Melange: Creating a "Functional" Internet

Anil Madhavapeddy et. al.
(*presented by Stefan Kalkowski*)

June 13, 2007

- Vulnerabilities in Internet protocols often in the marshaling and buffer management code
- Type-unsafe languages are vulnerable to security and reliability problems
- Use of type-safe languages implies performance penalities

- Strong static typing and bounds checking
- Meta-programming for marshaling and demarshaling packets

Objective Categorical Abstract Machine Language

- ML derivative (functional language)
- fast automatic memory management (garbage collection)
- allows imperative and object-oriented programming
- bounds-checking at runtime

Meta Packet Language

- Specification of binary network protocols (no IDL)
- With pre-compiler with OCaml backend
- can be used to create bidirectional parsers
- produced code minimises memory allocation and bounds-checking overhead
- *zero-copy* interface
- non-lookahead decision-tree parsing algorithm

MPL Grammar

- ordered list of named fields
- wire builtin types (bit and byte fields, unsigned integer 16, 32, 64)
- MPL types (integers, strings and booleans)
- target language types
- *classify* keyword for parsing decisions depending on already evaluated fields
- attributes for specifying invariants, default values or alignment restrictions
- *variant* statement maps values to labels

```
packet ethernet {
  dest_mac: byte[6];
  src_mac: byte[6];
  length: uint16 value(offset(eop)-offset(length));
  classify (length) {
    46..1500: "E802_2" ->
      data: byte[length];
    0x800: "IPv4" ->
      data: byte[remaining()];
    ...
    };
    eop: label;
}
```

- extra library that includes I/O and buffer management

Packet Environment

```
type env = {
    buf: string;
    len: int ref
    base: int;
    mutable sz: int;
    mutable pos: int;
}
```

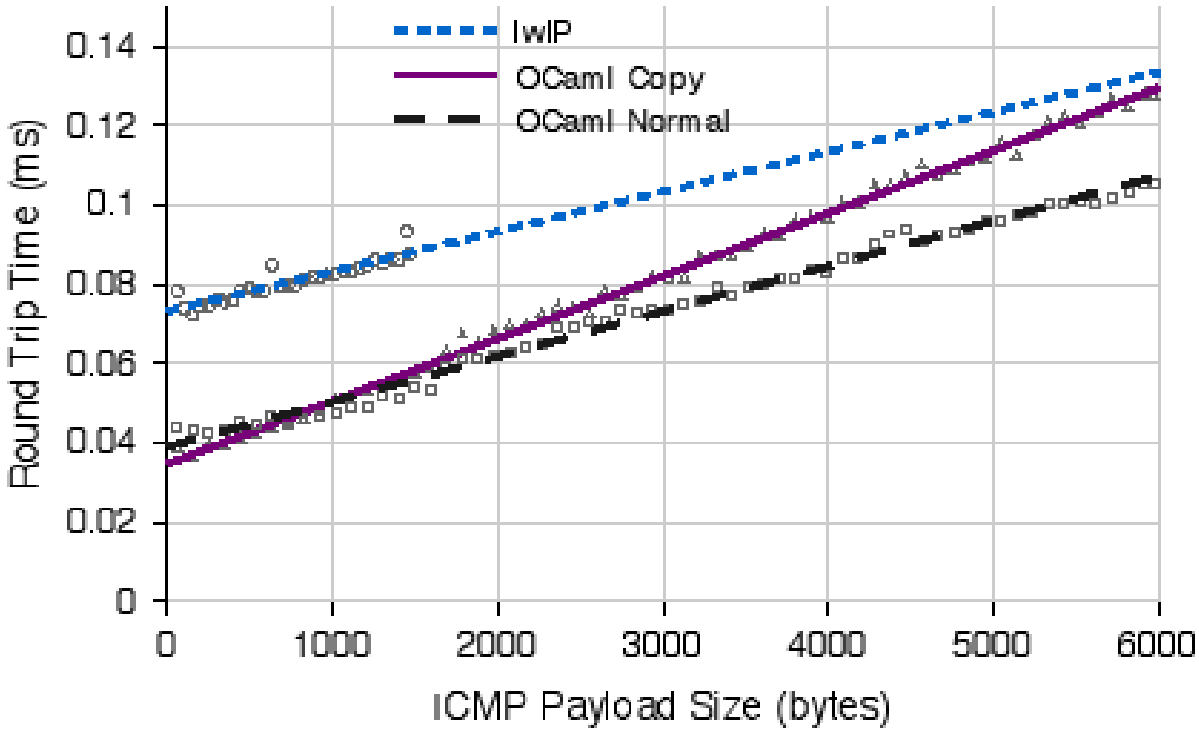
OCaml Interface

- generated OCaml code internally has imperative coding style
- exposes functional objects as packet representations
- packet creation passes partially nested to minimise copying
- enables reflecting of network packets

