

Strategic Choices in  
Designing the Internet

*Suzhou Institute  
for Advanced Study  
of USTC*

Michael J. O'Donnell  
*The University of Chicago*  
Assistant: Xiang Sen, USTC

(12 June 2004; revised 17 July 2004)

- Instructor: Michael J. O'Donnell  
odonnell@cs.uchicago.edu
- Assistant: Xiang Sen  
xiangsen@ustc.edu
- Web page:  
[http://221.6.69.11/~od/Strategic\\_Internet/](http://221.6.69.11/~od/Strategic_Internet/)
- Schedule:
  - 09:00-12:00 lectures—15 minute break around 10:30
  - 12:00-14:00 lunch, free discussion, work on exercises
  - 14:00-16:00 questions, lectures

- Required exercises: 1 set each day; free discussion; write your answer in English, email to Xiang Sen next day.
- Optional advanced exercises: Select from the list; work on your own; email to Xiang Sen by 19 July.
- 80-89: good job on all required exercises.
- 90-100: also very good job on at least one advanced exercise.

## Nature of course

- Consider a logical sequence of design choices leading to the Internet.
- Evaluate the reason for each choice, and the impact of each choice.
- Preview future choices.

Logical sequence is not exactly historical—  
a simplification of history.

## Why study design choices?

- Help design the next Internet
- Be a good netizen
- Anticipate future developments
- Use design principles in applications
- Learn to read primary sources (RFCs)

## Modern method, ancient wisdom

“Designing application-level protocols is really no different than designing core network protocols.”  
—Peterson and Davie

“The core and the surface  
Are essentially the same.”  
—Lao

Existence is beyond the power of words  
To define:  
Terms may be used  
But are none of them absolute.  
In the beginning of heaven and earth there were  
no words,  
Words came out of the womb of matter;  
And whether a man dispassionately  
Sees to the core of life  
Or passionately  
Sees the surface,  
The core and the surface  
Are essentially the same,  
Words making them seem different  
Only to express appearance.  
If name be needed, wonder names them both:  
From wonder into wonder  
Existence opens.

—Lao, *Tao De Jing* (translation by Witter Bynner)

## Design layers come from scope of agreements

- Choices that affect more agents are more fundamental.
- Choices that constrain other choices are more fundamental.

If we're lucky, these two dimensions are reasonably compatible.

## Layers of Internet design

Routing	Capacity management	Address management	Naming (relocation)
Addressing			
Delivery (forwarding)			

## Internet protocols in design layers

- Delivery/forwarding: IP
  - Addressing: IP, UDP, TCP
    - \* Routing: ARP, ICMP, RIP, OSPF, CIDR, IGP, BGP
    - \* Capacity management: TCP, ICMP
    - \* Address management: ARP
  - Naming/relocation: DNS

Why not addressing below delivery?

## Choices to study

1. Stateless message forwarded by stateless router
2. Two-level addresses
3. End-to-end acknowledgment, not hop-by-hop
4. End-to-end congestion control
5. Best-effort, no reservation
6. DNS: name service, no handle service
7. Future authentication: chain of trust?

## Unusual evaluations of choices

Evaluate choices by principles, results:

- good
- bad
- neutral
- illusory: not really a choice

Principles don't tell us what choices to make. Choice of relevant principle and form of application are too open. Principles help us discuss and understand choices. (Analogy to analogy)

Sometimes a designer announces a decision, but some agents may ignore the decision without affecting the enterprise. These are illusory choices.

## RFC791

### Basic purpose of IP

#### 1.1 Motivation

The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses.

Very little information here.

- “Transmitting blocks of data” and “fixed length addresses” are technical choices.
- “datagrams,” “sources,” “destinations” are definitions.
- “hosts” is not quite accurate—contradicted later.

## RFC791

### Limitation of IP

#### 1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, ...

## RFC791

### Addressing is fundamental

#### 1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

- Addressing turns out to be the strategically crucial function.
- Fragmentation would be important if link technology developed differently.
- “transmit internet datagrams toward their destinations” summarizes forwarding.

## RFC791

### Protocol: common rules

#### 1.4. Operation

...

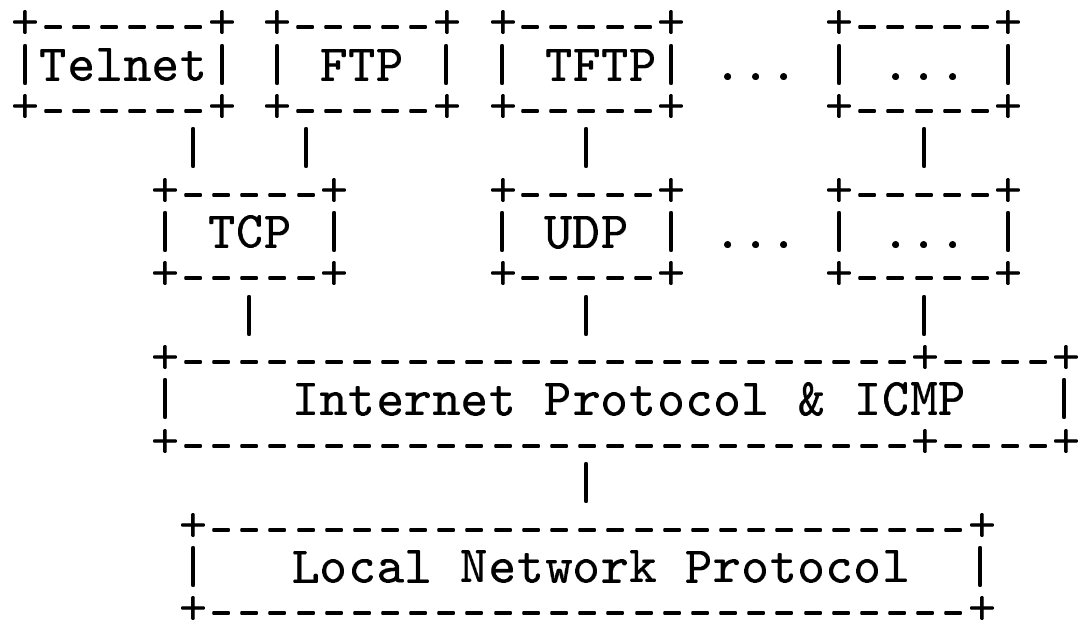
The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields.

IP is determined by the rules that must be shared by all routers (“internet module”s).

# RFC791

## Protocol relationships

### 2.1. Relation to Other Protocols



Protocol Relationships

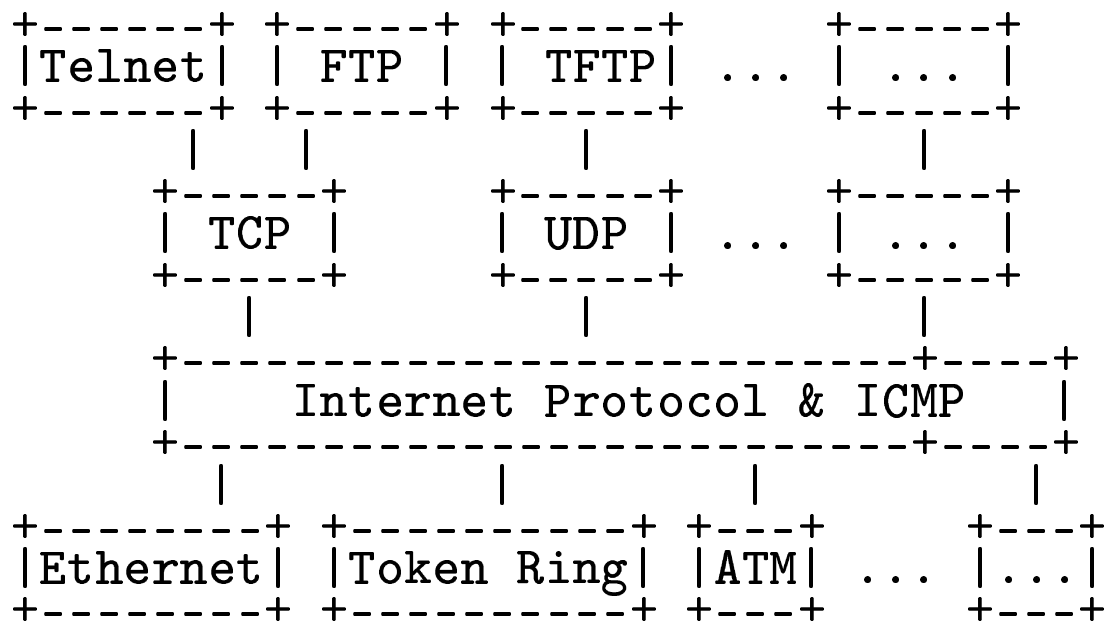
Figure 1.

## OSI network model vs. Internet

OSI	Internet	
Application	Telnet, FTP, ...	
Presentation		
Session	(open/close)	
Transport	TCP	UDP
Network	IP	
Data link	Local Network Protocol	
Physical		

## RFC791

### Protocol relationships, my version



From P&D Figure 1.14

IP is a *design* bottleneck, because other protocols and subsystems may vary independently as long as they agree on IP in between.

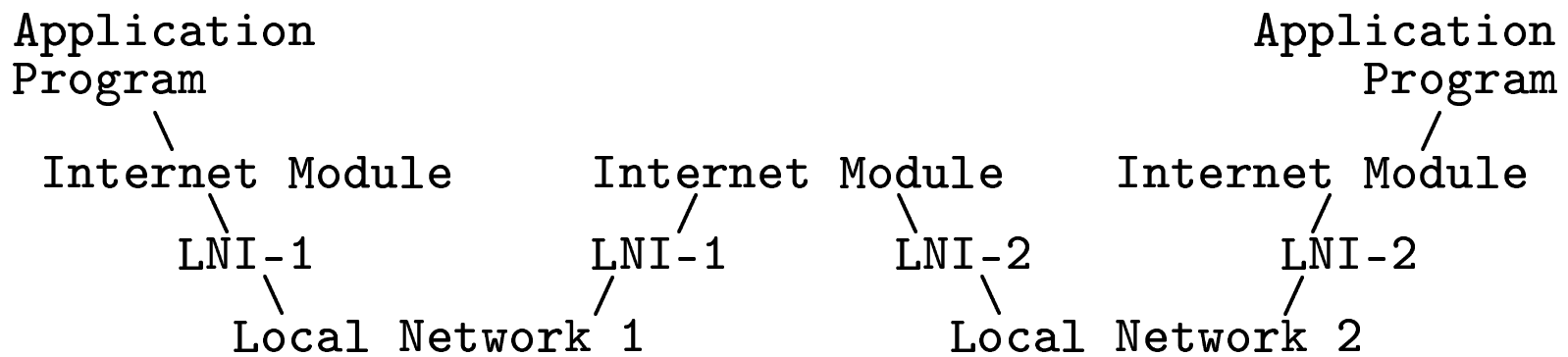
The diagram is not really the “network architecture” [P&D] of the Internet. Rather, it is a paradigm architecture for particular suites of services and implementation within the Internet. But it can be, should be, and is varied according to need.

IP is the part that all routers must agree on. The real “network architecture” is IP, plus infrastructural protocols, such as routing protocols and DNS.

# RFC791

## Packet traversing network and protocols

### 2.2. Model of Operation



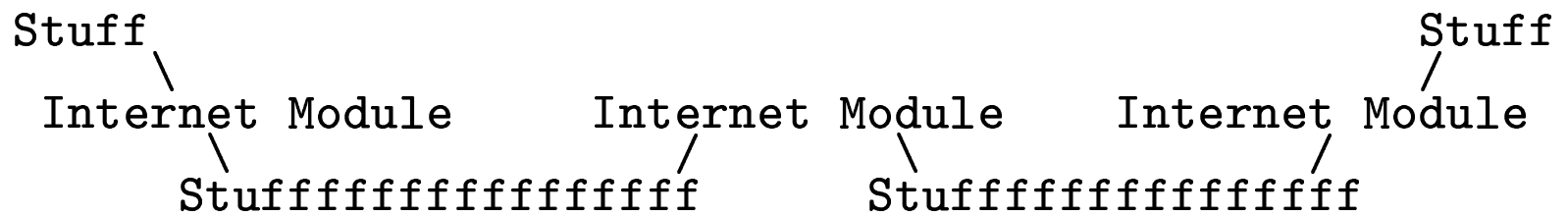
Transmission Path

Figure 2

# RFC791

## Packet traversing network and protocols, my version

### 2.2. Model of Operation



The basic integrity of the Internet derives from the fact that each *Internet Module* (router or protocol implementation) sees the appropriate IP datagram at each hop. The particular interaction between each *Internet Module* and its *LNIs*, and the entire organization of the *LNIs* and *Local Networks* can be varied locally, as long as each *Internet Module* sees that IP datagram.

This is analogous to the way that implementations of *abstract datatypes* or *objects* may use any representation of data, even different representations at different times, as long as they respond correctly to every public procedure call.

Another analogy: in hardware, the electrical representation of bits may vary around the circuitry. This is fine as long as the signals received by memory and by output devices satisfy the appropriate standards.

## Violations of layering

- PPP over Ethernet
- IP tunnelling (over IP), mobile IP
- NAT—port incorporated into address
- Ethernet over IP  
(distributed virtual machines,  
Prescience Lab)

## RFC791

### Names, addresses, routes

#### 2.3. Function Description

...

#### Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses.

“An address indicates where it [what we seek] is” seems very vague, but anything more specific in this style of discussion turns out to be wrong—e.g., an address is not just the address of a host, nor of a network interface. There is technical meaning to the name/address/route distinction, but it depends on a very different type of discussion.

Later we investigate names, addresses, routes, and also handles more carefully. They are defined in terms of their portability, and who has authority to bind their interpretations.

## RFC791

### Addresses of what?

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

This contradicts the earlier assertion that addresses indicate hosts.

