Master Course
Computer Networks
IN2097

Prof. Dr.-Ing. Georg Carle

Chair for Network Architectures and Services
Department of Computer Science
Technische Universität München
http://www.net.in.tum.de

Outline

Signalling

Internet Architecture

Design Principles

Future Internet

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

Maintaining network state

Design Principles

Goals:

- identify, study common architectural components, protocol mechanisms
- what approaches do we find in network architectures?
- *synthesis:* big picture

Maintaining Network State

state: information *stored* in network nodes by network protocols

- updated when network "conditions" change
- stored in multiple nodes
- often associated with end-system generated call or session
- examples:
  - ATM switches maintain lists of VCs: bandwidth allocations, VCI/VPI input-output mappings
  - RSVP routers maintain lists of upstream sender IDs, downstream receiver reservations
  - TCP: Sequence numbers, timer values, RTT estimates

Hard-state

- state *installed* by receiver on receipt of *setup message* from sender
- state *removed* by receiver on receipt of *teardown message* from sender
- *default assumption:* state valid unless told otherwise
  - in practice: failsafe-mechanisms (to remove orphaned state) in case of sender failure e.g., receiver-to-sender "heartbeat": is this state still valid?
- examples:
  - ISDN Signalling, ATM Signaling
  - TCP

Soft-state

- state *installed* by receiver on receipt of setup (trigger) message from sender (typically, an endpoint)
  - sender also sends periodic *refresh* message: indicating receiver should continue to maintain state
- state *removed* by receiver via timeout, in absence of refresh message from sender
- default assumption: state becomes invalid unless refreshed
  - in practice: explicit state removal (*teardown*) messages also used
- example:
  - RSVP

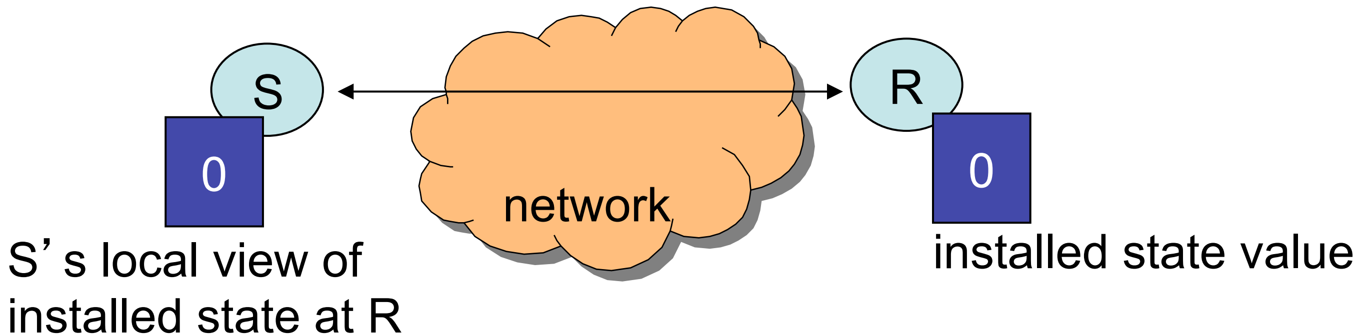
State: senders, receivers

- sender: network node that *(re)generates* signaling (control) messages to install, keep-alive, remove state from other nodes

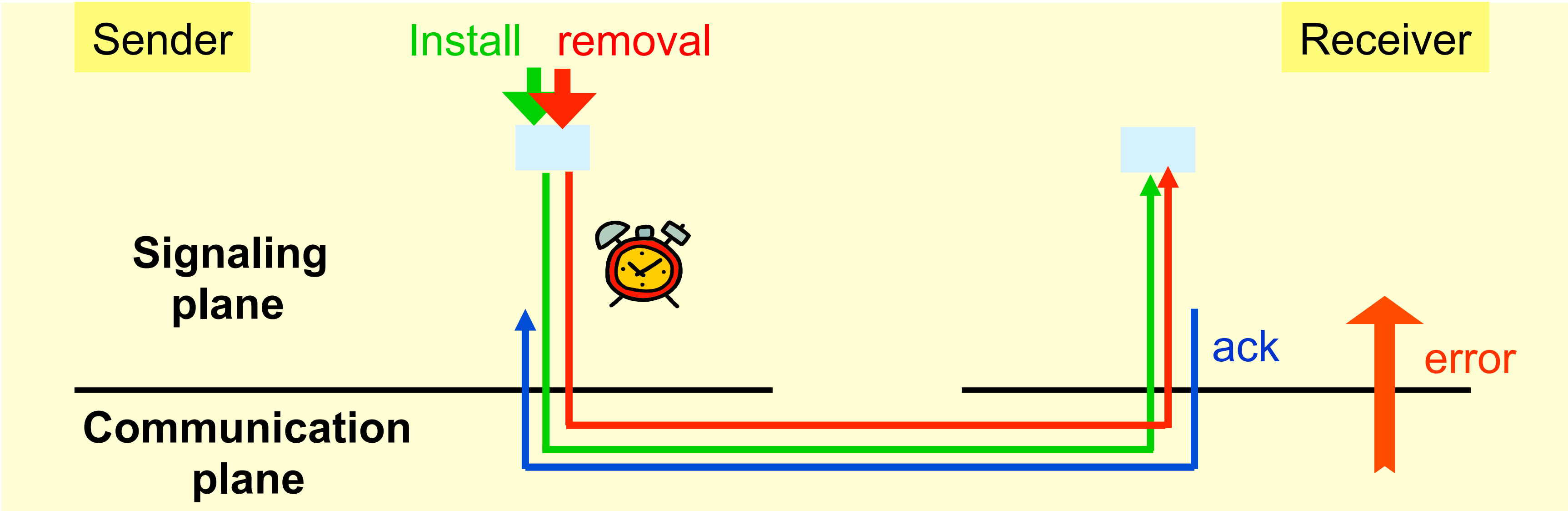
- receiver: node that creates, maintains, removes state based on signaling messages *received* from sender

Let's build a signaling protocol

- *S:* state *S*ender (state installer)
- *R:* state *R*eceiver (state holder)
- desired functionality:
  - S: set values in R to 1 when state "installed", set to 0 when state "not installed"
  - if other side is down, state is not installed (0)
  - initial condition: state not installed

