Autonomic Computer Systems (CS321)

Chemical Networking II:
Protocol Examples and Self-Healing Protocols

Thomas Meyer
Computer Science Department, University of Basel, Switzerland

November 29th, 2011

Contents (two lectures)

Stochastic protocols (last lecture)
from Ethernet to reactive systems: an overview

Chemical networking (last lecture)
a formal approach to describe stochastic reactive systems

The three levels of structuring (distributed) computations

- symbolic (last lecture)
  we know it all: 1 + 2 => 3; HTTP GET + File => 200 OK
- dynamic (last lecture)
  more challenging: global behavior emerging from the *dynamic* behavior of network nodes
- organizational
  use the long-term trend of dynamic behavior to carry out computation

From robust to self-healing protocols
combining all the three levels to achieve robust protocol behavior and protocol code that heals itself

Contents (two lectures)

Stochastic protocols
from Ethernet to reactive systems: an overview

Chemical networking
a formal approach to describe stochastic reactive systems

The three levels of structuring (distributed) computations

- symbolic
  we know it all: 1 + 2 => 3; HTTP GET + File => 200 OK
- dynamic
  more challenging: global behavior emerging from the *dynamic* behavior of network nodes
- organizational
  use the long-term trend of dynamic behavior to carry out computation

From robust to self-healing protocols
combining all the three levels to achieve robust protocol behavior and protocol code that heals itself

The Three Levels of Structuring Chemical Computation

Applicable to local and distributed computation

Symbolic Chemical Computation
The result of the computation is found in (the structure) of a single molecule instance

- Examples

Dynamic Chemical Computation
The result of the computation is found in the number of instances of a species

- Dynamic behavior of (artificial) chemical reactions - the Law of Mass Action
- Equilibrium properties
- Simulating the dynamic behavior on a computer - the Gillespie algorithm
- Chemical Networking Protocols

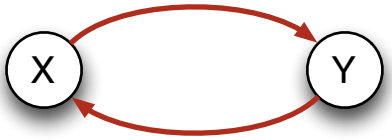
Organizational Chemical Computation
The result of the computation is found in the presence/absence of instances of molecular species

- Examples

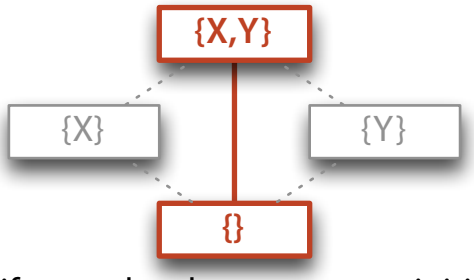
Organizational Chemical Computation

Result = Presence/Absence of Chemical Species

- Information: not represented symbolically, not represented by concentrations (numeric), but by the **presence/absence of species (binary)**
- **Organization = set of species that a system is able to maintain for a long time**
  *same concept as equilibrium, but applied to **sets***
- Example: Intuitively, what are the organizations of the following reaction system?



if molecules are present initially

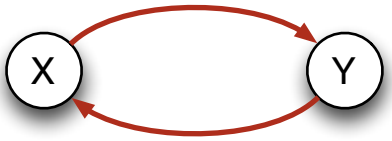


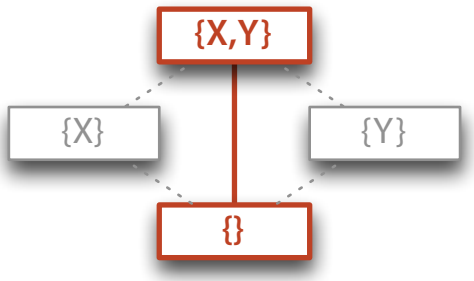
if no molecules are present initially

Organizational Chemical Computation

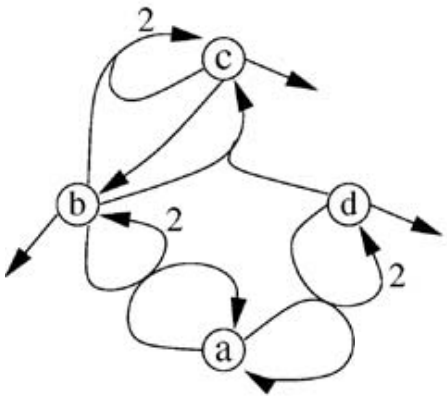
Result = Presence/Absence of Chemical Species

- Example: Intuitively, what are the organizations of the following reaction system?





- What are the organizations of the following more complex reaction system?
- Wish: A theory to determine organizations in an arbitrary reaction network



Chemical Organization Theory — Intuitively

Dittrich et al, *Chemical Organization Theory*, Bulletin of Mathematical Biology, (69) 4, pp. 1199-1231, 2007

- **Definition:** An **organization** is a set of molecular species that is **closed** and **self-maintaining**
- Intuitive approach: Examine the fate of molecular species:
  - Is the following set of polymer molecules **closed**?

$$x \longrightarrow xx \longrightarrow xxx \longrightarrow xxxx \longrightarrow \ldots$$

    - **No:** Each species "generates" a new species.
  - Is the following set of molecules **self-maintaining**?

$$x \longleftrightarrow y \rightarrow \varnothing$$

    - **No:** Eventually, both species will disappear
- In an organization, the *set of species* neither grows or shrinks

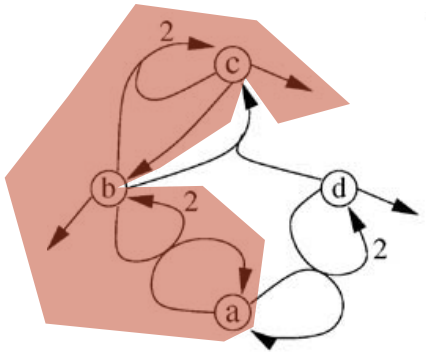
Chemical Organization Theory — Formally

Dittrich et al, *Chemical Organization Theory*, Bulletin of Mathematical Biology, (69) 4, pp. 1199-1231, 2007

- **Definition:** An **organization** is a set of molecular species that is **closed** and **self-maintaining**
- **Closed:** The set only contains species that can be produced by reactions among those species (The set cannot "generate" new species.)
- **Self-Maintaining:** The set of species is able to maintain its mass under influence of reactions among those species

Closed Sets:
0,a,b,d,ab,bc,ad,**abc**,bcd,abcd

Semi Self-Maintaining Sets:
0,a,ab,bc,ad,**abc**,abd,abcd

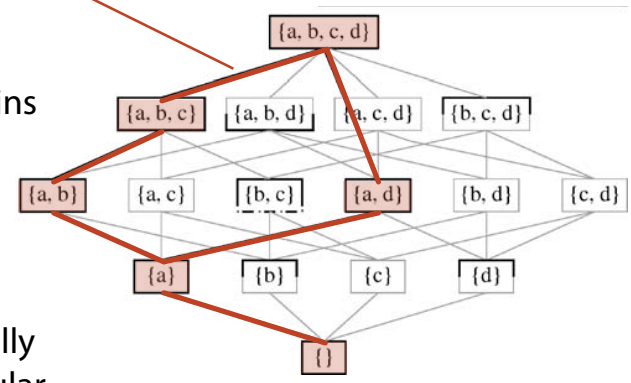


Semi Organizations: (intersection)
0,a,ab,bc,ad,**abc**,abcd

Chemical Organization Theory

Relation between Organizations and Dynamic Fixed Points

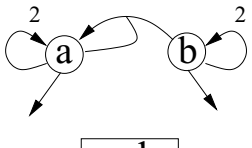
- Chemical Organization Theory first determines the stability of the molecular **sets** before analyzing the fixed point in the concentration vector space.
- Properties of chemical organizations:
  - The set of organizations form a **lattice**.
    - Each organization is part of a super-organization and contains sub-organizations.
  - **Fixed points** are instances of organizations.
    - A reaction system is dynamically stable only if the set of molecular species is closed and self-maintaining.

