Mobile Code – a Long Term Perspective

Christian Tschudin, University of Basel

Invited talk, Prognet 2004, Lake Windermere, UK

Theme: Mobile Code Research as a broad and new challenge

– active networking: several systems, little innovation
– revisiting mobile code, communications and computations
– go low level, novel models, novel architectures!

Some Observations

- Active networking (AN) so far:
  strong program, weak results
- AN languages:
  geared towards human engineers
  (extending what they know)
  instead of mobile code's needs, potential.
- Accept radical definition of "active networking":
  distributed self-modifying code
  (otherwise it's not active)

Surprise Entry: Berkeley Motes

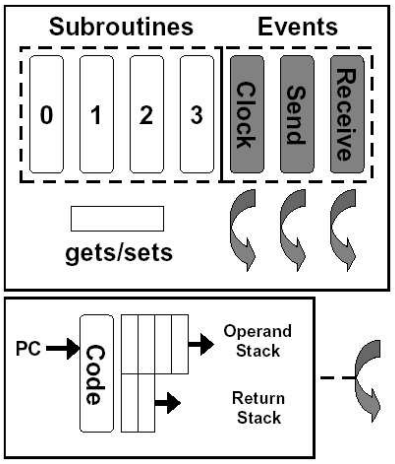
Active networking today is happening **elsewhere**:

- Wireless sensor networks
- “Smart dust” vision
- Can’t program tiny nodes on an individual basis:
  – network computing (e.g. data fusion)
  – “retask” the whole networks
- Extreme constraints:
  – a few kbit/s, a few kB RAM, MIPS range at most

Berkeley Motes and TinyOS

- First level of programmability:
  – sensors used for one task (at a time)
  – “decorated C” (nesC)
  – bundle app code with TinyOS libraries
  – viral (machine) code distribution as a base service
  (tool chain infested with Java based tools)
- 2nd level:
  – virtual machine “Maté”
  – capsules (24 Bytes)

Maté Programming Environment (2002)

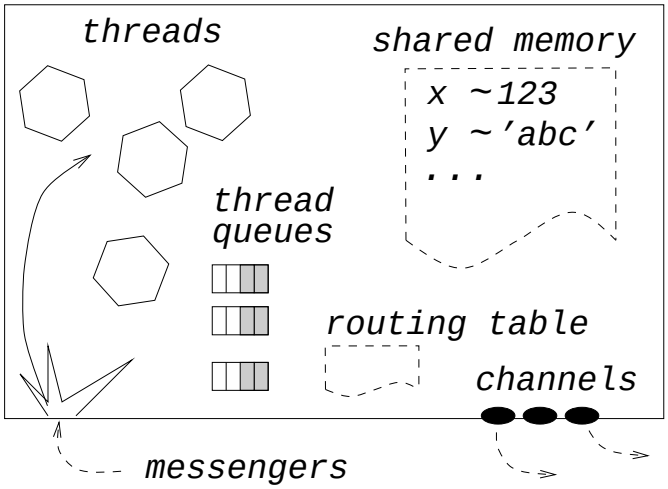


Maté

Mate Context

- Capsules executed on arrival
- Have their own context (stacks, PC)
- Capsules as subroutines
- Shared variable(s)

Flashback: Messenger M0 (1993)



Programming Example for Maté

```
pushc 1    # Push one onto operand stack
add        # Add the one to the stored counter
copy       # Copy the new counter value
pushc 7    #
and        # Take bottom three bits of copy
putled     # Set the LEDs to these three bits halt
```

Comments
– “persistent” counter on the stack
– timer event calls this capsule
– increments counter, copies the bits into LED latch

Maté Self-Assessment

Refrishingly honest:

- “language not usable”
  (UCB technical report, 2004)

But what conclusions to draw?

