Internet Measurement and Data Analysis (11)

Kenjiro Cho

2011-11-30

review of previous class

Class 8 Distributions

- normal distribution and other distributions
- confidence intervals
- statistical tests
- exercise: generating distributions, confidence intervals
- **assignment 2**

Class 9 and 10 Free Discussion

- how to do research

today's topics

Class 9 Measuring time series of the Internet

- Internet and time
- network time protocol
- time series analysis
- exercise: time series analysis

time in measurement

- absolute time
  - UTC (Universal Coordinated Time)
    - the international standard time based on atomic clocks
- relative time
  - difference between events
- clock adjustment
  - clock could jump forward or backward!
  - ntp slews clock if difference is less than 128ms

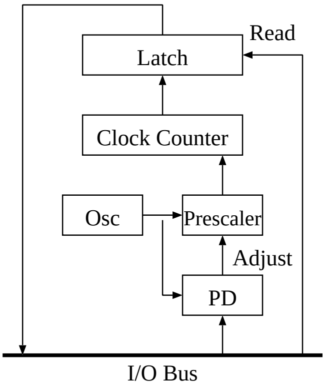
clock uncertainty

- clock uncertainty
  - synchronization
    - difference of 2 clocks
  - accuracy
    - a given clock agrees with UTC
  - resolution
    - precision of a given clock
  - skew
    - change of accuracy or of synchronization with time
- time precision
  - local clock skew/drift: 0.1-1sec/day
  - NTP: synchronizes clock within 10-100ms
  - tcpdump timestamp: 100usec-100msec (usually $< 1msec$)

PC clock

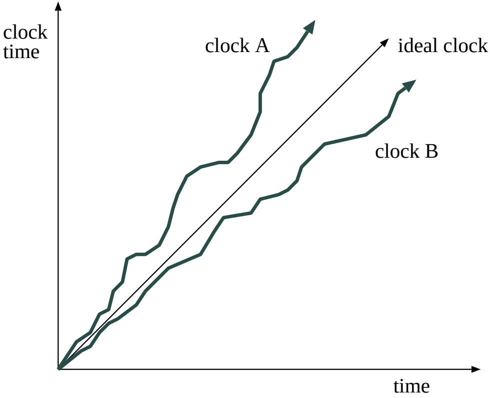
i8254 programmable interval timer

- free-running 16-bit down-counter
  - driven by 1,193,182 Hz oscillator
  - when counter becomes zero, generates interrupt, and reloads the counter register



clock drift

- oscillator drift
  - hardware error margin: 10^{-5}
    - 0.86 sec/day within the spec
  - drift heavily affected by temperature



alternative clocks

- Pentium TSC (Time Stamp Counter)
  - a 64bit free-running counter driven by CPU clock
  - issues with variable clock rate and multi-processors
- ACPI (Advanced Configuration and Power Interface)
  - a free-running counter provided by power management unit
- Local APIC (Advanced Programmable Interrupt Controller)
  - timer with interrupt function embedded on each processor
- HPET (High Precision Event Timer)
  - a new time specification of IA-PC
  - built in chipsets since around 2005
- external clock source
  - GPS, CDMA, shortwave radio
    - access overhead of the interfaces

OS time management

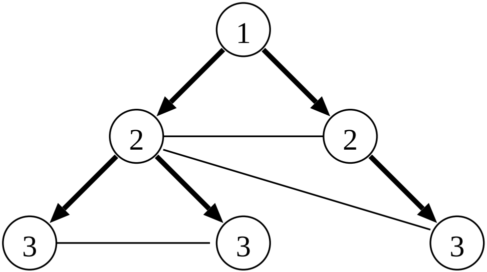
- OS manages software clock
  - initialized at boottime from time-of-day chip
  - updated by hardware clock interrupts
- standard UNIX sets the clock counter (and divider) to interrupt every 10ms (configurable)

UNIX gettimeofday

- older OS has only clock-interrupt resolution
- modern OS has much better resolution
  - interpolate software clock by reading the remaining counter value
    - resolution: 838ns (1 / 1193182)
  - inside kernel
    - access to the i8254 register: 1-10usec
    - conversion to struct timeval: 10-100usec
  - user space - kernel
    - system call overhead: 100-500usec
    - process might be scheduled: 1-100msec or more
- timer events (e.g., setitimer):
  - triggered only by timer tick (10msec by default)
  - effects of process scheduling

NTP (Network Time Protocol)

- multiple time servers across the Internet
  - primary servers: directly connected to UTC receivers
  - secondary servers: synchronize with primaries
  - tertiary servers: synchronize with secondary, etc
- scalability
  - 20-30 primaries, 2000 secondaries can synchronize to $< 30ms$
- many features
  - cope with server failures, authentication support, etc



