Computer Networks

Acknowledgements

- Slides contain material adapted from lecture notes developed by
  - Authors of "Computer Networks" book by Andrew Tanenbaum & David Wetherall
- Slides contain images from other online resources

What is a network?

"A group or system of *interconnected* people or things"

Example networks

- *Postal* network
  - Delivers letters/packages between sender & receiver
  - Sender & receiver identified by *postal address*

Example networks

- Plain old *telephone* network
  - Provides voice communication between users
  - User identified by *telephone number*

Computer Network

- Provides communication between *computers*
  - I.e., computers can *exchange* data

What can computer networking give us?

- *Communication* or information sharing
  - E.g., electronic mail, message broadcast, etc.
- *Resource sharing* (both hardware & software)
  - E.g., printers, disks, application software
- *Remote* computing (location *independence*)
  - Geography need not be a restriction!
- …

Computer Network

- Provides communication between *computers*
  - I.e., computers can *exchange* data
- Need a way to *identify* computers
  - In the Internet, this is done via *IP addresses*

IP address

- IP address identifies any device connected to the Internet
- IPv4 *dotted decimal* notation
  - Four 8-bit numbers separated by dots
  - 10101100 00010000 00000000 01100100 → 172.16.0.100
- IPv6 notation
  - 8 groups of 4 hexadecimal digits each separated by colons
  - 2001:0DB8:0000:0000:0000:ff00:0042:8329

Who are the *actors* involved?

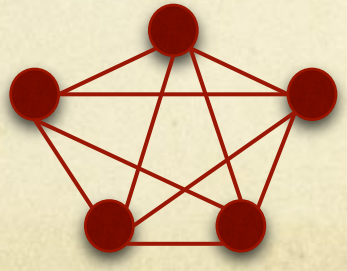
- *Source* or *sender* [host/end system]
  - Generates data to be transmitted
- *Transmitter* [router/switch]
  - Converts data into transmittable form (signals)
- *Communication medium*
  - Carries data (in the form of signals)
- *Receiver* [router/switch]
  - Converts signals into data
- *Destination* or *receiver* [host/end system]
  - Receives data

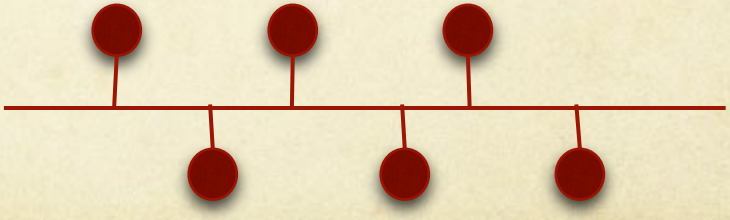
Medium of communication

- Networks may use different communication media...
  - *Wired* network
    - Co-axial cables, fibre-optic cables, etc.
  - *Wireless* network
    - WiFi, bluetooth, etc.

Who is connected to *whom*?

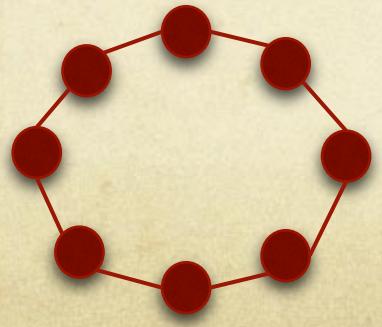
- *Every* comp can directly communicate with *every* other comp
  - *Mesh* topology
    - Each computer connected to every other computer
  - *Bus* topology
    - All computers connected to *common* bus

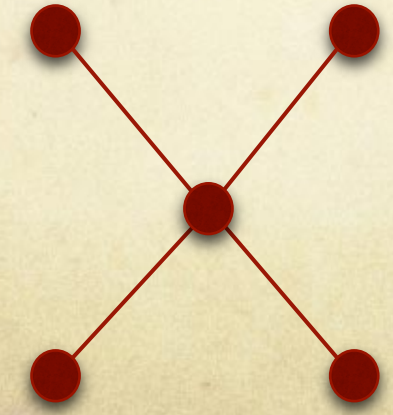


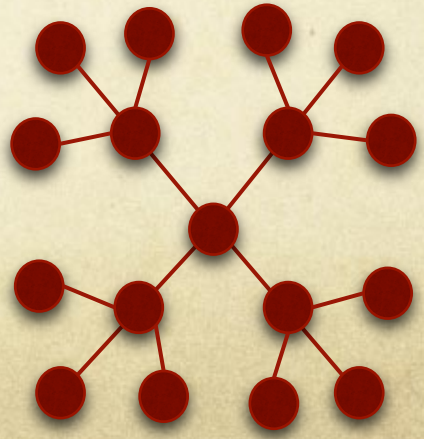


Who is connected to *whom*?

- We can always get to a computer *via* another
  - *Ring* topology
    - Each computer connected to two neighbors
  - *Star* topology
    - Computers connected to *central* hub/computer through spokes
  - *Extended star* (tree) topology
    - Improves scalability of star topology!

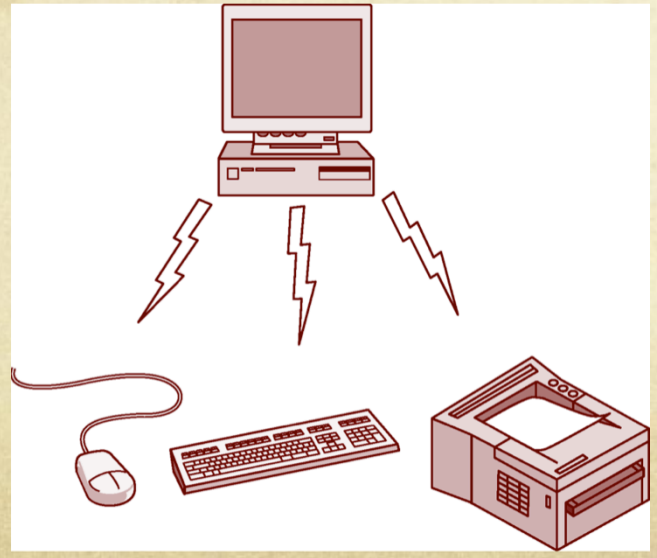


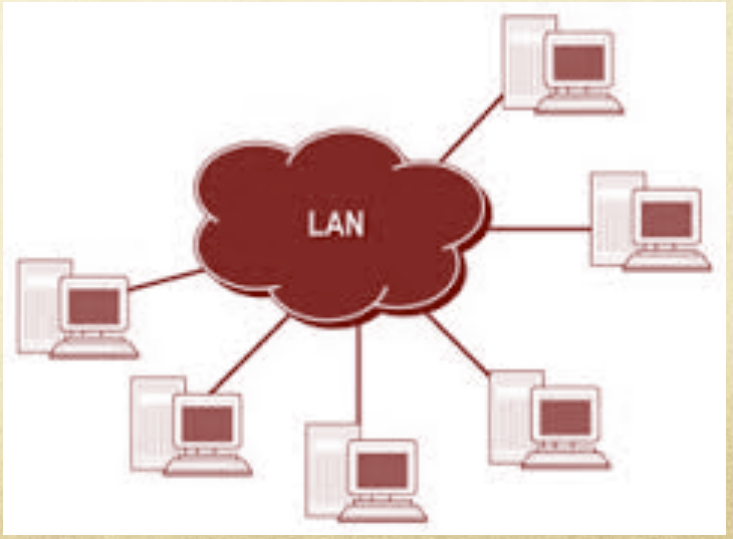




How *large* is a computer network?

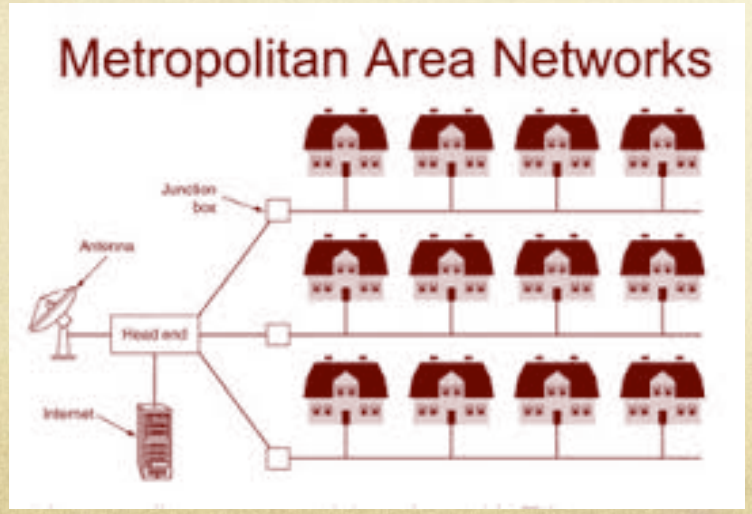
- Network scale can vary...
  - Within 1 square meter or so
    - Personal Area Network (*PAN*)
  - Within room, building or single campus
    - Local Area Network (*LAN*)

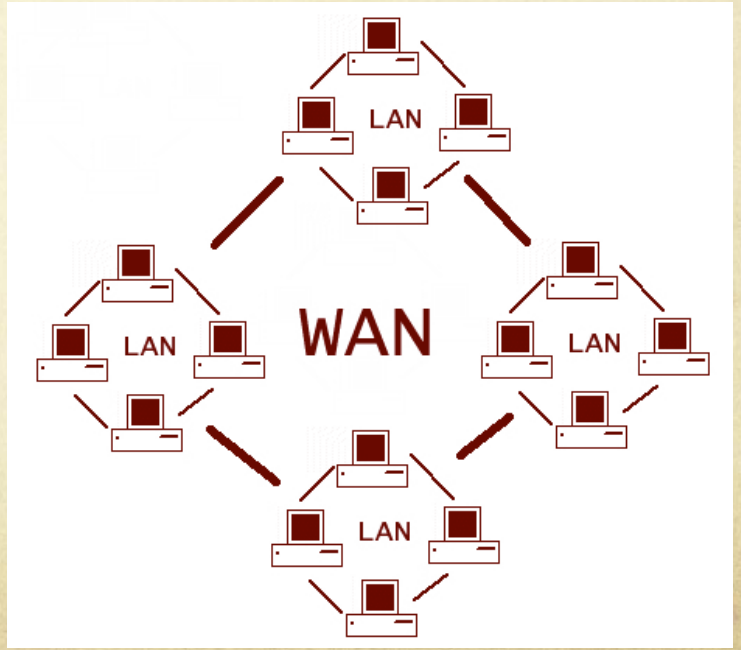




How *large* is a computer network?

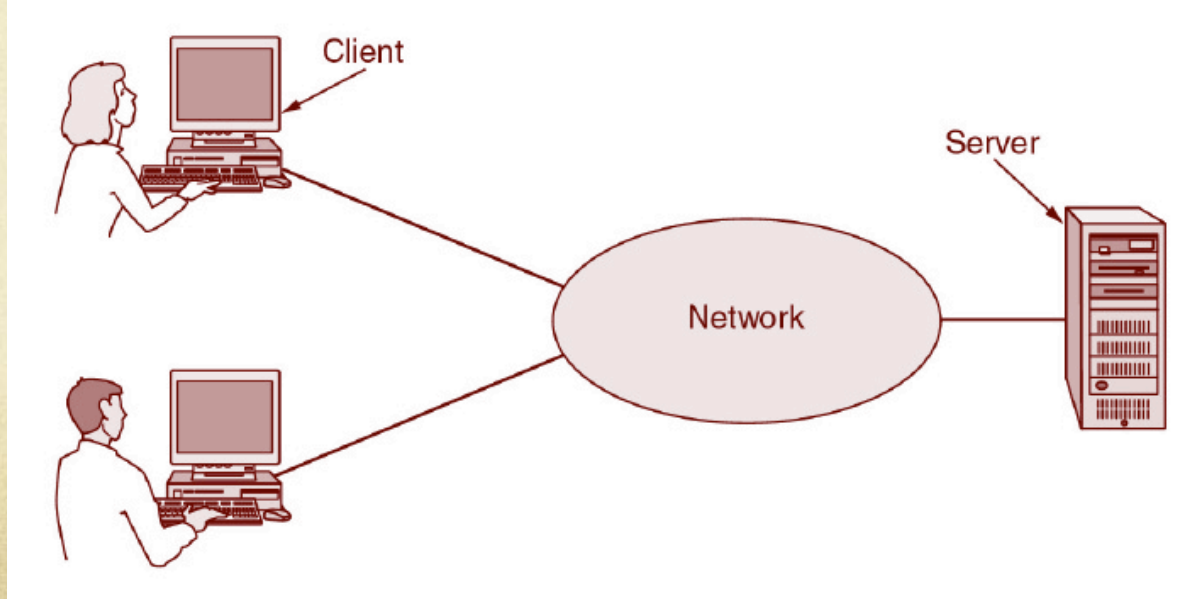
- Network scale can vary…
  - City wide
    - Metropolitan Area Network (*MAN*)
  - Country or continent wide and beyond
    - Wide Area Network (*WAN*)





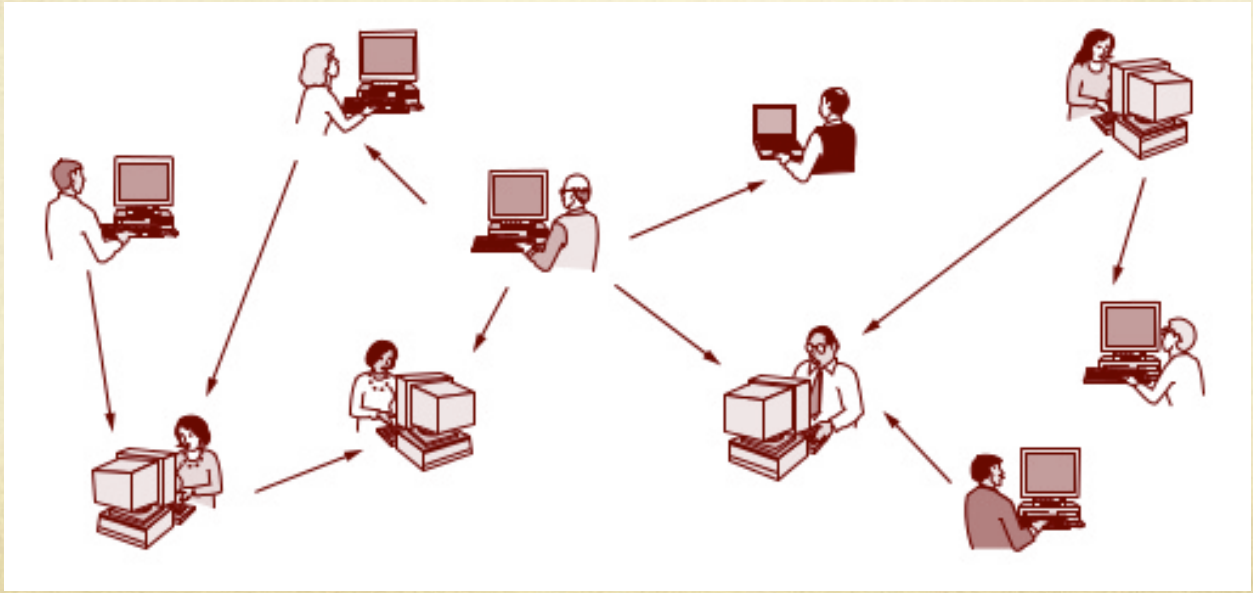
Network architecture/model

- Central computer provides services & resources
- Other computers request for services & resources
- → *Client-server* model (e.g., online banking)



Network architecture/model

- All computers in network are equal...no hierarchy
- → *Peer-to-peer* model (e.g., BitTorrent)



Network software

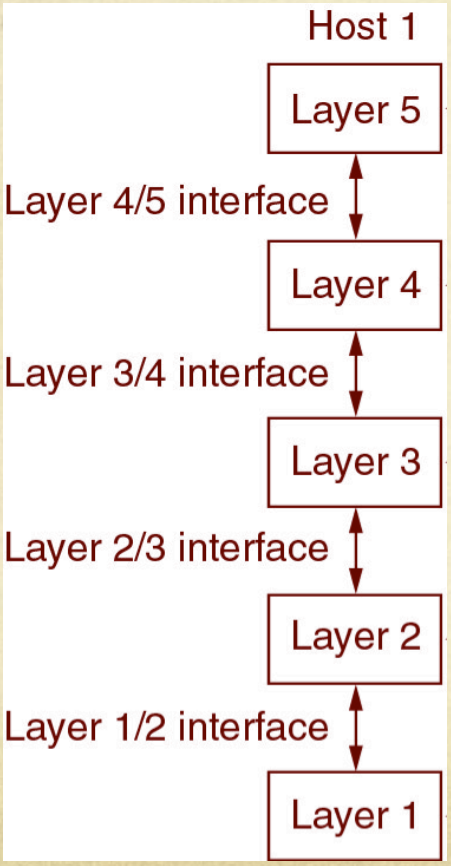
- Special software needed on computer to enable *networking*
  - I.e., to enable sending/receiving of data to/from computer
- Computer initiating data transfer → *sender*/*source* computer
- Eventual recipient of data → *receiver*/*destination* computer

Design goals

- *Reliability* → should operate correctly
- *Security* → should provide data protection
- *Resource sharing* → should efficiently share available network resources
- *Scalability* → should be able to support increase in size

