# Implementation principles

## Goals:

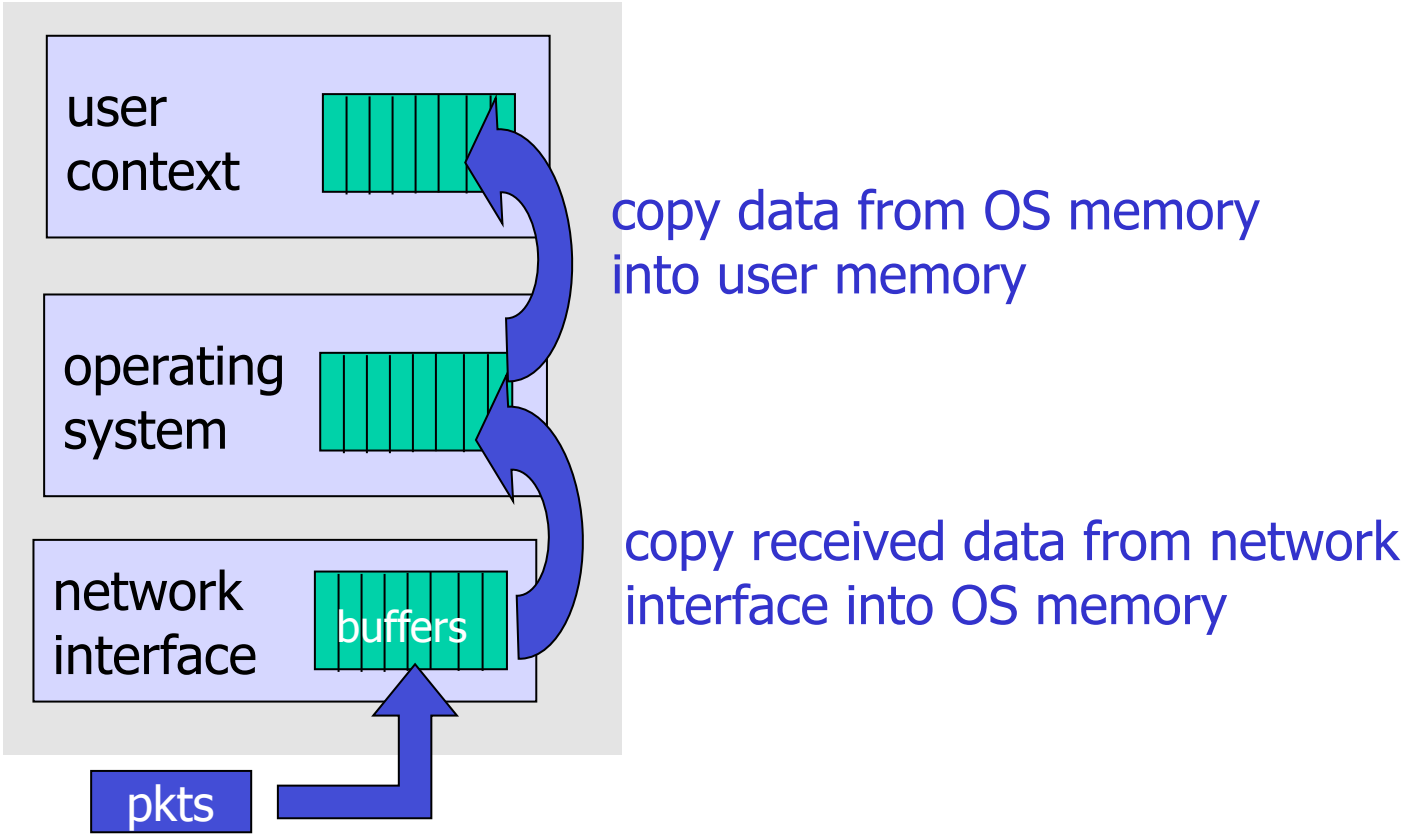
- Identify, study principles that can guide implementation of network protocols
  - Common principles among many protocols
- *Synthesis:* Big picture

## 3 classes of principles:

- System principles
- Improving efficiency while retaining modularity
- Making it go fast
- Cautionary tales: *Think twice*

