EEL-4736/5737 Principles of Computer System Design

Lecture Slides 16

Textbook Chapter 7

Network layers

Introduction

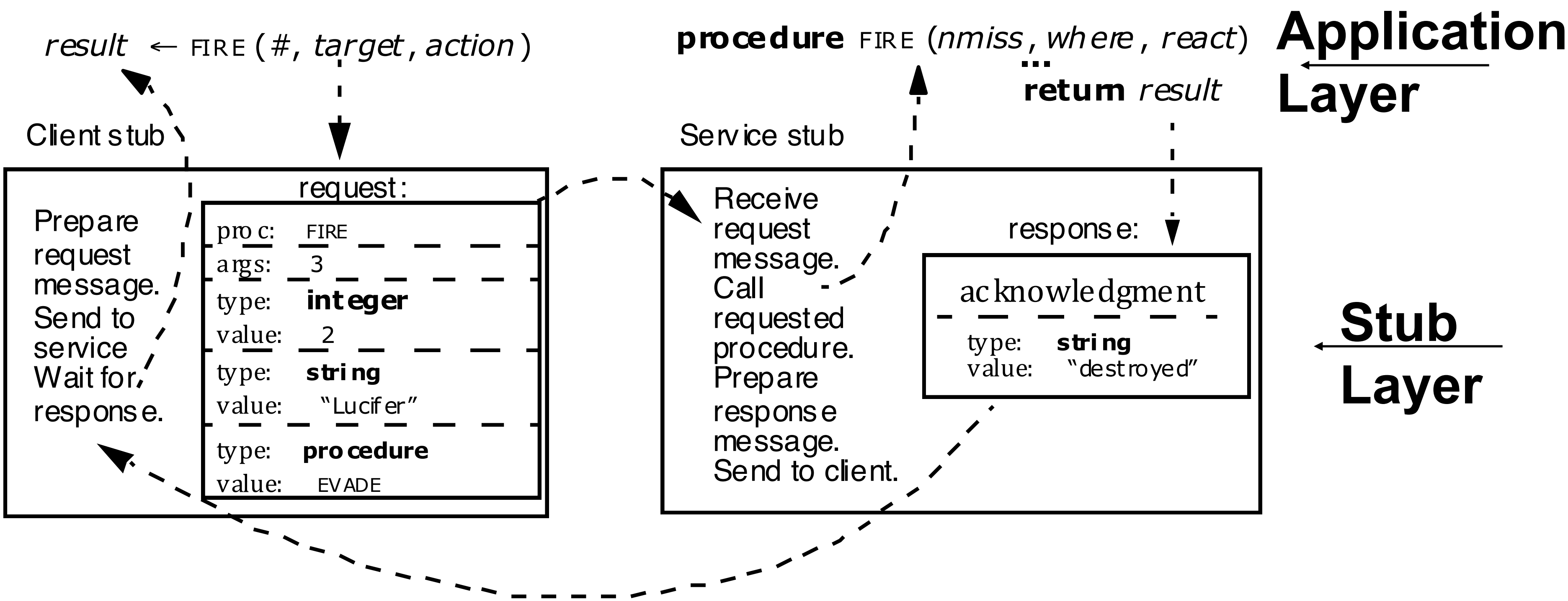
- How to deal with the properties of networks
- 1. Networks encounter a vast range of
  - Data rates
  - Propagation, transmission, queuing, and processing delays.
  - Loads
  - Numbers of users
- 2. Networks traverse hostile environments
  - Noise damages data
  - Links stop working
- 3. Best-effort networks have
  - Variable delays
  - Variable transmission rates
  - Discarded packets
  - Duplicate packets
  - Maximum packet length
  - Reordered delivery

Divide-and-conquer

- Rather than addressing all these problems at once, networks apply the *layering* design principle extensively
- Each layer follows a *protocol*, and responsibility to address different challenges we have described are assigned to appropriate layer(s)

