CS 4450
Network Fabric

Based on:


3. Benson’s original slide deck from IMC10.
Performance of distributed systems depends heavily on the datacenter interconnect
Example - MapReduce

How MapReduce Works?

Map() Shuffle Reduce() -

Source: https://blog.sqlauthority.com/2013/10/09/big-data-buzz-words-what-is-mapreduce-day-7-of-21/
Evaluation Metrics for Datacenter Topologies

• Diameter – max #hops between any 2 nodes
  • Worst case latency

• Bisection Width – min #links cut to partition network into 2 equal halves
  • Fault tolerance

• Bisection Bandwidth – min bandwidth between any 2 equal halves of the network
  • Bottleneck

• Oversubscription – ratio of worst-case achievable aggregate bandwidth between end-hosts to total bisection bandwidth
Legacy Topologies

Ring  Mesh  Star  Fully Connected

Line  Tree  Bus

3-Tier Architecture

Source: CS 5413, Hakim Weatherspoon, Cornell University
Big-Switch Architecture

Figure 2: A traditional 2Tbps four-post cluster (2004). Top of Rack (ToR) switches serving 40 1G-connected servers were connected via 1G links to four 512 1G port Cluster Routers (CRs) connected with 10G sidelinks.

Source: Jupiter Rising, Google
Goals for Datacenter Networks (circa 2008)

• 1:1 oversubscription ratio – all hosts can communicate with arbitrary other hosts at full bandwidth of their network interface
  • Google’s Four-Post CRs offered only about 100Mbps
• Low cost – cheap off-the-shelf switches

Source: A Scalable, Commodity Data Center Network Architecture. Al-Fares et al.
Fat-Trees

Advantages of Fat-Tree Design

• Increased throughput between racks
• Low cost because of commodity switches
• Increased redundancy
Software Control

• Custom control plane
  • Existing protocols did not support multipath, equal-cost forwarding
  • Lack of high quality open source routing stacks
  • Protocol overhead of running broadcast-based algorithms on such large scale
  • Easier network manageability

• Treat the network as a single fabric with O(10,000) ports

• Anticipated some of the principles of Software Defined Networking
Issues – Congestion

High congestion as utilization approached 25%
• Bursty flows
• Limited buffer on commodity switches
• Intentional oversubscription for cost saving
• Imperfect flow hashing
Congestion – Solutions

• Configure switch hardware schedulers to drop packets based on QoS
• Tune host congestion window
• Link-level pause reduces over-running oversubscribed links
• Explicit Congestion Notification
• Provision bandwidth on-the-fly by repopulating
• Dynamic buffer sharing on merchant silicon to absorb bursts
• Carefully configure switch hashing to support ECMP load balancing
Insights Gained

• 75% of traffic stays within a rack (Clouds)
  • Applications are not uniformly placed
• Half packets are small (< 200B)
  • Keep alive integral in application design
• At most 25% of core links highly utilized
  • Effective routing algorithm to reduce utilization
  • Load balance across paths and migrate VMs
• Questioned popular assumptions
  • Do we need more bisection? No
  • Is centralization feasible? Yes
A Clean Slate 4D Approach to Network Control and Management
Complex Architecture of Networks

- Path-computation logic is bundled with packet handling.
  - In IP networks, path-computation logic is governed by distributed protocols such as OSPF, IS-IS, and EIGRP.
  - In Ethernet networks, path-computation logic is embedded in the Spanning Tree Protocol.
- Network-level objectives can be different from best-effort packet delivery.
- Incremental changes of the control-plane only leads to complex and fragile networks.
A Clean Slate Approach

- Redesign the network via the 4D approach: decision, dissemination, discovery, and data.
A Clean Slate Approach

- Redesign the network via the 4D approach: decision, dissemination, discovery, and data.
Advantages of the 4D Architecture

- Separate networking logic from distributed system issues.
- Higher robustness.
- Better security.
- Accommodating heterogeneity.
- Enabling of innovation and network evolution.
Challenges of the 4D Architecture

- Complexity apocalypse.
- Stability failures.
- Scalability problems.
- Response time.
- Security vulnerabilities.
Next Steps

- Decision plane:
  - Satisfying network-level objectives.
  - Coordination between decision elements.
  - Hierarchy in the decision plane.

- Dissemination plane.

- Data plane.
Software-Defined Networking

- Separating the control logic (control plane) from the forwarding mechanism (data plane).

