A Self-Healing Multipath Routing Protocol

Thomas Meyer
Computer Science Department
University of Basel
Bernoullistrasse 16
CH-4056 Basel, Switzerland
th.meyer@unibas.ch

Lidia Yamamoto
Computer Science Department
University of Basel
Bernoullistrasse 16
CH-4056 Basel, Switzerland
lidia.yamamoto@unibas.ch

Christian Tschudin
Computer Science Department
University of Basel
Bernoullistrasse 16
CH-4056 Basel, Switzerland
christian.tschudin@unibas.ch

ABSTRACT

This paper introduces a self-healing multipath routing solution that is resilient to code knock-out attacks i.e., the software copes with changing network topologies and perturbations of link characteristics as well as loss of parts of its code base. We use the Fraglets artificial chemistry for building a distributed reaction network that spans the whole network topology. Unlike the use of explicit metric values in today's network protocols, our approach relies on the *concentration* of routing table entries to stochastically decide which path a packet should take through the network. A reinforcement mechanism rewards successful forwarding rules that compete against each other for delivering data packets over alternative paths. Our self-healing solution is based on the systematic use of self-replicating code that constantly rewrites itself.

Keywords

Chemical computing, routing, protocols, self-healing, Fraglets

1. INTRODUCTION

The rapidly increasing demands on today's computer networks lead to more complex distributed applications. Soon, human supervision and configuration of those systems will not be feasible anymore, because the complexity of human intervention is increasing, too. Therefore IBM introduced the vision of Autonomic Computing [8]. According to this concept regulatory interventions shall be performed autonomously by the system to a large extent. The goal is to achieve system properties like self-configuration, self-healing, self-optimization, and self-protection.

By now network protocols try to cope with perturbations within the network. For example, a transport protocol like TCP tolerates and corrects corrupted packet delivery and adapts its transmission rate to the bandwidth of the end-to-end transmission path; routing protocols re-route packets around defective links or nodes. Some latest routing protocols even use bio-inspired concepts [5, 11] since biological systems exhibit the desired self-organizing properties.

In our research we are interested in software that is able to detect and repair erroneous code execution originating either from programming mistakes or from an unreliable execution environment like probabilistic chips [4]. In most programming languages available today it is hardly possible to monitor the program's execution and recover from a deviation from the expected behavior. Moreover, repairing the program implies that we are able to modify the code, for which a sound theoretical framework [3] is still missing.

We achieve self-healing software by means of an artificial chemistry called Fraglets. In Fraglets [18] virtual molecules are used to model code, data and network packets. These molecules react with each other and – in doing so – compute results by rewriting themselves. Within this framework it is easier to modify code at run-time by creating new molecules that enable new reactions and thereby change the reaction and computation flow.

The main contribution of this paper is to present a self-healing routing protocol. We organize the corresponding software as cooperative self-replicating program parts that constantly regenerate themselves. Limited resources then lead to the necessary competition where defective parts that are not able to replicate anymore (e.g., due to some code knock-out attack) are expunged. As a result, only viable program parts survive. The protocol consists of two intertwined subsystems: route dissemination and regulation. Both use code self-replication to withstand code deletion attacks.

This paper is structured as follows: Section 2 provides an introduction to the Fraglets artificial chemistry, demonstrates how to achieve self-healing software in this framework and shows the methods we used to analyze the resulting chemical programs. Section 3 contains the description of the self-healing multipath routing protocol. In Sect. 4 we present simulation results of typical network perturbations as well as deletion attacks towards the own code base, before we compare to related work and assess the results in Sect.5.

2. AN ARTIFICIAL CHEMISTRY FOR PROTOCOL EXECUTION

This section presents the context in which we have developed the routing protocol. We first summarize the Fraglets artificial chemistry. Then we show how self-healing code can be obtained in this system. Finally, we provide a method for predicting the behavior of protocols that have been designed using this model.

2.1 Fraglets

Fraglets [18] is an artificial chemistry characterized by the triple (S, R, A) (in accordance to [6]). The set of molecules S is the set of all possible words w of arbitrary length over a finite alphabet of symbols Σ, thus $S = \{w | w \in \Sigma^*\}$. (In this paper we use the terms "molecules" and "fraglets" interchangeably.) Fraglets is an instance of a tag system [14], a string rewriting system in which the leftmost symbol identifies the rule to apply. The set of reaction rules R is therefore implicitly defined by a finite set of production rules that operate on molecules. A subset of the Fraglet production

Bionetics '08, November 25-28, 2008, Hyogo, Japan
Copyright 2008 ICST 978-963-9799-35-6.

Table 1: A subset of Fraglet production rules. `A`, `B`, and `X` are symbols $\in \Sigma$, `TAIL`, and `NEXT` are words $w \in \Sigma^*$.

Instr.	Substitution pattern
sexch	`[sexch NEXT A B]` $\rightarrow$ `[NEXT B A]`
snode	$_{n_i}$`[snode X NEXT]` $\rightarrow {}_{n_i}$`[NEXT` n_i`]`
send (unicast)	$_{n_i}$`[send` n_j `TAIL]` $\rightarrow \begin{cases} {}_{n_j}\texttt{[TAIL]} & \text{if } (n_i, n_j) \in E \\ \epsilon & \text{otherwise} \end{cases}$
send (broadcast)	$_{n_i}$`[send all TAIL]` $\rightarrow_{n_j}$`[TAIL]` $(\forall j : (n_i, n_j) \in E)$
match	`[match A TAIL1]+[A TAIL2]` $\rightarrow$ `[TAIL1 TAIL2]`
matchs	`[matchs A TAIL1]+[A TAIL2]` $\rightarrow$ `[TAIL1 TAIL2]+[A TAIL2]`
matchp	`[matchp A TAIL1]+[A TAIL2]` $\rightarrow$ `[matchp A TAIL1]+[TAIL1 TAIL2]`

rules is shown in Tab. 1. For example, molecule

`[sexch NEXT A B]` $\rightarrow$ `[NEXT B A]`

starts with instruction `sexch`, which treats its tail as data stack (the last symbol is the top of the stack) and swaps the last symbols. For example, when applied to `[sexch a b c d e]` this will result in `[a b c e d]`. Instructions can also be manipulated in this way (*code rewriting*). Note that stack instructions are a new addition to the original instruction set.