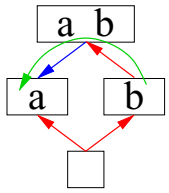


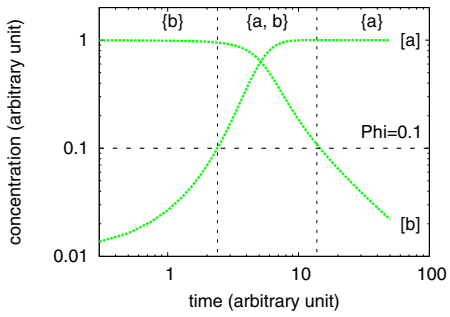
Chemical Organization Theory

The system may move between organizations

- Molecular species may come and go
- This is reflected by the system's movement in the lattice of organizations
- **Downward movement:** Species disappear (e.g. due to unstable organizations, random fluctuations, random dilution flux)
- **Upward movement:** Only by explicit injection of new species (e.g. caused by mutation).







Computing With Organizations

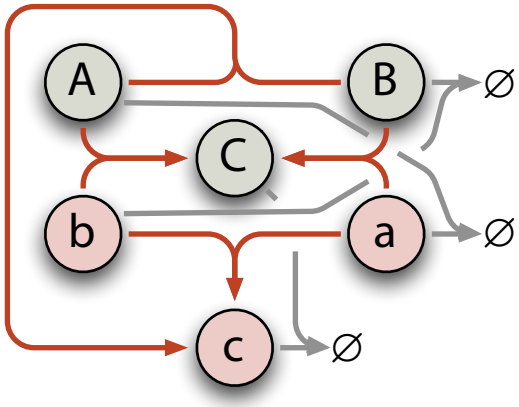
Example: The chemical XOR gate

The chemical XOR gate is a chemical reaction network among six species:

Dittrich et al, Chemical Organization Theory as a Theoretical Base for Chemical Computing, Workshop on Unconventional Computing, 71-83, 2005

- Species: $\mathcal{S} = \{a, b, c, A, B, C\}$
  - Inputs and outputs represented by presence of species:
    - a: Input 1 = 0 A: Input 1 = 1
    - b: Input 2 = 0 B: Input 2 = 1
    - c: Output = 0 C: Output = 1
- Algorithm: well-stirred vessel, law of mass action (e.g. Gillespie's algorithm)

- Reactions:

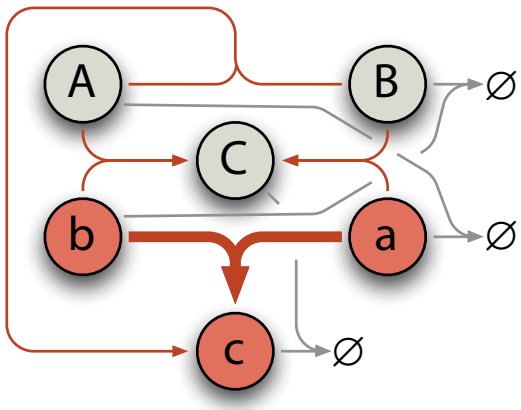


Computing With Organizations

Example: The chemical XOR gate

Truth Table:

Input 1 (A/a)	Input 2 (B/b)	Output (C/c)
0	0	0
0	1	1
1	0	1
1	1	0

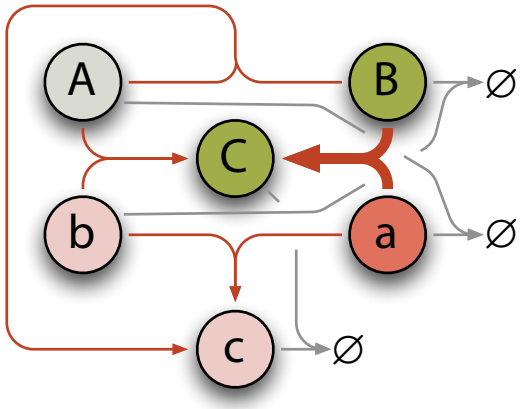


Computing With Organizations

Example: The chemical XOR gate

Truth Table:

Input 1 (A/a)	Input 2 (B/b)	Output (C/c)
0	0	0
0	1	1
1	0	1
1	1	0

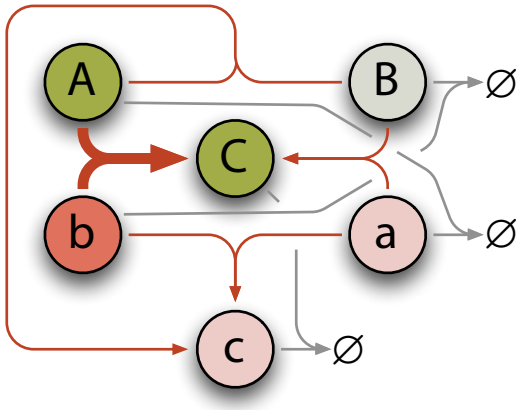


Computing With Organizations

Example: The chemical XOR gate

Truth Table:

Input 1 (A/a)	Input 2 (B/b)	Output (C/c)
0	0	0
0	1	1
1	0	1
1	1	0

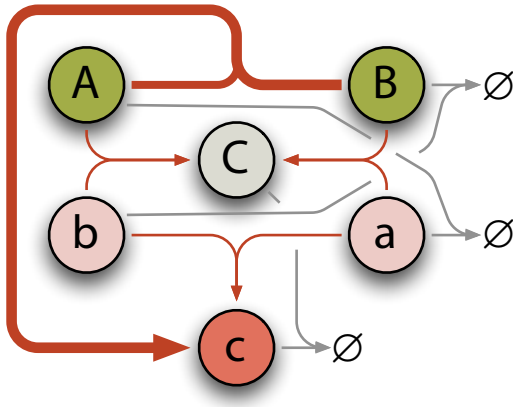


Computing With Organizations

Example: The chemical XOR gate

Truth Table:

Input 1 (A/a)	Input 2 (B/b)	Output (C/c)
0	0	0
0	1	1
1	0	1
1	1	0



Artificial Chemical Computing

Summary

- Artificial chemical computing is an unconventional way of organizing computation.
- Artificial Chemistries compute in an inherently parallel way
- There are three fundamentally different levels at which AChems **represent and process information**:

1. Symbolically within the structure of the molecules
   efficient, halts when result present
2. As the concentrations of molecular species (multiplicity of molecules)
   robust, perpetual reaction execution, result at equilibrium
3. With the presence/absence of molecules of different , species
   even more robust, perpetual reaction execution, result at equilibrium

Chemical Networking Protocols

Summary

- Artificial chemistries provide a model for **parallel computation**.
- Computation may be carried out **symbolically**...
  result within the molecule
- ...or **dynamically**
  result is reflected by the concentration (or even by the presence/absence of molecules)
- The **law of mass action** maps concentrations to reaction rates.
  - This allows chemical protocols to convey state information (concentrations) from one node to another.
  - The execution flow is random, the timing of reactions is stochastic.
  - Nevertheless, deterministic ODEs approximate the macroscopic dynamic behavior chemical programs.

Chemical Networking Protocols

Summary

- The dynamic behavior of traditional networking protocols and distributed algorithms are hard to analyze.
  - The **analysis** of chemical networking protocols is **often easy**:
    Chemical analysis tools (perturbation analysis, metabolic control analysis) can directly be applied to artificial chemical protocols.
- Chemical networking protocols can be **synthesized from simple reaction networks** (chemical design patterns).
- Chemical networking requires a **different mind set**:
  - Design/study dynamics first, function afterwards.
  - Try to state distributed algorithms as equilibrium processes.
  - Benefit: Equilibria/steady-state is stable to perturbations.