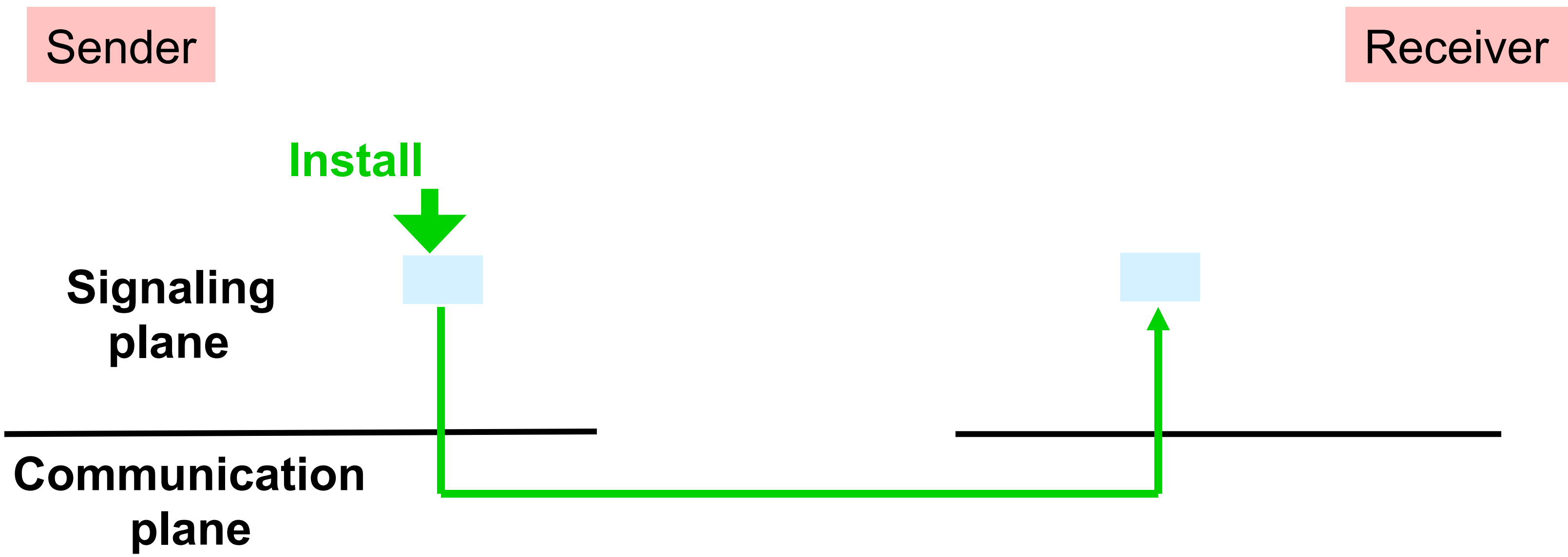


Hard-state signaling



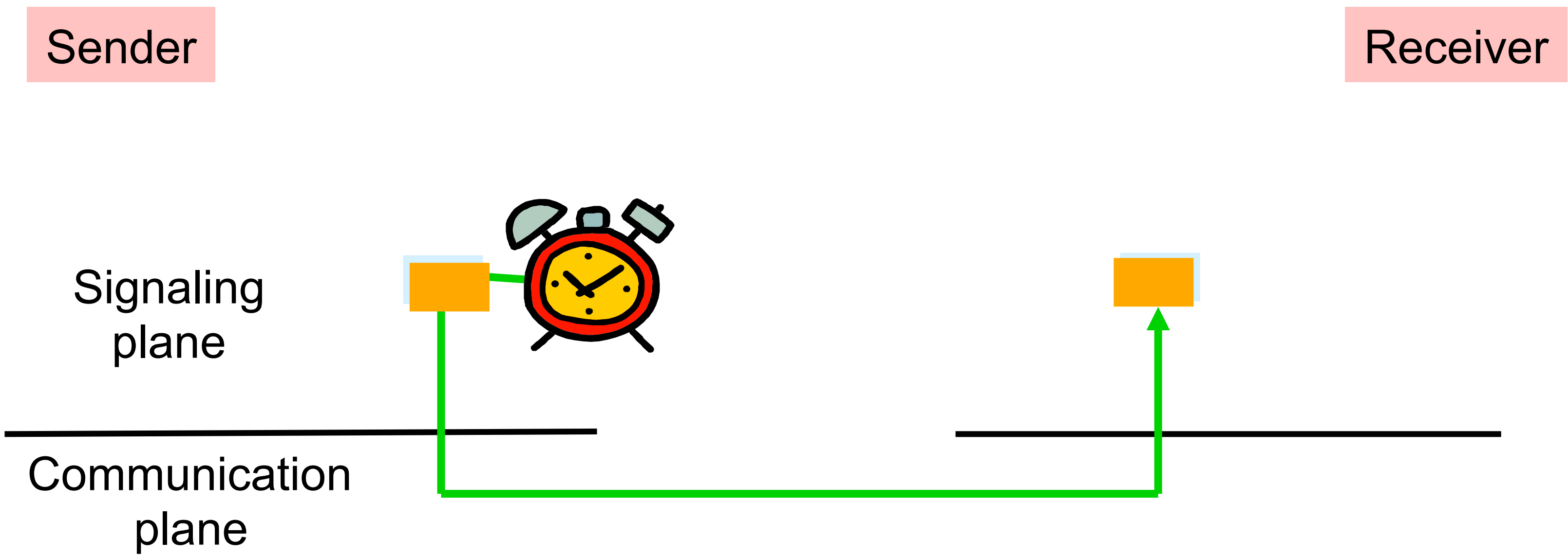
- reliable signaling
- state removal by request
- requires additional error handling
  - e.g., sender failure

Soft-state signaling



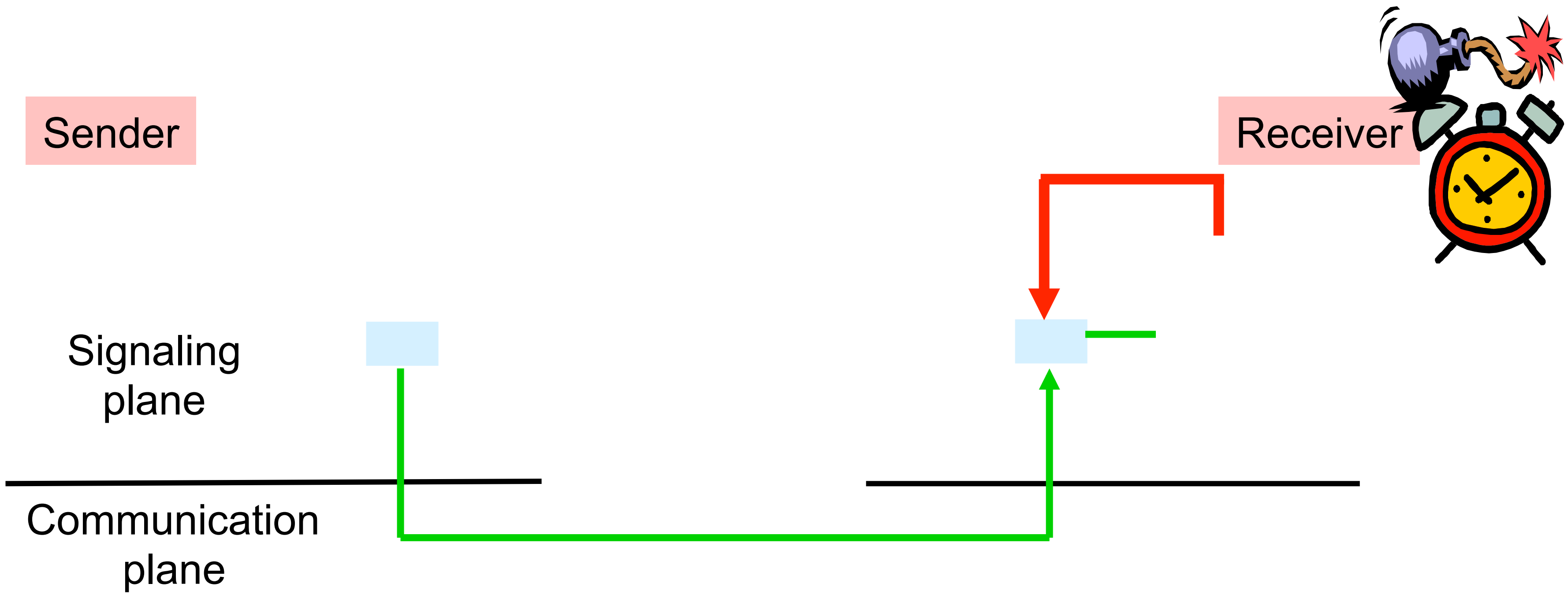
- best effort signaling

Soft-state signaling



- best effort signaling
- refresh timer, periodic refresh

Soft-state signaling



- best effort signaling
- refresh timer, periodic refresh
- state time-out timer, state removal by time-out
  - Does not preclude explicit state removal
    as optimisation for performance enhancement

