Master Course
Computer Networks
IN2097

Prof. Dr.-Ing. Georg Carle
Christian Grothoff, Ph.D.
Dr. Nils Kammenhuber

Chair for Network Architectures and Services
Institut für Informatik
Technische Universität München
http://www.net.in.tum.de

Chair for Network Architectures and Services – Prof. Carle
Department for Computer Science
TU München

Internet Architecture and Future Internet

Outline

- What's wrong with today's Internet?
- Some general architecture principles
- Concepts for a Future Internet

What’s wrong with today’s Internet?

- Spam, Spit (=VoIP spam), worms, viruses, DDoS attacks
- Highly insecure routing protocols (e.g., BGP)
- No QoS mechanisms for end users
  (although protocols exist: DiffServ, RSVP, ToS headers in IP, …)
- Multihoming/IP roaming: difficult! IP address changes → connections break; TCP only can use one IP interface (although solutions exist: Mobile IP, SCTP, …)
- Number of prefixes in BGP routing tables grows dramatically.
  Reasons: growing number of providers; companies doing multi-homing or changing providers while keeping their address space; IPv6
- Almost no IP addresses left. (IPv6 exists, but nobody is using it.)
- Routing not very well understood. Routing and connection problems hard to analyse: what‘s the reason, who is to blame?
- TCP not well adapted to wireless networks: high delay variation, packet losses not induced by congestion, …

Problem: Routing security

- Recall that BGP manages traffic between providers
- By attacking BGP (e.g., injecting false information), IP packets headed for specific prefixes can be “hijacked” (e.g., diverted to the attacker)
- Problem: BGP is dangerously insecure! See the following two examples…

Prominent incident #1: Pakistan blocking YouTube

- YouTube was banned in Pakistan by law

- Pakistan Telecom thus was required to block access to YouTube
  - Pakistan Telecom announced YouTube IP prefix via iBGP, so that IP packets directed to YouTube traffic got routed into a black hole
  - Accidentally, they announced it via eBGP, too

- Consequences:
  - ASes in topological vicinity of Pakistan Telecom saw the (black hole) route in addition to the YouTube route
  - Due to the shorter AS path, they preferred it over the original
  - Result: YouTube not available in large part of the world

More details: Renesys blog, *Pakistan hijacks YouTube,* Feb 24th, 2008

Prominent incident #2: China hijacks the Internet

- Very similar to the Pakistan incident
- On Apr 8th, 2010, China telecom incorrectly announced 50,000 prefixes it did not own
  - N.B. 50,000 prefixes out of globally 350,000 prefixes = 15%; ***but not*** 15% of all traffic!
  - Prefixes included US government, US military, etc.
    In total, prefixes from 170 countries (including China!)
- Consequences
  - Many ASes in topological vicinity of China Telecom thus shifted the routes for the affected prefixes to China Telecom
  - A small ISP would have crumbled under the tsunami of traffic, but China Telecom is a big player, so they could handle it.
    In other words: Traffic now got routed via China Telecom
  - The event lasted about 20 minutes
- Deliberately?
  - Most certainly not!

More details: Renesys blog, *China's 18-Minute Mystery,* Nov 18th, 2010

What's wrong with BGP?

- Every AS can announce announce an IP prefix – even if it‘s not its own!
  - Inter-AS routing works on the honour system (like, e.g., Hawala)
  - Some plausibility filtering of BGP updates is done, but it's optional
- BGP sessions (=TCP connections) are not cryptographically encrypted.
- Very weak authentication
- Conjecture: Presumably a lot of bugs in router OSes that would allow buffer overflow exploits etc.

- Could become a great cyberwar battlefield!

Butler, Farley, McDaniel, Rexford: *A Survey of BGP Security Issues and Solutions* Proceedings of IEEE, 2009

Problems with routing (2)

- Dynamics of BGP are hard to understand
  - Many different vendor implementations
  - Complex configuration
  - A lot of potential error sources
  - Many effects still not understood
- Routing issues are hard to debug
  - No global view!
  - N.B.: BGP peers are competitors at the same time, so hide as much information as possible
  - “Where does the error come from? Who is to blame?” – Often, these questions are very difficult to answer.

- Solution: More research!?

Problems with routing (3)

- How long does it take to react to a hardware failure? (e.g., cable cut)
  - MPLS and layer-2 protocols: a few milliseconds
  - OSPF and intradomain routing: 200ms – 2s
  - BGP: seconds to minutes. Sometimes even hours!
- Reasons:
  - Unrestricted BGP would send out a lot of update messages
  - Thus, many mechanisms built in to reduce # of messages: Route flap damping, MRAI timer, …
  - Delaying messages means delaying information about topology changes!
- Solutions:
  - Layer-2 switching, MPLS Fast ReRoute (MPLS FRR) within a provider's network (~10ms)
  - IP Fast ReRoute (IPFRR) being standardised by IETF; scope: mainly within a provider's network
  - No widely accepted solutions for interdomain FRR yet

Problems with routing (4): End host mobility, end host multi-homing

- When a laptop changes its network connection from WLAN to 3G/UMTS, it gets a new IP address
- Existing TCP and UDP connections break, e.g.:
  - Persistent HTTP connections (usually transparent)
  - Instant messenger chats (úsually short interruption)
  - VoIP conversations
  - Ongoing HTTP or FTP requests like file downloads
  - VPN tunnels, ssh sessions, …
- And why can’t I simply bundle multiple interfaces to increase bandwidth?
  - Each interface has its individual IP address

- Structural problem:
  - IP address = ***identifier*** for TCP endpoint
  - IP address = ***locator*** for IP routing

Problems with routing (5): Network mobility, network multihoming

- Network mobility:
  - Suppose company C is customer of provider P
  - Provider P owns prefix 10.0.0.0/8
  - C is assigned network 10.11.0.0/16
  - Now C wants to change to another provider X
  - Solution 1: C is assigned completely new IP addresses from X. A lot of administrative overhead!
  - Solution 2: The new prefix 10.11.0.0/16 is announced by X. (N.B.: No conflict with 10.0.0.0/8 due to longest prefix matching)
- Network multihoming:
  - Same as above, but C wants to use a link to X as a backup
  - By accident, the link C—P is cut. C now sends out packets via X.
  - But how do the reply packets come to X, not to P?
  - Solution: The prefix 10.11.0.0/16 was previously announced by P *and* X (X probably announces a worse path by employing AS path prepending). After the cable cut, P withdraws the prefix: The world switches to X.
- OK, a bit cumbersome… but where's the problem here?

Problems with routing (6): IP address space fragmentation and number of routing table entries

- Number of IP prefixes in globally operating routers: ≥350,000 and rapidly growing
- Reasons:
  - Many new providers, many new users, especially in emerging markets
  - Companies that want to keep their IP addresses while
    - changing providers
    - or doing multi-homing (i.e., be connected to Internet via two different providers)
  - Plus: In the future, we'll see more and more IPv6 prefixes
- Is there a problem with that?
  - Routers need more memory, faster CPUs
  - Linecards become more expensive (more silicon needed for hardware-based longest prefix match with more entries)
  - Increasing number of BGP updates, ASes, AS paths
    - More BGP traffic
    - …which means: more BGP instability!

Mobility, multi-homing: Solutions (1)

- Mobile IP
  - Keep a permanent IP address when roaming at a relay
  - Mobile IPv4: Relay (“home agent”) introduces delays; issues with firewalls. Mobile IPv4 is dead.
  - Mobile IPv6 Route Optimisation: Tell shortcut to “real” IP address
  - Complex; largely unknown. (And: Who uses IPv6 anyway?)
  - Purpose: A solution for end hosts or *small* mobile networks (e.g., all end nodes within a train)
- HIP (Host Identity Protocol)
  - A host maintains a permanent unique identifier
  - Identifiers have 128 bit (=length of an IPv6 address):
HIP ID can be inserted as “layer 3.5” between IP and TCP or UDP
  - When IP address changes, a host can inform peers about the new IP address for the identifier using HIP
  - Cryptographically protected from hijacking à la BGP
  - Purpose: A solution for end hosts

[Mobility,] multi-homing: Solutions (2)

- SCTP (Stream Control Transmission Protocol)
  - ~successor to TCP (and UDP)
  - One SCTP association (connection) can use multiple interfaces
  - Problems:
    - Firewalls and NATs reject traffic that is not TCP, UDP, ICMP
    - Slightly more difficult to use than TCP or UDP
    - Nobody knows it ⇄ nobody uses it
  - Purpose: A solution for end hosts. Mainly addresses multi-homing; mobility is difficult.

