Master Kurs Rechnernetze
Computer Networks
IN2097

Prof. Dr.-Ing. Georg Carle
Dr. Thomas Fuhrmann

Institut für Informatik
Technische Universität München
http://www.net.in.tum.de

Architecture: the big picture

Architecture: the big picture

Goals:

- identify, study principles that can guide network architecture
- "bigger" issues than specific protocols or implementation wisdom,
- *synthesis:* the *really* big picture

Overview:

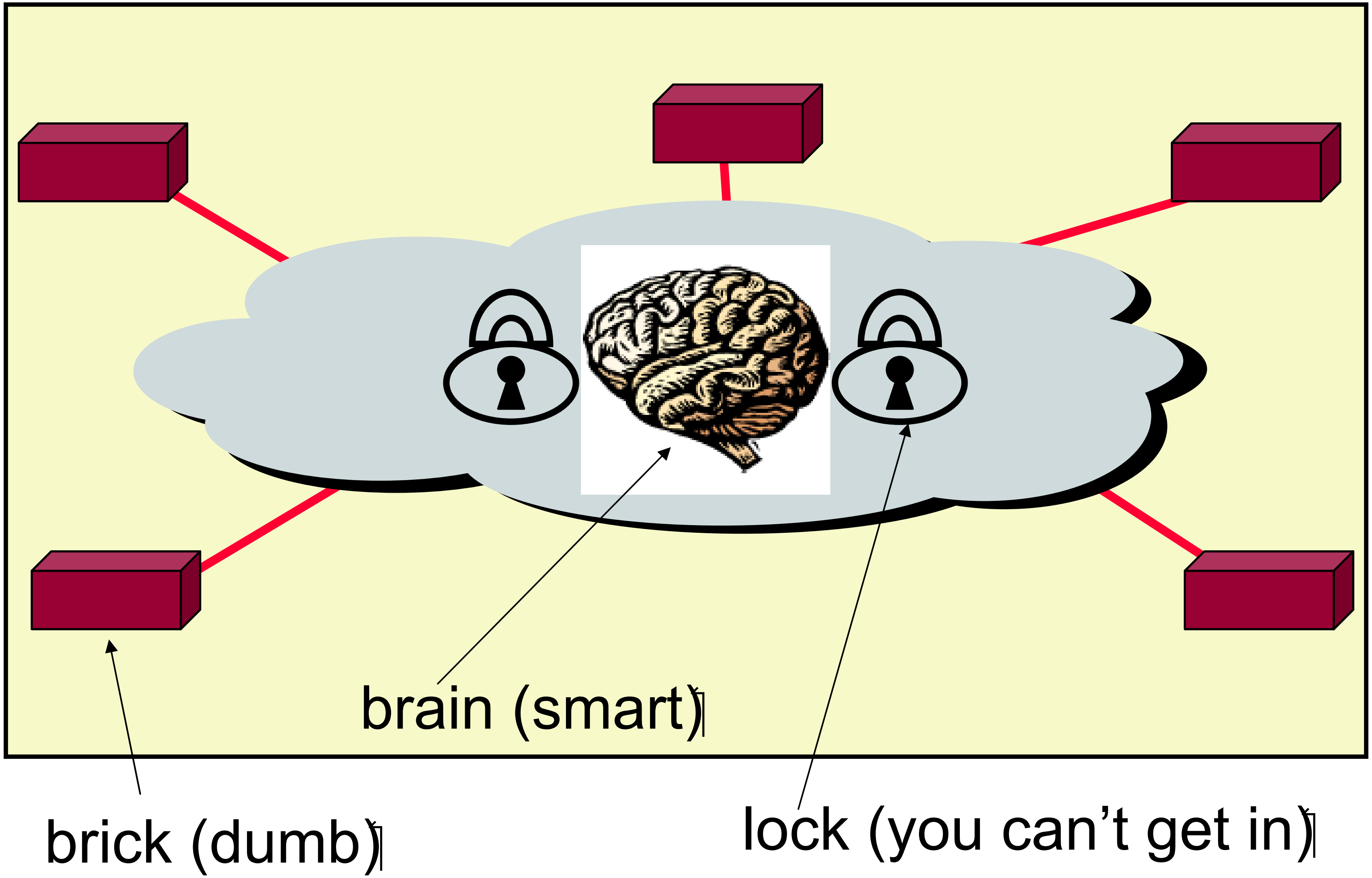
- Internet design principles
- rethinking the Internet design principles
- packet switching versus circuit switching revisited

Key questions

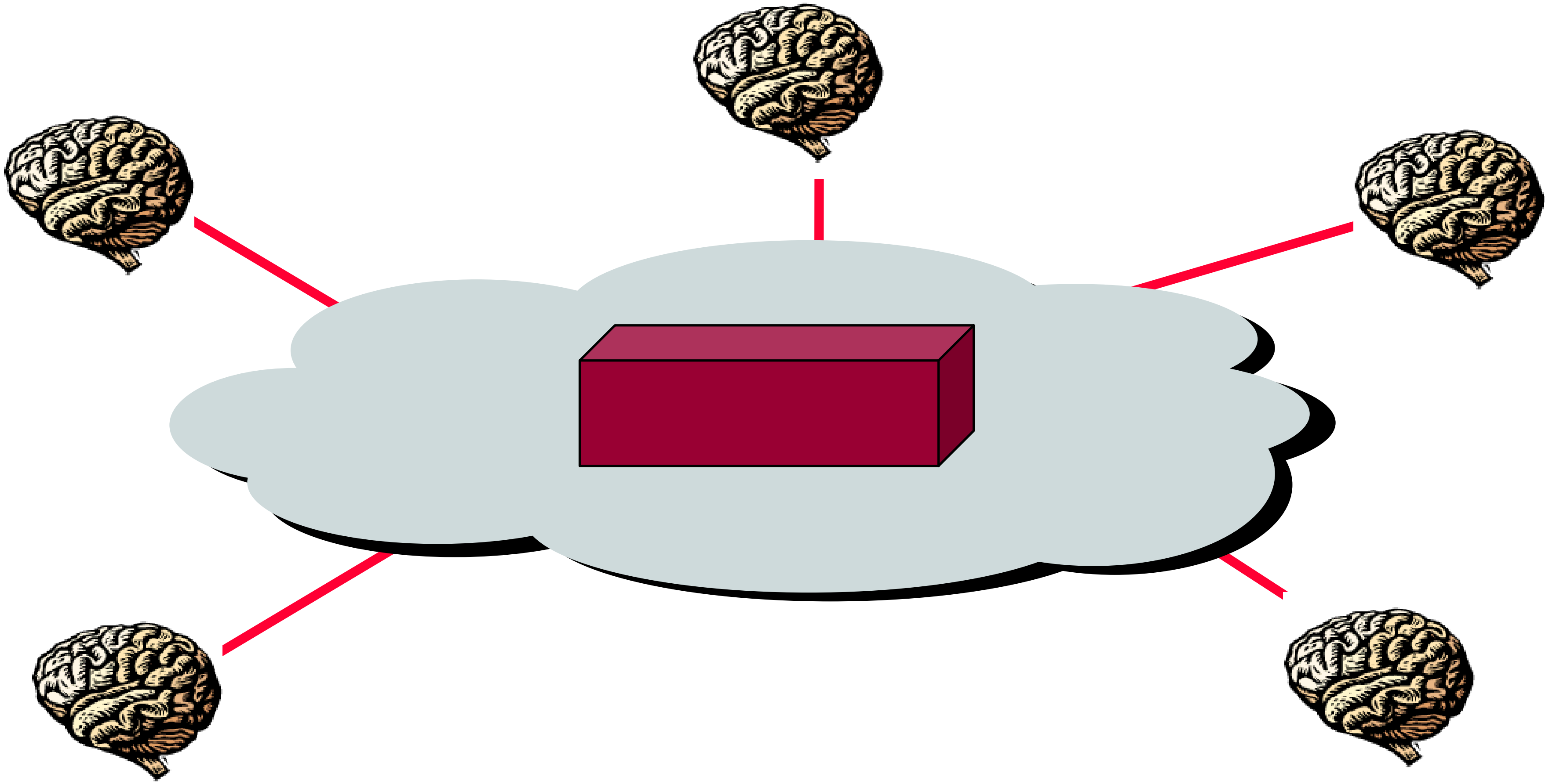
- How to decompose the complex system functionality into protocol layers?
- Which functions placed *where* in network, at which layers?
- Can a function be placed at multiple levels?

