MAPI-X
MAPI on top of the Intel IXP1200 Network Processor

Trung Nguyen
Willem de Bruijn
Georgios Portokalidis
Herbert Bos
so far mainly focused on common NICs and DAG

- non-programmable
- all the smarts are in MAPId

however, originally: push smarts to lowest level

- in the hardware
- so HW needs to be **programmable**
- COMBO6

or an existing network processor

- IXP1200
- MAPI-X allows users to push filters to HW
so far mainly focused on common NICs and DAG

- non-programmable
- all the smarts are in MAPId

however, originally: push smarts to lowest level

- in the hardware
- so HW needs to be **programmable**
- COMBO6

or an existing network processor

- IXP1200
- MAPI-X allows users to push filters to HW
IXP1200 Architecture
High-level overview

NPU (IXP1200)

Control Processor

StrongARM

Microengines

Gbps ports

scratch

Pentium

mmap

SRAM

DRAM

Filters, etc.
MAPI-X

• implementation based on FFPF
  • minimise copying, context switching
  • support common NICs, special hardware (e.g. IXP1200)
MAPI-X

- implementation based on FFPF
  - minimise copying, context switching
  - support common NICs, special hardware (e.g. IXP1200)
MAPI-X

- implementation based on FFPF
  - language neutral
  - flows can be combined
MAPI-X on FFPF

- **buffers in SDRAM**
  - shared packet buffer (circular)
  - index buffer (circular)
  - memory area
- **memory mappable**
  - “synchronisation” across PCI
  - by StrongARM
- **Zero copy (and other)**
  - ME(0) dumps pkts (moves W)
  - host explicitly advances R
  - “slow reader preference”
**MAPI-X on FFPF**

- **MEs check packets**
  - ME(0) places pkts in SDRAM
  - ME(1-5) determine whether pkts stay there

- **StrongARM code**
  - talks to host processor krl code
  - receives “advance read” actions
  - acts accordingly: move R*

- **implemented via polling**
  - tasklets to poll at reasonable times
To copy or not to copy

- "zero copy" is nice
  - provided host code doesn’t access the packets much
  - else: all accesses across PCI
- "copy once" is nice
  - if pkt accessed a lot in host
  - and no more in MEs
- both models are supported
  - the model can be chosen by MAPI-X configuration parameters
The flow expressions

- code in ME(1-5)
  - should be loadable by users
  - flexible admission control to check whether a user can do this
  - in terms of privileges and in terms of available resource
- code in MEs is plugged in
  - in a “slot” in a skeleton function
  - compiled to ME object code
- microengine C is used
  - new compiler currently developed
Software structure

- Applications
  - FFPF toolkit
  - MAPI
- Userspace
  - libpcap
  - FFPF_{linux}
- Userspace API
  - FFPF_{linux}
- Kernelspace
  - PCI daughterboard lib
  - FFPF_{ixp}
- ixp1200
MAPI and FFPF

**MAPI**

- **design**
  - `create_flow(eth0)`
  - `set_option(type_raw)`
  - `apply_func(bpf_filter)`
  - `apply_func(bytecount)`

**FFPF**

- **implementation**
  - `create_flow(ixp0)`
  - `set_option(type_raw)`
  - `apply_func(bpf_filter)`
  - `apply_func(bytecount)`
  - `bpf(proto ip)`
  - `device(ixp0)`
  - `device(eth0)`

H. Bos – Leiden University 30/04/2004
Flows and filters

Filters

- passive
  - fpl2_compiled()
  - bytecount()
  - bpf(bpf_expr)
  - device(ethX)
  - device(ixpX)
  - etc ...

Flows

(Active instantiation)

- bytecount
- bpf(proto ip)
- device(ixp0)
- device(eth0)

\texttt{dev(*)->bpf(ip)->bytecount}
Multiple flows

dev(*) -> bpf(ip) -> bytecount

dev(*) -> bpf(ip) -> fpl2()

dev(ixp0) -> fpl2()

dev(ixp0) -> bpf(proto ip)

device(ixp0) -> device(eth0)

fpl2() -> bytecount
multiple environments

kernelspace

userspace

board (ixp)
ixp1200  kernelspace

gigabit  

zero-copy ioremap(..)

index buffer  index buffer

packet buffer  packet buffer

ixp0
kernel to userspace

kernelspace  userspace

zero-copy mmap(..)

bpf(..)

index buffer  index buffer

packet buffer  packet buffer

reg exp(..)
Evaluation

- full evaluation not complete
  - current version of the code only just finished
  - a slower implementation sustained >40 kpps for all packet sizes
- implementation on StrongARM
  - was able to keep up with “line rate”
- conclusion so far
  - MAPI implemented with function pushed to hardware
  - IXP1200 platform that supports MAPI
  - FFPF is highly suited for implementing MAPI