Fraglets O(1) Interpreter:
Essential goal or awkward restriction?

BIONETS Fraglets meeting, Brussels, Jan 29, 2007

Christian Tschudin, UBasel

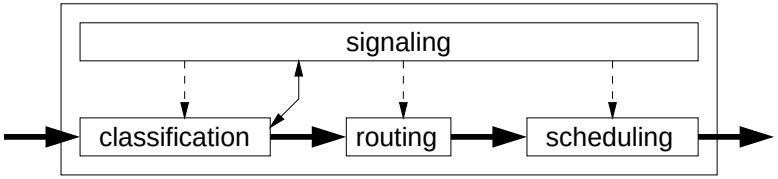
“Some whys, some hows, some donts”

a) O(1) interpreter: fraglet length
b) O(1) interpreter: store size
c) The case of string manipulations
d) The case of number tags, and tag comparisons

(This is not a fraglets tutorial, some familiarity is needed.)

Origin of Fraglets: Active Networking (AN)

Networking in the “fast path” of routers



- Killer argument against AN in every packet: too slow and no match with reality: fastpath in routers consists of a single lookup
- My goal: gradual AN “spectrum”
  – one instruction per packet OK,
  – two instructions probably OK too,
  – some limit. After this, packet goes into slow path.

Speed Concern

- Since early (2002), I cared about **molops**
- molops = “molecular operations/sec”
- Historic values were:
  – 500’000 molops (Alpha, 2002)
- Not tested recently, but probably quite bad today:
  – “random selection” not well implemented

O(1) Forwarding

Default action of classical router is forwarding.

- A fraglets system should include forwarding behavior, implement it with high performance
- Example: Let incoming packets do source routing:
  packet = [$\texttt{dest}_1$: $\texttt{dest}_2$: ... $\texttt{dest}_n$: payload]
- [matchp : $\texttt{dest}_i$: send : $\texttt{NextHopAddr}_i$]
  in each node does the job:
  – this rule is a “forwarding entry”
  – node does not need to know full content of incoming packet

O(1) Forwarding (contd)

Some consequences:

- Impose strict header matching, no deep packet inspection
- Avoid packet copy, permit lazy receive:
  – leave the packet as long as possible in line-card buffer
  – copy its content only if needed (e.g. send)
- Preserve wormhole routing capabilities
  (we can start forwarding when first symbol is read).

My dream:

– **Photonic fraglets** i.e., light path switching with tag matching!

– could we have RFID+ tags store one fraglet, phys mobility?

O(1) Transformation Instructions

In general: routers do store-and-forward operation.
Reading full packet, we can parse it at the same time

- Are our transformation instructions O(1) ?
  (i.e.: not dependent on packet length)
- In principle, O(1) true for almost all our transformations so far: `nul, exch, dup` etc
- What about `split` ? Hopefully we can handle this with auxil. data obtained during parsing (“where are the stars”) and propagate this info across all packet manipulations.

Overview

a) O(1) interpreter: fraglet length

b) O(1) interpreter: store size

c) The case of string manipulations

d) The case of number tags, and tag comparisons

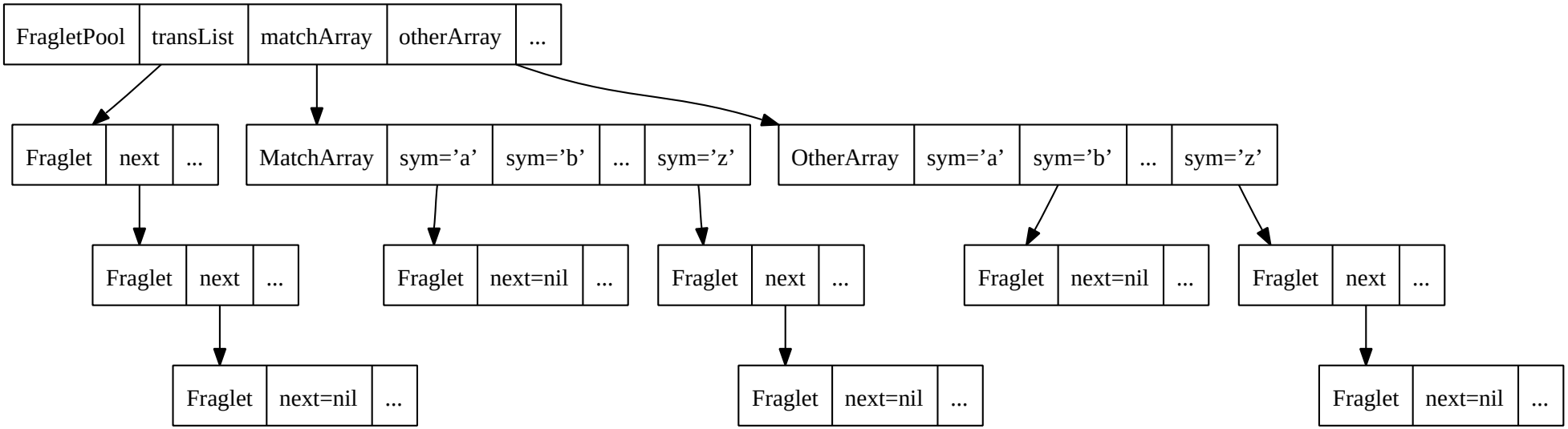
O(1) Reaction Rules

Conceptually, a fraglet must be tested with every other fraglet in the pool for reactions. Is this O(n), n the pool size, or O(1)?