Active Networking Timeline

1969	Decode Encode Language (RFC 5)
1982	Softnet
1993	M0 Messengers
1994	Java
1997	Morphnet
1998	NetScript, ANTS, PLAN, Safetynet

All models are sequential, languages boil down to register- and stack machines (no surprise!).

AN Progress since then . . .

- Infrastructure (prog. routers)
  AMnet, Click, Janos, NodeOS, Promethos, Scout. . .
- Interfaces (P15xx, ForCES)
- Network Processors:
  IXP1200, PowerNP, PayloadPlus, MMC, EZchip,
  Motorola C-Port
- Few new AN languages, e.g. ESP
  but a Java companion (C#)

$\Rightarrow$ more of the same // Will it ever become CS textbook stuff?

Where should Progress come from?

- Sensor people say: currently too low level
  – go marcoprogramming, netwide DB etc
- Prognet04 says: go sideways and upwards
  – combinatorial (AN+p2p, AN+ad hoc, AN+. . .)
  – mitigating (policies)

Go higher level? Nth-generation language? More curbing systems?

I say:
– follow Feynman (“There’s plenty room at the bottom”)
– get the foundation right.

What’s wrong with AN today?

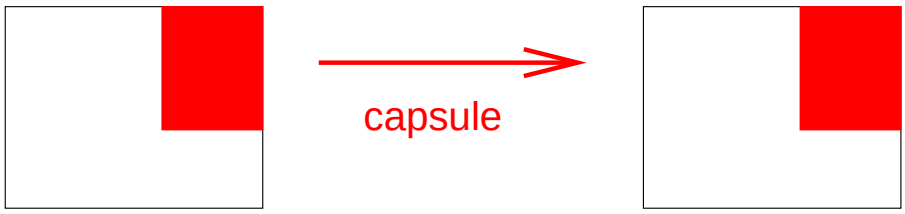
- No good formal model for code mobility
  – π-calculus: basically static, simulate mobility
  – Ambients: heavy weight agents
- Still stuck with capsule style
  – encoding problems
  – fuzzy borders between transient/persistent/shared memory
- Or chunk-wise download: requires a network, in a first place
- No AN network architecture yet: just an IP add-on
- No algorithmic progress (genuin active protocols)

AN not really active, today

Active protocols a mere question of efficiency?
Active protocols are fun for experimenting . . .

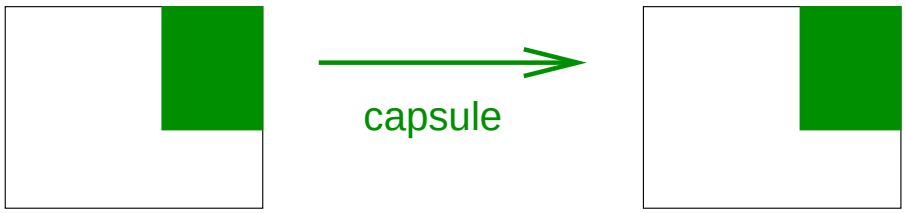
- . . . but once we know the useful *N* protocols, we can turn/burn them into a passive implementations and add a big `switch()` statement for user choice.
- That is, capsules as well as programmable routers: an economy tradeoff (choice or download)
- See Sensornets:
  Maté capsules better if task duration <= 5 days

AN not really active, today



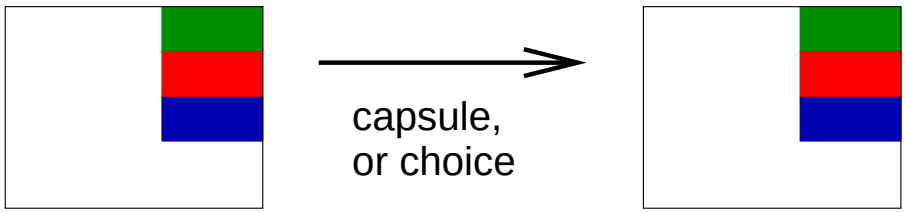
The great active protocol 1.

