Architecture: The big picture

Goals:

- Identify, study principles that can guide network architecture
- "Bigger" issues than specific protocols or implementation tricks
- *Synthesis:* The *really* big picture

Overview:

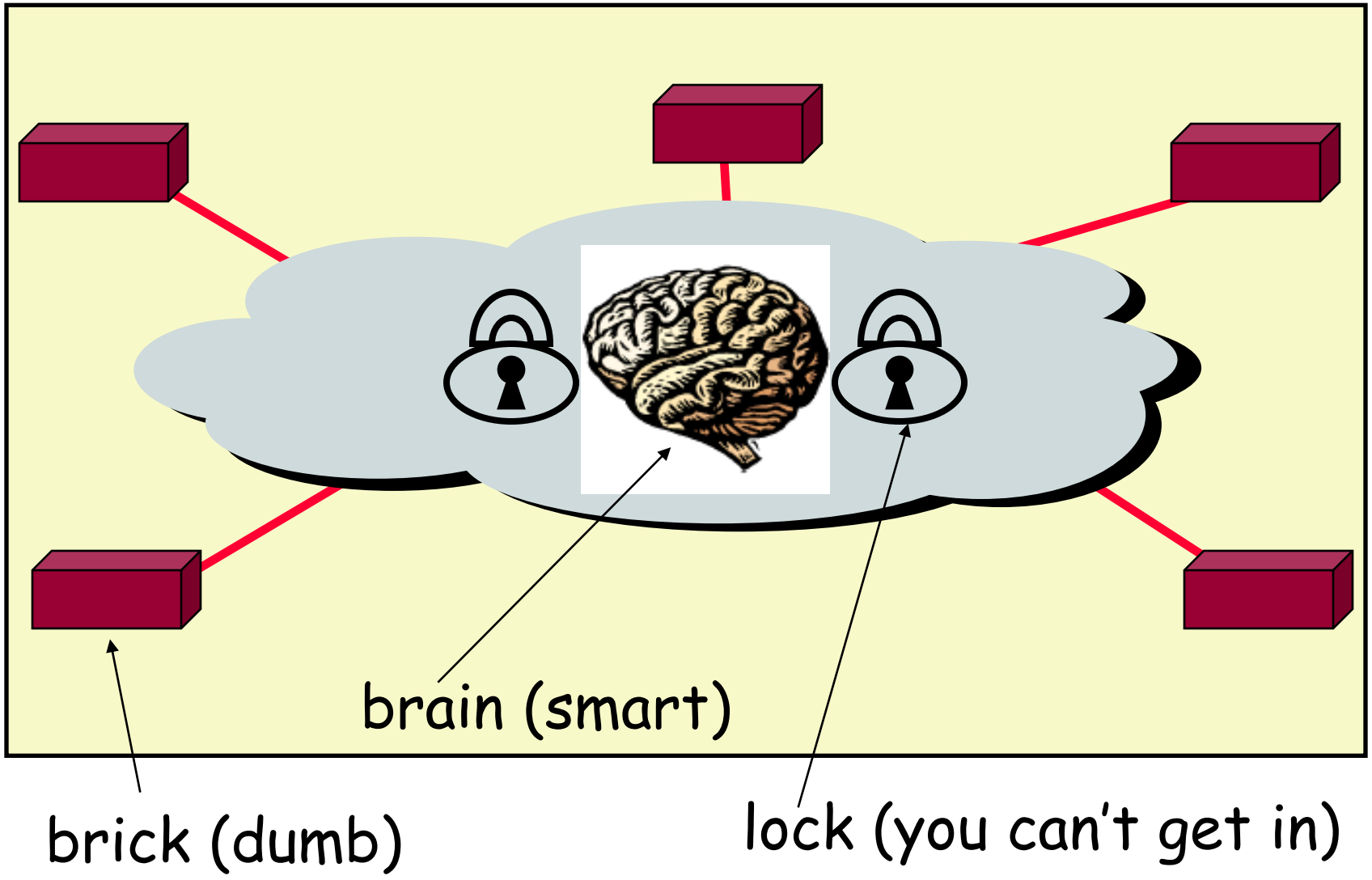
- Telephone network architecture
- Internet design principles
- Rethinking the Internet design principles