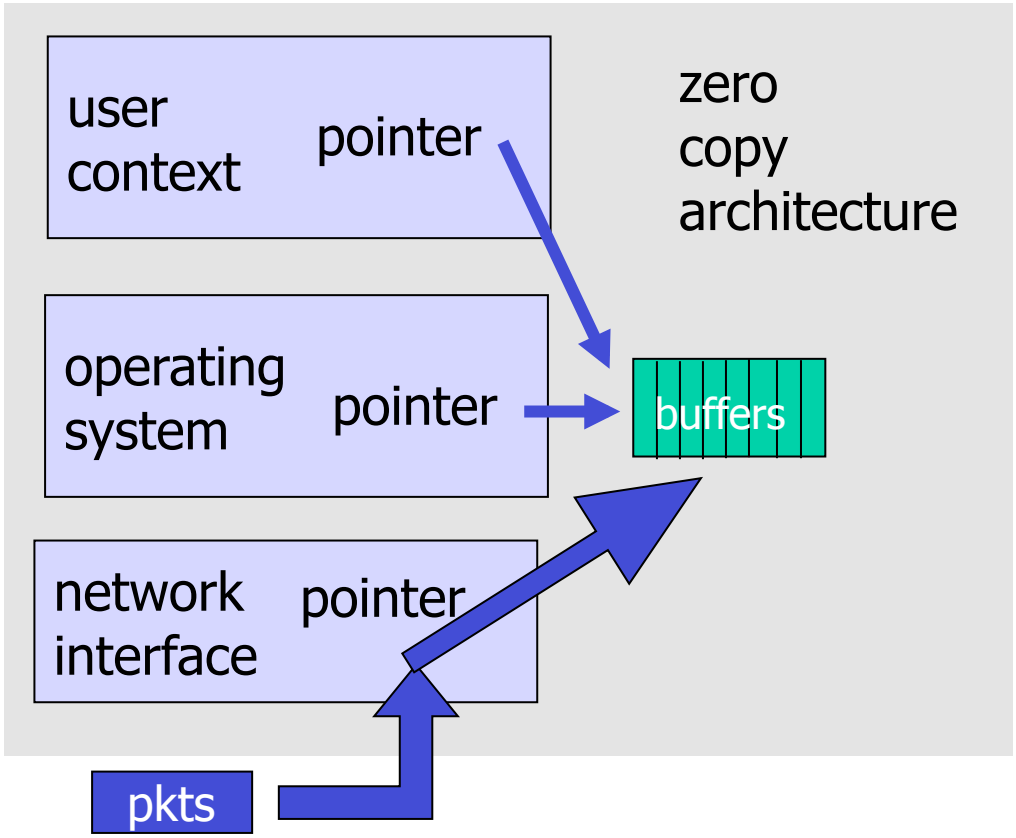
# P1: Avoid *obvious* waste in common situations

- Obvious waste occurs when one does something twice, but (with thought) could do it only one (or never)



# P1: Avoid *obvious* waste in common situations

- Obvious waste occurs when one does something twice, but (with thought) can do it only one (or never)




Real-world example?

## P2a: Use precomputation to shift computation in time

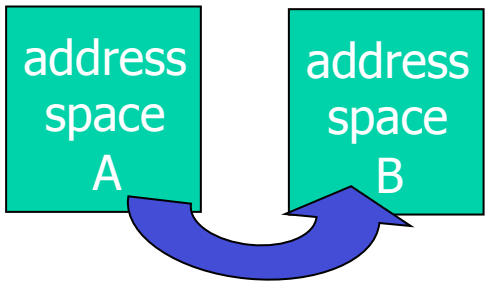
- Precompute quantities to save time at the point of use
  - Precompute/initialize packet headers for packets in a connection
  - Stored video data to be streamed to client:
    Store video data on disk prepackaged into IP packets. Finish filling out packet header when sending

Real-world examples?

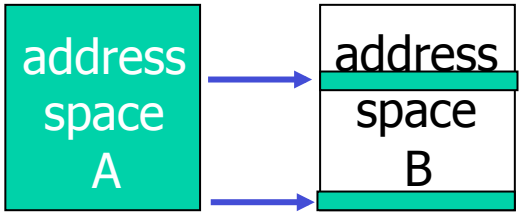
-
-

# P2b: Lazy evaluation: Only do work when it is needed

- Postpone work in hope not needed later, or more time later to do the work. Examples:
  - Data arrives in wrong byte ordering (e.g., big-endian, on little endian machine): Only swap bytes when data actually read (savings if some bytes not read)
  - Copy-on write:




Copy everything then use:
byte-by-byte copy




Copy-on-write:
only copy bytes if user B writes
(makes different from user A)

P2b: Lazy evaluation: Only do work when it is needed

## Real world examples

- Students reading assigned readings:
  "Lazy evaluation" says do the reading only if it is on a homework!

