Advanced Computer Networks
263-3501-00

Datacenter Transport

Patrick Stuedi
Spring Semester 2013
High-Performance Networking: Last week

- Network topologies
  - Mesh, Torus, Tree

- Data link layer and switching fabric
  - Ethernet, Infiniband
  - Lossy and lossless data link layer

- Addressing, Configuration, Routing
  - MAC and IP address configuration
  - ARP

- Transport layer
  - Datacenter TCP
  - TCP offloading
  - Incast Problem

- End-host interfaces
  - RDMA
High-Performance Networking:
Last week

- Network topologies
  - Mesh, Torus, Tree

- Data link layer and switching fabric
  - Ethernet, Infiniband
  - Lossy and lossless data link layer

- Addressing, Configuration, Routing
  - MAC and IP address configuration
  - ARP

- Transport layer
  - Datacenter TCP
  - TCP offloading
  - Incast Problem

- End-host interfaces
  - RDMA
Why Ethernet is hard to scale

- Ethernet Spanning Tree protocol is not designed for large datacenters
  - Does not leverage multipath if available
  - Spanning tree allows only one path between any src/dst pair
    - Limits bandwidth
    - Low reliability

- Packet floods
  - Switches discover hosts and creating routing entry
  - Switches must forget table entries periodically to support host mobility
  - Switch receiving a packet for unknown host will flood the packet on all ports

- Switch state
  - Can become large if the entire datacenter is one layer-2 network
Scaling Ethernet in Portland

- Portland:
  - Research at Data Center Network Group, UC San Diego

- Single layer-2 network with up to 100K ports
  - 1M endpoints (through virtualization)
  - VM migration while keeping IP address

- Minimize amount of switch state

- Towards zero-configuration
  - No subnet or hierarchical IP addresses, dhcp etc.

- First-class support for multi-path routing

- Uses a Fat-Tree Topology as shown in slide 32
Scaling Ethernet in Portland: Key principles

- **Host IP address:**
  - Node identifier, fixed even after VM migration

- **Pseudo MAC:** node location
  - In-network rewriting of MAC address
  - PMAC address changes depending on location of host
  - PMAC address encodes location of host
  - PMAC used to do routing

- **Fabric Manager:** centralized lookup service
  - Maintains IP->PMAC mappings
  - Replaces ARP
  - Lookup is unicast instead of broadcast
PMAC format

PMAC: pod.position.port.vmid
Autoconfiguration of PMACs at Switches

- Location Discovery Messages (LDM) exchanged between neighboring switches
  - Discovery at bootup
- LDM protocol helps switches to learn
  - Tree level (edge, aggregation, core)
  - Pod number
  - Position number
- Configuration does not involve broadcast
I am an edge switch (ES) if I receive LDM message on my uplink only

Aggregation switches (AS) get messages from ES as well as from unknown switches

Core switches get messages on all ports from AS
Autoconfiguration: Position number

- Run agreement protocol: propose random position number
- Use aggregation switches to ensure no two edge switches are assigned the same position number
Autoconfiguration: Pod number

- Use directory service to get the pod number
Switch communicates constructed PMAC for hosts to Fabric manager
Portland Routing

- Since PMAC encode the location of a host each switch can, based on PMAC, decide to
  - Route packet to aggregation switch if in the same POD
  - Route upwards if in a different POD

- Multipath through ECMP
  - Equal-cost multi-path routing
  - Loadbalancing: hash flows/packets to paths
Edge switches Intercept ARP requests, contacts fabric manager
- ARP reply contains PMAC
- Routing based on PMAC
VL2: A Scalable and Flexible Data Center Architecture

- Alternative datacenter architecture to Portland
  - Portland is network-centric: intelligence in switches
  - VL2 is end server-centric: intelligence in servers
  - Microsoft Research

Key ideas of VL2:

- Each server has two IP addresses
  - AA: Application address (location independent)
  - LA: Location dependent address