Key questions

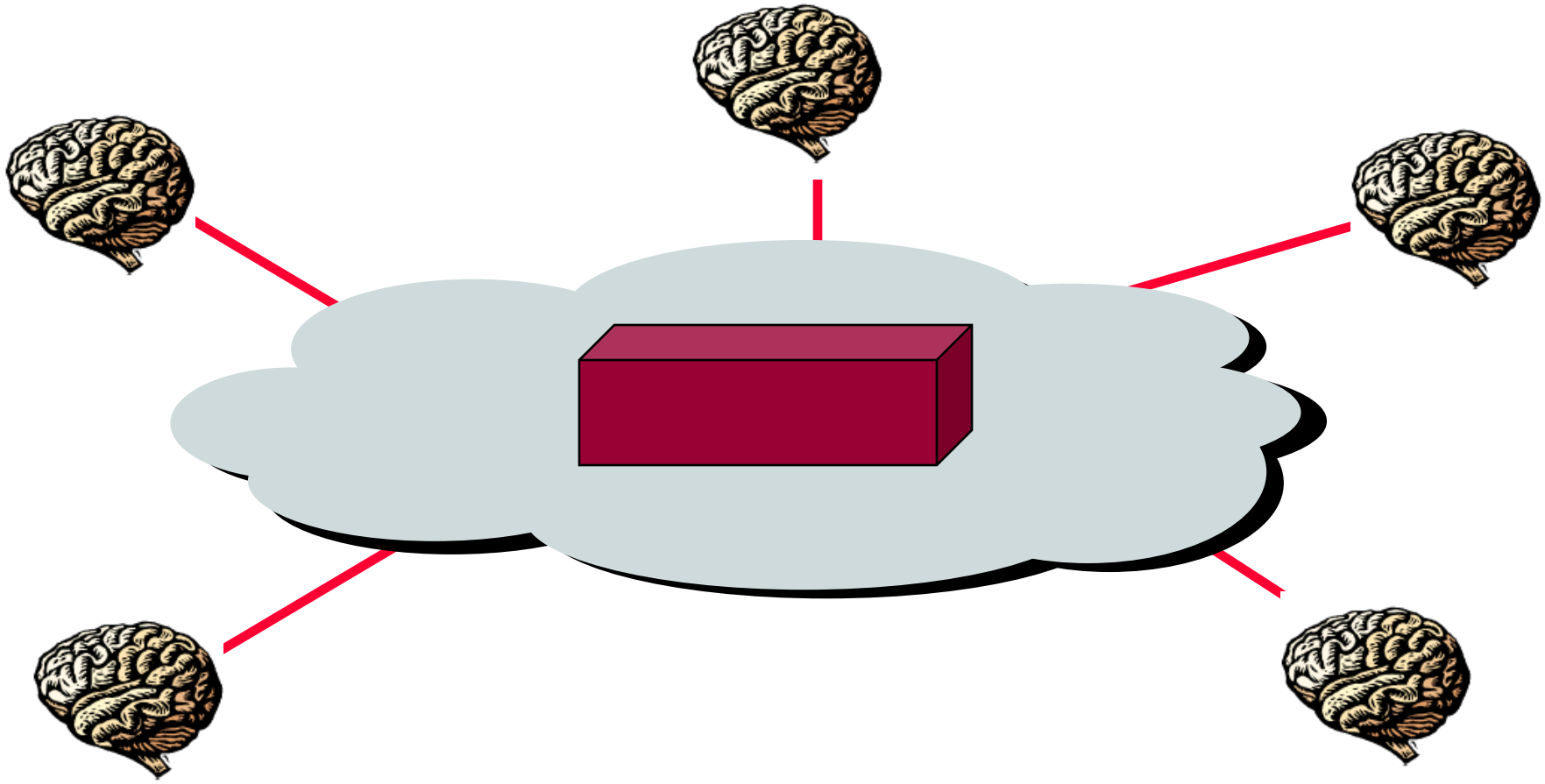
- How to decompose the complex system functionality into protocol layers?
- Which functions placed *where* in network, at which layers?
- Can/should a function be placed at multiple levels ?