Contents (two lectures)

Stochastic protocols
from Ethernet to reactive systems: an overview

Chemical networking
a formal approach to describe stochastic reactive systems

The three levels of structuring (distributed) computations

- symbolic
  we know it all: 1 + 2 => 3; HTTP GET + File => 200 OK
- dynamic
  more challenging: global behavior emerging from the *dynamic* behavior of network nodes
- organizational
  use the long-term trend of dynamic behavior to carry out computation

From robust to self-healing protocols
combining all the three levels to achieve robust protocol behavior and protocol code that heals itself

From Robust to Self-Healing Protocols

- A Chemical Networking Protocol Example:
  C3A - A Chemical Congestion Control Algorithm
- Self-healing protocols:
  How protocol software is able to heal itself from the loss of parts of its own code
  - A method to make chemical software self-healing
  - Example: A self-healing link-load-balancing protocol

From Robust to Self-Healing Protocols

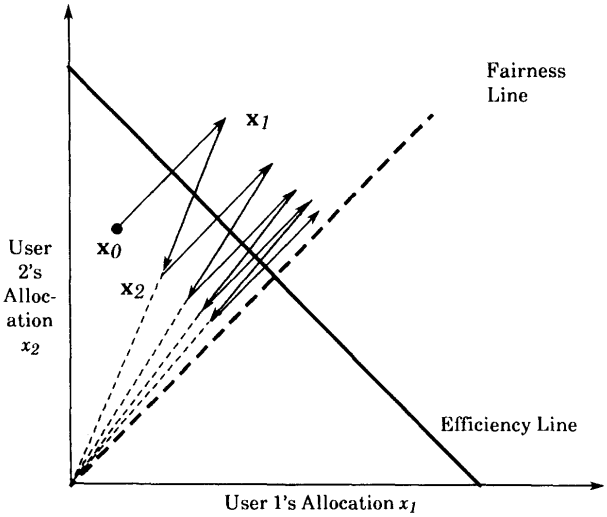
- A Chemical Networking Protocol Example:
  C3A - A Chemical Congestion Control Algorithm
- Self-healing protocols:
  How protocol software is able to heal itself from the loss of parts of its own code
  - A method to make chemical software self-healing
  - Example: A self-healing link-load-balancing protocol

Congestion Control in the Internet

- The bandwidth of network links is a limited resource:
  Computers send data faster than the capacity of links.
- In the Internet: **End-to-end** congestion ctrl: senders have to throttle their transmission rate to grant a fair allocation of the bandwidth
- This is a typical **control task**:
  - Input: (Limited) Congestion information from the network
    - Packet loss ⇒ sender detects missing ACK
    - Queues are filled ⇒ delay increases ⇒ sender detects increased RTT
  - Output: Transmission rate
  - Goal: **efficiency and fairness**
    - All link capacities shall be exploited
    - Every stream shall obtain the same bandwidth share

Congestion Control in TCP Reno

- Transmission rate is indirectly controlled by the congestion window (CWND)
  CWND: *Number unacknowledged packets "on-the-wire"*
- **Additive increase / multiplicative decrease** method:
  - CWND is increased by one for each acknowledged packet
  - CWND is divided by two for each packet loss detected
- Distributed optimization algorithm
- Can we design a similar congestion control algorithm in a chemical way?

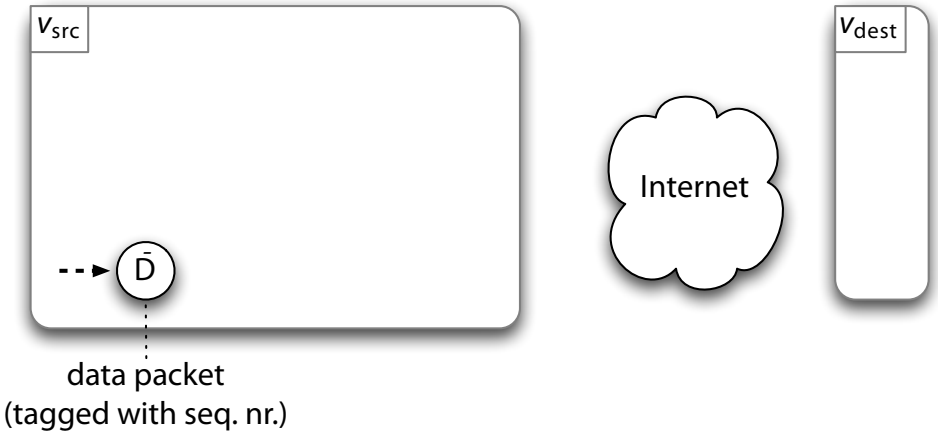


Chiu et al, Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, Computer Networks and ISDN Systems, 17(1): 1-14, 1989

Design of a Chemical Congestion Control Algorithm

Step 1/6: Treatment of data packets

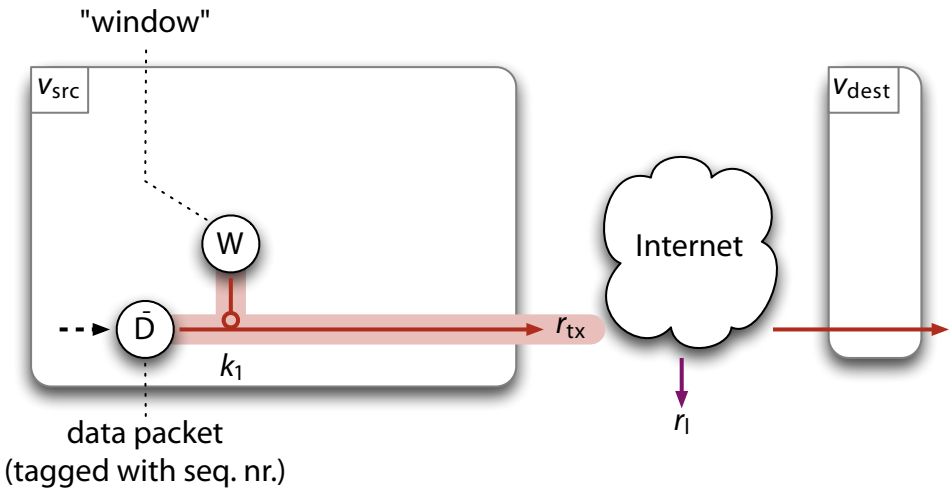
- Data packets shall be sent from source to destination over the (congested) Internet.
- Data packets are injected into the source node (species D).
- Data packets are tagged with a sequence number.



Design of a Chemical Congestion Control Algorithm

Step 2/6: Tunable transmission rate

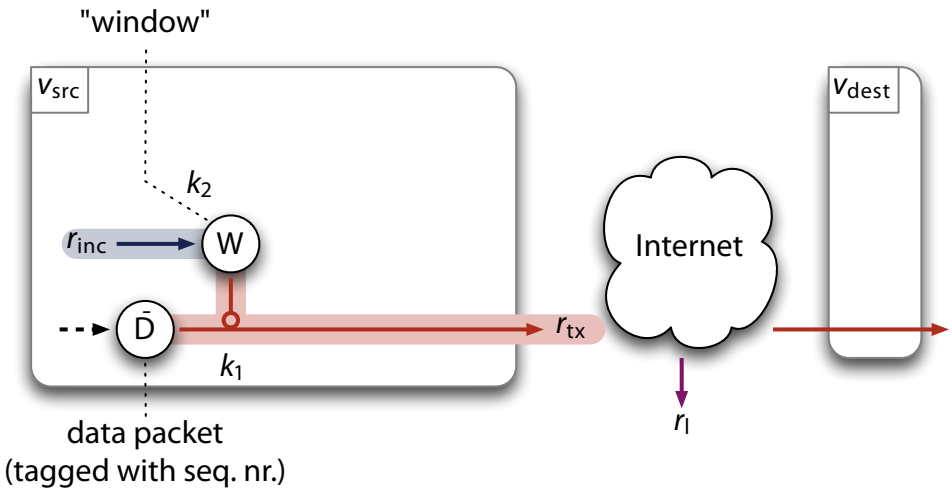
- The number of "window" molecules determines the transmission rate: law of mass action: $r_{tx} = k_1 wd$
- Some packets are lost: r_l



