A Self-healing Multipath Routing Protocol

Thomas Meyer, Lidia Yamamoto, Christian Tschudin
Computer Science Department, University of Basel, Switzerland

BIONETICS 2008, Hyogo, Japan
November 26th 2008
Motivation

• Today's software depends on reliable hardware.
• But future hardware will be unreliable:
  • Smart Dust: nano computers, large quantity, low price [Bahar 2001]
  • Probabilistic Chips (PCMOS): low energy consumption [Palem 2007]
• Consequences:
  • Execution of wrong instructions
  • Corruption of code in memory

Our Goal:

Reliable execution in unreliable environments

Provide methods for designing resilient software
Contents

- Motivation
- Method
  - Fraglets: an artificial chemistry
  - Self-healing code using autocatalytic Quines
- Multipath routing prototocool design
- Results
- Conclusions
Fraglets: An Artificial Chemistry

- Fraglets is a constructive artificial chemistry based on string rewriting rules.

- Fraglets runs in a virtual machine (interpreter).

[Dittrich et al 2001] [Tschudin 2003]
Self-replicating Code: *Quines*

- Reactions consume their educt code and data molecules.
- Code must continuously be regenerated. Or better, code has to regenerate itself!
- A self-replicating *Quine* duplicates its active part and blueprint when processing a data molecule. [Yamamoto et al 2007]

![Diagram of self-replicating code]

- Limited vessel capacity ⇒ selective pressure ⇒ survival of replication (defective, non-replicating code will be displaced)
Self-healing Code: Autocatalytic Quines

- In steady-state, blueprints and active rules are present with equal concentration.
- The code is resilient to knock-out attacks.
- The code is resilient to most mutation attacks.

- This results in emergent code homeostasis
Self-healing Routing Protocol

- How to build more complex self-healing programs based on *Quines*?
- In our case: Route data packets to a labeled service (e.g. “A”) within a network of Fraglet nodes:
Routing Protocol Overview: The Cell Metaphor

- **Information:** Routing table entries (RTEs) are exchanged between the nuclei of neighbor nodes.
- **Expression:** Passive routing table entries (RTEs) are activated.
- **Operation:** Active rules forward data packets to the next hop.
Routing Table Entry (RTE) Dissemination

- Routing Table = Multiset of molecules: \([\text{RTE svc hop1 \ldots hopn}]\)
- Every service injects a local RTE: \([\text{RTE svc}]\)
- Every node periodically injects trigger molecules.
- A Quine periodically obtains random RTEs from neighbors.
- Result: Slow diffusion of reachability information across the network.

[Diagram showing the dissemination process with nodes and molecules.]
Expression of Routing Table Entries (RTEs)

- The "riboquine" periodically expresses an arbitrary RTE.
- The concentration of a forwarding rule is proportional to the concentration of its corresponding RTE.
- Forwarding rules for the same destination service compete for incoming data packets.
- Forwarding rules are consumed when forwarding a data packet!
Forwarding Path Reinforcement

- Forwarding rules must replicate themselves.
- We only let those forwarding rules replicate that successfully delivered a data packet to the destination service.
- The replication latency is equal to the node-to-end round trip time of the data packet.
- The growth rate of a forwarding Quine is equal to the rate of successfully delivered data packets.
- Multiple paths may coexist.
Summary of Operation

- The exchange of RTEs is needed to announce new services and their reachability (slow process, protected in the nucleus)
- By expressing the RTEs the „riboquine“ injects new or displaced rules (novelty) into the main vessel
- In the harsh environment of the main vessel, forwarding rules compete for data packets; successful rules increase their fitness
Results: Link Dropout Behavior

- The protocol shows quick reaction to packet loss, but only weakly favors shorter paths.

Forwarding Rule Concentrations in node 2

Start of the data stream (0.4Mbps)

Link 2-5 dropout

Link 2-4 dropout

End-to-end Packet Loss and Delay

Packet Loss, Delay [s]
Results: Resilience to Code and Data Destruction

- The software is resilient to destruction attacks (e.g. Random destruction of 80% of the molecules)
Conclusions

- We are able to build software for unreliable environments.
- Compared to existing routing protocols, our approach makes consequent use of the chemical paradigm on the code level.
  - We gain resilience to instruction loss and to most mutations.
  - The protocol is not forced to symbolically calculate statistics (e.g. link load, throughput, etc.). Molecule concentrations represent protocol states; packet rates are used to exchange information.
  - The resulting traffic regulation is an emergent property of the “chemical” reaction network that spreads over the communication network.
- However, the proposed protocol only selects for low packet loss paths; paths with long delays are not punished. The RTE dissemination process does not scale well for large networks.
- For the future we need additional methods to design “chemical” protocols that anticipate the emergent effects of local changes to the global reaction network.
Gillespie Algorithm

- **Goal:** Simulate the behavior of chemical reactions (ODE)
- **Algorithm:** *Exact Stochastic Simulation of Chemical Reactions* [Gillespie 1977]

\[ \frac{d \tilde{x}_5}{dt} = x_5 m_x \approx \frac{P(R_1)}{\Delta t} \approx \frac{x_5 m_x}{w_0} \]