- http://tdrwww.exp-math.uni-essen.de/inhalt/forschung/sctp_fb/sctp_intro.html
- Violin Yanev, SCTP, Ausarbeitung im Blockseminar Future Internet, SS2010

Mobility, multi-homing: Solutions (3)

- LISP (Locator–ID separation protocol)
  - Two types of IP addresses (i.e., disjoint address spaces):
    - End hosts / networks at edge use EIDs (end point IDs)
    - Routers in network core use RLOCs (routing locators)
  - IP Packets with EIDs are encapsulated into IP packets with RLOCs at *ITRs* (ingress tunnel routers), unwrapped at *ETRs* (egress TRs)
  - Idea:
    - Packets with EIDs are tunneled through opaque RLOC net
    - Many different EIDs, whereas #RLOCs ~ network size
  - Purpose:
    - Facilitate *network* mobility (changing providers)
    - Facilitate *network* multihoming
    - Allow these without further inflating BGP routing tables
    - *Not in focus:* host mobility…
  - Being standardised by IETF and Cisco

http://lisp4.cisco.com/lisp_over.html

Problem: Energy efficiency

- Context 1: Green IT
  - Communication infrastructure uses more and more energy
  - Bad for the environment (CO_2)
  - Increases costs
- Context 2: Mobile nodes
  - How long did a fully charged mobile phone last 10 years ago?
    And how long does a fully charged iPhone, Android, Symbian phone last?
- Solution A: Develop better hardware. (Not a problem of network research…)
- Solution B: Develop protocol improvements or new protocols that can help saving energy
  - Examples
    - 802.11b (11 Mbit/s, 100mW, range <500m): battery hog
    - UMTS HSPA (7 Mbit/s, 250mW, range ~2km): battery friendly
    - GSM (384 kbit/s, up to 2W, range ~5km): even friendlier to battery
  - Some principles (only a small selection; there's much more to it!):
    - Energy-efficient routing (wireless mesh networks)
    - Reduce broadcasts (wakes up receivers)
    - Try to send packets in one burst (wake up less often)

Problem: Malicious traffic, malicious users

- Unwanted traffic: Spam, Spit (=VoIP spam), DDoS attacks
- Worms, viruses, break-ins

- Basic problems:
  - Buggy implementations (perhaps not a problem of the network)
  - Most protocols do not use reliable authentication (e.g., SMTP)
  - The network cannot filter out undesired traffic on-demand
    - Static filters are widely used, though (rate limits, IP blocks, …)
  - "Bullet-proof hosting": ISPs that do not react to abuse reports
  - Users who are agnostic about security issues leave the door wide open for attackers ("I don't need a virus scanner, because I already have a firewall")

- Solution: ??

Problems with congestion control (1)

- Congestion control is done in end hosts (TCP)
  - Detects and minimizes congestion along the path
  - Path is determined by IP routers, not by end hosts
- What if alternate path exists without congestion?
  - Only changes in routing tables can shift traffic to uncongested paths
- Problem #1: Routing protocols normally do not react to routing!
  - Historic reason: Arpanet used traffic-adaptive routing, which led to oscillations → bad experiences
  - Technical reason: (Nonlinear) feedback loops with delay, plus coupled feedback loops → difficult
  - Today, providers use Traffic Engineering: Analyze link usage in network, change OSPF weights (etc.) to improve performance
    - Time scale: hours, not seconds

Problems with congestion control (2)

- Problem #2: What if I need TCP-like congestion control, but not the other TCP features? (e.g., for video streams)
  - Solution: Use DCCP instead of TCP or UDP.
  - But nobody uses it; Firewall issues (cf. SCTP); …

- Problem #3: TCP-friendliness
  - New congestion control schemes must be fair to existing TCP implementations: They must not take away all bottleneck bandwidth, but they must leave standard TCP its fair share
  - Restricts solution space for new congestion control schemes

- Problem #4: Radio networks
  - Hidden terminal effect, fading → packet losses.
But TCP treats them as sign of congestion.
  - High variability in link delays. Also bad for TCP.
  - Solution: Introduce additional retransmission features on layer 2…

Problems with delay (1)

- Long network delays. Example:
  - 1,100km from here to Salerno, including detours of cable.
    Speed of light in fibre (not vacuum!) is about 200,000km/s.
    Expected propagation delay: about 5.5ms, i.e., RTT=11ms.
  - Real value: 38ms!
- Even longer protocol delays.
  Example for accessing a Web site:
  - [ARP request, response: within LAN → negligible]
  - DNS lookup, possibly across several hierarchy levels: ≥ 1 RTT
  - [In future: HIP negotiation? +1RTT]
  - Sending TCP SYN, waiting for SYNACK: +1 RTT
  - [In case of HTTPS: SSL/TLS negotiation: +1 RTT]
  - Sending HTTP request, receiving HTTP response header: +1 RTT
  - In total: 3…5 RTT *plus* transmission delays until we receive the first byte of the content
  - Try it with www.cnu.ac.kr (Chongdu National University, Daejeon, Korea). The delay is certainly *not* due to a small bandwidth!

Delay: solutions

- Solution 1: Caching (e.g., HTTP proxies)
  - Persistent HTTP connection to proxy
  - Hopefully, proxy has cached object: Saves DNS lookup and TCP handshake; and presumably RTT to proxy is small
  - But if not, add processing delay of proxy plus RTT to proxy… not good.
- Solution 2: Distribute access points across the network
  - DNS-based (used frequently for Content Delivery Networks, e.g., Akamai, Google/YouTube):
    - If you enter www.youtube.com, you get a different IP address depending on your location in the network.
    - Principle: One DNS name, many hosts
  - Anycast-based (used for some root nameservers and 6to4 gateways):
    - You use the same IP address (e.g., 199.7.83.42, the L root name server run by ICANN), but this IP prefix is announced by multiple ASes across the globe
    - Principle: One IP address (!), many hosts
  - Both solutions are rather expensive

Problems with delay (2): Delay-tolerant networks

- Case 1: Underdeveloped regions
  - No permanent connections (at least not fast ones)
  - But: could transport USB sticks/hard disks back and forth “Never underestimate the bandwidth of a station wagon”:
    Delivering 1 TByte every 10 days is about 10Mbit/s!
- Case 2: Sparse mobile wireless networks (e.g., desaster areas)
  - Transmission often interrupted
  - People with Bluetooth/ad-hoc WLAN devices wander around and meet
- Case 3: Space travel
  - You can’t beat the speed of light
  - Moon: 2s RTT, sun: 17min RTT, outer planets: several hours
  - Pre-scheduled times without connectivity (e.g., while behind a planet)
- Obviously, we can’t do things like, e.g.:
  - TCP: handshake = 1 RTT; furthermore timeout = 120s
  - DNS or other lookups prior to sending request
- Solutions:
  - Very, very old: uucp
  - Bundle Protocol

Problems with quality of service (QoS)

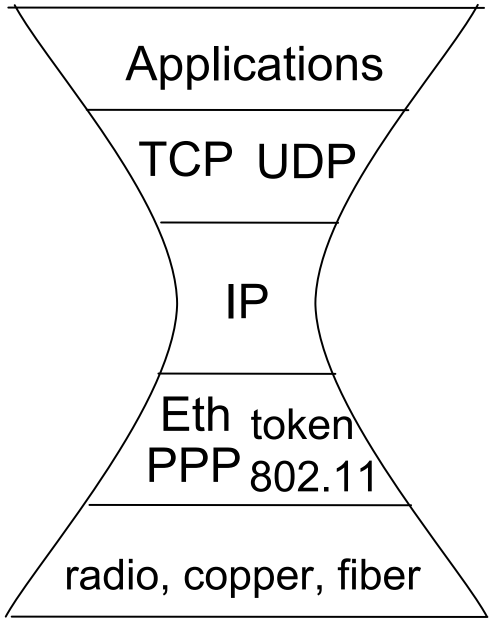
- Quality of service:
  - Interactive traffic (e.g., online games) more important than bulk traffic (e.g., e-mails, file transfers)
  - Therefore, give bandwidth guarantees and/or prefer it in queueing: Smaller queueing delays and/or smaller delay variation and/or reduced packet loss, etc.
- Solution:
  - IP TOS field (type of service) has been existing for years
  - Signalling protocols for establishing QoS connections (IntServ, DiffServ, RSVP,…) have been existing for years
- Status quo:
  - Being used within provider networks (e.g., for customer VPNs)
  - But: No end user software uses it!
- Problems:
  - Who pays when priority traffic goes across provider boundaries?
  - How to identify who has to pay?
  - What about DDoS attacks? (technically *and* financially)

Problems with multicast: Same as with QoS!