# RFC791

## Datagram Format

### 3.1. Internet Header Format

Version:	4 bits	4 bits
IHL:	4 bits	
Type of Service:	8 bits	4 bits
		Flow label: 24 bits
Total Length:	16 bits	16 bits
Identification:	16 bits	
Flags:	3 bits	
Fragment Offset:	13 bits	
Time to Live:	8 bits	8 bits
Protocol:	8 bits	8 bits
Header Checksum:	16 bits	
Source Address:	32 bits	128 bits
Destination Address:	32 bits	128 bits
Options:	variable	

## IP forwarding algorithm RFC950

```
IF ip_net_number(dg.ip_dest) = ip_net_number(my_ip_addr)
  THEN
    send_dg_locally(dg, dg.ip_dest)
  ELSE
    send_dg_locally(dg,
      gateway_to(ip_net_number(dg.ip_dest)))
```

## IP forwarding algorithm in textbooks

- **if** *address matches here* **then** keep
- **else** forward to better neighbor
  - **if** address matches neighbor *A* **then** deliver to *A*
  - **else** forward by route table

## IP forwarding essentials my version

Choice: stateless forwarding

- **if** *address matches here* **then** keep
- **else** forward to better neighbor

- The algorithm contains very little choice, but illuminates necessities.
- Outer structure (keep or forward) just says that we won't send away a packet addressed to us.
- Inner structure of forward (deliver or iterate) merely says we use underlying service when possible.
- Use of table just recognizes that network topology is fluid, so there must be a local representation of local routing information.
- Definition of "better" neighbor is unconstrained, as long as it eventually leads to delivery:
  - closer to target, or
  - better routing information, or ...

## Choice by omission

Key choices in forwarding algorithm:

- *address* doesn't change
- router state doesn't change
- no control messages

If *address* had changed, we wouldn't call it "*address*."

Each packet must have some token controlling delivery.

Stateless forwarding causes that token to be an absolute address.

## Why is forwarding fundamental?

**Principle: design infrastructure bottom-up.**

1. Multihop network is a system of forwarding routers (unavoidable).
2. Try simplest forwarding algorithm that provides value.

Design applications top-down; design infrastructure bottom-up.

Infrastructure overly constrained by intended application usually fixes constraints that prove unfortunate in the future.

The most important applications are the ones we haven't thought of yet.

E.g., in 1970, suppose someone tried to create WWW.

WWW was possible because the more primitive internet allowed ftp, gopher, etc. as precursors.

## Why stateless forwarding?

**Principle:** in distributed system, keep information local.

**Principle:** for robust system, avoid dependence on previous correctness.

- Use of state creates dependence on previous calculation.

## Why no control messages?

**Principle:** minimize overhead of critical resource.

- Purpose of network is to provide communication resource to applications.
- Avoid messages that are not critical to applications.

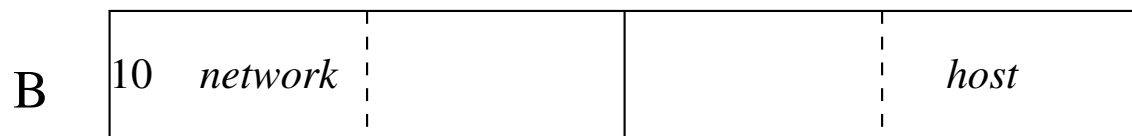
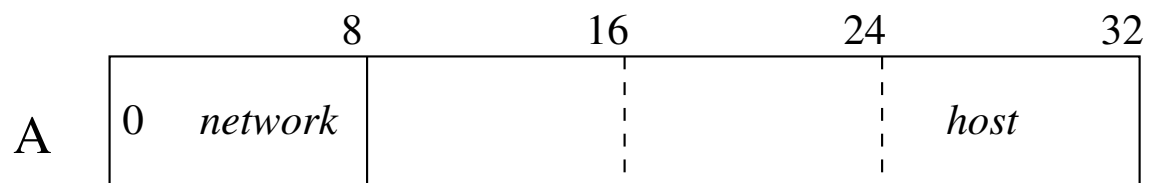
## Absolute address vs. directions

**History:** *UUCP* used directions naming all intermediate hosts: `host1!host2!...!hostn`

- Port to new sender?
  - address: just copy
  - directions: must translate
- Translate directions through net?
  - Tricky—reversibility, cycles, long routes.
  - Can't send directions out-of-band.
- Look up directions from address?
  - Need directions to the directory.

## Official definition of IP address

Choice: Two-level address (*network/host*)



## Choice of two-level address is illusion

### History:

1. 1980–81: RFC 760, 791 2-level addresses
2. 1984–85: RFC 917, 940, 950 subnets
3. 1992–93: RFC 1338, 1519 CIDR

Subnetting/CIDR may be practiced locally, transparently to outsiders

- Subnet is virtual host in larger net.

## RFC950

### locality of subnetting

The third approach is to explicitly support subnets. This does have a disadvantage, in that **it is a modification of the Internet Protocol**, and thus requires changes to IP implementations already in use (if these implementations are to be used on a subnetted network). However, these changes are relatively minor, and once made, yield a simple and efficient solution to the problem. Also, the approach **avoids any changes that would be incompatible with existing hosts on non-subnetted networks.**

## RFC950 incremental subnetting

When appropriate design choices are made, it is possible for hosts which believe they are on a non-subnetted network to be used on a subnetted one.

The choice of a specific 2-level address format vs. subnetting vs. CIDR is illusory *as a choice affecting IP*, but it is a serious choice for routing protocols, since it affects the format and meaning of routing messages (IGP, BGP).

Real choice: aggregated addresses.

- Aggregation allows compact route tables.
- Mostly a consequence of forwarding algorithm.

## CIDR routing tables

Net prefix	Network prefix as bits	Nexthop
C4.5E.2.0/23	11000100.01011110.0000001	A
C4.5E.2.0/22	11000100.01011110.000001	B
C4.5E.C0.0/19	11000100.01011110.110	C
C4.5E.40.0/18	11000100.01011110.01	D
C4.4C.0.0/14	11000100.010011	E
C0.0.0.0/2	11	F
80.0.0.0/1	1	G

Table 4.17 (Exercise 46, Chapter 4) P&D

## Routing protocols

Forwarding: use of routes.

Routing: discovery of routes.

- ARP: query and announce single hops
- IGPs: query and announce internal routes
- BGP: query and announce distributed routes

## ARP queries by host

ARP is algorithmically trivial, but deals with problems of distributed data.

- Local cache of  $\langle \text{IP}, \text{LANaddress} \rangle$  pairs
- Store only IPs
  - used by this host, or
  - targetting this host
- Broadcast query for unknown IP
- Monitor all broadcast responses
- Delete entry after 15 minutes

Internet history demonstrates that a loosely co-ordinated network can route scalably using address aggregation.

## END-TO-END ARGUMENTS IN SYSTEM DESIGN

J.H. Saltzer, D.P. Reed and D.D. Clark  
M.I.T. Laboratory for Computer Science

This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level.

To what extent are “end-to-end” and “high level” the same?

## End-to-End Arguments

### Failure at low level

The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

**Principle:** Many qualities at lower level affect only performance at higher level.

Sometimes the lower-level solution fails not because of lack of knowledge at the lower level, but because knowledge needs to be presented to the higher level, and the lower level is too fragmented. This happens in our intermediate acknowledgment example.

## End-to-End Arguments acknowledgment in file transfer

The extra effort expended in the communication system to provide a guarantee of reliable data transmission is only reducing the frequency of retries by the file transfer application.

## End-to-End Arguments disastrous example

A too-real example

[Many software source files lost at MIT. Programmers depended on reliable network links. Gateway computer corrupted the files.]

## No hop-by-hop acknowledgment

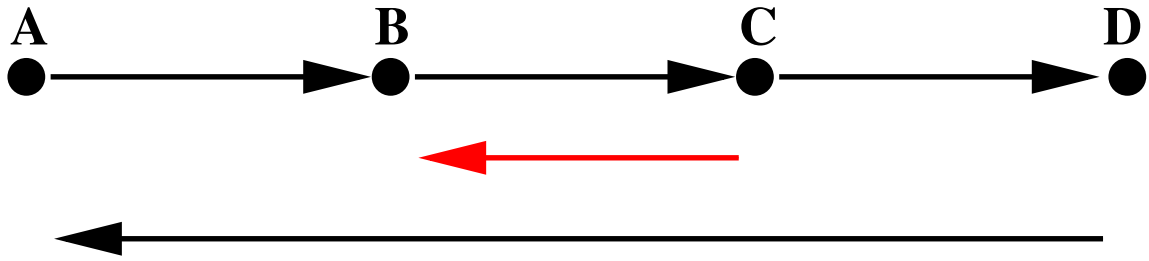
**Choice:** Don't require  
intermediate acknowledgment

**Principle:** Many qualities at lower level  
affect only performance at higher level.

Intermediate acknowledgment

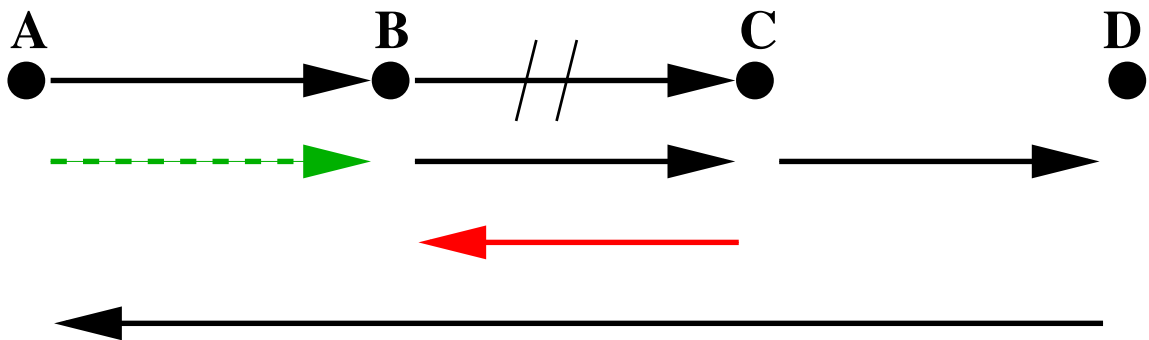
- doesn't provide reliability.
- seldom helps performance

## Intermediate acknowledgment



Successful delivery

---



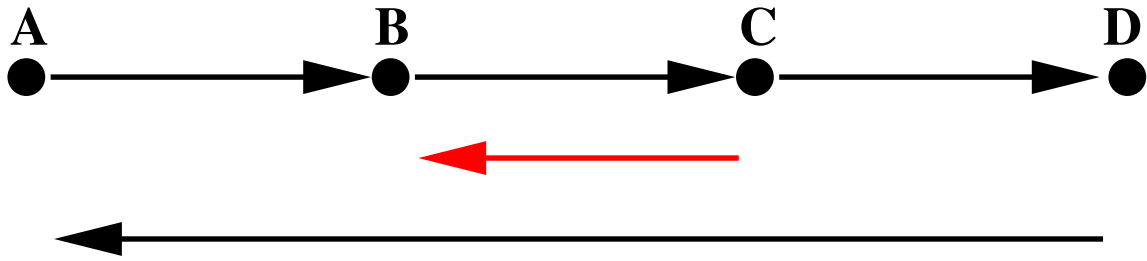
Failure in **B** segment

---

Successful delivery: CB acknowledgment and also DA acknowledgment so A can free its buffer

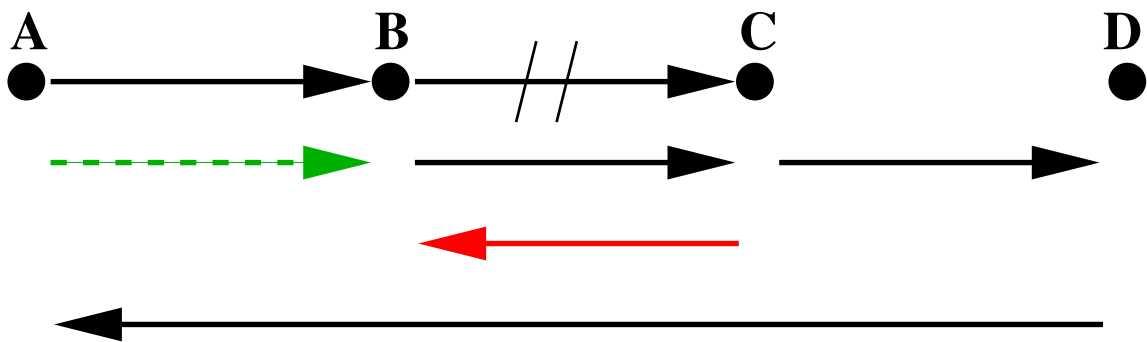
CB acknowledgment does not add reliability, but it may improve performance

## Intermediate acknowledgment



Successful delivery

---



Failure in **B** segment

---

Intermediate ack is cheaper if:

$$RTT(BC) \cdot buffer + (1 - \Pr(fail)) \cdot \text{cost}(CB) < \Pr(fail) \cdot \text{cost}(AB)$$

CB acknowledgment costs a message buffer during BCB round trip in *all* cases

CB acknowledgment costs a CB message in successful cases

CB acknowledgment saves AB retransmission when BC hop(s) fail(s)

The formula looks precise, but costs are not obvious.

- CB ack cost is mainly potential congestion interfering with other messages
- AB retransmission cost includes congestion and additional latency in this message

In principle, when  $C=D$ , CB ack may be bundled into DA ack. This requires collaboration by the final recipient, which is usually hard to arrange.

Additional costs (not in the formula):

- A might time out during BD retransmission, leading to duplicate packets in the net;
- lost CB ack leads to duplicate packets in the net;
- more complicated routers.

The additional costs all argue against the intermediate ack.

## Reasoning about intermediate ack

- Path cost is roughly proportional to length
- Typical  $\Pr(\textit{fail})$  is about 0.01
- Typical AB path is 1–30 hops

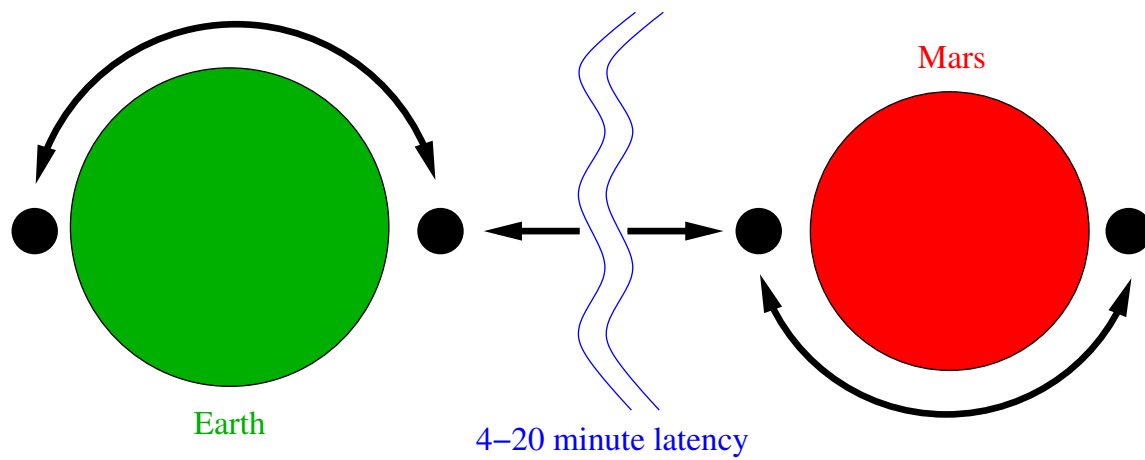
So expected cost of 1-hop ack is usually more than expected cost of retransmission from source.