Answer these questions in context of Internet, telephone net

Common view of the telephone network



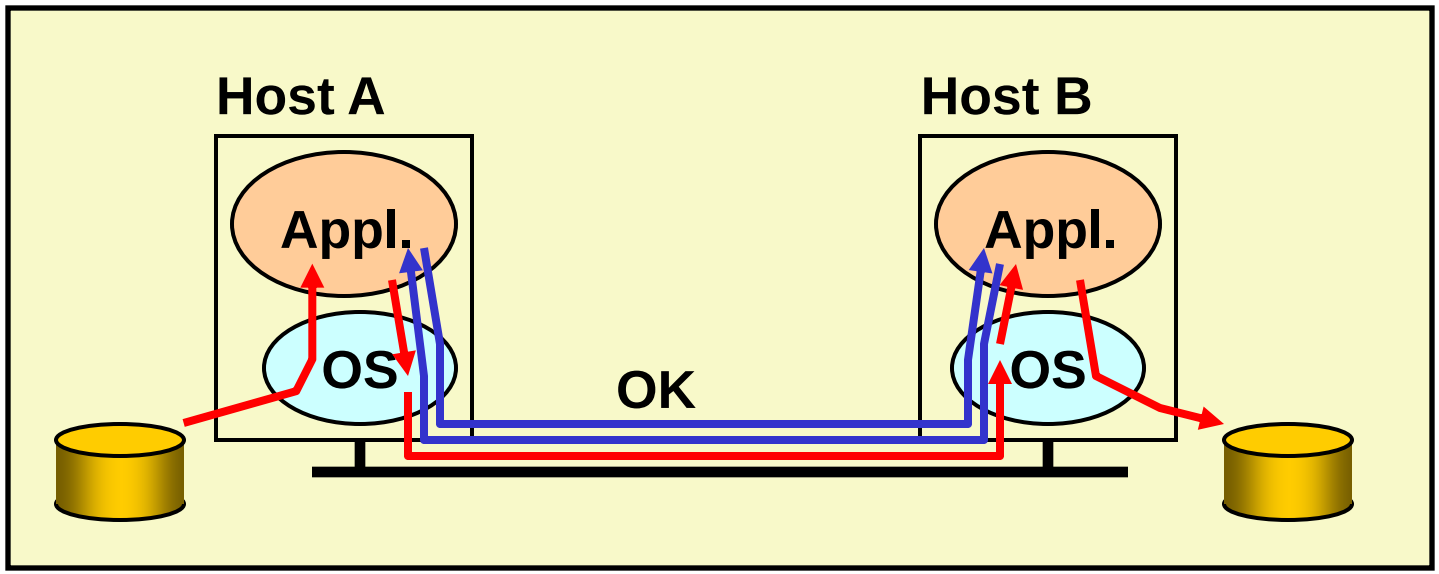
Common view of the IP network



Internet End-to-End argument

- "…functions placed at the lower levels may be *redundant* or of *little value* when compared to the cost of providing them at the lower 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: Make each step unreliable and use end-to-end check and retry

Discussion

- Solution 1 not good enough!
  - What happens if the sender or/and receiver misbehave?
  - The receiver has to do check anyway!

- Thus, full functionality can be entirely implemented at application layer
- *No* need for reliability at lower layers

Discussion

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

A: Yes, but only to improve performance

- Example:
  - Assume high error rate in network
  - Reliable communication service at data link layer might help (why)?
  - Fast detection/recovery of errors

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 (E2E) 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 …
  - Application layer functionality
  - Network services, e.g., DNS

  implemented at application level

Internet & E2E argument (1)

Discussion: Congestion control, flow control

- Why at transport, rather than link or application layers?
  - Claim: Common functions should migrate down the stack
    - Everyone shares same implementation: No need to redo it (reduces bugs, less work, etc…)
    - Knowing everyone is doing the same thing, can help
  - Congestion control too important to leave up to application/user:
    - True but hard to police
    - Tcp is "outside" the network; compliance is "optional"
    - We do this for fairness (but realize that people could cheat)

Internet & E2E argument (2)

Discussion: Congestion control, flow control: Why at transport, rather than link or application layers?