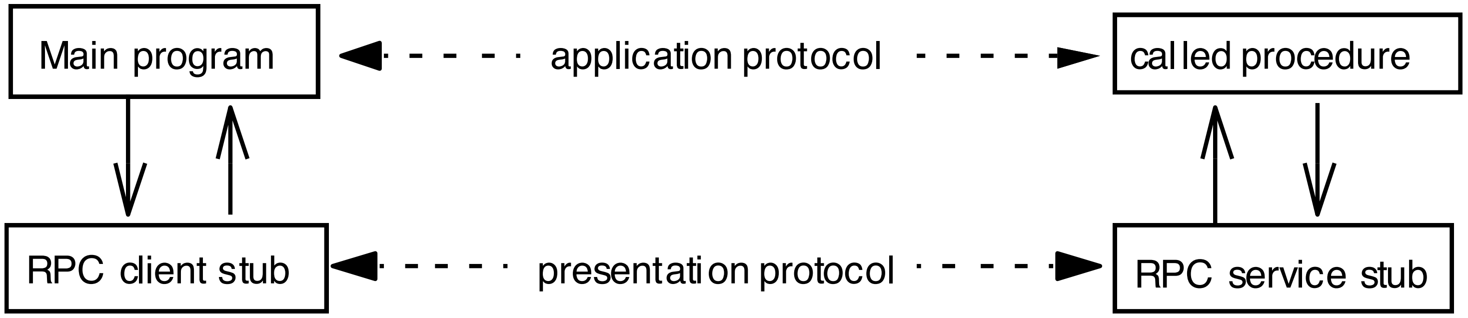
Protocols

- Specify all possible kinds of interactions between entities in the system
  - Examples: RPC, Network File System (NFS)



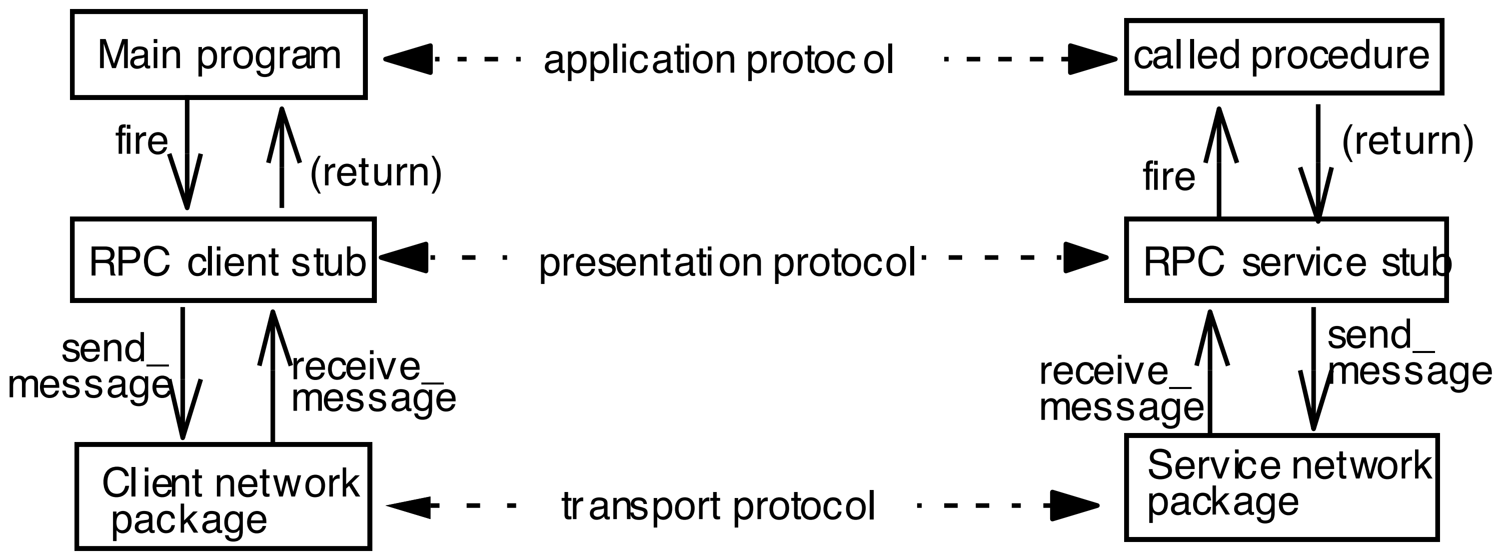
Layering protocols

- Application protocol: not concerned with how RPC arguments are marshaled, timeout/retry
  - It is concerned with what each procedure implements
  - E.g. NFS READs vs REMOVEs, idempotency, etc
- Presentation protocol: not concerned with what the procedure implements