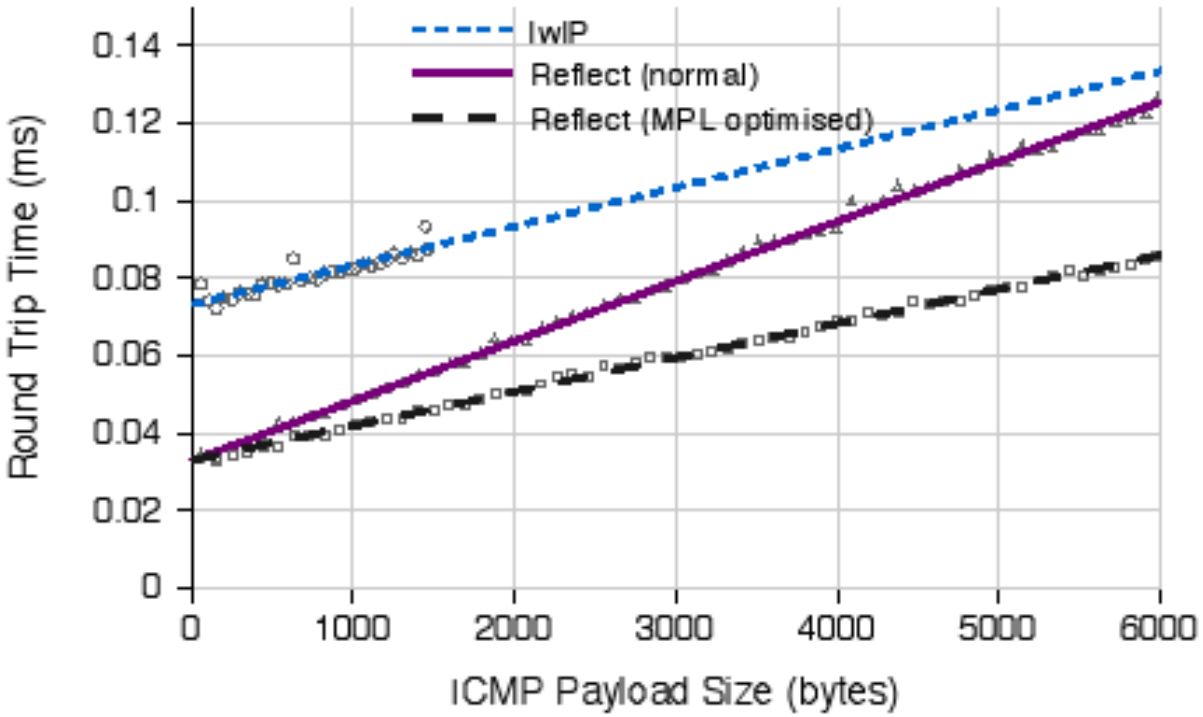
Overall performance test

- OpenBSD with tun/tap interface
- ICMP Echo-Reply test
- Opponent: 1wIP user-level networking stack
- shows impact of additional copying and bounds checking