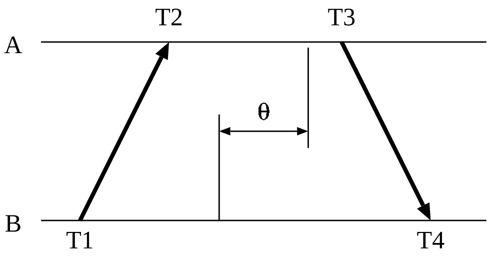
NTP synchronization modes

- multicast (for LAN)
  - one or more servers periodically multicast
- remote procedure call
  - client requests time to a set of servers
- symmetric protocol
  - pairwise synchronization with peers

NTP symmetric protocol

measuring offset and delay

- $a = T2 - T1 \qquad b = T3 - T4$
- clock offset: $\theta = (a + b)/2$, assuming symmetric round-trip
- roundtrip delay: $\delta = a - b$

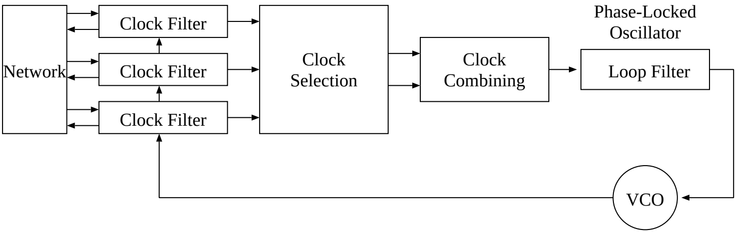


every message contains

- T3: send time (current time)
- T2: receive time
- T1: send time in received message

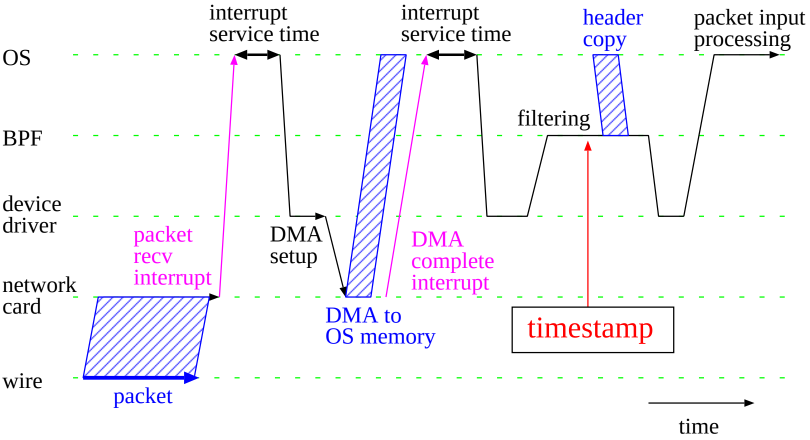
NTP system model

- clock filter
  - temporally smooth estimates from a given peer
- clock selection
  - select subset of mutually agreeing clocks
  - intersection algorithm: eliminate outliers
  - clustering: pick good estimates
- clock combining
  - combine into a single estimate



BPF timestamp on BSD Unix

- timestamp usually placed after 2 interrupts: recv packet, DMA complete
  - recv packet, DMA complete



self-similarity in network traffic

analysis of dynamic behaviors which change over time

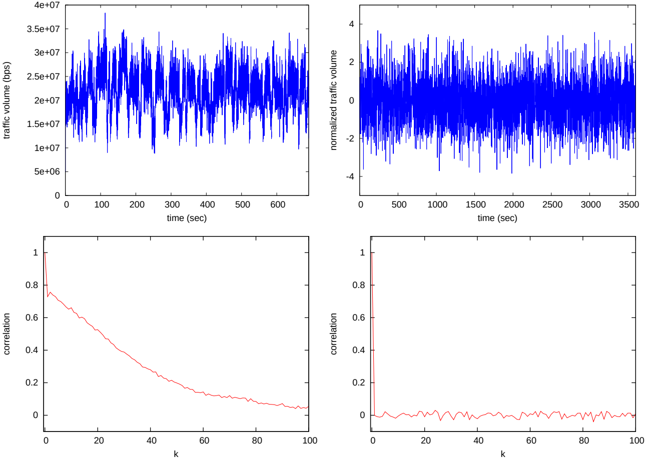
- difficult for mathematical modeling
- only limited tools are available

topics

- autocorrelation
- stationary process
- long-range dependence
- self-similar traffic

autocorrelation of network traffic

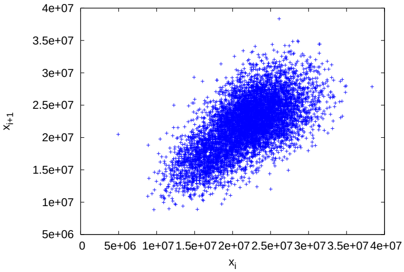
- trends (influence from the past) and periodicity (day, week, season)
- autocorrelation: correlation between two values of the same variable at different times