- Answer these questions in context of
  - Internet
  - Telephone network
    (Nickname 1: Telco — telecommunications provider)
    (Nickname 2: POTS — "plain old telephone system")

Common View of the Telco Network



Common View of the IP Network



The Internet End-to-End principle

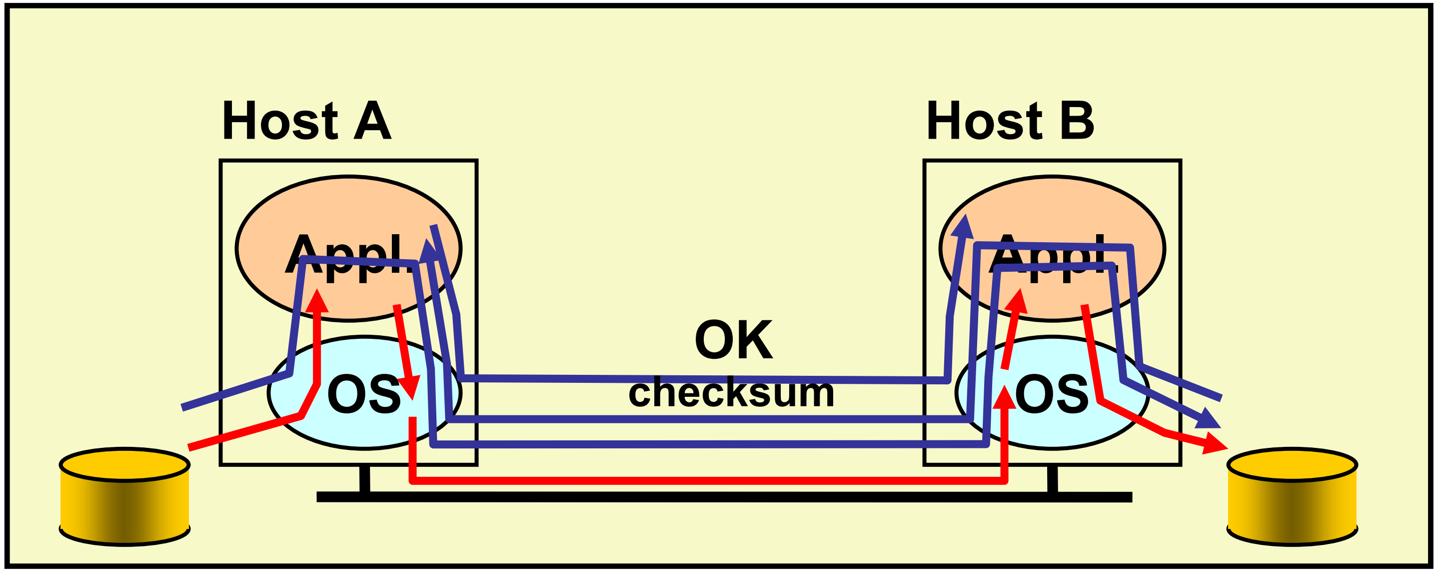
Internet End-to-End Principle

- "…functions placed at the lower levels may be *redundant* or of *little value* when compared to the cost of providing them at the higher level…"

- "…sometimes an *incomplete* version of the function provided by the communication system (lower levels) may be useful as a *performance enhancement*…"

- This leads to a philosophy diametrically opposite to the telephone world of dumb end-systems (the telephone) and intelligent networks.

Example: Reliable File Transfer



- Solution 1: make each step reliable, and then concatenate them

- Solution 2: each step unreliable: end-to-end check and retry

Discussion

- Is solution 1 good enough?
  - No — what happens if components on path fail or misbehave (bugs)?
- Is reliable communication sufficient:
  - No — what happens if disk errors?
- so need application to make final correctness check anyway
- Thus, full functionality can be entirely implemented at application layer; *no* need for reliability from lower layers

Discussion

Q: Is there any reason to implement reliability at lower layers?

A: YES: "easier" (and more efficient) to check and recovery from errors at each intermediate hop

- e.g.: faster response to errors, localized retransmissions

Trade-offs

- application has more information about the data and semantics of required service (e.g., can check only at the end of each data unit)
- lower layer has more information about constraints in data transmission (e.g., packet size, error rate)
- *Note:* these trade-offs are a direct result of layering!

Internet & End-to-End Argument

- network layer provides one simple service: best effort datagram (packet) delivery
- transport layer at network edge (TCP) provides end-end error control
  - performance enhancement used by many applications (which could provide their own error control)
- all other functionality …
  - all application layer functionality
  - network services: DNS
  - ⇨ implemented at application level

E2E Argument: Interpretations

- One interpretation:
  - A function can only be completely and correctly implemented with the knowledge and help of the applications *standing at the communication endpoints*
- Another: (more precise…)
  - A system (or subsystem level) should consider only functions that can be *completely and correctly* implemented within it.
- Alternative interpretation: (also correct …)
  - Think twice before implementing a functionality that you believe that is useful to an application at a lower layer
  - If the application can implement a functionality correctly, implement it a lower layer *only* as a performance enhancement

End-to-End Argument: Critical Issues

- End-to-end principle emphasizes:
  - *function placement*
  - *correctness, completeness*
  - *overall system costs*
- Philosophy: if application can do it, don't do it at a lower layer — application best knows what it needs
  - add functionality in lower layers iff
    (1) used by and improves performances of many applications,
    (2) does not hurt other applications
- allows *cost-performance* tradeoff

Internet Design Philosophy (Clark' 88)

In order of importance:

Different ordering of priorities would make a different architecture!

0. Connect existing networks
   - initially ARPANET, ARPA packet radio, packet satellite network
1. Survivability
   - ensure communication service even with network and router failures
2. Support multiple types of services
3. Must accommodate a variety of networks
4. Allow distributed management
5. Allow host attachment with a low level of effort
6. Be cost effective
7. **Allow resource accountability**

1. Survivability

- Continue to operate even in the presence of network failures (e.g., link and router failures)
  - as long as network is not partitioned, two endpoints should be able to communicate
  - any other failure (excepting network partition) should be transparent to endpoints
- Decision: maintain end-to-end transport state only at end-points
  - eliminate the problem of handling state inconsistency and performing state restoration when router fails
- Internet: stateless network-layer architecture
  - No notion of a session/call at network layer
  - Example: Your TCP connection shouldn't break when a router along the path fails
- Assessment: ??

2. Types of Services

- Add UDP to TCP to better support other apps
  - e.g., “real-time” applications
- arguably main reason for separating TCP, IP
- datagram abstraction: lower common denominator on which other services can be built
  - service differentiation was considered (remember ToS field in IP header?), but this has never happened on the large scale (Why?)
- Assessment: ?