- Each switch has a LA

- VL2 agent on each server intercepting ARP
  - Mapping of AA to LA

- Routing:
  - Lookup of LA of switch which serves the dst node
  - Tunnel application packet to switch using the LA of the switch

VL2 shares concepts with Portland: Fat-Tree, Directory Server, ECMP
Each AA is has an associated LA (LA of ToR Switch), mapping stored in VL2 Directory

Routing: Server traps packet and encapsulates it with the LA address of the ToR of the destination

Load balancing: Source ToR encapsulates packet to a randomly chosen intermediate switch
High-Performance Networking: Where are we

- Network topologies
  - Mesh, Torus, Tree

- Data link layer and switching fabric
  - Ethernet, Infiniband
  - Lossy and lossless data link layer

- Addressing, Configuration, Routing
  - MAC and IP address configuration
  - ARP

- Transport layer
  - Datacenter TCP
  - TCP offloading
  - Incast Problem
  - Multipath TCP

- End-host interfaces
  - RDMA
What's Special about Data Center Transport?

- Application requirements (particularly, low latency)
How Does Search Work?

- Time is money
  - Strict deadlines (SLAs)
- Missed deadlines
  - Lower quality
- Many requests per Query
  - Tail latency matters

Worker Nodes

Deadline = 250ms
Deadline = 50ms
Deadline = 10ms

Partition/Aggregate Application Structure

source: stanford CS244
What's Special about Data Center Transport?

- Application requirements (particularly, low latency)
- Particular Traffic Patterns (partition/aggregate)
Datacenter Workloads

- Partition/Aggregate (Query) → Bursty, Delay-sensitive
- Short messages (50KB-1MB) (Coordination, Control state) → Delay-sensitive
- Large Flows (1MB-100MB) (Data) → Throughput-sensitive
Flow Size

> 65% of Flows are < 1MB

> 95% of Bytes from Flows > 1MB
What's Special about Data Center Transport?

- Application requirements (particularly, low latency)
- Particular Traffic Patterns (partition/aggregate)
- Commodity Switches – shallow buffers (e.g., 4MB buffer shared between 48 1Gbit/s ports and two 10Gbit/s ports)
TCP in Datacenters

Problems and Goals of TCP in Datacenters

- **Incast:**
  - A request/response pattern in DC application leads to queue overflows and TCP retransmissions

- **Low network latencies**
  - RTT measurements not fine granular enough
  - RTT determines TCP retransmission timeout

- **Multipath**
  - Want to leverage multi-path capabilities of the network topology
The Incast Problem

Worker 1

Worker 2

Worker 3

Worker 4

Synchronized fan-in congestion: Caused by Partition/Aggregate.

Aggregator

TCP timeout

RTOmin = 300 ms
TCP Incast (2)

- Incast event measured in a production environment
  - Request forwarded in over 0.8 ms (800 microseconds)
  - All but one response returning in 12.4 ms
  - Retransmission after RTO: 300 ms
Queue Buildup

Large flows buildup queues: Increase latency for short flows.
Approach 1: Decrease of RTO

- Roundtrip timeout typically set based on measured RTT+X, with RTO >= RTO\_min

- Problem:
  - Most Linux TCP implementations do not measure RTT as fine granular as needed for datacenters
  - RTO\_min typically too large

<table>
<thead>
<tr>
<th>Scenario</th>
<th>RTT</th>
<th>OS</th>
<th>TCP RTO_min</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAN</td>
<td>100ms</td>
<td>Linux</td>
<td>200ms</td>
</tr>
<tr>
<td>Datacenter</td>
<td>&lt;1ms</td>
<td>BSD</td>
<td>200ms</td>
</tr>
<tr>
<td>SAN</td>
<td>&lt;0.1ms</td>
<td>Solaris</td>
<td>400ms</td>
</tr>
</tbody>
</table>

- Idea:
  - Reduce RTO\_min
  - Measure RTT using high-resolution timers in us granularity
Lowering RTO helps

- High throughput for up to 47 servers

- Microsecond TCP + no minRTO
- 1ms minRTO
- Unmodified TCP (200ms minRTO)
Approach 2: Datacenter TCP