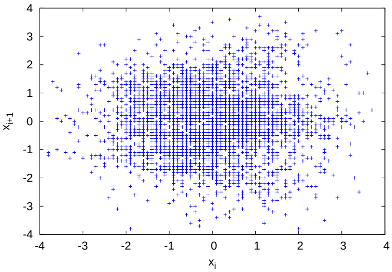


real traffic (left) and randomly generated traffic (right) timeseries (top) and autocorrelation (bottom)

autocorrelation and lag plot

- lag plot: scatter plot of x_i and x_{i+k}
  - simple way to observe whether autocorrelation exists
  - larger k can find longer cycles of repeating patterns





sample lag plot: real traffic (left) and randomly generated traffic (right)

autocorrelation

- stochastic process
$$\{x(t), t \in T\}$$
- autocorrelation: correlation between two values of the same variable at times t_1 and t_2
- autocorrelation function
$$R(t_1, t_2) = E[x(t_1)x(t_2)]$$
- autocovariance
$$Cov(t_1, t_2) = E((x(t_1) - \mu_{t_1})(x(t_2) - \mu_{t_2})] = E[x(t_1)x(t_2)] - \mu_{t_1}\mu_{t_2}$$

stationary process

- time-series X_t is stationary if
  - mean does not change with time: $E(X_t) = \mu$
  - and autocovariance depends only on k

$$\gamma_k = Cov(X_t, X_{t+k}) = E((X_t - \mu)(X_{t+k} - \mu))$$

$$\gamma_0 = Var(X_t) = E((X_t - \mu)^2)$$

- autocorrelation coefficient
  - autocovariance normalized by variance
  - shows influence of the past

$$\rho_k = \frac{\gamma_k}{\gamma_0}$$

white noise

white noise: stationary process whose autocorrelation coefficient is zero

$$\rho_k = 0 \ (k \neq 0)$$

IID process (independent identically distributed process)

- white noise with constant mean and variance
    - IID process often appears in the literature
- X_t is IID
    - independent: X_t is independent (no autocorrelation)
    - identically distributed: X_t follows the same distribution

non-stationary process

- non-stationary
  - mean changes with time
  - or, autocovariance changes with time
- hard to tackle mathematically
  - generally, take differential time-series to make it stationary
- stationarity test
  - by power spectral density
    - if power-law exponent > 1.0, non-stationary
- network data: sometimes, non-stationary behaviors are observed
  - caused by congestion, attack, etc

power spectral density

- power spectral density of a stationary random process is the fourier transform of the autocorrelation function
  - from time-domain to frequency-domain

$$S(f) = \int_{-\infty}^{\infty} R(\tau) e^{-2\pi i f \tau} d\tau$$

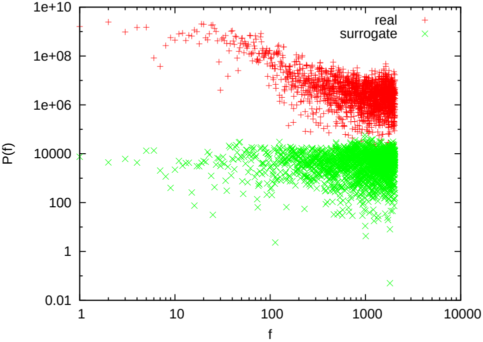
  - power spectral density

$$P(f) \equiv |S(f)|^2 + |S(-f)|^2,\ 0 \leq f < \infty$$

- power spectral density gives relative power contributed by each frequency component

characteristics of power spectral density

- white noise: $P(f) \sim const$
- self-similar (long-range dependence):
  $P(f) \sim f^{-\alpha}, 0 < \alpha \leq 1.0$
- 1/f fluctuation: $\alpha = 1.0$
- non-stationary: $\alpha > 1.0$



example: real traffic (red) and randomly generated traffic (green)

short-range dependence and long-range dependence

autocovariance shows the influence of each time difference k

sum of autocovariance of all time differences k gives a total view

- short-range dependence
  - $\sum_k \rho(k)$ is finite

$$\sum_{k=0}^{\infty} |\rho(k)| < \infty$$

  - $\rho(k)$ decays at least as fast as exponentially
  - characteristics
    - fluctuates around mean
    - not affected by long past
- long-range dependence
  - $\sum_k \rho(k)$ is infinite

$$\sum_{k=0}^{\infty} |\rho(k)| = \infty$$

  - autocorrelation coefficient decays hyperbolically
  - characteristics
    - values far from mean can be observed

self-similar traffic

network traffic is not exactly self-similar but often better modeled than other models