3. Variety of Networks

- Very successful (why?)
  - because the minimalist service; it requires from underlying network only to deliver a packet with a “reasonable” probability of success
- …does not require:
  - reliability
  - in-order delivery
- The mantra: IP over everything
  - Then: ARPANET, X.25, DARPA satellite network..
  - Subsequently: ATM, SONET, WDM…
- Assessment: ?

Other Goals

- Allow distributed management
  - Administrative autonomy: IP interconnects networks
    - each network can be managed by a different organization
    - different organizations need to interact only at the boundaries
    - … but this model complicates routing
  - Assessment: ?

- Cost effective
  - sources of inefficiency
    - header overhead
    - retransmissions
    - routing
  - …but “optimal” performance never been top priority
  - Assessment: ?

Other Goals (Cont)

- Low cost of attaching a new host
  - not a strong point → higher than other architecture because the intelligence is in hosts (e.g., telephone vs. computer)
  - bad implementations or malicious users can produce considerably harm (remember fate-sharing?)
  - Assessment: ?

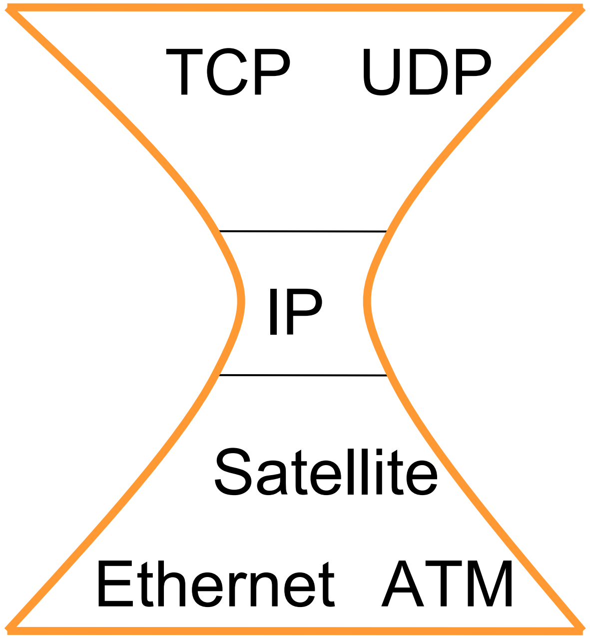
- Accountability
  - Assessment: ?

What About the Future?

- Datagram not the best abstraction for:
  - resource management, accountability, QoS
- new abstraction: flow (see IPv6)
  - Typically: (src, dst, #bytes) tuple
  - But: “flow” not precisely defined
    - when does it end? Explicit connection teardown? Timeout?
    - *src* and *dst* =...? ASes? Prefixes? Hosts? Hosts&Protocol?
  - IPv6: difficulties to make use of flow IDs
- routers require to maintain per-flow state
- state management: recovering lost state is hard
- in context of Internet (1988) we see the first proposal of “soft state”!
  - soft-state: end-hosts responsible to maintain the state

Summary: Internet Architecture

- packet-switched datagram network
- IP is the glue (network layer overlay)
- IP hourglass architecture
  - all hosts and routers run IP
- stateless architecture
  - no per flow state inside network



IP hourglass

Summary: Minimalist Approach

- Dumb network
  - IP provide minimal functionalities to support connectivity
  - addressing, forwarding, routing
- Smart end systems
  - transport layer or application performs more sophisticated functionalities
  - flow control, error control, congestion control
- Advantages
  - accommodate heterogeneous technologies (Ethernet, modem, satellite, wireless, ...)
  - support diverse applications (telnet, ftp, Web, X windows)
  - decentralized network administration

But that was yesterday

……. what about tomorrow?

Rethinking Internet Design

What's changed?

- operation in untrustworthy world
  - endpoints can be malicious: Spam, Worms, (D)DoS, ...
  - If endpoint not trustworthy, but want trustworthy network
    ⇨ more mechanisms in network core

- more demanding applications
  - end-to-end best effort service not enough
  - new service models in network (IntServ, DiffServ)?
  - new application-level service architecture built on top of network core (e.g., CDN, P2P)?

Rethinking Internet Design

What's changed (cont.)?

- ISP service differentiation
  - ISP doing more (than other ISPs) in core is competitive advantage
- Rise of third party involvement
  - interposed between endpoints (even against will)
  - e.g., Chinese government, recording industry, Vorratsdatenspeicherung
- less sophisticated users

All five changes motivate shift away from end-to-end!

What's at stake?

"At issue is the conventional understanding of the "Internet philosophy"

- freedom of action
- user empowerment
- end-user responsibility for actions taken
- lack of control *"in"* the net that limit or regulate what users can do

The end-end argument fostered that philosophy because they enable the freedom to innovate, install new software at will, and run applications of the users choice."

[Blumenthal and Clark, 2001]

Technical response to changes

- Trust: emerging distinction between what is “in” network (*us*, trusted) and what is not (*them*, untrusted).
  - ingress filtering
  - emergence of Internet UNI (user network interface, as in ATM)?
- Modify endpoints
  - harden endpoints against attack
  - endpoints/routers do content filtering: Net-nanny
  - CDN, ASPs: rise of structured, distributed applications in response to inability to send content (e.g., multimedia, high bw) at high quality

Technical response to changes

- Add functions to the network core:
  - filtering firewalls
  - application-level firewalls
  - NAT boxes
  - active networking

… All operate within network, making use of application-level information

  - which addresses can do what at application level?
  - If addresses have meaning to applications, NAT must "understand" that meaning

Epilogue: will IP take over the world?

- Reasons for success of IP:
  - *reachability:* reach every host; adapts topology when links fail.
  - *heterogeneity:* single service abstraction (best effort) regardless of physical link topology

- many other claimed (or commonly accepted) reasons for IP's success may not be true

  …. let's take a closer look

1. IP already dominates global communications?

- business revenues (in US$, 2007):
  - ISPs: 13B
  - Broadcast TV: 29B
  - Cable TV: 29.8B
  - Radio broadcast: 10.6B
  - Phone industry: 268B
- Router/telco switch markets:
  - Core router: 1.7B; edge routers: 2.4B
  - SONET/SDH/WDM: 28B, Telecom MSS: 4.5B

Q: IP equipment cheaper?
Economies of scale?
(lots of routers?)

Q: per-device, IP is cheaper
(one line into house, multiple devices)

Q: # bits carried in each network?

Q: Internet, more traffic and congestion is spread among all users (bad?)

2. IP is more efficient?

- Statistical multiplexing versus circuit switching
- Link utilization:
  - Avg. link utilization in Internet core: 3% to 30% (ISPs: never run above 50%!)
  - Avg. utilization of Ethernet is currently 1%
  - Avg. link utilization of long distance phone lines: 33%
- low IP link utilization: purposeful!
  - predictability, stability, low delay, resilience to failure
  - at higher utilization: traffic spikes induce short congestion periods → deterioration of QoS

- At low utilization, we loose benefits of statistical multiplexing!

3. IP is more robust?

- "Internet was built to sustain a nuclear war" — marketing vapor!
  - Remember large-scale network outages, e.g. on Sep 11th 2001?