- Multicast:
  - Like broadcast, but to a specific group of recipients
  - For example, for streaming a TV program via a network
- Solution:
  - IP multicast addresses have been existing for years
  - IP multicast routing protocols have been existing for years (e.g., M-OSPF)
- Status quo:
  - Being used within provider networks (e.g., for triple-play with IPTV)
  - But: No end user software uses it!
- Problems:
  - Who pays when a multicast packet enters a provider via one link, and copies leave the network via 100 links?
  - How to identify who has to pay?
  - What about DDoS attacks? (technically *and* financially)

Problems with layers (1)

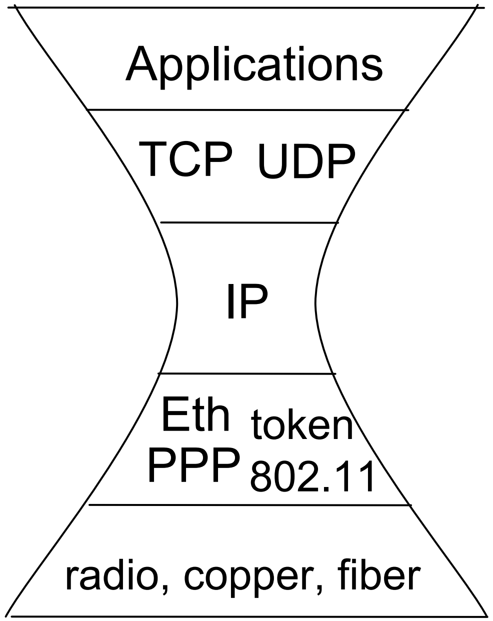
Original idea: The IP hour glass figure



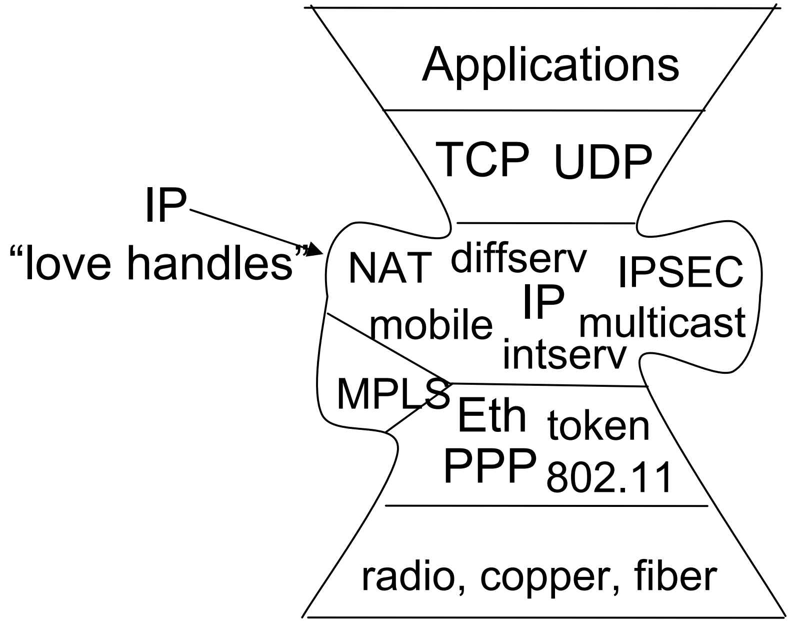
IP “hourglass”

Problems with layers (2)

Supporting new applications and services → losing the IP hour glass figure



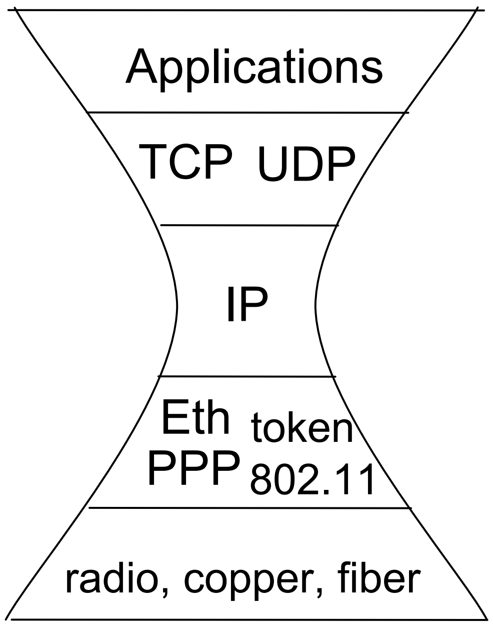
IP “hourglass”



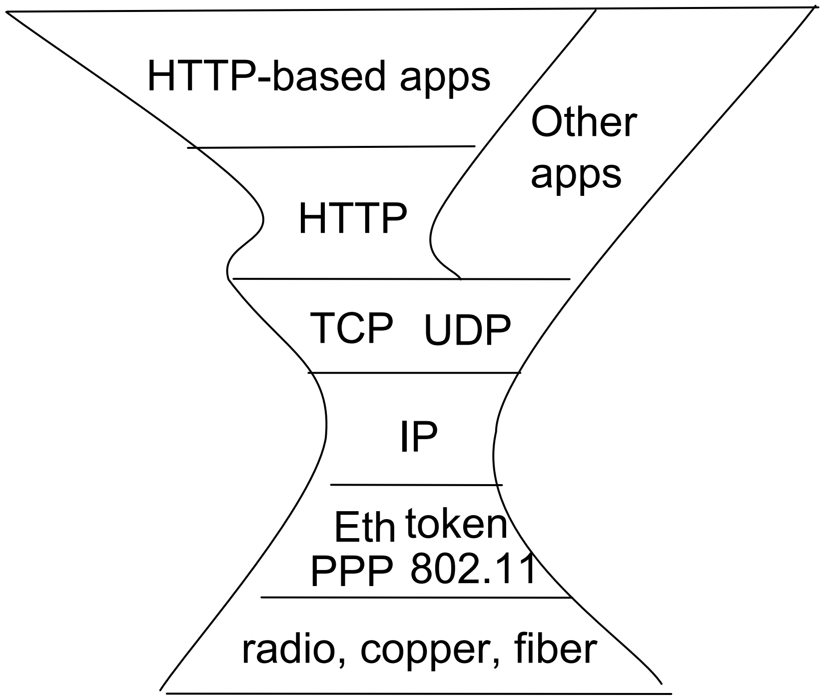
Middle-age IP = “hourglass” ?

Problems with layers (3)

- We used to have: * over IP
- We have today: * over HTTP over TCP over IP
  (e.g., Skype phone calls, YouTube video streams)



Original idea:
IP is greatest common denominator



Today:
HTTP is greatest common denominator

Fundamental Problem: The Internet only just works

Many more cases of "a solution exists in theory, but not in practice":

- Example 1: IPv6
  - We're out of IPv4 addresses.
  - IPv6 has been there for 15 years, but it's still not being used
- Example 2: DNSsec
  - By injecting false information into the DNS base, you could conduct attacks similar to BGP (e.g., Pakistan–Youtube or China Telecom)
  - DNSSEC: cryptographically signed DNS entries
  - Recently installed for some TLDs, but no browser/resolver uses it

Fundamental reason: *Never touch a running system*

- Corollary:
  Only do something new when the old system really, really starts to hurt
  - = the “ossification” of the Internet architecture

- Examples in the past:
  - TCP/IP was born when NCP (Arpanet) went out of control
  - TCP congestion control was deployed only when a congestion collapse was imminent
  - DNS was deployed only when the centrally managed HOSTS.TXT file grew too large and got unmanageable
  - CIDR IP prefixes (instead of the old class A, B, C networks) and NAT were deployed when IP addresses got too scarce
  - Used cookies, hidden forms, GET-IDs for tracking sessions in HTTP (=sessionless)
  - PPP, DHCP, NAT, POP3 when more users connected from home
  - ssh instead of Telnet/rlogin (encryption; automated X11 forwarding)