A Fraglet node, or reaction vessel, contains a multiset of fraglets. Nodes communicate by exchanging fraglets. Formally, the network is described by a graph $G = \{N, E\}$, where the set of nodes $N = \{n_1, \ldots, n_k\}$ is connected by bidirectional network links or edges $E = \{e_1, \ldots, e_l\}$. Consider Fig. 2 for an example topology. Two nodes n_i and n_j are called neighbors iff $\exists e = (n_i, n_j) \in E \vee \exists e = (n_j, n_i) \in E$. In this case we define $\text{adj}(n_i, n_j) = \text{adj}(n_j, n_i) = 1$, or 0 if there is no direct connection between the nodes. Nodes are able to send fraglets over the attached links in either broadcast, anycast or unicast mode (see the `send` instruction in Tab. 1).

The third element of an artificial chemistry triple, A, is an algorithm that describes the dynamic behavior of the system. At any time, several chemical reactions are potentially viable, but only one will be executed. The algorithm A selects which molecules have to be processed in the next iteration and how long the system has to wait before the next reaction shall occur. Fraglets uses the Gillespie algorithm [7] that models the chemical law of mass action [2].

Each node is an independent artificial chemical reactor, described by its own triple (S_i, R_i, A_i). The transmission of fraglets between two neighbor nodes n_i and n_j is done by the `send` instruction, which converts a molecule $M_i \in S_i$ in vessel n_i into the same word $M_j \in S_j$, but located in the neighbor vessel n_j, or $M_i \rightarrow M_j$ in the notation of chemical reactions. This spans a global distributed reaction system defined by the triple (S, R, A), where $S = \bigcup_{i \in N} S_i$ and $R = \bigcup_{i \in N} R_i$.

2.2 Self-Healing Code

We aim at software that is robust to the destruction of parts of its own code base. The software has to continuously observe its own health state, detect an attack and autonomously take counteractions by repairing itself. Obviously, this behavior cannot be performed by static code; the program must have the possibility to modify its own code at runtime. A kind of software homeostasis as demanded in [15] can be achieved by self-replicating program parts where each operational block replicates its own code while being executed.

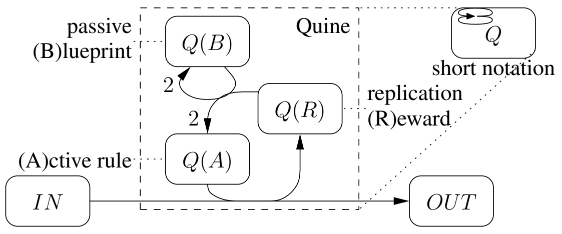


Figure 1: Autocatalytic quine

In [13] we proposed to use *autocatalytic quines* to achieve self-healing code. In computer science, “quines” are programs that produce their own source code as output [17]. An autocatalytic quine is a program fragment that produces more than one copy of itself and therefore self-replicates. In Fraglets this can be realized by a set of three fraglets. Figure 1 shows the reaction network of such an autocatalytic quine. The cycle starts when an input data molecule (IN) is available: The active rule ($Q(A)$) reacts with the data molecule (and disappears). The reaction product performs the desired computation and generates both a molecule representing the result of the computation (OUT) as well as a replication reward ($Q(R)$). The latter molecule consumes the blueprint of the quine ($Q(B)$), which contains a description of how to regenerate *two* copies of the active rule and of itself. Because of this duplication, the number of quine molecules will grow exponentially if the input molecules are continuously replenished.

In order to prevent unbound growth of a reaction vessel we extend the reaction algorithm A to keep track of a vessel's capacity N. After each iteration, the extended reaction algorithm examines the multiset M and counts the number of molecules exceeding N ($\Delta = |M| - N$). If $\Delta > 0$ the algorithm randomly selects Δ molecules in M and removes them from the multiset. This creates a “dilution flow” that operates randomly on all molecules of a vessel as soon as it would overflow.

Reaction chains with no growth in molecules would quickly become extinct in case they are put in competition with autocatalytic quines, which grow in number. The many duplicates of the quine in Fig. 1, for example, will have a better chance to occupy the space of the vanishing molecules than the molecules from a static chain. Considering now a competition between autocatalytic quines, survival will depend on how aggressive their growth is. If several quines process the same stream of data molecules, for example if the second quine consumes the data molecules produced by the first quine, their replication is coupled. If one of the quines in this processing chain is eliminated completely, the whole chain will die out. However, we have previously shown that if at least one of each quine instances is present, the whole program built out of such cooperating quines is able to survive [13]. There we also examined under which circumstances such quines are resistant to other attacks such as mutations.

The protocol software discussed in this paper makes use of coupled autocatalytic quines such that the program is robust to the destruction of any participating molecule. In the whole system, code portions as well as data molecules, like routing table entries, continuously rewrite themselves.

2.3 Dynamical Analysis

Algorithm A, the Gillespie algorithm, stochastically simulates chemical reaction rates according to the law of mass action. In every iteration it calculates which reaction occurs next and when this reaction occurs based on a probabilistic selection. The dynamic behavior of the chemical reaction network can be predicted by means of a deterministic approach. Putting the dilution flow aside for a moment, the time evolution of the concentration vector can be ap-

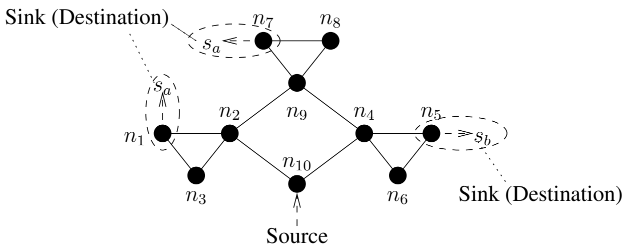


Figure 2: Network topology with 10 nodes and two services.

proximated by a system of ordinary differential equations (ODEs)

$$\dot{\tilde{\mathbf{n}}}(t) = \mathbf{S}\mathbf{v} \tag{1}$$

where $\tilde{\mathbf{n}}(t) = (\tilde{n}_1, \ldots, \tilde{n}_m)^T$ is a vector of the number of molecules of each type in the reaction vessel, $\mathbf{S}$ is the stoichiometric matrix, and $\mathbf{v} = (v_1, \ldots, v_r)^T$ the reaction velocity vector according to the law of mass action: For instance, the velocity of a bimolecular reaction r, $2A + 3B \xrightarrow{k} C$, is given by $v_r = kn_A^2 n_B^3$, where k is a given kinetic constant for the reaction.