- Low RTO not enough: Does not avoid queue buildup
- Datacenter TCP (DCTCP):
  - Mark packets in switches using Explicit Congestion Notification (ECN) if they experience congestion
  - Scale the TCP window down proportionally to the number of packets with ECN bit set

<table>
<thead>
<tr>
<th>ECN Marks</th>
<th>TCP</th>
<th>DCTCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 1 1 0 1 1 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 40%</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 5%</td>
</tr>
</tbody>
</table>

**Default TCP**

**Data Center TCP**
DCTCP Algorithm

- **Switch-side:**
  - Mark packets if Queue length > K using ECN bit

- **Receiver-side:**
  - Echo bit back to sender with delayed ACKs

- **Sender-side:**
  - Maintain running average 'a' of fraction of packets marked (value of 'a' between 0 and 1)
    - 'a' close to 0 means low congestion
    - 'a' close to 1 means high congestion
  - Adaptive window decrease/increase: `cwnd = cwnd x (1-a/2)`
DTCP achieves full throughput (not shown in Figure) while taking up a very small footprint in the switch.
Mutipath TCP (MPTCP)

- Traditional Topologies are Tree-based
  - Poor performance
  - Not fault tolerant

- Shift towards multipath topologies
  - Portland, VL2

- Idea: Leverage multiple path at the TCP level to
  - Improve bandwidth (aggregation)
  - Improve fairness
  - Improve robustness
Multipath TCP (MPTCP): Bandwidth aggregation

- Fat-Tree provides full bisection bandwidth, why do we need multiple paths anyway?
  - Full bisection bandwidth only if perfectly load-balanced traffic
  - Hosts may have multiple interfaces

- What about ECMP load balancing?
  - Static: hashing of flows to paths at the switch level
    - Does not take into account flow size, network utilization, etc.
  - Leads to hotspots and unfair bandwidth allocation

- MPTCP establishes multiple subflows on different paths
  - With many random paths, MPTCP will find at least one good unloaded path and moves most of its traffic on that path
MTCP: Connection Management
MTCP: Connection Management
MTCP: Connection Management

Enable MPTCP if SYN has MP_CAPABLE
MTCP: Connection Management

Enable MPTCP if SYN has MP_CAPABLE

ENABLED
MTCP: Connection Management

client learns about additional interfaces of server

Enable MPTCP if SYN/ACK has MP_CAPABLE

Enable MPTCP if SYN has MP_CAPABLE

SYN/ACK MP_CAPABLE

ENABLED
MTCP: Connection Management

Enable MPTCP if SYN/ACK has MP_CAPABLE

Enable MPTCP if SYN has MP_CAPABLE

Works in data centers, problem when using MPTCP across the Internet: 6% of access networks remove unknown options
MTCP: Connection Management

Enable MPTCP if SYN/ACK has MP_CAPABLE

ENABLED

Enable MPTCP if SYN has MP_CAPABLE

ENABLED
MTCP: Connection Management
MTCP: Connection Management

Subflows can be between different interfaces or between the same pair of IP addresses but different ports.
MTCP: Connection Management
MTCP: Connection Management
MPTCP relies on ECMP to hash different subflows to different paths
MTCP: Sending Data

MPTCP stripes TCP across the subflows

Additional TCP options allow the receiver to reconstruct the received data in the original order
MTCP: Sending Data

When used over the Internet, middleboxes may drop ACKs of unseen data packets.
MTCP Performance

FatTree, 128 Nodes

FatTree, 8192 Nodes

Throughput (% of optimal)

No. of MPTCP Subflows

TCP 2 3 4 5 6 7 8

FLOW
PKT
High-Performance Networking: Where are we

- Network topologies
  - Mesh, Torus, Tree

- Data link layer and switching fabric
  - Ethernet, Infiniband
  - Lossy and lossless data link layer

- Addressing, Configuration, Routing
  - MAC and IP address configuration
  - ARP

- Transport layer
  - Datacenter TCP
  - TCP offloading
  - Incast Problem
  - Multipath TCP