(a) Distributed protocol
(b) Software-defined networking
Software-Defined Networking

• Separate control plane and forwarding plane
• Common, open, agnostic vendor-agnostic interface
  • Control forwarding devices across different hardware/software devices
• OpenFlow
  • Proliferation of header fields complicates protocol
  • Multiple stages of rule tables
  • Difficult to scale due to lack of flexibility
• Goal: *tell the switch how to operate*
Abstract Forwarding Model

• Arriving packets are handled by the parser
  • No assumptions about headers’ intent
• Header fields passed to the match-action tables
  • Ingress - modify packet, determine egress port and proper queue
  • Egress – modify packet, prepare for operations (e.g. multicast)
P4: Programming Protocol-Independent Packet Processors
P4: A Solution

• Raise the level of abstraction for programming the network
• General interface between the controller and switches
• Reconfigurability
  • Controller can redefine the packet parsing and processing
• Protocol Independence
  • Switch is decoupled from specific packet formats
  • Controller specified packet parser and match-action tables
• Target Independence
  • Compiler handles switch capabilities, not the controller programmer
Next Steps

- Control plane programming: NOX, Gude et al., 2008.
- SD-Internet Exchange Points: iSDX, Gupta et al., 2016
- Separating the edge and fabric control: Fabric, Casado et al., 2012.
Datacenter Congestion Control: DCTCP
TCP in the Datacenter context

- TCP in the context of Datacenters is not optimal

- TCP’s demands on limited buffer space in data center switches are too high

- Queues at switches become too congested, and impacts performance of “foreground” traffic
What we need:

• Low latency for short flows and high burst tolerance
  • many applications use a Partition/Aggregate workflow pattern
  • requirements for low latency directly impact the quality of application results
• High utilization for large flows
  • Need to continuously update internal data structures in applications and the data inside these data structures
Main Insights and Contributions
Partition/Aggregate Design pattern

• Used in many large scale web-applications
• Latency is key metric
• Network delays play a big role in application design
• Meeting deadlines with TCP is very difficult, so some developers resort to a
Workload Characterization

• Three types of workloads presented:
  • Query traffic
  • Short message traffic – coordinates cluster activities
  • Background traffic – ingests and organizes data
Query Traffic

• Follow Partition/Aggregate pattern
• consists of very short, latency-critical flows
• High Level Aggregators (HLA) partition queries to a large number of Mid Level Aggregators (MLA) and workers
• Servers act as an aggregator for some queries while also acting as worker for other queries
Background Traffic

- Runs concurrent to query traffic
- Consists of large and small flows
- Most flows are small, but most bytes are a part of large flows
- Update flows – update data to workers
- Short message flows – update control state of workers
Flow Concurrency and Size

- Median number of concurrent flows: 36
- In summary: large flows, small flows, and bursty query flows coexist in a datacenter

Figure 5: Distribution of number of concurrent connections.
Incast

- Can occur even if flow sizes are small
- A response that causes incast will usually miss aggregator deadline
- Current solution: jittering
Queue Buildup

• Caused by long-lived greedy TCP flows and when long/short flows both traverse same queue
• packet loss on short flows cause incast
• short flows experience increased latency
• Since latency is caused by queueing, the only solution is to shrink queues
Buffer Pressure

- buffer space is a shared resource
- Results in packet loss and timeouts
Multipath TCP (MPTCP)

- Enables topologies that single path TCP cannot utilize
- Searches for multiple paths simultaneously
- Links congestion response of subflows on different paths to move traffic away from congestion
MPTCP Cont’d.

• Extends TCP so that a single connection can be striped across multiple paths
• Can explicitly move traffic off more congested paths and place it on less congested ones
• Chooses paths randomly
• Additional TCP operations reconstruct the received data
pFabric

• Transport Protocol based on the idea that flow scheduling should be decoupled from rate control
• Goal: Design the simplest transport protocol that provides near optimal Flow Completion Time (FCT)
• Downsides: Requires specialized hardware, potentially expensive to deploy in real networks
pFabric Implementation

• Switch Design
  • Priority scheduling: if a port is idle, packet with the highest priority buffered is dequeued and sent out
  • Priority Dropping: drops lowest priority packet in buffer to make room

• Data structures
  • Queue of actual packets
  • Queue of packet metadata
Packet Scheduling and Rate Control

• Starvation Prevention: Dequeue earliest packet from the flow that has highest priority packet in the queue