In order to take into account the dilution flow, one has to model the removal of the excess molecules after each iteration step. Stadler et al. showed in [16] that such systems can be described by the *catalytic network equation*. In our case this can be written as

$$\dot{\mathbf{n}}(t) = \dot{\tilde{\mathbf{n}}}(t) - \frac{\mathbf{n}(t)}{N}\Phi(t) \tag{2}$$

where $\Phi(t)$ is the total overproduction given by

$$\Phi(t) = \sum_{i=1}^{m} \dot{\tilde{n}}_i(t) \tag{3}$$

Unfortunately, such a system of differential equations cannot be solved analytically, even for a simple reaction network like that of a single autocatalytic quine. Instead, one has to use a numerical integrator available in mathematical packages like Scilab [1]. Figure 5 shows the concentration traces of a reaction network for both, the stochastic Fraglet simulation as well as for the deterministic prediction using Scilab. The meaning of the reaction network that generates these values will be explained in a later section; here we point out that the deterministic calculation accurately predicts the stochastic simulation.

3. SELF-HEALING ROUTING CODE

The task of a routing protocol is to setup forwarding state in a network's nodes such that data packets can be forwarded across the net. At the same time, the forwarding state should maximize throughput and minimize packet losses. In our case we additionally wish that the routing software is able to tolerate code deletion attacks, meaning that the routing protocol continues to operate and self-heals as soon as possible.

3.1 Definitions

Throughout this paper we will base our simulations on the network topology as shown in Fig. 2 which consists of 10 nodes connected by bidirectional links. At selected nodes we have *services* which are the final destinations to which data packets shall be delivered. Each service, as well as each node, has a globally unique flat identifier.

The same service may be replicated and made available at different locations in the network. In this case, data packets can be sent to any node that provides this service. Figure 2 shows that service

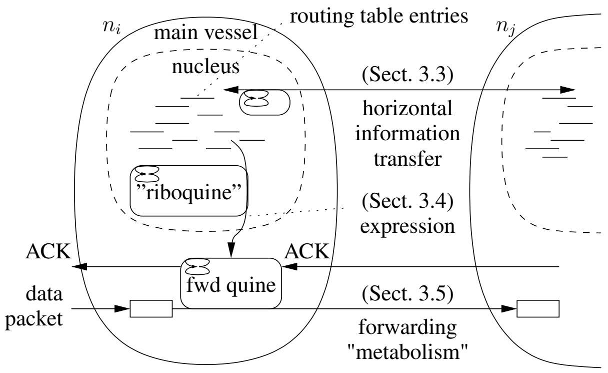


Figure 3: Cell metaphor used to structure the protocol

s_a is registered at nodes n_1 and n_7. This enables a data centric network model [9] where packets for a given destination service are forwarded to any (preferentially to the closest) node where the service is present. In this model, the routing protocol becomes responsible for disseminating the availability of all services in the net.

Formally, we extend the topology by a set of services $S = \{s_a, s_b\}$ and a set of associations between services and nodes (bindings) $B = \{(s_a, n_1), (s_a, n_7), (s_b, n_5)\}$. A service s_m is bound to node n_i iff $\exists b = (s_m \in S, n_i \in N) \in B$. Alternatively, we can define a function $\mathrm{bnd}(s_m, n_i) = 1$ if service s_m is bound to node n_i, or 0 otherwise.

3.2 Protocol Overview

Our protocol was inspired by the structure of an eukaryotic cell [12] which features a nucleus. Each network node has two reaction vessels: The main vessel serves as forwarding engine that forwards incoming data packets to one of the neighbor nodes (see Fig. 3). A second vessel called "nucleus" contains the "genome", which in our case accumulates information about the topology of the network in form of routing table entries. Linking both vessels, there are "riboquines" (inspired by ribosomes in cells) which are responsible for "expressing" the nucleus' routing table entries into forwarding rules in the main vessel.

The first task of our protocol is to **gather information about the network topology**: Each node has to obtain knowledge about which service can be reached over which neighbor node(s) as in traditional distance vector protocols. Unlike in traditional routing protocols, we do not aim at immediately finding the best path to a service; instead, the transmission paths are later reinforced by the forwarding engine.

Path reinforcement is based on a competition and reward mechanism for forwarding rules. Forwarding rules are <destination service, next hop> tuples represented by an active fraglet that potentially reacts with a passive data packet destined to that service. On each node the Gillespie algorithm randomly selects one of the forwarding rules matching the packet's destination, based on the corresponding concentrations. When a packet finally reaches the destination service, an acknowledge packet is sent back along the reverse path. This acknowledge packet reacts with all corresponding rules that previously forwarded the data packet and triggers their replication. Thus, whenever a forwarding rule contributed to the successful delivery of a packet, it is duplicated, thereby increasing its concentration and hence the probability of being chosen for a next forwarding task.

The reward mechanism just described results in the most efficient path to get reinforced. Less efficient paths quickly vanish due to the dilution flow: If one forwarding rule is allowed to replicate, another

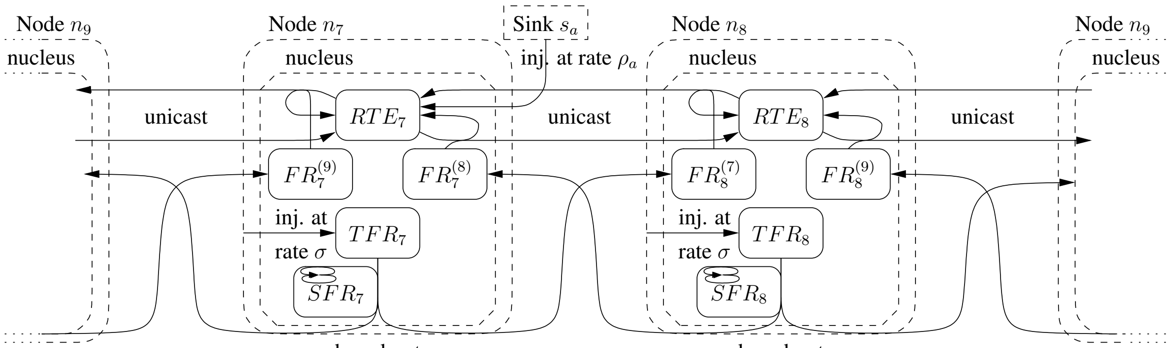


Figure 4: Distributed reaction network to disseminate routing table entries.