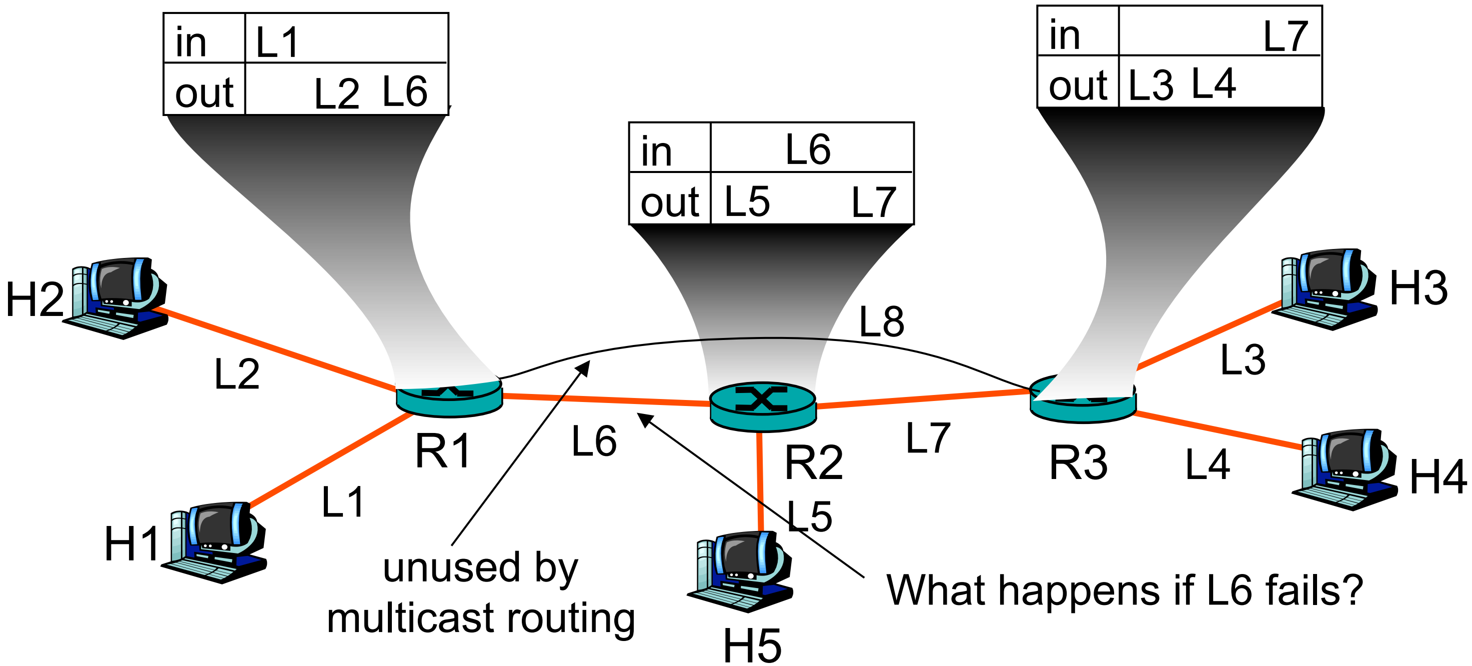
Soft-state: claims

- “Systems built on soft-state are robust” [Raman 99]
- “Soft-state protocols provide .. greater robustness to changes in the underlying network conditions…” [Sharma 97]
- “obviates the need for complex error handling software” [Balakrishnan 99]

What does this mean?

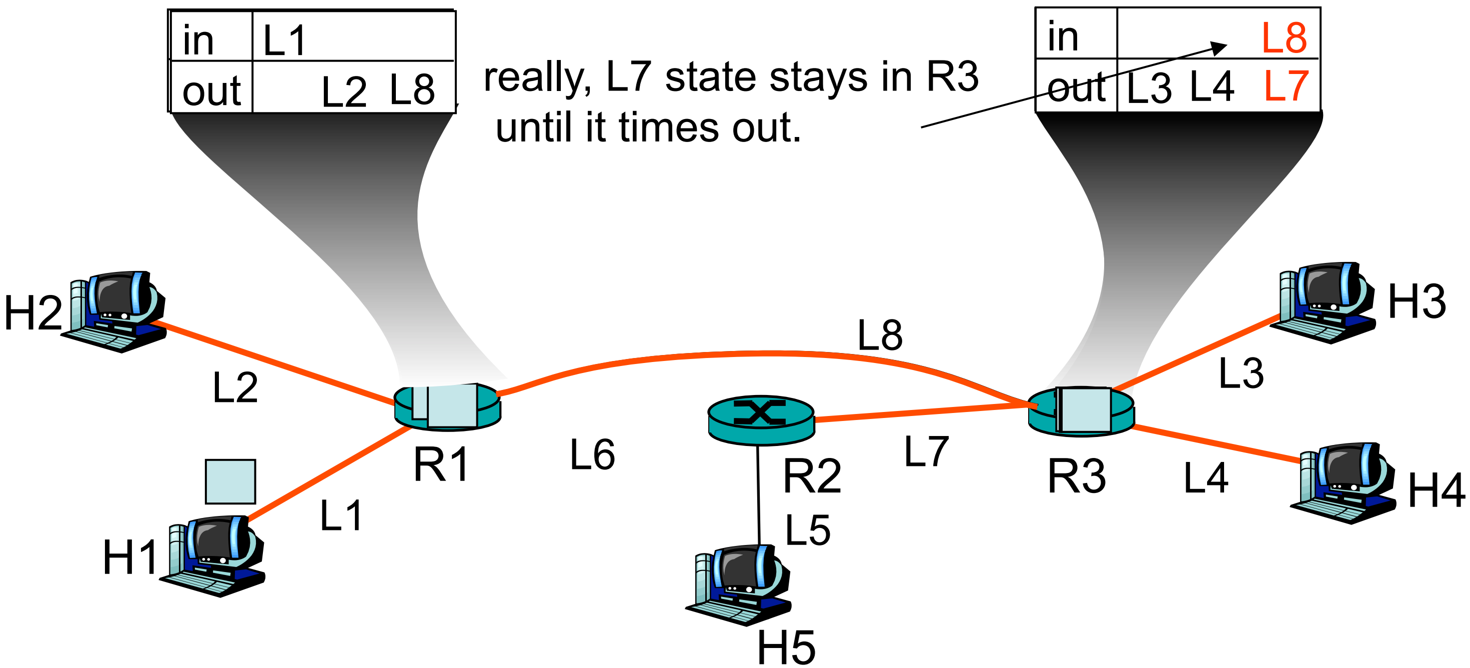
Soft-state: "easy" handling of changes

- Periodic refresh: if network "conditions" change, refresh will re-establish state under new conditions
- example: RSVP/routing interaction: if routes change (nodes fail) RSVP PATH refresh will *re-establish* state along new path



Soft-state: "easy" handling of changes

- L6 goes down, multicast routing reconfigures but…
- H1 data no longer reaches H3, H4, H5 (no sender or receiver state for L8)
- H1 refreshes PATH, establishes *new* state for L8 in R1, R3
- H4 refreshes RESV, propagates upstream to H1, establishes new receiver state for H4 in R1, R3



Soft-state: “easy” handling of changes

- “recovery” performed transparently to end-system by normal refresh procedures
- no need for network to signal failure/change to end system, or end system to respond to specific error
- less signaling (volume, types of messages) than hard-state from network to end-system but…
- more signaling (volume) than hard-state from end-system to network for refreshes