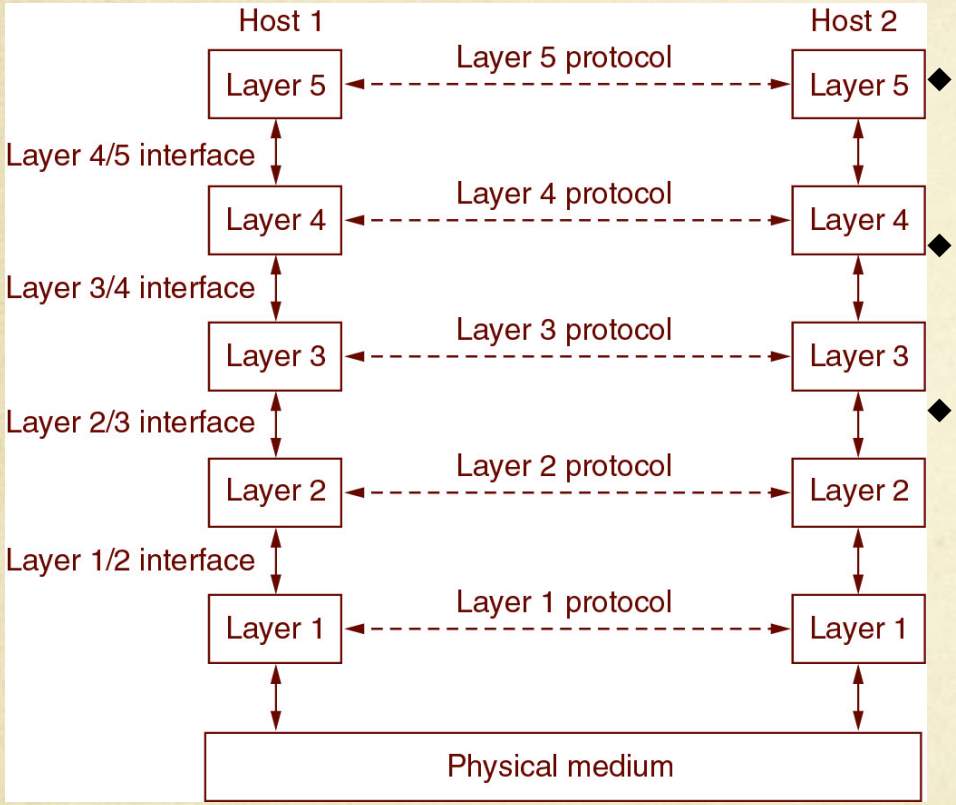
Network software design

- *Layered* design used to manage complexity of software
  - Divide responsibilities among different layers



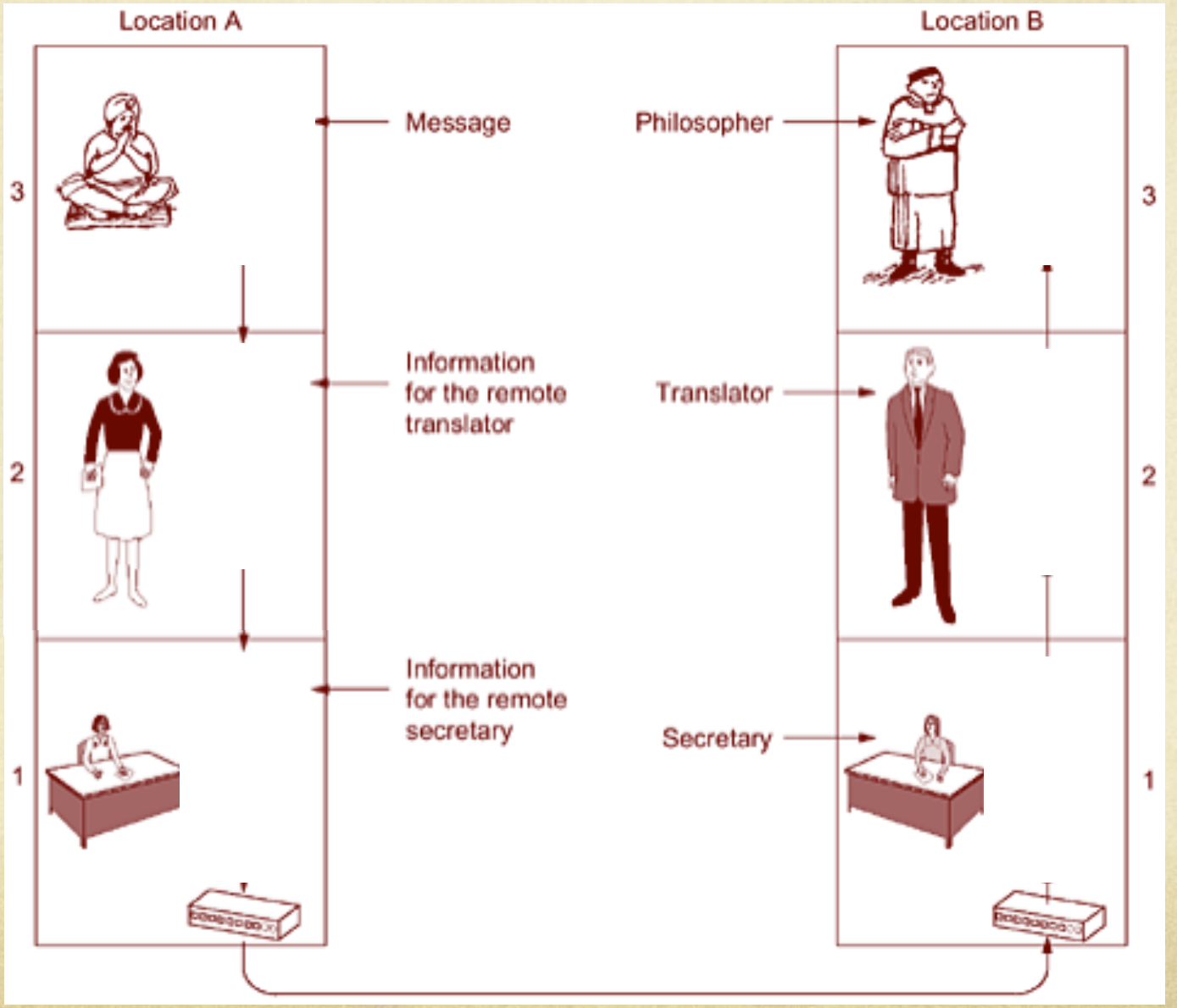
- Each layer
  - *Abstracts* set of services (i.e., hides underlying details)…
  - … & provides well-defined *interface* to layer above it
  - *Directly* interacts only with layer immediately below & above it

Layered approach



- Data travels from top to bottom layer on *sender* side
- Data travels from bottom to top layer on *receiver* side
- Each layer uses *protocol* for interaction with same layer in other computers
  - Although there is no direct interaction…
  - …layer *n* of one computer *understands* information communicated by layer *n* on other computer

Layering & protocol analogy...



OSI reference model

- OSI stands for *Open Systems Interconnection*
- Has *seven* layers
- Specifies *function* of each layer...
- ...but does not specify exact *services*/*protocols* to use

OSI reference model

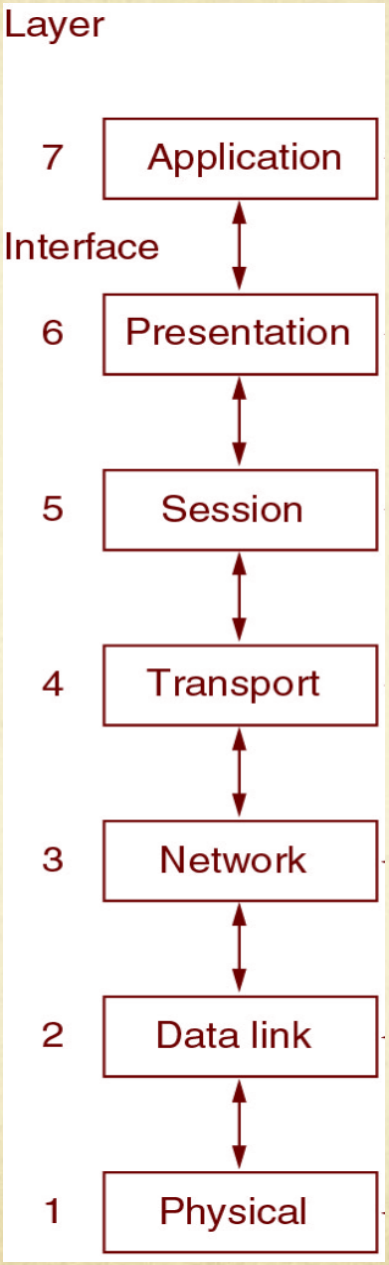
- *Application* layer
  - User interface to networking services
  - Provides variety of commonly used functions
    - E.g., file transfer, e-mail, etc.
- *Presentation* layer
  - Handles formatting details for data
    - E.g., encryption, compression, stream formatting, etc.
- *Session* layer
  - Provides services to initiate, maintain & end connection b/w sender & receiver

OSI reference model

- *Transport* layer
  - Provides *end-to-end* message delivery service
  - Controls rate of transfer & ensures network is not overloaded
    - → *Flow control* & *congestion control*
- *Network* layer
  - Controls actual *transfer* of unit of data through network
  - Determines path to take from source to destination (*routing*)
  - Forwards unit of data to next node/router in path
  - Shares *congestion control* responsibility

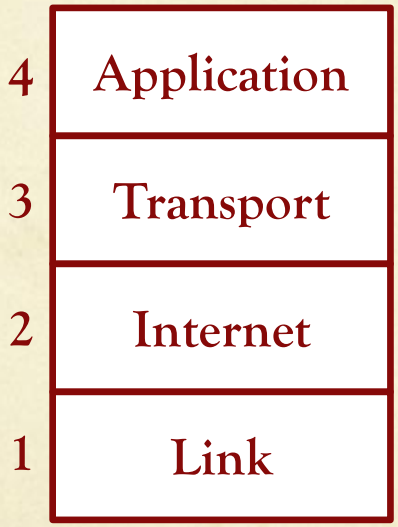
OSI reference model

- *Data link* layer
  - Transmits data over *link* between two nodes
  - Performs *error* and *flow* control over link
  - Determines when computer has the *right* to access medium
    - Medium Access Control (MAC) sub-layer
- *Physical* layer
  - Consists of basic networking *hardware*
  - Transmits/receives *raw bits* over communication channel
  - Determines how connection is established
  - Determines mode of transmission


Layer
7
Application
Interface
6
Presentation
5
Session
4
Transport
3
Network
2
Data link
1
Physical

TCP/IP reference model

- Has *four* layers
  - Was used in *ARPANET* (research n/w sponsored by DoD)



- Named after two of its primary *protocols*
  - *In this model, protocols are more important than strict layering*
  - *So, we will mention specific protocols in this context*

TCP/IP reference model

- *Application* layer
  - Roughly covers functions of *application*, *presentation* & *session* layers in OSI model
  - Widely-used application layer protocols for user services
    - Simple Mail Transfer Protocol (*SMTP*)
    - Hypertext Transfer Protocol (*HTTP*)
    - File Transfer Protocol (*FTP*)

TCP/IP reference model

- *Transport* layer
  - Similar to *transport* layer in OSI model
  - Provides *end-to-end* message delivery service independent of underlying network
    - Can provide *reliability* if needed
  - Provides *error control*, *flow control* & *congestion control*
  - Widely used protocols
    - Transmission Control Protocol (*TCP*)
    - User Datagram Protocol (*UDP*)

TCP/IP reference model

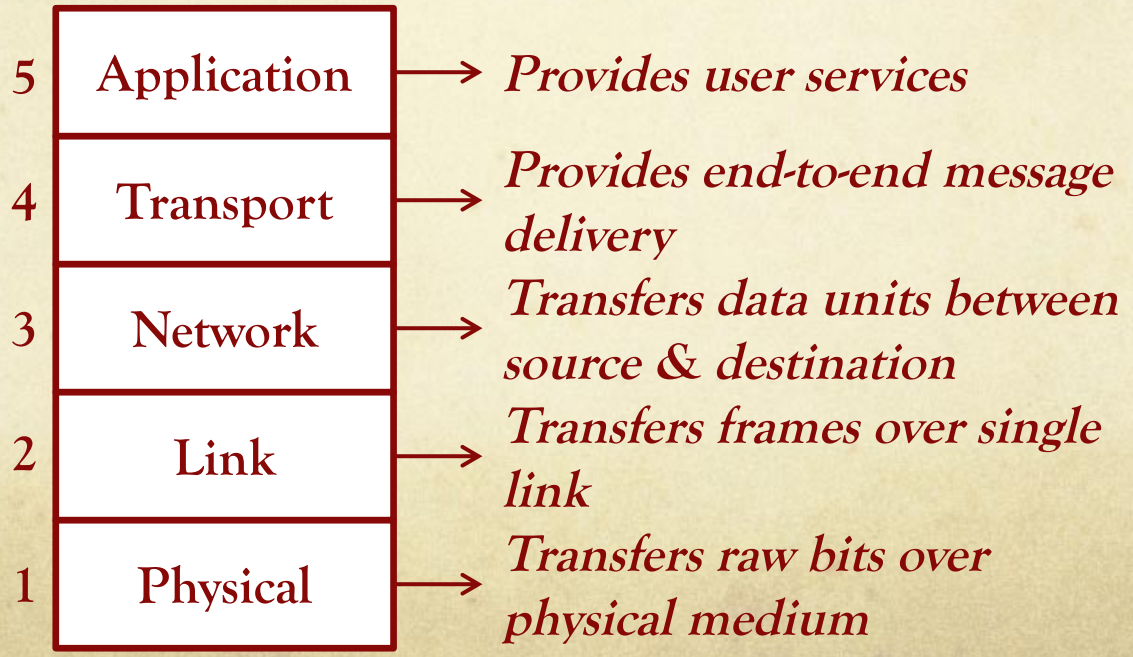
- *Internet* layer
  - Roughly corresponds to *network* layer in OSI model
  - Responsible for sending *units of data* over network
    - Performs *routing* for units of data
  - Provides unreliable (*best-effort*) service
    - Data units could arrive *out of order* at destination
  - Principal, very widely used protocol
    - Internet Protocol (*IP*)

TCP/IP reference model

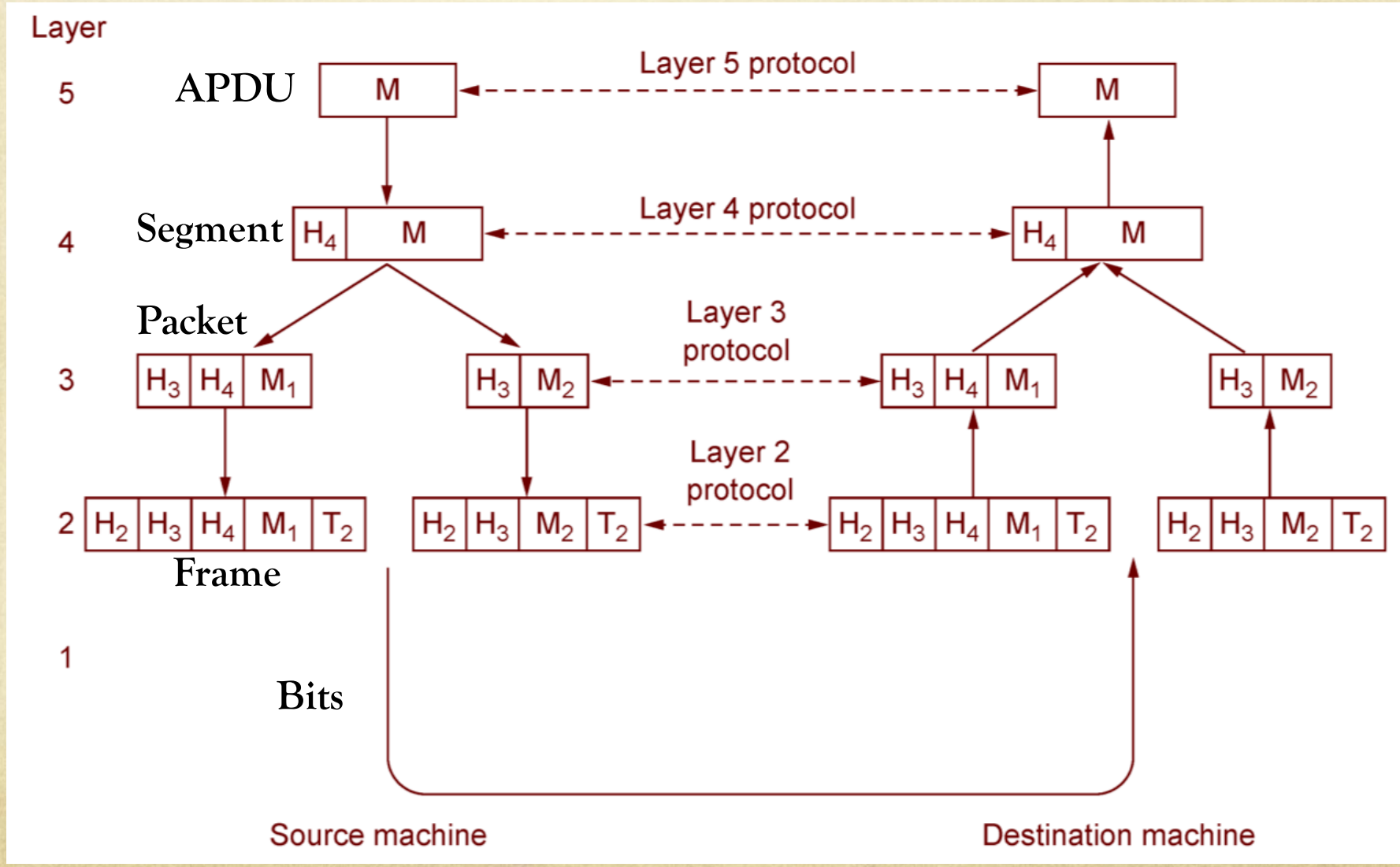
- *Link* layer
  - Roughly corresponds to *data link* & *physical* layers in OSI model
  - Responsible for moving data units over *link* between two *hosts*
    - Provides *interface* between hosts & transmission links
    - Includes *protocols* to describe local network topology

Discussion

- *OSI* model strength is model or layering itself
  - *Excluding presentation & session layer*
- Strength of *TCP/IP* model is its protocols
- Tanenbaum book considers *hybrid* model



Peer layer communication



Types of network services

- Layer can provide two types of services to layer above it
  - *Connection-oriented* service
    - Establish (logical) connection between source & destination
      - Might include negotiation to decide on parameters
    - Use connection to transfer data
    - Release/end connection
  - *Connection-less* service
    - No connection is established between source & destination
    - Each unit of data carries full destination address
    - Data unit routed through network independent of other data units

Types of network services

- Each of these services can be of two kinds
  - *Reliable* service
    - Ensures data accuracy & integrity
    - Requires acknowledgement by receiver
    - Has significant overhead
  - *Unreliable* service
    - Does not check to see whether data received
    - Data could sometimes get lost/delayed/corrupted
    - Less overhead

Next on the agenda is understanding the details of the layers & some widely-used protocols within some of them.