Design of a Chemical Congestion Control Algorithm

Step 3/6: Linear increase of the transmission rate

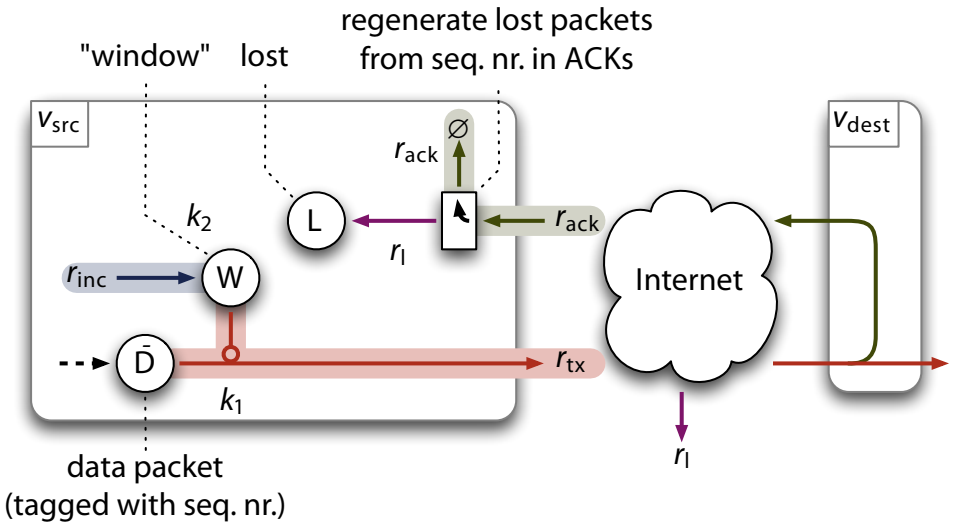
- Window molecules W are injected at a constant rate
- The concentration of W increases linearly
- The **transmission rate increases linearly**



Design of a Chemical Congestion Control Algorithm

Step 4/6: Measure lost packets

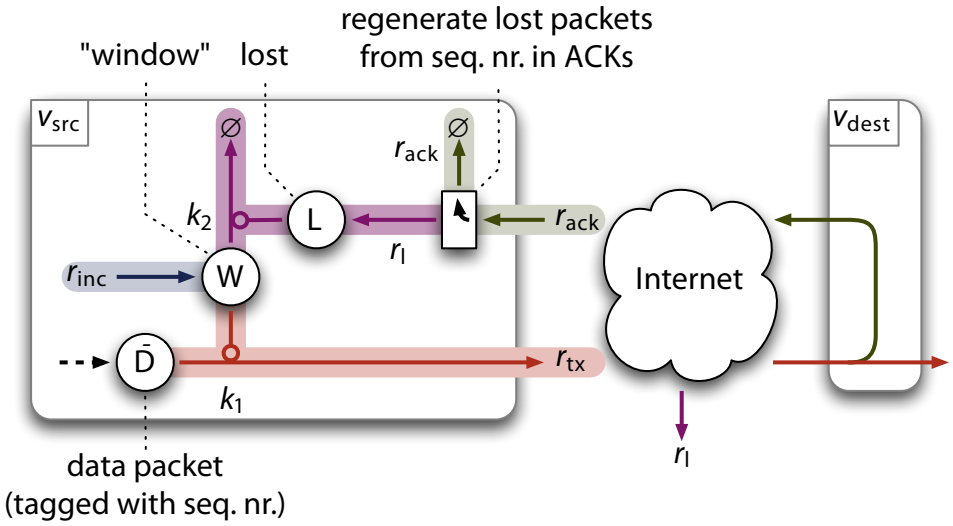
- The source node regenerates the lost packets whenever it detects a jump in the sequence number of received ACKs
- Species L contains the number of packets lost



Design of a Chemical Congestion Control Algorithm

Step 5/6: Closing the control loop

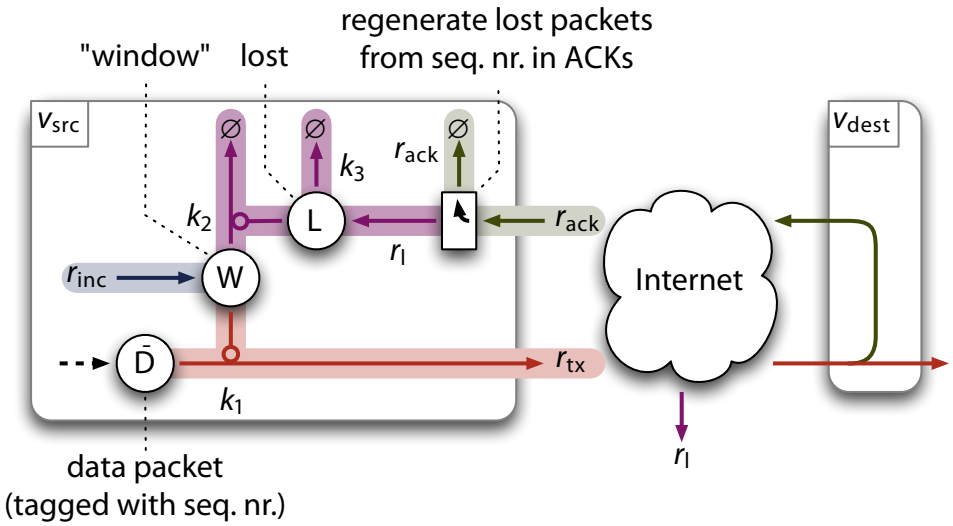
- Each (regenerated) lost packet catalyzes the destruction of window molecules
- Law of mass action: the more lost packets, the faster the decrease of W



Design of a Chemical Congestion Control Algorithm

Step 6/6: Adjusting the control loop

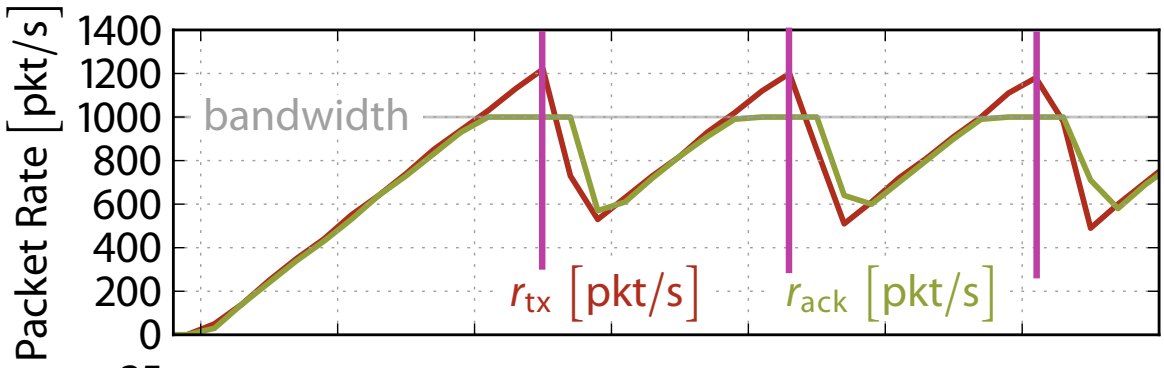
- The influence of a lost packet to the decrease of W has to be stopped eventually:
- L molecules are decayed by a first-order reaction.



Chemical Congestion Control Algorithm

Results

Single packet flow over a bandwidth limited link



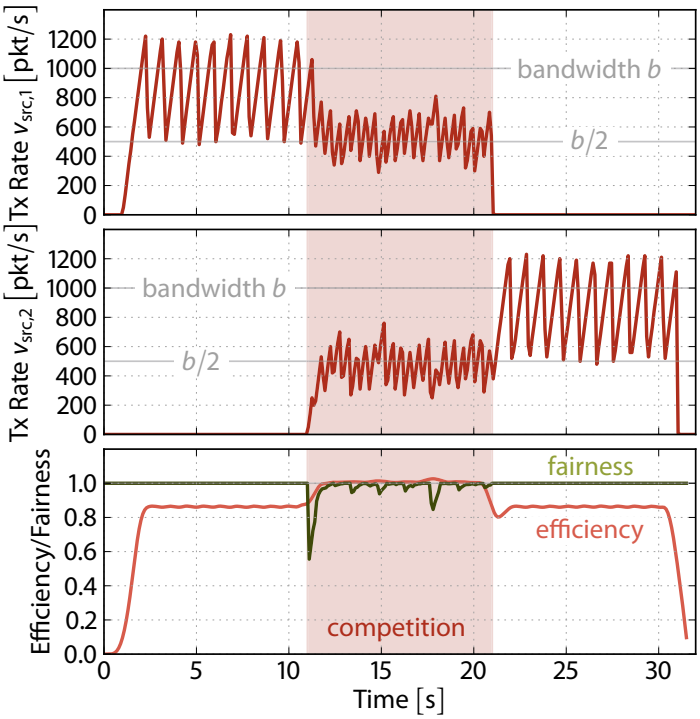
- Transmission rate continuously increases
- If a **packet loss** is detected at the sender, the transmission rate is divided by two

Chemical Congestion Control Algorithm

Results

Two competing packet flows over the same bandwidth limited link:

- **Efficiency:**
  Saw-tooth pattern does not allow a single stream to fully exploit the bandwidth *(efficiency < 1)*
- **Fairness:**
  The algorithm is fair:
  competition → both senders decrease their transmission rate to the same level.



