When Should the Network Be the Computer?

Dan Ports Jacob Nelson

Microsoft Research

In-Network Computation is a Reality

Reconfigurable network devices are now deployed in the datacenter!

Protocol-Independent
Switch Architectures

FPGA
Network Accelerators

Originally designed to support new network protocols,
these also have powerful systems applications!

What can we do with programmable networks?

What can we do with programmable networks?

- consensus: NOPaxos, NetPaxos, P4xos
- concurrency control: Eris, NOCC
- caching: IncBricks, NetCache, Pegasus
- storage: NetChain, SwitchKV
- query processing: DAIET, SwitchML, Sonata, NetAccel
- applications: key-value stores, DNS, industrial feedback control

…

What can we do with …gram… …rks?

35x
increase in E2E transaction
throughput

45%
latency reduction

- …sensus: NOPaxos, NetPaxos, P4xos
- concurrency control: Eris, NOCC
- caching: IncBricks, NetCache, Pegasus
- …ain, SwitchKV
- …DAIET, Switc…
- applications: key-value stores, D…

…

2 billion key-value
ops/second

88% reduction in servers
required to meet SLO

What can we do with programmable networks?

- consensus: NOPaxos, NetPaxos, P4xos
- concurrency control: Eris, NOCC
- caching: IncBricks, NetCache, Pegasus
- storage: NetChain, SwitchKV
- query processing: DAIET, SwitchML, Sonata, NetAccel
- applications: key-value stores, DNS, industrial feedback control

…

What can we do with programmable networks?

What *should* we do with programmable networks?

Outline

1. **What is this?**
   Hardware Background

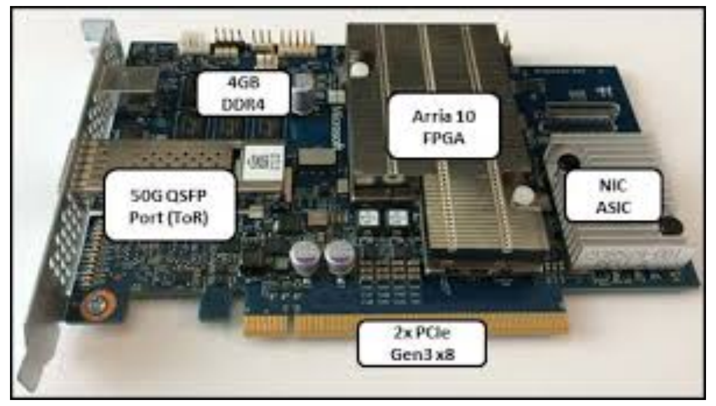
2. How should we use it?
   Principles for In-Network Computation

3. What should we use it for?
   Classifying Application Benefits

4. What's next?
   Open Challenges for In-Network Computation

In-Network Computation Platforms

Programmable switch ASICs
application-specific pipeline stages
line rate processing up to 64 x 200GbE

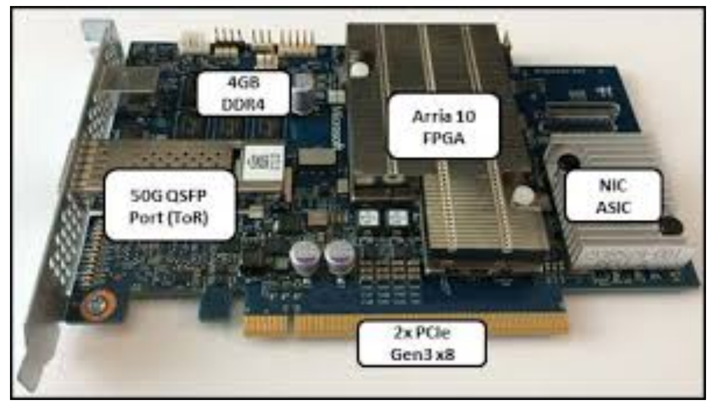


FPGA-based smartNICs
usually 1-2 network links (10-100GbE)

Other architectures:
multicore network processors?

In-Network Computation Platforms

Programmable switch ASICs
application-specific pipeline stages
line rate processing up to 64 x 200GbE



FPGA-based smartNICs
usually 1-2 network links (10-100GbE)

Other architectures:
multicore network processors?

higher throughput

more compute / memory

Deployment Options

In-fabric deployment:

- place computation directly on existing network path
- captures all traffic, has essentially no latency
- complex deployment

End-device deployment:

- accelerator that's connected to the network, not part of it

Outline

1. What is this?
   Hardware Background

2. **How should we use it?**
   Principles for In-Network Computation

3. What should we use it for?
   Classifying Application Benefits

4. What’s next?
   Open Challenges for In-Network Computation

Offload primitives, not applications

Tempting to offload existing application directly into network device

… but it's unlikely to match the resource constraints of the device

Instead, use a narrowly circumscribed in-network primitive

- co-design system with primitive; offload only the common case
- easier development and deployment

Make primitives reusable if possible

Example: Network-Ordered Paxos

Simple primitive: ***network sequencing***
switch adds sequence number to client requests

Application protocol handles dropped messages, replica failure

Offloads only the core functionality (& common case) to network device

Contrast w/ NetPaxos & P4xos,
which move entire application to network devices

[J. Li et al, *Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering*, OSDI'16]

Keep state out of the network

Network devices fail, and don't have (fast) durable storage

End-to-end argument means the application will need to handle reliability anyway

…so keep as many of the complex failure cases in application logic as possible

Minimize network changes

Major challenge is to co-exist with
existing protocols and routing strategies

Related: not all datacenter switches will be (sufficiently) programmable

Useful applications can still be built!