rule has to die. This means that if there is a preferential path to a certain destination, the corresponding forwarding rules may completely displace rules for a suboptimal alternative path. This would be a problem if the forwarding rules would be the only place where we keep the topological information about the network's structure. However, this information is kept inside the nucleus which effectively shields connectivity data from the fierce competition and path optimization. The role of the nucleus is therefore to **cultivate a variety of alternative paths** and to continuously inject them into the forwarding engine. Even if suboptimal paths are squeezed out when another path suddenly becomes more attractive, the nucleus provides the initial rule for it, which can be re-instantiated later.

In Sect. 3.3 we first show how routing information is disseminated among the nuclei to build a routing table entry multiset in each node. Then, in Sect. 3.4 we discuss the "riboquines" before we analyze the self-optimizing forwarding engine in Sect. 3.5.

3.3 Dissemination of Routing Table Entries

In the context of an artificial chemistry a routing table may be represented by a multiset of molecules where each molecule instance contains a $\langle s_m, \mathrm{path}(s_m)\rangle$ tuple, where $\mathrm{path}(s_m)$ is a list of nodes along the path to the destination service. In order to keep a high variety of alternative paths to the same destination, a dissemination protocol periodically obtains copies of routing table entries from all neighbors. Figure 4 shows the resulting distributed reaction network for a subnet of our topology, and for a given node n_i it can be formally described as

$$\forall n_i \in N: \tag{4}$$

$$\emptyset \xrightarrow{\rho_r} RTE_i \quad \forall s_r \in S : \mathrm{bnd}(s_r, n_i) = 1$$

$$\emptyset \xrightarrow{\sigma} TFR_i \tag{5}$$

$$TFR_i + SFR(A)_i \longrightarrow SFR(R)_i + \sum_{n_j \in N} \mathrm{adj}(n_i, n_j) FR_j^{(i)} \tag{6}$$

$$SFR(R)_i + SFR(B)_i \longrightarrow 2SFR(A)_i + 2SFR(B)_i \tag{7}$$

$$FR_i^{(j)} + RTE_i \longrightarrow RTE_i + RTE_j \tag{8}$$

Every node to which a service is bound periodically injects service specific routing table entries into its nucleus. For example, node n_7 (Fig. 2) injects an entry referring to service s_a (`n7[RTE sa]`) into its nucleus subvessel, at a certain rate ρ_a (Fig. 4). In the reaction graph, a routing table entry in node n_i is referred to as RTE_i; the formal chemical reaction is shown in Eq. (4).

The dissemination program in node n_i is realized using an autocatalytic quine SFR_i which consists of three molecules $SFR(A)_i$, $SFR(R)_i$, and $SFR(B)_i$. The replication of SFR_i (7) is triggered by a periodically injected fraglet TFR_i (5) whose injection rate σ sets the pace for the exchange of routing table entries. Consequently, the quine broadcasts active fetch route packets ($FR_j^{(i)}$) to all neighbors at rate σ (6). This is also the rate at which entries from neighbor nodes are fetched, by the $FR_j^{(i)}$ fraglets (8). An incoming $FR_j^{(i)}$ fraglet reacts with a randomly selected routing table entry RTE_j on the remote side, makes a copy of it, appends its own node identifier n_j to the copy, and sends it back to the node that originally sent the fetch request. This is shown in the following reaction trace:

```
nj[matchs RTE snode _ send ni RTE]  ($FR_j^{(i)}$)
+ nj[RTE sa]                        ($RTE_j$)
→ (nj[RTE sa])                      ($RTE_j$)
+ nj[snode _ send ni RTE sa]
→ nj[send ni RTE sa nj]
→ ni[RTE sa nj]                     ($RTE_i$)
```

The last line shows that node n_i ultimately collected an RTE entry which states that service s_a is reachable via node n_j. The multiple RTE entries will accumulate in the nucleus and will be subject to a dilution flow: This automatically removes entries for services that no longer exist.

From the reactions in Eqs. (4)–(8) we generated the differential equation system according to the catalytic network equation and simulated them in Fraglets. Figure 5 shows the outcome in both cases for the scenario where node n_{10} joins the network at $t = 300$ and node n_2 leaves the network at $t = 500$. At $t = 300$, the concentration of s_a molecules starts to rise in node n_{10} because it starts to broadcast fetch route messages to its neighbors (Fig. 5). Because of the dilution flow the concentration reaches a steady-state in which there are more entries to service s_a over the link to neighbor n_2 than n_4. When node n_2 disconnects at $t = 500$ s, the concentration of routing table entries for s_a slowly change to favor the remaining link via n_4.

The routing table is slow in adapting to topological changes. E.g. it takes about 500 s to forget about link (n_{10}, n_2) after node n_2 has been switched off. But since we are only interested in the diversity of alternative paths, this does not matter. The forwarding engine will adapt faster on changing link characteristics as we will discuss later.

Note that the routing table entries are passive molecules, i.e. they cannot perform actions themselves. The next task for our protocol is to generate active forwarding rules based on the passive entries.

3.4 Expression of Routing Table Entries

The riboquines[1], introduced in Sect. 3.2, "express" the collected routing table entries for ultimately producing the forwarding rules. Riboquines are periodically triggered at rate ϕ to translate the nu-

[1]We use the word "riboquine" in analogy to the ribosomes in eukaryotic cells that translate genetic information (mRNA) into proteins [10].