The Physical Layer

Electrical, timing & other interfaces for transfer of data bits between two adjacent nodes

The (Data) Link Layer

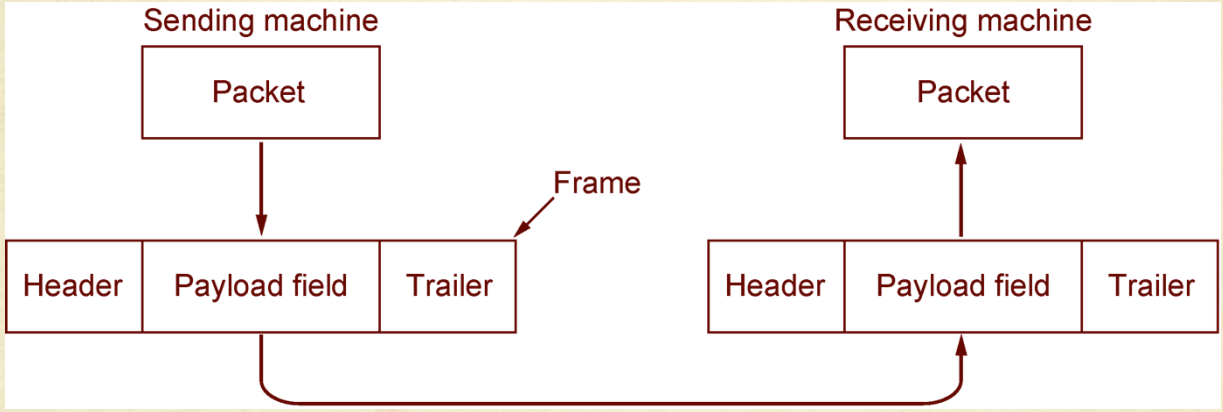
What does the link layer do?

- Provides well-defined services to the network layer…
  - Transfer of data unit b/w *adjacent* nodes (i.e., over *single* link)
    - Unit of data given by network layer is called a *packet*
    - Use *physical* address of nodes (MAC address)
    - Receiver must be able to *distinguish* packets from each other
  - Detect and/or correct transmission errors
    - Ensure packet that reaches receiver network layer is *error-free*
    - *Error control*
  - Regulate flow of packets
    - Ensure receiver does not get *inundated*!
    - *Flow control*

These design goals form recurring theme in multiple layers!

Packet transmission

- Receiver must be able to *distinguish* packets from each other
  - Encapsulates packets given by network layer into *frames*
    - Adds *header* & *trailer* to packets for this



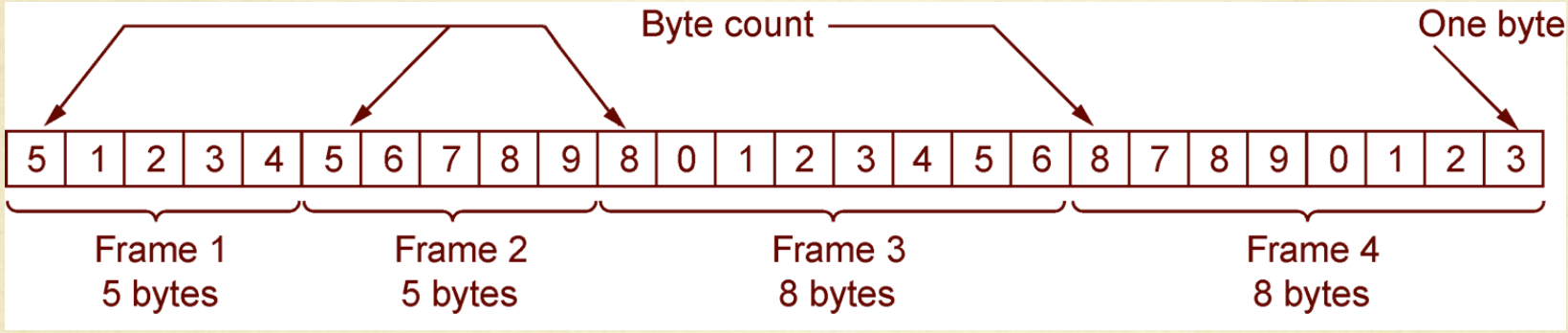
  - Need *understanding* b/w sender & receiver link layers on format of header/trailer
    - I.e., need link layer *protocol* understood by both computers

Framing

- *Byte count*
- *Flag bytes with byte stuffing*
- *Flag bits with bit stuffing*
- *Physical layer coding violations*

Byte count

- Field in header contains number of bytes in frame



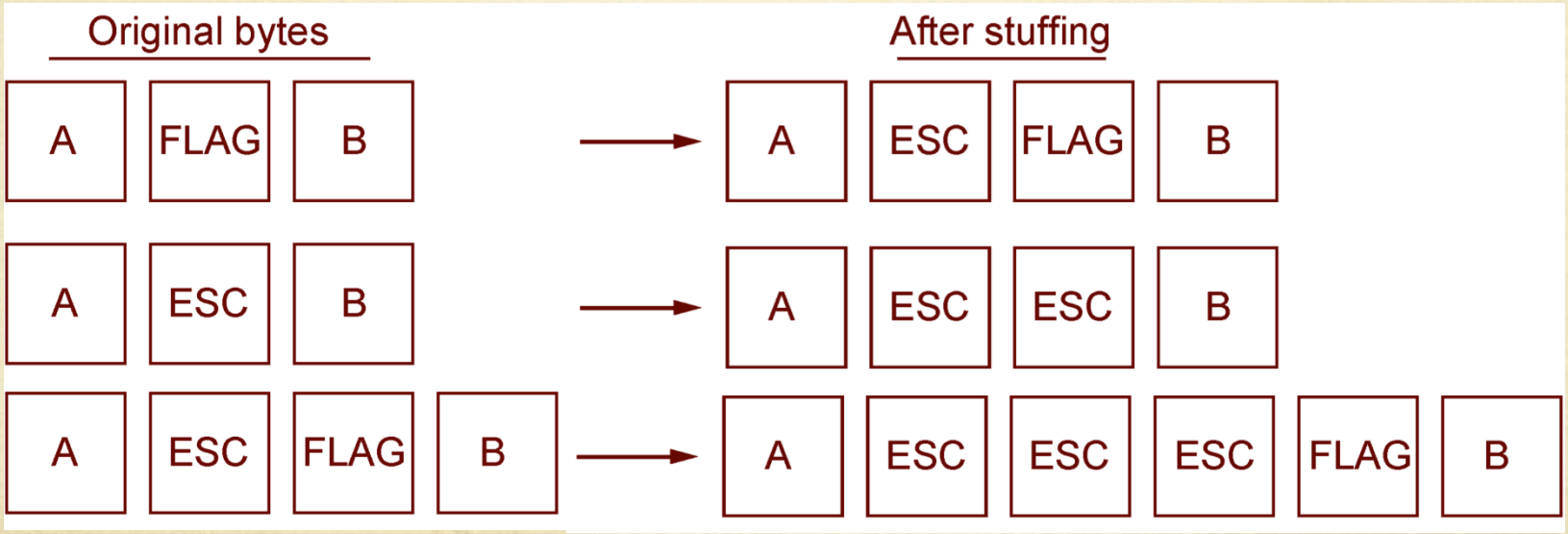
Flag bytes with byte stuffing

- Each frame starts & ends with special bytes
  - Could use same byte – *flag byte* – for start & end

- If receiver sees two *consecutive* flag bytes
  - End of one frame & beginning of next frame

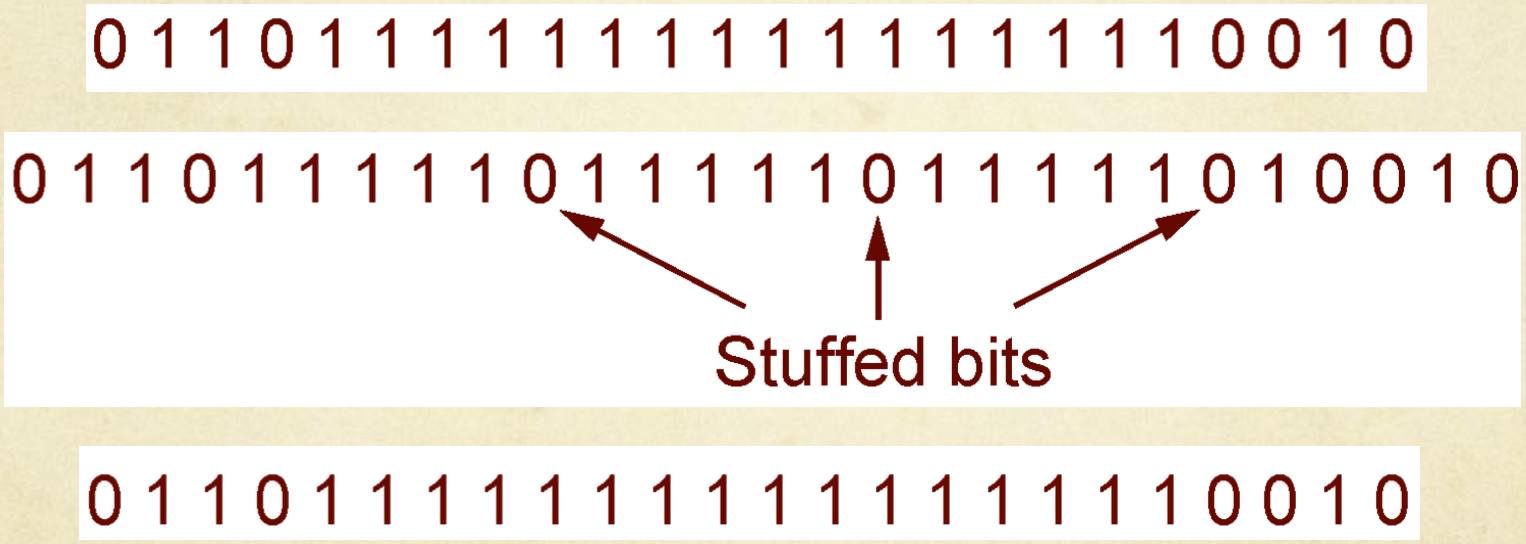
Flag bytes with byte stuffing

- What if flag byte happens to occur within payload?
  - *Byte stuffing* – insert special *escape byte* before flag byte
  - Escape byte itself can be escaped if needed



Flag bits with bit stuffing

- Each frame starts & ends with special *sequence of bits*
  - *Bit stuffing* used if payload contains same sequence of bits

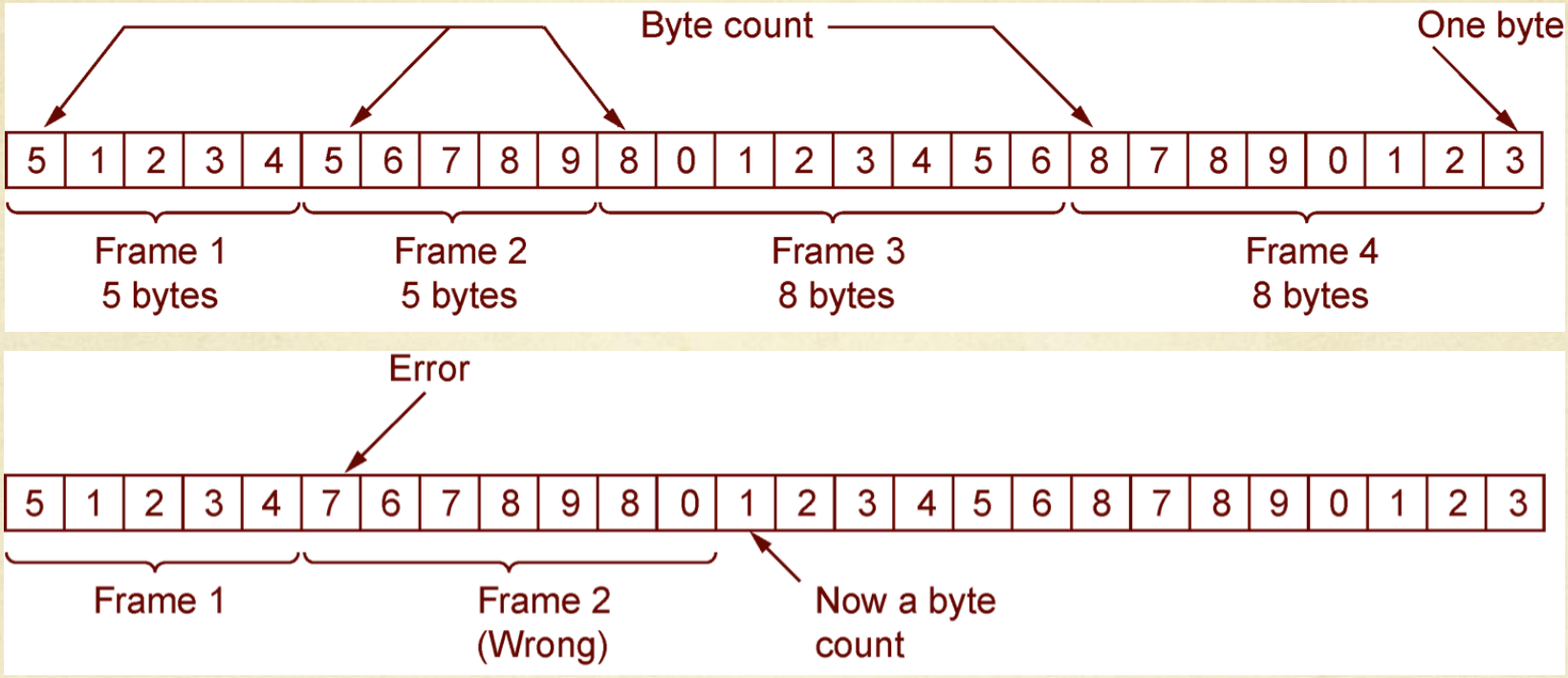


- In byte/bit stuffing, *frame lengths vary* based on payload

Physical layer coding violations

- *Encoding* bits as signals includes redundancy
  - Some signals can never occur in payload
  - Use these "*reserved*" signals as frame delimiters
  - In practice, combination of different methods may be used!

Error control



- Frame header/trailer include information for
  - *Error detection*

    and/or
  - *Error correction*

Error detection

- Include information that allows receiver to *detect* errors
- If error, receiver can request *re-transmission* of data
- Error *detecting* codes
  - Parity
  - Checksum
  - Cyclic redundancy check
- Useful for transmission media where errors are *rare*

Error correction

- Include information that allows receiver to *detect* & *correct* errors
- Error *correcting* codes
  - Hamming code
  - Reed-Solomon code
- Useful for transmission media where errors occur *often*

The Network Layer

What does the network layer do?

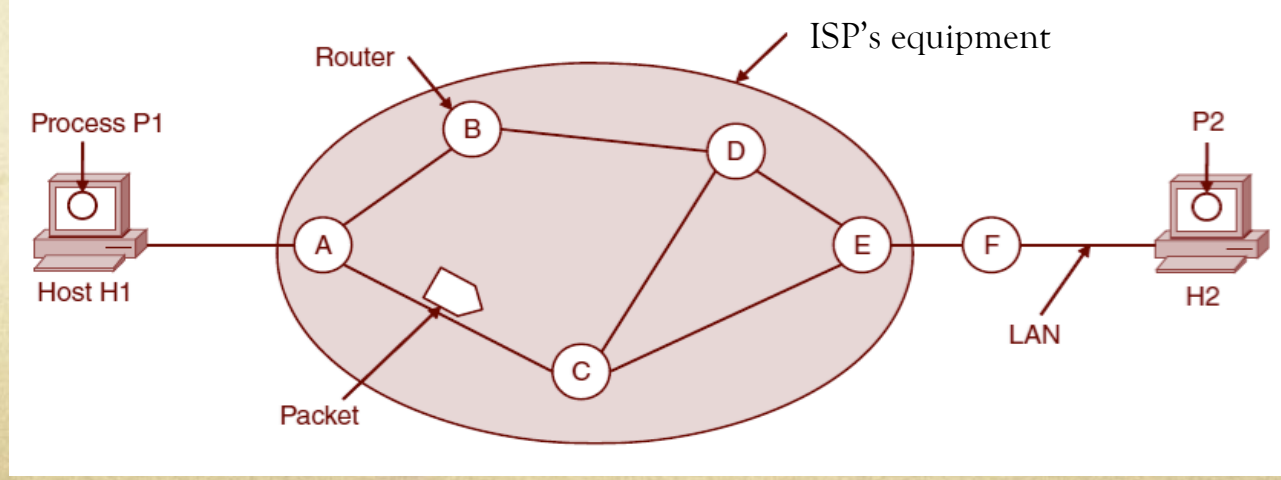
- *Goal*: Transfer packets b/w *endpoints* via multiple links
- *Routing & forwarding*
  - Find path from source to destination
  - Forward packets along path
- *Congestion control*
  - Must avoid/recover from network congestion

What does the network layer do?