- Median IP network availability: downtime: 471 min/yr
- Avg. phone network downtime: 5 min/yr

- Convergence time with link failures:
  - BGP: ≈ 3–15 min,
    intra-domain: ≈ 0.1–1 s (e.g., OSPF)
  - SONET: 50 ms

- Inconsistent routing state
  - human misconfigurations
  - in-band signaling (signaling and data share same network)
  - routing computation "complex"

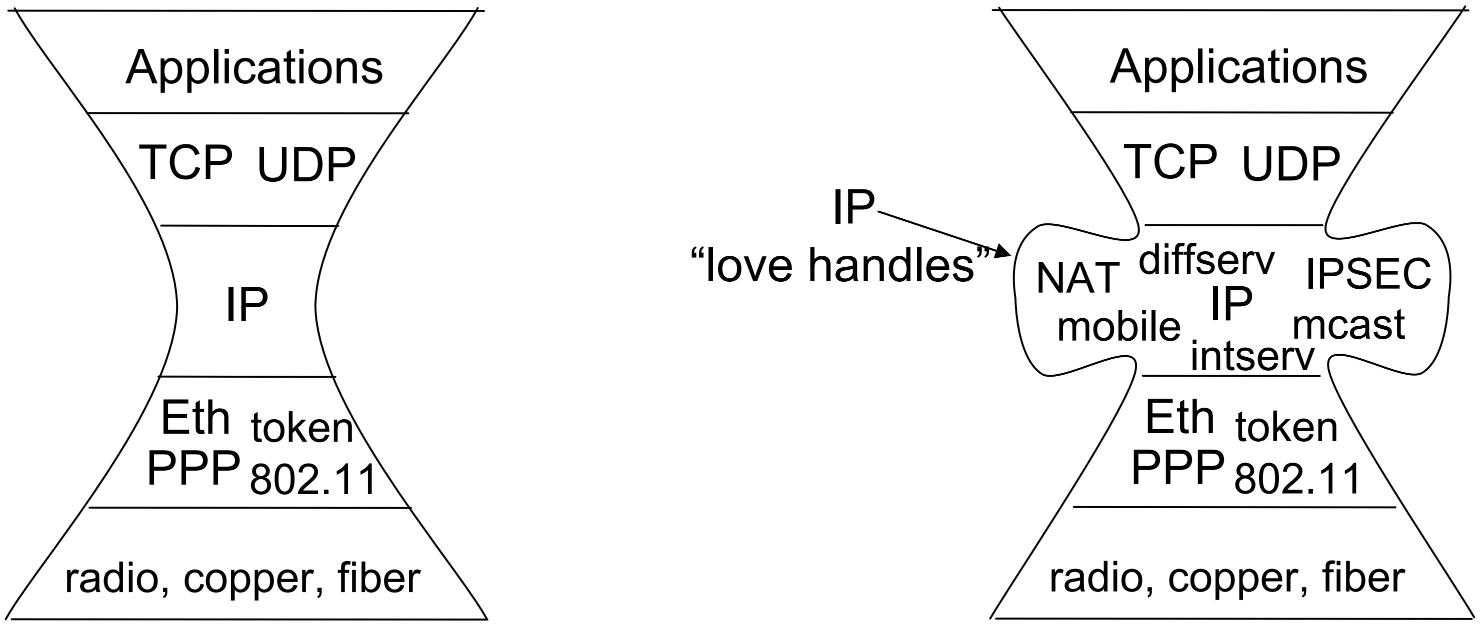
4. IP is simpler?

- Intelligence at edge, simplicity in core
  - Cisco IOS: 8M lines of code
  - Telephone switch: 3M lines of code

- Linecard complexity:
  - Router: 30M gates in ASICs, 1 CPU, 300M packet buffers
  - Switch: 25% of gates, no CPU, no packet buffers

Discussion: benefits of IP?

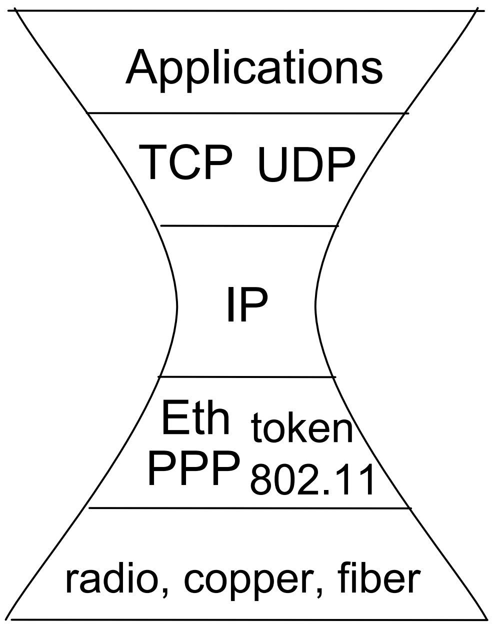
Big picture: supporting new applications – losing the IP hour glass figure?



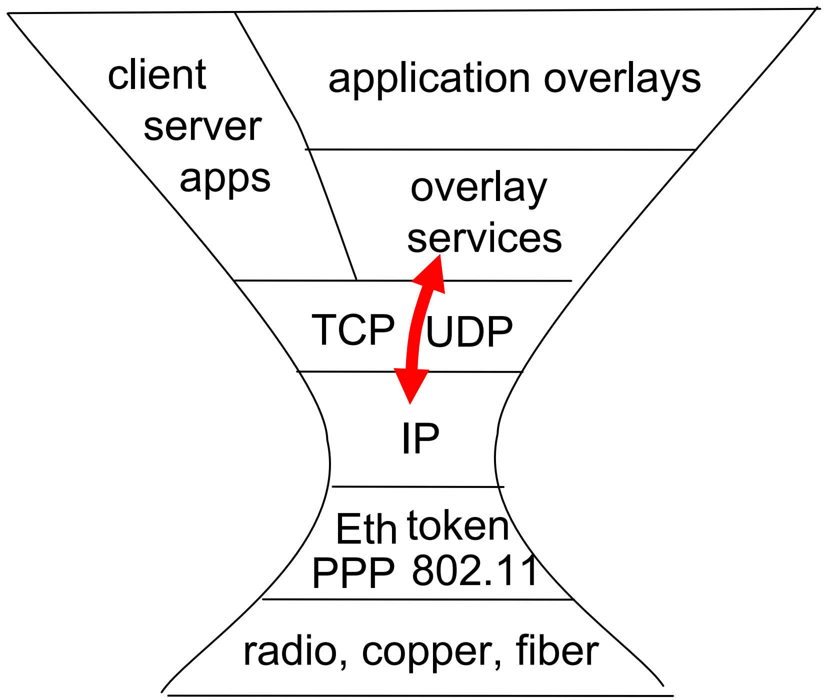
IP "hourglass"

Middle-age IP "hourglass" ?

Big picture: supporting new applications – losing the IP hour glass figure?



IP “hourglass”



Some advice on protocol design

- A loose collection of important thoughts related to protocol design
- ... actually, not only protocol design, but also
  - Programming in general
  - Systems in general (e.g., workflows in companies)
  - Life :)

Thought-triggering questions (1)

What problem am I trying to solve?

- have at least one ***well-defined*** problem in mind
- solve other problems without complicating solution?

Will my solution scale?

- Think about what happens if you're successful: protocol is used by millions
- Does the protocol make sense in small situations as well?

Thought-triggering questions (2)

How “robust” is my solution?

- adapt to failure/change
  - self-stabilization: eventually adapt to failure/change
  - Byzantine robustness: will work in spite of malicious users
- What are the underlying assumptions?
  - What if they are not true? catastrophe?
- maybe better to crash than degrade when problems occur: signal problem exists
- techniques for limited spread of failures
- protocol should degrade gracefully in overload, at least detect overload and complain

Further thoughts

Forward compatibility

- think about future changes, evolution
- make fields large enough
- reserve some spare bits
- specify an options field that can be used/augmented later

Parameters...