AN not really active, today



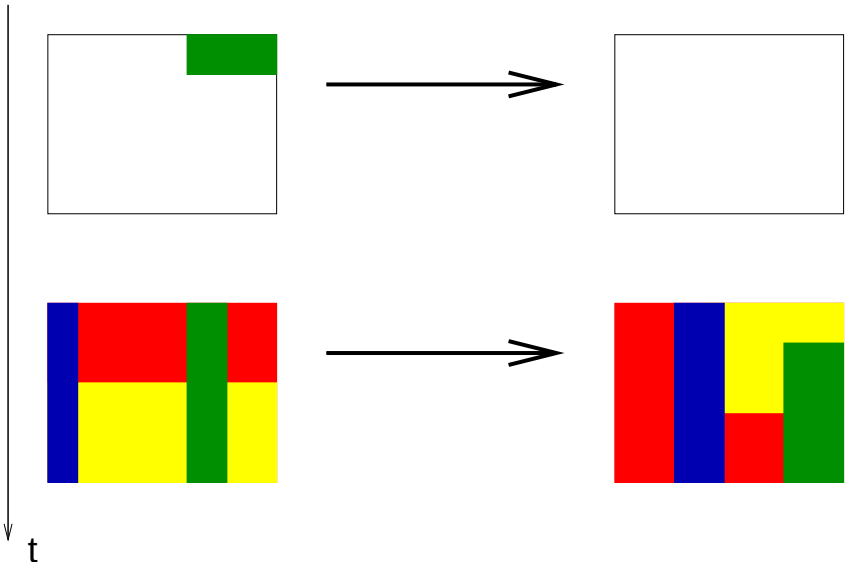
The great active protocol 2.

AN not really active, today



The great active protocol 1, 2 or whatever you choose.
Applies even more to programmable routers.

Active Networking – if it was really active



The system as a whole is changed, evolves.

An Attempt for a “activeness” Criteria (1)

A protocol is active
iff
removing the protocol from the system is more complex than writing a replacement protocol.

Goals:
– we want to exclude simple “plug-in/out”
– we want to exclude a simple system reboot

That is: an active protocol can have *persistent* effect!

An Attempt for a “activeness” Criteria (2)

Note on previous definition:

- At a different timescale we have this already
- Our (OS, network) systems evolve this way:
  – adding new functionality
  – almost impossible to fully replace old one
  – largely depends on humans in the loop
- Mobile code promise:
  – we can speed up this up
  – partial implementations: Windows updates
  – ultimately: self-modifying code at run-time.

Packet Languages and Chemical excution models

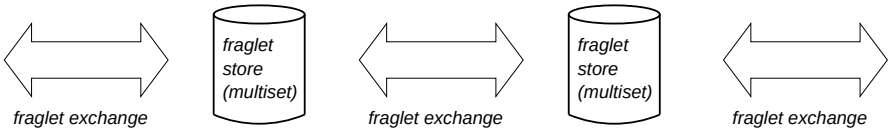
What other models then register+stack? First pointers:

- Post rewriting systems (1930ies)
  as a basis for packet header processing
- Gamma (Banatre and Le Metayer, 1986)
- Growing Point Language (Coore, amorphous computing, 1999)

Introducing Fraglets and metabolic protocol implementations . . .

Fraglets

Fraglet = computation fragment = “packet” = “rule” = “molecule”



- Fraglet = string *W* of symbols:
  W = [s_0 : s_1 : ... : s_n] (sequence of pkt headers)
- Head symbol selects fraglet processing $\rightarrow$ fraglet rewriting
- Two types of fraglet processing (shall be *O(1)*):
  – single fraglet transformation
  – chemical reaction (involving two fraglets)

A Fraglet Language – Packet Header Processing!