- *Internetworking*
  - Must support *joining* of multiple networks into network of networks
  - Must provide *uniform addressing* across entire network of networks
  - Details of physical network, number/types of nodes & network topology, etc. must be *hidden* from transport layer

Routing basics

- Routing → identifying *path* between endpoints
- What happens along path?
  - *Store-and-forward* packet switching
    - Wait for full frame to arrive & *link* layer to verify frame
    - *Forwards* packet to next router along identified route
    - When destination reached, send packet up to *transport* layer



Routing goals

- Could have *multiple* possible routes between A & B
  - Need to make appropriate choice
- Goals of routing algorithm
  - Should deliver packets to their *intended* destination
  - Should have low overhead (*simplicity*)
  - Should be *efficient* (minimize delay, maximize throughput, …)
  - Should cope with changes in topology & traffic (*robustness*)
  - Should not take forever to converge to choice (*stability*)
  - Should treat different nodes *fairly*

Types of routing algorithms

- *Non-adaptive* or *static*
  - Routes fixed offline & simply stored in tables
  - Makes sense if there's only one clear choice
- *Adaptive* or *dynamic*
  - Routes changed to reflect changes in network state
    - E.g., topology, traffic changes
  - Have some optimization criteria

Routing algorithm examples

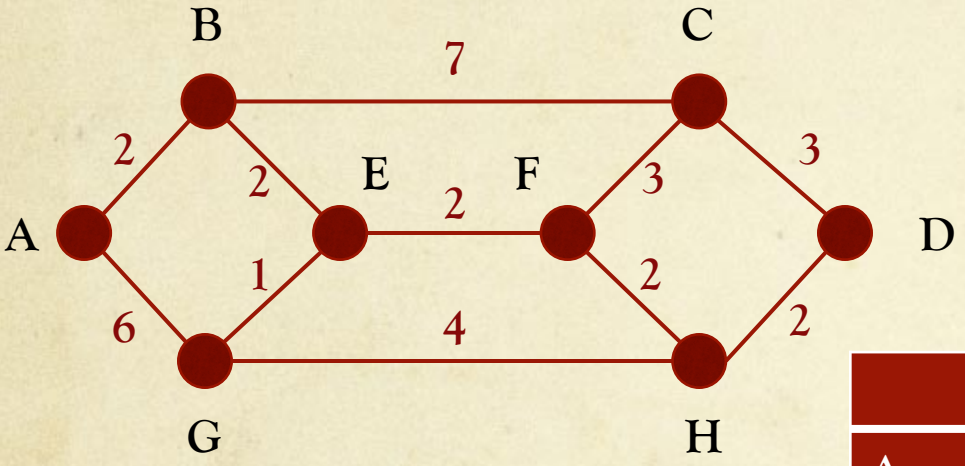
- Shortest path routing
- Flooding
- Distance vector routing
- Link state routing
- Hierarchical routing

Shortest path routing

- Route packets along *shortest* path b/w src & dst
- Shortest path calculated & *stored* in routing table
- Shortest path could mean
  - Lowest number of hops
  - Shortest geographic distance
  - Lowest average delay (i.e., fastest path)
- Can assign *labels* using weighted "distance"

Djkstra's algorithm

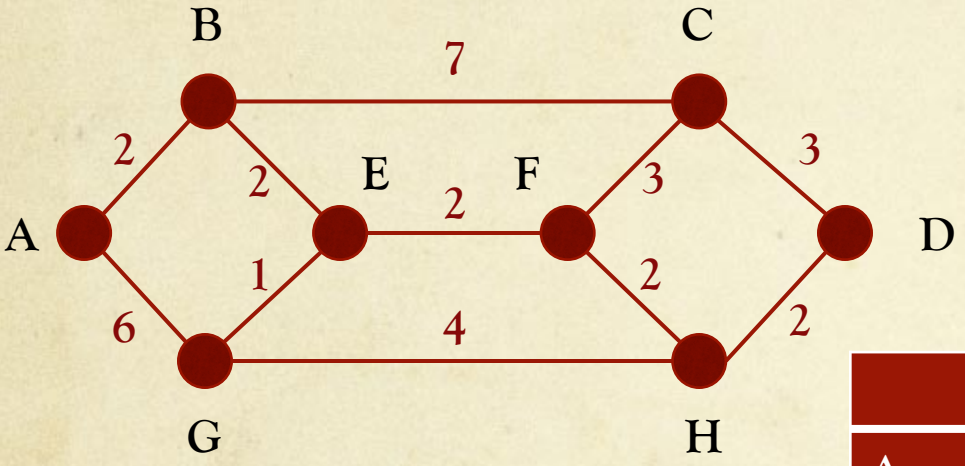
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A								

Djkstra's algorithm

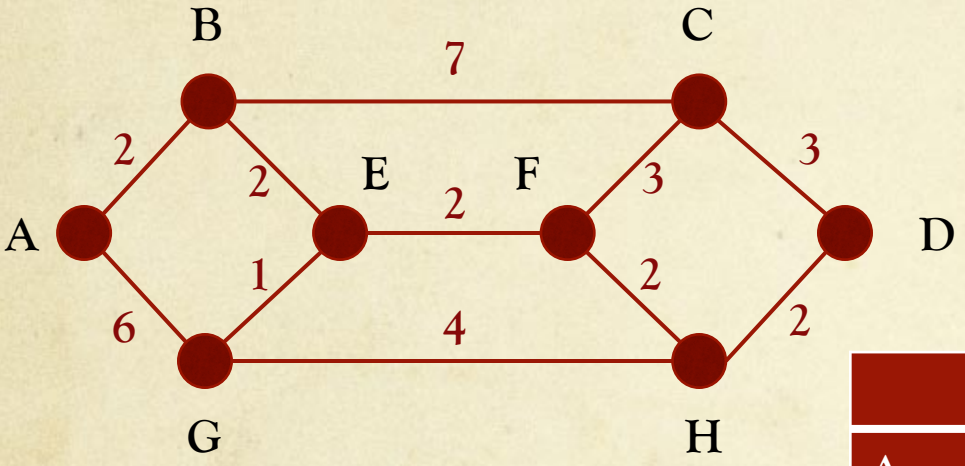
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B								

Djkstra's algorithm

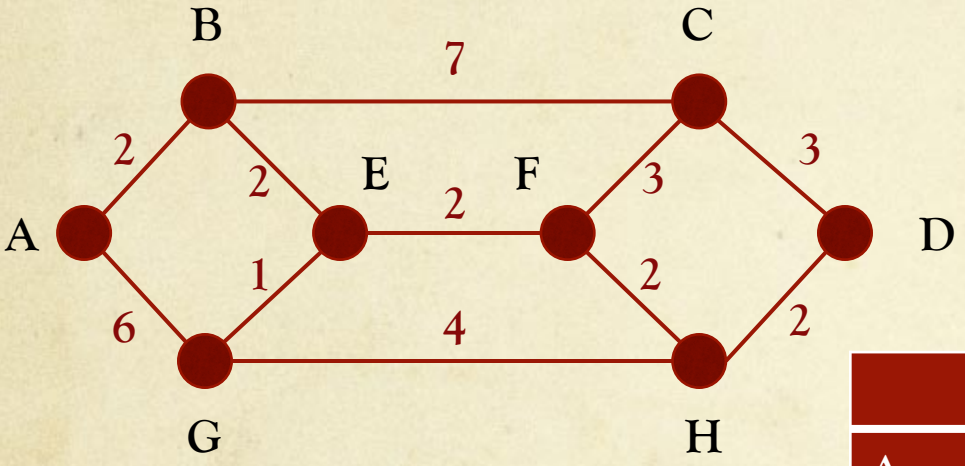
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B	0_A	2_A	9_B	∞_A	4_B	∞_A	6_A	∞_A
E								

Djkstra's algorithm

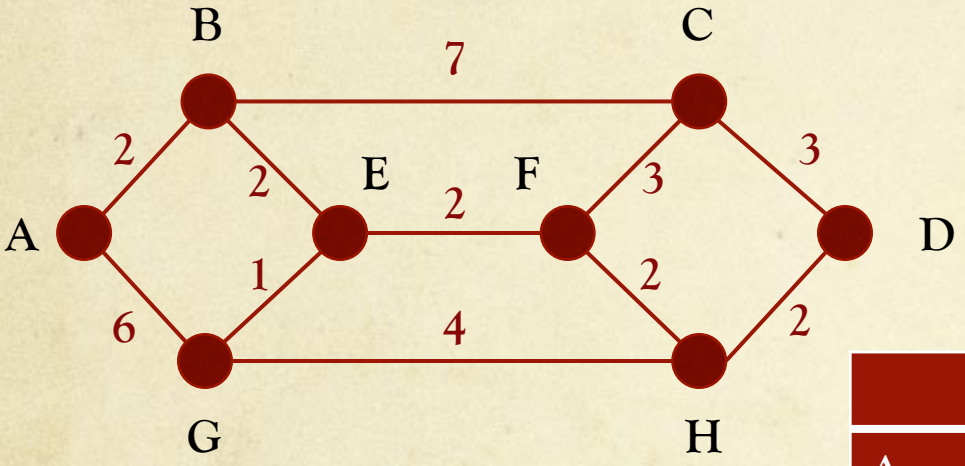
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B	0_A	2_A	9_B	∞_A	4_B	∞_A	6_A	∞_A
E	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	∞_A
G								

Djkstra's algorithm

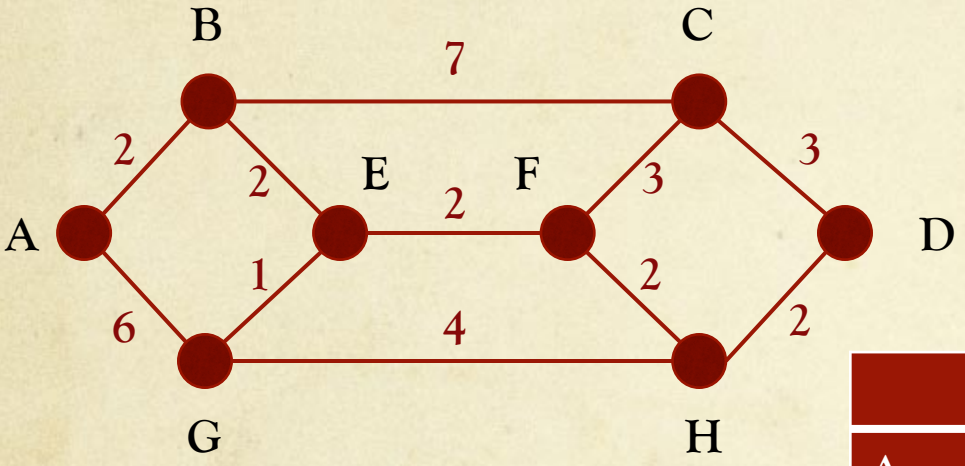
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B	0_A	2_A	9_B	∞_A	4_B	∞_A	6_A	∞_A
E	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	∞_A
G	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	9_G
F								

Djkstra's algorithm

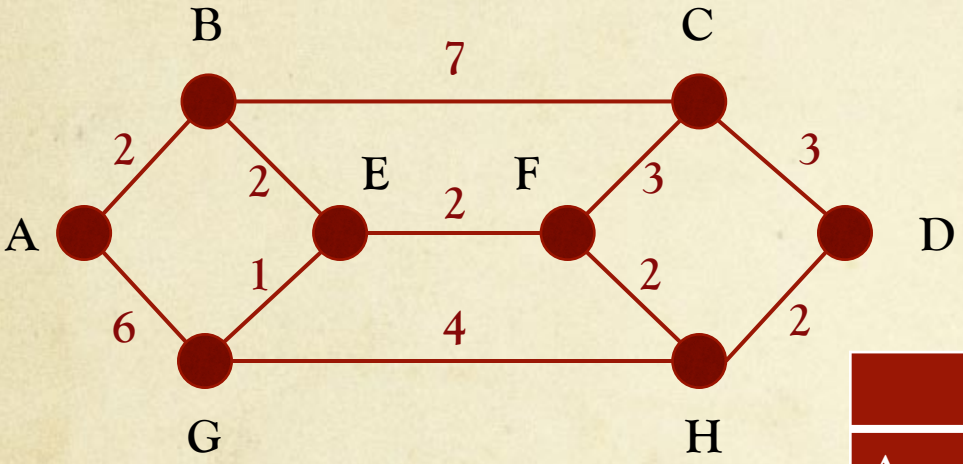
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B	0_A	2_A	9_B	∞_A	4_B	∞_A	6_A	∞_A
E	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	∞_A
G	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	9_G
F	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	8_F
H								

Djkstra's algorithm

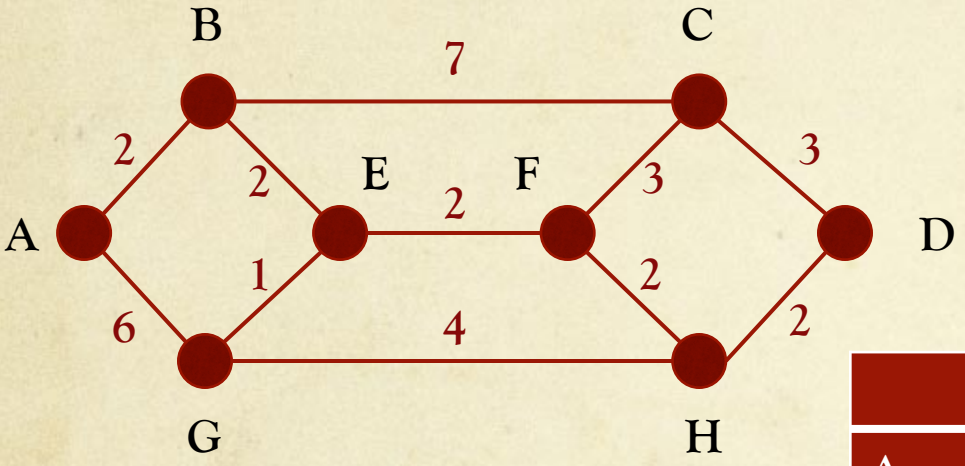
Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B	0_A	2_A	9_B	∞_A	4_B	∞_A	6_A	∞_A
E	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	∞_A
G	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	9_G
F	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	8_F
H	0_A	2_A	9_B	10_H	4_B	6_E	5_E	8_F
C								

Djkstra's algorithm

Finds *shortest* path b/w given *src* & *all* other nodes in n/w



	A	B	C	D	E	F	G	H
A	0_A	2_A	∞_A	∞_A	∞_A	∞_A	6_A	∞_A
B	0_A	2_A	9_B	∞_A	4_B	∞_A	6_A	∞_A
E	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	∞_A
G	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	9_G
F	0_A	2_A	9_B	∞_A	4_B	6_E	5_E	8_F
H	0_A	2_A	9_B	10_H	4_B	6_E	5_E	8_F
C	0_A	2_A	9_B	10_H	4_B	6_E	5_E	8_F
D								

Hierarchical routing

- As network size increases, routing tables get very *large*!
- Set up *hierarchy* among routers
  - Form regions
  - Each router knows details of its own region
  - Designated routers to get outside region
- Concept can be extended to *multiple* levels of hierarchy
  - Region, cluster, zone, …

When are routing decisions made?

- *Connectionless* service [*widely used*]
  - Routing decisions made independently for every packet
  - Subsequent packets may or may not take same route

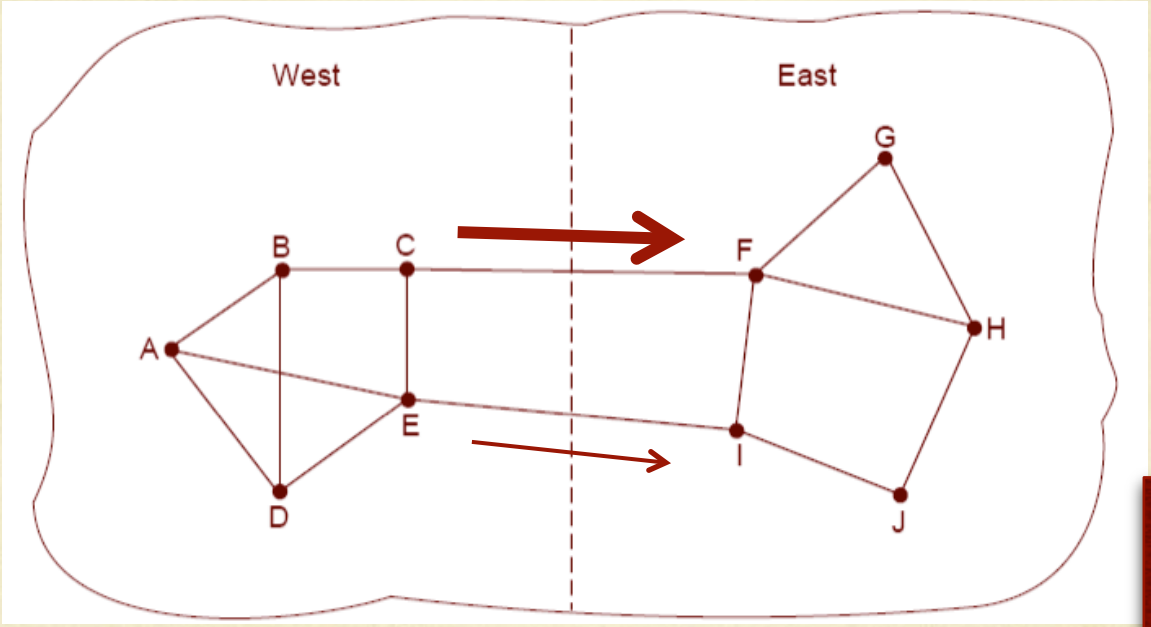
- *Connection-oriented* service
  - Routing decision made during connection establishment
  - Same route used for *all* packets on same connection

Congestion control

- Responsibility shared by *transport* & *network* layers
  - We discuss network layer aspects here
- Several approaches could be taken to handle congestion
  - Let routing algorithm handle it (*traffic-aware routing*)
  - Decrease traffic/load
  - Increase capacity
    - Add more resources to network (*provisioning*)
    - *More of a long-term solution*

Traffic-aware routing

- *Goal*: consider *load* & direct traffic away from *hotspots*
- Choose routes depending on traffic, not just topology



  - I.e., use EI for West-to-East traffic if CF is loaded
- Could lead to *oscillations* in routing tables

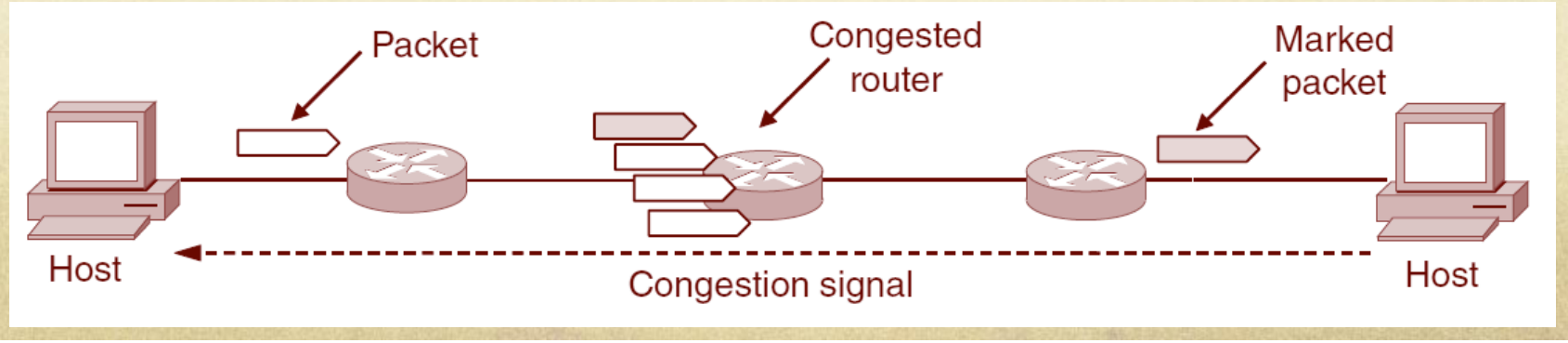
Just routing all traffic in another direction could cause that new direction to get overloaded!