## Where are the ends?

Failure might not be in the network itself, but in the sending or receiving host, at any level of application or OS.

## Example: InterPlanetary IP (IPN)

Where would intermediate acks help the most?



Acks across the space link are expensive:  
long buffer retention and expensive link.

Save packets that have just crossed the  
space link, and ack final receipt.

Buffer space is cheaper on Earth, so Mars-  
Earth packets are more likely to be buffered  
on Earth than Earth-Mars packets on Mars.

## End-to-End Arguments

lower level provides performance,  
not correctness

The amount of effort to put into reliability measures within the data communication system is seen to be an engineering trade-off based on performance, rather than a requirement for correctness.

## End-to-End Arguments more examples

- Acknowledgment/checksum—structurally the same
- Secure transmission of data
- Duplicate message suppression
- Guaranteeing FIFO message delivery
- Transaction management

End-to-End Arguments  
principles are not rules:  
where are the endpoints?

The end-to-end argument is not an absolute rule, but rather a guideline that helps in application and protocol design analysis; one must use some care to identify the end points to which the argument should be applied.

Example: real-time audio vs. audio file transfer

## Active Networking and End-To-End Arguments

Comment by David P. Reed, Jerome H. Saltzer, and David D. Clark

active networking ... comprises attaching programs to data packets, with the intent that those programs be executed at points within the network. ... The question being raised is whether or not this idea directly violates an end-to-end argument.

PostScript is an analagous active printing language, but lacks the resource-sharing problem.

The real problem is to throttle computing resources so that no user achieves deliberate or accidental DoS to others.

## End-to-End Arguments modularization of authority

End-to-end arguments address design more than implementation and implementation more than execution—that is, they suggest who should provide the code, not which box it should run on. ... Programmability in a lower layer ... defer[s] design choices upwards in the layering ... and later in time, even though the resulting functions may actually take place deep inside the network.

## End-to-End Arguments

design infrastructure bottom-up

A lower layer of a system should support the widest possible variety of services and functions, so as to permit applications that cannot be anticipated.

**Principle:** the most important uses of a tool are the ones that haven't been invented yet.

Example: FTP  $\leftrightarrow$  Gopher  $\leftrightarrow$  WWW

## End-to-End Arguments

design infrastructure bottom-up

- Higher-level layers ... are free to ... organize lower-level network resources to achieve application-specific design goals efficiently. (application autonomy)
- Lower-level layers ... should provide only resources of broad utility across applications, while providing ... effective sharing of resources and resolution of resource conflicts. (network transparency)

Although the authors emphasize “not one, but two complementary goals,” the two are equivalent to the application. The first protects applications from overconstraint by the infrastructure. The second protects applications from bad behavior by other applications. It is right to consider them separately, because they are addressed differently, and a given feature may improve one while harming the other.

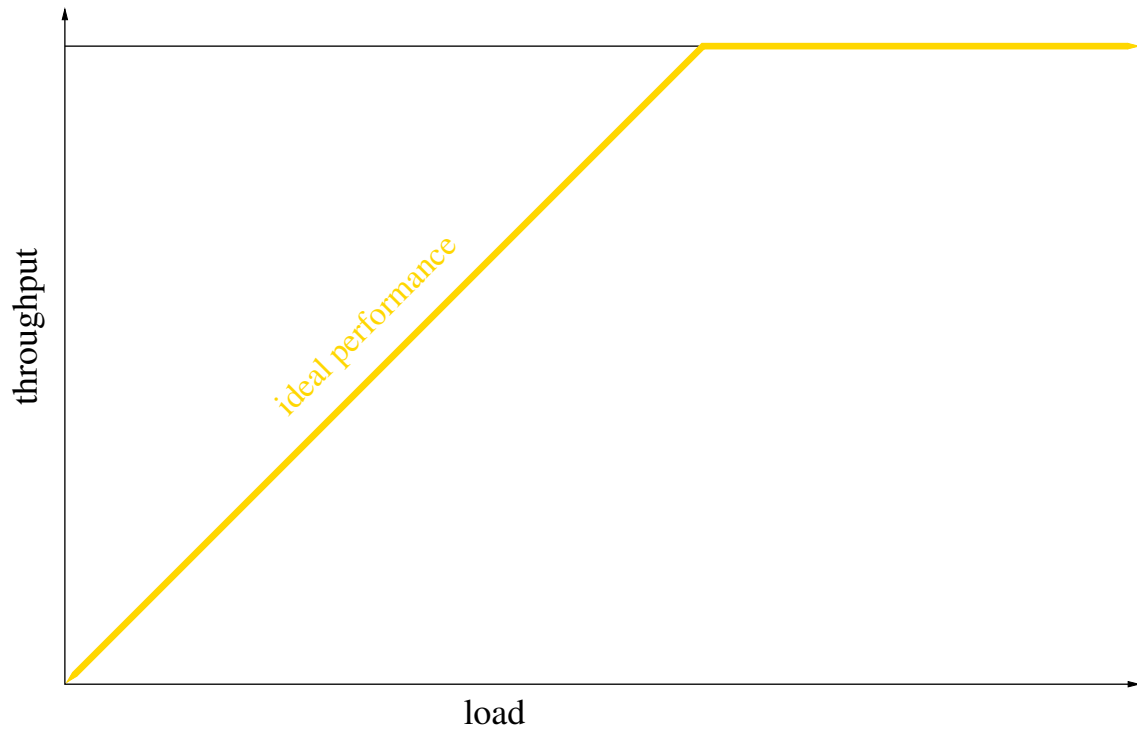
## Congestion control

**Choice:** End-to-end congestion control in TCP

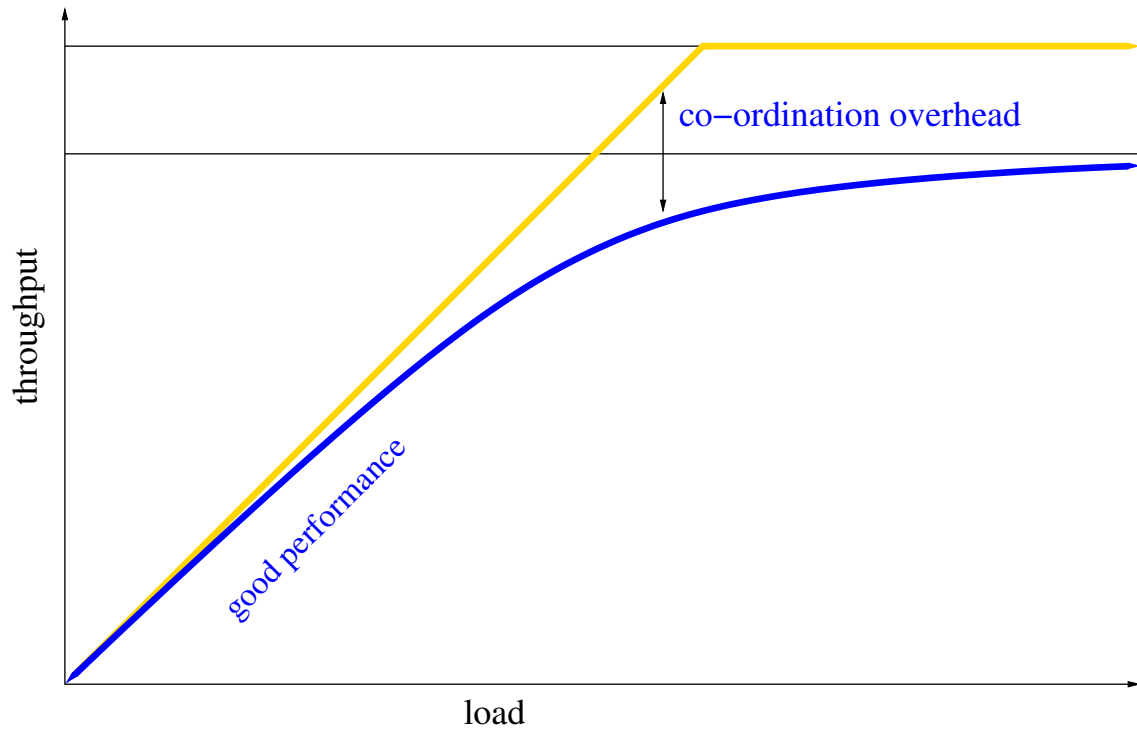
### History:

1. 1980–81: RFC 761, 793, RCP
2. c. 1987: Internet congestion collapse
3. 1988: Van Jacobson SIGCOMM, congestion control in TCP
4. 1997–99: RFC 2001, 2581, TCP congestion control

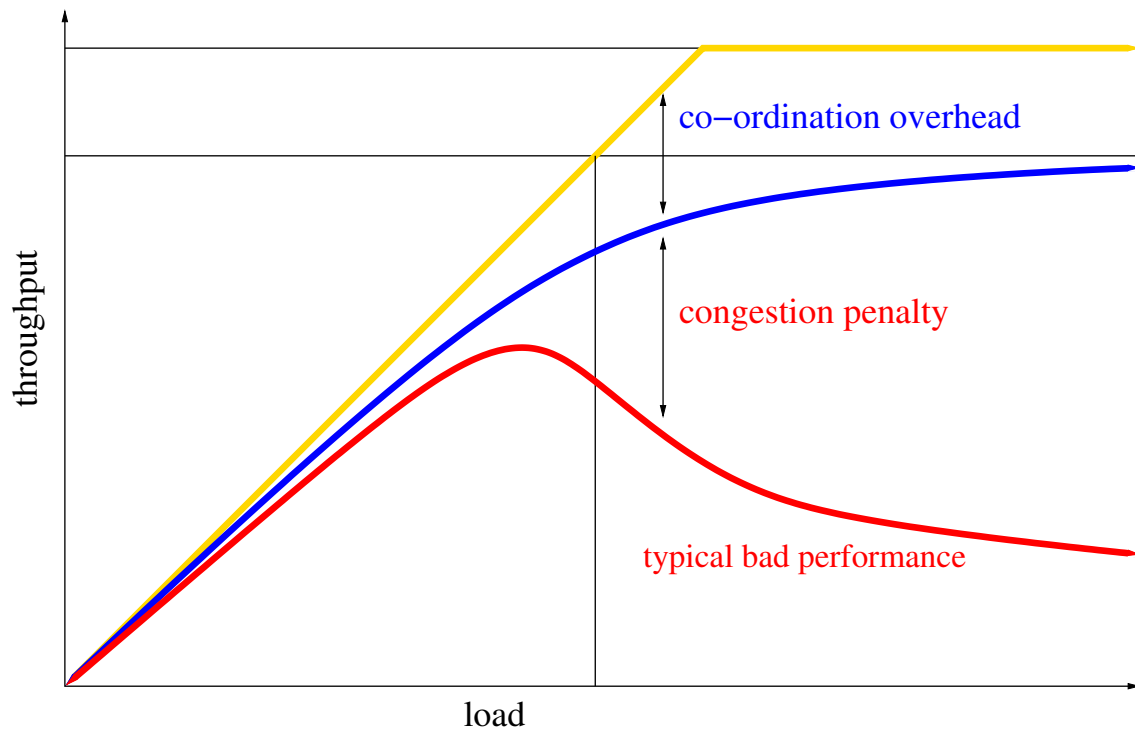
## Throughput vs. load



## Throughput vs. load



## Throughput vs. load

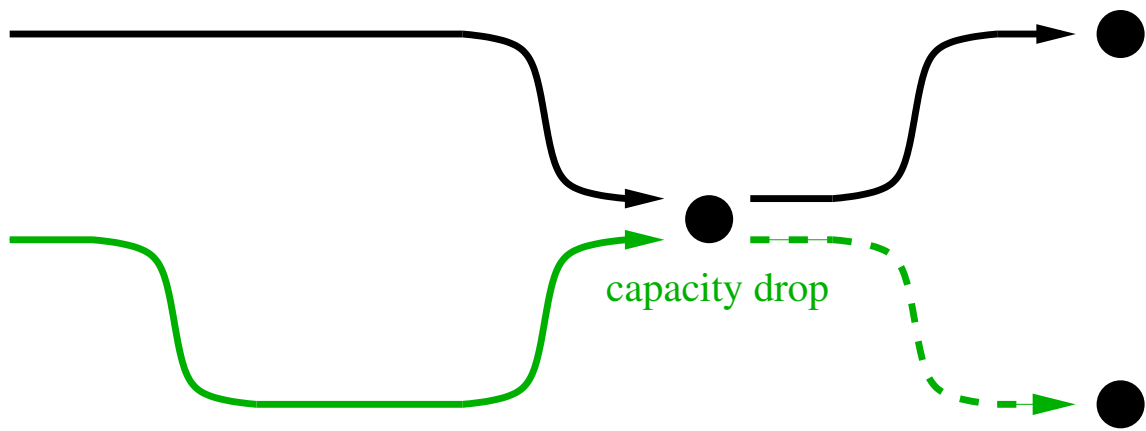


## Reasons for bad performance

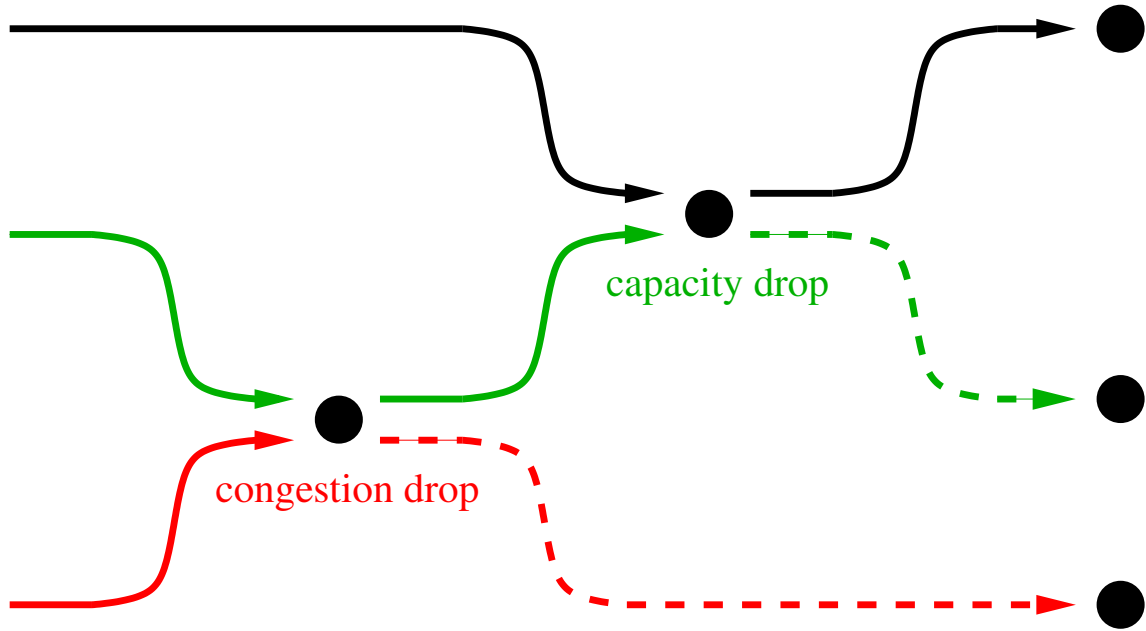
- One hop: collisions waste time on bus
- Multihop:
  - capacity mismatch
  - packet kills another packet, then dies

Peterson and Davie call the loss of throughput at high load “congestion collapse.”

