration so as to only forward packets up to the bandwidth limitation of path p_2. □

8.1 Surviving Errors

The load balancing protocols reaches (chemical) equilibrium, where it provides the optimal traffic partition. Deviations from the equilibrium, for example by lost molecules on one network path, are compensated by increasing the population of Quines that forward packets over the opposite path. Moreover, the code itself is organized in circuits of self-replicating molecules: In

the face of execution and mutation errors, the system will eventually regenerate the lost code and, after some transition time, autonomously finds back to equilibrium. In fact, packet loss as well as code loss is treated by the same mechanism and in the same way.

Figure 23 depicts the protocol's response to an extraordinary error where the system loses the majority of its molecules. In an OMNeT++ (Varga, 2001; Varga & Hornig, 2008) simulation we fixed the input rate r to 200 pkts/s and the bottleneck capacity to $b = 40$ pkts/s. At time $t = 50$ s, 80 % of all molecules (code and data) are neutralized in node 1. The forwarding rates, as can be seen in the figure, remain unaffected while the code fully regenerates itself within 10 seconds!

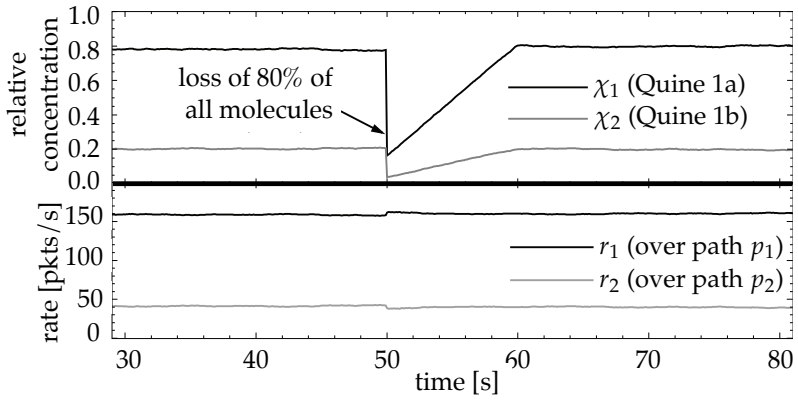


Figure 23: OMNeT++ simulation: Relative concentrations of the forwarding Quines in node 1 during the loss of molecules and the corresponding packet forwarding rate (unaffected).

The load balancing protocol discussed in this chapter may also be used for other tasks. Meyer, Yamamoto, and Tschudin (2008) used it as a self-adapting forwarding engine of a self-healing routing protocol implementation. We point out that the protocol, although self-healing, is not as elaborate as existing load-aware protocols in many respects. However, a noteworthy observation is that it implements *intrinsic bandwidth estimation* by relying on the rate of actual data packets, akin to ACK pacing in TCP (Aggarwal, Savage, & Anderson, 2000), whereas other existing bandwidth estimation techniques numerically calculate the bandwidth based on the rate of separate probe packets or on their inter-arrival time (Easwaran & Labrador, 2004).

9 Future Work

Although the presented approach of using chemical Quines to achieve code-level robustness is promising, there are many open questions left. Some of the areas we will target in the near future are hardware implementations, the design of an optimized instruction set, and the long-term goal of enabling evolvable networking software.

9.1 Virtual Chemical Machine in Hardware

Our move to study fault tolerance at the pure software level implicitly makes assumptions on the underlying execution environment: Ideally, each molecule would be carried by a "physical" thread. However, in our implementation we place a single virtual machine (VM) per node that serves full vessels. An error in the vessel will hurt many molecules instead of single

items. Therefore, one should investigate hardware chip designs where "molecule threads" and "memory access" is highly parallel and as decoupled as possible. Note that due to the self-healing properties of our software, the hardware implementation can exhibit spurious errors. This relates to efforts of Chakrapani et al. (2007) who proposed probabilistic chips, i.e. CMOS devices whose behavior is rendered probabilistic by noise. The advantage of such chips is their feasible manufacturing process in the nanometer scale and their lower energy consumption achieved by a lower operating voltage. The problem of such hardware is that it may expose physical noise up to the instruction level, an environment where traditional programming paradigms and languages obviously fail.