## P2c: Batching to share overhead

- Process work (e.g., packet processing) in batches to amortize setup overhead
- Example:
  - When processing data "up" (or "down") protocol stack, do multiple packets from connection at same time

Real-world example:

- Sending a bunch of printouts to print room at once (only one trip over)
- Laundry – you don't do it every day!
- Factory: Make a lot of one item at a time to only incur machine setup costs once

# P3a: Trading certainty for time

- If a deterministic approach is too slow, try a randomized approach
  - Random access protocols (Ethernet)
  - Statistical sampling of packet flows

## Real-world examples?

- Sampling customers (surveying)
- Shopping schedule: Too much of a hassle to go shopping every sat morning so buy takeout for dinner

# P3b: 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
  - Hot potato routing

## Real-world examples?

- Managing budget
- Quality/time tradeoff: Decreasing marginal returns:
  A 90% solution is good enough. Go one and do the next thing once you're 90% there

## P4: Leverage other system components

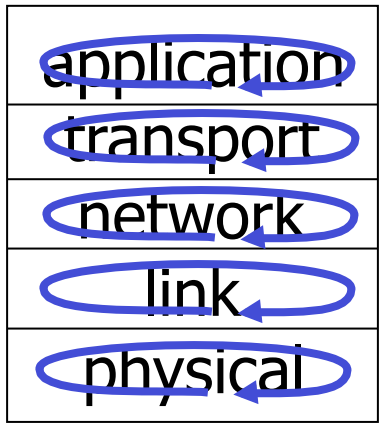
- Network protocols run in computing context:
  - Lay out data structures and implement algorithms to enhance caching effects
  - Realize memory is pages, lay out data structures to not cross page boundaries
- Trade memory for speed: Use precomputed lookup tables to get a value, rather than computing on fly each time

Real-world examples?

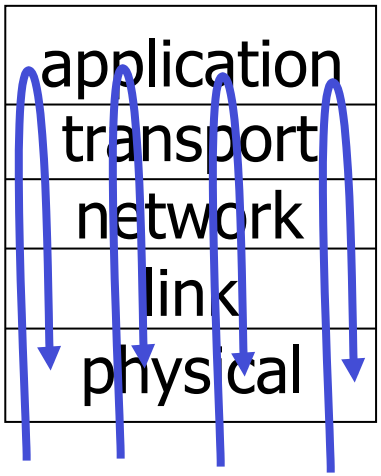
- Precompute answers to questions at a press conference or talk ahead of time

# P5: Throw hardware at the problem

- Special-function hardware: CRC calculation, encryption
- Export processing functions “off-board”
  - Network-interfaces
- Parallelize protocol functions
  - Parallelized protocols (by layer, by connection, by packet)




thread per layer




thread per connection

P6: Replace inefficient general-purpose routines

P7: Avoid unnecessary generality

- One size fits all can lead to inefficiencies
- by specializing, can gain performance (at cost of code bloat)
  - TCP bundles congestion control and error control. What if you want congestion control but don't care about error control (e.g., real-time delivery of multimedia data)
  - other example: ATM cell size (optimized for voice but used for lots of other types of traffic)
  - but then why do we have only a small number of protocols/ approaches at each layer?

## Real-world examples

- help lines: treat you like a dummy, takes time to get specialized assistance.
- standardized pants

## P8: Don't confuse specification with implementation

- 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

P9: Pass information, such as hints, across interfaces

P10: Pass information in protocol headers

- Hints: Additional information that, if correct, can improve protocol performance/processing
- P10 example:
  - Each arriving TCP segment contains the receiver-side memory location for the TCP connection record for that segment
  - Receiver initially passes this information to sender, sender includes in all subsequent segments
  - Receiver can find TCP connection records in future without lookup

Real-world examples:

- Letters of recommendation
- Passing information to sales people (order #, phone #) before actual conversations allows them to prefetch all data

## P11: Optimize for the expected case

- Future system behavior often (but not always) follows expected pattern. Protocol processing sped up by assuming common case happens
- Example 1: Assume next arriving packet at receiver is from same connection as previous one
  - Keep pointer to last accessed TCP connection record, check that one first before searching connection record list
- Example 2: Assume TCP segments arrive in order
  - Can answer question: “is this packet in order?” faster than “is this packet in my receiver window?”
  - Keep pointer to last byte copied into user’s socket buffer, next in-order byte will follow that