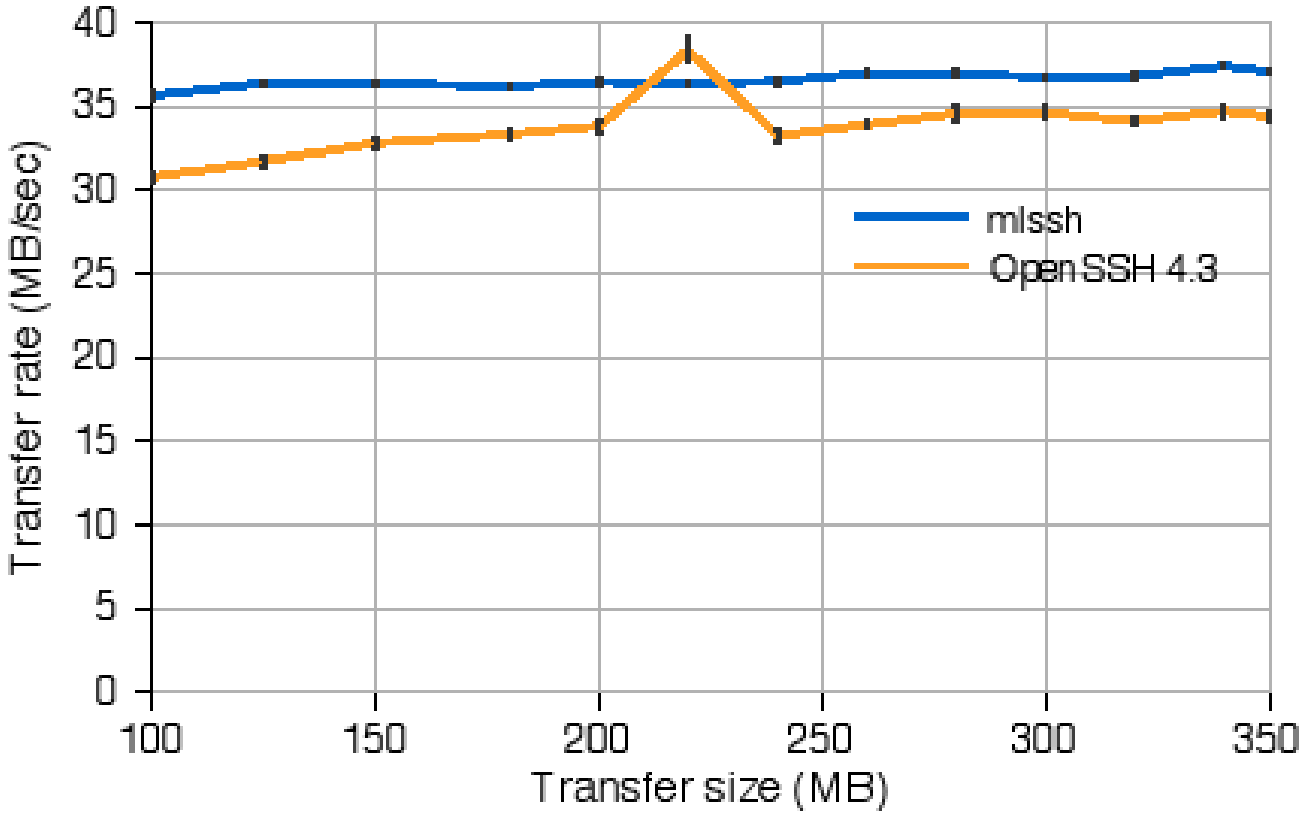
- Latencies for lwIP vs OCaml functional version which copies data and a normal MPL version *(lower gradient is better)*