- Why flow control in TCP, not (just) in app
  - Shared TCP buffers at receiver meant to control flow at TCP level (otherwise unfairness)

- Shared resources is an important reason to push controlling functionality to point at which resources are shared
- Corollary: Do active queue management (e.g., RED) in network
  - Question: How much does careful controlled sharing buy you?

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

E2E 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 to lower layers iff
    (1) used by/improves performance of many apps
    (2) does not hurt other applications
  - Allows *cost-performance* tradeoff

E2E argument: Discussion

- End-end argument emphasizes correctness and completeness, not
  - Complexity: Is complexity at edges result in a "simpler" architecture?
  - Evolvability, ease of introduction of new functionality: Ability to evolve because easier/cheaper to add new edge applications than change routers?
  - Technology penetration: Simple network layer makes it "easier" for IP to spread everywhere

Summary: E2E argument

- If the application can do it, don't do it at a lower layer -- anyway the application knows the best what it needs
  - Add functionality in lower layers iff it is
    (1) used and improves performances of a large number of applications
    (2) does not hurt other applications

- Success story: Internet

Internet design philosophy (Clark'88)

In order of importance:

0. Connect existing networks
   - Initially ARPANET and ARPA packet radio network
1. Survivability
   - Ensure communication service even under network/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

Different ordering of priorities may make a different architecture!

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 e2e transport state only at end-points
  - Eliminates problem of handling state inconsistency and performing state restoration when router fails
- Internet: STateless network architecture
  - No notion of a session/call at network layer

Grade: A- because convergence times are relatively slow

- BGP takes minutes to coverge
- IS-IS OSPF take ~ 10 seconds

2. Types of Services

- Add UDP to TCP to better support 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?), but this has never happened on the large scale (Why?)

Grade: A- proven to allows lots of application to be invented and flourish (except MM, but maybe that's not a transport service issue)

3. Variety of Networks

- Very successful (why?)
  - Because of 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, ...
  - Now: ATM, SONET, WDM, …

Grade: A can't name a link layer technology that IP doesn't run over (carrier pigeon RFC)

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
  - Grade: A for implementation, B for concept (disagreement)
- Cost effective
  - Sources of inefficiency
    - Header overhead
    - Retransmissions
    - Routing
  - …but "optimal" performance never been top priority
  - Grade: A

Other goals (2)

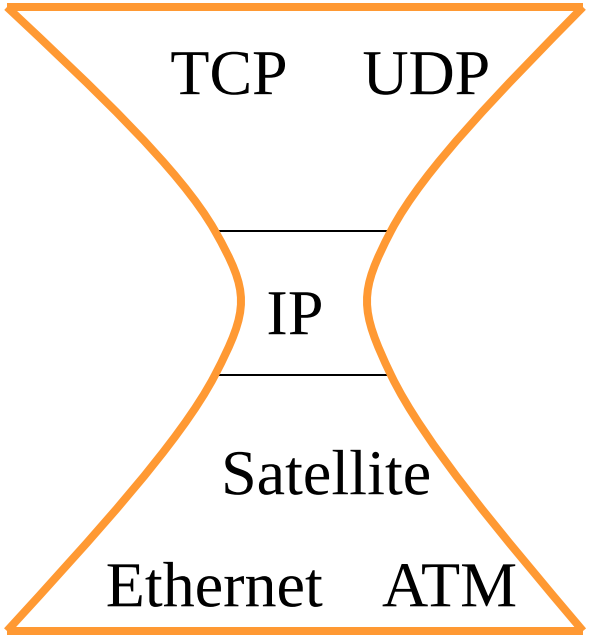
- Low cost of attaching a new host
  - Not a strong point → higher than other architecture because the intelligence is in hosts (e.g., telephone vs. computer)
  - Bad implementations or malicious users can produce considerably harm (remember fate-sharing?)
  - Grade: C but things are improving with dhcp, autoconfigurations. Looks like a higher grade possible some time in the future
- Accountability
  - Internet gets an "F" Grade

What about the future?

- Datagram not the best abstraction for:
  - Resource management, accountability, QoS
- Mew abstraction: Flow (see OpenFlow, IPv6)
  - But no one knows what a flow *is*
- Routers require to maintain per-flow state
- State management:
  - Recall: Recovering lost state is hard
  - Here we see proposals for "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 system
  - 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
  - If endpoint not trustworthy, but want trustworthy network -> more mechanism in network core
- More demanding applications
  - End-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 (2)

What's changed?