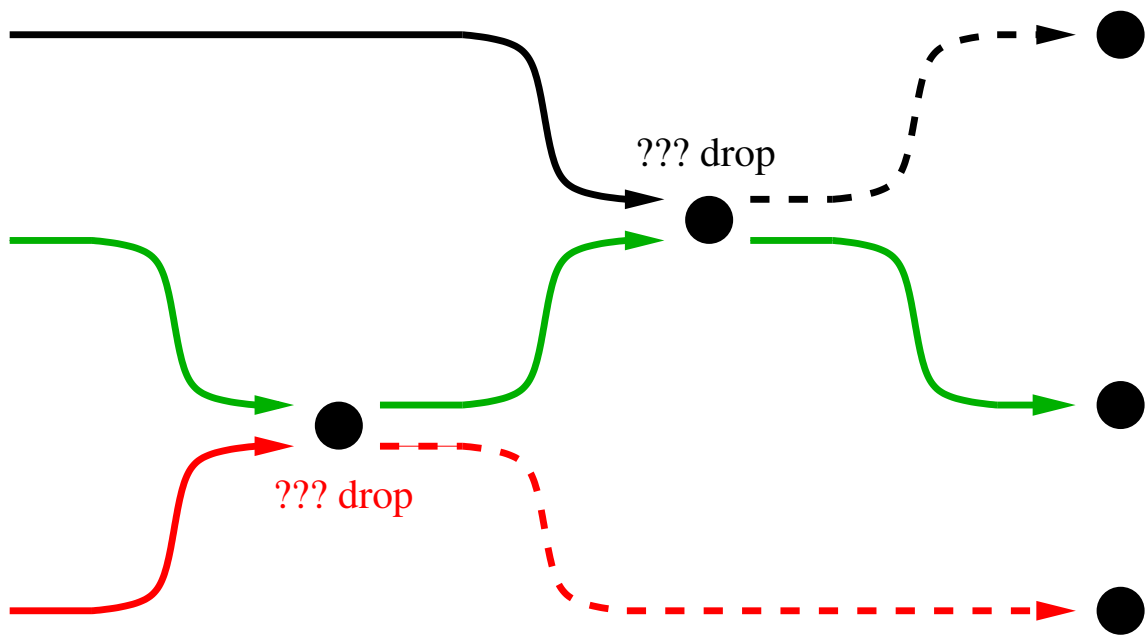
## Capacity drop vs. congestion drop



## Capacity drop vs. congestion drop



Which kind of drop?



I got the idea for classifying packet drops as *capacity* drops or *congestion* drops from the literature on memory caches. Cache misses are classified as *compulsory*, *capacity*, or *conflict*. The point of such classification is to notice how much performance improvement is conceivable by fixing a particular part of the system.

## TCP sole place for congestion control?

### Pro:

- dropping at source is optimal

### Con:

- bad modularization
- **authority-incentive mismatch**

TCP is overloaded with too much functionality—insufficient modular organization. But that is purely a problem in software structure. It is solved by breaking TCP into a set of protocols, each of which solves a different problem, but which may all be used together. Peterson & Davie apply the same reasoning when they break RPC into BLAST, CHAN, and SELECT in 5.3.

The authority-incentive mismatch is a problem in basic network architecture, and cannot be solved by more careful design of individual pieces of software, it requires an expansion of the functionality of a large fraction of routers.

## Case for router congestion control

**Principle:** align authority with incentive

- consequence of flooding is global
- congestion *rewards* attack by flooding

So network routers should control congestion.

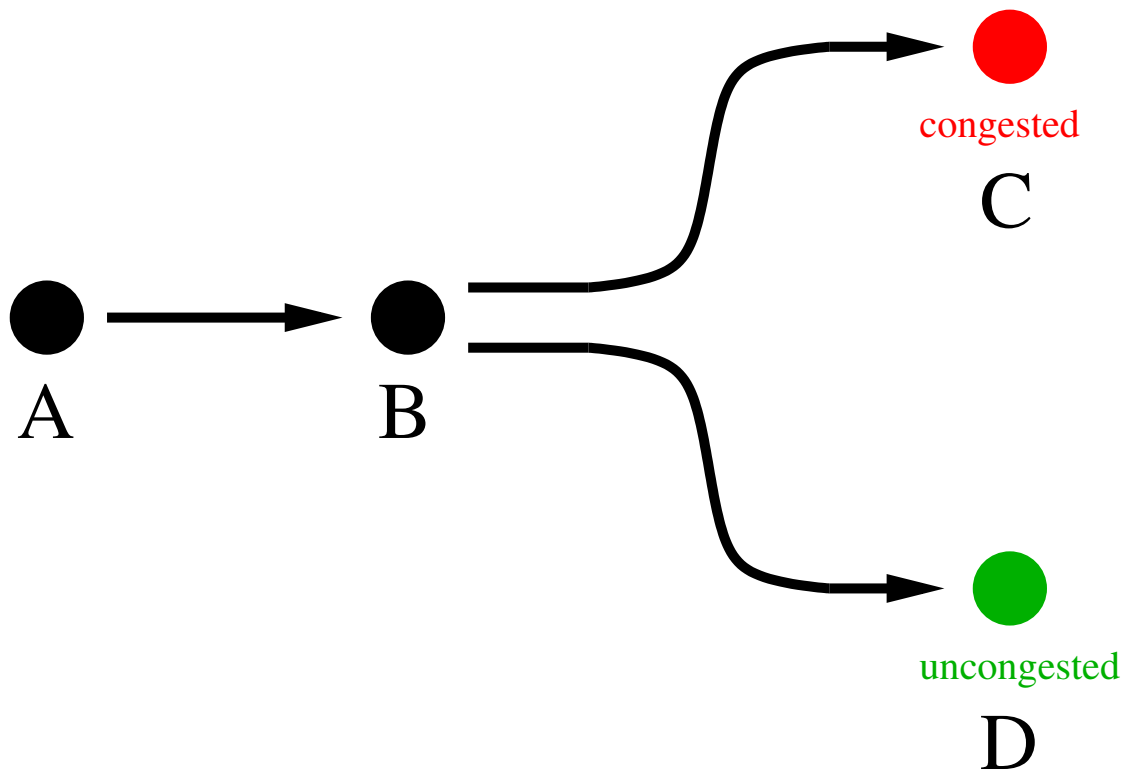
A number of projects have studied ways to apply incentives to packet sources by delivering more packets from sources that show congestion avoidance. But these incentives do not affect DoS attackers. And they may be recognized too slowly by ignorant sources.

## Push back congested flows

**A**daptive **H**op-**B**y-**H**op **A**ggregate  
congestion control,

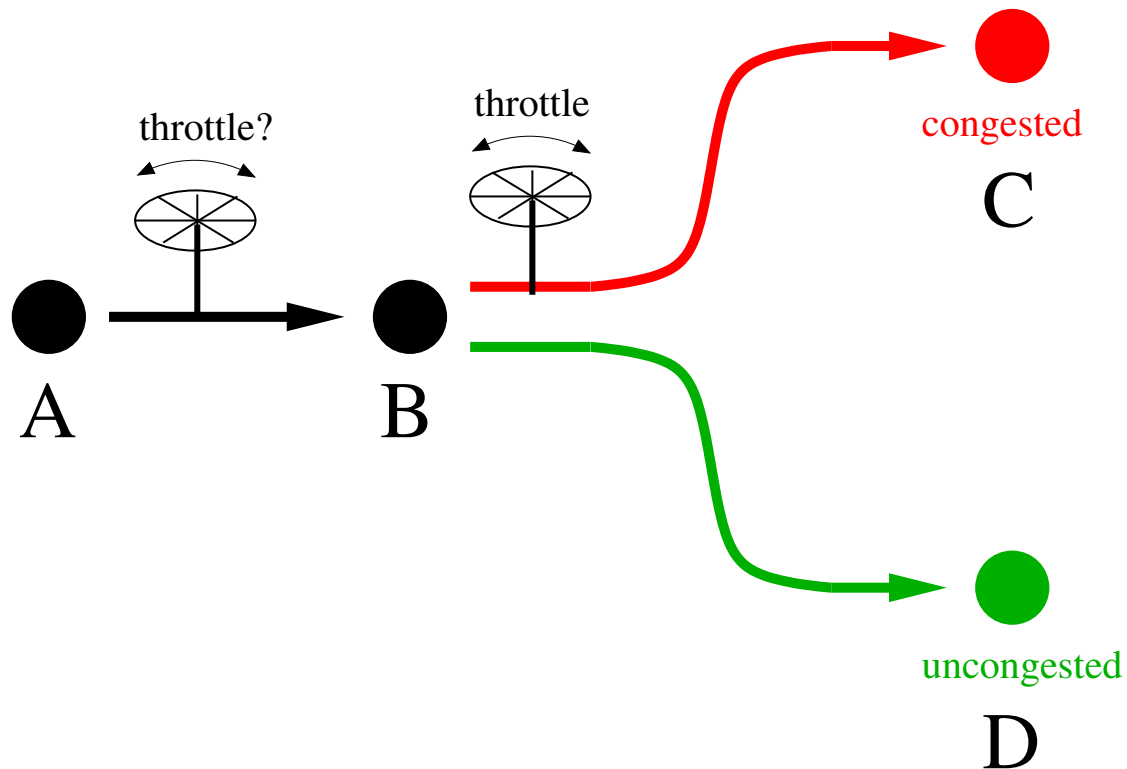
Michael Greenwald, *U. Pennsylvania*

## AHBHA congestion control



**Problem:** Congestion drops at B, A, ...

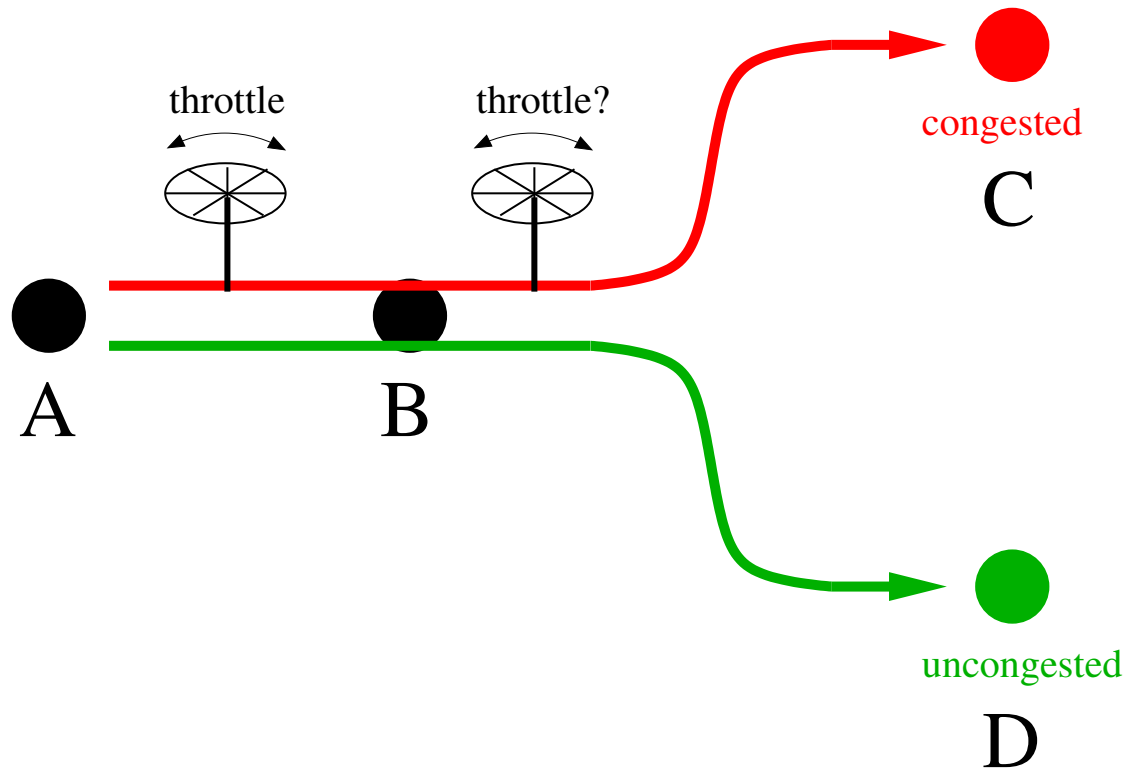
## AHBHA congestion control



**Problem:** B cannot be labelled (un)congested

- Throttling **BC** is insufficient
- Throttling **AB** underutilizes **BD**

## AHBHA congestion control



**Solution:** Split route table entries at A

- **AB** throttle avoids congestion drop at B
- **BC** throttle optional

Routes from A		
CIDR prefix	Next hop	throttle?
129.4.112/20	B	no
192.8.48/20	B	no
193.146.128/18	B	no

split

Routes from B		
CIDR prefix	Next hop	throttle?
129.4/16	C	yes
192.8.54/24	C	yes
192.8.56/24	D	no
193.146.128/18	D	no

Routes from A		
CIDR prefix	Next hop	throttle?
129.4.112/20	B	yes
192.8.54/24	B	yes
192.8.56/24	B	no
193.146.128/18	B	no

split  
split

Routes from B		
CIDR prefix	Next hop	throttle?
129.4/16	C	yes
192.8.54/24	C	yes
192.8.56/24	D	no
193.146.128/18	D	no

## AHBHA adaptation

- Iterate table splits backwards past congestion drops
- Collapse table entries when congestion eases

- What determines the queue length at a router?
  - First, analyze steady load
  - Then, consider load variation
- Why might LIFO and dropping the oldest packet in queue be good policies?
- With FIFO and dropping new arrival, show how a longer queue may *increase* congestion.