Chemical Congestion Control Algorithm

Assessment

- The C3A is a **hybrid** protocol:
  - **User-information** is encoded **symbolically** *inside* the (data) packets
  - **Control** is carried out **implicitly** by coupling packet rates with a chemical reaction network.
  - The two information processing levels are orthogonal:
    can be used at the same time without bothering each other
- Advantage of controlling the dynamics with a chemical reaction network:
  - Intuitive design with the help of the reaction network graph
  - Sound math. analysis methods (Chemical Organization Theory, ODEs)
  - Easy to extend (by adding additional reactions)

From Robust to Self-Healing Protocols

- A Chemical Networking Protocol Example:
  C3A - A Chemical Congestion Control Algorithm
- Self-healing protocols:
  How protocol software is able to heal itself from the loss of parts of its own code
  - A method to make chemical software self-healing
  - Example: A self-healing link-load-balancing protocol

Self-Healing

Three levels of self-healing:

1. **Service configuration level (IBM's approach):**
   - "The system automatically detects, diagnoses, and repairs localized software and hardware"
   - Mostly centered around configuration errors
     (e.g. regression tests for SW updates)
   - Mostly based on log analysis
2. **Self-Healing inside the applications (beyond IBM):**
   - what algorithms to use?, what data structures?, what encoding?, etc.
   - requires trade-offs and (internal) choice: this alg. is more robust, but slower
   - this is still parametrization

Self-Healing

3. Deep self-healing:

- "The system asserts goal integrity over long times"
- Note: Code can change, changing the "performance envelope"!
- **Self-healing at the code level**:
  - Application controls its own code, not only parameters
  - Can walk algorithm space, not just config space
- First step:
  - Assert code integrity over long times
  - Evolution is not possible without robustness
    Wagner: *Robustness and Evolvability in Living Systems*, Princeton University Press, 2007

Artificial Chemistries for Self-Healing

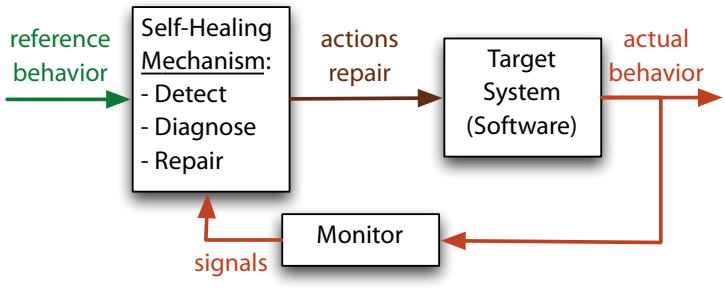
- Current systems are so far not able to deal with code problems (no autonomic debugging system): undecidability, halting problem, etc.
- Why artificial chemistry, why Fraglets?
  - Choice of next rule to be executed depends on the "concentration" of rules, not on the exact sequence of rules, e.g. as they appear in the source file
  - **Redundancy by multiplicity** is a natural programming model in an artificial chemistry, or chemical computing model
  - Can we use these properties to fully inspect and recover from failures by **regenerating** lost or corrupted code?
    - lost code = lost molecule → inject more of it into the chemical "soup"

Resilience to Instruction Loss

- Dealing with data loss or corruption is common in networking
  - retransmit lost packet
  - error detection/correction codes
- But how about code?
- How can code get lost or corrupted?
  - explicit instruction deletion or corruption, e.g. by an attacker
  - unfaithful execution, bugs that affect the execution flow

Self-Healing Software: With List

- Software should:
  - **Detect** improper execution (detection, diagnosis)
  - Continue operation in spite (a limited amount of) execution errors (**resilience**, graceful degradation)
  - Regain full operation and resilience (repair, **healing**)

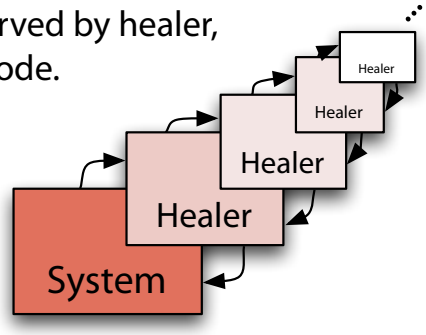


- First step: resilience to instruction loss
  - Deal with lost instructions (not with corrupted ones)
  - **Goal:** **resilience against the removal of code parts** in a given program/protocol

Traditional Approach

The 'Healer' is a component separate from the system

- Generally, three ingredients are necessary:
  1. We need to **detect** that code is missing.
  2. We need **redundant information** to restore missing code.
  3. We need **dynamic control** of this repair mechanism.
- Traditional approach: Dedicated healer component
  - Software has to be observed by healer, which detects missing code.
  - Healer is software, too. We also need an observer of the healer,...
  - Leads to infinite regression.



Approach Inspired By Population Dynamics

Balance of Growth and Death

- Break the infinite regression:
  The program code has to detect missing code **itself**.
- Make use of ideas from population dynamics: **balance of growth and death**.
  - **Growth:** Code continuously creates a copy of itself
    - A set of molecules continuously replicates itself, generating more and more identical replicas
  - **Death:** Code has a limited lifetime
    - AChem: When the memory is exhausted, each newly generated molecule randomly replaces another one *(remember: mate-replicate-spread game)*
  - Intuition: Defective code is not able to replicate and dies out.
  - Problem: Code replication is the result of the code's inner structure.
    How to write a program that replicates itself?

Quine: A Program that Outputs its own Source Code

- A **Quine** is a program that generates its own source code as output.
- Willard van Orman Quine (1908-2000): Philosopher and logician, examined indirect self-reference as in the following fragment:

"is not a sentence" is not a sentense

- Quines exist for each Turing-complete programming language.
- Quines consist of a description (**blueprint**) of the code and its **active version**.
  - e.g. in C: (see also the *Quine Page*)

```
 char BP[] =
"char BP[] = %c%s%c;main() {printf(BP,34,BP,34);}";
                  main() {printf(BP,34,BP,34);}
```

blueprint

active code

A Quine in Fraglets

A replicating Quine in Fraglets that generates an additional copy.

active part

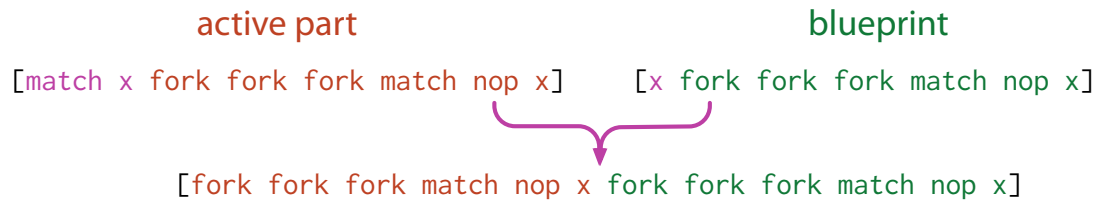
```
[match x fork fork fork match nop x]
```

blueprint

```
[x fork fork fork match nop x]
```

A Quine in Fraglets

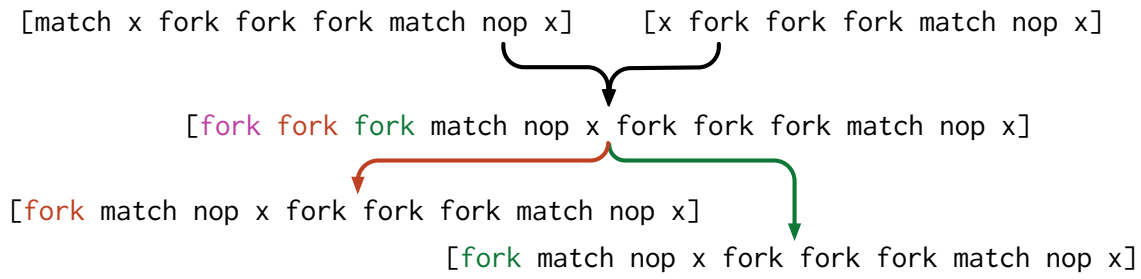
A replicating Quine in Fraglets that generates an additional copy.