- Protocol parameters can be useful
  - designers can’t determine reasonable values
  - tradeoffs exist: leave parameter choice to users
- Parameters can be bad
  - users (often not well informed) will need to choose values
  - try to make values plug-and-play

Simplicity vs Flexibility versus optimality

- Is a more complex protocol reasonable?
- Is “optimal” important?
- KISS: “The simpler the protocol, the more likely it is to be successfully implemented and deployed.”
- 80:20 rule:
  80% of gains achievable with 20% of effort

Why are protocols overly complex?

- design by committee
- backward compatibility
- flexibility: heavyweight swiss army knife
- unreasonble stiving for optimality
- underspecification
- exotic/unneeded features

Trading accuracy for time

- If computing the exact result is too slow, maybe an approximate solution will do
  - optimal solutions may be hard: heuristics will do (e.g., optimal multicast routing is a Steiner tree problem)
  - faster compression using "lossy" compression
    - lossy compression: decompression at receiver will not exactly recreate original signal

- Real-world examples?
  - games like chess: can't compute an exact solution

Don’t confuse specification with implementation

- A general problem of computer scientists!
- Specifications indicate external effects/interaction of protocol.
- How protocol is implemented is up to designer
- Programming language specifications: in addition to specifying *what*, tend to suggest *how*.

- real-world example: recipe
    1. Cut onions
    2. Cut potatoes
    3. Put onion and potatoes into pot and boil

  steps 1 and 2 can obviously be interchanged……

Network Simulation

Network Simulation

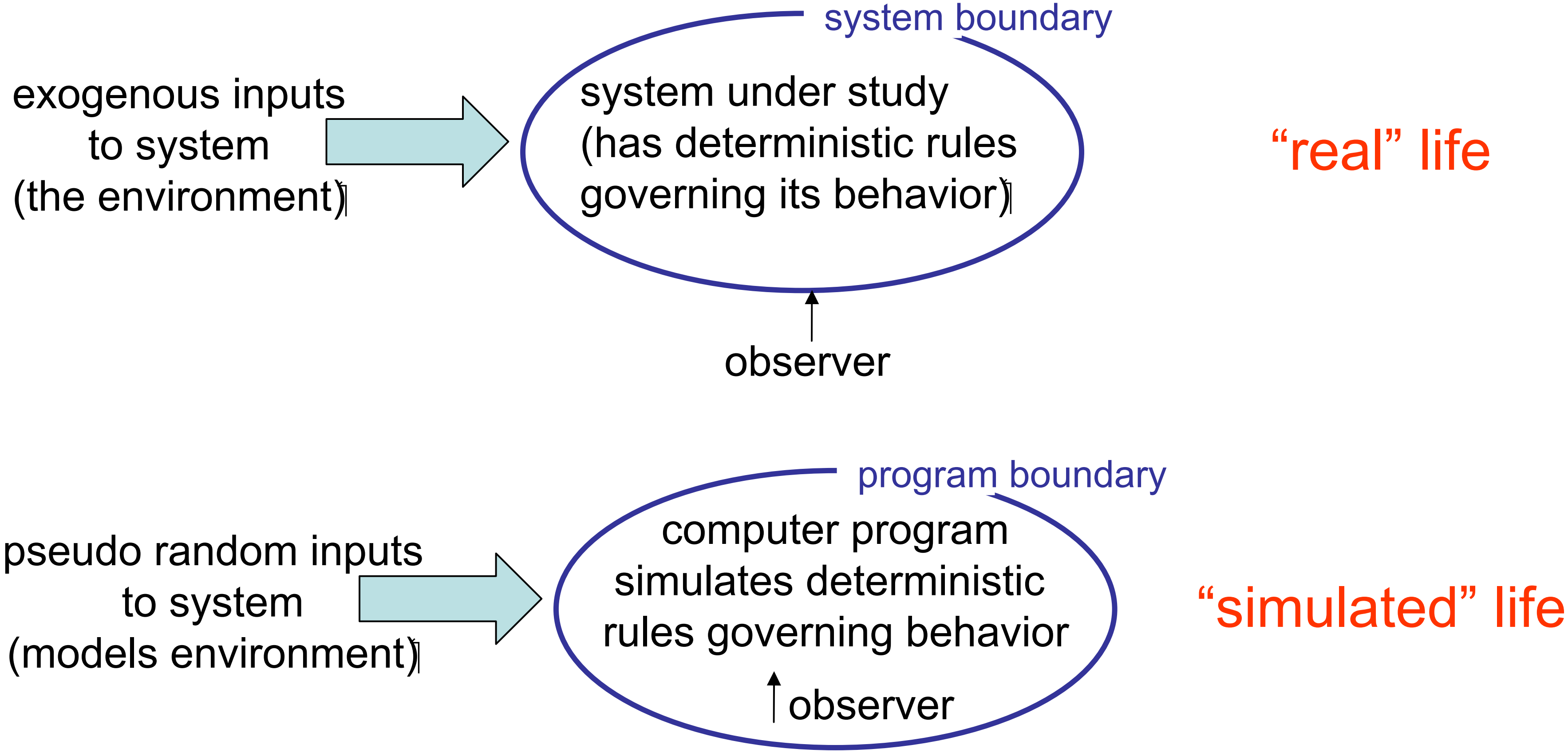
Motivation:

- Learn fundamentals of evaluating network performance via simulation

Overview:

- fundamentals of discrete event simulation
- analyzing simulation outputs

What is simulation?



Why Simulation?

- *goal:* study system *performance, operation*
- real-system not *available, is complex/costly or dangerous*
  (e.g.: space simulations, flight simulations, network with 1000s of routers)
- quickly evaluate design *alternatives*
  (e.g.: different system configurations)
- evaluate *complex functions* for which closed form formulas or numerical techniques not available

Simulation: Advantages/Drawbacks

- advantages:
  - save lives, money
  - find bugs (in design!) in advance
  - generality: over analytic/numerical techniques
  - detail: can simulate system details at arbitrary level
- drawbacks:
  - caution: does model reflect reality?
  - large scale systems: lots of resources to simulate (especially accurrately simulate)
  - may be slow (computationally expensive – 1 min real time could be hours of simulated time)
  - art: determining right level of model complexity
  - statistical uncertainty in results