- \( X_5 + M_x \to 2X_0 + M_x \)
- \( X_2 + M_x \to 2X_2 + M_x \)

reaction system

Simulation (Gillespie)
Autocatalytic Quine – Reaction Network

\[
\begin{align*}
& \text{bp match } x \text{ split match bp fork fork fork nop bp } \ast \text{ sum y 1} \\
& \text{match bp fork fork fork nop bp} \\
& \text{fork fork fork nop bp} \\
& \text{match } x \text{ split match bp fork fork fork nop bp } \ast \text{ sum y 1} \\
& \text{fork nop bp} \\
& \text{match } x \text{ split match bp fork fork fork nop bp } \ast \text{ sum y 1} \\
& \text{match } x \text{ split match bp fork fork fork nop bp } \ast \text{ sum y 1} \\
& \text{split match bp fork fork fork nop bp } \ast \text{ sum y 1 5} \\
& \text{sum y 1 5} \\
& [y 6]
\end{align*}
\]
Selective Pressure

a) (exponential) growth: duplication during replication

b) non-selective death: randomly delete molecules to maintain a const. population

- Only autocatalytic sets survive.
- Defective, non-replicating code will be displaced.
Cooperative Quines

- Independent Quines compete against each other \(\Rightarrow\) only one will survive.
- We build self-healing programs on cooperative Quines by using composition and compartmentation.

**Composition**

- Composition links Quines by a common data stream.
- Compartmentation simulates spatial separation.
Data Yield in Serial Autocatalytic Quines

\[ F(\rho_x) \rightarrow \rho_x \rightarrow x \rightarrow \varphi_x \rightarrow \rho_{t1} \rightarrow t1 \rightarrow \varphi_{t1} \rightarrow \rho_{t2} \rightarrow t2 \rightarrow \varphi_{t2} \rightarrow \rho_{tn} \rightarrow tn \rightarrow \rho_y \rightarrow y \]

\[ F(\rho_{t1}) \]

\[ F(\rho_{t2}) \]

\[ F(\rho_{tn}) \]

\[ F(\rho_{tn}) \]
Robustness to Destruction Attacks

\[ N = \frac{100}{2 \times \#\text{quines}} \]

- robustness, yield, req. CPU power
- req. CPU power
- vessel capacity \( N \)
- data injection rate \( \rho_x \)
- destruction rate \( \delta \)

\[ \frac{\delta}{\rho(2 \cdot \#\text{quines})} \]
Robustness to Mutation Attacks

- Robustness to a single mutation event (edit distance = 1)

![Bar chart showing robustness results]

- **Infinite closure**
  
  \[
  \text{[matchp } \ x \ x\text{] } + \ [x] \rightarrow \ [\text{matchp } \ x \ x\text{] } + \ [x \ x]\]

- **Inert**
  - no reaction for injected input data fraglet

- **No results**

- **Invalid results (}>95\%\)**

- **Invalid results (}>50\%\)**

- **Valid results (}>50\%\)**

- **Valid results (}>95\%\)**

Persistent Rules:

- [matchp x sum y 1]
  - 85%

- 100-fold redundancy
  - 93%

- Quines 100-fold redundancy
  - 85%

...
“Chemical” Communication Protocols

- The Fraglet instruction set allows for sending molecules (unicast \texttt{send node2 ...data...} and broadcast \texttt{send all ...data...}).

\[
d[X_1]/dt = d[x_1]/dt = -[X_1][x_1] + [X_2][x_2] \quad \text{packet rate} = [X_1][x_1]
\]

\[
d[X_2]/dt = d[x_2]/dt = [X_1][x_1] - [X_2][x_2] \quad \text{packet rate} = [X_2][x_2]
\]

\[
\text{Steady-state:} \quad [X_1][x_1] = [X_2][x_2]
\]

- Protocol states are encoded as molecule concentrations.
- Information is exchanged by means of packet rates. The packet rate is proportional to the reaction rate according to the Gillespie algorithm. [Gillespie 1977]
Routing Table Entry (RTE) Dissemination

- The RTE concentration slowly adjusts to network topology changes.
- The stochastic process (random educt selection + Gillespie) is accurately approximated by a deterministic model described by ODEs.
Forwarding Path Reinforcement

- Competition among forwarding rules yields a higher concentration of successful rules ⇒ higher reaction probability
- Coexistence of different forwarding paths is possible: multipath routing
Multipath

- The protocol exploits multiple paths
- For high packet rates packets are dropped by the dilution flow!

![Network Topology Diagram]

![Graphs showing transmission rate, received rate, link bandwidth, path bandwidth, and packet loss vs. packet rate]

Dilution flow starts to attack data packets.
Packet Loss Caused by the Dilution Flow

- In vessels with low capacity the incoming data packet stream may displace forwarding rules faster than they can be regenerated

The vessel capacity (external parameter) correlates with the node's CPU speed (more molecules $\Rightarrow$ higher reaction rates)

- The resulting packet loss allows neighbor nodes to deviate traffic around heavily loaded nodes