Soft-state: refreshes

- refresh messages serve many purposes:
  - trigger: first time state-installation
  - refresh: refresh state known to exist ("I am still here")
  - <lack of refresh>: remove state ("I am gone")
- challenge: all refresh messages unreliable
  - problem: what happens if first PATH message gets lost?
    - copy of PATH message only sent after refresh interval
  - would like triggers to result in state-installation a.s.a.p.
  - enhancement: add receiver-to-sender refresh_ACK for triggers
  - sender initiates retransmission if no refresh_ACK is received after short timeout
  - e.g., see paper "Staged Refresh Timers for RSVP" by Ping Pan and Henning Schulzrinne
  - approach also applicable to other soft-state protocols

Signaling Spectrum

periodic refresh

Soft-state (SS)

SS + explicit removal
IGMPv2/v3

SS + reliable trigger/ removal
ST-II

Hard-state

SS + reliable trigger
RSVP new version

- best effort periodic state installation/refresh
- state removal by time out
- RSVP, IGMPv1

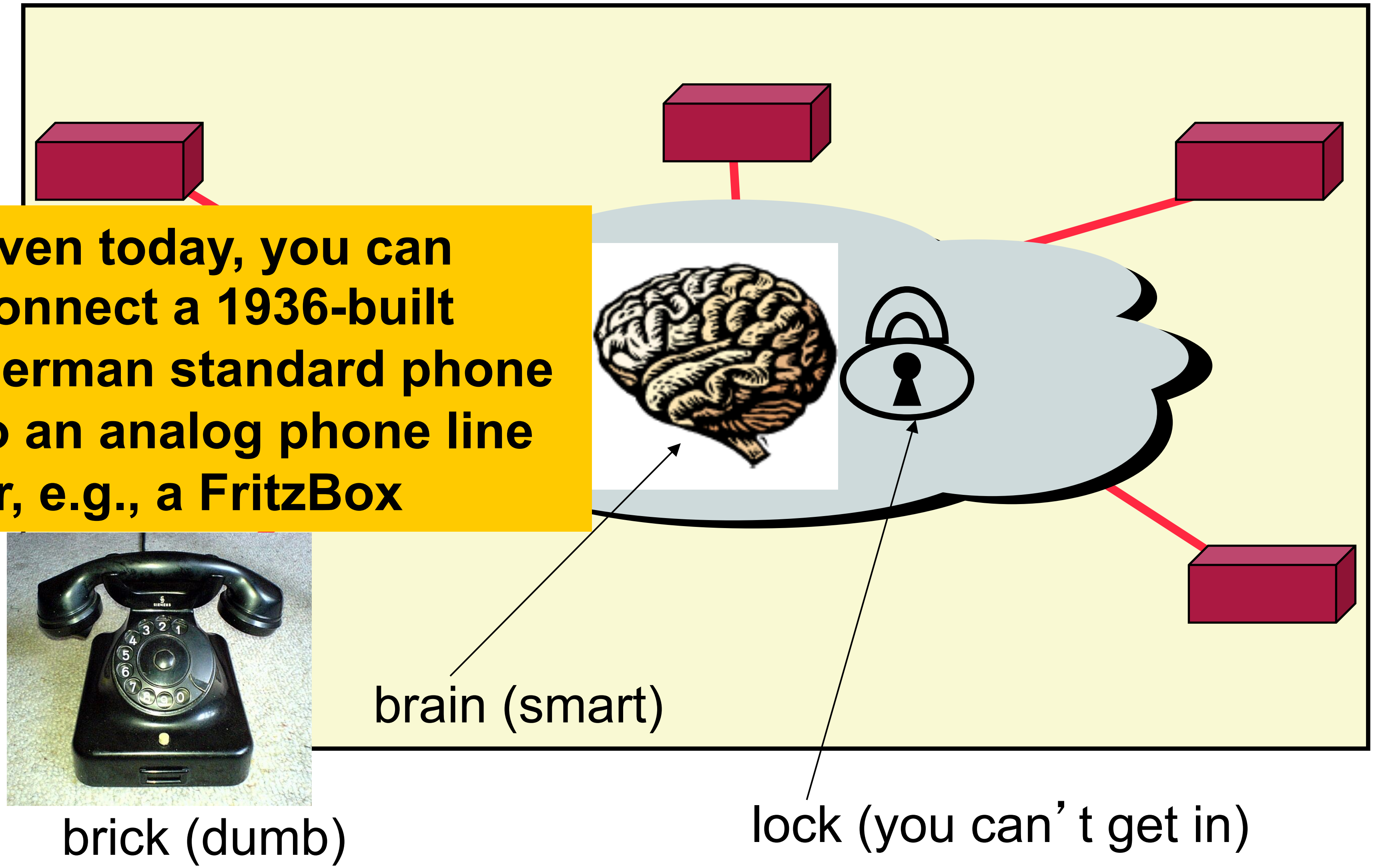
- reliable signaling
- explicit state removal
- requires additional mechanism to remove orphan state
- Q2931b ATM Signalling

Chapter:
Internet Architecture

Structure

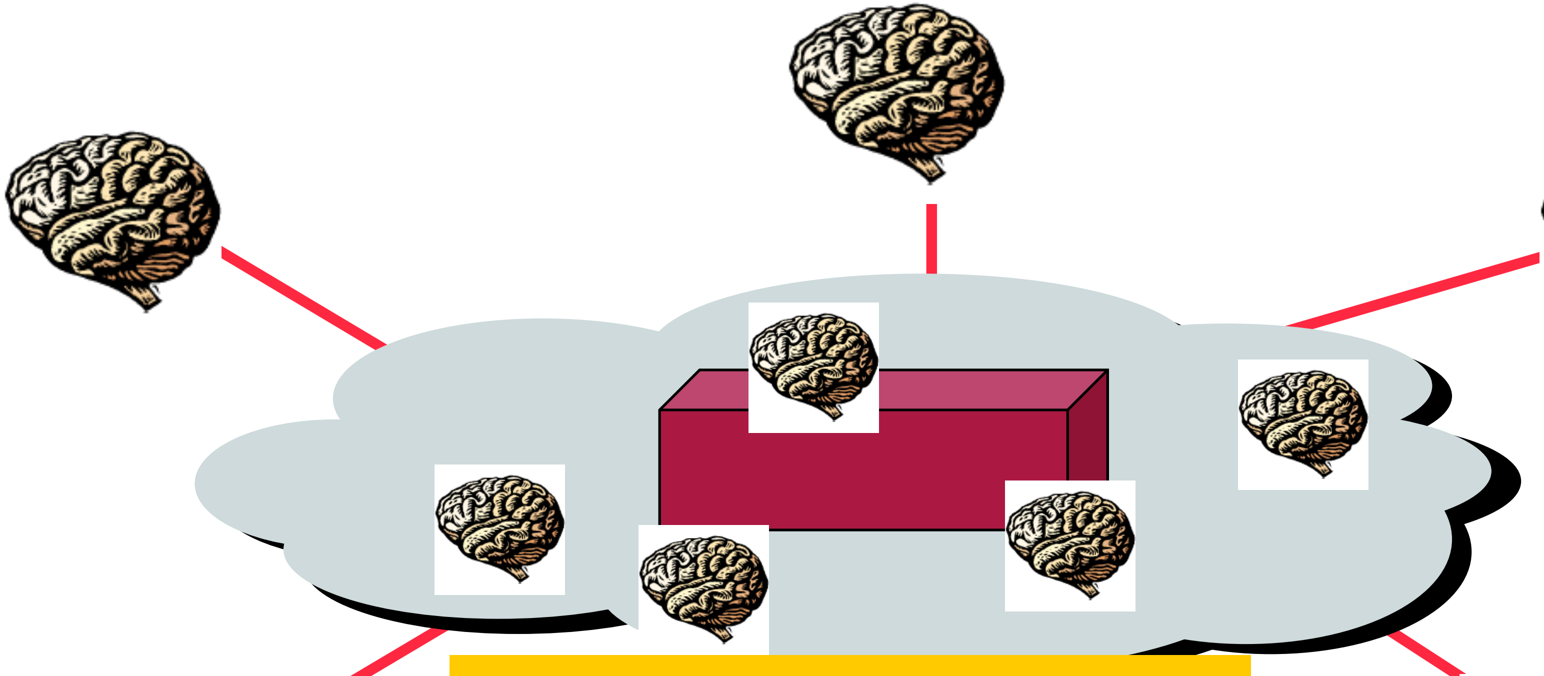
- Internet architecture
  - Requirements, assumptions
  - Design decisions
- Shortcomings and „Future Internet“ concepts
  - „Legacy Future Internet“: IPv6, SCTP, …
  - Security
  - QoS, multicast
  - Economic implications, „tussle space“
  - Mobility and Locator–ID split
  - In-network congestion control
  - Modules instead of layers
  - Delay-tolerant/disruption-tolerant networking
  - Content-based networking/Publish–subscribe architectures
  - Evolutionary vs. Revolutionary/Clean-slate
  - Design for Privacy