Traffic-aware routing

- Solutions to oscillation problem
  - Change routes slowly, not all at once
    - Use external *traffic engineering* to decide on weights, etc.
  - Use multiple paths at once
    - I.e., *distribute* traffic over links (e.g., use both CF & EI)

Traffic throttling

- Routers *determine* when congestion is impending
- Routers *notify* senders of impending congestion
- Some approaches
  - Send *choke packet* to senders causing congestion
    - Sender required to reduce rate of transmission
  - Set flag in packets notifying impending congestion
    - Explicit congestion notification (*ECN*)



Admission control

- Useful in *connection-oriented* service
- Refuse new *connection* (virtual circuit) if network can handle it without being congested
- How do you know if new connection will cause congestion?
  - Need some way to represent traffic state
  - Approaches include *leaky bucket*

And if nothing else works...

- ...routers resort to *load shedding*
- I.e., routers simply *drop* packets when inundated
- *Which* packets should be dropped?
  - Depends...
  - File transfer → *old* packet worth more than *new* one
  - Real-time streaming → *new* packet worth more than *old* one

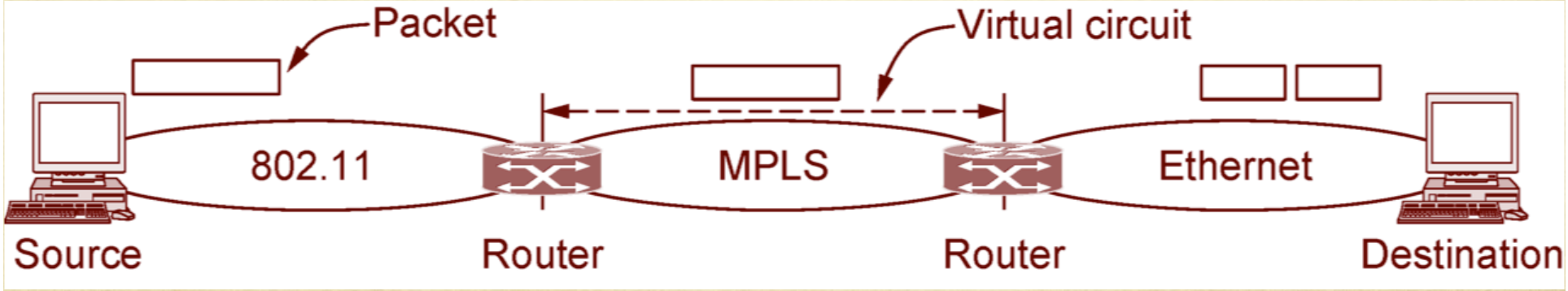
Internetworking

- Several routing algorithms considered so far assume
  - *Homogeneous* network
  - All nodes use common protocols in all layers

- *Not* true in reality
  - Networks may differ in several ways...

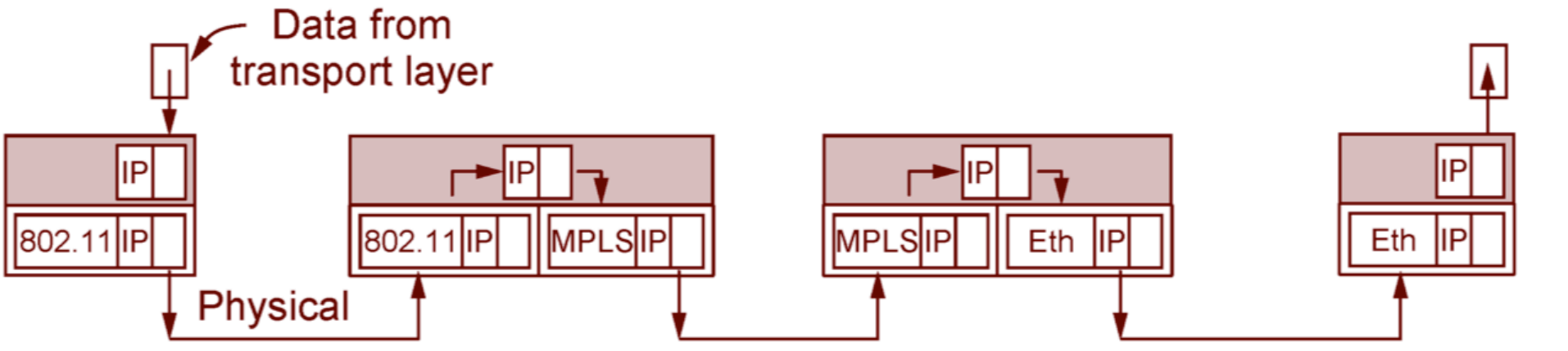
Internetworking

- In what ways do networks differ?
  - *Services*: connectionless versus connection oriented
  - *Addressing*: different sizes, flat or hierarchical
  - *Packet size*: different maximum sizes
  - *Quality of Service*: priority, ...



One solution – Cerf & Kahn

- *Common* network layer above *different* physical networks
  - E.g., Internet Protocol (*IP*): foundation of the *Internet*

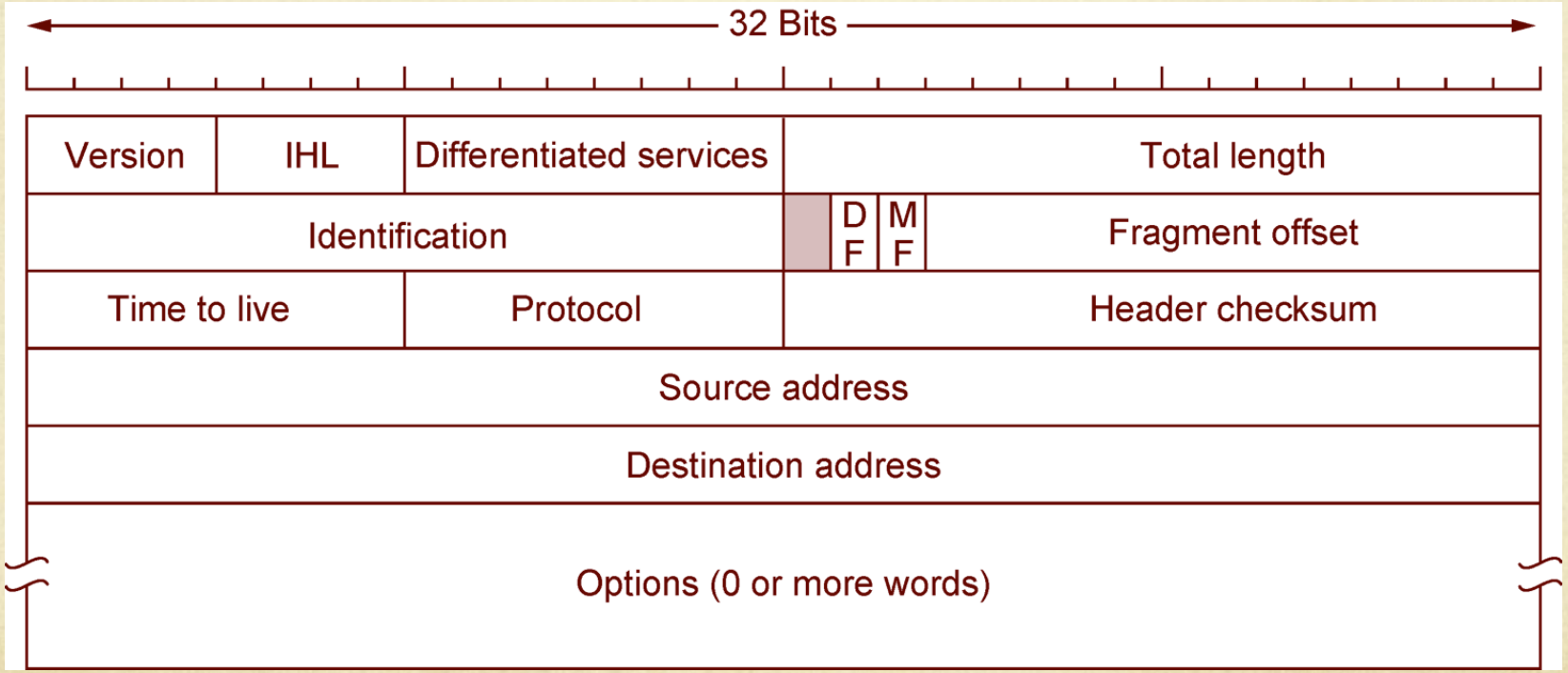


Internet Protocol (*IP*)

- Common network layer protocol
  - Glue that holds *Internet* together
- *Connectionless*, *best effort* service
  - No reliability guarantees
  - No ordering guarantees
- IP versions
  - Version 4 (*IPv4*)
  - Version 6 (*IPv6*)

IP version 4 (*IPv4*)

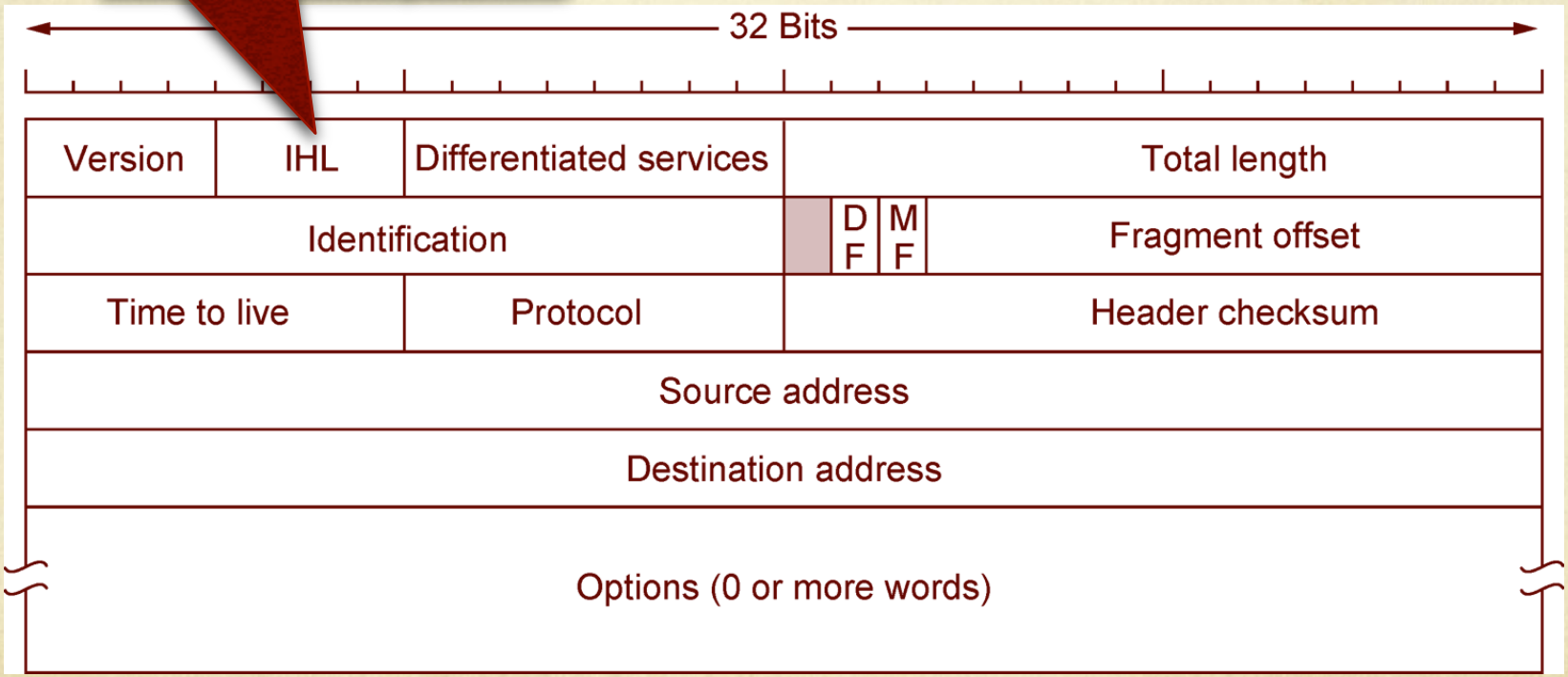
- IPv4 *header* included in all network layer packets



IP version 4 (*IPv4*)

Header length in 32-bit words – min 5 words (20 bytes), max 15 words (60 bytes)

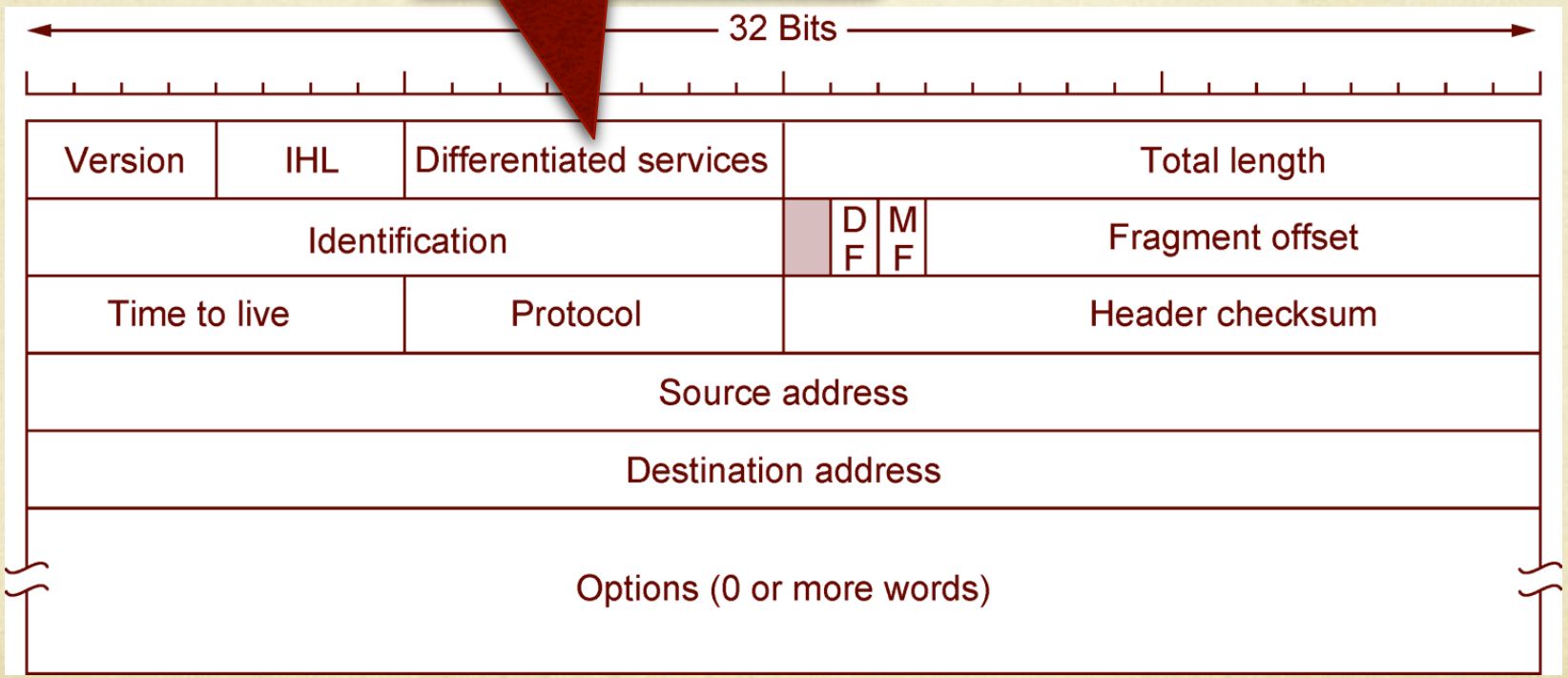
- I ded in all network layer packets



IP version 4 (*IPv4*)