- scale-invariant
- long-range dependence
- autocovariance decays exponentially

$$\rho(k) \sim k^{-\alpha} \quad (k \to \infty) \quad 0 < \alpha < 1$$

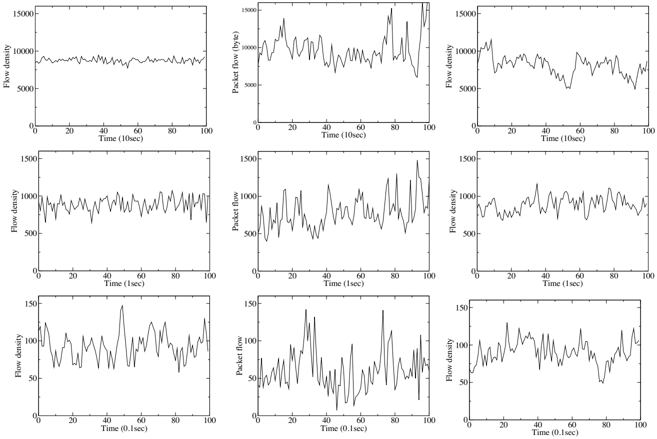
- similarly, power spectral density decays exponentially
  - larger contributions by low frequency components

$$P(f) \sim |f|^{-\alpha} \quad (f \to 0)$$

- infinite variance

self-similarity in network traffic

- exponential model (left), real traffic (middle), self-similar model (right)
- time scale: 10sec (top), 1 sec (middle), 0.1 sec (bottom)



previous exercise: generating normally distributed random numbers

- using a uniform random number generator function (e.g., rand in ruby), create a program to produce normally distributed random numbers with mean u and standard deviation s.

box-muller transform

basic form: creates 2 normally distributed random variables, z_0 and z_1, from 2 uniformly distributed random variables, u_0 and u_1, in $(0, 1]$

$$z_0 = R\cos(\theta) = \sqrt{-2\ln u_0}\cos(2\pi u_1)$$

$$z_1 = R\sin(\theta) = \sqrt{-2\ln u_0}\sin(2\pi u_1)$$

polar form: approximation without trigonometric functions
u_0 and u_1: uniformly distributed random variables in $[-1, 1]$,
$s = u_0^2 + u_1^2$ (if $s = 0$ or $s \geq 1$, re-select u_0, u_1)

$$z_0 = u_0\sqrt{\frac{-2\ln s}{s}}$$

$$z_1 = u_1\sqrt{\frac{-2\ln s}{s}}$$

previous exercise: box-muller random number generator

```
# usage: box-muller.rb [n [m [s]]]
n = 1 # number of samples to output
mean = 0.0
stddev = 1.0

n = ARGV[0].to_i if ARGV.length >= 1
mean = ARGV[1].to_i if ARGV.length >= 2
stddev = ARGV[2].to_i if ARGV.length >= 3

# function box_muller implements the polar form of the box muller method,
# and returns 2 pseudo random numbers from standard normal distribution
def box_muller
  begin
    u1 = 2.0 * rand - 1.0  # uniformly distributed random numbers
    u2 = 2.0 * rand - 1.0  # ditto
    s = u1*u1 + u2*u2    # variance
  end while s == 0.0 || s >= 1.0
  w = Math.sqrt(-2.0 * Math.log(s) / s) # weight
  g1 = u1 * w  # normally distributed random number
  g2 = u2 * w  # ditto
  return g1, g2
end
# box_muller returns 2 random numbers.  so, use them for odd/even rounds
x = x2 = nil
n.times do
  if x2 == nil
    x, x2 = box_muller
  else
    x = x2
    x2 = nil
  end
  x = mean + x * stddev  # scale with mean and stddev
  printf "%.6f\n", x
end
```

exercise: autocorrelation

- compute autocorrelation using traffic data for 1 week

```
# ruby autocorr.rb autocorr_5min_data.txt > autocorr.txt
# head -10 autocorr_5min_data.txt
2011-02-28T00:00 247 6954152
2011-02-28T00:05 420 49037677
2011-02-28T00:10 231 4741972
2011-02-28T00:15 159 1879326
2011-02-28T00:20 290 39202691
2011-02-28T00:25 249 39809905
2011-02-28T00:30 188 37954270
2011-02-28T00:35 192 7613788
2011-02-28T00:40 102 2182421
2011-02-28T00:45 172 1511718
# head -10 autocorr.txt
0 1.0
1 0.860100559860259
2 0.859909329457425
3 0.8568488888567
4 0.856910911636432
5 0.853982084154458
6 0.850511942135165
7 0.848741549347501
8 0.845725096810473
9 0.840762312233673
```