Outline

1. What is this?
   Hardware Background

2. How should we use it?
   Principles for In-Network Computation

3. **What should we use it for?**
   Classifying Application Benefits

4. What's next?
   Open Challenges for In-Network Computation

Classifying applications

Three axes:

1. How many operations per packet ? constant? linear? greater?
2. How much state required? constant? linear? greater?
3. Packet gain (# packets sent / # received) 1? less? greater?

Classifying applications

Three axes:

1. How many operations per packet ? constant? linear? greater?
2. How much state required? constant? linear? greater?
3. Packet gain (# packets sent / # received) 1? less? greater?

Rules of thumb:

- if packet gain ≠ 1, suggests in-switch deployment benefits
- if state-dominant, suggests middle box deployments
- if linear (or greater) operations/state per packet: is it feasible?

Classifying applications

Classifying applications

App	Ops/packet	State/packet	Packet gain

Classifying applications

App	Ops/packet	State/packet	Packet gain
Network sequencing	O(1)	O(1)	|replicas|

Classifying applications

App	Ops/packet	State/packet	Packet gain
Network sequencing	O(1)	O(1)	|replicas|
Virtual networking	O(1)	O(|flow table|)	1

Classifying applications

App	Ops/packet	State/packet	Packet gain
Network sequencing	O(1)	O(1)	|replicas|
Virtual networking	O(1)	O(|flow table|)	1
Replicated storage / caching	O(1)	O(|dataset|)	1
DNN training	O(|packet|)	O(|packet|)	1/|workers|
DNN inference	O(|input|^2)	O(|model|)	1

Case study: load balancing

[X. Jin et al, *NetCache: Balancing key-value stores with fast in-network caching*, SOSP 17]

Case study: load balancing

NetCache [SOSP'17]: caching a few very popular K/V objects in switch gives provable load balancing for skewed workloads

[X. Jin et al, *NetCache: Balancing key-value stores with fast in-network caching*, SOSP 17]

Case study: load balancing

NetCache [SOSP'17]: caching a few very popular K/V objects in switch gives provable load balancing for skewed workloads

State-dominant: required memory = |cached objects|

Model suggests not this is not well suited for switch (!)

[X. Jin et al, *NetCache: Balancing key-value stores with fast in-network caching*, SOSP 17]

Case study: load balancing

NetCache [SOSP'17]: caching a few very popular K/V objects in switch gives provable load balancing for skewed workloads

State-dominant: required memory = |cached objects|

Model suggests not this is not well suited for switch (!)

- limitations on storage, object size are problematic
- these restrictions are worse in production environments

[X. Jin et al, *NetCache: Balancing key-value stores with fast in-network caching*, SOSP 17]

Case study: load balancing

Can we get the same benefits another way?

Alternative: ***replicate*** the most popular objects
and forward read requests to any server with available capacity

Network primitive: switch acts as directory:
tracks location of objects and finding least loaded replica

Result: same load balancing benefits, but
state requirement now proportional to ***metadata*** size (400x reduction)

[J. Li et al, *Pegasus: Load-Aware Selective Replication with an In-Network Coherence Directory*, arXiv, 2018]

Outline

1. What is this?
   Hardware Background

2. How should we use it?
   Principles for In-Network Computation

3. What should we use it for?
   Classifying Application Benefits

4. **What’s next?**
   Open Challenges for In-Network Computation

Open Challenges

- Multitenancy & isolation
- Logical vs wire messages
- Encryption
- Scale & decentralization
- In-device parallelism
- Interoperability

Multitenancy and Isolation

Multitenancy and Isolation

Most systems now assume that only one application is running in any given device

Can we eventually allow multiple applications, potentially from mutually distrusting tenants?

Both security and resource isolation concerns

Could provide isolation either at the compiler level or with virtualization-like hardware features
(cf. FPGA isolation mechanisms, e.g. AmorphOS)

Making Application State Transparent

Impedance mismatch: switches deal with packets, not application-level messages

Most research systems are, e.g., using UDP packets with custom headers for application-specific state

This requires each application to reinvent reliable delivery, concurrency control, etc

Is there a more general solution?

Making Application State Transparent

Worse: what if data is encrypted?

Some hope for solving this question:

- many primitives don't actually operate on message contents
  e.g., network sequencing
- others do only simple operations so
  homomorphic encryption techniques may be possible
  e.g., addition for aggregation operators

Conclusion

Programmable network devices are a powerful new technology!

Need to think of these not as a place to drop in existing applications but to implement new primitives

For the right applications, serious potential gains are possible: line-rate throughput, lower latency, or better resource utilization

These gains can easily pay for the cost + complexity of accelerators