## Quality of service, guarantees, reservations

- Current IP service: “best effort.”
- “**Quality of Service**”: distinguish different types of service.
- “Guarantee” is commercial, not technical.
- Technical offering is reservation, not guarantee.
- Reservations reshape failure distribution.

## QoS is really reservation

**Choice:** Add reservations  
to Internet service?

The impact of reservations is to reshape  
the statistical distribution of success/failure.

Lee Breslau, Scott Shenker.

“Best-effort versus reservations:  
a simple comparative analysis.”

*ACM SIGCOMM '98*

## Example of failure w/wo reservation

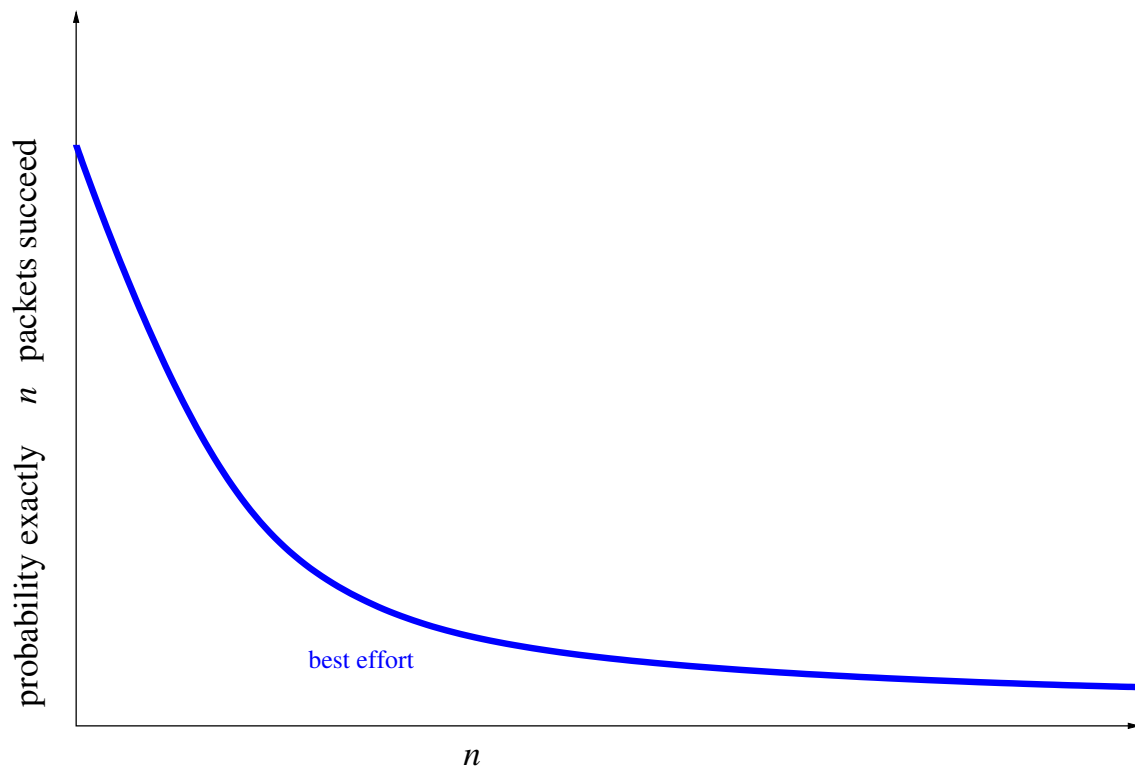
Load network with applications that send a stream of packets, quit at first drop.

Probability exactly  $n$  packets succeed:

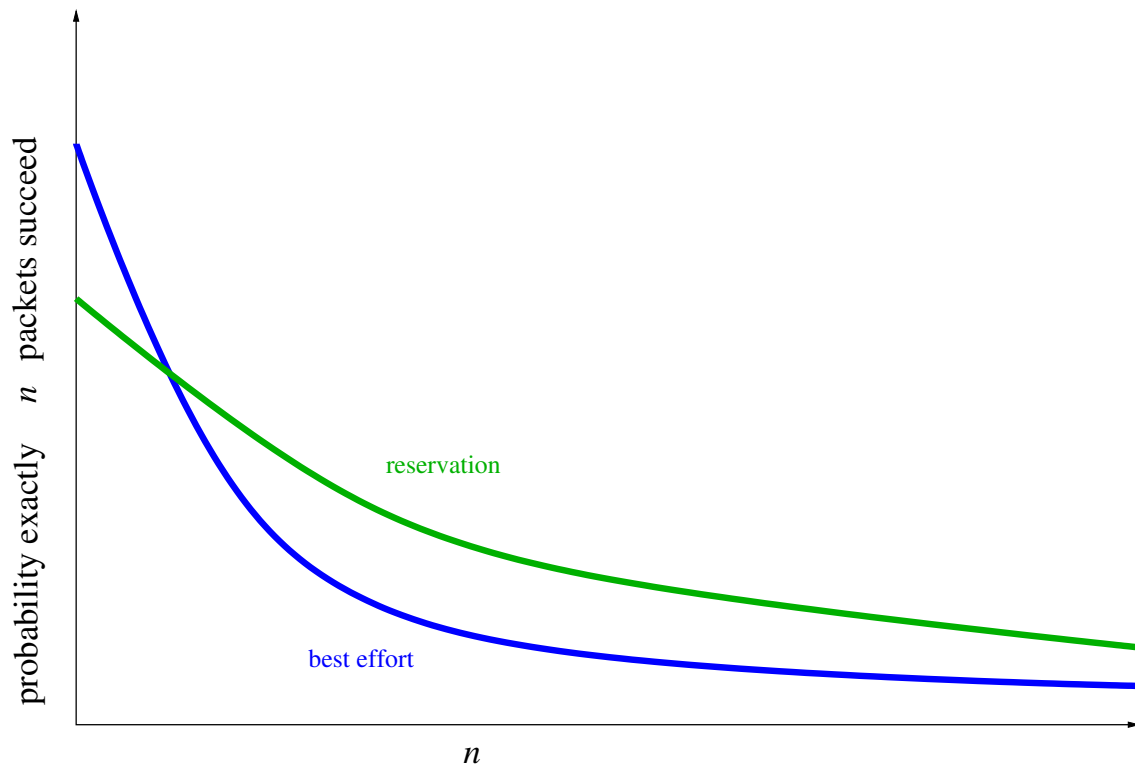
$$\begin{aligned}\Pr(n) &= \Pr(\textit{fail}) \cdot (1 - \Pr(\textit{fail}))^n \\ &= \Pr(\textit{fail}) \cdot e^{\ln(1 - \Pr(\textit{fail}))n}\end{aligned}$$

Reservation reduces  $\Pr(\textit{fail})$ , flattens exponential density.

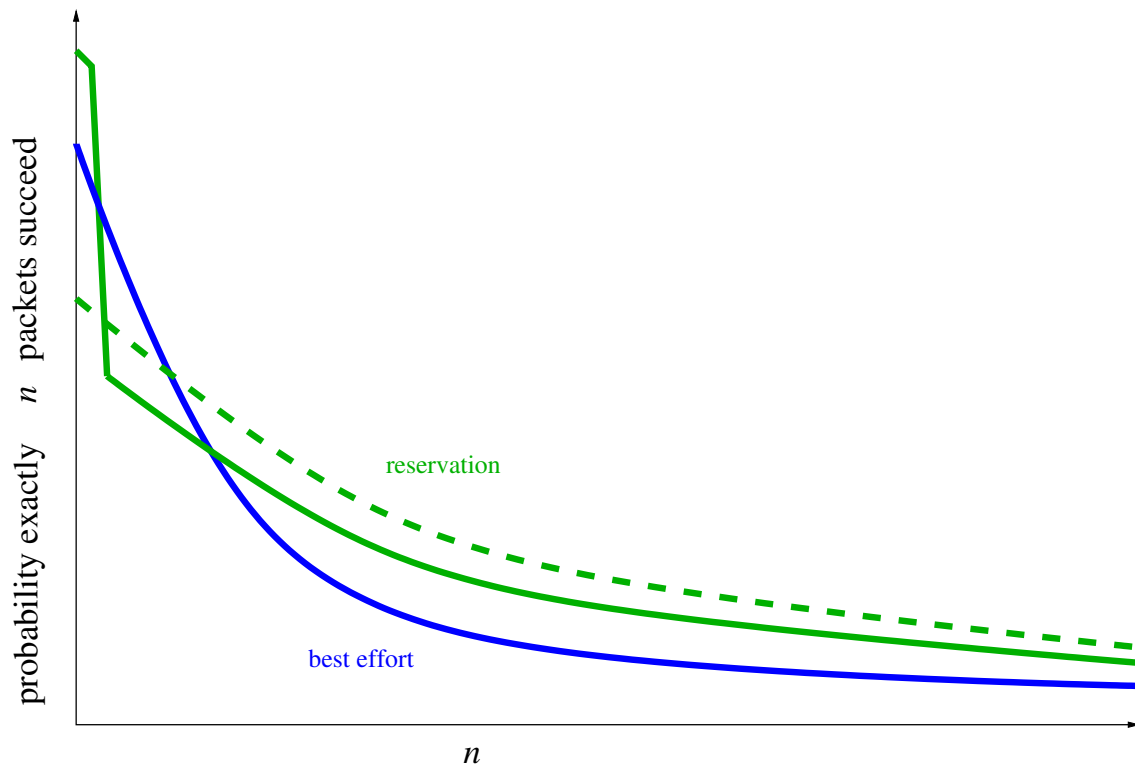
## Probability of delivering prefix best effort



## Probability of delivering prefix given reservation



## Probability of delivering prefix requesting reservation



## Requesting reservation changes failure granularity

- Flatter exponential has larger mean—  
exceeds network capacity
- Spike at 0 packets reduces mean to  
 $\leq$  best effort
- Packet failure shifted to  
reservation failure

Real reservations have finite duration.  
So they follow flatter curve to expiration,  
then steeper curve.

These distributions are examples, not the only possibility.

Other arrangements may not show all failures in the density curve, but failure is there.

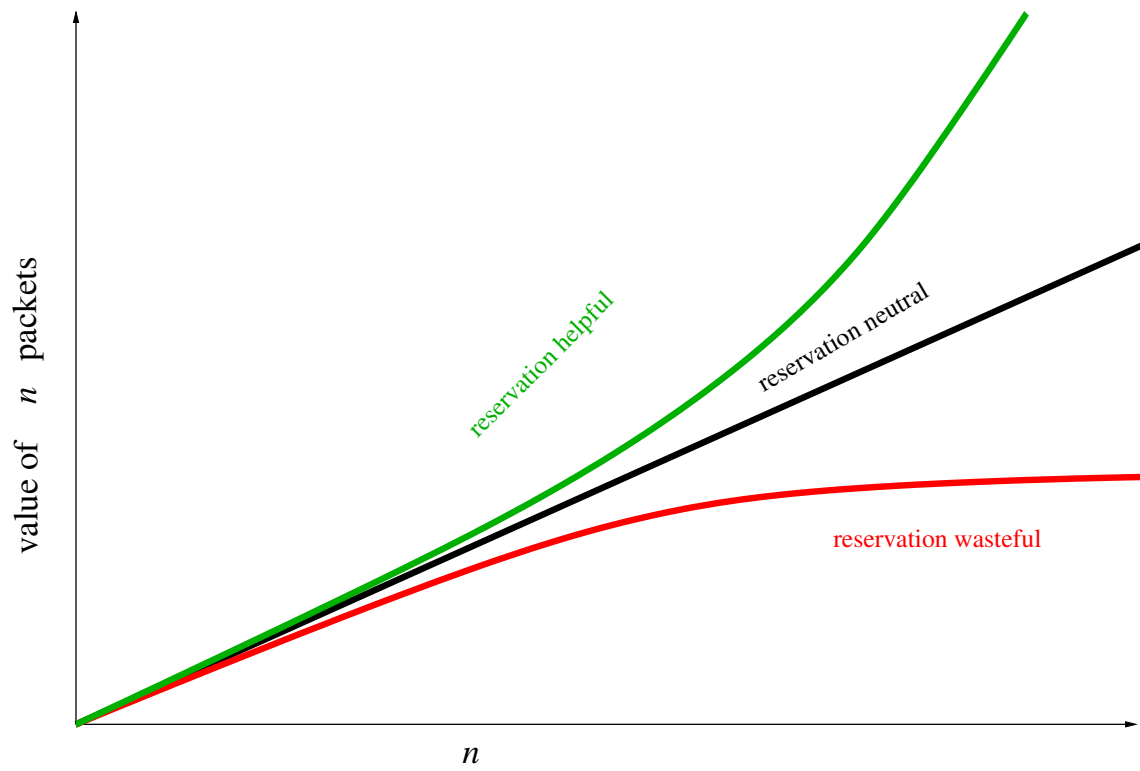
- Reduce success of best effort (steeper density curve)—more reservation requests, higher spike at 0.
- Charge money for reservation—failure by running out of money.