- ISP service differentiation
  - ISP doing more (than other ISPs) in core maybe a competitive advantage
- Rise of third party involvement
  - Interposed between endpoints (even against will)
  - E.g., Chinese gov't, US recording industry
- Less sophisticated users

All five changes may motivate shift away from end-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 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 (2)

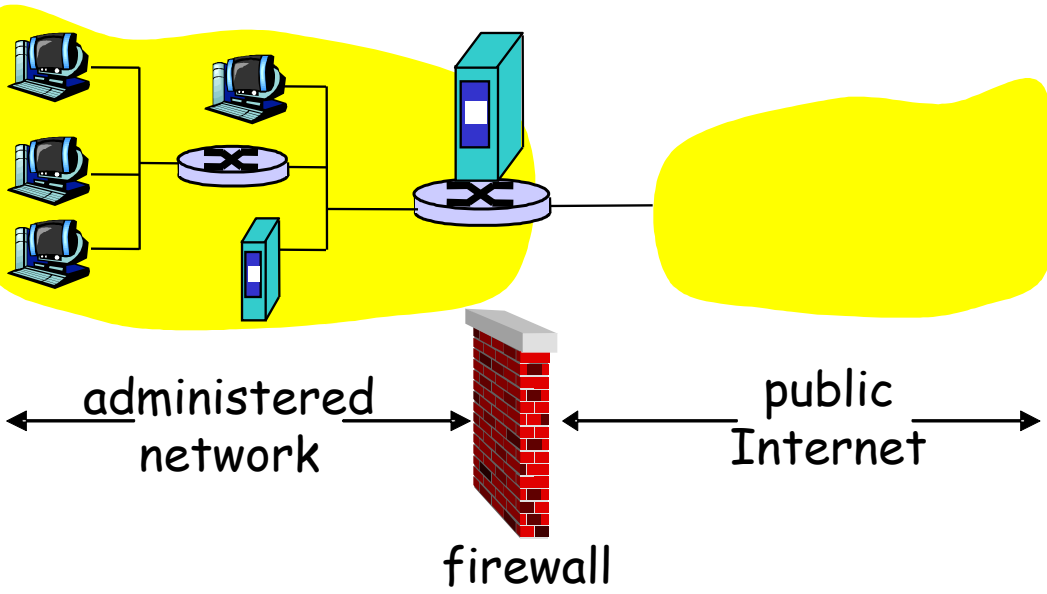
- Add functions to the network core:
  - Filtering firewalls
  - Application-level firewalls
  - NAT boxes
  - Active networking
  - Network virtualization

… 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

Firewalls

firewall

Isolates organization's internal net from larger Internet, allowing some packets to pass, blocking others.



Firewalls: Why

Prevent denial of service attacks:

- SYN flooding: Attacker establishes many bogus TCP connections, no resources left for "real" connections.

Prevent illegal modification/access of internal data.