P11: Optimize for the expected case

## Real world examples:

- Service lines (e.g., help desks): Expect that user is novice (common case). If you know what you're doing, it takes longer to get help.
- Remote controls and user interfaces: common functions are fast and easy

## P12: Add or exploit state to gain speed

- Maintain state so that you don't have to compute something every time
- Example: Resource allocation
  - Keep track of resources allocated so know what is free (alternative: go around to all resource users, compute what is being used. What not used is free

Real world example:

- Checking account balance

P13: Optimize degrees of freedom

P14: Use special techniques with small sets of values

P15: Use algorithmic techniques

# Some cautionary tales

Q1: Is it worth improving performance?

- Does performance increase have high complexity cost?
- KISS: keep it simple silly
- E.g., router has so many protocols. Would routers be more “robust” if there were fewer protocols?

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

# Some cautionary tales

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

# Some cautionary tales

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)

# Some cautionary tales

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.

# Any missing cautionary tales?

- Make sure no one else has done it
  - Corollary: If you have thought it up, it's likely that someone else has (or will soon) too
- Stress: complexity versus performance tradeoff: what really matters?
- Evaluate benefits under meaningful conditions

# Radia Perlman's Folklore of protocol design

- Collect various tricks and "gotchas" in protocol design.
- "Here are several ways to solve problem X", with technical explanation of pros/cons
- Some "real world" examples

We'll cover most, not all, "tricks and gotchas"

# 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."

Charles Mingus

"Making the simple complicated is commonplace; making the complicated simple, awesomely simple, that's creativity!"

Why are protocols overly complex?

- Design by committee
- Backward compatibility
- Flexibility: Heavyweight swiss army knife
- Unreasonable striving for optimality
- Underspecification
- Exotic/unneeded features

# More folklore/advice:

Know the problem you are trying to solve:

- Have at least one well-defined problem in mind
- Solve other problems without complicating solution?

Think about scaling

- 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

Operation above capacity

- Protocol should degrade gracefully in overload, at least detect overload and complain
- How does protocol break and die?
- Can't just die under overload

Identifiers: Two approaches

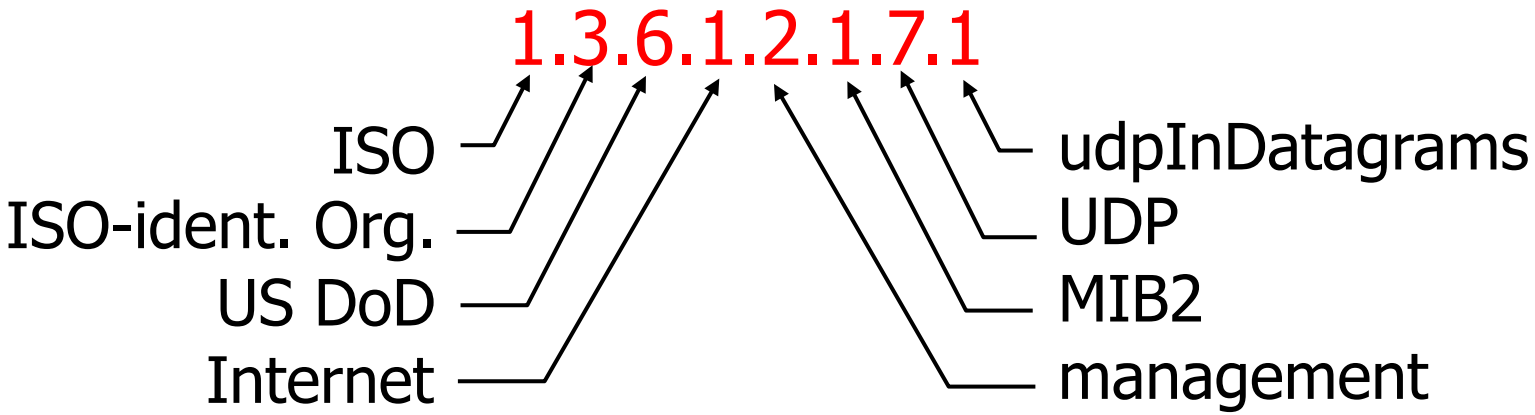
- Highly encoded universal IDs: E.g., upper layer protocol # assigned by IANA
- General purpose object identifiers, as in ASN.1

