Queueing theory: review
- a queueing system consists of
  - an arrival process
  - a service process
  - capacity
  - a queueing discipline (FIFO, LIFO, RR)
- reviewed exponential and pareto (heavy-tailed) distributions

Queueing notation:
format is A/B/C/D, where
- A = arrival process (shorthand notation for distribution)
- B = service process (ditto)
- C = number of servers
- D = system capacity

Examples:
M/M/1/100: exponential arrivals, exponential service times, 1 server, capacity of 100
D/G/3: deterministic arrivals, “generally distributed” service times, 3 servers, infinite capacity

More values
\( \lambda \) = (average) arrival rate, in requests/sec
\( \mu \) = (average) service rate, in requests/sec
\( 1/\mu \) = (average) service time, in seconds
\( \rho = \lambda/(c\mu) \) = system load (indicates how many requests are in service, on average)

Note that the load needs to be strictly below 1 for the system to be stable!

Little's Law
We always want to know how many requests are in the system, and how long it takes each request to be served. Knowing the average is often sufficient.

Little's Law demonstrates the relationship between the average number of requests in the system and the average time requests spend in the system:

\[ L = \lambda W \]
- L = average number in system
- W = average wait time, in seconds
Can use the same formula to determine the time spend just in the queue, or just the number in the queue, on average.

**Finite queues**
In the real world, queues are not infinite (though mathematically we can often assume that they are). Such queues must have a *drop policy*.
Some drop policies:
- drop tail
- selective discard

**Back to the Internet**
Now we have a different abstraction for routers: that of a single-server, drop-tail FIFO queue
This suits data traffic well, BUT may not be adequate for today's mix of traffic.
Proposal: priority queueing (multiple queues, single server).
This leaves the question: How do we decide how to separate traffic into queues?
TUNE IN MONDAY FOR THE ANSWER!