- IPv4 *header* etwork layer packets

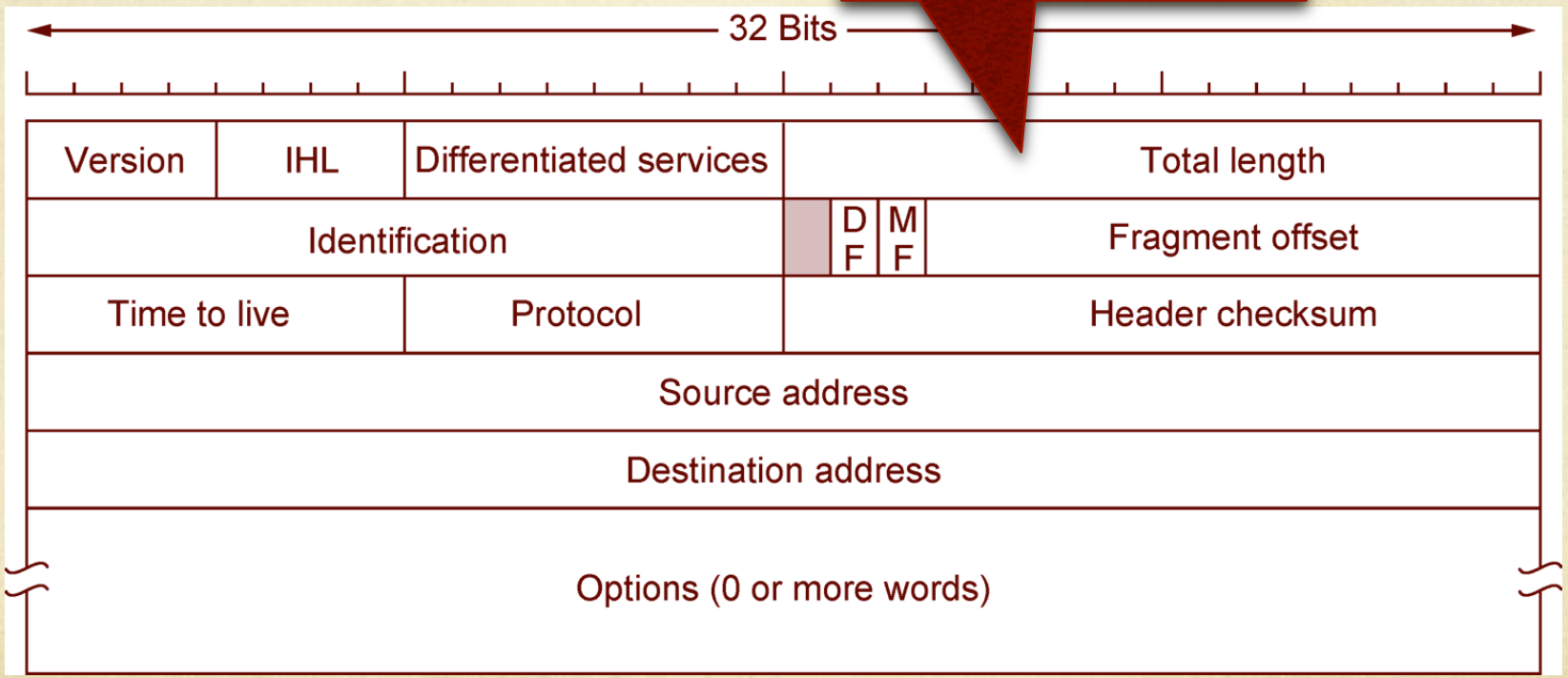
Current usage: Service class & explicit congestion notification bits



IP version 4 (*IPv4*)

- IPv4 *header* included in all ckets

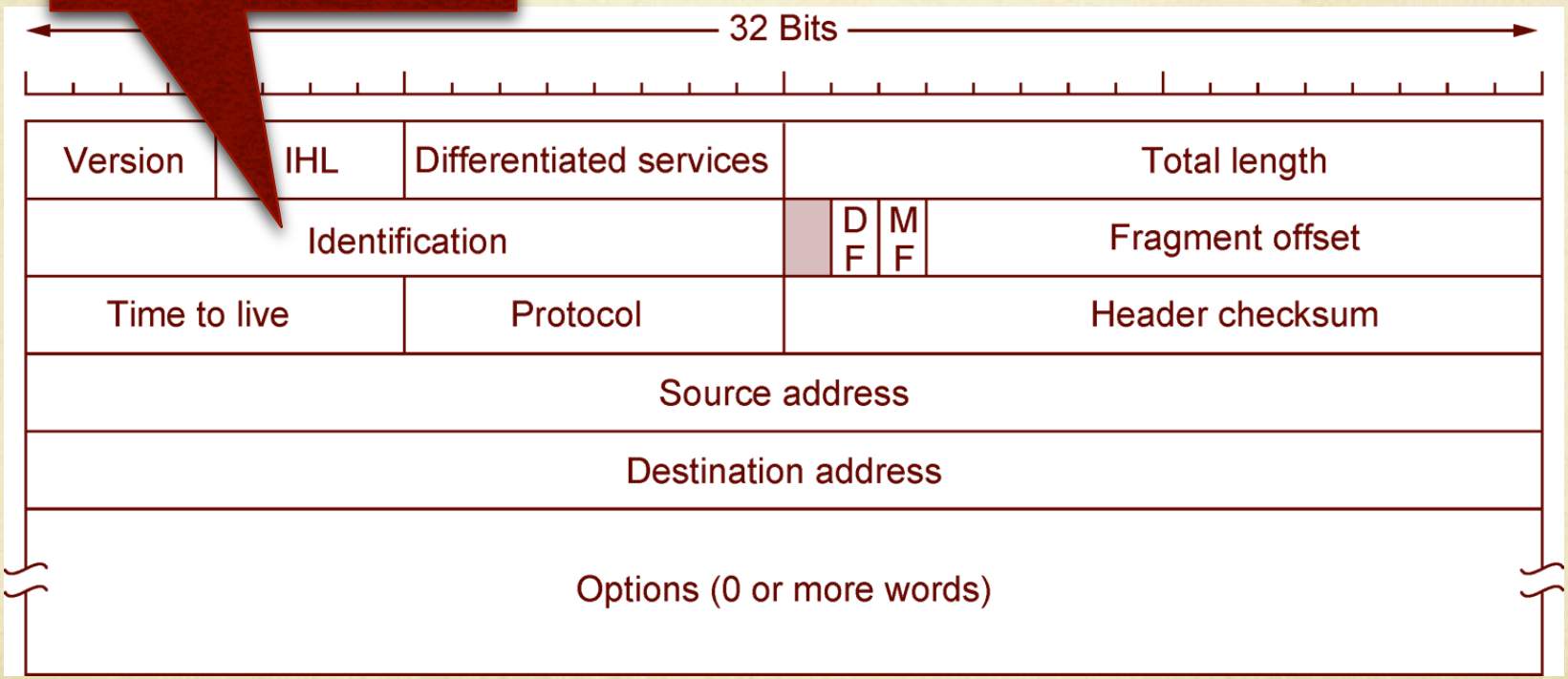
Total length of entire packet (header + data) – max 64KB



IP version 4 (*IPv4*)

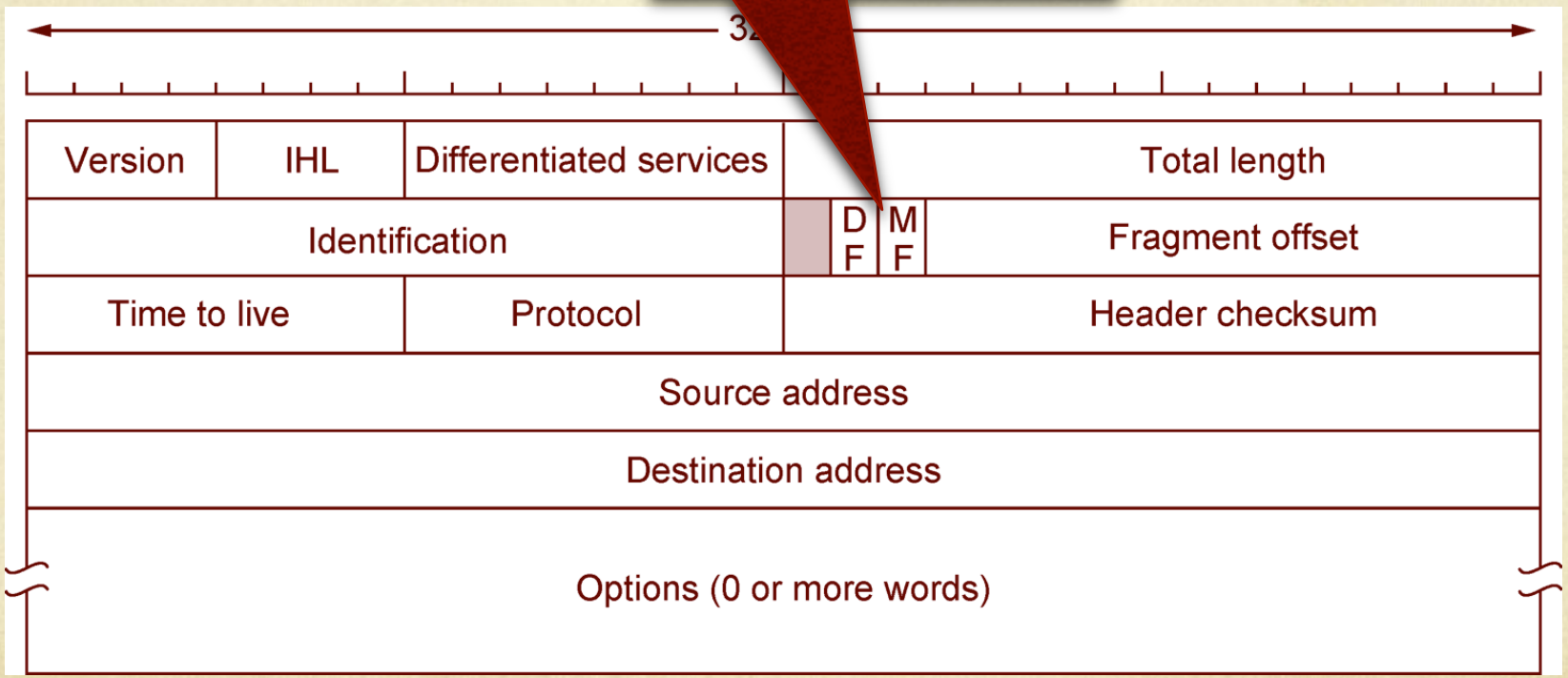
- uded in all network layer packets

Packet identification when fragmentation is used



IP version 4 (*IPv4*)

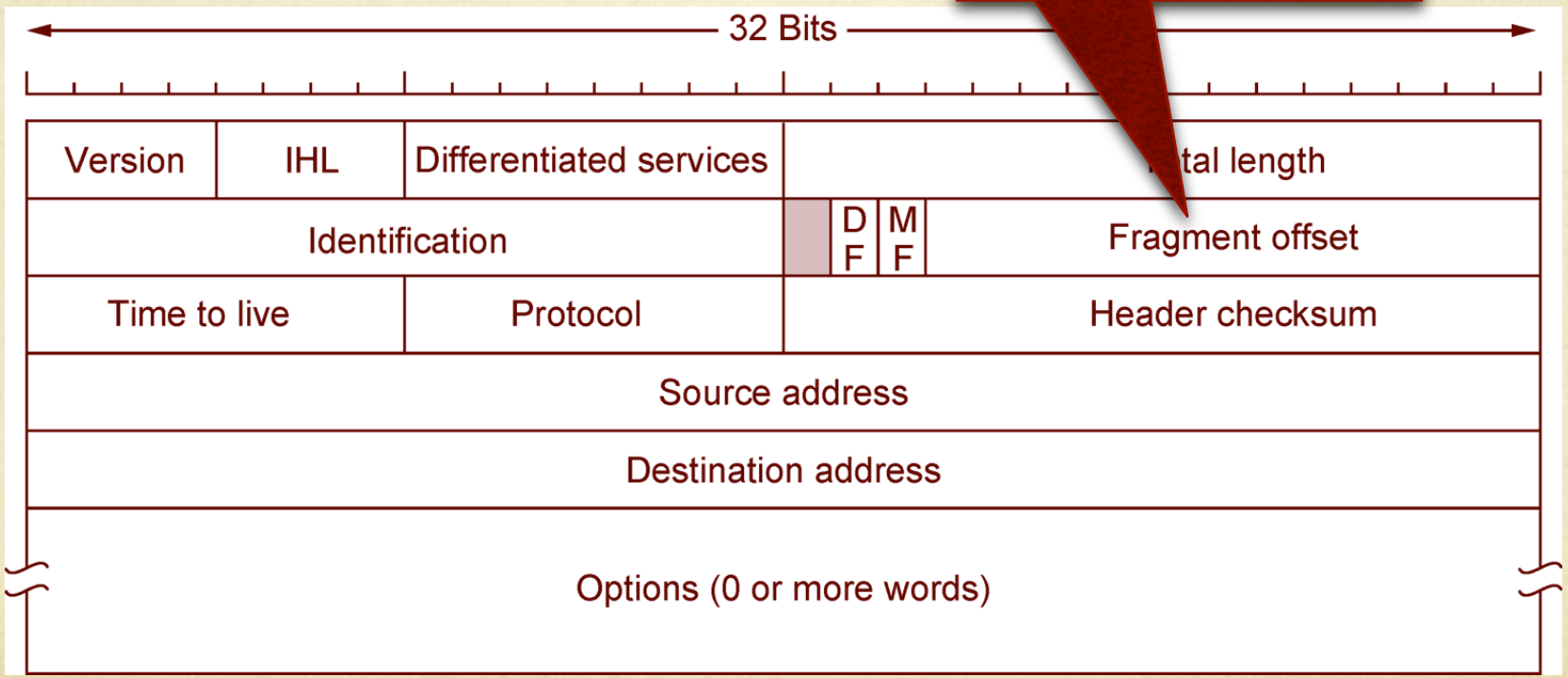
- IPv4 *header* included[illegible]ayer packets



IP version 4 (*IPv4*)

- IPv4 *header* included in all netw s

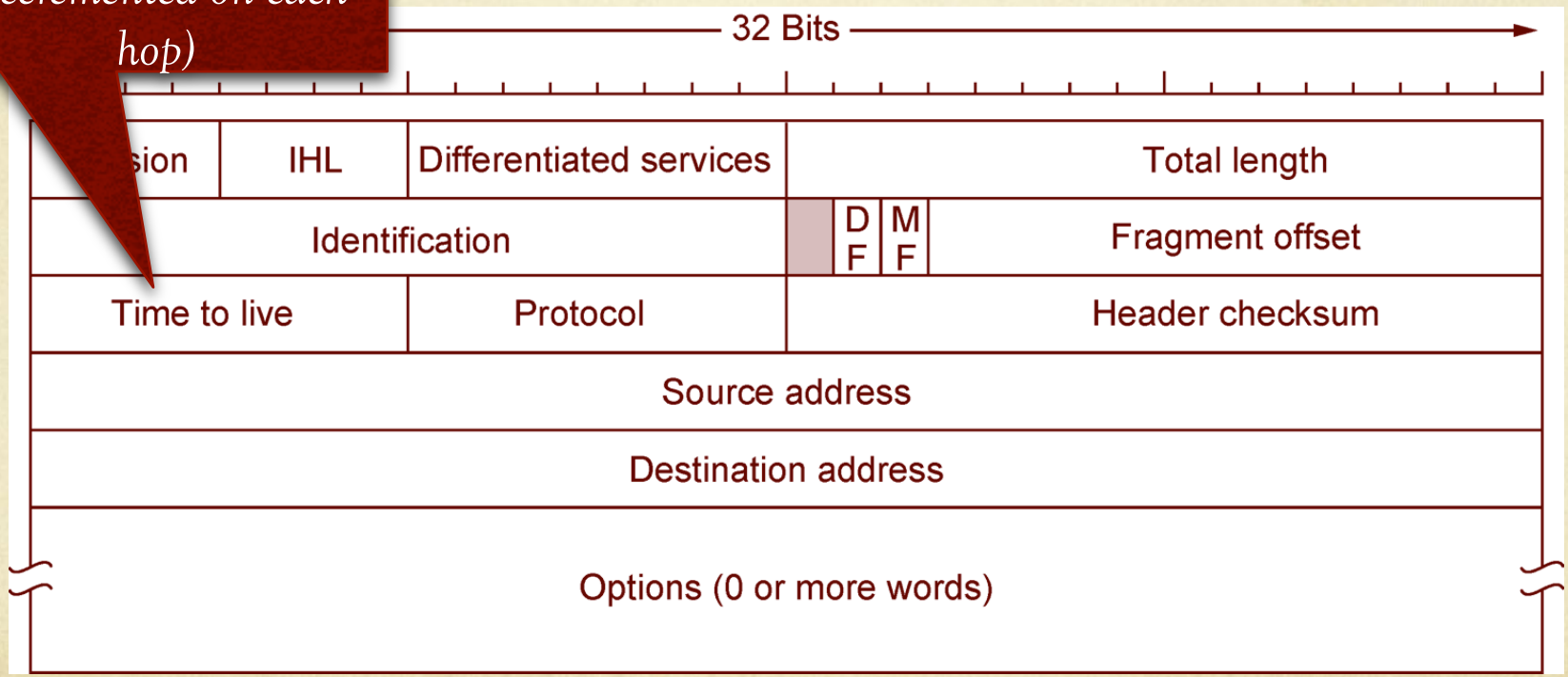
Fragment offset within packet (max 8K fragments per packet)



IP version 4 (*IPv4*)

Counter used to limit packet lifetime (decremented on each hop)

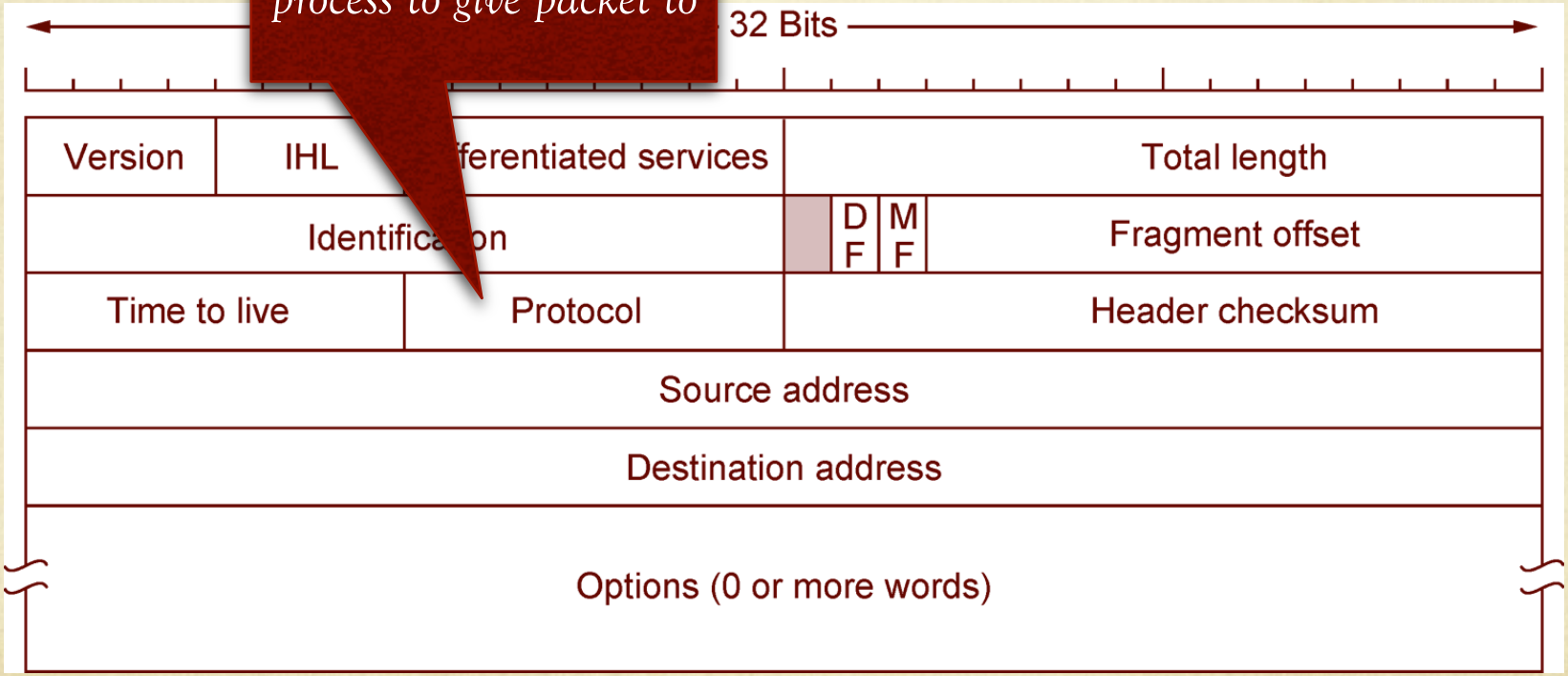
r included in all network layer packets



IP version 4 (*IPv4*)