• Rate Control: because of scheduling algorithm, need for rate control is minimal

• Exception: When a packet traverses multiple hops only to be dropped at a downstream link
Rate Control Policy

• Flows start at line rate
• Use SACKS, additive increase for every ACK
• Packet drops are detected by timeouts
• If a certain number of timeouts occur, flow enters “probe mode” where it periodically retransmits minimum sized packets and re-enters slow start after receiving an ACK
pHost

• Similar design principles as pFabric, but aims to rely only on commodity network hardware

• Allows programmers to customize the packet scheduling algorithm to achieve different policy goals

• Especially useful when datacenters are shared by multiple users/applications

• Paper is co-authored by Rachit!
Basic Transport Mechanism

• Built around a host-based scheduling mechanism
• Uses requests-to-send (RTS), per-packet token assignment, and receiver-based selection of pending flows
• Scheduling at destination, scheduling at source, priority level of each packet, and number of free tokens per source allow for different performance goals without network modification
Solution: Data Center TCP (DCTCP)

- TCP-like protocol
- Uses Explicit Congestion Notification (ECN) for congestion detection
- Implicit rate control by keeping queues small/empty
- Provides high burst tolerance and low latency for short flows
- Allows applications to handle much more background traffic without interrupting foreground traffic
- Increasing Foreground traffic does not cause timeouts
DCTCP Algorithm

• Goal: Achieve high burst-tolerance, low latency, and high throughput, with commodity shallow buffered switches
• Achieves these goals primarily by reacting to congestion in proportion to extent of congestion
• DCTCP source reacts by reducing window by a factor that depends on the fraction of marked packets
DCTCP Algorithm (Cont’d)

• Simple marking at switch
  • arriving packet will be marked if the queue has more than k elements. Otherwise, not marked.

• ECN Echo at receiver
  • uses delayed ACKS, otherwise similar to TCP receivers

• Controller at the Sender
  • sender maintains an estimate of fraction of packets that are marked, called alpha, which is updated once for every window of data
DCTCP Algorithm (Cont’d)

\[ \alpha \leftarrow (1 - g) \times \alpha + g \times F, \]

- Where \( F \) is the fraction of packets that were marked in the last window of data
- \( g \) is the weight given to new samples
- if \( \alpha \) close to 0, low congestion, if \( \alpha \) close to 1, high congestion
Benefits

• Solves three problems:
  • Queue Buildup
    • DCTCP senders start to react as soon as queues have > K elements in them
    • reduces queuing delays on congested ports
    • allows for more headroom to absorb bursts
  • Buffer Pressure
    • a congested port’s queue length does not grow exceedingly large
  • Incast
    • incast scenario: large number of synchronized small flows hit the same queue
    • early/aggressive marking allows DCTCP to tame the size of follow up bursts
Evaluation Summary

• pHost and pFabric seem to do better than DCTCP with websearch and data mining, but DCTCP does the best for total request completion times in Incast traffic patterns
• unclear how MPTCP does against DCTCP,
• all of these options are better than TCP
Evaluation (Cont’d)

95\textsuperscript{th} percentile of query completion time

<table>
<thead>
<tr>
<th></th>
<th>Without background traffic</th>
<th>With background traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>9.87ms</td>
<td>46.94ms</td>
</tr>
<tr>
<td>DCTCP</td>
<td>9.17ms</td>
<td>9.09ms</td>
</tr>
</tbody>
</table>

With Dynamic Buffering in DCTCP:

Figure 23: Completion time: query traffic
Evaluation compared to pFabric

Figure 6: Total request and individual flow completion times in Incast scenario. Note that the range of the y-axis is different for the two plots.

Figure 7: Overall average normalized flow completion time for the two workloads at various loads.
Evaluation compared to pFabric

Figure 8: Web search workload: Normalized FCT statistics across different flow sizes. Note that TCP-DropTail does not appear in part (b) because its performance is outside the plotted range and the y-axis for part (c) has a different range than the other plots.

Figure 9: Data mining workload: Normalized FCT statistics across different flow sizes. Note that TCP-DropTail does not appear in part (b) because its performance is outside the plotted range.
Future Work and Discussion
Questions
Discussion Questions

• Why is today’s class centered around DCTCP instead of more efficient transport protocols like pHost and pFabric?
• Is it worthwhile to continue looking into transport protocols similar to TCP, or should we look into different concepts like decoupling rate control from packet scheduling, like pFabric and pHost?