- End-host interfaces
  - RDMA
Network latencies in Data centers

- Factors that contribute to latency in TCP datacenters
  - Delay: cost of a single traversal of the component
  - RTT: total cost in a round-trip traversing 5 switches in each direction
- OS overhead per packet exchanged between two hosts attached to the same switch: $\frac{(2\times15)}{(2\times2.5+2\times15+10)}=66\%$ (!!!)
Packet Processing Overhead

- **Sending-side:**
  - Data is copied from the application buffer into a socket buffer
  - Data is DMA copied into NIC buffer

- **Receiver side:**
  - Data is DMA copied from NIC buffer into socket buffer
  - Data is copied into application buffer
  - Application is scheduled (context switching)
Throughput and CPU load at 1Gbit/s and 10Gbit/s

- Throughput limited because of high CPU load

- RX side typically more CPU intensive because highly asynchronous
TCP Offloading

- What is TCP offloading
  - Moving IP and TCP processing to the Network Interface (NIC)

- Main justification for TCP offloading
  - Reduction of host CPU cycles for protocol header processing, checksumming
  - Fewer CPU interrupts
  - Fewer bytes copied over the memory bus
  - Potential to offload expensive features such as encryption
TCP Offload Engines (TOEs)

User mode
Application

Kernel
Sockets
TCP
IP
NIC driver

Ethernet
Network IC

becomes

User mode
Application

Kernel
Sockets
NIC driver

TCP
IP
Ethernet
Network IC

TOE
Problems of TCP offloading

- Moore’s Law worked against “smart” NICs
  - CPU's used to get faster
- Now many cores, cores don't get faster
  - Network processing is hard to parallelize
- TCP/IP headers don’t take many CPU cycles
- TOEs impose complex interfaces
  - Protocol between TOE & CPU can be worse than TCP
- Connection management overhead
  - For short connections, overwhelms any savings
Where TCP offload helps

- Sweet spot for TCP offload might be apps with:
  - Very high bandwidth
  - Relatively low end-to-end latency network paths
  - Long connection durations
  - Relatively few connections

- Typical examples of these might be:
  - Storage-server access
  - Cluster interconnects
User-level networking: Remove OS from the data path

- Transport offloading is not enough!
  - Still have system call overhead, context switch, memory copying

- U-Net:
  - Eicken, Basu, Buch, Vogels, Cornell University, 1995
  - Virtual network interface that allows applications to send and receive messages without operating system intervention
  - Move all buffer management and packet processing to user-space (zero-copy)

- Traditional networking architecture
  - Kernel controls the network
  - All communication via kernel

- U-Net architecture:
  - Application access network directly via MUX
  - Kernel involved only in connection setup
U-Net Building Blocks

- **End points**
  - application’s handle into the network

- **Buffer area**
  - hold message data for sending or buffer space for receiving

- **Message queues**
  - hold *descriptors* pointing to buffer area
U-Net communication

- Initialization:
  - Create one or more endpoints
  - Register user buffers with endpoints and associated them with a tag

- Sending
  - Composes the data in the endpoint buffer area
  - Push a descriptor for the message onto the send queue
  - NIC transmits the message after marking it with the appropriate message tag.

- Receiving:
  - Push a message descriptor with pointers to the buffers onto the receive queue.
  - Incoming messages get de-multiplexed based on the message tag
  - Data is placed within the target buffer of the application by the NIC
History of User-Level Networking

- U-Net one of the first (if not the first) system to propose OS-bypassing

- Other early works
  - SHRIMP: Virtual Memory Mapped Interfaces, IEEE Micro, 1995
  - “Separating Data and Control Transfer in Distributed Operating Systems”, Thekkath et. al., ASPLOS'94

- Efforts of U-Net eventually resulted in the Virtual Interface Architecture (VIA)
  - Specification jointly proposed by Compaq, Intel and Microsoft, 1997

- VIA architecture has led to the implementation of various high performance networking stacks: Infiniband, iWARP, Roce:
  - Commonly referred to as RDMA network stacks
  - RDMA = Remote Direct Memory Access
RDMA Architecture