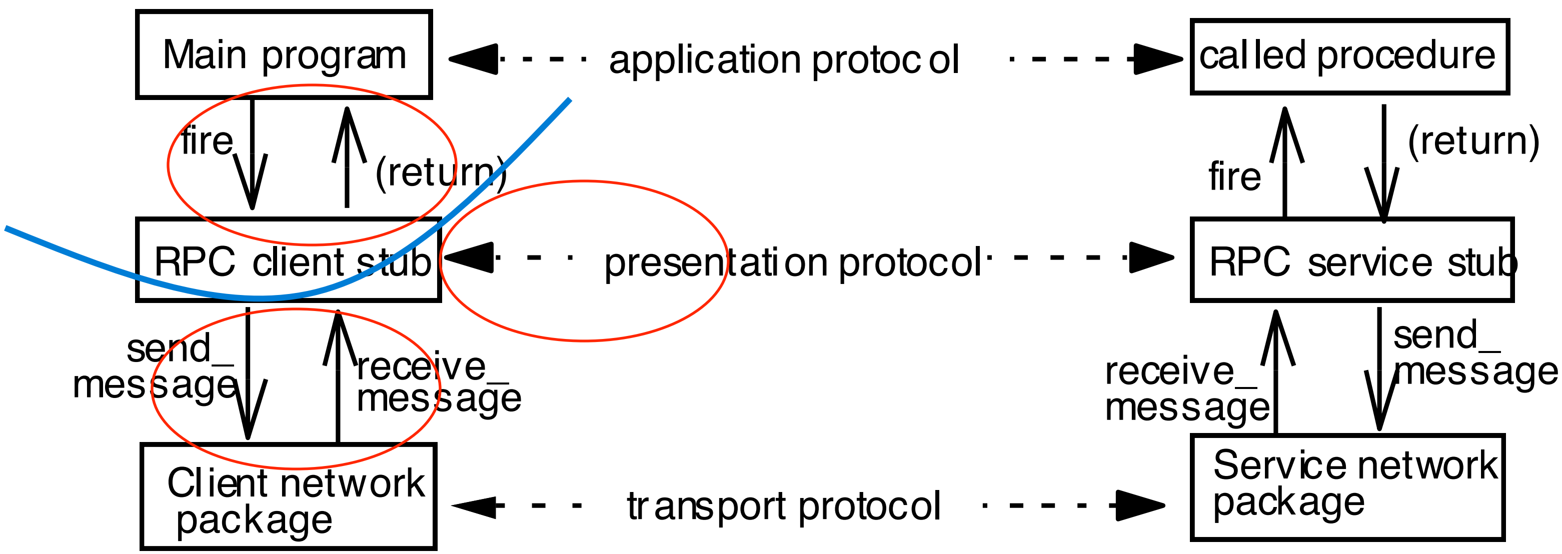
Layering protocols

- Client/service stub protocol needs the primitives to send/receive messages over the network
  - Transport protocol



Layering protocols

- Modules interact with three interfaces
  – Upper layer and lower layer
  – Same layer, on remote end
- Module *hides* interfaces from upper layer



Networking Layers

- Protocols can be layered in many different ways, and different networks will implement layers differently
- One well-known reference model (OSI, Open System Interconnect) establishes seven layers
  - Physical, data link, network, transport, session, presentation, application
- We will focus our discussion on a simpler model that helps highlight key system design challenges and principles

3-layer model

- From bottom up:
  - Data link layer:
    - Responsible for moving data from one point to another through a link
  - Network layer:
    - Responsible for forwarding data through intermediaries between source and destination
  - End-to-end layer:
    - Everything else at the endpoints necessary to support an application’s functionality