computing autocorrelation functions

autocorrelation function for time lag k

$$R(k) = \frac{1}{n} \sum_{i=1}^{n} x_i x_{i+k}$$

normalize by $R(k)/R(0)$, as when $k = 0$, $R(k) = R(0)$

$$R(0) = \frac{1}{n} \sum_{i=1}^{n} x_i^2$$

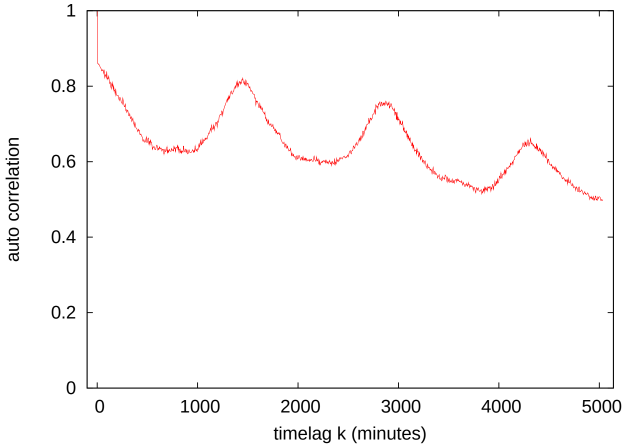
need 2n data to compute $k = n$

autocorrelation computation code

```
# regular expression for matching 5-min timeseries
re = /^(\d{4}-\d{2}-\d{2})T(\d{2}:\d{2})\s+(\d+)\s+(\d+)/

v = Array.new() # array for timeseries
ARGF.each_line do |line|
  if re.match(line)
    v.push $3.to_f
  end
end

n = v.length # n: number of samples
h = n / 2 - 1 # (half of n) - 1

r = Array.new(n/2) # array for auto correlation
for k in 0 .. h # for different timelag
  s = 0
  for i in 0 .. h
    s += v[i] * v[i + k]
  end
  r[k] = Float(s)
end

# normalize by dividing by r0
if r[0] != 0.0
  r0 = r[0]
  for k in 0 .. h
    r[k] = r[k] / r0
    puts "#{k} #{r[k]}"
  end
end
```

autocorrelation plot

```
set xlabel "timelag k (minutes)"
set ylabel "auto correlation"
set xrange [-100:5140]
set yrange [0:1]
plot "autocorr.txt" using ($1*5):2 notitle with lines
```



assignment 2: normal distribution, histogram and confidence interval

- the purpose is to understand normal distribution and confidence interval
- assignment
  1. generate 10 sets of normally distributed numbers with varying sample size.
  2. create 2 histogram plots for sample size 128 and 1024
  3. compute confidence interval of mean for the 10 sets, and make a plot
- items to submit
  1. 2 histogram plots
  2. a plot of interval estimation for the 10 sample sets
- submission format: a single PDF file including 3 plots (2 histogram plots and 1 interval estimation plot)
- submission method: upload the PDF file through SFC-SFS
- submission due: 2011-12-03

assignment details

1. generate 10 sets of normally distributed numbers with varying sample size.
    - use the box-muller code in today's exercise
    - use your height in cm for mean, and half of your foot size in cm for standard deviation
    - with varying sample size $n = \{4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$.
2. create 2 histogram plots for sample size 128 and 1024
    - confirm that the generated random numbers follow normal distribution
    - select appropriate bin size for histograms using commonly used boundaries for heights (e.g., 1cm, 2cm, 5cm, etc)
3. compute confidence interval of mean for the 10 sets, and make a plot
    - confirm that confidence interval changes according to sample size.
    - for each of the 10 sample sets, compute the confidence interval of mean. Use confidence level 95%, confidence interval $\mp 1.960 \frac{s}{\sqrt{n}}$.
    - plot the results of the 10 sets in a single graph; X-axis for sample size n in log-scale, Y-axis for mean and confidence interval in linear scale. (the plot should look similar to slide 17).

summary

Class 9 Measuring time series of the Internet

- Internet and time
- network time protocol
- time series analysis
- exercise: time series analysis

next class

No Class on 12/7

Class 12 Measuring anomalies of the Internet (12/14)

- anomaly detection
- spam filters
- Bayes' theorem
- exercise: anomaly detection