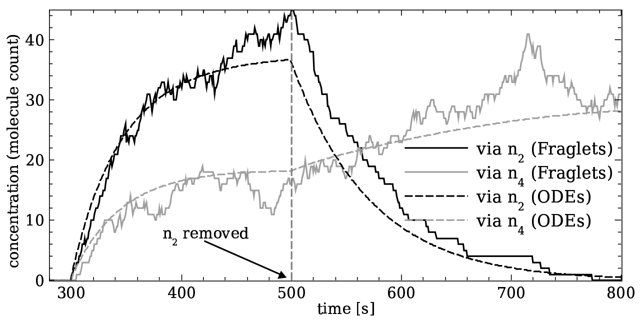


Figure 5: Routing table entries for service s_a in node n_{10} (see topology in Fig. 2) of a stochastic simulation in Fraglets, and an ODE prediction.

cleus' passive routing table entries into active forwarding rules for the node's main vessel.

The riboquines are located in the nucleus[2], although they expel the generated active rules to the outer vessel. This location is important because it protects the molecules involved in the dissemination and expression of routing table entries from the competition for resources in the outer vessel. Note that in a steady-state when there are no data packets to be forwarded in the outer vessel, the concentration of forwarding rules is proportional to the concentration of the corresponding routing table entries in the nucleus.

The riboquine generates two different kinds of forwarding rules, one for the case where the target service is locally present, and another one for a remote service. Technically, the first case can be recognized by the fact that the corresponding RTE fraglet has length 2, for example `[RTE sa]` for a local service s_a, meaning that we should install a local delivery rule like `[match sa deliver]`. In the other case the entry has length greater than 2 (e.g. `[RTE sa n1 n2]`) and the riboquine expresses a forwarding rule of the type `[match sa send n2 sa]`. The exact structure and operation of the forwarding rules will be discussed in the next section.

3.5 Forwarding Path Reinforcement

A natural way of implementing a forwarding engine in Fraglets is to use active molecules (`[match sx send ...]`) as forwarding rules: A data packet of the form `[sx...]` will react with such a rule, leading to the actual forwarding (`send`). Such forwarding rules are injected by the "riboquines". For example, after some time the forwarding state in node n_{10} could look like

```
n10[match sa send n2 sa]60
n10[match sa send n4 sa]20
n10[match sb send n4 sb]40
```

There are two competing rules for a data packet `[sa my data]`. According to the law of mass action, the reaction probability is proportional to the concentration of the reaction rules. In our example, 75 % of the packets to s_a will be sent to neighbor n_2 while 25 % travel over n_4. The following reaction trace shows how the first forwarding rule reacts with a data packet and sends it to the next hop:

```
n10[match sa send n2 sa] + n10[sa my data]
→ n10[send n2 sa my data] → n2[sa my data]
```

A forwarding rule is therefore consumed for each data packet that is sent. The rate of replenishment by the riboquines may not match the rate of the incoming data packets. Therefore a mechanism is needed that regulates the concentration of forwarding rules by selectively replicating them.

[2]Unlike in biological cells where ribosomes are located in the rough endoplasmic reticulum within the cytoplasm.

Replicating Forwarding Rules. As sketched in Sect. 3.2, we will install a stream of feedback fraglets that reward those rules which participated in a successful delivery of a data packet. Recall from Fig. 1 that a reward $Q(R)$ for a quine act as a trigger for its replication. We now defer the reward until an acknowledgment is received as depicted in Fig. 6. Because the reward has to travel backwards, we construct the reverse route during the forwarding by appending the id of each node that the data packet traversed.

At the end of the forwarding chain, a delivery quine creates an acknowledge packet and embeds the collected rewards inside. This ACK fraglet travels backwards and, by successively delivering the rewards, triggers the replication of all those quines that created a good forwarding rule. Consequently, these quines increase their relative concentration compared to the other forwarding quines in the same node, resulting in a higher strength of that forwarding path compared to competing paths.

If the stream of data packets travels over an overloaded link, some of the (data as well as ACK) packets fall victim to buffer overflows and are lost. Naturally, the concerned forwarding quine will never receive a reward in such cases: It is indirectly punished by the fact that there may be a competing path that is able to replicate its quines faster.

4. RESULTS

This section shows how our protocol responds to link, delay and loss disruptions, as well as code deletion attacks. We will always use the network topology shown in Fig. 2 and study the impact on a data stream from a source in node n_{10} to destination service s_a. First, in Sect. 4.1 we show that after a link outage the data stream is deviated over other available links. In Sect. 4.2, we demonstrate that packets are preferably sent over links without packet loss with little impact on the data stream itself. Section 4.3 uncovers a rather counterintuitive phenomenon: delayed paths seem to have a small benefit compared to ideal ones. Finally, in Sect. 4.4 we highlight the robustness of the protocol software against code and data deletion attacks.

4.1 Link Loss

The most typical topological change in a network is the temporary outage of a link. Here we consecutively disconnect link (n_{10}, n_2), and link (n_{10}, n_4). Aside from these outages, the links are ideal: they neither lose packets nor deliver them delayed. The results are summarized in Fig. 7.

At $t = 200$ s, the concentration of forwarding rules (Fig. 7(a)) reflects the concentration of the corresponding routing table entries in the nucleus which favors the transmission path over neighbor n_2. At $t = 300$ s, we start injecting data packets with a rate of $20\,\frac{\text{packets}}{\text{s}}$. Consequently, as shown in Fig. 7(b), most of the traffic is sent to n_2 and thus this forwarding rule is able to replicate faster than the competing rule over n_4.

At $t = 400$ s, the link (n_{10}, n_2) is disconnected. The remaining rules for $\langle s_a, n_2\rangle$ still send packet over the broken link, but they don't receive acknowledgements and are not able to replicate anymore; the packet loss rises to 100 % (Fig. 7(c)). The death of these rules is to the credit of $\langle s_a, n_4\rangle$ rules, which are able to gain concentration and probability of being chosen. Hence the data traffic is redirected over the remaining link.

Later, at $t = 500$ s, we reconnect link (n_{10}, n_2). Fig. 7(c) shows similar packet loss rates and end-to-end delays for either link. Thus the quine over n_2 only has a slight advantage, resulting in a weak attraction towards this path. This situation changes when the now preferred link (n_{10}, n_4) is dropped at $t = 700$ s, rules over n_4 starve and the link over n_2 is strongly preferred.