9.2 Instruction Set Design and Probabilistic Network Services

Recently, we observed that some instructions are more fragile to bit errors than others (e.g., the `fork` tag). A redundant binary encoding scheme (forward error correction or just bit error flagging) for Fraglet symbols would lower the fraction of harmful mutations. For example, a fraglet starting with a corrupted symbol would be non-reactive but still exposed to the non-selective dilution flux and will eventually be displaced.

An even more radical approach would be to let an instruction encode a probability distribution of actions on (Fraglets) strings. Currently, we are far from devising such an execution scheme for practical use, but it would resonate well with the mindset that exposing some randomness of networking services to the end user, e.g., accepting packet loss in a video-stream, instead of providing delivery guarantees, is just fine.

9.3 Self-Optimization

We believe that self-healing is a first required step towards evolving software. Quines replicate and compete against each other for the limited memory resources, variation is provided externally by occasional bit alterations or execution errors of the underlying hardware platform. Besides accidental faults – against which we developed the chemical Quines – an unreliable execution environment also bears the chance of a beneficial mutations. Because improvement is rare compared to induced faults, we think that evolving software first needs to be robust. This is backed up by Wagner (2007) who studied the relation between robustness and evolvability and showed that the latter is not achievable without the first. Again, the design of an instruction set enabling evolvable chemical programs is of paramount importance and a challenge for future work.

10 Conclusions

In this work we reported that self-replicating programs – Quines – in a chemical context and equipped with self-induced dynamic behavior, are a basis for self-healing programs and networking protocols. We implemented the Quines in Fraglets, an artificial chemistry developed to design networking protocols, and decorated them in order to achieve different properties and functionality.

When scheduled according to the law of mass action, a population of replicating Quines

grows hyperbolically. In a reaction vessel with limited capacity, this Quine has intrinsic self-healing properties without relying on an external controller.

Because of the behavioral equivalence of the Fraglets reaction regime to real chemical reactions, we are able to use the same mathematical tools used to analyze chemical reactions. For example, phase-type distributions can be used to quantify the survival probability of Quines whereas the Chemical Organization Theory analyzes the steady-states qualitatively.

We found out that certain self-replicating programs are able to survive longer in a networking context,than on a single machine. We also demonstrated the use of Quines for real networking protocols, for example a link load balancing protocol that recovers from execution errors and spurious alteration of bits in the memory.

Unlike other bio-inspired approaches that extract the essence of a natural mechanism and then try to import it into existing networking machinery, we make a step forward by putting chemical laws in the core of packet processing engines: It seems that such an approach simplifies the transfer of desirable robustness properties from nature to manmade systems. Network service robustness must be understood as a code-rewriting task with homeostatic properties where the software continuously and actively has to maintain its healthy state. We believe that this is a necessary requirement for the future's truly adaptive and evolving protocols.

Acknowledgments

This work has been supported by the Swiss National Science Foundation through SNF Project Self-Healing Protocols (2000201-109563). We would also like to thank Lidia Yamamoto for her continuous contribution to our research.

References

Abrash, H. I. (1986). Studies concerning affinity. *Journal of Chemical Education*, *63*, 1044–1047.

Aggarwal, A., Savage, S., & Anderson, T. (2000). Understanding the performance of TCP pacing. In *Proc. 9th annual joint conference of the* IEEE *computation and communication societies (*INFOCOM *2000)* (Vol. 3, pp. 1157–1165).