- e.g., attacker replaces CIA's homepage with something else

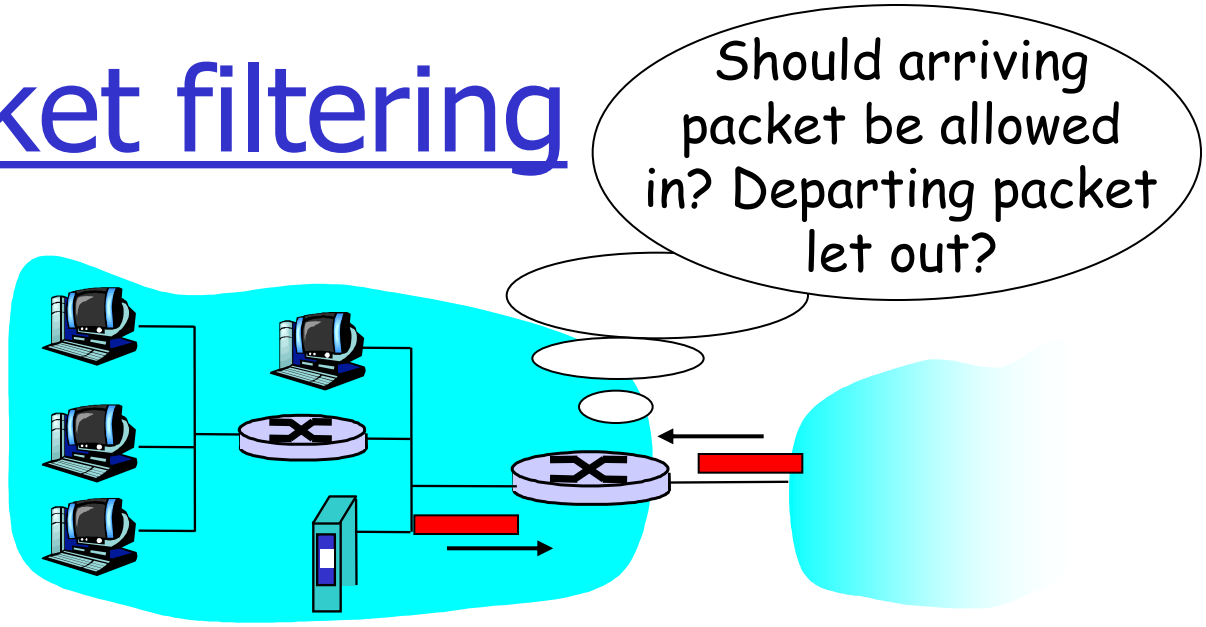
Allow only authorized access to inside network

- (set of authenticated users/hosts)

Two types of firewalls:

- Application-level
- Packet-filtering (stateless/stateful)

Packet filtering

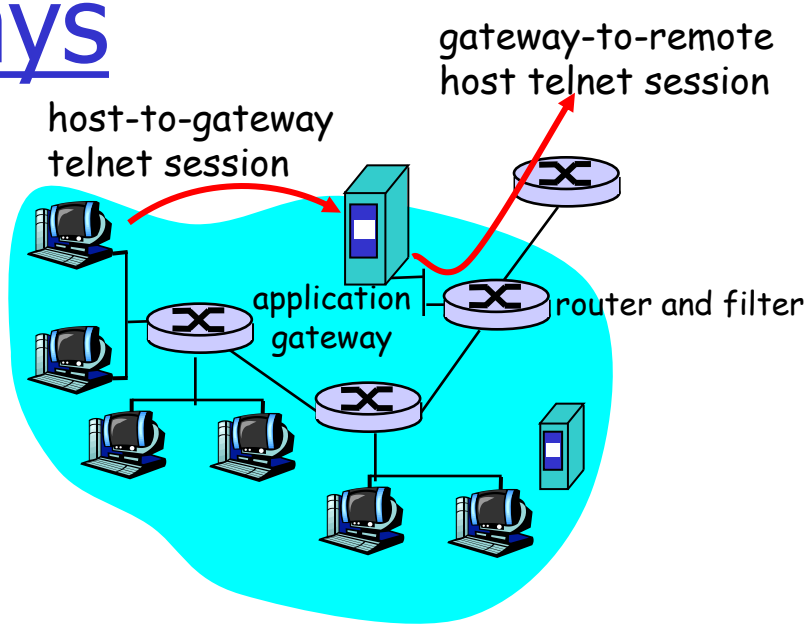


- Internal network connected to Internet via router firewall
- Router filters packet-by-packet, decision to forward/drop packet based on:
  - Source IP address, destination IP address
  - TCP/UDP source and destination port numbers
  - ICMP message type
  - TCP SYN and ACK bits

Packet filtering (2)

- Example 1: Block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23.
  - All incoming and outgoing UDP flows and telnet connections are blocked.
- Example 2: Block inbound TCP segments with ACK=0.
  - Prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside.

Application gateways



- Filters packets on application data as well as on IP/TCP/UDP fields.
- Example: allow select internal users to telnet outside.

1. Require all telnet users to telnet through gateway.
2. For authorized users, gateway sets up telnet connection to dest host. Gateway relays data between 2 connections
3. Router filter blocks all telnet connections not originating from gateway.