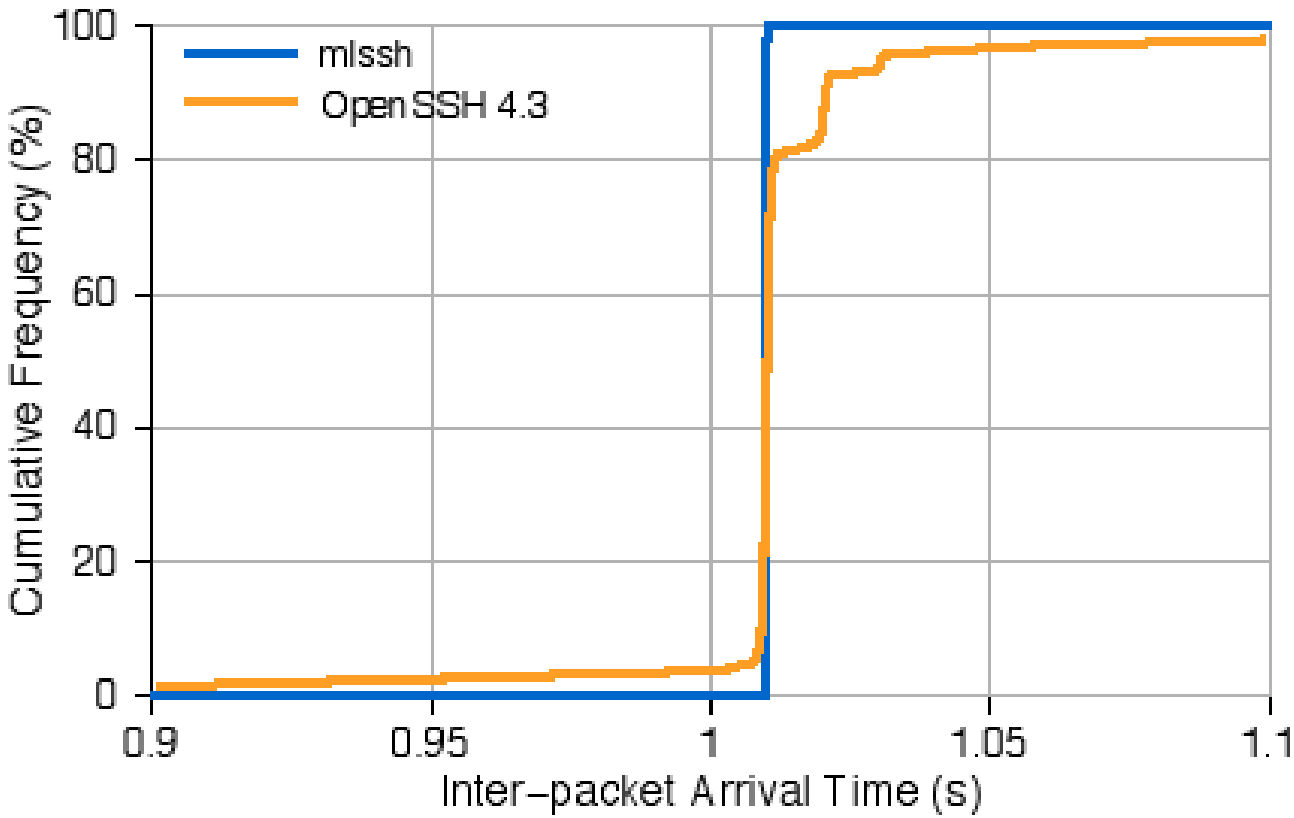
- Latencies for lwIP vs OCaml reflector version with MPL bounds optimisation off and on *(lower gradient is better)*



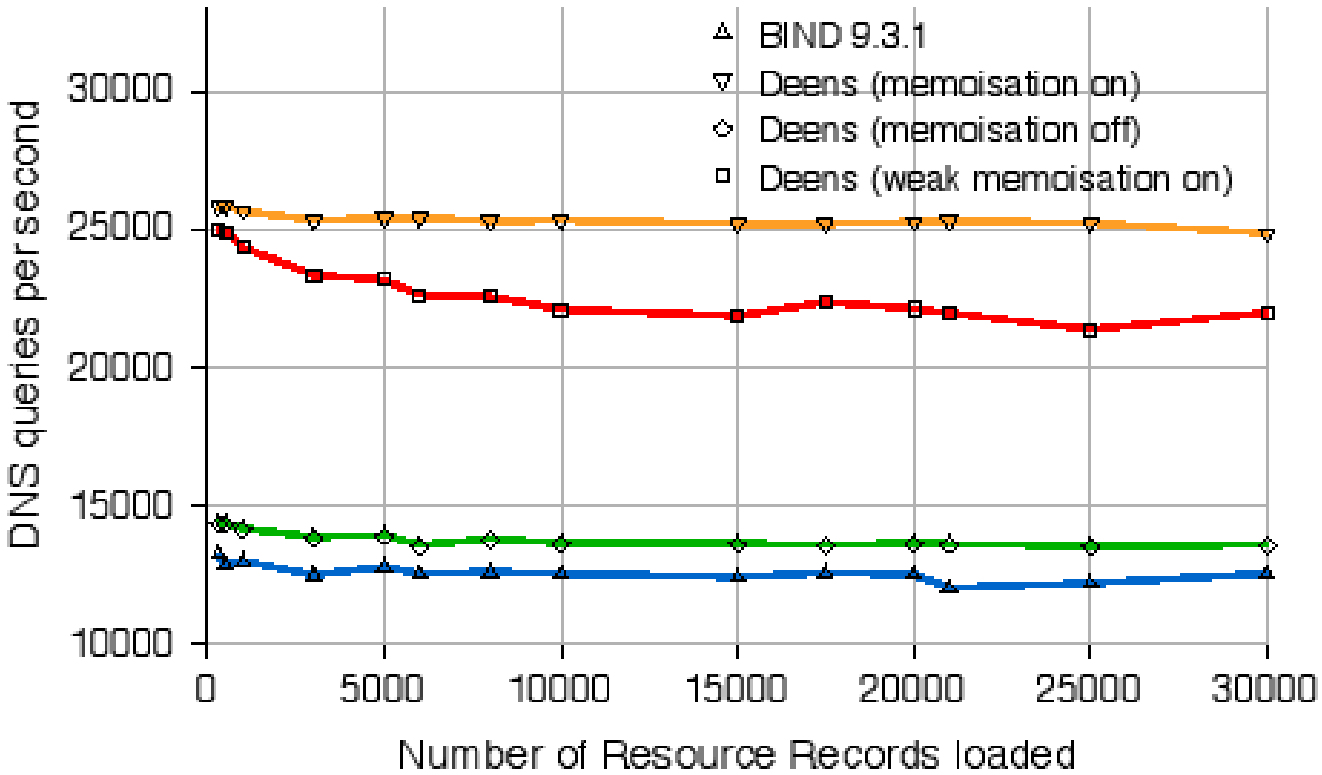
- Throughput of OpenSSH vs MLSSH without encryption and message hashing