Anckaert, B., Madou, M., & de Bosschere, K. (2007). A model for self-modifying code. In J. Camenisch, C. Collberg, N. Johnson & P. Sallee (Eds.), *Proc. 8th international conference on information hiding* (Vol. 4437, pp. 232–248). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Banâtre, J. P., Fradet, P., & Radenac, Y. (2006). A generalized higher-order chemical computation model. *Electronic Notes in Theoretical Computer Science*, *135* (3), 3–13.

IBM. (n.d.). Autonomic computing: IBM's perspective on the state of information technology.

Calude, C. S., & Paŭn, G. (2001). *Computing with cells and atoms: an introduction to quantum,* DNA *and membrane computing*. CRC Press.

Campbell, S., Chancelier, J.-P., & Nikoukhah, R. (2006). *Modeling and simulation in scilab/scicos*. New York, NY, USA: Springer.

Chakrapani, L. N., Korkmaz, P., Akgul, B. E. S., & Palem, K. V. (2007). Probabilistic system-on-a-chip architectures. ACM *Transactions on Design Automation of Electronic Systems*, *12* (3), 1–28.

Di Caro, G., & Dorigo, M. (1998). Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, *9*, 317–365.

Dittrich, P. (2005). Chemical computing. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto & O. Michel (Eds.), (Vol. 3566, pp. 19–32). Lecture Notes in Computer Science. Berlin / Heidelberg: Springer.

Dittrich, P., & Banzhaf, W. (1998). Self-evolution in a constructive binary string system. *4* (2), 203–220.

Dittrich, P., & Speroni di Fenizio, P. (2007). Chemical organization theory. *Bulletin of Mathematical Biology*, *69* (4), 1199–1231.

Dittrich, P., Ziegler, J., & Banzhaf, W. (2001). Artificial chemistries - a review. *7* (3), 225–275.

Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., ... Zambonelli, F. (2006). A survey of autonomic communications. *1* (2), 223–259.

Easwaran, Y., & Labrador, M. (2004). Evaluation and application of available bandwidth estimation techniques to improve TCP performance. In *Proc. 29th annual* IEEE *international conference on local computing networks* (pp. 268–275).

Fernando, C., & Rowe, J. (2007). Natural selection in chemical evolution. *Journal of Theoretical Biology*, *247* (1), 152–167.

Fontana, W., & Buss, L. W. (1994). ''The arrival of the fittest'': toward a theory of biological organization. *Bulletin of Mathematical Biology*, *56* (1), 1–64.

Freitas Jr., R. A., & Merkle, R. C. (2004). *Kinematic self-replicating machines*. Georgetown, TX, USA: Landes Bioscience.

Gelenbe, E., Lent, R., & Xu, Z. (2001). Design and performance of cognitive packet networks. *Performance Evaluation*, *46* (2–3), 155–176.

Ghosh, D., Sharman, R., Rao, H. R., & Upadhyaya, S. (2007). Self-healing systems - survey and synthesis. *Decision Support Systems*, *42* (4), 2164–2185.

Gibson, M. A., & Bruck, J. (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, *104* (9), 1876–1889.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, *81* (25), 2340–2361.

Gillespie, D. T. (1992). A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, *188* (1–3), 404–425.

Gillespie, D. T. (2007). Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, *58*, 35–55.

Glaz, J. (1979). Probabilities and moments for absorption in finite homogeneous birth-death processes. *Biometrics*, *35* (4), 813–816.

Hofstadter, D. (1979). *Gödel, Escher, Bach: an eternal golden braid*. Basic Books.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press.

Hutton, T. J. (2002). Evolvable self-replicating molecules in an artificial chemistry. *Artificial Life*, *8* (4), 341–356.

Johnson, B. W. (1996). An introduction to the design and analysis of fault-tolerant systems. In *Fault-tolerant computer system design* (pp. 1–87). Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *36* (1), 41–50.

Kleene, S. C. (1938). On notation for ordinal numbers. *Journal of Symbolic Logic*, *3* (4), 150–155.

Langton, C. G. (1984). Self-reproduction in cellular automata. *Physica D: Nonlinear Phenomena*, *10* (1–2), 135–144.