- Assume finite symbol set S
- Partition the fraglet pool by symbols
- Roughly $1 + 2|S|$ partitions:
  a) one for all fraglets starting with a transformation keyword
  For each non-keyword symbol $s \in S$:
    b) one for all fraglets of the form `[ match : s : ...]`
    c) one for all fraglets of the form `[ s : ... ]`

O(1) Reaction Rules (contd)

Internal data structure (at least since fraglets-0.10, July 2003)



For a given symbol s, examine `matchArray` and `otherArray`:

– check whether the two lists for s are non-empty,

– in this case: reaction. Needs constant time to decide, O(1) !

O(1) Reaction Rules (contd 2)

A careful analysis still to be done:

- In the worst case, need to walk through all symbols:
  ∃ explicit list of match candidates, instead of searching?
  [Internal side note: this relates to the attempt with “hints”]
- Partitions are currently implemented as linked lists,
  we parse all of them twice for adding *random selection*
  among possible matches: how to avoid this?

(Guess: maintain vector of fireable fraglets)

Overview

a) O(1) interpreter: fraglet length

b) O(1) interpreter: store size

c) The case of string manipulations

d) The case of number tags, and tag comparisons

String Manipulations – not O(1) ?

I'm not sure that O(1) applies to everything already inside fraglets, even in theory:

- `split` needs additional implementation study
- `match` (aka `strcat()`) needs additional study. Example:

  ```
  [match : a : veryLongTail1] [a : tail2] --> ...
  ```

  currently implemented with buffer copy

List of other desirable string manipulations clearly outside O(1):

`subst()`, `strchr()`, `strcmp()`, `index()`, `sort()` ...

and therefore rejected **:-(**

Overview

a) O(1) interpreter: fraglet length

b) O(1) interpreter: store size

c) The case of string manipulations

d) The case of number tags, and tag comparisons

Adding Numbers to Fraglets

Lidia proposed: **numbers**, operation on numbers, comparison

First set of examples:

```
[ sum : t1 : 1 : 2 : tail ]          --> [ t1 : 3 : tail ]
[ < : tif : telse : 6 : 7 : tail ]   --> [ tif : tail ]
[ = : tif : telse : 4 : 5 : tail ]   --> [ telse : tail ]
```

- These are **transformations**, no problem to spot them in O(1)
- (Natural) number support was added in fraglets-0.20

Numbers as Symbols

Problem with numbers as special category:
Can you “match” on numbers? Example:

```
[ match : 2 : tailA ] [ 2 : tailB ]  --> [ tailA : tailB ]
```

- Seems natural, so yes, we will support it.
- Problem: `matchArray` now as big as number space (2^{32}) !
- Can be solved by converting `matchArray` into hash table and open addressing.
  Implementation is ongoing (first attempt was buggy)

Numbers as Symbols: (Over-) Generalization?

As pointed out by Lidia: Shouldn't '=' be equivalent to 'match' ?

```
[ match : 2 : tailA ] [ 2 : tailB ]  --> [ tailA : tailB ]
[ =     : 2 : tailA ] [ 2 : tailB ]  --> [ tailA : tailB ]
```

and by generalization:

```
[ <     : 3 : tailA ] [ 4 : tailB ]      --> [ tailA : tailB ]
```

Note: '=', '<' now a reaction, not a transformation anymore !

- 'match' and '=': it's just renaming, still O(1) execution.
- Can '<'-reaction be implemented in O(1)? Probably not.

Proposal: Compact Fraglet Instructions vs Extensions

Pragmatic proposal: Distinguish among

- **“compacts”**: O(1) instructions
- **“extensions”**: for often-used fraglet manipulations, although not O(1). Example: `strlen()`, `subst()`
- All extensions OK? Should, at least theoretically, be implementable with compact instructions only:
  – some nodes provide them natively
  – others emulate them with fraglets.

Debate: how to handle things not expressible with compacts?
`doubleMatch`, `membrane`, `tagNotPresent ...` what else?

Outlook 1: Hardware Tricks vs. Theory ?

After all, packets have finite length L, same for fraglet pool:

- Given this, any operation on a packet is O(1),
  even scalar (1 clock cycle)! Just throw enough hardware at it.
- Example: `strchr()` needs L comparison gates, easy

But “mind the curves”:
– economics (cheap devices)
– technology limits (light path)
still will bind us to sequential execution, and packet size will increase (ethernet: yesterday 1.5KB, today 4KB, tomorrow 64KB?)

Outlook 2: Is there an Auto-Catalytic Threshold ?

Is strict header matching (single tag) sufficient
to support emerging “life cycles”?

- Fraglets have no “deep structure”, parenthesis etc
  (except perhaps `split()` support)
- Closed system: no symbol conservation yet,
  – symbols are consumed, they evaporate
  – but we can write “code explosion” easily

Hypothesis: yes, it’s possible (although perhaps not aesthetic)
And if we add “decaying” fraglets? $\rightarrow$ We trust in resilient SW

Conclusions

- I continue to defend O(1), for keeping the path open to use fraglets in the network core, as well as other resource constrained environments like sensor nets.
- Natural number support was added to fraglets
- Number operations as reactions? Would have to abandon O(1)! Also new semantic questions: Is this what we want?

```
[ + : 3 : tailA ] [ 4 : tailB ]  ?--> [ 7 : tailA : tailB ]
```

- More discussions ahead: signed, rationals, reals, membranes

Questions, comments?