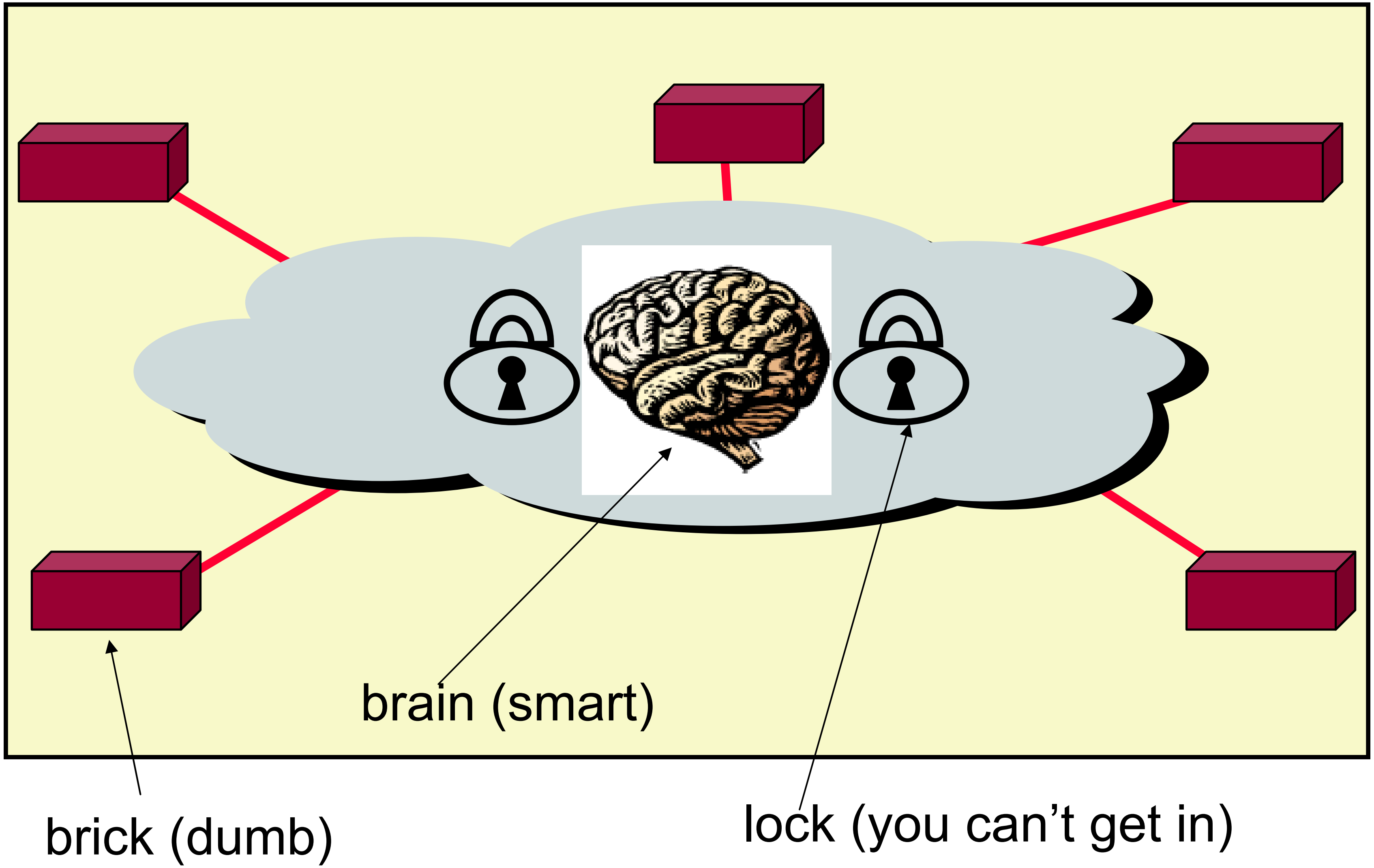
Mark Handley: *Why the Internet only just works.* BT Technology Journal, 2006

Future Internet

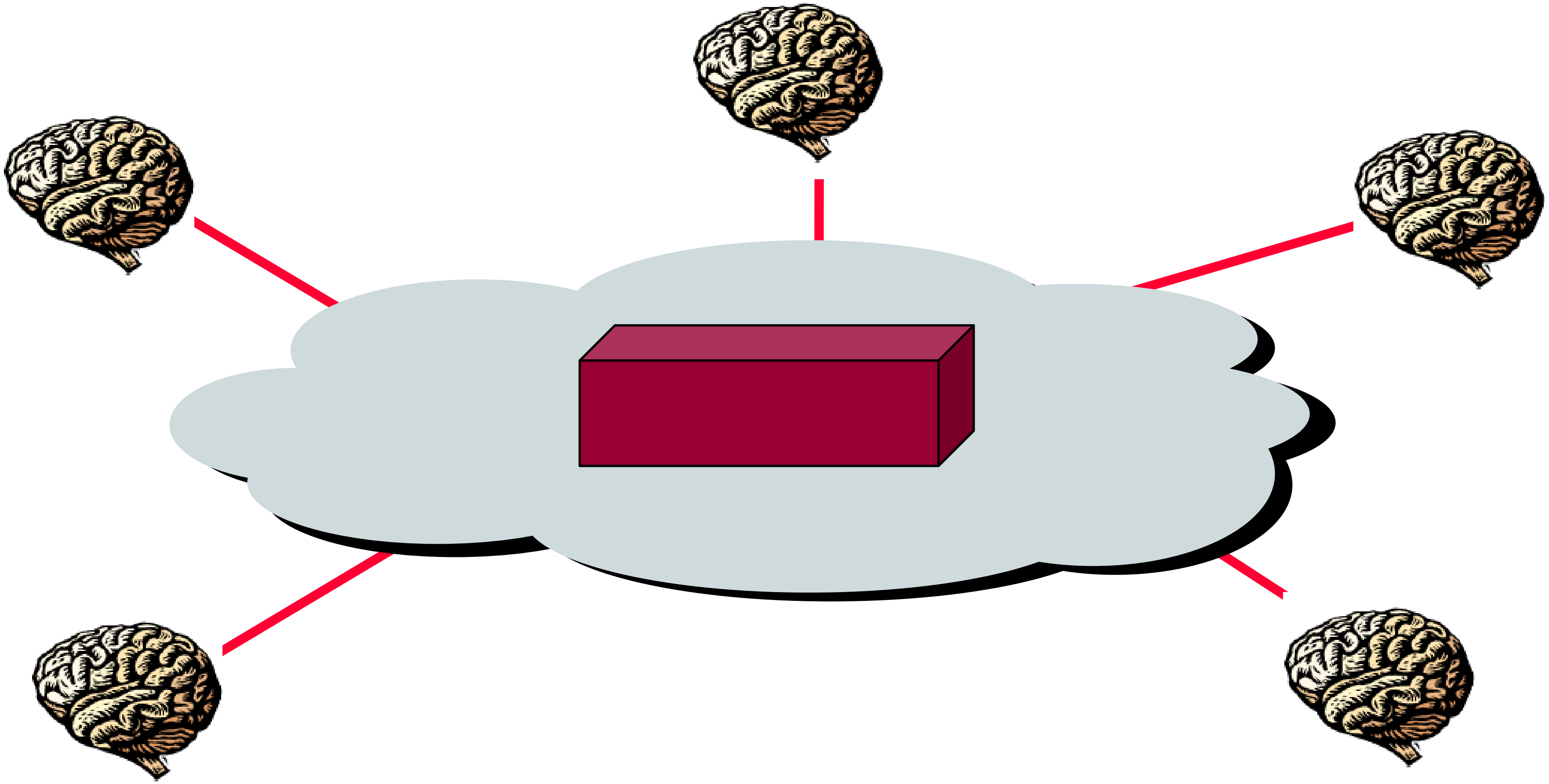
- There seem to be some fundamental flaws in the architecture of the Internet, so let's fix them
- Important research direction: A lot of money and effort going into this
- Sarcastic view:
  - Traditional network research was about developing new protocols to improve services and performance of the Internet
  - But Future Internet research is about developing new protocols to improve services and performance of the Internet
- But there is more to it – we have started asking (and answering) fundamental questions about the network architecture

- So… what are the basic concepts behind the architecture of today's Internet?