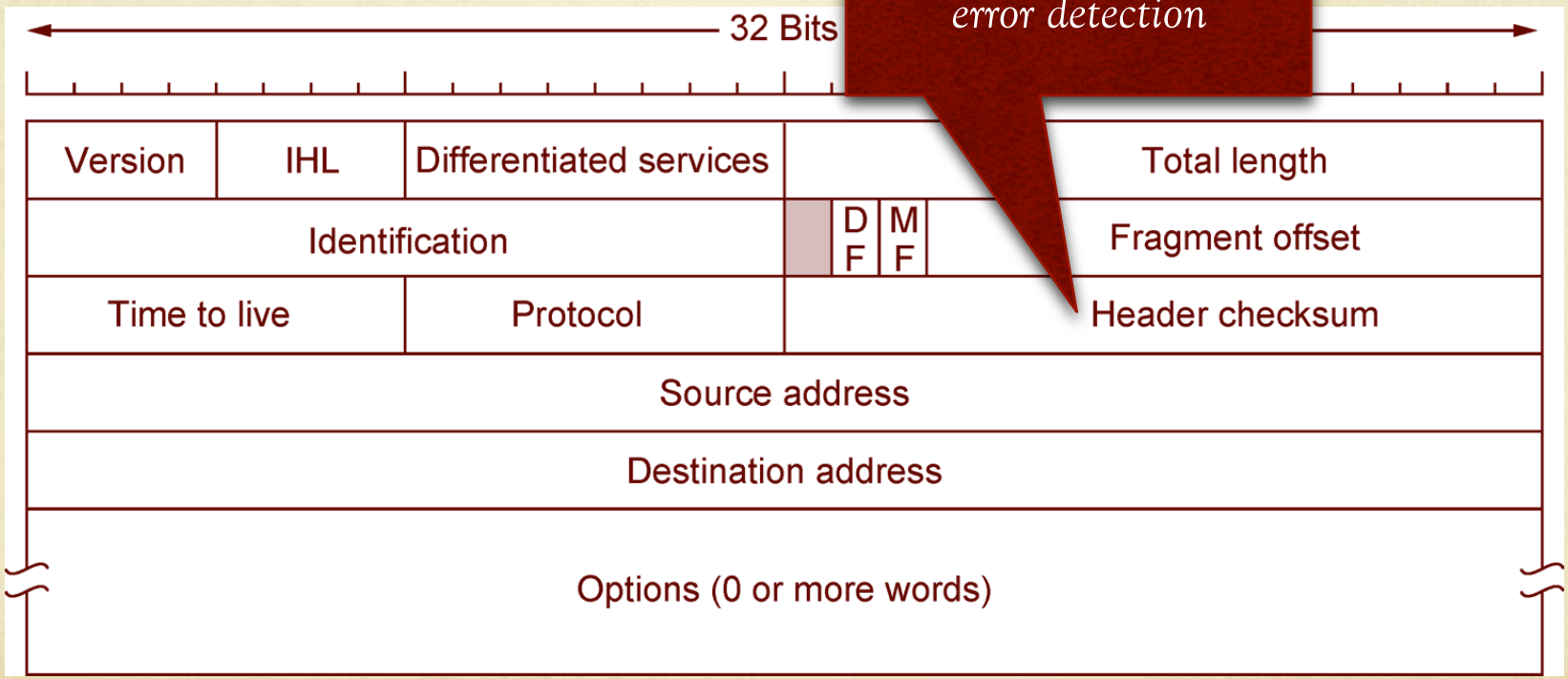
- IPv4 h… …n all network layer packets

Which transport layer process to give packet to



IP version 4 (*IPv4*)

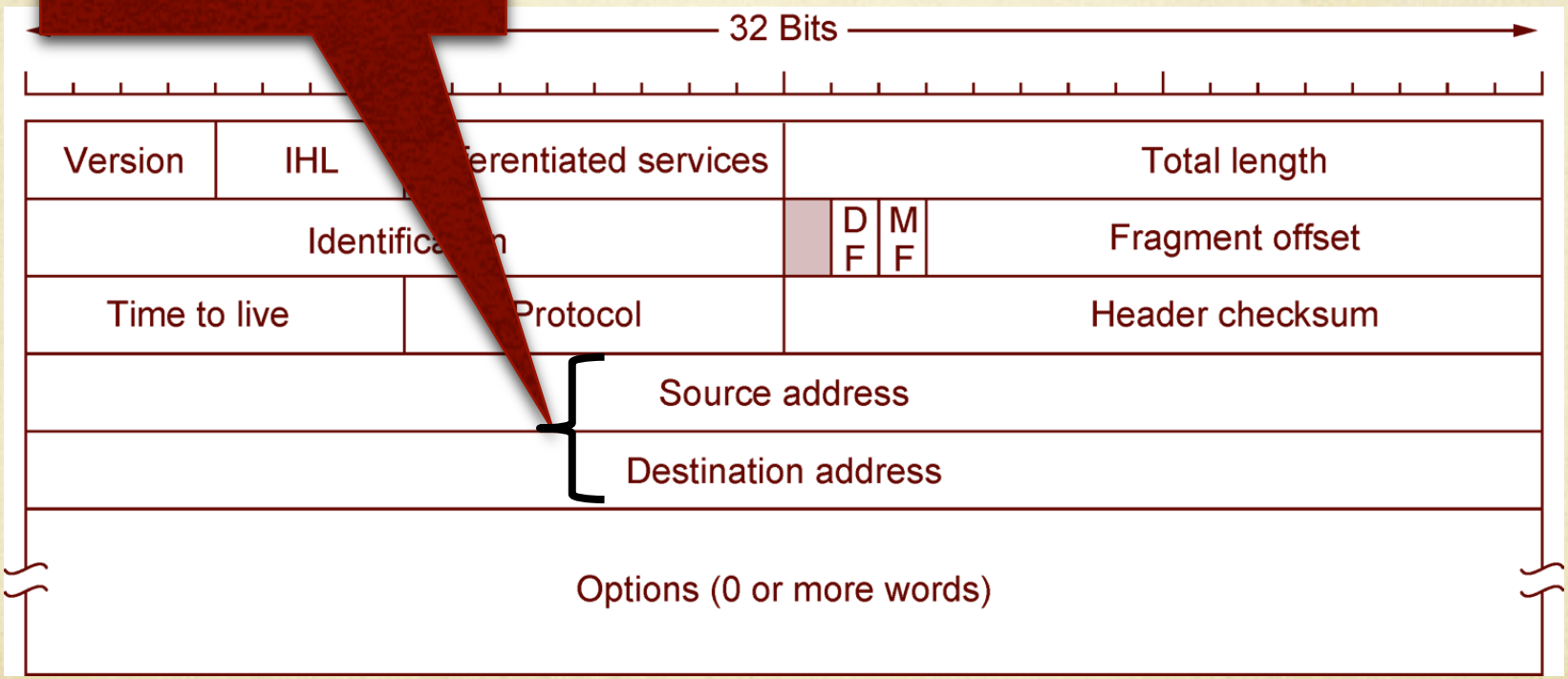
- IPv4 *header* included in all [illegible]ckets



IP version 4 (*IPv4*)

- luded in all network layer packets

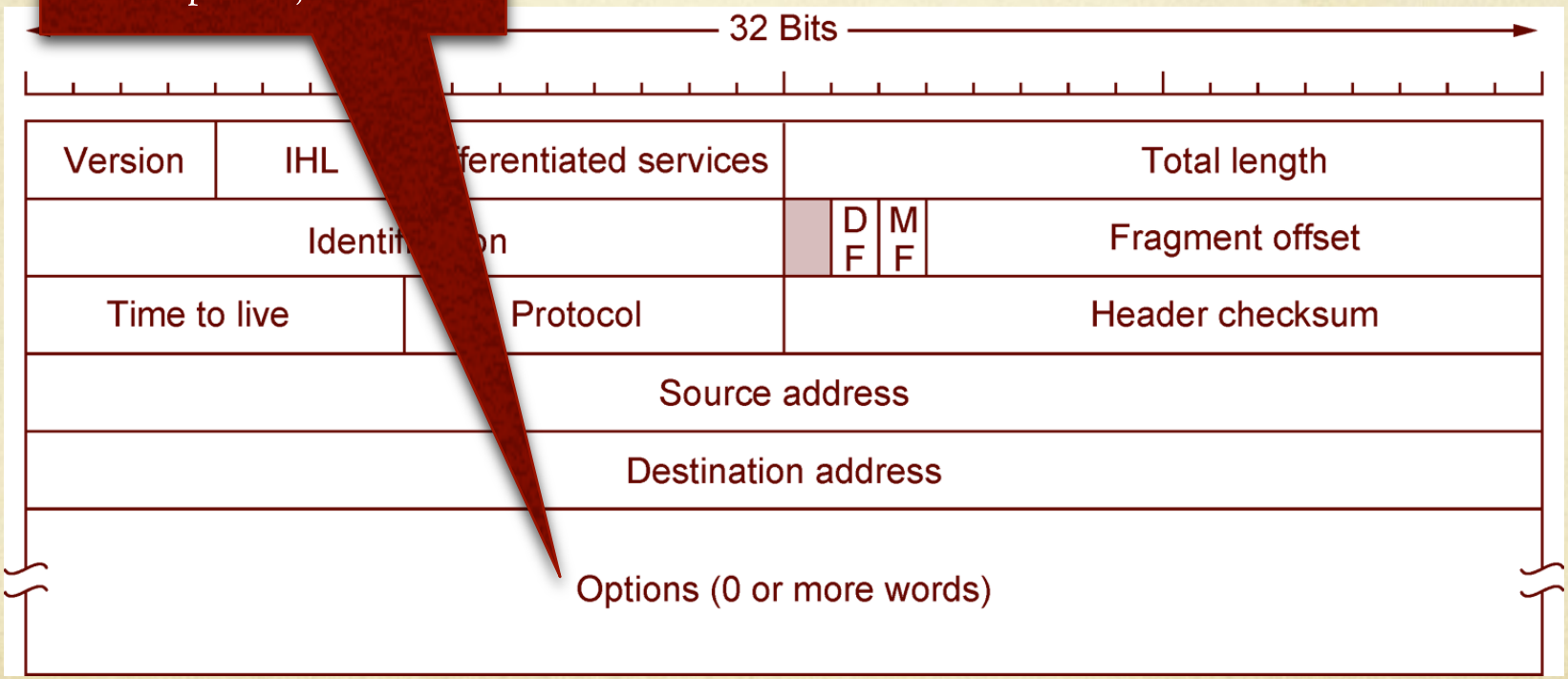
Source & destination endpoint addresses



IP version 4 (*IPv4*)

- luded in all network layer packets

Specify options – security options, routing options, etc.

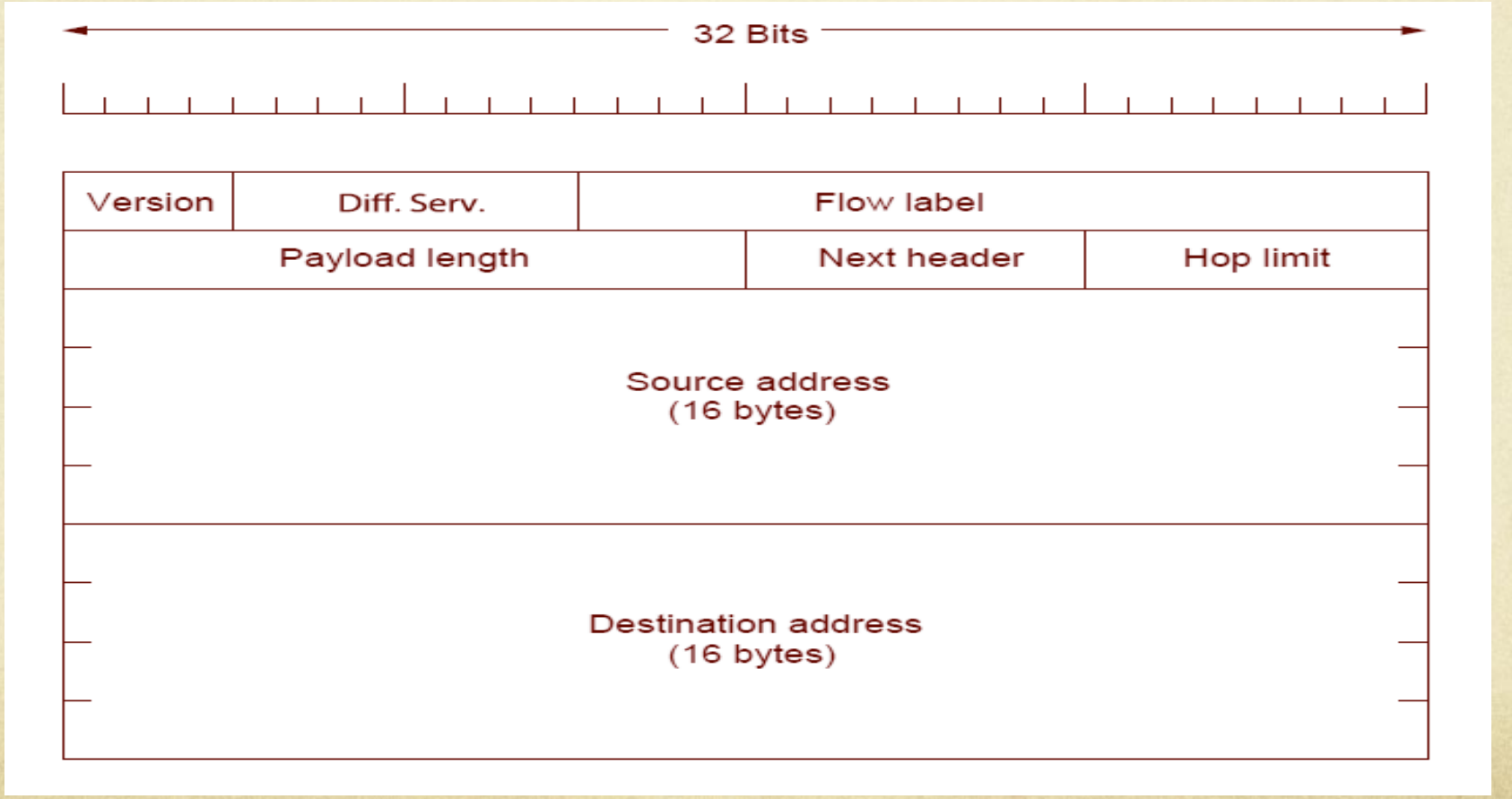


Keeping up with growth

- IPv4 can't keep up with *growing size* of the Internet
- New version → IPv6
  - Supports *128-bit* address
  - Has *simpler* header → reduces processing overhead
  - Better support for *options*
- Has not been easy to deploy on Internet-scale
  - Does not internetwork easily with IPv4!