3-layer model

- From top down:
  - End-to-end layer:
    - Applications need to transmit *messages* or establish *streams* with other endpoints
    - Enforces communication semantics
      - E.g. RPC, at-least-once
    - Requests for messages/streams, breaks them into smaller *segments;* call upon lower layer to send each segment
    - Copes with lost or duplicate segments; places segments in order
  - Applications can be thought as 4th layer, and end-to-end thought as multiple layers, but interfaces are not always clear-cut
  - Example: NFS, RPC over TCP or UDP

3-layer model

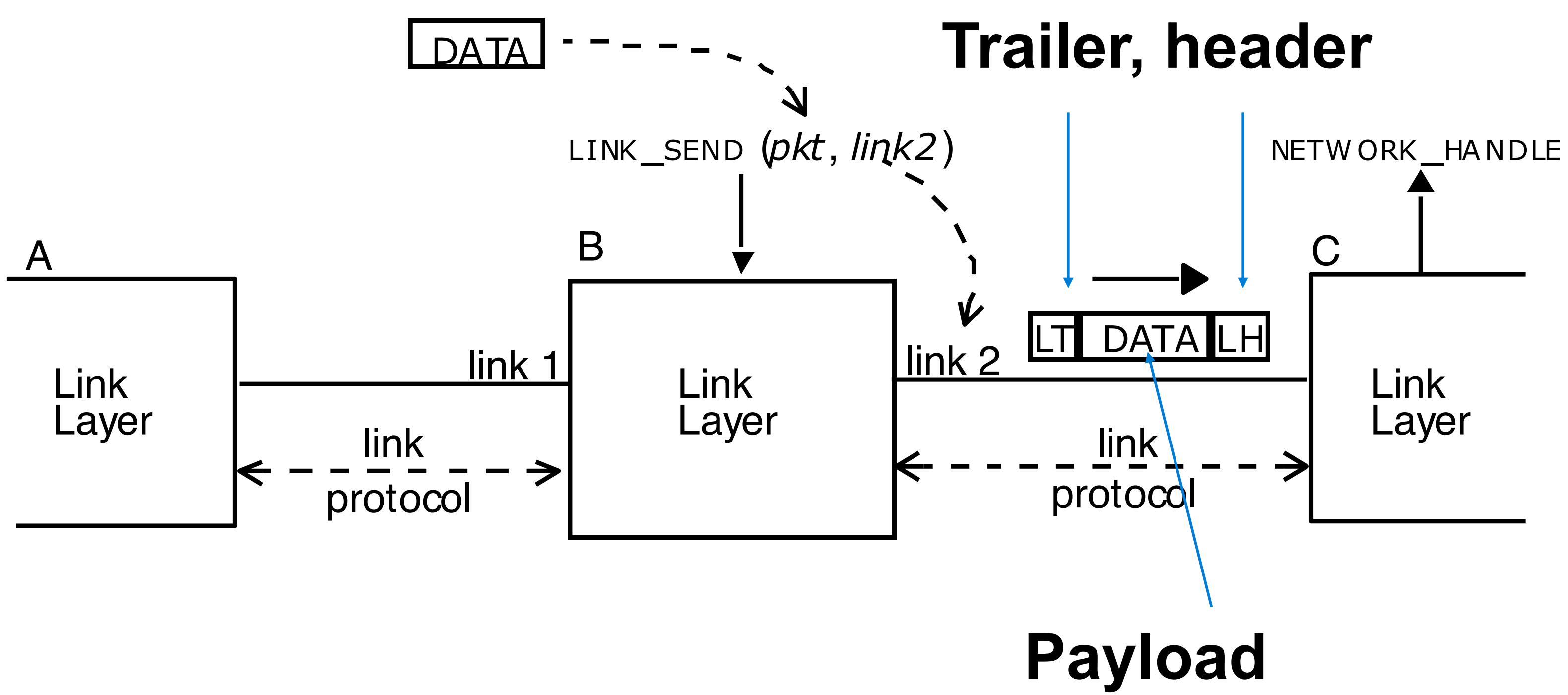
- From top down:
  - Network layer
    - Accepts *segments* from the end-to-end layer
    - Creates packets and transmits them over the network, choosing which links to follow, from source to destination
    - Example: IP
  - Link layer
    - Accepts *packets* from the network layer
    - Creates and transmits *frames* along a single link (between forwarders and/or network attachment devices)
    - Example: Ethernet