- The **active part** reacts with the **blueprint**.
- The original molecules are replaced by a transient fraglet `[fork...]`

A Quine in Fraglets

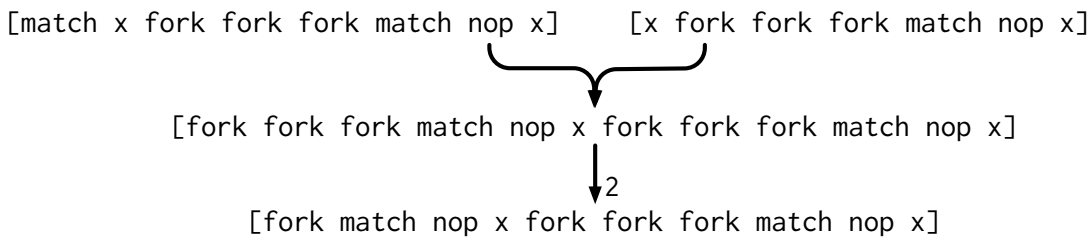
A replicating Quine in Fraglets that generates an additional copy.



- The transient fraglet splits up into two instances.

A Quine in Fraglets

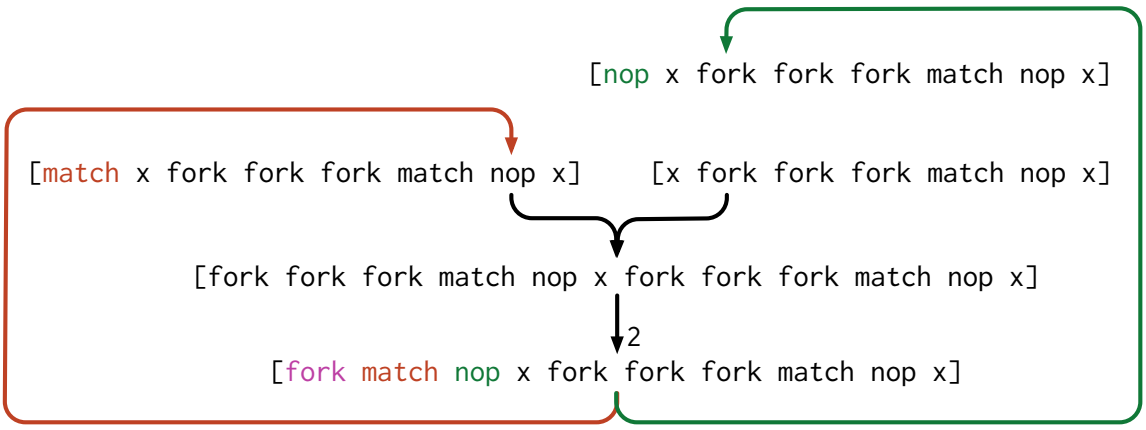
A replicating Quine in Fraglets that generates an additional copy.



- Actually, the two instances are identical. This is, two copies of another transient fraglet are generated.

A Quine in Fraglets

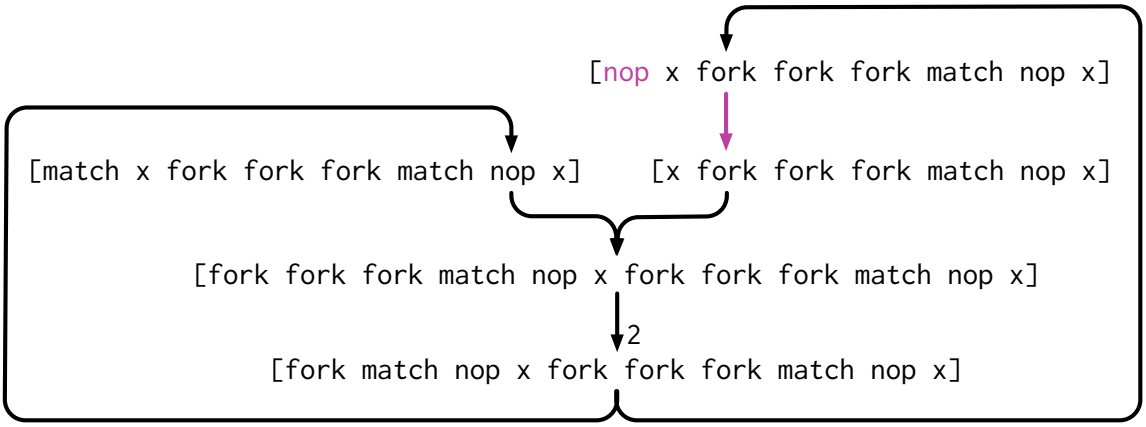
A replicating Quine in Fraglets that generates an additional copy.



- Both copies individually split up into two parts:
  One is the original **active part**, the other...

A Quine in Fraglets

A replicating Quine in Fraglets that generates an additional copy.



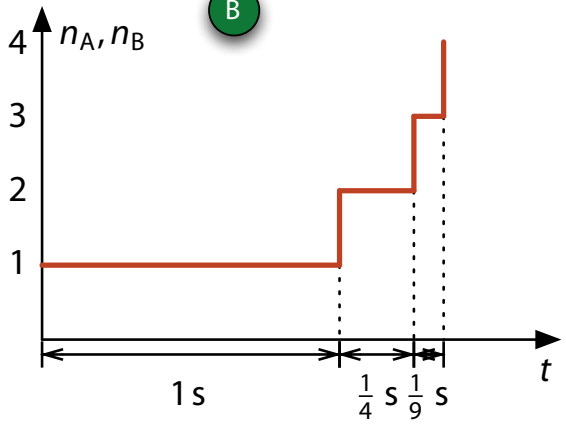
- Both copies individually split up into two parts:
  One is the original **active part**, the other produces the original **blueprint**.

Growth for Fraglets Quines

- The **active molecule (A)** and the **blueprint (B)** implicitly define the following autocatalytic reaction:

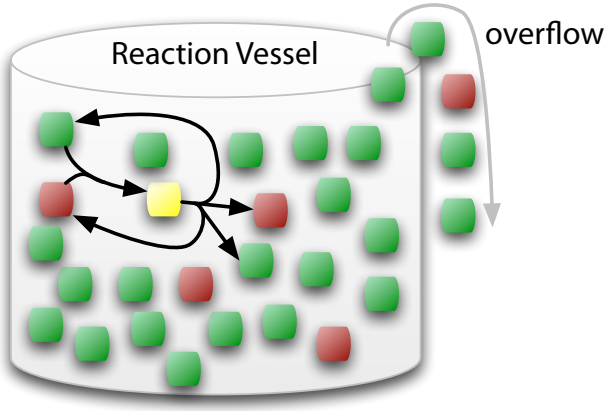
$$A + B \longrightarrow 2A + 2B$$

- The population of both species **grows hyperbolically** (reach an infinite concentration in finite time).
- Not realistic if each molecule occupies some memory.