# SNMP naming

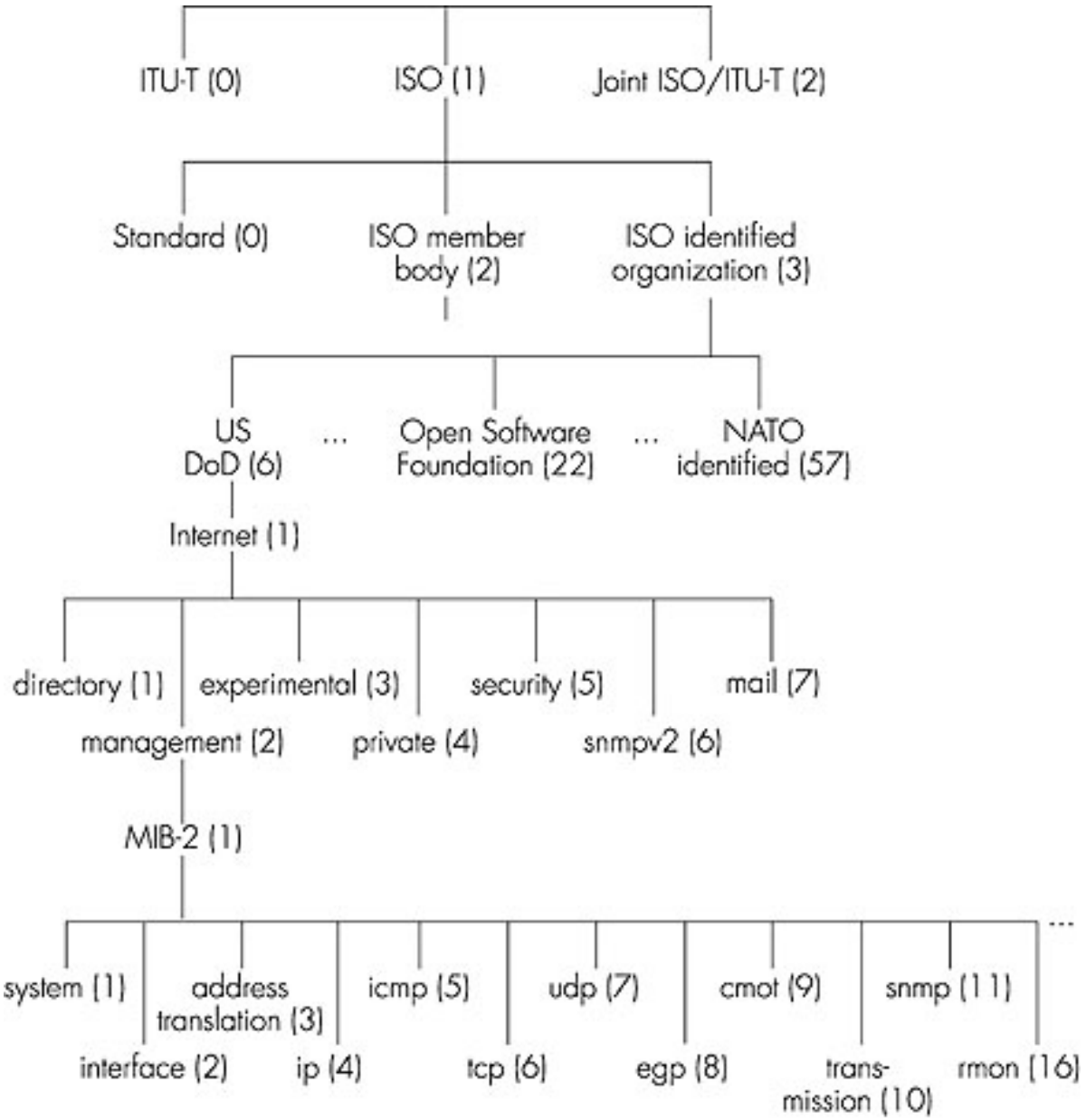
*Question:* How to name every possible standard object (protocol, data, more ...) in every possible network standard??