- Unary operation – special symbols *pop, send* etc:
  (*V* and *W* are words)

$$[\ \textit{discard} : V\ ] \rightarrow \emptyset$$

$$[\ \textit{pop} : t : s_i : V\ ] \rightarrow [\ t : V\ ]$$

$$[\ \textit{split} : V : * : W\ ] \rightarrow [\ V\ ] \text{ and } [\ W\ ] \quad (V \text{ has no } *)$$

$$[\ \textit{send} : n : V\ ] \rightarrow \text{send } [\ V\ ] \text{ to node } n$$

- Binary operation – two fraglets “react”:

$$\left.\begin{array}{r} [\ \textit{match} : s_0 : V\ ] \\ [\ s_0 : W\ ] \end{array}\right\} \rightarrow [\ V : W\ ]$$

Sample “Fraglet Program”: Change a Fraglet’s Tag

In: [i : W]

Out: [o : W]

Program: [match : i : o]

Execution Trace:

$$\left.\begin{array}{r}[\text{ match} : \text{i} : \text{o }] \\ [\text{ i} : \text{W }]\end{array}\right\} \Rightarrow [\text{ o} : \text{W }]$$

Sample “Fraglet Program”: Change Fraglet Tag (contd)

Persistent version of “match”: matchp

In: [i : W]

Out: [o : W]

Program: [matchp : i : o]

Execution Trace:

$$\left.\begin{array}{r}[\text{ matchp} : \text{i} : \text{o }] \\ [\text{ i} : \text{W }]\end{array}\right\} \Rightarrow [\text{ matchp} : \text{i} : \text{o }],\ [\text{ o} : \text{W }]$$

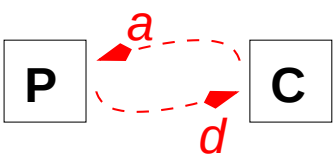
Sample “Fraglet Process” : Producer/Consumer

a.k.a “confirmed delivery protocol”

Prod: [matchp : ack : data : Payload]

Cons: [matchp : data : split : ack : $*$: discard]

Ack: [ack]



Execution Trace:

[a]

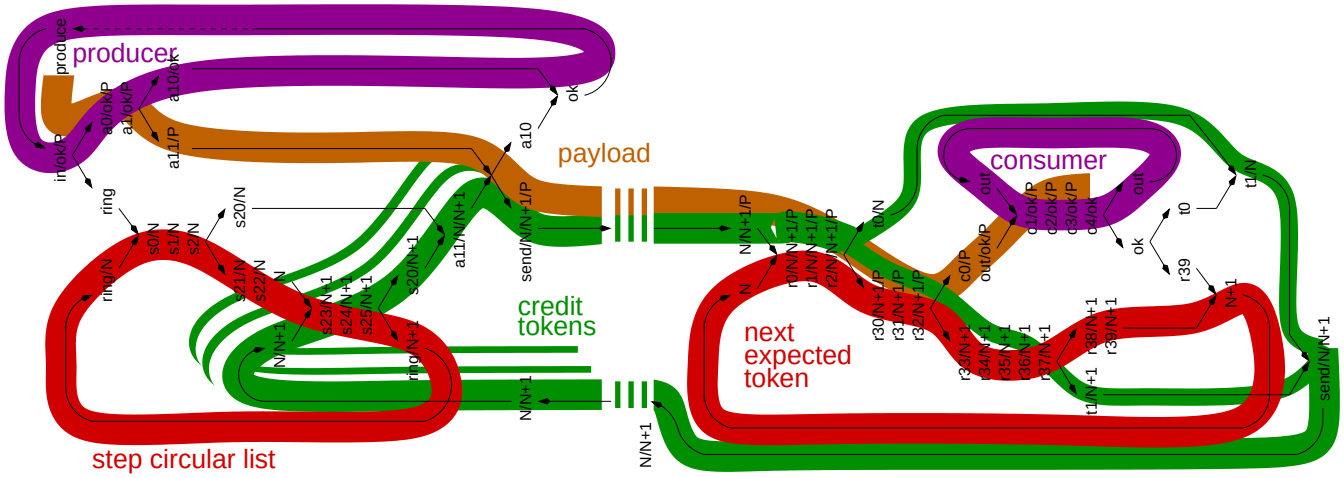
[matchp : a : d : Payload], [a] $\Rightarrow$ [d : Payload]

[matchp : d : split : a : $*$: discard], [d : Payload] $\Rightarrow$ [split : a : $*$: discard : Payload]

[split : a : $*$: discard : Payload] $\Rightarrow$ [a], [discard : Payload]

$\Rightarrow$ [a] . . .

