Network Functions

Lecture 17, Computer Networks (198:552)
Fall 2019
Internet: the textbook view

- A set of smooth pipes from source to destination
- All functionality and guarantees at edge
- Best effort packet forwarding in core
Internet: Reality

Reality: Lots of “glue”
Middleboxes!
What are middleboxes?

- Specialized **applications**
  - Security, application optimization, network management
  - **Stateful** across pkts & connections

- Specialized **appliances**
  - Hardware boxes with custom management interfaces

- Thousands to millions of $$ in equipment (capex)
  - ... and more to operate, upgrade, optimize their performance (opex)
Significant deployments in enterprises

Figure 1: Box plot of middlebox deployments for small (fewer than 1k hosts), medium (1k-10k hosts), large (10k-100k hosts), and very large (more than 100k hosts) enterprise networks. Y-axis is in log scale.

Source: Aplomb, Justine Sherry et al., SIGCOMM ‘12
Concerns for network operators

• **Upgradability**: hardware vs. software

• **Expertise** to operate: configure and optimize

• **Monitoring**: Need visibility & diagnostics when things fail

• **Load management**: what if network traffic suddenly increases?

• Could really use an SDN-like approach 😊
Network Function Virtualization (NFV)

- Encapsulate specialized app as software network functions
- Run network functions on general-purpose processors
- Easier to debug, upgrade, manage, and develop
- Reduce costs through server consolidation
  - Multiplex different network functions onto the same hardware
- Q: how to make network functions run fast?
Options to build fast and flexible NFs

• Option 1: take fast switches and make them programmable
  • Programmable routers exist today, but insufficiently flexible for NFs
  • Ex: security pattern using regular expression on multiple TCP packets

• Option 2: take slow gen-purpose processors and make them fast
  • Single servers are 10—100X slower than optimized routers
  • Too big a gap to close even with performance advancements

• RouteBricks takes option 2
  • Enable programmability
  • Make up for single-server performance using many servers
RouteBricks
Implementing fast and flexible routers on general-purpose processors
Goals

- Build fast high-port-density router from $N$ slow servers
- Each server has a few external-facing ports each of rate $R$
  - Many other internal-facing ports to connect to each other
- Each server processes packets at a small speedup ($c*R$)
  - Independent of the total number of servers or ports
- Limited number of ports per server
- Be able to get 100% throughput on any external port
- Each internal port not much faster than an external port
- Avoid reordering packets
- Each input port gets its fair share of output port capacity
Two main steps

• Build a cluster of servers that talk to each other
  • Leverage *parallelism across servers*

• Improve performance of single server processing
  • Leverage *parallelism of components within each server*
Step (1): Cluster topology & routing

- Suppose servers connect to each other in a **full mesh**
  - N*N links

- How should they route amongst each other?
- (Ingress server knows which egress server the pkt goes to)

- Single path vs. multi-path?
- Static vs. adaptive to load?
- Centralized vs. distributed?
Distributed multipath routing

Same servers acting in 3 logical layers
Distributed multipath routing with VLB

Use VLB to split ingress data across intermediate layer
Distributed multipath routing with VLB

Internal link from ingress to int. layer only runs at rate 2R/8 (Q: why?)
But VLB costs you...

- Q: What’s the total rate at which servers process packets if you routed directly to the egress server from the ingress?

- $2R$ (full duplex)
Distributed multipath routing with VLB

3R (full duplex)
Direct VLB

• If the traffic matrix were uniform, could you do better?
  • uniform: contribution to every output port was the same from every input port, i.e., $R/N$

• Idea: don’t shuffle data unnecessarily to intermediate layer
  • Send the $R/N$ destined to that output server at int. layer itself

• General case: send up to $R/N$ of demand to an egress directly to the egress server
  • Valiant-load-balance the rest of the traffic to that server
Direct VLB

Cut off entire layer to bring per-server rate to 2R
Topology design

• All this is great with a full mesh
• But full mesh requires that each server have N ports
  • Problematic at large N
  • Servers can’t support too many NIC ports!

• Use intermediate stages of (real) physical servers
  • Typical topologies: butterfly, torus
  • Increase the number of nodes connected
  • While keeping the per-node degree the same
Butterfly topology

• Each node has constant degree $k$

• If number of nodes in the first and last stage is $N$, need $n = \log_k N$ stages

• Also called a $k$-ary $n$-fly
Step (2): Improve single-server perf

- Leverage parallelism, not just among CPU cores
  - Memory bus, IO-bus, socket links, PCI bus
Step (2): Improve single-server perf

- Leverage parallelism, not just among CPU cores
  - Memory bus, IO-bus, socket links, PCI bus
Step (2): Improve single-server perf

- Leverage parallelism, not just among CPU cores
  - Memory bus, IO-bus, socket links, PCI bus
Step (2): Improve single-server perf

- Leverage parallelism, not just among CPU cores
  - Memory bus, IO-bus, socket links, PCI bus
Improve how cores access pkts from NIC

Principle: use single core to process pkt to completion

Using cores in parallel v.s. pipeline

(a) 1.2 Gbps/FP (shared cache)
0.6 Gbps/FP (different cache)

(b) 1.7 Gbps/FP

Legend: NIC port  Core  NIC queue

Why? Sync overheads
Improve how cores access pkts from NIC

- Accessing each NIC queue requires core to lock the queue
  - Too contentious at high rates: cores need to wait to get lock

- Principle: assign one NIC queue per port to each core

- Corollary: **NICs must have multiple queues** else perf suffers
Performance

Basic forwarding on a single server

Aggregate router performance: N=4 servers with 1 ext port/server
35 Gbit/s at Abilene packet size and 12 Gbit/s at 64-Byte packet size
Summary of RouteBricks

• Design topology and routing well to parallelize across servers
• Optimize parallelism within the server’s computer system

• RouteBricks published in 2009, same year as VL2
  • Many similarities in design; both very influential

• However, RB not predominant NF arch. Many things to consider:
  • Cost (of servers, links)
  • Using CPUs just to move packets rather than app processing
  • Complexity of the overall solution
  • Availability of merchant silicon to easily load-balance across servers