Link Layer

- Handles linking devices in a network segment
  – Node to switch, switch to switch
  – Move bits of a packet through a link, hiding physical mechanisms involved
- Simplified interface:
  – LINK_SEND (data_buffer, link_identifier)
    - Data_buffer: reference to memory containing data to be transmitted
    - Link_identifier: names one out of multiple possible links a device is connected to

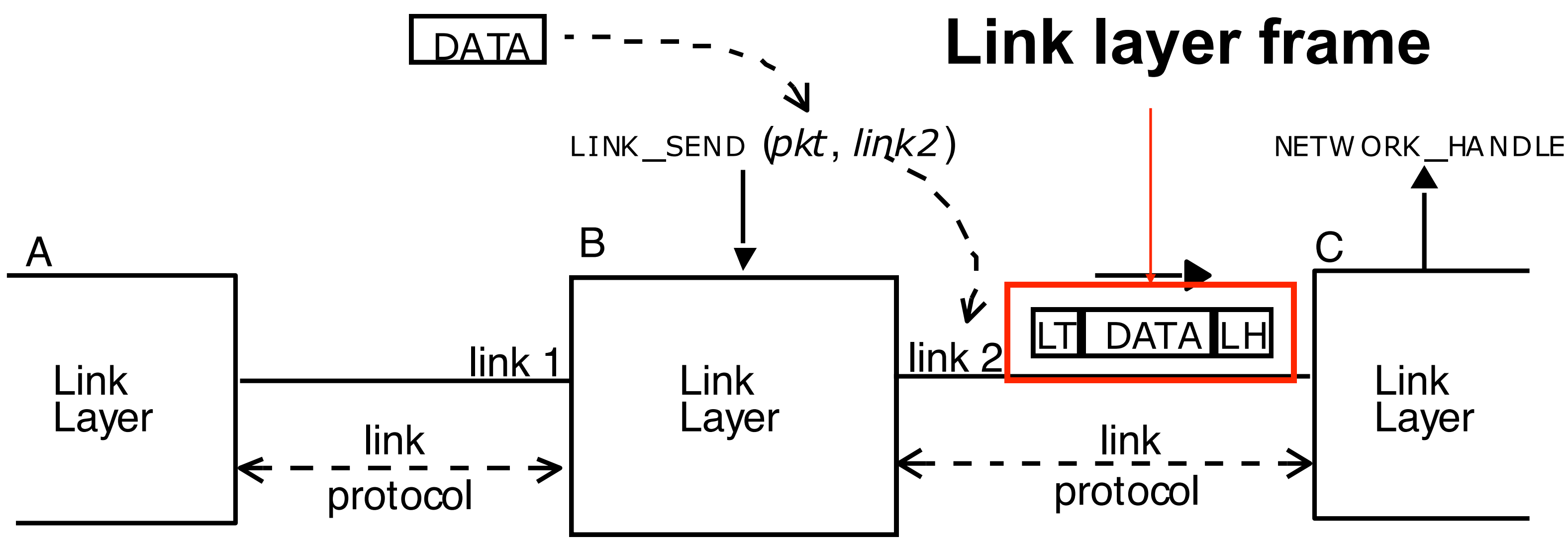
Example

- Node B has two links
- Forwarding A <-> C requires SEND to name two links, and a buffer
  - Link identifier is *local* to device



