Towards Fully Synchronized (and Programmable) Datacenter Networks

Vishal Shrivastav, Cornell University

28 Jul 2016
University of Cambridge
Outline

• DTP: Datacenter Time Protocol [SIGCOMM’16]
• SHOAL: Synchronized rack scale network stack [Ongoing]
• Packet-level scheduler for datacenters [Future work]
• Conclusion
Globally Synchronized Time via Datacenter Networks

Joint with
Ki Suh Lee, Han Wang and Hakim Weatherspoon (Cornell)
Goals and Motivation

To scalable synchronize clocks in a datacenter with high precision

- Scalable – Entire datacenter
- High precision – bounded precision; e.g. no two clocks differ by more than hundreds of nanoseconds

Capability essential for network and distributed applications

- Networks – One-way delay, packet-level scheduling, etc
- Distributed systems – consensus, snapshots, event ordering, etc
Problem: Clock synchronization is non-trivial

- Precision: difference between any two clocks
- Typical clock offset synchronization
  - Offset
Problem: Clock synchronization is non-trivial

- Precision: difference between any two clocks
- Typical clock offset synchronization
  - Offset = \( \frac{(t_1 - t_0) - (t_3 - t_2)}{2} \approx \frac{(d + \text{offset}) - (d - \text{offset})}{2} \)

\[ d = \text{delay} \]

GPS Atomic clock

Time master

\[ t_0 \]

\[ t_1 \]

\[ t_2 \]

\[ t_3 \]
Problem: Clock synchronization is non-trivial

• Precision: difference between any two clocks
• Problems affecting precision
  – Reading remote clock: timestamps, network stack, network jitter
  – Resynchronization frequency
Problem: Clock synchronization is non-trivial

- Synchronization Protocols

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Scalability</th>
<th>Overhead</th>
<th>Extra Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTP</td>
<td>us</td>
<td>Good</td>
<td>Moderate</td>
<td>None</td>
</tr>
<tr>
<td>PTP</td>
<td>sub-us</td>
<td>Good</td>
<td>Moderate</td>
<td>PTP-enabled devices</td>
</tr>
<tr>
<td>GPS</td>
<td>ns</td>
<td>Bad</td>
<td>None</td>
<td>Timing signal receivers, cables</td>
</tr>
</tbody>
</table>
Solution: Use PHY to synchronize clocks

- Protocol in the PHY
  - Not subject to changes in network conditions
  - No protocol stack overhead
  - No network overhead
  - Scalable: peer-to-peer and decentralized
Datacenter Time Protocol (DTP)

• 10 Gigabit Ethernet
  – Idle Characters (/I/) and Control blocks (/E/)

  – Standard requires at least 12 idle characters /I/ between pkts
    • i.e. At least one 64-bit Control Block /E/ between pkts

  – Idle characters / control blocks sent even if no packets to send

  – DTP overwrites idle characters (control block) to send protocol messages

DTP does not effect standard at all
DTP: System Architecture

Application
Transport
Network
Data Link
Physical

Me d ia A cc e ss Con trol (MA C)
Re co nciliati on S ublay er (RS)

TX 32bit
RX 32bit

Physical Layer (PHY)

Physical Coding Sublayer (PCS)

Encode
DTP TX
Scrambler
Gearbox

DTP CTRL

Decode
DTP RX
Descrambler
Blocksync

TX 16bit
XSBI 644.53125MHz
RX 16bit

Physical Medium Attachment (PMA)
Physical Medium Dependent (PMD)
DTP: Benefits

Precise and bounded synchronization

• 4 oscillator ticks (25ns) bounded peer-wise synchronization
• 150ns precision synchronization for an entire datacenter
• Free – No network traffic: Use the PHY!

No two clocks differ by more than 150ns in the entire datacenter
DTP: Evaluation

• Compare measured precision of DTP and PTP
  – Measurement and observation period was two days

• PTP: Compare precision between Timeserver and Servers
  – Mellanox NIC (hardware), IBM G8264 Switch, Timekeeper server

• DTP: Compare precision between leaf servers and switches
  – Terasic DE5 FPGA-based development Net board
PTP – Idle Network (No Network Traffic)

Clocks differ by a few hundred nanoseconds
PTP – Medium Loaded Network (4Gbps Traffic)

Clocks differ by *tens of microseconds*
PTP – Heavily Loaded Network (9Gbps Traffic)

Clocks differ by hundreds of microseconds
Clocks \textit{never} differed by more than 4 clock ticks, 25ns \textit{Bounded Precision}
Next steps

Integration with OSNT (Open Source Network Tester)
(with Jong Hun Han, Gianni Antichi and Andrew Moore)