IP version 6 (*IPv6*)

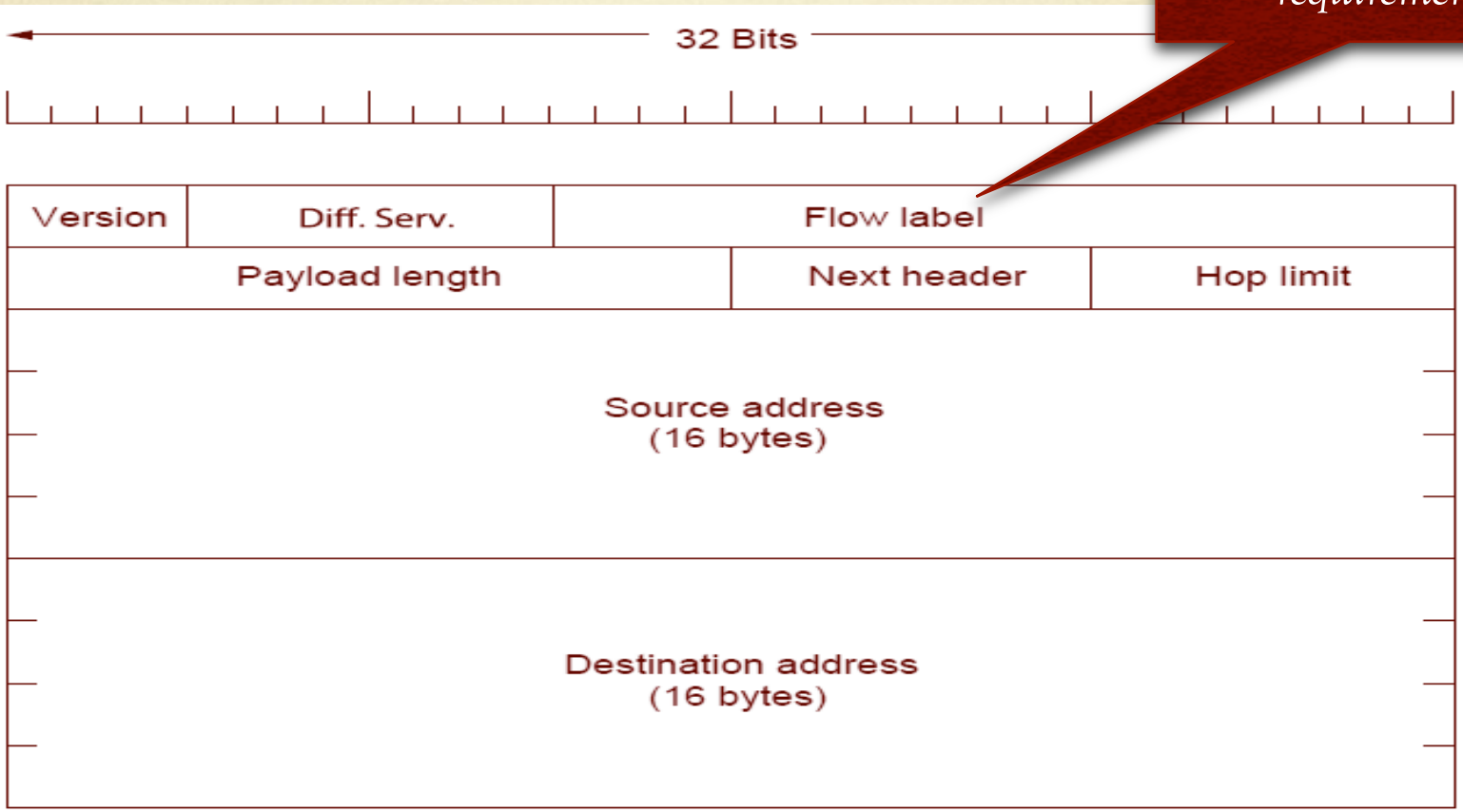
- *Required* header



IP version 6 (*IPv6*)

- *Required* header

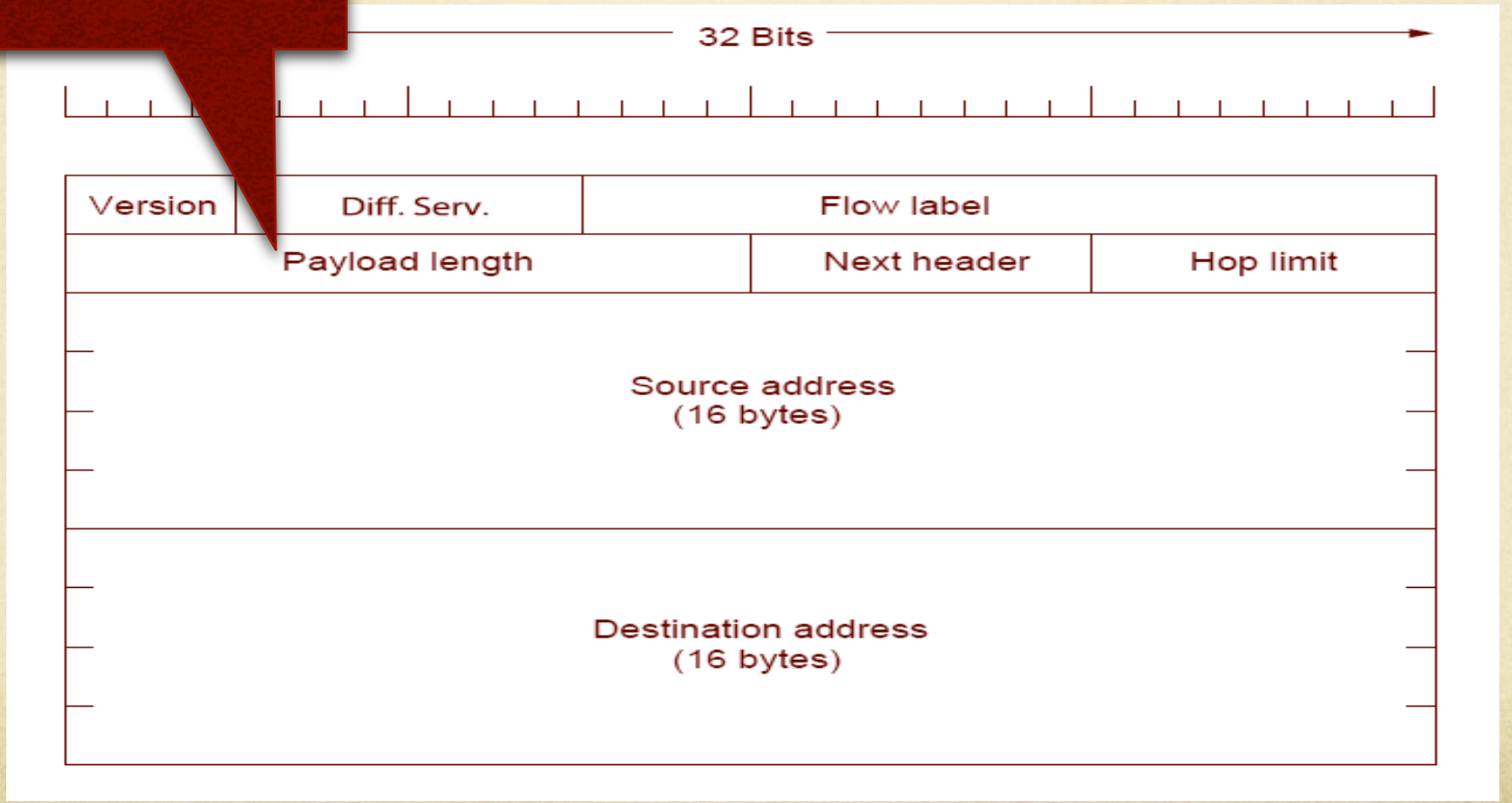
Way to make groups of packets with same requirements



IP version 6 (*IPv6*)

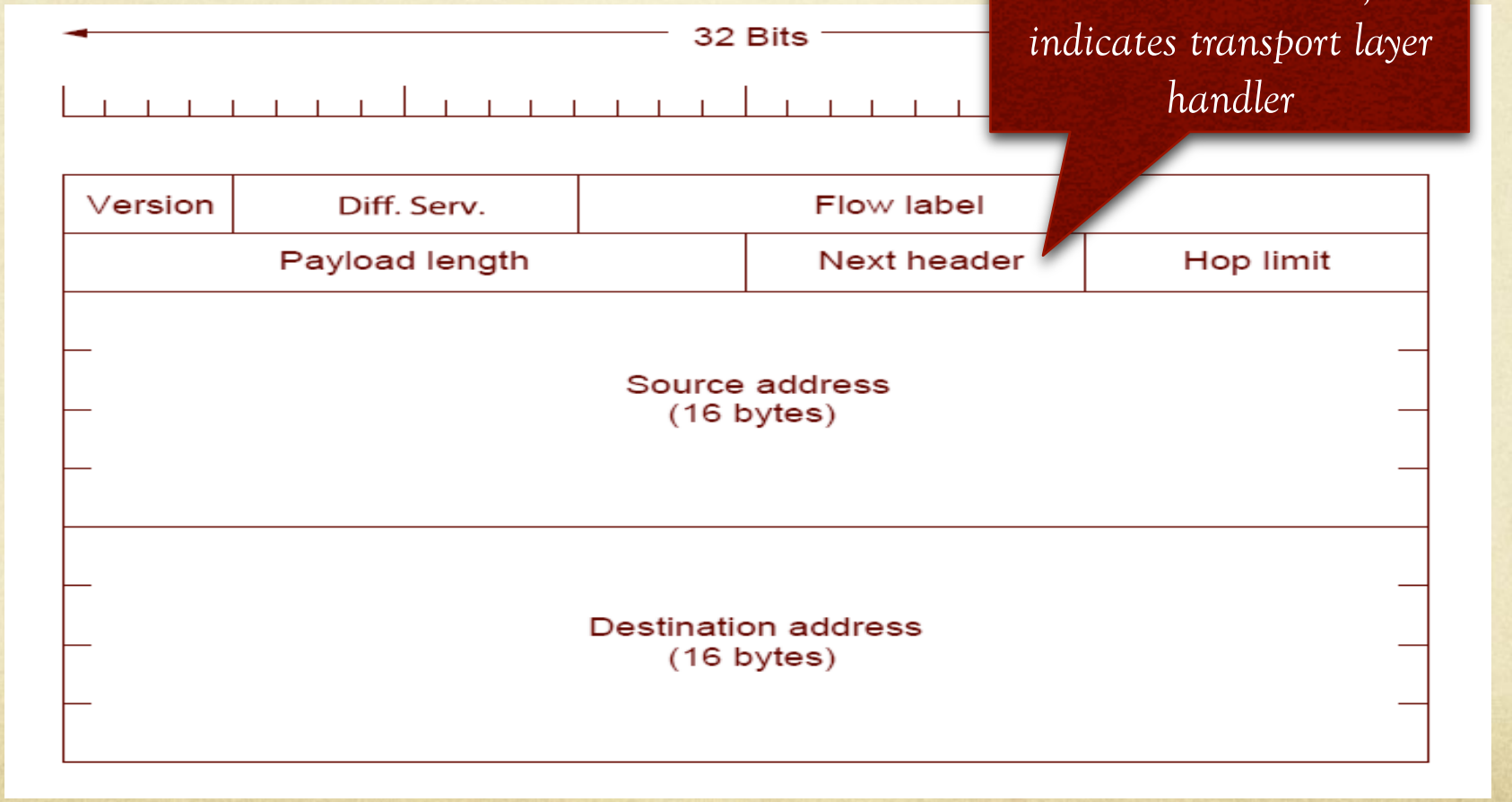
eader

Length of payload **after** *required header*



IP version 6 (*IPv6*)

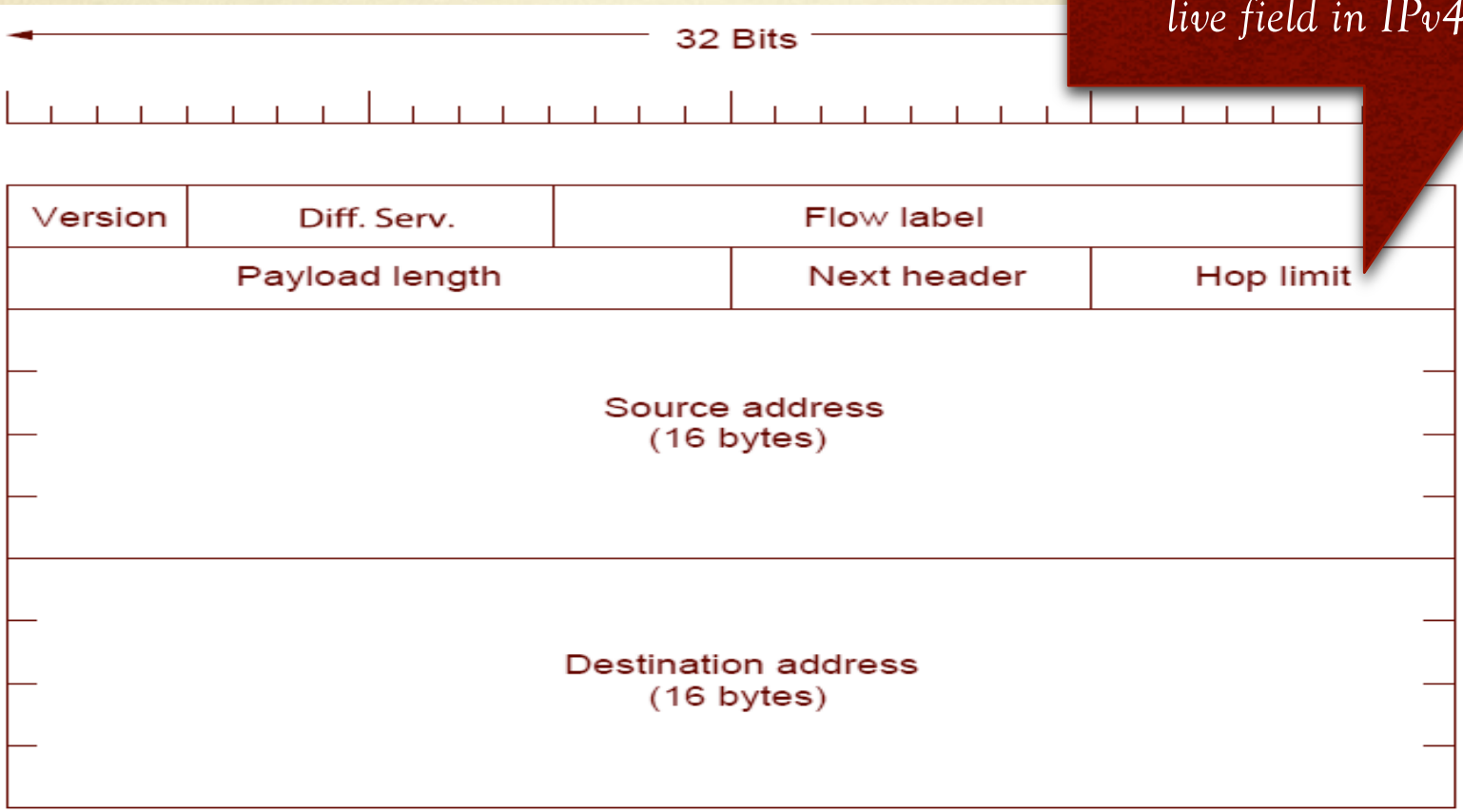
- *Required* header



IP version 6 (*IPv6*)

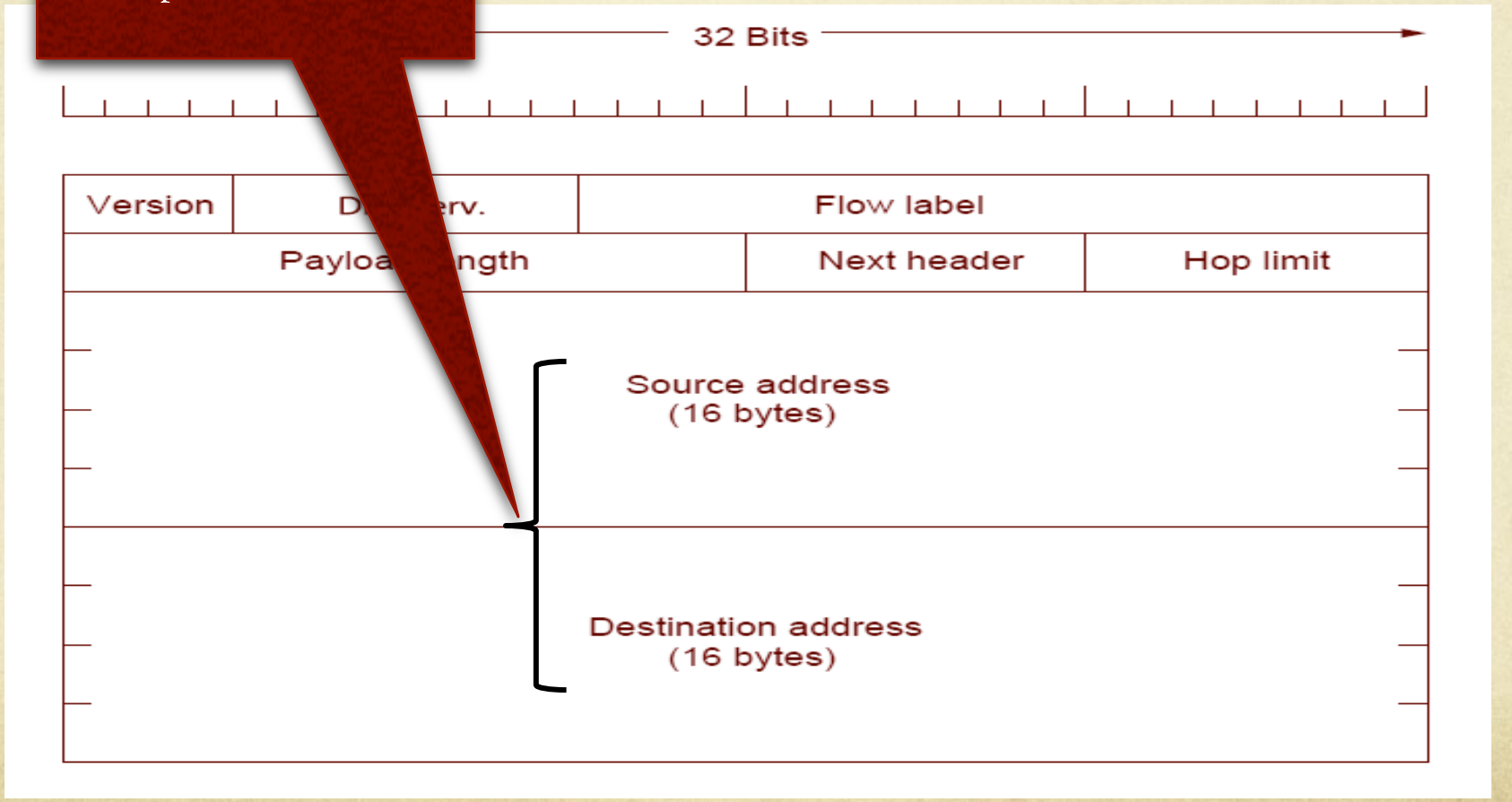
- *Required* header

Limits packet lifetime (same leaning as time to live field in IPv4)



IP version 6 (*IPv6*)

- *Source & destination endpoint addresses*



IP version 6 (*IPv6*)

- Extension headers handle other functionality

Extension header	Description
Hop-by-hop options	Miscellaneous information for routers
Destination options	Additional information for the destination
Routing	Loose list of routers to visit
Fragmentation	Management of datagram fragments
Authentication	Verification of the sender's identity
Encrypted security payload	Information about the encrypted contents