Common View of the Telco Network: Smart network, dumb endpoints

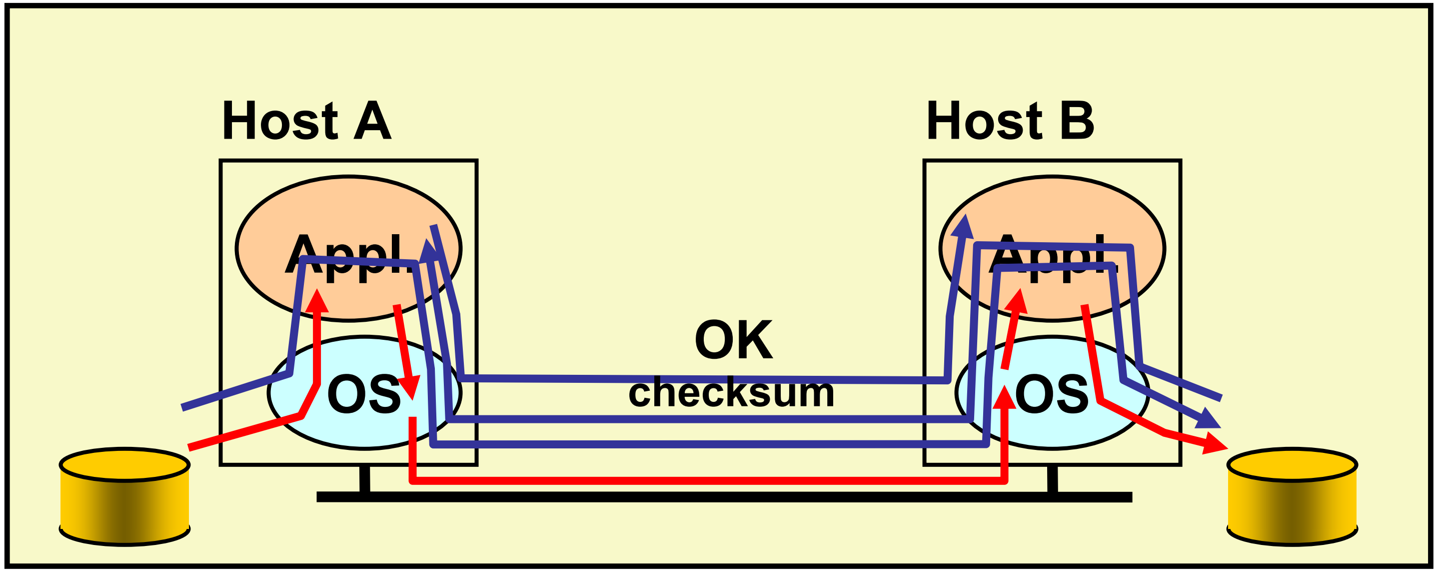


Common View of the IP Network: Dumb network, smart end hosts



The Internet End-to-End principle

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 (…the Internet way)

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
- Concrete example: Error correction on wireless links (in spite of TCP packet loss detection)

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

Internet & End-to-End Argument

- Discussion: congestion control, flow control: why at transport, rather than link or application layers?
- congestion control needed for many applications (assumes reliable application-to-TCP data passing)
- many applications “don’t care” about congestion control – it’s the network’s concern
- consistency across applications — you ***have*** to use it if you use TCP (social contract — everybody does)
- why do it at the application level
  - Flow control — application knows how/when it wants to consume data
  - Congestion control — application can do TCP-friedly congestion control

Internet & End-to-End Argument

- Discussion: congestion control, flow control: Why not at the link layer?
  1. Not every application needs it/wants it
  2. Lots of state at each router (each connection needs to buffer, need back pressure) — it's hard
  3. Congestion control in the entire network, e.g., load-adaptive dynamic IP routing? — multiple reasons against it:
     - hard to do
     - prone to oscillations
     - didn't work out in ARPANET → "never again" attitude

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

End-to-End Argument: Discussion

- End-end argument emphasizes correctness & completeness, but does not emphasize…:
  - *complexity:* Does 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 to change routers?
  - *technology penetration:* Simple network layer makes it "easier" for IP to spread everywhere

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 (except 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 from 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: Ethernet, Frame Relay, ISDN, FDDI, ATM
  - Today: SONET/SDH, WDM, WLAN, DSL, GSM
- Assessment: ?

Other Goals

- Allow distributed management
  - Administrative autonomy: IP interconnects networks
    - Each network can be managed by a different organisation
    - Different organisations 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 considerable harm (e.g., DHCP server running on laptop in LAN; ARP spoofing)
  - Assessment: ?

- Accountability
  - Works well if you only consider data volumes: just count bytes
  - Hard to do if you want to differentiate different kinds of traffic
    - Network neutrality: Pay extra money if you want to access Facebook, Youtube etc. in good quality
    - QoS: Cannot establish QoS connections across providers, because: who pays?
  - Very hard to pin down who did what (e.g., who is responsible for that strange BGP behaviour?)
  - Assessment: ?

Many implicit assumptions from the old days do not hold any longer

1970s, 1980s:

- Network is used by scientists/government. Very few malicious users, if any.
- Tens of networks, hundreds of hosts, thousands of users
- Network is jointly operated by public institutions without financial/economic interests.
- Host and network administrators are benevolent and not malicious. And they know their job. Normal (potentially unknowing, malicious) users do not have administrator privileges.

Today:

- Network is used by all kinds of people. Many malicious users (who even have financial incentives, e.g., phishing).
- Thousands to millions of networks, billions of hosts and users
- Network operated by competing companies.
- Unknowing users (“I don’t need a virus scanner, since I have a firewall”) and even malicious users (crackers, script kiddies) administrate their own hosts. Or even entire networks (e.g., bullet-proof hosting).

Technical response to changes

- Trust: emerging distinction between what is “in” network (*us*, trusted) and what is not (*them*, untrusted).
  - Firewalls, NATs
  - Ingress filtering
- 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

What’s missing?

Missing:

- No built-in security features (e.g., Spam, DDoS, worms)
- Architecture does not reflect economic relations (“tussle”) (e.g., QoS, multicast)
- A routing system that is understandable, efficient, fast, and easy to debug
- Host and network mobility
- …many more features, these are just the most important ones

But be careful not to lose:

- Possibility to communicate anonymously (e.g., Tor, Gnunet)
- Network neutrality
- Network neutrality
- Possibility to communicate anonymously
- …many more features, these are just the most important ones

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.”

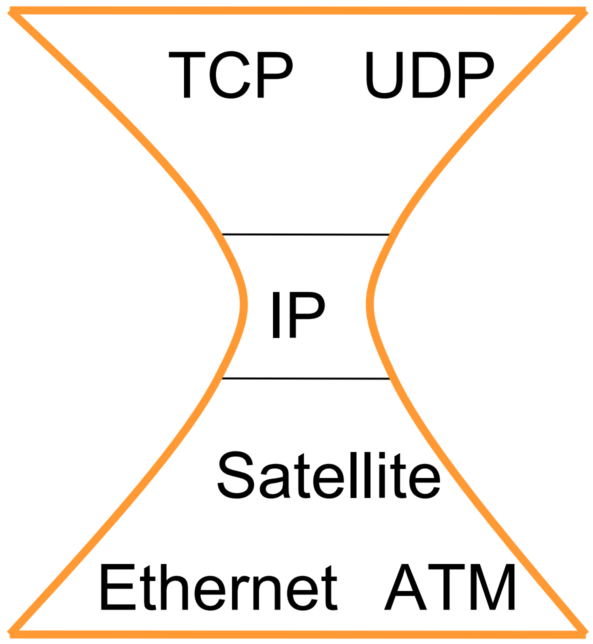
[Blumenthal and Clark, 2001]

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!

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

Before we go on: Architecture components, concepts, principles

- Protocol machines
- Packets, continuous data stream
- PDUs
- Connection-oriented, connectionless; circuit-switched, packet-switched
- Layer abstraction
- Routing, forwarding
- Data plane, signalling plane
- In-band vs. out-of-band; separation of control and data
- Addressing, naming
- Lookups, indirection
- Virtualisation
- Flow control, congestion control
- Error correction, error recovery
- Randomisation
- Multiplexing
- Unicast, multicast, broadcast; point-to-point, point-to-multipoint

Where have we been?