- Integration on NetFPGA-SUME
- Larger user base for DTP
- Integrated hardware platform to test novel networking functionalities
Topics covered in paper

• External Synchronization
• Fault tolerance
• Incremental deployment
• Modifications for 40G and 100G Ethernet links
Outline

• DTP: Datacenter Time Protocol [SIGCOMM’16]
• SHOAL: Synchronized rack scale network stack [Ongoing]
• Packet-level scheduler for datacenters [Future work]
• Conclusion
Efficient network stack for rack scale fabric

Joint with
Hitesh Ballani, Paolo Costa (Microsoft Research Cambridge)
Asaf Valadarsky (The Hebrew University of Jerusalem)
Ki Suh Lee, Han Wang, Hakim Weatherspoon (Cornell)
Rack scale Architecture

Motivation: Increase Performance per $\$

- **Standard rack**
  - 40 servers
  - 80 CPUs

- **Rack scale computer**
  - e.g. Boston Viridis
  - Server = Calxeda SoC
  - 900 CPUs

*Pictures borrowed from https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/legtchenko*
Network Fabric for Rack scale Architecture

- **ToR switch**
  - 1000 ports
  - Full bisection bandwidth: 10 Tbps
  - Cost: $$$$$

- **Clos network**
  - >1000 ports
  - High power and cost

- **Direct connect**
  - Oversubscribed
  - Multi-hop routing
  - Higher, less predictable latency
  - Static topology

*R2C2 [Costa et al, SIGCOMM’15]*

*Pictures borrowed from https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/legtchenko*
Network Fabric for Rack scale Architecture

• **XFabric** [Legtchenko et al, NSDI’16]

Packet switching over a physical circuit-switched network

*Pictures borrowed from https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/legtchenko*
SHOAL

• System runs according to a static schedule
  
  \[
  \begin{array}{cccc}
    & S0 & S1 & S2 & S3 \\
    T0 & 1 & 2 & 3 & 0 \\
    T1 & 2 & 3 & 0 & 1 \\
    T2 & 3 & 0 & 1 & 2 \\
  \end{array}
  \]

  Circuit configuration for timeslot 0
  -- a permutation of nodes

  Schedule for server 0 -- sends to every other node once every epoch

• 2-hop routing (via 1 intermediate node)
  – Approximates Valiant load balancing

  Made feasible by tight synchronization between the nodes
Outline

• DTP: Datacenter Time Protocol [SIGCOMM’16]
• SHOAL: Synchronized rack scale network stack [Ongoing]
• Packet-level scheduler for datacenters [Future work]
• Conclusion
Fully distributed packet-level scheduler for datacenters

Joint with
Gianni Antichi (University of Cambridge)
Hakim Weatherspoon (Cornell)
and maybe more...
Goals for the Scheduler

- Ensures “Zero” queuing
- No network jitter
- Provides strong latency and throughput guarantees
- Highly scalable
- Highly Fault tolerant

..... even under high load and Incast
Proposal

• Discrete timeslots (length of smallest packet)

• Load balancing and arbitration in dataplane
  – Fast control feedback loop in the switches
  – Implementation using P4
  – On queuing, arbitration feedback sent to the end-hosts

• End-hosts send (or not send) based on feedback

_Tight synchronization to ensure efficient arbitration_
Outline

• DTP: Datacenter Time Protocol [SIGCOMM’16]
• SHOAL: Synchronized rack scale network stack [Ongoing]
• Packet-level scheduler for datacenters [Future work]
• Conclusion
Conclusion

• DTP - a clock synchronization protocol in PHY
  – Highly scalable
  – Bounded nanosecond level precision
  – No network overhead
  – Needs hardware modification (just like PTP)

• Can use tight synchronization (via DTP) to build
  – An efficient network stack for rack-scale fabric
  – Fully-distributed packet-level scheduler for datacenters
Thank you!
State-of-the-art

Load Balancing

- ECMP
  - choose among equal cost paths by taking a hash of the 5-tuple
  - stateless, simple
  - Hash collisions, bad with asymmetry and link failures

- CONGA [Alizadeh et al, SIGCOMM’14]
  - Global congestion knowledge
  - Control feedback loop in the dataplane

Does not solve Incast problem
State-of-the-art

“Zero” Queuing

- Fastpass [Perry et al, SIGCOMM’14]
  - centralized arbiter in software
  - End-hosts send demands, arbiter allocates timeslots and path
  - Uses PTP for synchronization – using DTP can improve performance

Centralized - not very scalable, single point of failure

Proactive - needs to communicate with arbiter everytime an app calls `send()`

Coarse grained - timeslots equal to MTU sized pkt