Common View of the Telephone Network



Even today, you can connect a 1936-built German standard phone to an analog phone line or, e.g., a FritzBox

Common View of the Internet



Stateless „dumb core"?
No longer true: Firewalls, transparent proxies, smart filtering, middleboxes, complex routing protocols like BGP, …

The Internet End-to-End principle

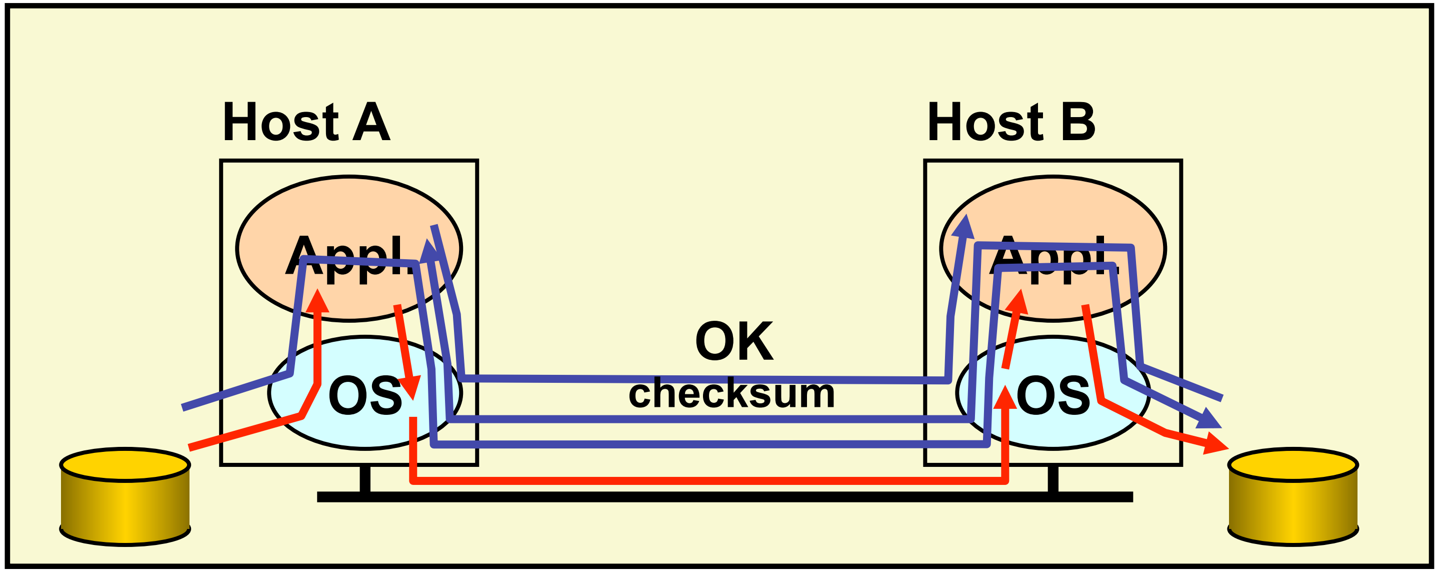
Internet End-to-End Principle

- "…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 fail or misbehave (bugs)?
- Is reliable communication sufficient?
  - No – what happens in case of, e.g., 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 at 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

Design Philosophy

- End-to-End Principle – [Saltzer ' 81]

  Saltzer, J. H., D. P. Reed, and D. D. Clark (1981) "End-to-End Arguments in System Design". In: Proceedings of the Second International Conference on Distributed Computing Systems. Paris, France. April 8–10, 1981. IEEE Computer Society, pp. 509-512
  http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf

- Internet Design Philosophy – [Clark' 88]

  The Design Philosophy of the DARPA Internet Protocols

  by David D. Clark, Proc. SIGCOMM '88, Computer Communication Review Vol. 18, No. 4, August 1988

  http://ccr.sigcomm.org/archive/1995/jan95/ccr-9501-clark.pdf

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 communication services
3. Must accommodate a variety of networks
4. Allow distributed management
5. Be cost effective
6. Allow host attachment with a low level of effort
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
- Remark: "Internet was built to survive global thermonuclear war" = urban legend; untrue

2. Support Multiple Types of Communication 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 (ToS bits in IP header), but this has never happened on the large scale (Why?)

3. Variety of Networks

- Very successful (why?)
  - Because of minimalism
  - Only requirement from underlying network: to deliver a packet with a "reasonable" probability of success
- …but does *not* require:
  - Reliability
  - In-order delivery
  - Bandwidth, delay, other QoS guarantees
- The mantra: IP over everything
  - Then: ARPANET, X.25, DARPA satellite network, phone lines, …
  - Today: Ethernet, DSL, 802.11, GSM/UMTS, LTE, …
  - Soon: 5G

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

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

Other Goals (Cont)

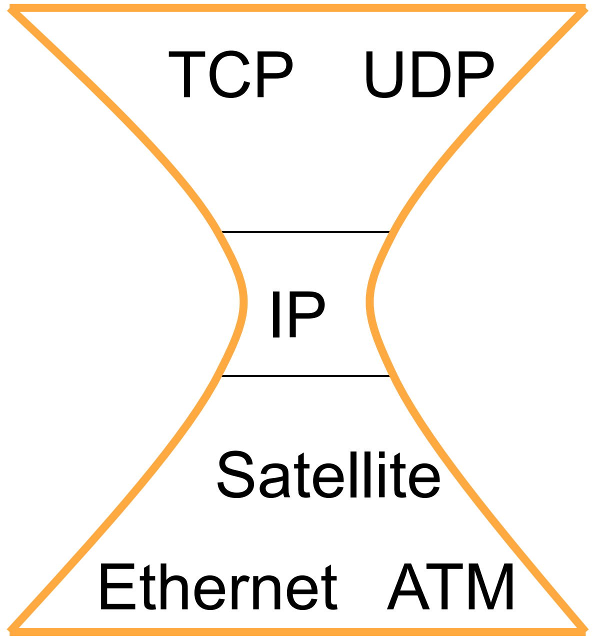
- Low cost of attaching a new host
  - not a strong point → higher than other architecture because the intelligence is in hosts (e.g., telephone vs. computer)
  - bad implementations or malicious users can produce considerably harm
- Accountability
  - Not a strong point: no financial interests (research network!)
  - Today: challenging in the view of privacy expectations vs. other requirements