NAT: Network Address Translation

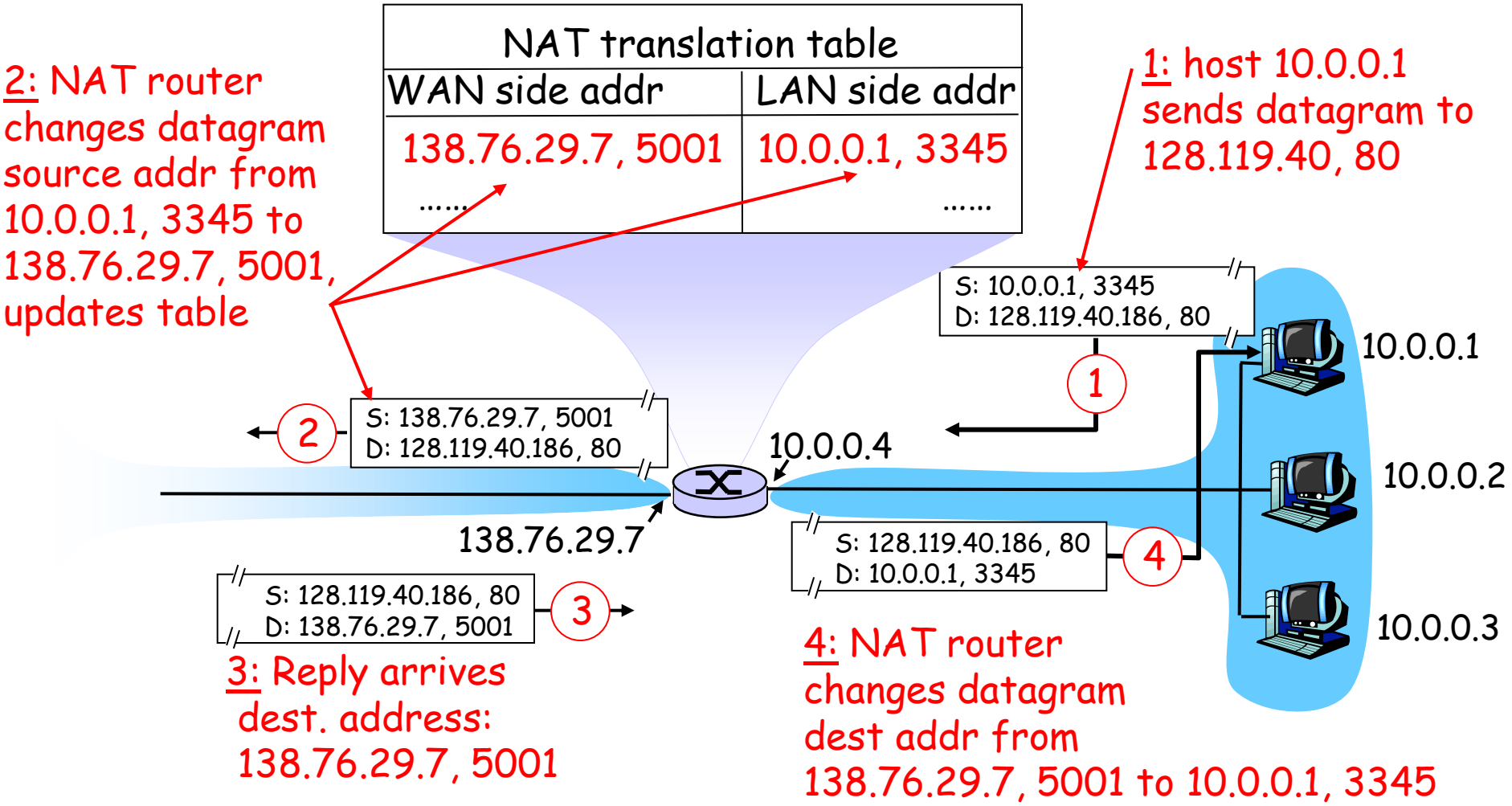
- Motivation: Local network uses just one IP address as far as outside word is concerned
  - No need to be allocated range of addresses from ISP: just one IP address is used for all devices
  - Can change addresses of devices in local network without notifying outside world
  - Can change ISP without changing addresses of devices in local network
  - Devices inside local net not explicitly addressable, visible by outside world (a security plus)

NAT: Network Address Translation

Implementation: NAT router must:

- *Outgoing datagrams: Replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.
- *Remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *Incoming datagrams: Replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: Network Address Translation (2)



NAT: Network Address Translation (3)

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - Routers should only process up to layer 3
  - Violates end-to-end argument
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - Address shortage should instead be solved by IPv6

What is an "Active Network"?