The Evaluation Spectrum

- Numerical models
- **Simulation**
- Emulation
- Prototype
- Operational system

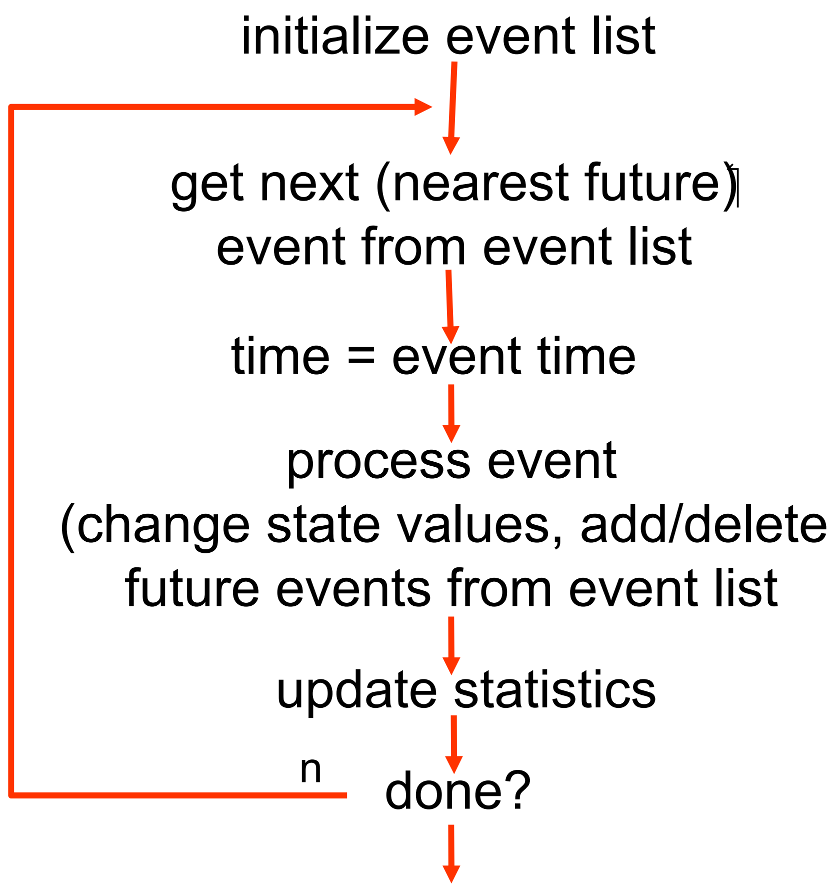
Programming a Simulation

What's in a simulation program?

- *simulated time:* internal (to simulation program) variable that keeps track of simualted time
- *system "state":* variables maintained by simulation program define system "state"
  - e.g., may track number (possibly order) of packets in queue, current value of retransmission timer
- *events:* points in time when system changes state
  - each event has associate *event time*
    - e.g., arrival of packet to queue, departure from queue
    - precisely at these points in time that simulation must take action (change state and may cause new future events)
  - model for time between events (probabilistic) caused by external environment

Discrete Event Simulation

- simulation program maintains and updates list of future events: event list
- simulator structure:



Need:

- well defined set of events
- for each event: simulated system action, updating of event list

Simulation time, real time

- Real time (CPU time) does not depend on simulated time, but:
  - # of events to process:
    No difference if 1 ms or 1 year between two subsequent events
  - computational cost to process an event:
    e.g., forwarding IP packet at router a lot easier than
    receiving TCP segment that finishes an HTTP request
- How does it scale?
  - # events linear with # involved routers/switches (path length)
  - # events linear with # end hosts producing workload packets
  - ***May*** be super-linear with # of nodes! (depends on topology)

Simulators can “cheat”

- Simulate every bit and byte in a packet?
  - Time consuming
  - Level of detail really needed?
- Probably not!
  - Packet *content* doesn’t matter for transmission delay, link delay, queueing delay
  - # of bytes in packet → time for sending this packet (transmission delay)
- Capture additional data in simpler form
  - e.g., store packet content conveniently as Java/C++/... object → no need to pack/unpack, encode/decode, parse, ...
  - “Cheating”? — Only if you simulate the impossible, e.g., entire Wikipedia in one Ethernet packet

Simulator must be trustworthy!

- "In our simulations, the plane wing never broke off..."
- Specify correct behaviour *in advance*
  - Which aspects do we want to simulate
  - Which ones not?
- Devise test scenarios, perform test simulations to check correct implementation of protocol(s)
- Important implementation rule:
  If illegal state is reached, print debug message and abort simulation
  (Recovery attempts do not make sense!)

Analyzing Output Results

- Each time we run the simulation, we will get different output results!
- (... only if we use different random number seeds each time)
  (... which we should do!)

distribution of random numbers
to be used during simulation
(interarrival, service times)

$\downarrow$

random number sequence 1 $\xrightarrow{\text{input}}$ simulation $\xrightarrow{\text{output}}$ output results 1

random number sequence 2 $\xrightarrow{\text{input}}$ simulation $\xrightarrow{\text{output}}$ output results 2

... ...

random number sequence M $\xrightarrow{\text{input}}$ simulation $\xrightarrow{\text{output}}$ output results M

Setting up a simulation

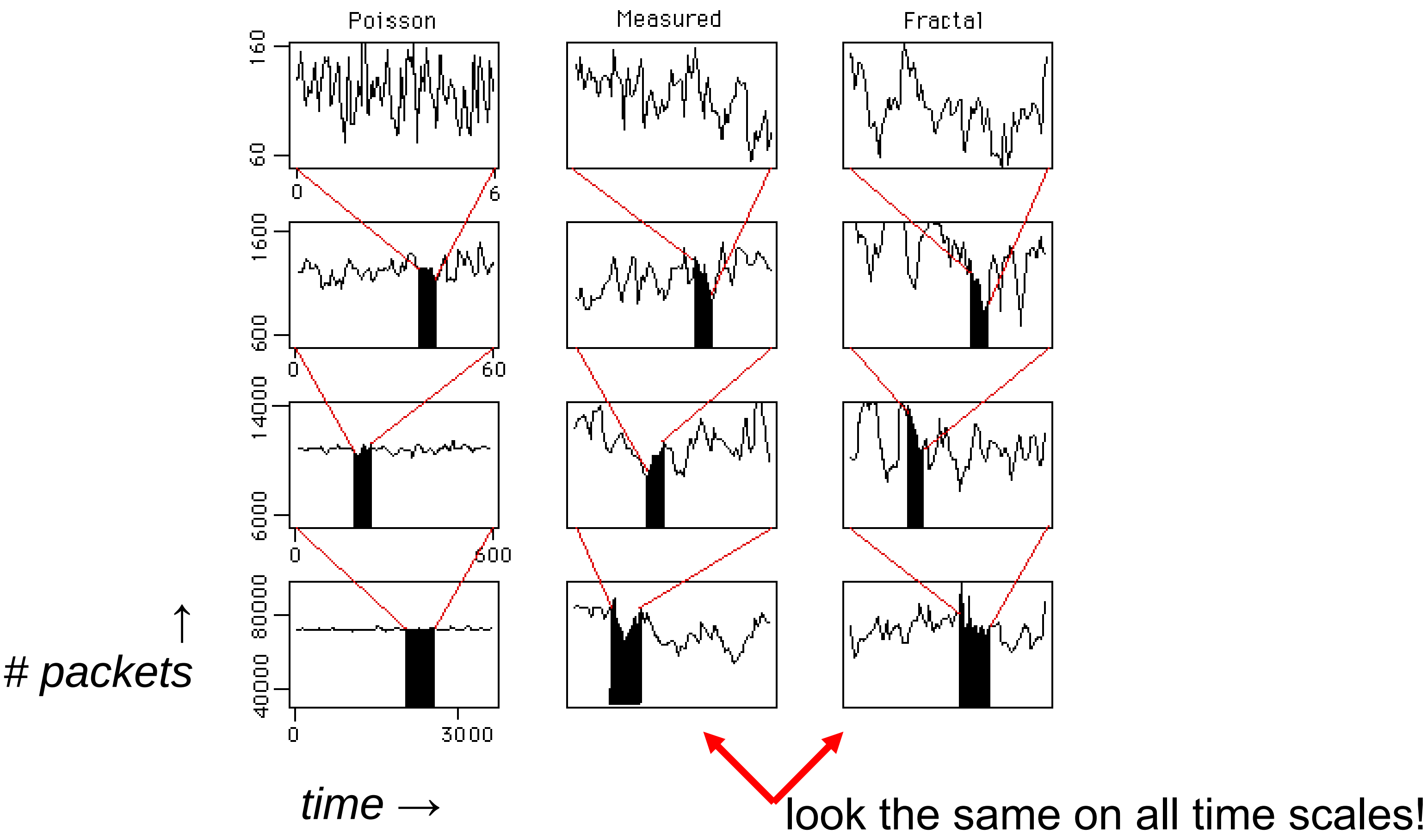
- Always in this order:
  - *What* do I want to show?
  - *How*?
- Which are important parameters, which are not?
- Realistic simulation set-up vs. computation time
  - network size: # of end hosts, # of routers
  - network topology
  - link speeds, queue lengths
  - simulation duration
  - workload / background noise
    - statistic generator?
    - replay `tcpdump` traces from real network?