Design Principles

- separation of control/data (signaling, ftp, http)
- randomization (CSMA-CD, router synch, routing)
- indirection (multicast, mobile IP, i**3)
- multiplexing: packet level (WFQ, priority), burst level, call level (routing in telephone net)
- virtualization (Internet, IP-over-ATM, MPLS, VLAN, VPN)

What's inside a protocol?

- An application that wants to communicate (or: an upper layer)
- An interface that allows to communicate with another protocol instance (or: a lower layer)
- PDUs (protocol data units) that can be sent and received via the interface
- The protocol state machine: Tells what we sent and what we expect to receive next

- Basically, a protocol can be viewed as an IPC facility!
(inter-process communication)

PDUs

- Format:
  - Header (usually)
  - Data
  - Trailer (not very often. Example: Ethernet CRC)
- Size:
  - Fixed? (Easier to parse) Variable? (Less waste of resources)
  - Large? Small? – It depends!
    - Make suitable for needs of application
    - Larger PDUs usually are more efficient in the network

Layers

- cf. ISO/OSI model, Internet model

Connections and circuits

Connection-oriented

- TCP
- phone network

Connectionless

- UDP
- SMS (from user perspective)

Circuit-switched

- Phone network

Packet-switched

- IP
- …thus, also TCP!

Setting up a connection

- Enrolment: Reserve memory, create data structures
- Establishment: Tell the other end that you want to communicate
- Synchronization: Negotiate parameters
- Data transfer

- Establishment and synchronisation usually joined together
  - No synchronisation: connectionless (e.g., UDP)
  - Two-way handshake
  - Three-way handshake (e.g., TCP)
  - Multi-round negotiations

Routing and forwarding

Forwarding

- Many nodes are not directly connected to each other
- Intermediate nodes have to *forward* (to relay, to switch,…) PDUs
- [N.B.: This is often referred to as "routing", but it's actually wrong…]

Routing

- Intermediate nodes have to know where they have to forward the PDUs to
- *Routing:* the process of (jointly!) determining the paths through the network and setting up the forwarding

Planes

- **Warning: plane ≠ layer!**

- Data plane
  - The part of the router (or network architecture) where the packets are *forwarded* to other nodes
  - High data volume
  - Processing done in hardware
- Signalling plane
  - The part of the router (or network architecture) where the routes are set up and topology / other network information is exchanged with other nodes (*routing*)
  - Low data volume (…if not, then there's something wrong)
  - Processing done in software
- Proposed concept: Management plane
  - A new, integrated part of the network architecture that allows to consistently manage the policies of ~all nodes in the network (i.e., their signalling planes)

In-band vs. out-of-band signalling

In-band signalling:
Protocol-related information in same channel as payload data

- Examples
  - HTTP: First headers, then the data
  - TCP: Headers control state machine, flow control, congestion control; data follows
  - IP: Routing protocols (e.g., OSPF, BGP) use IP packets to exchange information
- Assessment
  - Keeps things simpler
  - Processing can be less efficient
  - Don't burn your bridges (e.g., router configuration via ssh…)

Out-of-band signalling:
Protocol-related information in channel separate from payload data

- Examples
  - FTP (traditional "active"): Commands on port 21, actual data on port 21
  - MPLS: only used for forwarding; but the tunnels are set up using an LDP (e.g., RSVP)
- Assessment
  - More channels: more complexity

Addressing and naming (1)

Address (Locator)

- Where is the destination of the PDU within the network's topology?
- Examples:
  - IP addresses (more precisely: IP prefix)
  - Phone numbers

Name (Identifier)

- Identify…
  - Hosts within same "area" of network (e.g., broadcast segment)
  - Processes within one host
- Examples:
  - IP addresses ☹
  - Names in phone book
  - DNS entries
  - Search keywords in filesharing networks, Web page search (Google), …

Addressing and naming (2)

Hierarchically

- Addressing
  - IP addresses (network prefixes)
  - Phone numbers (country code + area code [+ in old analogue times: initial digits within number] + MSN [+extension])
  - Helps structureing the network
- Naming
  - DNS (. .de .tum.de .in.tum.de .net.in.tum.de)
  - People (Family name, First name, perhaps middle names depending on culture)
  - Facilitates distributed administration

Flat

- Addressing
  - MAC addresses (N.B. vendor prefixes do not serve any naming purpose
  - IP addresses within one broadcast domain
  - Nowadays: Mobile phone numbers (international roaming; keeping the number after provider change)
  - AS numbers
- Naming
  - GPG/PGP keys
  - Search keywords for Google
  - People’s first names
  - Hosts within same network

Lookups, Indirection

- Lookup services
  - Convert names to addresses
    - DNS
    - Phone book
    - Google! (Search keywords to URLs)
  - Convert addresses to other addresses
    - ARP (IP addresses to MAC addresses)

- Indirection
  - N.B.: The URL example showed that the same thing can be an address as well as a name. Other examples:
  - Mobile IP (home agent points to actual location)
  - HTTP Redirects (status codes 301, 302, 303, 307)
  - Multicast! (One address represents an entire group of hosts)

Virtualisation, overlay networks

- Create new functionality on top of existing functionality
  - "*on top*": layer-wise perspective
  - Compare terms to host virtualisation: "Windows VM running *inside* a Linux machine"
- Examples
  - Skype, P2P file sharing networks, Tor build a peer-to-peer overlay consisting of TCP and UDP connections on top of the existing Internet
  - PlanetLab builds a world-wide experimentation test bed on top of the existing Internet
  - MPLS builds a virtual network on top of layer 2
  - The Internet
    - was originally an overlay on top of the phone network (WAN connections = modem lines)
    - is still is an overlay on top of various different network technologies (Ethernet, WLAN, GSM/3G/UMTS/LTE, SONET/SDH, …)
      - Layer 2 topology may look vastly different from what we can see when we do traceroute (layer 3)
  - A rather **daring** assertion: TCP builds a lossless in-order virtual byte stream service on top of the lossy no-order datagram-oriented IP service. (Many wouldn't consider this to be virtualisation, though.)

Error detection, error recovery

- Error types
  - Corruption (bit flips; e.g., due to radiation)
  - Loss (entire PDUs missing; e.g., dropped during congestion)
  - Number of occurences: Just one flip / drop, or a burst of flips / drops
- Error detection
  - Checksums (e.g., CRC)
    - IP header
    - Ethernet
  - Byte/PDU counters
    - TCP (segment#, ACK#)
  - Timeouts
    - TCP
    - DHCP
- Error recovery
  - Just ignore it and try your best (GSM voice codec)
  - Retransmission (TCP)
    - Needs retransmission control
  - Forward error correction: Transmit slightly redundant signal that allows to restore full information if only a small bit of information is lost.
    (Think of it as doing RAID-5 on packets.)

Flow control, congestion control

- Flow control: Don’t overwhelm the receiver with too much data
  - E.g., fast Web server sending data to a small phone with slow CPU
- Congestion control: Don’t overwhelm the network with too much data
  - E.g., fast Web server connected to Internet via 2 Mbit/s DSL line

- Cf. TCP lecture

Randomisation

- Sometimes, determinism would just be too expensive
- In that case, use clever (!) randomisation
- Examples:
  - Backoff timer in Ethernet, DHCP
  - Key generation in cryptography

- Think about this:
  - 10 Mbit/s TokenBus:
    - Deterministic bus access, no collisions
    - Can use full hardware speed
    - Expensive, has been dead for decades
  - 10 Mbit/s Ethernet with CSMA/CD:
    - Randomised bus access tries to minimise collisions
    - The more collisions, the more bandwidth is wasted
    - Cheap, widely used, offspring protocols live on in 100 Gbit/s Ethernet (although without CSMA/CD, but switched!)