What About the Future

- Datagram not the best abstraction for:
  - resource management,accountability, QoS
- new abstraction: flow (see IPv6)
  - flow not precisely defined (when does it end?)
  - IPv6: difficulties to make use of flowids
- 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
  - IP hides transport/application details from network
  - IP hides network details from transport/application
- Stateless architecture
  - No per-flow state inside network
  - Intelligence (i.e., state keeping) in end hosts, but not in core



IP hourglass

The KISS principle

- KISS = “Keep it simple, stupid!”
- Success of…
  - IP
  - Ethernet
  - RISC processors
  - SIP (vs. H.323)
- “Building complex functions into network optimizes network for small number of services, while substantially increasing cost for uses unknown at design time”

Internet Evolution

Internet architecture: Some explicit or implicit assumptions

- A research network
  - No economic/business/judicial aspects, no competition
  - Cooperative, perhaps even altruistic participants
- Knowledgeable and responsible end users; administrators even more so
- Almost no malicious participants
  - Perhaps some malicious users? (➔ password protection),
  - …but no malicious systems administrators,
  - …and certainly no malicious network operators
- A couple of thousand nodes, perhaps a million users
- No mobility: End hosts will not shift their position within network
- Most links are wired; packet loss indicates network congestion
- Just a temporary solution

- **…and yet it still works!? Amazing!**

But that was yesterday

… what about tomorrow? Or even: today?

Rethinking Internet Design

What's changed?

1. Operation in untrustworthy world

- Endpoints can be malicious
- If endpoint not trustworthy, then want trustworthy network
  ⇨ more mechanism in network core

2. 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, Information-Centric Networking)

Rethinking Internet Design

What's changed (cont.)?

3. ISP service differentiation

- ISP doing more (than other ISPs) in core is competitive advantage

4. Rise of third party involvement

- Interposed between endpoints (even against will of users)
- e.g., Chinese government, US recording industry

5. 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-to-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]
Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World
ACM Transactions on Internet Technology, Vol. 1, No. 1, August 2001, Pages 70 –109, http://groups.csail.mit.edu/ana/Publications/PubPDFs/Rethinking%20the%20design%20of%20the%20internet2001.pdf

Technical response to changes

- 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

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

  ⇨however, dealing with trust is hard (non-transient properties!)

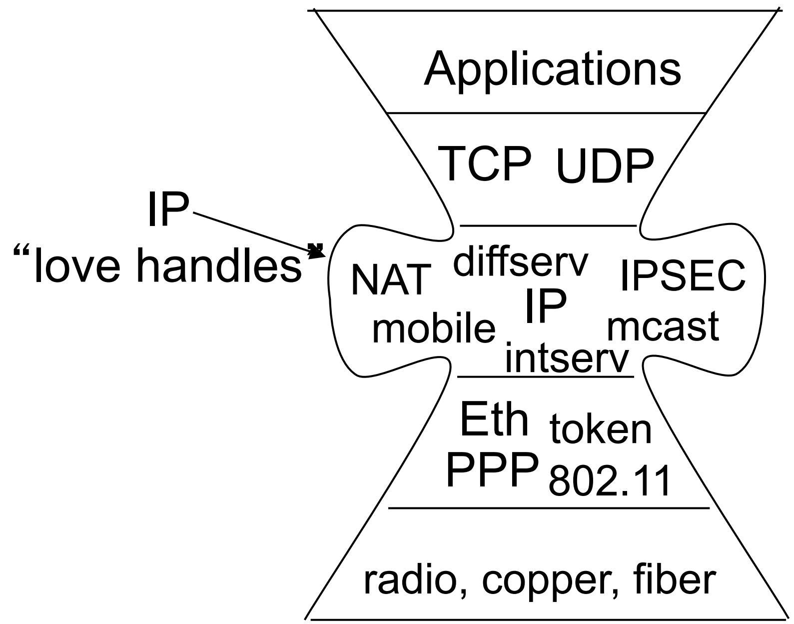
Technical response to changes

- Add functions to the network core:
  - Filtering firewalls
  - Application-level firewalls
  - NAT boxes
  - Transparent Web proxies

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.
    Difficult!

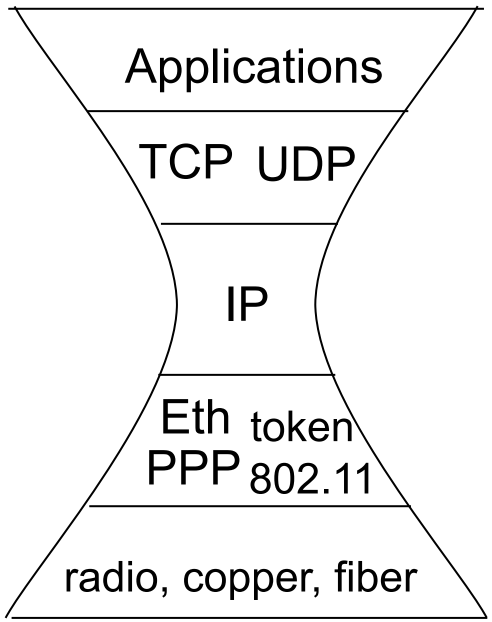
Big picture: supporting new applications – losing the IP hour glass figure?



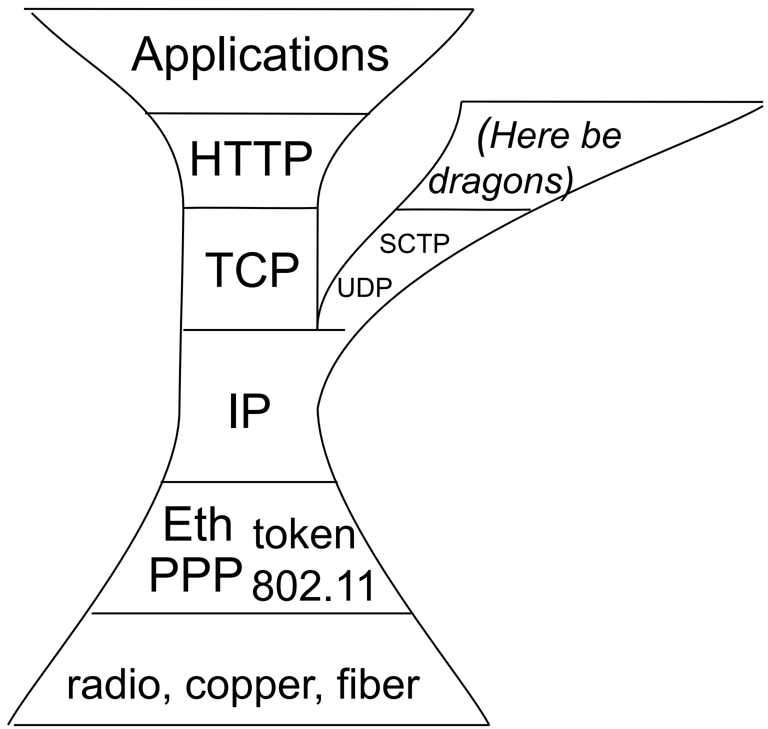
IP "hourglass"

Middle-age IP "hourglass"?