*Answer: ISO Object Identifier tree:*

- Hierarchical naming of all objects
- Each branch point has name, number



# OSI Object Identifier Tree




Check out www.alvestrand.no/harald/objectid/top.html

# Assigned Internet Protocol numbers

From RFC 1700:

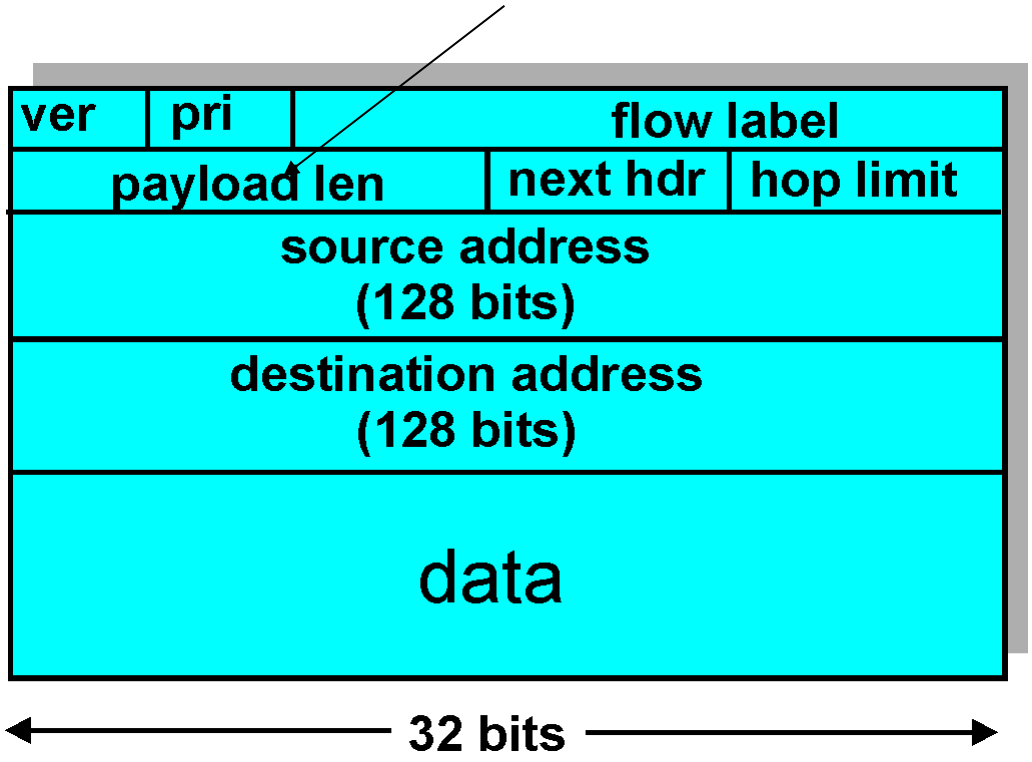
| Decimal | Keyword | Protocol | References |
|---|---|---|---|
| 0 | | Reserved | [JBP] |
| 1 | ICMP | Internet Control Message | [RFC792,JBP] |
| 2 | IGMP | Internet Group Management | [RFC1112,JBP] |
| 3 | GGP | Gateway-to-Gateway | [RFC823,MB] |
| 4 | IP | IP in IP (encasulation) | [JBP] |
| 5 | ST | Stream | [RFC1190,IEN119,JWF] |
| 6 | TCP | Transmission Control | [RFC793,JBP] |
| 7 | UCL | UCL | [PK] |
| 8 | EGP | Exterior Gateway Protocol | [RFC888,DLM1] |
| 9 | IGP | any private interior gateway | [JBP] |
| 10 | BBN-RCC-MON | BBN RCC Monitoring | [SGC] |
| 11 | NVP-II | Network Voice Protocol | [RFC741,SC3] |
| 12 | PUP | PUP | [PUP,XEROX] |
| 13 | ARGUS | ARGUS | [RWS4] |
| 14 | EMCON | EMCON | [BN7] |
| 15 | XNET | Cross Net Debugger | [IEN158,JFH2] |

# More folklore/advice:

## Optimize for common case

- Seen this before
- Nice example: IPV6 payload (packet) length field

- Payload length:
  2 bytes
- If packet longer:
payload length = 0
4 byte length field found
in IP options
- Designers chose against
4-byte header to optimize
common case:
2 bytes are enough



# 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

## 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

# More folklore/advice:

Making systems "robust": Many forms of robustness

- Immediately adapt to failure/change
- Self-stabilization: Eventually adapt to failure/change
- Byzantine robustness: Will work in spite of malicious users

- Maybe better to crash than degrade when problems occur: Signal problem exists
- Techniques for limited spread of figures

# Missing folklore/advice:

- 
- 
- 
- 

# The end: Implementation principles

Goals:

- Identify, study principles that can guide implementation of network protocols
  - Common principles among many protocols
  - "Folklore" of protocol design
- *Synthesis:* Big picture
- *Architecture and implementation:*
  Both more art than science