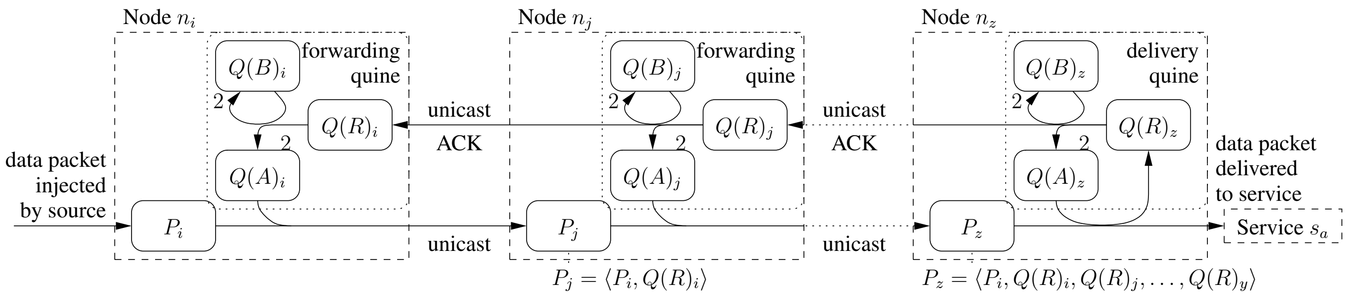


Figure 6: Reaction network of a single forwarding path as a chain of autocatalytic forwarding quines with deferred replication.

4.2 Packet Loss

Next, we examine the situation where the link (n_{10}, n_4) exhibits a packet loss probability of 20 % while the remaining network is still ideal. Figure 8 shows these results.

After $t = 300$ s, when data packet are injected, the forwarding rules over the loss-free link receive all acknowledgments and therefore win the competition. Consequently, rules over n_4 soon become extinct. Unfortunately, this means that there is no alternative forwarding rule when the primary link is disconnected at $t = 400$ s. In the next 100 s no packets are forwarded anymore until a data packet is finally consumed by one of the rules that are continuously generated by the riboquine. After this obstacle is overcome, the rule quickly becomes stronger, because there is no competition. The observed down time can be reduced by increasing the rate ϕ at which the riboquine injects new forwarding rules. This again highlights the importance of the separate nucleus which maintains the diversity of alternative paths. Finally, at $t = 500$ s, when we reconnected link (n_{10}, n_2), the lossless path quickly outperforms the alternative path over n_4 as desired. A direct comparison of Figs. 7(b) and 8(b) shows that the attraction of the path over n_2 is much stronger in the latter scenario.

4.3 Delay

Finally, we examine the behavior of the protocol when having links with delayed packet delivery. We again start with an ideal network with the exception that link (n_{10}, n_4) exhibits a delay of 1 second. A simulation trace of this scenario is shown in Fig. 9(c).

Surprisingly, after $t = 500$ s, when link (n_{10}, n_2) is reconnected, the re-adaption until $t = 700$ s is slower than for the ideal network: The concentration of forwarding rules (Fig.9(a)) over the delayed link to n_4 is not suppressed as quickly as in Fig. 7(a) where the link has no delay. Even though path n_2 wins the competition, the delayed path has a small benefit compared to the ideal one. One would expect that the delayed data packets and acknowledgments would result in slower replication of the forwarding quine over n_4. However, there is another aspect that has not been discussed so far:

Even if the round-trip time is longer over a delayed path, the rate at which the acknowledgments return is the same. After one initial round-trip time the forwarding quine then replicates with the same rate as its competitor. Even if the latter had a head start in gaining weight, the force to displace other molecules is the same for both quines. Furthermore the queueing capacity of a link rises with its delay; more packets are on the wire at the same time, and these packets are not subject to the dilution flow of any reaction vessel anymore. Remember that data packets in a reaction vessel are also subject to dilution, since they have to wait a short time until being picked up by a forwarding rule. This idle time is dictated by the Gillespie algorithm that simulates the collision of molecules in a well-stirred reaction vessel. If a data stream is able to "store" packets in the non-diluted environment of a delayed link, it has a small advantage. This explains why the reaction network does not react to delays as intuitively suggested.

The fact that the path over ideal link finally wins the competition is due to the fact that in our topology the data stream passes less nodes in average: The system is more sensitive to the number of hops than to the actual round-trip time. This behavior needs further investigation.

4.4 Self-Healing Aspects

We used an artificial chemistry for packet routing because of its ability to feature self-healing properties. We first constrained each reaction vessel by a dilution, which forced us to carefully design programs that constantly replicate in order to survive in this environment. As a result, our protocol software consists of several interdependent quines that obey two different robustness concepts:

(i) In the nucleus we foster a protected environment in which every node is able to control its net production rate; there is no uncontrolled inflow of molecules. The reference clocks, generated by the injection frequencies of the trigger molecules (ρ, σ, and ϕ), control the rate at which routing table entries are imported from neighbors and therefore the growth rate of the molecules in the nucleus; the riboquine expels its products to the main vessel. Thus the dynamic behavior of the nucleus is predictable, and since the whole active code in the nucleus is made of quines, the whole dissemination and expression process resists deletion attacks as shown in [13].

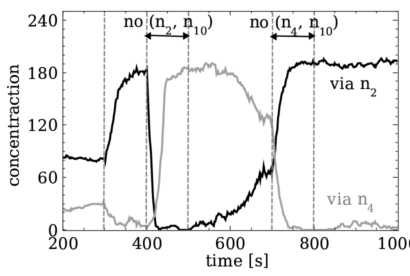
(ii) The quines in the forwarding engine are not able to reproduce themselves immediately, since the required reward is deferred by the acknowledge mechanism. The forwarding quine may therefore become extinct, for example if the data packet or acknowledge packet is lost. However, this extinction of code is desired, because it enables populations of successful forwarding quines to grow instead. Furthermore riboquines periodically repopulate the forwarding engine with fresh rules.

Figure 10 shows a simulation trace where molecules suffer destruction attacks. At $t = 400$ s, we randomly remove 50 % of all molecules from both reaction vessels in node n_{10}. Even though the reduction is clearly visible in Fig. 10(a) the effect on data rate, packet loss, and delay is hardly noticeable. In a long lasting attack it may happen that the data packets start to accumulate due to the reduced number of forwarding rules. As soon as the attack is over the accumulated data packets will temporarily increase the reaction rate of the forwarding quine due to the law of mass action, which will compensate for the reduced transmission rate during the attack.