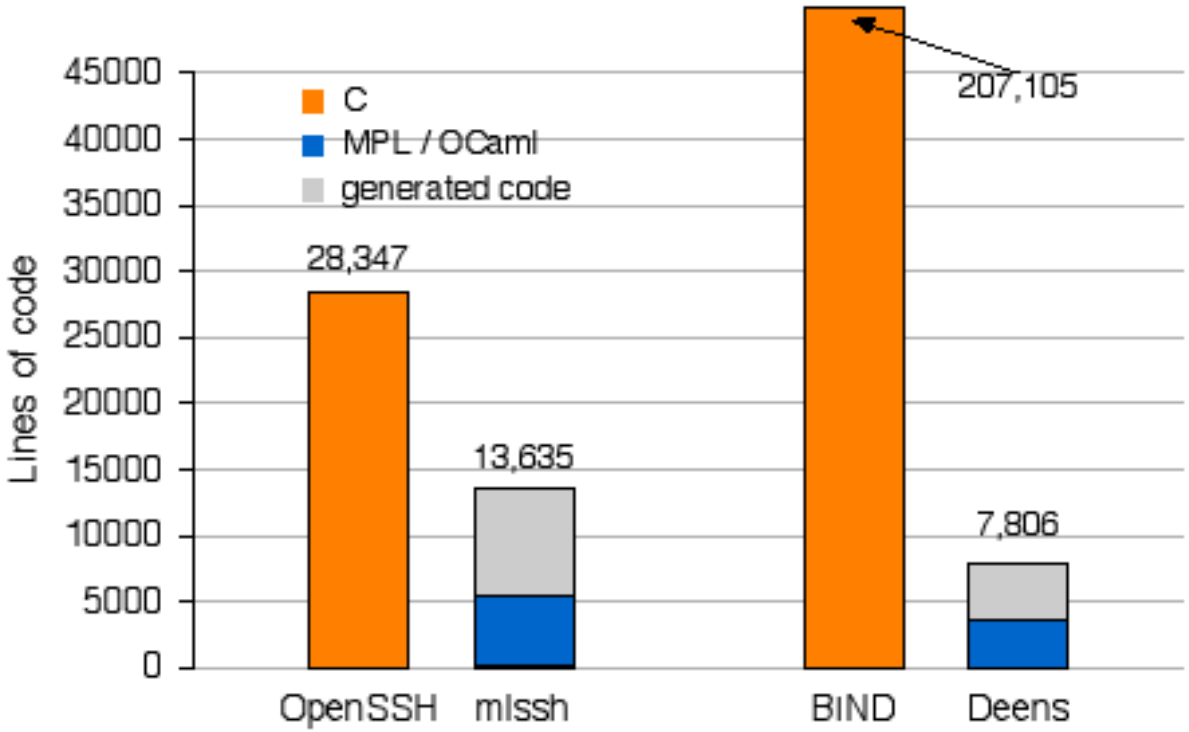
- Culmulative Distribution Function of inter-packet arrival times



- BIND vs DEENS throughput



- Relative code sizes



Summary

- using type-safe implementations doesn't mean performance loss in general
- using OCaml memory management without over-using major heap results in even better performance, than manual memory management
- smaller code basis helps to keep the survey and to change protocol implementations
- OO-style framework helps to combine and integrate internet protocols within new apps

???