Review of Networking Concepts (Part 2)

Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute
shivkuma@ecse.rpi.edu
http://www.ecse.rpi.edu/Homepages/shivkuma

- **General System Design techniques**
  - **Multiplexing, virtualization**
  - **Parallelization & pipelining**
  - **Batching, Randomization,**
  - **Locality and hierarchy,**
  - **Separating data & control, Extensibility**
- **Performance**

System Design Ideas

- ***Resources:*** space, time, computation, money, labor.
  - Design a system to tradeoff cheaper resources against expensive ones (for a gain)
- ***Protocol:***
  - Specification: message semantics and actions taken while sending/receiving them.
  - Interface between layers is also called the architecture. Interface design crucial because interface outlives the technology used to implement the interface.

System Design Ideas (contd)

- ***Layering and encapsulation:***
  - Allows a subroutine abstraction between a layer and its adjacent layers.
  - Layering allows pipelined implementations.
  - ***Application layer framing:*** packet format at every layer flexible and application-defined
- ***Circuit-switching***: resource (circuit) reservation followed by time-bound transmission.
  - Resources wasted if unused: expensive.
  - Straightforward to assure quality for voice (150 ms round trip delay, 64 Kbps bandwidth).

Design Ideas (contd)

  - Time slots have no meta-data (header) associated. All relevant meta-data is inferred from timing and state installed during circuit/connection-setup.
- ***Packet-switching:*** packets with meta-data (header) and store-and-forward type transmission.
  - Very efficient – can exploit multiplexing gains both in space and time (see below).
  - Cost: self-descriptive header per-packet, buffering and delays for applications. (tradeoff space and time for money)

Statistical Multiplexing

- Reduce resource requirements by exploiting statistical knowledge of the system. Specifically, choose service rate such that:
  - average rate <= service rate <= peak rate
  - Muxing Gain = peak rate/service rate.
  - Cost: buffering, delays for applications. Tradeoff space and time resources for money
  - Useful only if peak rate differs significantly from average rate.

Spatial and Temporal Multiplexing

- Spatial muxing: Decrease resource sizing expecting smaller set of sources to be active at any time instant.
  - Cost: call-blocking (time)
- Temporal muxing: even if many are active at any particular time instant, expect that the average over time will be much smaller.
    - Cost: buffers and meta-data (headers) in packets (space).
- Note: We need packet switching to exploit both spatial and temporal gains.

Virtualization

- Virtualization: If Quality of Service (QoS) is met, the multiplexed shared resource may seem like a *unshared virtual resource*.
- Multiplexing + indirection = virtualization, i.e., refer the virtual resource as if it were the physical resource itself.
  - Eg: virtual memory, virtual circuit, socket ports in BSD, telephone call.
  - Indirection requires binding and unbinding...

Pipelining and Parallelism

- Goal: trading computation for (gain in) time.
- Degree of parallelism: response time x throughput
- Linear speedup: split up task into N independent subtasks, each requiring same amount of time.
  - Response time speedup of N. Throughput constant. Degree = N
- Pipelining: Can't independently split subtasks - the subtasks may be serially dependent.
  - We can get speedup in throughput, but NOT in response time by using pipelining

Batching

- *Goal:* trading response time for (gain in) throughput
- Batching is good when:
  - overhead per task increases less than linearly w/ number of tasks
  - time to accumulate a batch is not too long.
  - Response time can be traded off
- Eg: Interrupt handling, Silly window avoidance in TCP
- TCP also has triggers to avoid batching for telnet packets -- when response time is important

Randomization

- Goal: Trade computation for (response) time
- Used in breaking ties without biases or high probability of repeat of tie.
  - Eg: Use of exponential backoff in broadcast multiple access (ethernet), avoidance of ACK or NAK implosion in reliable multicast, or in some routing algorithms.

Locality and Hierarchy

- *Locality*: Critical in exploiting smaller, faster resource to create an illusion of a larger, faster resource.
  - The larger, slower resource, is accessed when item is not found in the smaller resource.
- *Hierarchy*: for scalability.
  - Loose hierarchies more efficient than strict ones (eg: children can interconnect).
  - Eg: managing name space or address allocation and forwarding.

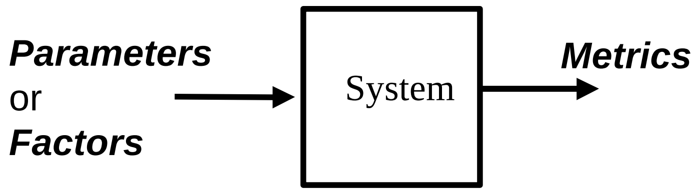
Miscellaneous ideas

  - Different types of hierarchy: topological, routing, traffic management, organizational.
- *Separating data and control:* Per-packet actions are part of critical data path -- fewer control actions => greater forwarding speed.
  - Greater separation of data and control => need to install more state in the network.
  - Eg: separate CCIS channel in telephony.
  - Eg: separate routing protocols in Internet.
- *Extensibility:* hooks for future growth. Eg: version field, reserved fields.

What is performance ?

- *How fast* **does computer A run MY program ?**
- **Is machine A** *generally faster than* **machine B, and if so,** *how much faster* **?**
- **Performance is one of the three factors** *driving* **architecture (interface design)**
  - **Others: <u>functionality</u> demanded, <u>technology</u> available**

Metrics and Parameters



- ***Parameters:*** clock rate, poisson inter-arrivals, ftp workload etc
- ***Metrics:*** throughput, response time, queue length.
  - Metric choice should characterize the design tradeoffs (in space, time etc) adequately
  - Metrics are usually functions of many factors. Use of one factor alone may be misleading.

More on Metrics/Parameters

- **User metrics:**
  - **How fast does MY program run => we need a measure of execution *time* ?**
- **System metrics:**
  - **How much is the system utilized ?**
  - **How much buffers do I need to provision ?**
  - **How many programs is it able to execute per second ?**
  - **=> Need a measure of *throughput, queue length***
- **Eg: Execution Time = Instrns/pgm *avg clock cycles/Instruction * time/clock cycle.**
- **T = I /P * CPI *Clock cycle time**
- **All three factors *combine* to affect the metric.**

Workloads, Benchmarks

- *Workload:* **a test case for the system**
- *Benchmark:* **A set of workloads which together is representative of "MY program". Should be reproduceable**
  - **Problem: combining metrics from each test case.**
  - **Pitfalls: ratio games.**

Test case \ Machine	A	B
1	1s	10s
2	100s	10s

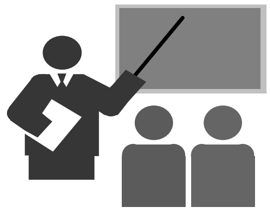
Which is faster, A or B ?

Effect on Design: Amdahl's law

- **Execution time after improvement =**

  Execution time *affected* **by improvement / speedup** + *Unaffected* **execution time**
- *Point:* **Speedup the common case !!**

Summary



- Constraints: space, time, computation,money
- Techniques:
  - Multiplexing, Parallelism and Pipelining,
  - Batching, Randomization,
  - Locality and hierarchy,
  - Separating data & control, Extensibility
- Performance