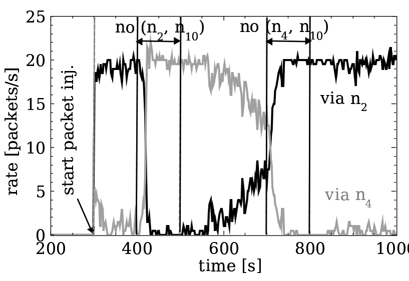
5. RELATED WORK AND ASSESSMENT

We now compare our protocol to two other biologically-inspired routing protocols (*AntNet* [5] and *MARAS* [11]), then critically assess our protocol in this context.

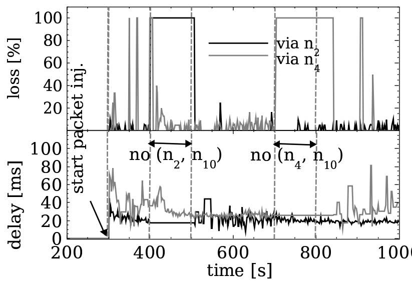
In **AntNet** [5], each node periodically sends forward ants to known destinations. The path of the forward ant is stochastically determined by the already existing routing table. The forward ant



(a) Forwarding rule concentration in n_{10}

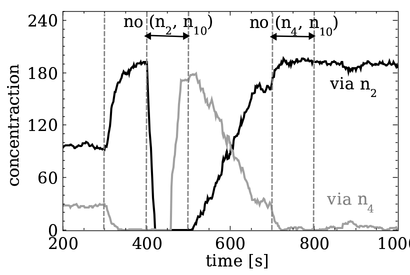


(b) Forwarding transmission rate at n_{10}

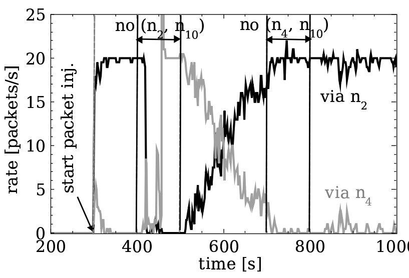


(c) End-to-end packet loss and delay

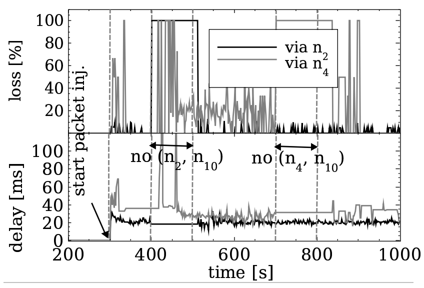
Figure 7: Stochastic simulation of data forwarding in Fraglets. Data packets are injected into node n_{10} for destination service s_a at rate $20\,\frac{\text{packets}}{\text{s}}$ (see topology in Fig. 2). At $t = 300\,\text{s}$ data injection starts, between $t = 400\,\text{s}$ and $t = 500\,\text{s}$ the link (n_2, n_{10}) is broken, and between $t = 700\,\text{s}$ and $t = 800\,\text{s}$ the link (n_4, n_{10}) is broken.



(a) Forwarding rule concentration in n_{10}

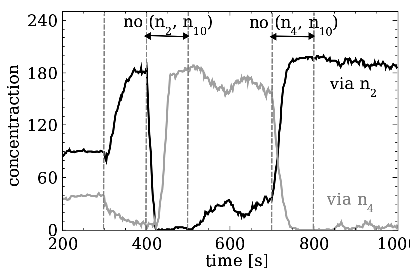


(b) Forwarding transmission rate at n_{10}

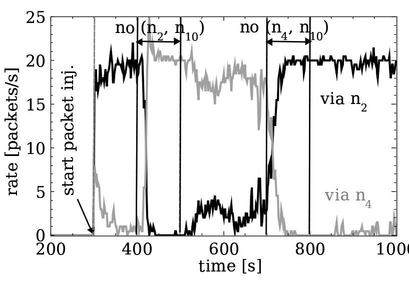


(c) End-to-end packet loss and delay

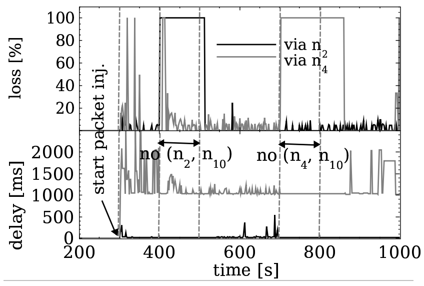
Figure 8: Same conditions as Fig. 7, but link (n_{10}, n_4) loses 20 % of the packets.



(a) Forwarding rule concentration in n_{10}

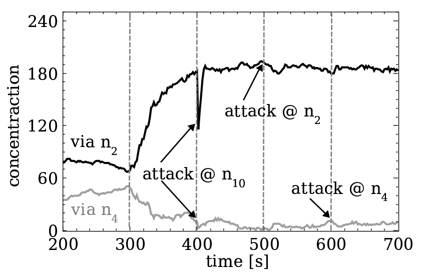


(b) Forwarding transmission rate at n_{10}

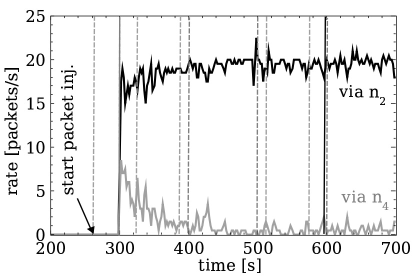


(c) End-to-end packet loss and delay

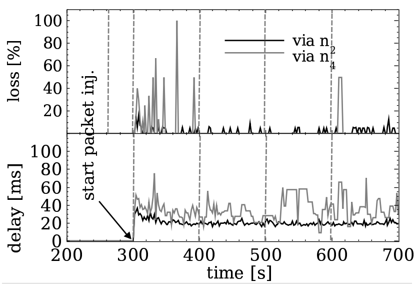
Figure 9: Same conditions as Fig. 7, but link (n_{10}, n_4) delays packet delivery by 1 s.