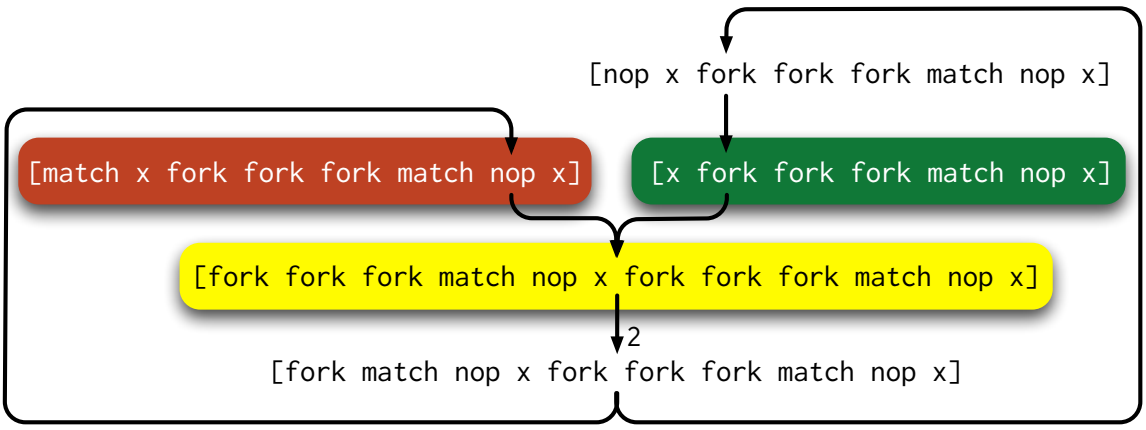


Death for Fraglets Quines

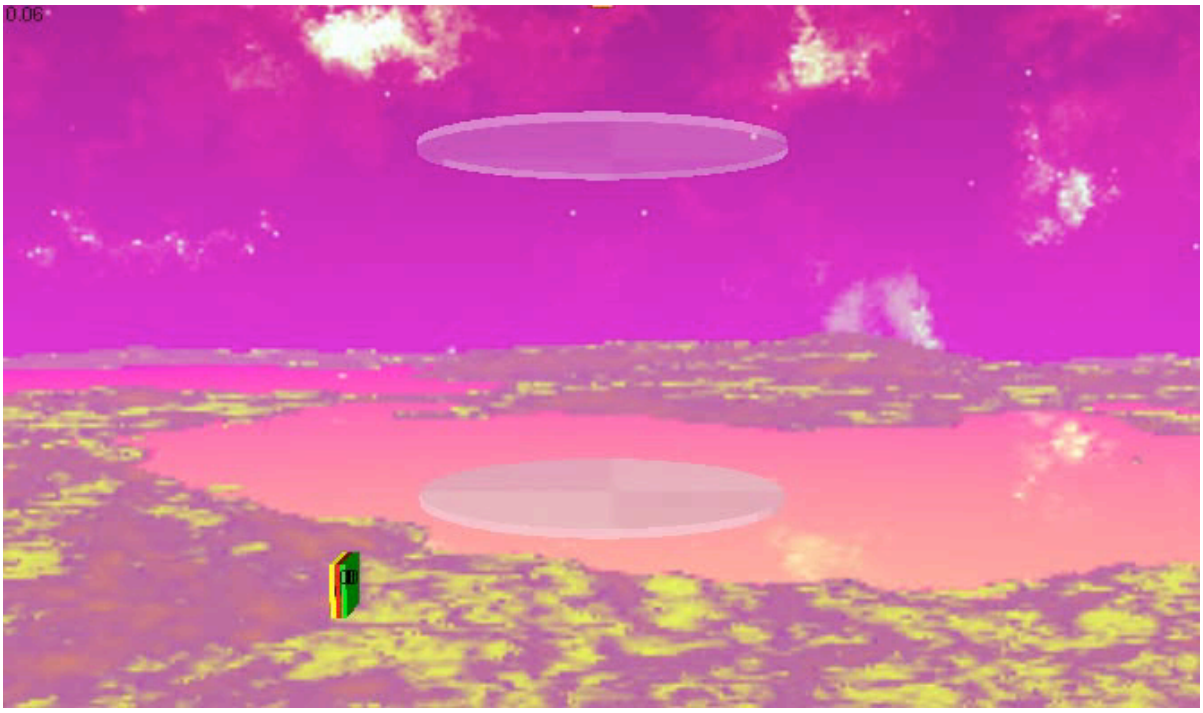
- Limited memory → Cannot store an infinite amount of fraglet strings
  → There is a natural (or imposed) upper bound of number of molecules: the **vessel capacity**.
- What happens if the population of molecules grows beyond this capacity?
  - Each newly generated molecule replaces a randomly chosen other molecule.
  - This simulates a **dilution flux**.
  - This applies **selective pressure** to the molecules in the vessel.



Balance of Growth and Death of Fraglets Quines

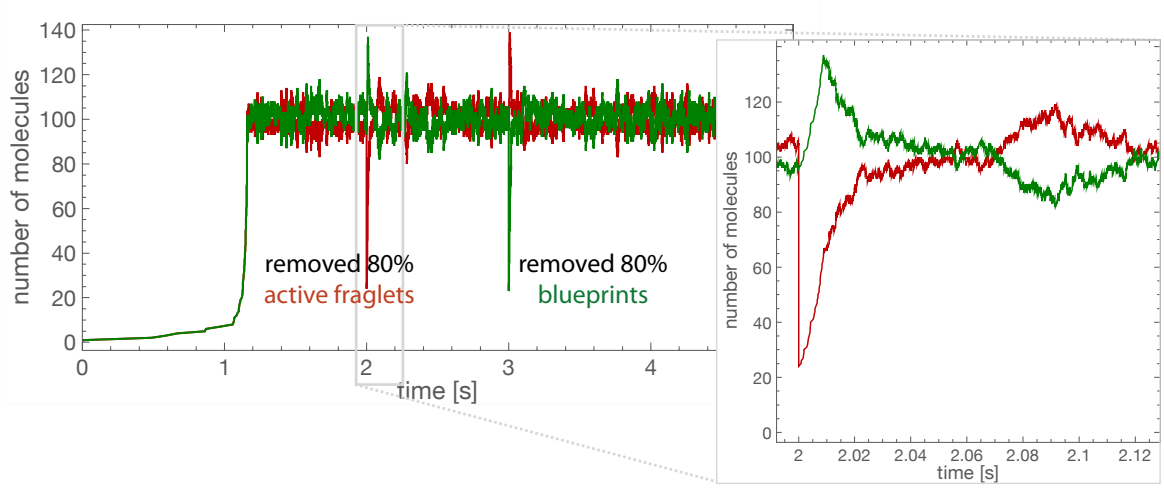


Balance of Growth and Death of Fraglets Quines



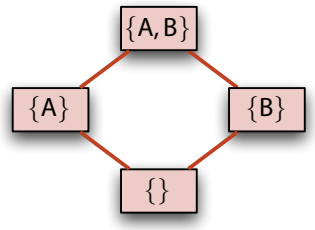
Balance of Growth and Death of Fraglets Quines

- Hyperbolic growth together with resource limitation leads to **survival of the common** (here: the first quinified program that populates the vessel).
- The program is **robust to invasion** from other programs.
- The program is **robust to deletion** of large parts of its own code.



Balance of Growth and Death of Fraglets Quines

- The organization diagram reveals the potential for coexistence of active parts (A) and blueprints (B)



- The resulting dynamics is described by the **Catalytic Network Equation**:
  N: vessel capacity; relative concentrations: $x_A = n_A/N, x_B = n_B/N$; in saturation: $x_A + x_B = 1$

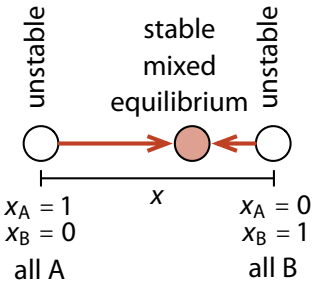
$$\dot{x}_A = \overbrace{x_A x_B}^{\text{growth}} - \overbrace{x_A \Phi}^{\text{death}}$$
$$\dot{x}_B = \underbrace{x_A x_B}_{\text{growth}} - \underbrace{x_B \Phi}_{\text{death}}$$
$$\Phi = 2 x_A x_B$$

Equilibrium: $\dot{x}_A = \dot{x}_B \equiv 0 \Rightarrow$

$x_A = 0;\ x_B = 1$

$x_A = 1;\ x_B = 0$

$\boxed{x_A = \frac{1}{2};\ x_B = \frac{1}{2}}$



Self-Healing Software by Autocatalytic Quines

- This approach is **different from traditional self-healing** techniques:
  - Self-healing by autocatalytic Quines requires no dedicated monitor/diagnose/repair modules.
  - We don't need to actively detect what code is missing!
    1. ~~We need to detect that code is missing.~~
    2. We need redundant information to restore missing code.
    3. We need dynamic control of this repair mechanism
  - **Inherent self-healing properties emerge** in this setting.
- But: There is no generic solution yet.
  - How to deal with mutations, or autocatalytic attackers (virus?)
  - Do we need an Artificial Immune System (AIS)?