Fraglet Protocol: Flow Control with N Credits (contd)



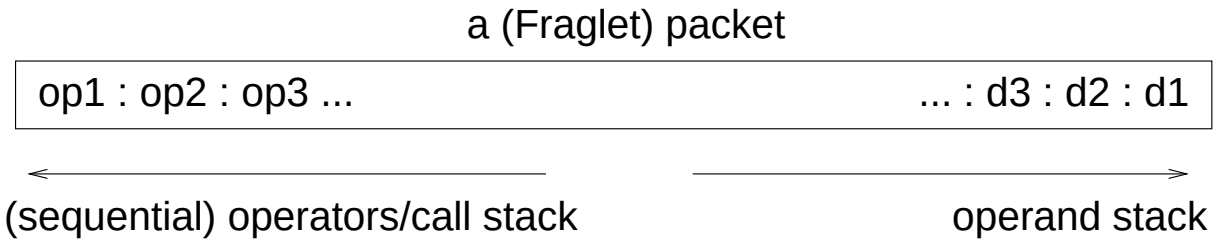
Total system: ca 35 axioms (fraglets), 3–10 symbols long

– everything is a fraglet (active packet), even credit tokens

– coupled control and processing loops

Fraglets and Sensornets

Extend Packet Header Processing: packet tail is a stack.



Redoing the Maté capsules, in Fraglets:

– `[ match : cnt : add : copy : push 7 : and : putled : cnt : 1 ]`
– the packet is the context (contains stack)
– moving contexts easy to express

Fraglets and Activeness

- Fraglet store is basically a big “soup”
- Difficult to cleanse a running system:
  – know all intermediate products
  – service could be used by others
- Link to network architecture:
  – packet header as “selector” (SelNet)
  – selector determines what processing to apply

AN: Beyond Hacking Execution Supports (1)

Real CS puzzles! Researching the foundation of mobile code.
E.g. Trustworthiness:

- Can you safely execute mobile code in a hostile environment?
  – compute secrets on a malicous host!
- Preliminary results (1998, with T. Sanders):
  using homomorphic encryption schemes
- Function is encrypted (encoded) such that
  – host can execute single basic instructions
  – but cannot extract the algorithm, nor the data!

AN: Beyond Hacking Execution Supports (2)

Robustness:

- Can you write a correctly working program that works despite removing an arbitrary instruction?
- Preliminary resuls (2004, with L. Yamamoto):
  use redundant execution paths, with Fraglets
- Gradual difference to packet loss
- Includes signaling which code was removed:
  – code can potentially heal itself!

A (unsolved puzzle): write an active packet Quine.

AN: Beyond Hacking Execution Supports (3)

Robust execution of Mobile Code

- Related area: Quantum computing
- Problem statement:
  – decay of “state” and noise in processing Qubits
  – need “error correcting execution”
  – has been demonstrated for logical gates

CS question: what algorithms (beside logical gates) are encodable
– *directly* (without relying on these gates)
– *dynamically* (not static gate topologies)

Conclusions

Mobile code as a broad research area: it has just started.

- Active networking:
  Currently little to report back home,
  little to put into CS textbooks
- Look beyond restricted (quasi active) mobile code
- Towards a real and active mobile code world
  – other execution models
  – other network architectures

Go lowlevel, go theory, go active, go self-modifying!