Analyzing simulation results

1. Devise simulation model
2. Implement model description for simulator program
3. Run simulator program
   - ...and again (with different random seed)
   - ...and again, etc. etc.
4. “Average” different outputs (← Gigabytes and more!)
5. Statistical analysis of simulator output(s)
6. Swear and go back to step 1 or 2

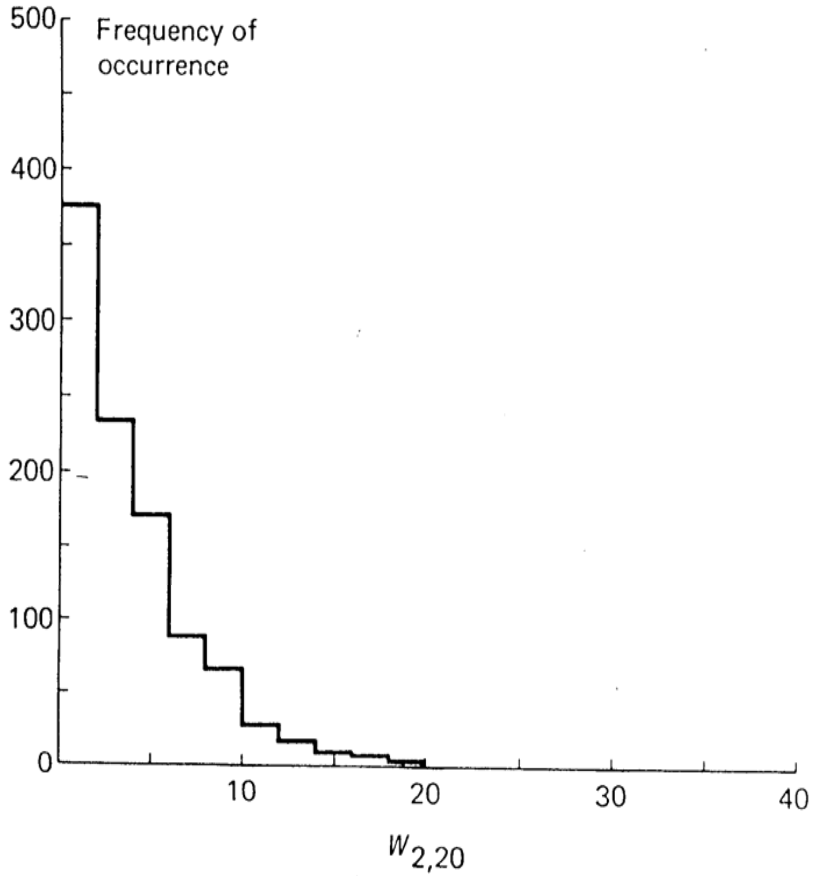
Speaking of “realistic traffic”

- Internet traffic: very bursty, frequent statistical spikes → nasty!
- Consistent with self-similar behaviour:

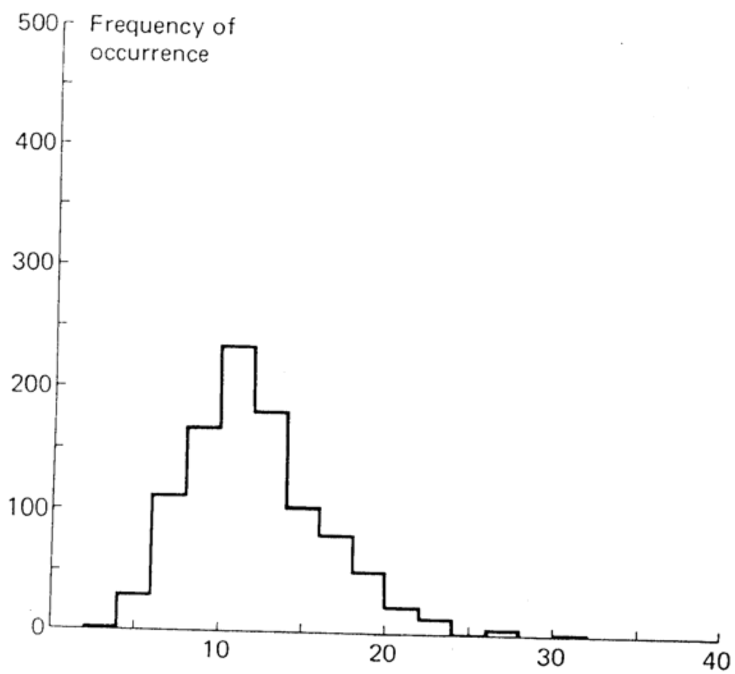


Effect of initial conditions

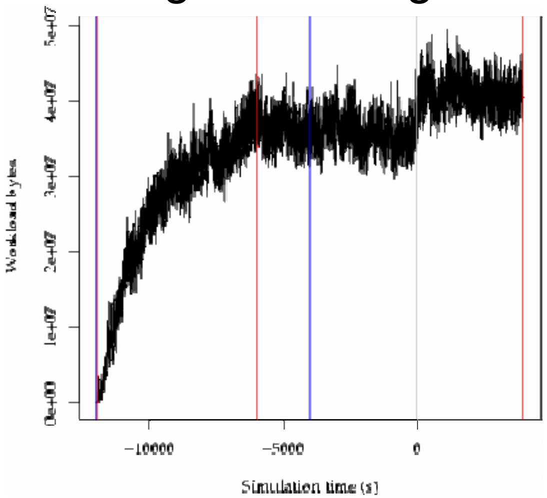
- Histogram of delay of 20th customer, given initially empty (1000 runs)



- Histogram of delay of 20th customer, given non-empty conditions



- Output results may converge to limiting “steady state” value if simulation run “long enough”



- Want to discard statistics gathered during transient phase, e.g., ignore first n_0 measurements:

$$T_i = \frac{\sum_{j=n_0}^{N_i} D_{ij}}{N_i - n_0}$$

Pick n_0 so that statistic is “approximately the same” for different random number streams and remains same as n increases

Confidence Intervals

- run simulation: get estimate V_1 as estimate of performance metrics of interest
- repeat simulation M times (each with new set of random numbers), get V_2, … V_M — *all different!*
- which of V_1, … V_M is "right"?

- intuitively, average of M samples should be "better" than choosing any one of M samples:

$$\bar{V} = \frac{\sum_{j=1}^{M} V_j}{M}$$

How "confident" are we in $\bar{V}$?