Meyer, T., & Tschudin, C. (2009). Chemical networking protocols. In *Proc. hot topics in computer networks (hotnets-*VIII*)*. ACM.

Meyer, T., & Tschudin, C. (2010). *Competition and cooperation of self-healing software* (Technical Report No. CS-2010-004).

Meyer, T., Yamamoto, L., & Tschudin, C. (2008). A self-healing multipath routing protocol.

Neuts, M. F. (1981). *Matrix-geometric solutions in stochastic models*. New York, NY, USA: Dover Publications Inc.

Normand, E. (1996). Single event upset at ground level. IEEE *Transactions on Nuclear Science*, *43* (6), 2742–2750.

Paŭn, G. (2000). Computing with membranes. *Journal of Computer and System Sciences*, *61* (1), 108–143.

Perrier, J.-Y., Sipper, M., & Zahnd, J. (1996). Toward a viable, self-reproducing universal computer. *Physica D*, *97*, 335–352.

Pradhan, D. K. (1996). *Fault-tolerant computer system design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Reis, G. A., Chang, J., & August, D. I. (2007). Automatic instruction-level software-only recovery. IEEE *Micro*, *27* (1), 36–47.

Shaw, M. (2002). ''Self-healing'': softening precision to avoid brittleness. In *Proc. 1st workshop on self-healing systems (*WOSS *'02)* (pp. 111–114). New York, NY, USA: ACM.

Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge, MA, USA: MIT Press.

Sipper, M. (1998). Fifty years of research on self-replication: an overview. *Artificial Life*, *4*, 237–257.

Stadler, P. F., Fontana, W., & Miller, J. H. (1993). Random catalytic reaction networks. *Physica D, 63* (3–4), 378–392.

Strogatz, S. H. (1994). *Nonlinear dynamics and chaos*. Studies in Nonlinearity. Westview Press.

Szathmáry, E. (1991). Simple growth laws and selection consequences. *Trends in Ecology and Evolution, 6* (11), 366–370.

Tempesti, G., Mange, D., & Stauffer, A. (1998). Self-replicating and self-repairing multicellular automata. *4*, 259–282.

Teuscher, C. (2007). From membranes to systems: self-configuration and self-replication in membrane systems. *Biosystems, 87* (2–3), 101–110.

Thompson, G. P. (2010). The Quine page (self-reproducing code).

Tschudin, C. (n.d.). Fraglets home page.

Tschudin, C. (2003). Fraglets - a metabolistic execution model for communication protocols. In *Proc. 2nd annual symposium on autonomous intelligent networks and systems (*AINS*)*.

Varga, A. (2001). The omnet++ discrete event simulation system. In *Proc. european simulation multiconference (*ESM *2001)*.

Varga, A., & Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Proc. 1st internation conference on simulation tools and techniques for communications (*SIMUTOOLS *'08)* (pp. 1–10).

von Neumann, J. (1966). *Theory of self-reproducing automata*. Champaign, IL, USA: University of Illinois Press.

Wagner, A. (2007). *Robustness and evolvability in living systems* (S. A. Levin & S. H. Strogatz, Eds.). Princeton Studies in Complexity. Princeton University Press.

White, T., & Pagurek, B. (1998). Towards multi-swarm problem solving in networks. *Multi Agent Systems, 1998. Proceedings. International Conference on*, 333–340.

Wilfredo, T.-P. (2000). *Software fault tolerance: a tutorial*.

Wong, V., & Horowitz, M. (2006). Soft error resilience of probabilistic inference applications. In *Proc. 2nd workshop on system effects of logic soft errors (*SELSE*)*.

Yamamoto, L., Schreckling, D., & Meyer, T. (2007). Self-replicating and self-modifying programs in fraglets. In *Proc. 2nd international conference on bio-inspired models of network, information, and computing systems (*BIONETICS *2007)*.