(a) Forwarding rule concentration in n_{10}



(b) Forwarding transmission rate at n_{10}



(c) End-to-end packet loss and delay

Figure 10: Same conditions as Fig. 7. At $t = 300\,\text{s}$ 50 % of all molecules in node n_{10} are forcefully destroyed; at $t = 400\,\text{s}$ a 50 % destruction attack is performed to neighbor n_2, and at $t = 400\,\text{s}$ to neighbor n_4.

records the number of hops and the latency during its journey. When reaching the destination it generates a backward ant that travels back to the source node and deposits pheromones on the nodes according to the quality of the path.

MARAS [11] assumes that the location of the destination service can be approximated using geographical information and therefore does not need to disseminate location information. Their forwarding engine also probabilistically selects one out of multiple possible next hops. When the packet reaches the destination, the quality of the path is evaluated and this information is sent back along the path to update the routing tables.

The concentration of chemical signals is used to indicate the quality of a path in both AntNet and our protocol. Unlike AntNet however, the dissemination part of our algorithm does not know the available services, since there is no feedback from the forwarding engine to the nucleus. The nucleus broadcasts unspecific fetch requests to its neighbors to learn about the existence of services. After expressing the entry, the forwarding engine already has a forwarding rule to deliver a packet to a destination to which it has never sent a packet before. However, this approach does not scale well when there are many services in the network. MARAS avoids this step by using geographical information to approximately find next hops towards the destination service.

The three algorithms use feedback information from the destination service to update their forwarding logic, which then stochastically selects a path. While the update frequency in AntNet is determined by the independent rate at which discovery ants are sent, the adaption speed of MARAS and our algorithm depends on the rate of the forwarded traffic. The main difference of our algorithm compared to the others is that it does not explicitly calculate the quality of a certain path. We never store a metric, delay or quality value symbolically in one of the molecules. Instead, the concentration of molecules together with the reaction algorithm leads to the emergent phenomenon of optimal path reinforcement. This mechanism is more robust when facing destruction attacks: a deleted entry does not distort the probability of choosing a certain path.

The presented protocol is not superior to comparable multipath routing protocols but has code healing abilities. One observed problem is that links with shorter delays are not preferred. In addition to this, there is a small portion of data packets that might be lost by dilution. Then, data packets have to wait a short time until they are randomly chosen by a forwarding rule, and finally, data packets may be reordered within a reaction vessel. However, the presented protocol correctly switches to alternative paths to avoid dropped links or nodes and tries to optimize for high throughput and therefore low packet loss.

6. CONCLUSIONS

Our results show that continuous code rewriting is a valid approach for self-healing applications that have to work over a network with variable topology and quality. A notable property of our multipath routing protocol is that there is no central control, supervision or health monitor: Instead, code homeostasis is used to solve – in one go – the tasks of self-healing, topology tracking, load balancing, link quality assessment and forwarding.

The dilution flow, that randomly eliminates excess molecules, is at the same time a panacea and curse. The difficult part is to engineer regulation loops that are stable despite the continuous pressure on code fraglets. On the other hand, dilution flow is the basis for running a competition among execution alternatives and for implementing a form of continuous garbage collection.

We have started to expand our research towards other "autocatalytic patterns" in order to enrich our design methodology beyond compact quines. On the other hand, designing distributed "chemical" applications by hand is challenging because of the difficulty in anticipating the emergent effects of a local change in a chemical rule. Instead, we would need a system that generates and tries alternative execution branches in an automatic way, creating competition and cooperation on a next level of "service super organisms". Based on the work presented here we believe that the encoding of state as rates (instead of numeric symbols), the pressure of a dilution flow and the principle of continuous and regulated code rewriting will play an important role for *code evolution in the net*.

7. ACKNOWLEDGMENTS

This work has been supported by the Swiss National Science Foundation, through SNF Project Self-Healing Protocols.

8. REFERENCES

[1] Scilab home page. http://www.scilab.org.

[2] H. Abrash. Studies concerning affinity. *J. Chem. Ed.*, 63:1044–1047, 1986. English translation of P. Waage and C. M. Guldberg's paper of 1864.

[3] B. Anckaert, M. Madou, and K. de Bosschere. A model for self-modifying code. In *Proc. 8th Information Hiding Conf.*, pages 232–248. Springer 4437, June 2007.

[4] L. N. Chakrapani, P. Korkmaz, B. E. S. Akgul, and K. V. Palem. Probabilistic system-on-a-chip architectures. *ACM Trans. Design Autom. Electr. Syst.*, 12(3), 2007.

[5] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *J. Artificial Intelligence Research (JAIR)*, 9:317–365, 1998.

[6] P. Dittrich, J. Ziegler, and W. Banzhaf. Artificial chemistries - a review. *Artificial Life*, 7(3):225–275, 2001.

[7] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[8] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[9] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling data centric routing in wireless sensor networks. Technical report ceng 02-14, USC Computer Engineering, 2002.

[10] D. L. J. Lafontaine and D. Tollervey. The function and synthesis of ribosomes. *Nature Reviews. Molecular Cell Biology*, 2(7):514–520, 2001.

[11] K. Leibnitz, N. Wakamiya, and M. Murata. Self-adaptive ad-hoc/sensor network routing with attractor-selection. *IEEE GLOBECOM '06*, pages 1–5, Nov. 2006.

[12] W. Martin and M. J. Russell. On the origins of cells. *Philos. Trans. R. Soc. Lond. B. Biol. Sci.*, 358(1429), 2003.

[13] T. Meyer, L. Yamamoto, and C. Tschudin. Robustness to code and data deletion in autocatalytic quines. *Trans. Comp. Sys. Bio.*, (to appear), 2008.

[14] E. Post. Formal reductions of the combinatorial decision problem. *Am. J. Math.*, 65:197–215, 1943.

[15] M. Shaw. Self-healing: softening precision to avoid brittleness. In *Workshop on self-healing systems (WOSS)*, pages 111–114, 2002.

[16] P. F. Stadler, W. Fontana, and J. H. Miller. Random catalytic reaction networks. *Phys. D*, 63(3-4):378–392, 1993.

[17] G. P. Thompson. The quine page. http://www.nyx.net/$\sim$gthompso/quine.htm.

[18] C. Tschudin. Fraglets - a metabolistic execution model for communication protocols. In *Proc. 2nd Symp. Autonomous Intelligent Networks and Systems*, Menlo Park, USA, 2003.