- **Traditional socket interface** involves kernel
- RDMA interface involves kernel only on **control path**, but access the RDMA capable NIC (rNIC) directly from user space on the **data path**
- Dedicated **verbs interface** used for RDMA, instead of traditional socket interface
RDMA Queue Pairs (QPs)

- Applications use 'verbs' interface to
  - Register memory:
    - Operating system will make sure the memory is pinned and accessible by DMA
  - Create a queue pair (QP)
    - send/recv queue
  - Create a completion queue (CQ)
    - RNIC puts a new completion-queue element into the CQ after an operation has completed
  - Send/Receive data
    - Place a work-request element (WQE) into the send or recv queue
    - WQE points to user buffer and defines the type of the operation (e.g., send, recv, ..)
RDMA Queue Pairs (QPs)

- Applications use 'verbs' interface to
  - Register memory:
    - Operating system will make sure the memory is pinned and accessible by DMA
  - Create a queue pair (QP)
    - send/recv queue
  - Create a completion queue (CQ)
    - RNIC puts a new completion-queue element into the CQ after an operation has completed
- Send/Receive data
  - Place a work-request element (WQE) into the send or recv queue
  - WQE points to user buffer and defines the type of the operation (e.g., send, recv, ..)

This is much like U-NET
RDMA operations

- **Send/Receive:**
  - Two-sided operation: data exchange naturally involves both ends of the communication channel
  - Each send operation must have a matching receive operation
  - Send WR specifies where the data should be taken from
  - Receive WR on the remote machine specifies where the inbound data is to be placed

- **RDMA (Remote Direct Memory Access):**
  - Two independent operations: RDMA Read and RDMA Write
  - Only the application issuing the operation is actively involved in the data transfer
  - An RDMA Write not only specifies where the data should be taken from, but also where it is to be placed (remotely)
  - An RDMA Read requires a buffer advertisement prior to data exchange
Example: RDMA Send/Recv (1)

- Sender and receiver have created their Qps and Cqs
- Sender has registered a buffer for sending
- Receiver has registered a buffer for receiving
Example: RDMA Send/Recv (2)

- Receiver places a WQE into its receive queue
- Sender places a WQE into its send queue
Data is transferred between the hosts
  - Involves two DMA transfers, one at the sender and one at the receiver
Example: RDMA Send/Recv (4)

- After operation has finished, a CQE is placed into the completion queue of the sender
RDMA implementations

- Infiniband
  - Compaq, HP, IBM, Intel Microsoft and Sun Microsystems
  - Provides RDMA semantics
  - First spec released 2000
  - Based on point-to-point switched fabric
  - Designed from ground up (has its own physical layer, switches, NICs, etc)

- IWARP (Internet Wide Area RDMA Protocol)
  - RDMA semantics implemented over offloaded TCP/IP
  - Requires custom NICs, but uses Ethernet

- RoCE
  - RDMA semantics implemented directly over Ethernet

- All of those implementation can be programmed through the verbs interface
Typical CPU loads for three network stack implementations
Datacenter Networks: Many more things we didn't discuss

- Green Networks
  - How to reduce the power consumption of large data center network installations
    - E.g., turn off network elements when not needed
- QoS and isolation
  - How to give QoS guarantees to application running in a shared data center
  - How to isolate network used by one application from traffic of other applications
- TCP offloading
  - Implementing entire TCP state machine in hardware
- ...many more
References

- “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network”, Sigcomm 2009
- “Data Center TCP (DCTCP)”, Sigcomm 2010
- “Improving Datacenter Performance and Robustness with Multipath TCP”, Sigcom 2011
- “How Hard Can it Be? Designing and Implementing a Deployable Multipath TCP”, NSDI 2012
- “Design, Implementation and Evaluation of Congestion Control for Multipath Tcp”
- VL2: A Scalable and Flexible Data Center Network, Sigcomm 2009
- “It's time for low latency”, HotOS 2011