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

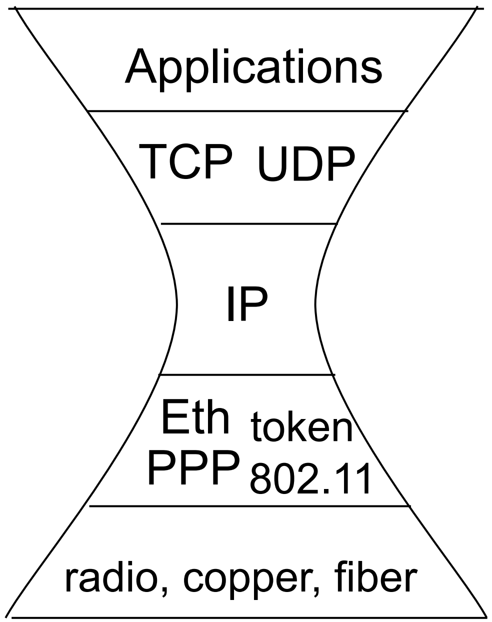


IP “hourglass”

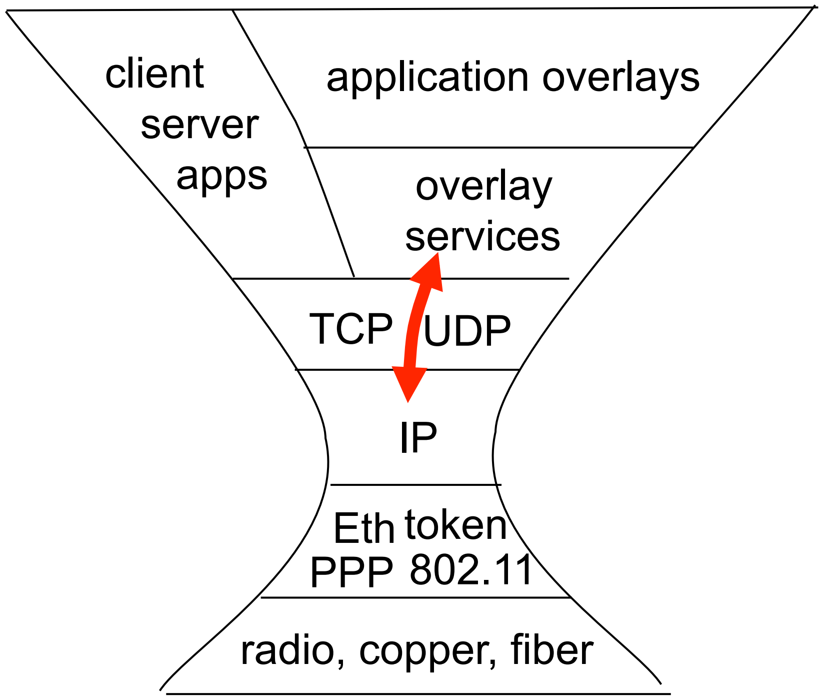


HTTP/IP “long-neck hourglass”

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



IP "hourglass"



Design for Privacy

Privacy by Design

- Design for Privacy - [Hoepman 2013]
  Privacy Design Strategies

  by Jaap-Henk Hoepman, Privacy Law Scholars Conference (PLSC) 2013, http://arxiv.org/abs/1210.6621
- Definitions
  - "A *privacy design strategy* is a design strategy that achie-ves (some level of) privacy protection as its goal."

    [Hoepman 2013]
  - "Privacy-Enhancing Technologies is a system of ICT measures protecting informational privacy by eliminating or minimising personal data thereby preventing unnecessary or unwanted processing of personal data, without the loss of the function- ality of the information system."

    [Borking and Blarkom et al. 2003] (ref. 35 in Hoepman 2013)

8 Privacy Design Strategies

1. Minimise
   - The amount of personal data that is processed should be restricted to the minimal amount possible
2. Hide
   - Any personal data, and their interrelationships, should be hidden from plain view
3. Separate
   - Personal data should be processed in a distributed fashion, in separate compartments whenever possible
4. Aggregate
   - Personal data should be processed at the highest level of aggregation and with the least possible detail in which it is (still) useful

[Hoepman 2013]

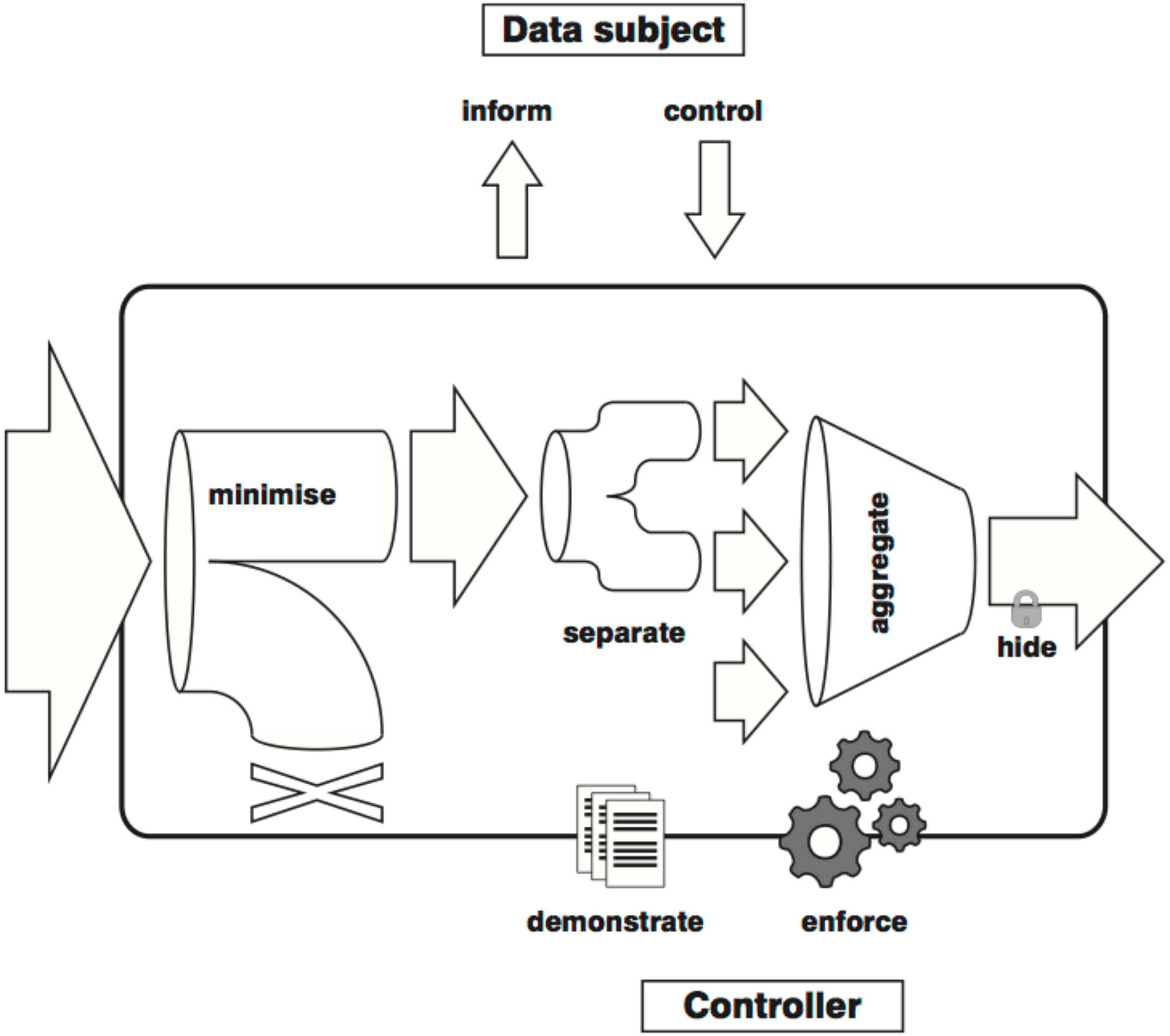
8 Privacy Design Strategies

5. Inform
   - Data subjects should be adequately informed whenever personal data is processed.
6. Control
   - Data subjects should be provided agency over the processing of their personal data
7. Enforce
   - A privacy policy compatible with legal requirements should be in place and should be enforced
8. Demonstrate
   - Be able to demonstrate compliance with the privacy policy and any applicable legal requirements