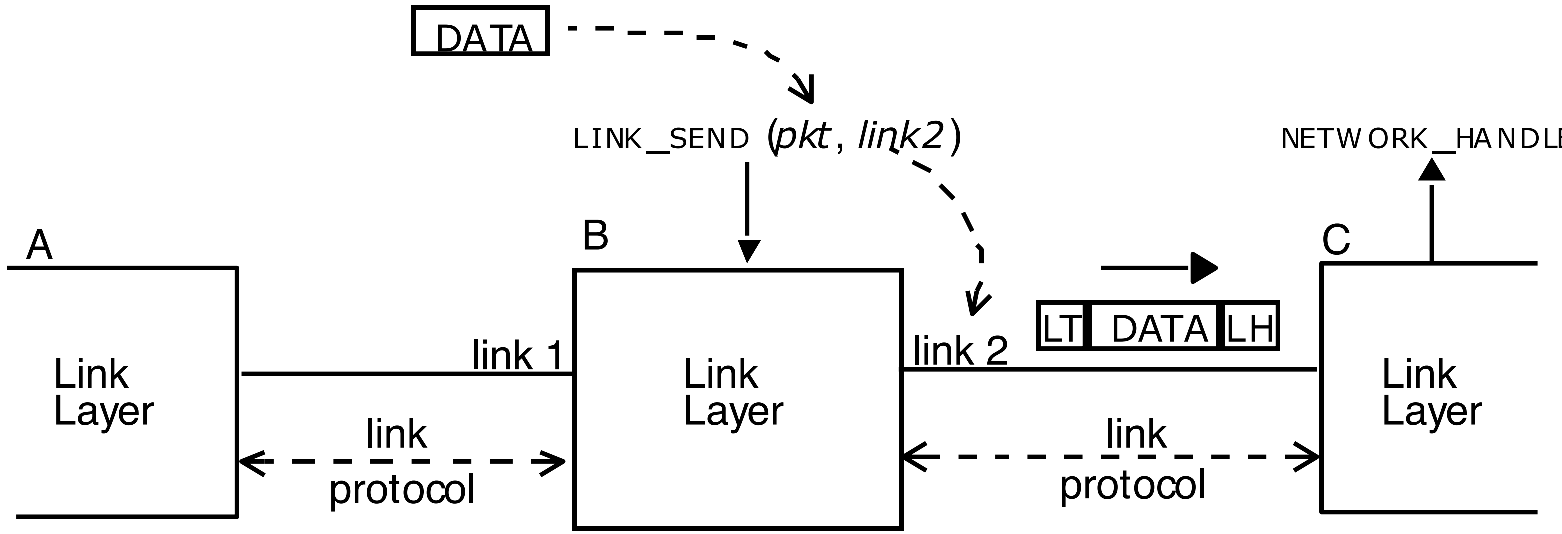
Example

- Header, trailer add information to frame
  - E.g. which network protocol to be used; source/destination link addresses; checksum for error detection

