Modeling NICs with Unicorn

Pravin Shinde, Antoine Kaufmann, Kornilios Kourtis, Timothy Roscoe
Systems Group, ETH Zurich

PLOS/Nov. 3 2013

[we thank Microsoft for their financial support]
We need to talk about NICs

[Shinde et al. HotOS 13]

- increasing complexity/diversity/functionality
  - checksum / segmentation offload, lots of hardware queues, hw filtering, virtualization, direct cache access, ...

- cores are not getting faster, but networks are

- current OSes fail to utilize NIC hardware resources
The Dragonet stack: talking about NICs

idea: model NIC and net stack as dataflow graphs (c.f. x-kernel, click)

Physical Resource Graph

- hw functions
- configuration

Logical Protocol Graph

- protocol state
- packet processing
The Dragonet stack: talking about NICs

idea: model NIC and net stack as dataflow graphs (c.f. x-kernel, click)

Physical Resource Graph

- hw functions
- configuration

Logical Protocol Graph

- protocol state
- packet processing

Embedding

(implements policy)

Embedded graph:

- part of LPG in hw
The Dragonet stack: talking about NICs
idea: model NIC and net stack as dataflow graphs (c.f. x-kernel, click)

Physical Resource Graph
- hw functions
- configuration

Logical Protocol Graph
- protocol state
- packet processing

Embedded graph:
- part of LPG in hw

this talk:

**Unicorn**: a language to talk about NICs
- how do we build the Dragonet graphs?
- how can we use them?
Unicorn: A language to talk about NICs

- a syntax for writing PRGs/LPGs
- a *common* abstract model for:
  - protocol state
  - NIC hardware functionality
Unicorn: A language to talk about NICs

- a syntax for writing PRGs/LPGs

- a *common* abstract model for:
  - protocol state
  - NIC hardware functionality

- basic building blocks:
  - Function nodes (F-nodes)
  - Operators
  - Configuration nodes (C-nodes)
Function nodes
(F-nodes)

- a single computation
- single input
- ports, each with (possibly) multiple outputs
  - when computation is done, one port is activated
  - subsequently, nodes connected to that port are activated
Operator Nodes

- ValidDst == T && ValidSrc == T
- otherwise

- multiple inputs: \( \{ T, F \} \times \text{operands} \)
- can be short-circuited
Operator Nodes

- Multiple inputs: \( \{T, F\} \times \text{operands} \)
- Can be short-circuited
- "representation sugar": double-line edges

\[ \text{ValidDst} = T \land \text{ValidSrc} = T \]
\[ \text{otherwise} \]
LPG example
(rx side)

- receive side
- simplified
- Ethernet/IP/UDP
- named ← UDP packets at port 53
- dhcpd ← UDP packets at port 67
LPG example
(rx side)

▶ packet enters net stack from NIC’s Q1
▶ nodes activated as processing progresses
LPG example
(rx side)

- UDP/*:53 steers packets to named
- UDP/*:67 steers packets to dhcpd
LPG example
(rx side)

- forward: packet processing
- reverse: dependencies
- dependencies “view” used in embedding
LPG example
(rx side)

packet processing

dependencies

- forward: packet processing
- reverse: dependencies
- dependencies “view” used in embedding
- we do not enforce order
  - flexibility in embedding
LPG example
(rx side)

PRGs use the same abstractions

but...
Modeling NIC Configuration

- modern NICS offer rich configuration options
- drastically modify behaviour of NIC
- configuration should be considered in embedding

Configuration nodes (C-Nodes)

- apply configuration value:
  - remove C-node and its edges
  - add a subgraph based on configuration value
PRG configuration example

(i82599: SYN filter + 5-tuple filters)
PRG configuration example
(i82599: SYN filter + 5-tuple filters)
PRG configuration example
(i82599: SYN filter + 5-tuple filters)
PRG configuration example (cont’d)
(i82599: SYN filter + 5-tuple filters)
PRG configuration example (cont’d)
(i82599: SYN filter + 5-tuple filters)

(X out HWIsTCPsyn)
true
false
C5TFilter
queues
default
Q1
Q0
Q2
Q3

(IPV4/UDP,*,*,*,53) → Q2
(IPV4/UDP,*,*,*,67) → Q3
PRG configuration example (cont’d)

(i82599: SYN filter + 5-tuple filters)

\[(\text{IPV4/UDP}, *, *, *, 53) \rightarrow Q2\]
\[(\text{IPV4/UDP}, *, *, *, 67) \rightarrow Q3\]
Node attributes

- F-nodes, operators → processing structure / dependencies
- C-nodes → configuration

annotate nodes with attributes for additional information

- modeling performance
  - annotate each node with perf. metrics (e.g., cpu utilization, latency)
  - reason about network stack performance
- implementation attribute
Implementation attribute for PRG software nodes

LPG part

- hw (i82599) supports:
  - verifying the IPv4 checksum
  - result passed in descriptors
    - needs to be checked in sw
Implementation attribute for PRG software nodes

hw (i82599) supports:
- verifying the IPv4 checksum
- result passed in descriptors
  - needs to be checked in sw

→ PRG software nodes
Embedding a configured PRG
(policy: maximize hw use)
Embedding a configured PRG
(policy: maximize hw use)

- add the NIC queue and its dependencies
  (as much of the PRG as possible)
Embedding a configured PRG
(policy: maximize hw use)

▶ add the NIC queue and its dependencies
  (as much of the PRG as possible)
▶ pick not-embedded LPG node with all its dependencies embedded
  (If LPG is acyclic, all nodes embedded or a node exists)
▶ repeat until all nodes embedded
Embedding a configured PRG
(policy: maximize hw use)

- add the NIC queue and its dependencies
  (as much of the PRG as possible)
- pick not-embedded LPG node with all its dependencies embedded
  (If LPG is acyclic, all nodes embedded or a node exists)
- repeat until all nodes embedded
Embedding a configured PRG
(policy: maximize hw use)

- add the NIC queue and its dependencies (as much of the PRG as possible)
- pick not-embedded LPG node with all its dependencies embedded (If LPG is acyclic, all nodes embedded or a node exists)
- repeat until all nodes embedded
Embedding a configured PRG
(policy: maximize hw use)

- add the NIC queue and its dependencies
  (as much of the PRG as possible)
- pick not-embedded LPG node with all its dependencies embedded
  (If LPG is acyclic, all nodes embedded or a node exists)
- repeat until all nodes embedded
- move PRG sw nodes beyond hw/sw boundary respecting dependencies
Current status:
(WiP)

- Embedding Algs (.hs)
- PRGs (i82599) (.unicorn)
- Unicorn Lang. (.hs)
- Unicorn Model (.hs)
- LPGs (.unicorn)
- Model Implementation (.hs)
- Model Execution (.hs)
- NIC (i82599)
- Intel’s DPDK (.c)

▶ stack responds to pings
Conclusions and Future work

Conclusions

▶ Unicorn: a language to “talk” about NICs
  ▶ used to model NIC functionality and network protocol
  ▶ F-nodes, operators, C-nodes, attributes

Future Work

▶ enrich our models / NICs / policies / embedding algorithms
▶ incremental embedding algorithms → reacting to LPG changes
  - e.g., balancing flows across hw queues
▶ implementation for performance
  - separate fast- and slow-changing LPG parts