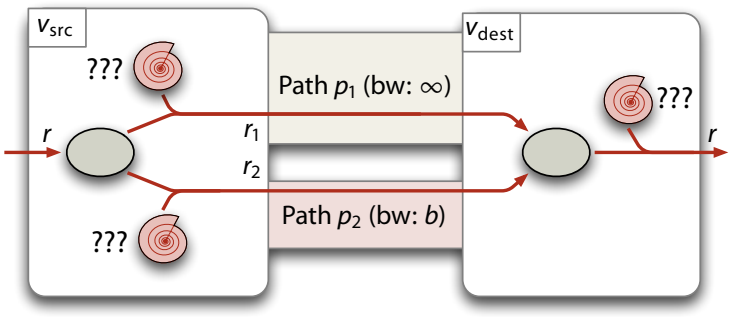
From Robust to Self-Healing Protocols

- A Chemical Networking Protocol Example:
  C3A - A Chemical Congestion Control Algorithm
- Self-healing protocols:
  How protocol software is able to heal itself from the loss of parts of its own code
  - A method to make chemical software self-healing
  - Example: A self-healing link-load-balancing protocol

A Self-Healing Link Load-Balancing Protocol

Meyer et al, *Chemical Networking Protocols*, Proc. 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), New York, 2009.

- **Goal:** Distribute a packet stream from source to destination over two redundant paths and
  - minimize packet loss,
  - recover from any molecule deletion attack.
- **Approach:** Install three rules that forward the injected data packets.
  - Two molecules in the source node forward the packet over either path.
  - The third molecule in the destination node delivers the packet.

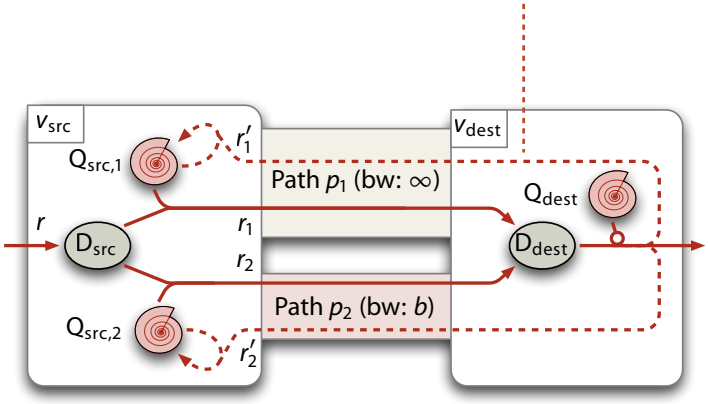


A Self-Healing Link Load-Balancing Protocol

Build the program from Quines

We build the three forwarding rules from modified **Quines**:

- $Q_{src,1}$ and $Q_{src,2}$ compete for the injected packet.
- However, $Q_{src,1}$ and $Q_{src,2}$ don't replicate immediately.
- The destination Quine Q_{dest} sends back an **acknowledgment** and replicates immediately.

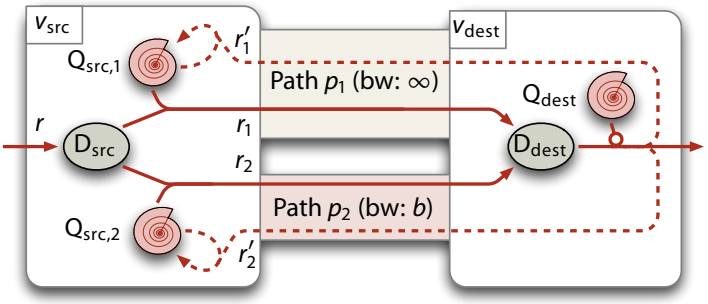


A Self-Healing Link Load-Balancing Protocol

Deferred Replication

Replication when receiving an acknowledgment:

- The received **acknowledgment triggers the replication** of the corresponding Quine in the source node ($Q_{src,1}$ or $Q_{src,2}$)
- If data packets or acknowledgments are lost, the rate of ACKs received (r'_i) is smaller than the rate of data packets sent (r_i) over that path.
- The replication rate of the Quines $Q_{src,1}$ and $Q_{src,2}$ is equal to the corresponding ACK rate.



A Self-Healing Link Load-Balancing Protocol

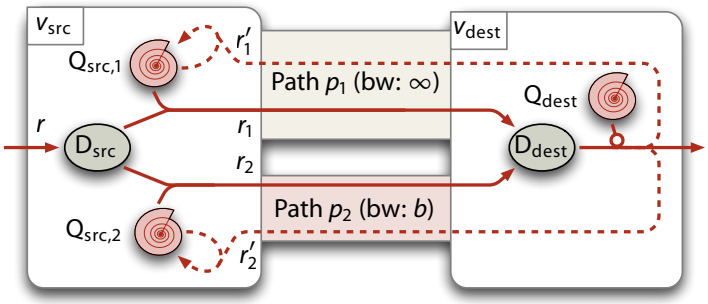
Minimization of packet loss

Emergence of packet loss minimization equilibrium:

- The Quines exhibit linear growth with (rate r_i') → **coexistence**
- Their relative concentration distribute according to their replication rate:

$$x_1 = \frac{r_1'}{r_1' + r_2'} \qquad x_2 = \frac{r_2'}{r_1' + r_2'}$$

- According to the law of mass action, packets are balanced over the paths proportionally to the concentrations of their Quines.
- Quine $Q_{src,2}$ reduces its concentration until no packets are los.

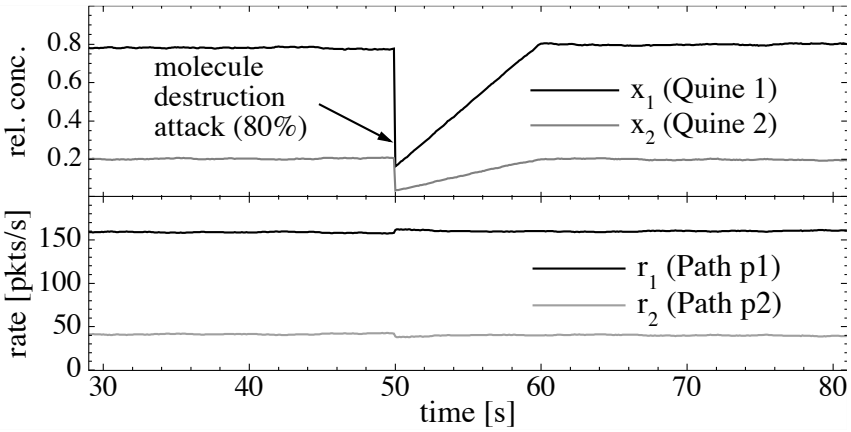


A Self-Healing Link Load-Balancing Protocol

Robust to Deletion Attacks

We built the program based on self-healing Quines only:

- The whole program is **robust to the deletion** of any of its parts.
- During such an attack, the remaining Quine copies continue to forward packets (only **slight degradation** of functionality).
- After a short time the **program fully recovers**.



From Self-Healing to Self-Evolving Protocols?

- Ultimate Goal: **Chemical program evolution** rather than design
  - Chemical systems quickly become complex (e.g. dynamics)
  - **Exploit emergent properties** (self-organization, robustness, etc.)
  - Unguided evolution of programs is still unsolved
  - Evolving networking protocols is even harder (e.g. distributed fitness evaluation, fairness rather than competition)
- The dynamics of chemical programs may give important insights:
  - Dynamics control which of the parallel execution path is processed at which speed.
  - Under selective pressure, the **growth rate** of a set of molecules implicitly **determines its fitness** (results from evolutionary dynamics).
  - Fairness problem: Quine - automatic fairness between code and data
- But still: many research questions open.....

Thank you