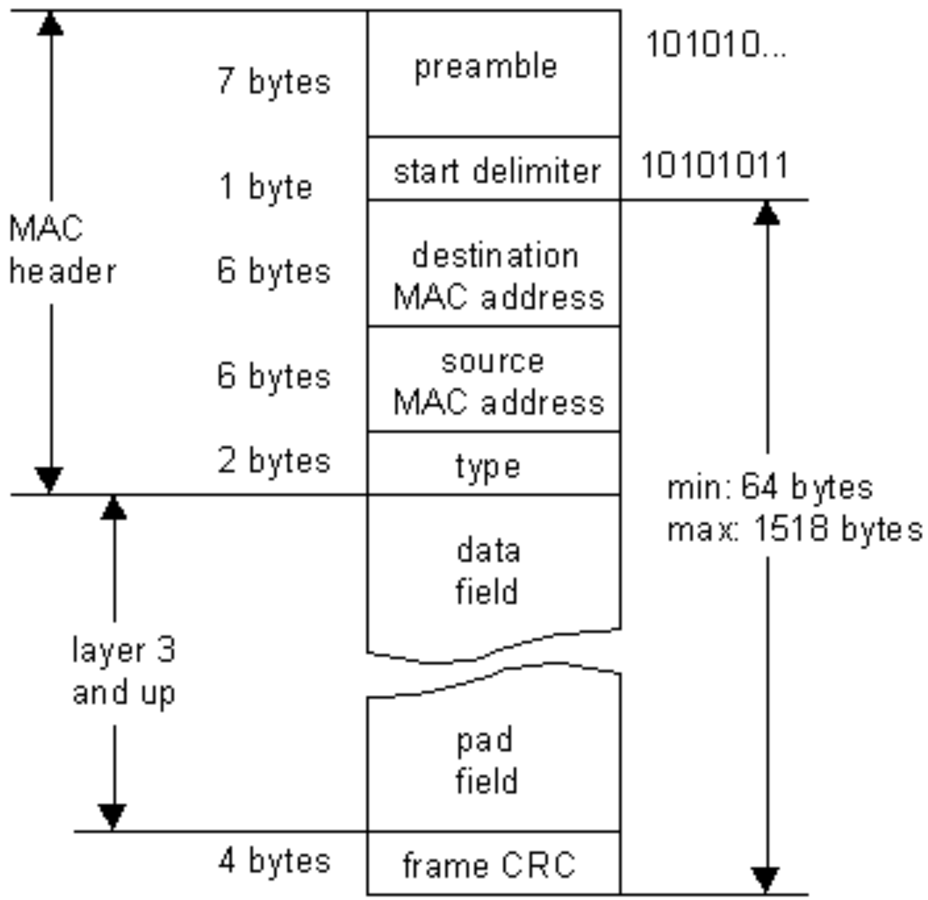


Receiving data

- Two approaches
  - Network layer polls link layer with a RECEIVE call to see if a frame has arrived
  - Link layer "up-calls" network layer when frames arrive – NETWORK_HANDLE