Multiplexing

- Combine multiple different signals together and send them across the same media
- At the other hand, we have to employ demultiplexing
- Examples:
  - Many different TCP connections across one Internet link (e.g., your computers at home are connected via one DSL line to the I'net)
  - Multiple wavelengths in one optical fiber can be used for different circuits
  - Two separate voice circuits across one single ISDN line
  - Multiple TV programs within one DVB signal

Fragmentation, reassembly

- Usually a consequence of
  - Multiplexing/demultiplexing
  - Layering
- Examples:
  - IP packets being fragmented, reassembled at receiver
  - IP packets being fragmented transparently into 48 byte ATM cells, reassmebled when exiting the ATM network

Ordering

No ordering

- Easier for network
- Perhaps harder for receiver
- Example: IP packets

Ordering

- Easier for receiver
- Example: TCP bytes

When to communicate

- Synchronous operation
  - Data transferred at fixed points in time
  - Usually on lower layers
  - Example: mobile phone networks (TDMA)
- Push
  - Sender just pushes data to receiver, whether wanted or unwanted
  - Example: IP (usually wanted, but DoS traffic is not wanted…)
- Pull (request/response)
  - Receiver requests data, sender sends desired data
  - Example: HTTP
- Publish/subscribe
  - Receiver describes the data he's interested in
  - Every time the sender comes across data matching the description, it is forwarded to the receiver
  - ~"Asynchronous request/response"
  - Example: RSS

*-cast, <X>point-to-<Y>point

- Unicast
  - "Normal" case: one node sends data to one other
- Multicast
  - One node sends data to specific group of other nodes
- Broadcast
  - One node sends data to all other nodes
- Point-to-point
  - Channel between two nodes
  - Usually, unicast
- Point-to-multipoint
  - One node communicates with many others
  - Usually, multicast/broadcast and replies via unicast
- Multipoint-to-multipoint
  - Nodes within a group communicate with each other
  - Replys also via multicast/broadcast

Security (1): Authentication, authorisation

Authentication

- Prove your identity
- Examples:
  - GPG signature under your e-mail
  - Entering correct login and password

Authorisation (access control)

- Depending on who you are, obtain access (or not)
- Examples:
  - File access rights in network file system
  - Access / no access to shared IMAP folders

Security (2)

- Integrity
  - Protection against unauthorised insertion or deletion of PDUs
  - Example: Transfer ***20,000€*** from account A to B
- Confidentiality
  - Ensure that contents of PDUs cannot be read by unauthorised parties
- Nonrepudiation (non-deniability)
  - Ensure that a communication party cannot deny that it has participated in a conversation
  - Examples:
    - Signed E-Mail (key identity-checked): non-deniable
    - OTRS encrypted Jabber conversation: deniable (design goal!)

Resilience

- The ability of a system to withstand failures, disruptions, and other challenges
- Acceptable network and service quality even under severe disruptions
- “Acceptable”: relative; depends on application and users
- Example #1: IP routing *within* a provider’s network
  - …is resilient: 500ms – 1s convergence time is acceptable (home end users reading e-mail, surfing the Web, downloading files,…)
  - …is not resilient: 500ms – 1s convergence time is utterly inacceptable (professional end users: online trading, telemedicine, video conferences)
- Example #2: 99.99% guaranteed availability for a provider’s network
  - Let’s see… 99.99% · 365 days = at most 1 hour downtime / year
  - Resilient for nor mal home end users
  - Not resilient for business users
- Example #3: VDSL line (60Mbit/s) with GSM backup line (384kbit/s)
  - Resilient for home end users (YouTube is slow, e-mail still works)
  - Not resilient for a small office with 30 employees

Back to the future Internet!

- Re-arrange some of these fundamental building blocks?

or

- Integrate them […anew, in abridged shape…] into the existing architecture?

Future Internet: how?

- Evolutionary approach
  - Tackle one problem at a time
  - Incrementally introduce new protocols to overcome weaknesses
    - IPv6
    - SCTP, DCCP
    - LISP, HIP, Mobile IPv6
  - Advantage: backwards compatibility
  - Disadvantage: Legacy burden; sometimes a radical cut is needed
- Revolutionary approach (“clean slate”)
  - Throw away the old architecture, and build a radically new one
  - Advantage: no legacy burden; more freedom to create sth. new
  - Disadvantage: it just won’t happen!

- Perhaps the two are more or less the same… (cf. next slide)

Where are we headed: Current/upcoming research topics

- Network management:
  - Measurement, automation (“management plane”)
  - Reflecting the fact that ISPs business entities (“tussle space”)
- Service management:
  - Application-level networks, overlays, distributed hash tables (DHT)
  - QoS: Not a solved problem end-end
- Wireless networking, mobility
- New types of networks:
  - Sensor nets, body nets, home nets
- Security:
  - Today: Lack of cryptographic signatures in many protocols
  - Today: Most traffic unencrypted (…good for measurements…)
  - Difficult: Accountability, non-repudiability, traceability vs. anonymity
- Resilience: more robust networks and services (reacting faster, keep up acceptable service quality under even more disruptive failures)
- Ease of use, deployment (but what are the research problems here?)

Future Internet: Some radical concepts

- Source routing in the core
- DHT-based routing and lookups
- Freely pluggable building blocks instead of fixed layers
- Content-centric networking

Radical concept: Source routing

- Current Internet: Routing purely destination-based
  - Hand your packet to the next hop and trust it will make a good decision
- Proposal: Source routing
  - Sending AS (*not:* sending host!) prescribes the exact route to receiving AS in packet header
  - If an intermediate AS doesn't like the route (←policy), it can drop the packet
  - Advantage: Do not rely on (unreliable, misconfigured, buggy) ASes along the path
  - Challenges: Security issues, accounting, …

Radical concept: Using DHTs for basic networking functions

- DHT (Distributed hash table):
  - Imagine a hash table. It has two operations:
    - put(key, object)
    - object = get(key)
  - Now imagine the data structure to be distributed among thousands of nodes. That's a DHT.
- Remember the architecture slide on *lookup* functionality? A DHT can be used for any of these, e.g.,
  - Mapping names to addresses
  - Higher-level indirection
  - Mapping addresses to routes
  - Storing network topology information
  - Storing routing policy information
  - …

Radical concept: Make layers more flexible

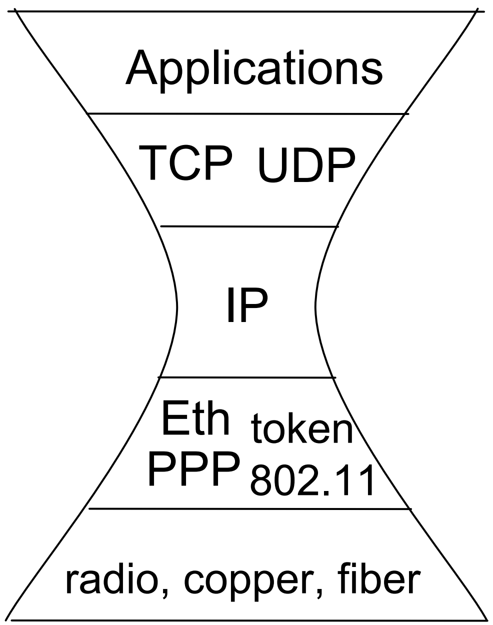
- Each application has different requirements concerning, e.g.,
  - Reliable ↔ non-reliable delivery
    - Retransmissions, FEC, timeouts, when to ignore errors
  - In-order ↔ unordered delivery
  - Congestion control ↔ predetermined bit rate
  - Flow control ↔ no flow control
  - Datagram delivery ↔ byte stream delivery
  - Connection-oriented ↔ connectionless
  - Integrity, confidentiality, authenticity, nonrepudiability, …
- Each media offers its own service
- Some ideas:
  - Application specifies requirements, network automatically transmits data using appropriate protocols
  - No fixed layers, but flexible building blocks (e.g., reliable data transfer module, flow control module, …). Application specifies how they should interact.