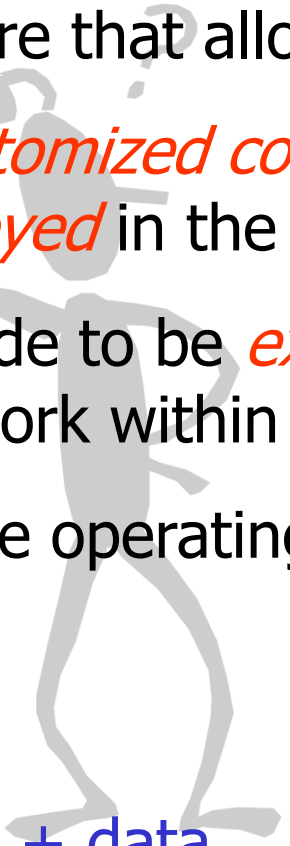
- Depends on who you ask!
- Active services: Application-level services exploiting position within the network to provide enhanced service
  - CDN
  - Streaming media caches
- Capsule approach: Packets carry programs, active node executes program when code-carrying packet arrives to active node
  - Code may determine what to do with packet
  - May implement other service: e.g., network management, reliable multicast

The capsule approach to active networks

- Network architecture that allows :
  - *Application-customized code to be dynamically deployed* in the network
  - Customized-code to be *executed* in controlled framework within network
- Similar to extensible operating systems (SPIN, Synthesize etc)
- The new equation:

  Packet = Code + data

  sort of like postscript



Capsules

- Type
  - Identifier for the forwarding routine to be executed (carries code by reference)
- Previous address
  - Where to get the forwarding routine from if it is not available in the present node (Code Distribution)
- Dependent Fields
  - Parameters for the forwarding code
- Payload
  - Header + data of higher layers

Active networking and E2E arguments

- End-end principle: Lower layers should have minimum functionality, but support widest variety of applications possible
  - Active networking: support *all* higher-level applications
  - Minimum common functionality: Ability to execute code: *Programmable versus pre-programmed* low layer functionality

Active networking: Transparency/efficiency?

- Transparency: Use of network by others not very visible (can more or less predict behavior of network)
- Active networking: Transparency difficult
  - Constrain interactions among programmable entities in router (who knows *what* they will try to do)
  - Like OS trying to constrain interaction among processes!
- Efficiency: Everything has to be programmable…

KISS

- Success of LAN protocols, RISC architecture: KISS!
- "Building complex functions into network optimizes network for small number of services, while substantially increasing cost for uses unknown at design time"
- "End-end argument does not oppose active networks per se but instead strongly suggests that enthusiasm for the benefits of optimizing current application needs by making the network more complex may be misplaced"

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 communication?

- Business revenues:
  - 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

2. IP is more efficient?

- Statistical multiplexing versus circuit switching
- Link utilization
  - Avg. link utilization in Internet core: 3% to 30%
  - Avg. utilization of Ethernet is currently: 1%
  - Avg. link utilization of long distance phone lines: 33%
- Low IP link utilization: On purpose!
  - Predictability, stability, low delay, resilience to failure

- At low utilization, we *forfeit* benefits of statistical multiplexing!

3. IP is more robust?

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

- Convergence time with link failures:
  - BGP: 3 – 15 minutes
  - 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

5. Support of real-time app's telephony over IP

  - Not yet

Discussion: Benefits of IP?

- IP supports many different types of data applications at a wide range of data rates
- Phone network: 1 or many services (voice, fax, touch-tone service, 800 numbers, teletype, hearing impaired services, lots of enhanced voice services, voicemail…

- IP traffic, services more diverse (?). IP works at higher bandwidths (factually true for end applications, but cores are both high speed)
- Claim: IP supports short bursty connections "better" (implicit: Less setup cost, less resources used – not that important given utilization figures)
- IP has 1 rtt transaction times, phone network is at least 2 rtt (setup plus transaction)