[Hoepman 2013]

8 Privacy Design Strategies

- process flow metaphor of the privacy design strategies



[Hoepman 2013]

Future Internet concepts

- Shortcomings and „Future Internet“ concepts
  - Security
  - QoS, multicast
  - Economic implications, „tussle space“
  - Mobility and Locator–ID split
  - In-network congestion control
  - Modules instead of layers
  - Delay-tolerant/disruption-tolerant networking
  - Content-based networking/Publish–subscribe architectures
  - Evolutionary vs. Revolutionary/Clean-slate

FIND: Future Internet Network Design

- New long-term US NSF initiative
- Questions:
  - Requirements: for the global network of 15 years from now - what should that network look like and do?
  - How would we re-conceive tomorrow's global network today, if we could design it from scratch?
- Major thrusts:
  - Security, manageability, mobility (DTN, naming, wireless)
  - I.e.: what the original Internet didn't get right

The Internet has security issues

- Problem #1: Cannot protect from unwanted traffic
  - Spam
  - DoS attacks
  - Wustrow, Karir, Bailey, Jahanian, Huston: *Internet background radiation revisited.* Proceedings of ACM/USENIX Internet Measurement Conference, 2010
- Solutions
  - Protocols
    - Cookies (e.g., TCP SYN cookies)
    - Permission-based sending (PBS) – c.f. Henning Schulzrinne
  - Treating the symptoms
    - Spam filters
    - Rate limiting at firewall
    - Tar pits, honey pots
    - Network intrusion detection systems (NIDS)
    - …

Multicast and QoS

- Multicast routing protocols (MOSPF, PIM, …) exist and work
- QoS protocols (IntServ, DiffServ, …) exist and work
- IP header *and* Ethernet header (802.1p) contain ToS bits

- …but no end user application is using it!
  - Multicast: Would be nice for online TV
  - QoS: Would be nice for throttling P2P and ftp downloads while increasing responsiveness of ssh and games and stability of VoIP calls and video streaming
- At least some „invisible“ usage
  - Prioritization of specific traffic within company networks
  - ISPs may give QoS guarantees for VPNs
  - TV over IP („Triple play“) uses multicast, but application not directly accessible by user

Why don’t ISPs offer multicast or QoS to end users?

1. Same chicken–egg problem / vicious circle as with IPv6:

No applications that really need it

No demand

No deployment in network

2. Who should pay once traffic crosses AS boundaries?
   - Who pays „expedited forwarding“?
     Sender AS, receiver AS, both?
   - Who pays in-network duplication for multicast?
     Sender AS, receiver ASes, or entire network?
   - How can sender/receiver be charged?
   - How can multicast sender know how much it will be charged?

Economic aspects, conflicting interests: „Tussle“

- Internet participants
  - Different stakeholders
  - Competition
  - Conflicting interests
- Examples
  - Users want to share music and videos – GEMA/RIAA don‘t
  - Users want secret communication – governments don‘t
  - ISPs need to cooperate – but are fierce competitors
- Call this aspect „tussle“
  - Internet architecture only partially reflects this (BGP policy routing)
  - Tussle Space: Future Internet architecture should anticipate various kinds of tussle and integrate defined mechanisms

Clark, Sollins, Wroclawski, Braden: Tussle in Cyberspace: Defining Tomorrow‘s Internet. Proceedings of ACM SIGCOMM, 2002

Content-based networking and publish–subscribe architectures (I)

- Observation:
  - IP addresses *hosts*
  - Browsers, P2P clients etc. address *content objects:* Specific Web pages, MP3 files with specific music, …
- Idea:
  - Address content chunks instead of hosts
  - Routers can replicate and/or cache popular chunks
- Requesting chunks:
  - Send interest/subscription request into network
  - Request will be forwarded from router to router
  - If matching content chunk(s) found, send them to requester

Content-based networking and publish–subscribe architectures (II)

- A lot of features automatically built in:
  - Multicast (even asynchronously!)
  - In-network caching
  - Resilience: If one router with content fails, it still will be available on other routers
  - Delay-tolerant networking: Routers cache contents anyway, so why not have the caching routers roam around as well?
  - Some protection from DoS attacks: I only get traffic that I requested

Content-based networking and publish–subscribe architectures (III)

- Some issues being addressed
  - Authenticity: How to make sure that malicious users cannot inject a fake version of, e.g., an online banking service?
  - Routing: How do routers know which interest packets should be forwarded to which neighbour(s)?
  - Versioning: How to make sure that old versions of a content object are quickly replaced in router caches (e.g., content object „current DAX level“ or „Mensa food plan“)
  - Protocol logic:
    - Subscription („send me all matching chunks“) vs. requests („send me one matching chunk“)
    - Timeouts
  - Protection from flooding induced by excessive subscription
  - Addressing scheme

Content-based networking and publish–subscribe architectures (IV)

- OK, sounds good for things like YouTube, heise.de, etc.
- But what about obvious peer–peer sessions? (ssh, VoIP, etc.)

- Solution:
  - Subscribe to contact requests
  - If contact request is received, subscribe to answer packets of contact request originator
  - Start sending out own data (e.g., own voice)
  - Receive answers from peer (e.g., acknowledgement packets; other‘s voice)

Content-based networking and publish–subscribe architectures (V)

- Some thoughts on the address length: How much do we need?
- Current Internet
  - IPv4: 32 bits = 4 billion addresses (about 30% used)
  - IPv6: 128 bits
- Consider something like a worst-case scenario:
  - Assume *every atom* is used to store one information chunk!
    - About 10^{80} particles in the visible universe
  - Every chunk changes its state every 10^{-44}s! (Planck Time)
  - For 1 million years!
  - We waste 99% of the address space! (IPv4: only 60% wasted)
  - How many bits do we need?
    - $\log_2 \left(10^{80} \cdot (10^{44} \cdot 60) \cdot (60 \cdot 24 \cdot 365 \cdot 10^6) \cdot 100\right)$
    = 463 bits = 58 Bytes. (N.B.: IPv6 header+TCP header = 56 Bytes)
    - One of the rare cases where exponential growth is in our favour!

Future Internet approaches

Revolutionary (clean slate)

- Today's Internet is broken by design
- Trying to fix it leaves us with *-over-HTTP-over-TCP-over-IP, i.e., with something like the memory model of Intel x86, 110V vs. 230V, and 50Hz vs. 60Hz power, …
- New architecture will be radically different
- ➔ Let's throw everything away and start completely anew to ~~get it right from the beginning~~ introduce new design mistakes

Evolutionary

- The Internet has been amended many times in the past:
  - Adding congestion control to TCP
  - Introduction of DNS instead of distribution of `/etc/hosts` text files
  - Introduction of classless interdomain routing instead of Class-A, Class-B, Class-C networks
  - Introduction of SSL, IPSec, ssh, …
  - Introduction of Multicast, ToS bits
  - Introduction of IPv6
- ➔ Let's fix the shortcomings incrementally by introducing new protocols: ~~Never change a running system~~ Create a truly unmanageable behemoth of conflicting protocols

Future Internet: Some readings

- Mark Handley: *Why the Internet only just works.*
  BT Technology journal, 2006
- Anja Feldmann: *Internet Clean-Slate Design: What and Why?*
  Editorial note, ACM CCR, 2007
- Akhshabi, Dovrolis: *The evolution of layered protocol stacks leads to an hourglass-shaped architecture.*
  Proceedings of ACM SIGCOMM, 2011

The end!