Confidence Intervals

- Can not get perfect estimate of true mean, m, with finite # samples
- Look for bounds: find c1 and c2 such that:

  Probability(c1 < m < c2) = 1 – α

  [c1,c2]: *confidence interval*

  100(1-α): *confidence level*
- One approach for finding c1, c2 (suppose α=0.1)
  - take k samples (e.g., k independent simulation runs)
  - sort
  - find largest value is smallest 5% → c_1
  - find smallest value in largest 5% → c_2

Confidence Intervals: Central Limit Theorem

- Central Limit Theorem: If samples $V_1, \ldots V_M$ independent (e.g., having repeated same simulation M times with different random numbers, and taken the average each time) and from same population with population mean m and standard deviation s, then

sample mean: $$\bar{V} = \frac{\sum_{j=1}^{M} V_j}{M}$$

is approximately normally distributed with mean u and standard deviation $\frac{\sigma}{\sqrt{M}}$

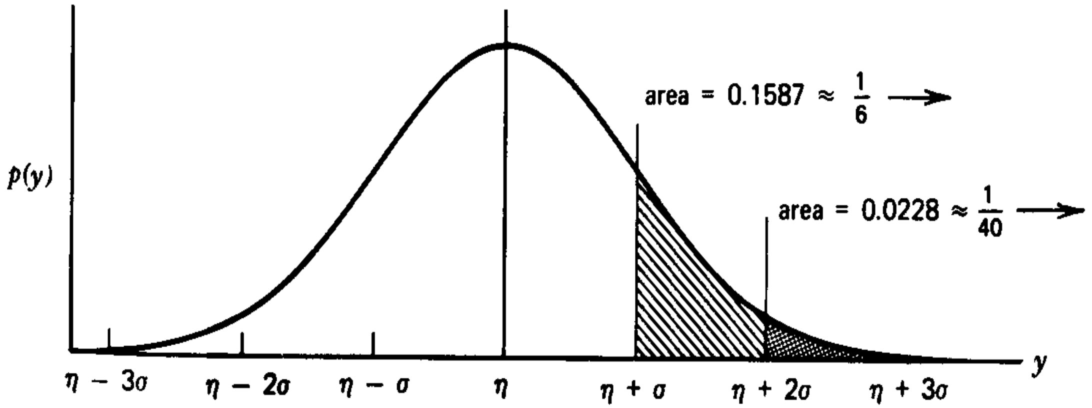


FIGURE 2.12. Tail areas of the normal distribution.

Confidence Intervals .. more

- Still don't know population standard deviation. So we estimate it using sample (observed) standard deviation:

$$\sigma_v^2 = \frac{1}{M-1}\sum_{m=1}^{M}\left(V_m - \bar{V}\right)^2$$

- Given $\bar{V}, \sigma_v^2$ we can now find upper and lower tails of normal distributions containing α · 100% of the mass

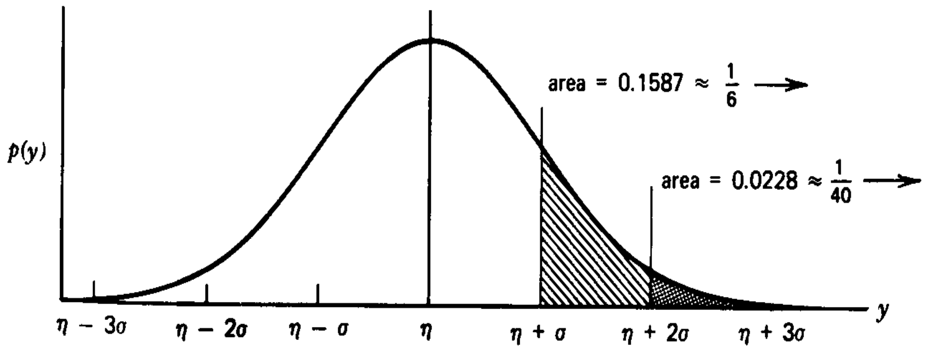


FIGURE 2.12. Tail areas of the normal distribution.

Confidence Intervals .. the recipe

- Given samples $V_1, \dots V_M$, (e.g., having repeated simulation M times), compute

$$\bar{V} = \frac{\sum_{j=1}^{M} V_j}{M}$$

$$\sigma_{v^2} = \frac{1}{M-1} \sum_{m=1}^{M} \left(V_m - \bar{V} \right)^2$$

95% confidence interval: $\bar{V} \pm \frac{1.96 \sigma_v}{\sqrt{M}}$

Interpretation of Confidence Interval

- If we calculate confidence intervals as in recipe, 95% of the confidence intervals thus computed will contain the true (*unknown!*) population mean.

- Actually, a bit more complex maths than shown:
  t distribution,
  χ^2 distribution, ...

  Here: Use large M > 30 to be on safe side

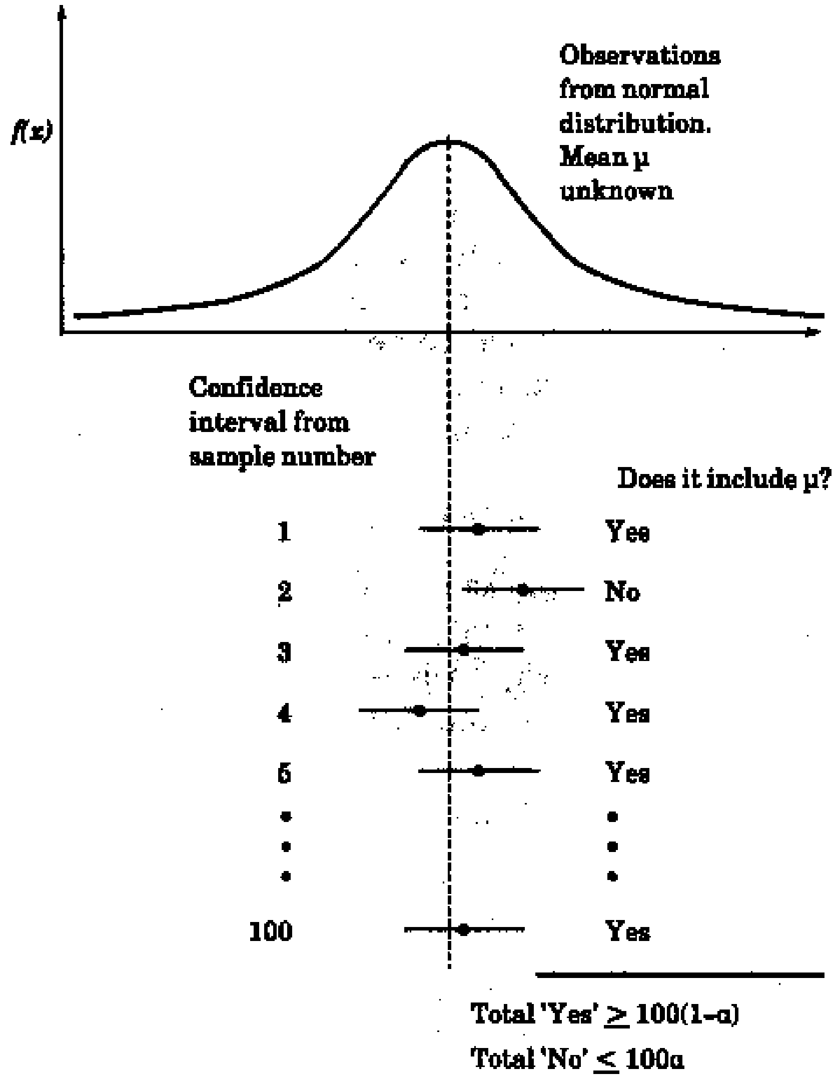


FIGURE 13.1 Meaning of a confidence interval.

Roundup: Can you trust their simulation result?

Given a paper containing simulation-based evaluation, can you trust it?

- Realistic network set-up? (size, topology, speed, ...)
- Realistic traffic? (keywords: self-similar, fractal, heavy-tailed, bursty)
- Statistic relevance?
  - Simulation duration
  - Number of independent simulation runs
  - Confidence intervals