Radical concept: Content-centric networking

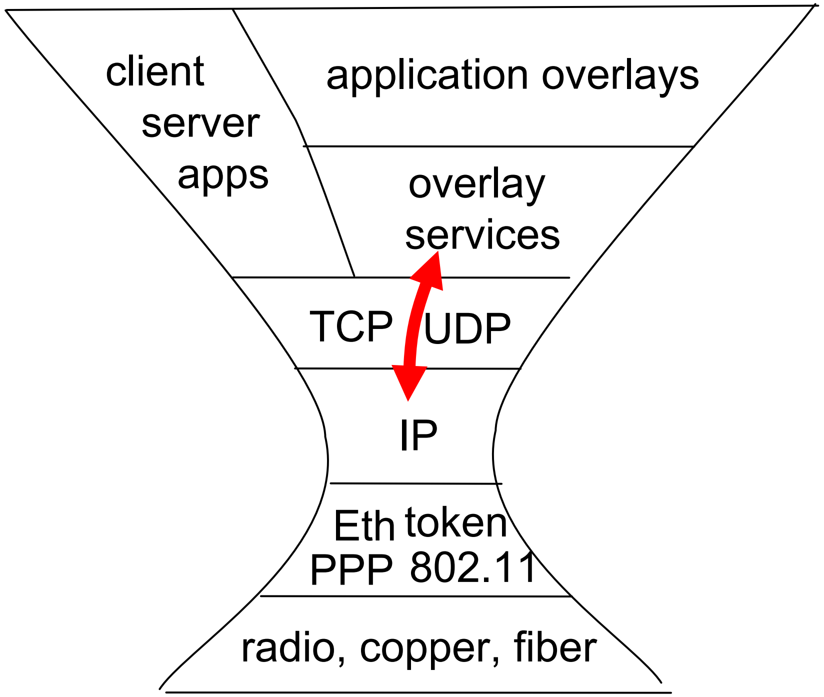
- Today's Internet:
  - Mainly request/response (e.g., HTTP)
  - Addressing: host-based / process-based (IP addresses, ports). Receiver has to know where it should send the request to.
- Content-centric networking:
  - Publish/subscribe: The receiver tells the network what kind of data it is interested in.
  - Addressing: content-based.
    - Simple example: specify the DOI
    - More elaborate: specify keywords associated with desired content (think of adding a "Google layer" on top of the network…)
    - Futuristic: semantic description of content
  - Advantage: Replicating/caching is easy, since we address data, not hosts.
    - Replication fosters load balancing
    - Replication increases resilience (no single source of failure)
    - Replication can reduce delays
  - Interesting questions: How to do routing, how to invalidate / withdraw / update cached data, what about dynamic data (e.g., dynamic Web pages), confidentiality / authenticity / integrity, business/economic aspects and policies, legal issues, how to emulate sessions like ssh or telephony (it's possible: subscribe to ACKs of the other end), …

Revolutionary Future Internet in an evolutionary development

- Let's just build the Future Internet as an overlay on top of the old one!
- After all, the Internet started as an overlay of the phone network…



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 the solution?

Will my solution scale?

- Think about what happens if you're successful:
  your protocol will be 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……

Seven Cautionary Questions

Q2: Is this really a bottleneck?

- 80% of gains achievable by focusing on 20% of system
- use profiling tools to see where time is spent

Seven Cautionary Questions

Q3: Effect of change on rest of system?

- does change increase performance in one place but slow down in other places?

Q4: Does an initial analysis indicate potential significant improvement is possible?

- is there room for improvement?
- how close to best possible performance ? Think about *bounds,* solutions (e.g., oracle) with unachievable performance

Seven Cautionary Questions

Q5: Is it worth adding custom hardware?

- ride Moore's curve (doubling of processing speed every 18 months) or use specialized hardware?

Q6: Can protocol changes be avoided?

- Rather than scrap existing protocol, tweak/rethink it to solve problem?
- Example: TCP's imminent demise predicted many times (e.g., TCP too slow for high-speed implementation)

Seven Cautionary Questions

Q7: Does prototype confirm initial promise?

- initial high-level analysis will miss details that could be important
- some people will never be convinced without an implementation

Q8: Will performance gains be lost if environment changes?

- think about if improvements limited to small number of environments
- example: same-connection, in-order packet assumptions won't hold in busy server.

More cautionary questions…:

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?

More folklore/advice

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

More folklore/advice

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

Properly parameterized?

- 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 (good-natured initial values)

Challenge: on beyond the data plane

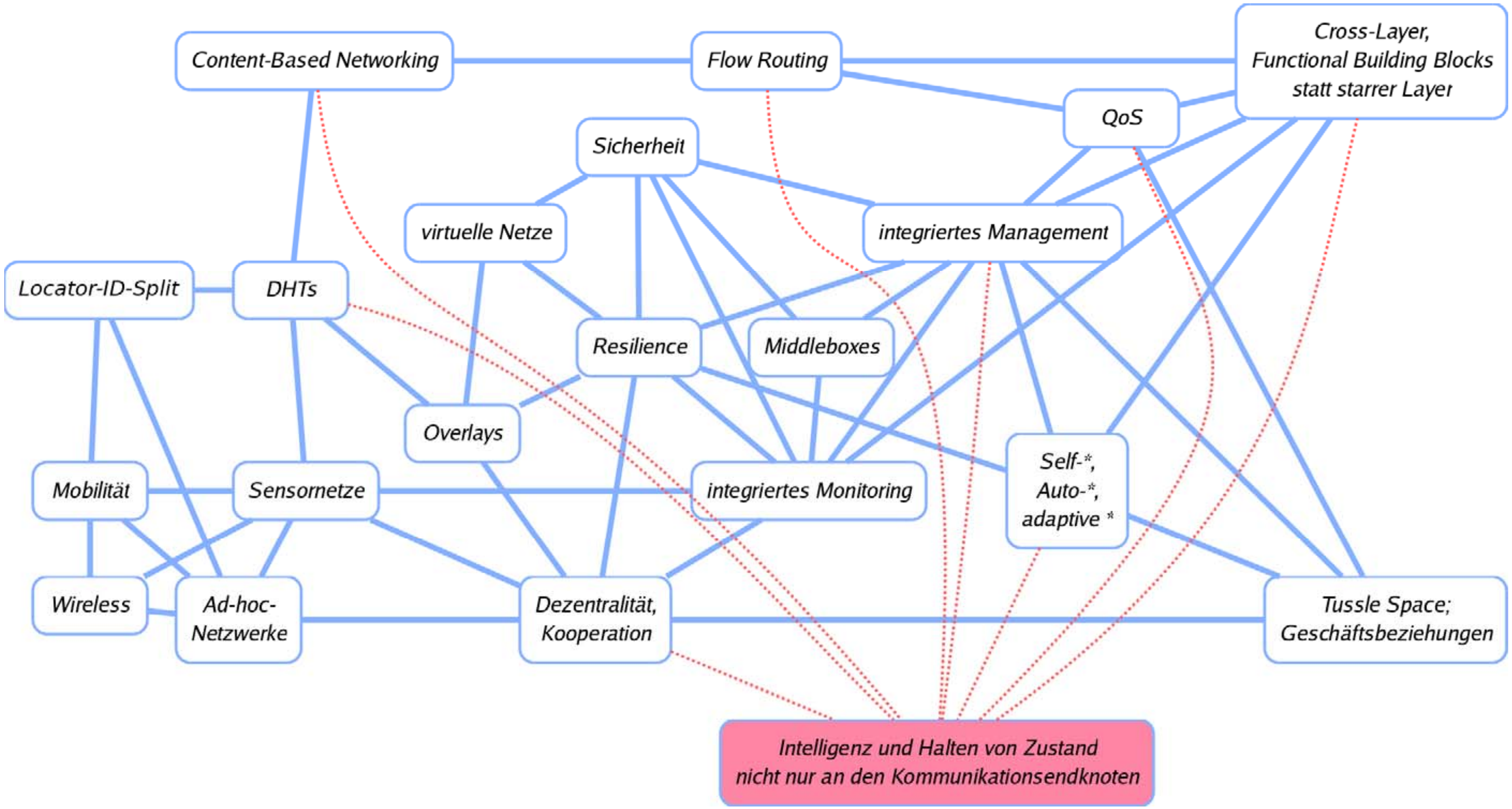
- Q: data plane performance really *the* major roadblock?
  - "robustness"
  - "complexity of control"
  - maintainability
  - evolvability
  - adaptability
  - reconfigurability
  - security
  - manageability

the "X-ities"

- Fundamental advances here are hard!
  - "efficiency" not always the most important measure
  - little/no past work on the "X-ities"
  - metrics and models still to be defined

Future Internet

(sorry for the German labels, but most notions are in English anyway…)



The *really* big picture

- Importance of user requirements

“It’s the end-user, stupid”

“It’s the application, stupid”

“It’s the network, stupid”

of course, not everyone agrees ….

Verizon product, purchased 2007

The end!