## Value of reservation to user

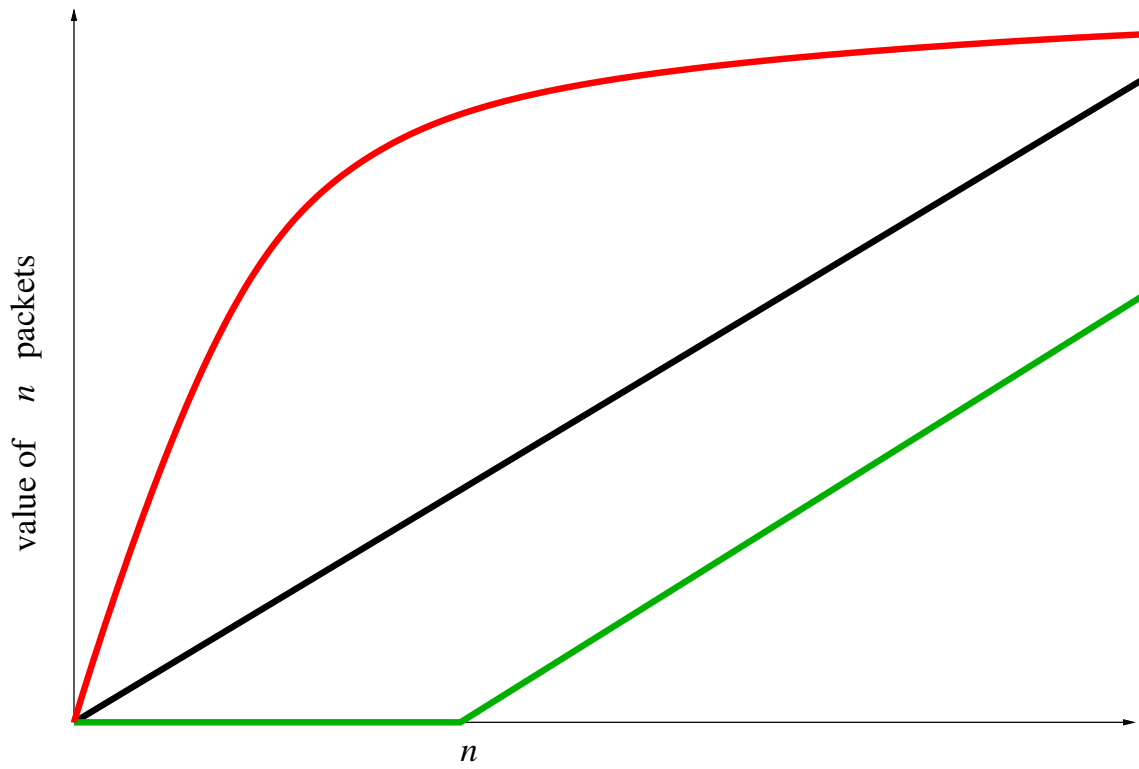
$\pi(n)$  is value of  $n$  packets to user

- Overall value =  $\sum \pi(n) \cdot \Pr(n)$
- $\pi''(n) > 0 \Rightarrow \pi(a + b) > \pi(a) + \pi(b)$ 
  - reservation helpful
- $\pi''(n) = 0 \Rightarrow \pi(a + b) = \pi(a) + \pi(b)$ 
  - reservation neutral
- $\pi''(n) < 0 \Rightarrow \pi(a + b) < \pi(a) + \pi(b)$ 
  - reservation wasteful

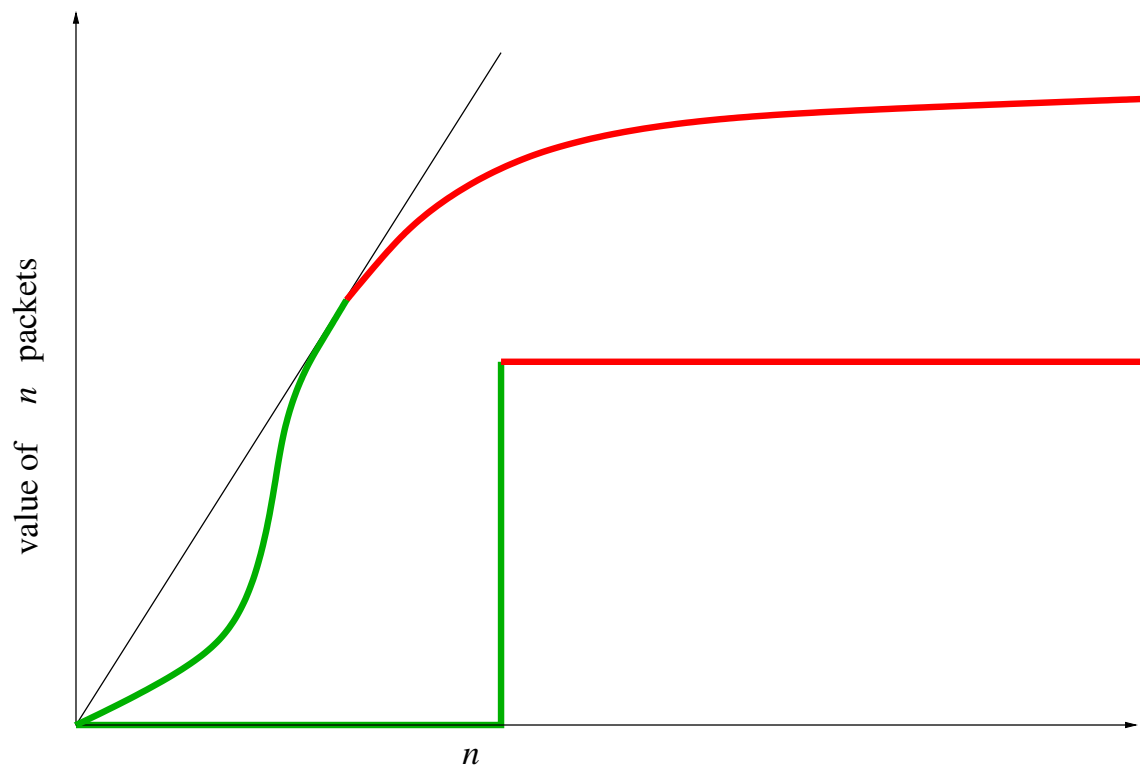
Value function: {super sub}linear



# Value functions, (un)helpful reservation



## Value functions limited helpful reservation



Maximize  $\pi(n)/n$ ?

For a reservation, suppose that we pay cost  $R$  to set up the reservation, plus  $S$  per packet. How does this affect the optimal size of the reservation?

Suppose that we pay cost  $P < S$  per packet delivered under best effort service. How do we choose whether or not to reserve?

## Share a fixed capacity

Breslau & Shenker consider partitioning the capacity of the network among flows.

Their  $\pi$  applies to bandwidth,  
not # of packets.

Considerations about shape of  $\pi$  are  
essentially the same.

## What issue will determine reservations?

- maximize global communication utility
  - allocate investment in capacity vs. reservation system
- throughput times delay = packets in transit
  - limits granularity of end-to-end decisions

reservations  $\Rightarrow$  charges?

## Why study DNS?

- Application layer, many reusable techniques
- Infrastructural in scope
- You may administer a DNS zone
- Serious social/political issues
- Very controversial, likely to change

## Domain Name Service

DNS is mainly used to map *names* to IP addresses.

Querier may use UDP or TCP.

Names of what?

For what purpose?

We saw that addresses don't necessarily refer to hosts, or network interfaces, but to anything that may be virtualized as such. The same sort of thing holds for names. A priori assertions about the meanings of these objects are only motivating, not technically constraining. The real meaning is determined by use. The key technical issues are

- What operations may be performed on names?
- Who may perform these operations?
- On what time scales do these operations work?

## DNS history

### History:

1. 1980–82 global table of flat host names  
HOSTS.TXT.
2. c. 1983 too many hosts.  
HOSTS.TXT too big to distribute/store,  
too complicated to update
3. 1983–87 RFC 882, 883, 1034, 1035,  
DNS

**Quibble:** Names/handles refer to *agents*,  
not to *hosts*.

E.g., `www.cs.uchicago.edu`.

**Principle:** *Everything* virtualizes.  
*Especially* in cyberspace.

## Hierarchical name space

**Domain name:** sequence of **labels**

total length  $\leq$  255 bytes

presented right-to-left, separated by “.”

Ex: label3.label2.label1.label0

**Label:** string of  $\leq$  63 bytes

“preferred” letter-digit-hyphen, case-insensitive

Ex: joy-37a

**Root label:** null string

Ex: joy-37a.cs.ustc.edu.cn.

6 labels, last is null

final “.” distinguishes absolute vs. relative

Like UNIX file path

- backwards
- “.” instead of “/”

Perhaps each label should have a type prefix—  
then incorporation of new alphabets might  
be easier.

Most software has forgotten the final dot,  
leading to ambiguity.

## Original intentions

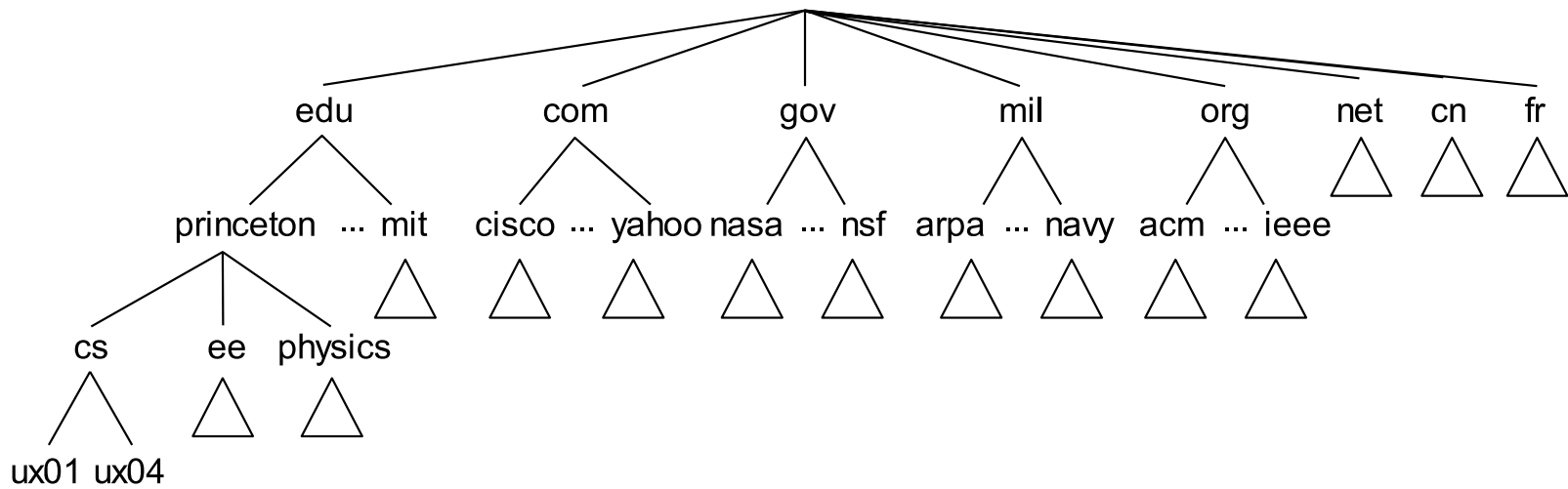
- Replace unsupportable HOSTS.TXT
- Map names to IP addresses
- Map names to anything useful
  - multiple values allowed
    - compatible with pre-IP mailbox practice

“consistent name space ... for referring to resources.”

—RFC1034

## Domain hierarchy

### Tree of domains



P&D Figure 9.2 (uk  $\leftrightarrow$  cn)

May attach a host *anywhere*: leaf or internal node.

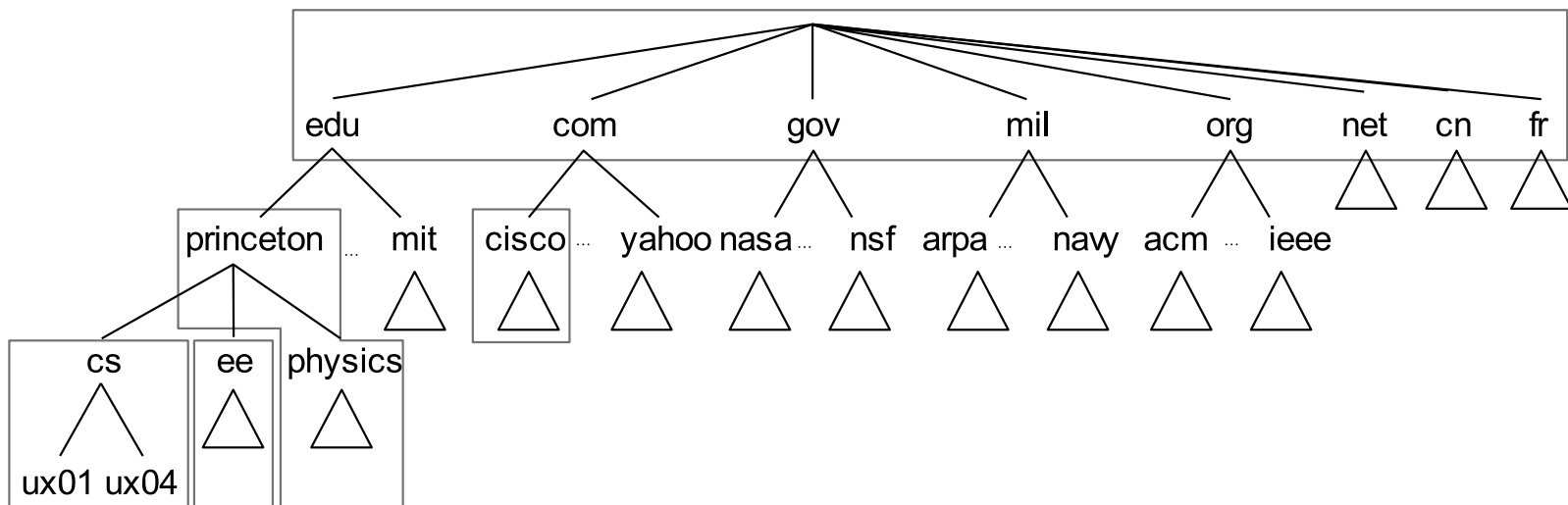
Relative paths need *final* “.” to distinguish *prefix* from full name.

Internal host names need *initial* “.” to distinguish *suffix* from full name.

## Zones of authority

Zone: contiguous subgraph of domain hierarchy controlled by one name server

Usually a single domain



P&D Figure 9.3

Errors: Root, edu, com, ... are separate zones.

## Name server authority hierarchy

Collapses domain hierarchy to zones



P&D Figure 9.4

Error: omitted edu, com servers

## Three independent structures

1. IP address tree
2. Domain name tree
3. Name-server *location*

There is a loose correspondence between domain name tree and IP address tree, because both reflect authority hierarchy.

