Implementing Routelets: Virtual Router support for the IXP

Andrew Campbell
Stephen Chou
Michael Kounavis
Vassilis Stachtos
Columbia University
virtual networks on-demand

IXP1200

IP Router
virtual networks on-demand

IXP1200

Virtual Routers
routelet: virtual router

programming environment

state

control plane

data path

binding interface base

metabus

routelet state

spawning controller

composition controller

allocation controller

datapath controller

control unit

input port

forwarding engine

output port
routelet: virtual router

programming environment

state

control plane

data path

binding interface base

metabus

spawning controller
composition controller
allocation controller

control unit
routelet: virtual router

- host processor
- memory units
- StrongARM Core
- microengines

binding interface base
metabus
control unit

routelet state
spawning controller
composition controller
allocation controller
datapath controller

input port
forwarding engine
output port
issues

– limited microstore space
– limited register addresses
– communication protocols are statically programmed
  • run-time binding of components, hard
– need for isolation
our approach

– data path
  • programming data paths
  • data path admission control
  • dynamic binding of data paths
– control plane
  • profiling routelets
  • managing routelet state
– programming environment
programming the data path

- input ports
- forwarding engines
- output ports

- virtual: router specification
- network processor specification
- physical: IXP1200 realization

- components
- symbols
- pipelines

- exit point
- entry point
- global variable
- input argument
- transport modules
profiling datapaths

• virtual router related:
  • a graph connecting input ports, forwarding engines, and output ports
  • ports and engines are modular

• network processor related:
  • components are grouped into pipelines.
  • pipeline components are executed by the same threads sequentially
  • component are augmented with symbols

• IXP1200 related:
  • symbols are associated with IXP1200 properties (e.g., register space, microstore addresses etc)
managing routelet state

routelet state

- SRAM: routing tables, interface information, SDRAM allocations
- SDRAM: forwarding tables, packet buffers, routelet composition information
- Scratchpad: flags, forwarding MIB, performance stats
**datapath admission control**

- **datapath**
  - specified as a set of pipelines; admitted if:
    - sufficient bandwidth
    - sufficient microstore space
    - sufficient contexts available
    - sufficient number of absolute registers

- **pipeline placement:**
  - exhaustive search has $m!$ complexity
  - simple and works OK
register assignment

• component registers are used as
  – pipeline registers (context relative GPRs)
  – input argument registers (context relative GPRs)
  – global variables (absolute GPRs)

• allocations
  – static for context relative registers
  – dynamic for absolute registers
register usage

argument registers (addresses: 9-12)

pipeline #2

shared absolute registers (addresses: 13-15)

component #1

algorithm #1

pipeline registers (addresses: 0-8)

component #2

algorithm #2

component #3

algorithm #3
dynamic binding system

network processor related specification

datapath constructor

IXP1200 related specification

binder

datapath admission control

transport modules (.tmd files)

StrongARM Core

microengines
microassembler extensions

preprocessor → assembler → uof2tmd

.ucp file → .uof file → .tmd file

.import variables
.export functions

.input arguments
.global variables
.entry points
.exit points

.header
dynamic binding

- transport modules:
  - are placed into a microstore
- immed instructions:
  - are filled with input argument values
- absolute registers:
  - are used as global variables
  - assigned during the admission control process
  - their addresses are introduced into the code dynamically
- branch instructions:
  - are added to link exit points with entry points
experiments

8 slow port configuration: 3 VNs supported

4 slow port configuration: 7 VNs supported

<table>
<thead>
<tr>
<th>module</th>
<th>VN</th>
</tr>
</thead>
<tbody>
<tr>
<td>vn demux</td>
<td>-</td>
</tr>
<tr>
<td>capacity arbitrator</td>
<td>-</td>
</tr>
<tr>
<td>verifier</td>
<td>IPv4</td>
</tr>
<tr>
<td>modifier</td>
<td>IPv4</td>
</tr>
<tr>
<td>lookup</td>
<td>IPv4</td>
</tr>
<tr>
<td>lookup Cellular IP</td>
<td>Cellular IP</td>
</tr>
</tbody>
</table>
implementation experiences

• we can support 3 IPv4 routelets in an 8 ‘slow’ port configuration!

• number of VNs heavily depends on the number of physical ports supported
  – For 4 physical ports the number of VNs increases to 7

• microstore space and need for forwarding at line rates
  – the main bottlenecks
implementation experiences

• dynamic binding work well,
  – but we need to halt a microengine to reprogram its microstore

• we want to investigate the programmability of the datapath without disrupting other VNs

• virtual networking overhead:
  – packet demultiplexing,
  – register copy operations
where are we

– data path
  • programming data paths
    – modular cellular IP and vanilla IPv4
  • data path admission control
  • dynamic binding of data path

– control plane
  • profiling routelets
  • managing routelet state

– genesis explorer

– ~18000 lines of C/C++, Java and microcode
where are we going

- currently single node implementation
- working on multiple node configuration
- multiple virtual networks demo August
thanks for listening!
genesis explorer
virtual network traffic

Cellular IP routelet

- **Input Port**
- **Verifier**
- **Forwarding Engine**
  - **Routelet State**
    - Paging Cache, Routing Cache
  - **Paging Update**
  - **Routing Update**

IPv4 routelet

- **Input Port**
- **Forwarding Engine**
- **Output Port**

- **Virtual Network Demultiplexor**
- **Physical Interface**