Domains for reverse lookup are crafted to match IP address tree.

## Authority for query resolution

- Query `galton.math.uchicago.edu`.
- Root NS  $\hookrightarrow$  `edu.` NS.
- `edu.` NS  $\hookrightarrow$  `uchicago.edu.` NS.
- `uchicago.edu` NS  $\hookrightarrow$  `galton.math.uchicago.edu`.

Notice 2-level jump in last step, because `uchicago.edu.` NS contains `math.uchicago.edu.` domain.

**Problem:** bottlenecks, especially at root.

**Solution:** distributed resolution with cacheing

- Query any convenient server
  
- Server may
  - resolve query
  - refer you to another server  
(iterative)
  - query another server for you  
(recursive)
  - cache records for future queries
  - return description of authority
  - answer related query

- name servers
  - authoritative
    - \* primary (master)
    - \* secondary (slave)
      - listed (in parent zone)
      - stealth (listed only in resolvers)
  - nonauthoritative (usually for local cache)
- resolvers (called by ap on local host)

- resolver(g.f.e.d.c.b.a.)
- resolver → NANS (g.f.e.d.c.b.a.)
- NANS has ANS for b.a. in cache
- NANS → b.a. (recursive)
- b.a. has authority for c.b.a. → NANS
- NANS → c.b.a. (iterative)
- c.b.a. has authority for d.c.b.a.,  
f.e.d.c.b.a. in cache → NANS
- NANS → f.e.d.c.b.a. (iterative)
- f.e.d.c.b.a. has authority for g.f.e.d.c.b.a.  
→ NANS (resolves to IP address)

- NANS → resolver
- resolver returns IP address

**Warning:** most queries are resolved in 1-3 steps, but there is one more complication.

## Principles of distributed data

- Meaning independent of source, target, context (queries and responses) whenever possible
- Authenticity independent of source
- Distinguish data, metadata
- Negation of data may be metadata
- Timeouts are crucial

Metadata never completely independent of context

Zone file  
boundaries of the zone

Zone database contains **R**esource **R**ecords

**S**tart **O**f **A**uthority record gives root node  
of zone

**N**ame **S**erver records give fringe of zone

## Zone file example zone boundaries

```
.      IN      SOA      (
```

```
)
```

```
MIL.      NS      <point to name server>
```

```
EDU.      NS      <point to name server>
```

From RFC1034

Zone file example  
pointers to name servers

```
.      IN      SOA      (
```

```
)
```

```
MIL.      NS      SRI.NIC.ARPA.
```

```
EDU.      NS      SRI.NIC.ARPA.
```

From RFC1034 (SRI-NIC → SRI.NIC)

Zone file example  
extra name servers

```
.      IN      SOA   SRI.NIC.ARPA.          (  
  
      )  
      NS     A.ISI.EDU.  
      NS     C.ISI.EDU.  
      NS     SRI.NIC.ARPA.  
  
MIL.   NS     SRI.NIC.ARPA.  
      NS     A.ISI.EDU.  
  
EDU.   NS     SRI.NIC.ARPA.  
      NS     C.ISI.EDU.
```

From RFC1034

Zone file example  
timeouts, etc.

```
.      IN      SOA    SRI.NIC.ARPA.  HOSTMASTER.SRI.NIC.ARPA. (
      870611      ;serial
      1800      ;refresh every 30 min
      300      ;retry every 5 min
      604800    ;expire after a week
      86400)    ;minimum of a day
      NS      A.ISI.EDU.
      NS      C.ISI.EDU.
      NS      SRI.NIC.ARPA.

MIL.  86400  NS      SRI.NIC.ARPA.
      86400  NS      A.ISI.EDU.

EDU.  86400  NS      SRI.NIC.ARPA.
      86400  NS      C.ISI.EDU.
```

From RFC1034

## Where are the IP numbers?

**A**ddress records map domains to IP numbers

```
NIC.ARPA.      IN      SOA  SRI.NIC.ARPA. (
...
)
SRI             A      26.0.0.73
```

## Infinite regress finding NS

**Problem:** We will never find a name server located in or below its own zone

**Solution:** Copy A records as “glue,” but they are not authoritative

Zone files should represent *authority* structure, but they seem to really describe *service* structure.

**Note:** Query resolution may involve recursive/iterative queries for the name servers.

Canonical **NAME** records map aliases to best names

**PointeR** records are used to map IP numbers to domains

```
USC-ISIC.ARPA.          CNAME  C.ISI.EDU.
SRI-NIC.ARPA.           A       26.0.0.73
73.0.0.26.IN-ADDR.ARPA. PTR     SRI-NIC.ARPA.
```

**Mailbox** records give a list of hosts to try mail delivery.

## RFC3596 vs. RFC2874

AAAA records are just A records with 128-bit IP numbers (RFC3596)

Proposal (RFC2874): support rapid IP renumbering

```
A.EX. A6 40 0000:0000:0011:CCEF:110D:A7B9:1111:002F B.EX.  
B.EX. A6 28 0000:0001:CA00:0000:0000:0000:0000:0000 C.EX.  
C.EX. A6  0 6333:AA10:0000:0000:0000:0000:0000:0000
```

A.EX.

resolves to

6333:AA11:CA11:CCEF:110D:A7B9:1111:002F

## Suffix delegation, RFC2874

**Delegate NAME:** map a DNS node to a subtree

A.EX.            DNAME    B.C.EX.

Z.A.EX. resolves the same as Z.B.C.EX.

CNAME applies to full domain, DNAME to prefix

(wish we had the initial ".")

## DNAME intended use

DNAME is intended to allow rapid renumbering in the IP-to-name domains.

Could be useful for domain name transfers.

## Original purposes of DNS

- Eliminate the requirement for people to remember host numbers (addresses).
- Provide stability when addresses changed.
- ... multiple addresses associated with a given host.

—RFC3467

## Conflicting DNS hierarchies

- authority
- service
- mnemonic → semantic  
→ cultural/political/legal/...

## DNS: names or handles?

- Shortcomings of IP numbers
  - hard to remember/guess
  - dependent on network topology/location

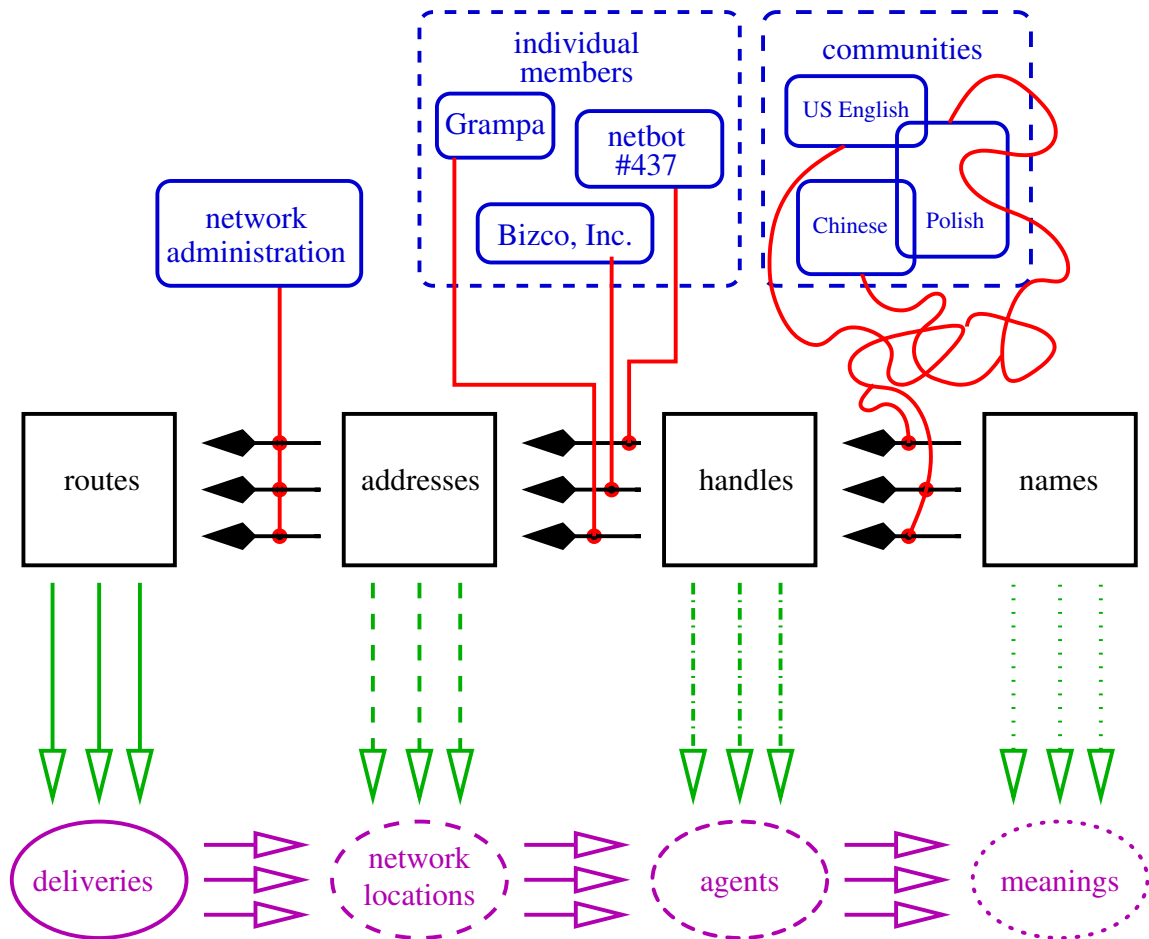
**Choice:** Introduce meaningful names,  
not meaningless handles?

The choice to have domain names refer to hosts is illusory.

Domain names resolve to IP addresses.

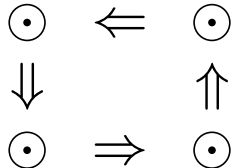
Reference is mental, not technical.

## 4 levels of identifiers



## Reference between identifiers

- Identifiers *resolve* from right to left.
- Identifiers *refer* downward.

- Square paths should close 

- All objects are virtual.
  - Meaning of objects derived from relations.
  - Correctness only in relations, not objects.
  - **Real design decision is who controls resolution.**

## Pseudohistory of network identifiers

- Routes in UUCP: `host1!host2!...!hostn`
  - Sally sends to Grampa's candy store:  
`gargoyle!anubis!monkey!candy`
  - No good for Paul—he must use  
`juniper!elm!granite!arthur!candy`
- IP numbers: `128.72.98.4`
  - Same address reaches Grampa for both Sally and Paul.
  - No good when Grampa moves, network topology changes.

Don't be confused by the fact that IP addresses are more robust than UUCP routes, but less mnemonic. These are two independent dimensions. We are mostly concerned here with robustness.

## Pseudohistory, continued

- Domain names: `candy.com`
  - Same name works for everyone.
  - Grampa can update resolver of `candy.com` when he moves.
  - Semantic value of `candy.com` leads to conflict.
    - \* Bigger candy company wants `candy.com`
    - \* C and Y company wants `candy.com`
  - Systemic conflict: Chinese alphabet,  
...

## Possible future

- Handles: `h1g5k0061A38F9A3540B9`
  - Random looking, but not much worse than phone, credit card numbers.
  - Can be self assigned with no central authority.
  - Names resolve to handles which resolve to addresses.
  - We still fight over names.
  - Other ways to find handles besides DNS: Yahoo, Google, ...

## Handles ubiquitous in computing systems

- Compiler:  
identifier → sym table index → address
- EMail:  
nothing → message id → message
- CORBA:  
??? → object reference → servant
- C:  
variable → file descriptor → file

Names only:

- **Principle:** When two services are equally easy to implement, choose the more valuable one.
  - Names more valuable than handles.

Handles and names:

- **Principle:** Put different services in different modules.
  - Name/handle resolution depend on different authorities.
- **Counterprinciple:** Valuable scarce resources are costly to defend.

The first principle—choose more valuable service—usually wins.

In Internet design, we are usually creating very plentiful services.

Scarcity of names with valuable meanings changes economics radically.

Story: gold-plated rowboat in sea of pirates.

## Authentication by provenance

Current DNS *authentication by provenance*:  
trust answer because of source

How do we know the source?

- IP address in query
- source IP in response
- query id in response
- port # in response

All can be forged.

Id and port can be eavesdropped, guessed.

How do we know IP of trusted NS?

Name server corruption: cache poisoning  
security-performance tradeoff

## DNSSEC signed DNS

**DNSSEC**urity extensions:  
authentication by digital signature  
chain of trust

## Security through cryptography

- Privacy/secretcy
- Authentication/signature

## Secret-key cryptography (e.g. DES)

- Create one random-looking key  $K$
- Key defines two functions,  $\underline{K}$ ,  $\overline{K}$ , with  $\overline{K}(\underline{K}(x)) = x$  (inverses)
- Share secret  $K$  among 2 or more communicants
- Important theory: given  $x$ ,  $\underline{K}(x)$ , intractable to discover  $K$  or compute  $\overline{K}$

## Public-key cryptography (e.g. RSA)

- Create two random-looking keys  $K_o$  (open) and  $K_s$  (secret)
- Each key defines a function  $\bar{K}$ , with  $\bar{K}_o(\bar{K}_s(x)) = \bar{K}_s(\bar{K}_o(x)) = x$  (inverses)
- Publish  $K_o$ , keep  $K_s$  secret
- Important theory: given  $K_o$ , intractable to discover  $K_s$  or compute  $\bar{K}_s$

## Using public-key cryptography

**Secrecy:** Send message  $x$  encoded as  $\bar{K}_o(x)$ .

Recipient decodes by  $\bar{K}_s(\bar{K}_o(x)) = x$ .

$x$  can be symmetric key.

**Signature:** Send message  $x$  signed as  $\bar{K}_s(x)$ .

Recipient verifies by  $\bar{K}_o(\bar{K}_s(x)) = x$ .

## Secure hashing (e.g. MD5, SHA)

- No key involved
- $H(x)$  much smaller than  $x$
- $H$  easy to compute
- Important theory: given  $x$ ,  
intractable to find  $y$  with  $H(y) = H(x)$

## Using secure hash

**Comparison:** Test  $H(x) = H(y)$   
to verify  $x = y$

**Signature:** Send  $\bar{K}_s(H(x))$  instead of  $\bar{K}_s(x)$ .  
Recipient verifies by  
 $\bar{K}_o(\bar{K}_s(H(x))) = H(x)$ .

## Performance of cryptography

RSA {en/de}crypt:	100 Kbps
DES {en/de}crypt:	100 Mbps
MD5 hash:	600 Mbps
SHA hash:	260 Mbps
RSA/MD5 signature: (1 Kb certificates)	100 Kbps
RSA/MD5 verify:	2000 Kbps
routing:	500-3000 packets/sec
DNS	100-5000 queries/sec

DNSSEC  
presigned RR sets  
RFC2065

**Problem:** Signature is too slow to use on each transaction.

**Solution:** Presign RR sets.  
Also reduces exposure of secret key.

## DNSSEC new RRs

**KEY** presents a public key

**Delegation Signer** presents a trusted key

**SIGN**ature presents a certificate for an RR set

**NeXT** allows presigned negative responses

DNSSEC  
KEY RR

a.b.ex. KEY ZONE|HOST DNSSEC 1 <key in base 64>

ZONE|HOST is mnemonic for a bitmap indicating which resolvers of a.b.ex. own this key, and other “flags”.

DNSSEC is mnemonic for a *protocol byte*, indicating which protocol the key is intended for.

1 is the number for the MD5/RSA signature algorithm.

Based on actual measurements, 10-30% of all delegations on the Internet have differing NS RRsets at parent and child.

—RFC3658

DNSSEC  
DS RR (RFC3658)

```
d.ex. KEY 256 3 1 (
    AQPwHb4UL1U9RHaU8qP+Ts5bV0U1s7fYbj2b3CCbzNdj
    4+/ECd18yKiyUQqKqQFWW5T3iVc8SJ0KnueJHt/Jb/wt
    ) ; key id = 28668
DS 28668 1 1 (
    49FD46E6C4B45C55D4AC69CBD3CD34AC1AFE51DE )
```

DS (when signed) indicates that the signer trusts the use of the specified key by d.ex. to sign its other keys.

28668 is a trivial hash (e.g. low-order bits) of the key, to distinguish from other keys owned by d.ex.

First 1 indicates that d.ex. uses this key with the MD5/RSA signature method.

Second 1 indicates that SHA-1 is used for the secure hash of the key.

49FD46... is the SHA-1 digest of the designated key.

Parent zone should sign and store DS record for each child.

## DNSSEC SIG RR

```
b.a.ex. SIG (  
    <RR types> <alg> <labels>  
    <TTL> <exp> <time> <keyid> a.ex.  
    <base 64 sig>  
    )
```

<RR types>: which types of RR are being signed.

<alg>: number identifying the signature method.

<labels>: length of the domain name that owns the RRs (needed because of shorthand notations).

<TTL> is an upper bound on TTL in the RRs.

<exp> is signature expiration.

<time> is time of the signature.

<keyid> is a trivial hash of the key in case the owner has multiple keys.

a.ex. is the owner of the key (not of the RRs).

Verifier must reconstruct exactly the text of the signed records in order to check the signature.

**Problem:** Unbounded number of queries for unassigned domains.

**Solution:** Presign intervals between assigned domains.

## DNSSEC NXT RR

`big.foo.tld. NXT medium.foo.tld. A MX SIG NXT`

Bitmap shows which RRs present for `big.foo.tld.`

What about last NXT in a zone?

Wrap around to SOA.

NXTs added automatically.

**Problem:** TTLs change too fast for presignature.

**Solution:** Sign upper bound on TTL.

## DNSSEC operations

- Who should sign what?
  - Authority signs, not possessor.
    - \* Zone signs its authoritative data.
    - \* Parent zone signs DS for child (chain of trust).
- Multiple signatures?
  - Possibly.
- Sign transactions?
  - Probably not.
- Expire signatures: small multiple of TTL.
- Expire keys: 1 month - 5 years.

## Infrastructure needed for PKC

Two technical system problems

- Manage private keys:  
use without revealing
- Distribute public keys:  
connect key to owner

Private key problem is difficult, but local.

Public key distribution is  
network infrastructure.

## Chain of trust in public keys

- Must distribute reliable  $\langle name, key \rangle$  pairs.
- Distribute  $\langle R's\ name, R's\ key \rangle$  out of band.
- R signs  $\langle A's\ name, A's\ key \rangle$ .
- A signs  $\langle B's\ name, B's\ key \rangle$ .
- Users collect  $\langle name, key \rangle$  pairs.

**Problem:** A chain is as strong as its weakest link.

(Person  $\leftrightarrow$  name is another link.)

## Web of trust (e.g. PGP)

- Distribute signed  $\langle name, key \rangle$  pairs promiscuously.
- Users evaluate total trustworthiness from multiple signatures.

CoT bundles  $\langle name, key \rangle$  distribution with evaluation.

WoT unbundles, allows arbitrary evaluation.

Unbundle further: distribute *keys* without *names*.

End-to-end principle applied to authentication.

Security is most robust when its requirements are satisfied by the OR of many conditions. CoT depends on the AND of many conditions. WoT provides ANDs and ORs, but still too many ANDs.

## Identity: relation and object

identity

3. The quality or condition of being the same as something else.

4. The distinct personality of an individual regarded as a persisting entity; individuality.

—The American Heritage  
Dictionary of the English Language

Oxford English Dictionary, and Merriam-Webster Dictionary had only the relational and qualitative meanings. I found identity as an object in the 3d dictionary I tried.

Careless speech in US CS community mostly uses identity as an object. E.g., “identity theft.”

## Identities from authority (top-down)

CoT, WoT associate keys with predetermined names/identities.

Authenticity determined by authority, communicated by PKC.

(authentication by *provenance*)

Outside of net, identities are tricky.

Before cryptographic signature, much authentication is by *unauthenticated provenance*. E.g., we trust the name server at a particular IP address, and we trust that we have really contacted that address. CoT and WoT authenticate individual links of provenance, and allow for multiple steps (analogous to art gallery tracing provenance of painting). But CoT/WoT do not provide total authentication from end to end.

## Identities from experience (bottom-up)

Instead of assuming identities,  
provide tools for constructing identities.

Identity: relation and object.

Derive identities from identity relation.

Outside of the net, the identity relation associates a series of transactions together. Our concept of identity as object derives from the accumulation of experience across those transactions.

CoT, WoT assume identities outside of net, try to import them to net. Instead, we may import the mechanisms that construct identities outside of net.

## Flat key distribution

**Idea:** Support *identity relation*,  
not *identities*.

A cryptographic “signature” identifies signer only as the owner of the signing key.

Signature supports identity relation.

That’s enough for the bottom-level infrastructure.

## Hash shortens key

**Problem:** Public keys are long  
(128 bytes or more)

**Solution:** Use hash of key

Free tradeoff: long hash for security,  
short hash for convenience.

## Public key servers

Provide tuples  $\langle key, hash(es), value \rangle$

Lookup by key or hash.

*value* may have any type.

- no responsibility for authenticity (*end-to-end* authentication)
- only facilitates discovery of keys

Notice analogy to best-effort packet delivery.

Key owner has permanent control of database entries:

- may switch key server
- may use multiple servers
- may re-establish *same* key when server dies

## Public-key protocol: datatypes

Use principles of distributed DB design.  
Network performance mostly involves lower level of implementation.

- *key* has form  $\langle \textit{sigmethod}, \textit{okey} \rangle$   
*sigmethod* gives the type of signature,  
*okey* the open key
- *hash* has form  $\langle \textit{hashmethod}, \textit{hashokey} \rangle$
- *value* type is free:  
typically address of owner
- *seqno* is an integer assigned by the owner  
larger number cancels smaller number
- *exp* is expiration date  
(crucial for distributed DB record)

## Public-key protocol: messages

Owner signs and announces records:

**establish**(*key, hash, time*) (do once)

**bind**(*hash, value, seqno, exp*)

**delegate**(*hash, rhash, seqno, exp*)

**assign**(*hash, rhash, time*)

(irrevocable)

**compromised**(*hash, time*) (irrevocable)

---

Server answers queries by *hash*, returns most recent signed record (by *seqno*).

**assign, compromised** beat **bind, delegate**

Have both **assign** and **compromised**,  
return both. Querier judges validity.

## Who signs records?

Important signature is signature of owner.

- E.g. **bind**(*hash*, *value*, *seqno*, *exp*) signed by *key* s.t.  $H(\textit{key}) = \textit{hash}$

Server signature adds nothing to positive query response.

Server may sign negative query responses.

Trusted 3d party may re-sign **assign**, **compromised** to certify timestamp.

## Purpose of messages

**bind** allows owner to (re)associate arbitrary *value* with *hash*.

Queries allow others to find *key* and *value* associated with a known hash.

Using *key*, querier authenticates response independent of server.

**delegate** allows owner to share one hash for mutiple distinguished roles.

Share with own or other's hash.

Split as resources/requirements dictate.

**assign** allows owner to upgrade cryptographic method, pass on or sell a role to another owner.

**compromised** allows owner to disavow a compromised key.

Notice that a given *key* cannot be re-assigned to a new agent, since the original owner cannot verifiably forget the private key. *assign* is crucial to allow full transfer. It is also crucial to allow upgrade as faster computers and clever attacks break the old cryptographic methods.

## How queriers use records

**bind**—use *value* until expiration, query again as desired.

**delegate**—requery *rhash* (after checking signature).

Server returns **delegate** record, not record for *rhash*, so querier may authenticate.

**assign**—requery *rhash*, and replace *hash* by *rhash* in local records.

Local replacement is crucial, since **assign** record will eventually disappear or be compromised.

Flat key distribution provides an infrastructure analogous to IP level of communication.

CoT, WoT, ... built on top of flat keys.

The idea in choosing to implement a layer of flat key distribution is *not* to provide exactly what users need/want. Rather, it's to provide the most that can be accomplished *without* central authority. That leaves us free to proceed with a variety of different strategies for further co-ordination. Think how we tried FTP and Gopher before HTTP.

That is, this is a bottom-up infrastructure design, not a top-down application design.

## Vulnerabilities

As long as owner protects private key, cryptosystem not cracked, querier who verifies cannot be fooled.

Loss of key  $\Rightarrow$  loss of updatability.

If key cracked, owner announces **compromised**  $\Rightarrow$  loss of that key.

Recipient of **assign** vulnerable to **compromised** announcement by previous owner.

Remedy: pay trusted 3d party to certify transfer with time stamp. (This makes time stamp on **compromised** crucial.)

## Extra services

Key owners may negotiate extra services (at a price) outside of the basic key server.

- Semantic services
  - association of *key* with *identity*, signed by trusted authority (CoT, WoT, ...).
  - time stamp signed by trusted authority.
- Performance services
  - **assign** record res-signed by trusted authority to survive obsolescence of cryptographic method.

It's very important that these extra services need no special co-operation or adjustment by the basic key server. They may be provided by completely separate agents. Particularly, the key server takes no responsibility for anything but storing records and returning answers to queries. The key server does not verify anything about the identity or character of the key owner.

## Associating keys with other identifiers

The basic PKI can leverage other systems of identifiers, by encoding hashes/keys into another sort of identifier. Then the hash/key inherits services from the other sort of identifier.

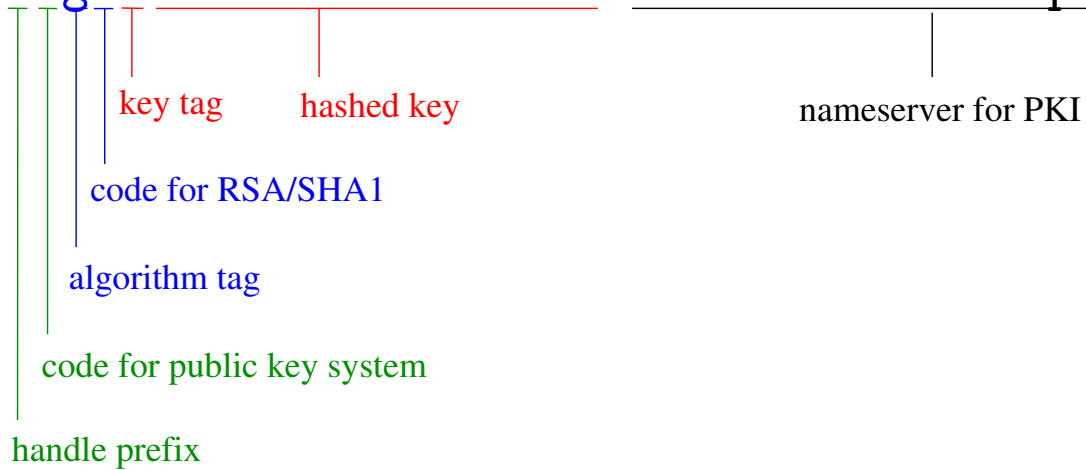
- public-key handles as domain names (Scott Nelson)
- Open Privacy Initiative *nym*s
- host names in secure NFS
- Simple Distributed Security Infrastructure (MIT)
- Simple Public Key Infrastructure (Carl Ellison)

Inexpert users might be unwilling or unable to manage private keys. Describe a service that provides password control of handles for such users. List several choices in the design, and discuss reasons for choosing a particular option. Include the choice to use or not to use hashes of public keys, and the choice of what to do with a private key if the system uses public keys.

## Hashes as domain names

`h1g<mmm>k<n...n>.<PKIroot>`

`h1g5k0061A38F9A3540B9.handleroot.example.org`



## DNSSEC as PKI

Most of PKI functionality already supported by DNSSEC.

PKI	DNSSEC	note
<b>bind</b>	RR, SIG	signed by key owner
<b>delegate</b>	A, SIG	
<b>assign</b>	DNAME, SIG	
<b>compromised</b>	revocation list	not in DNS records

Add code to check owner signatures on receipt of records.

Not necessary for authenticity, but prevents attack by misinformation.

Hierarchical name structure allows interleaving with conventional names.

Discuss uses for PKI zones in the middle of DNS paths. What combinations are not likely to be useful? Think about sequences of reorganization that a handle owner might need to perform.

## Unnumbered point-to-point IP RFC1812

First scheme:

... two routers connected by an unnumbered point to point line are not really two routers at all, but rather two half-routers that together make up a single virtual router. The unnumbered point to point line is essentially considered to be an internal bus in the virtual router.

Second (preferred) scheme:

... a router that has unnumbered point to point lines also has a special IP address, called a router-id in this memo. The router-id is one of the router's IP addresses (a router is required to have at least one IP address). This router-id is used as if it is the IP address of all unnumbered interfaces.

The first scheme doesn't generalize well to a mesh of PPP links.

It also doesn't allow IP software to be used on the PPP link.