Example - Ethernet



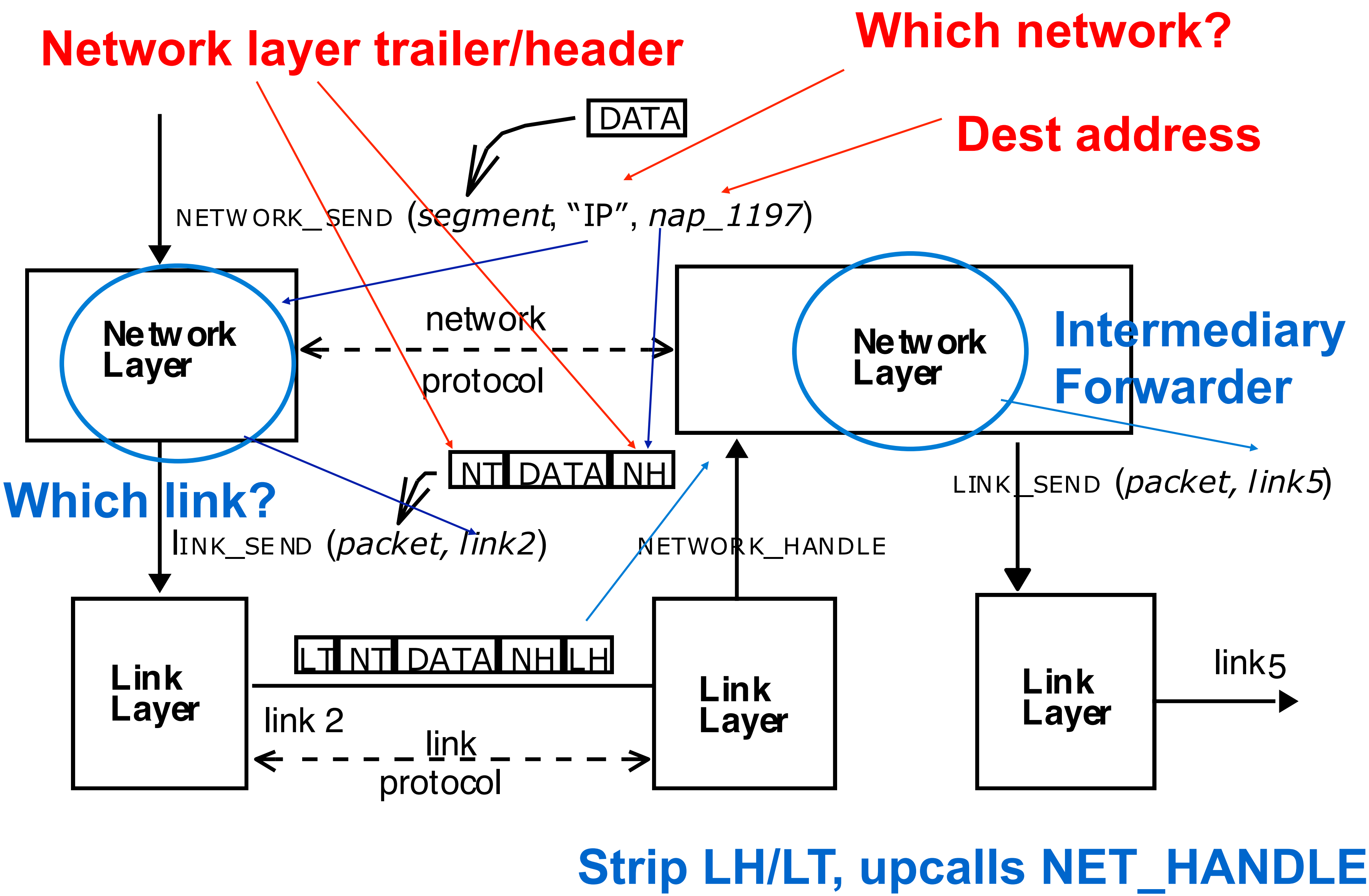
Network Layer

- A segment enters the network at a network attachment point (AP), and is received at another AP
  - Generally, need to traverse multiple links
- Network layer provides
  - Systematic naming of APs
  - Creating a packet
  - Determine which links to traverse and forwarding along this path

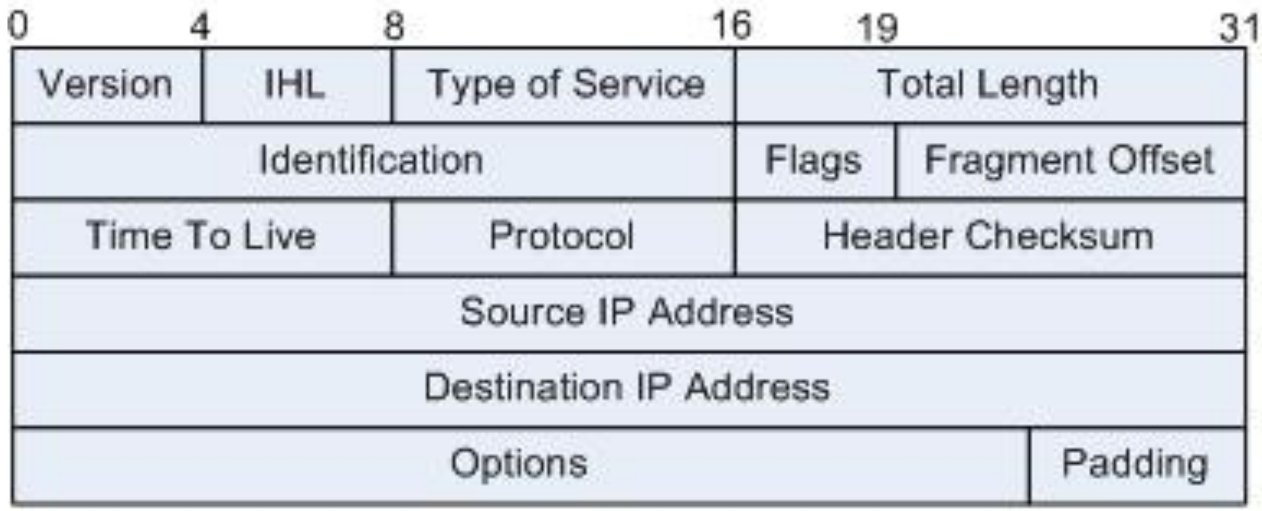
Network Layer

- Interface:
  – NETWORK_SEND (segment_buffer, network_identifier, destination)
- Segment buffer:
  – Payload
- Network_identifier:
  – Which network to send packet to
- Destination:
  – Name of destination network attachment point (network-layer address)

Network Layer



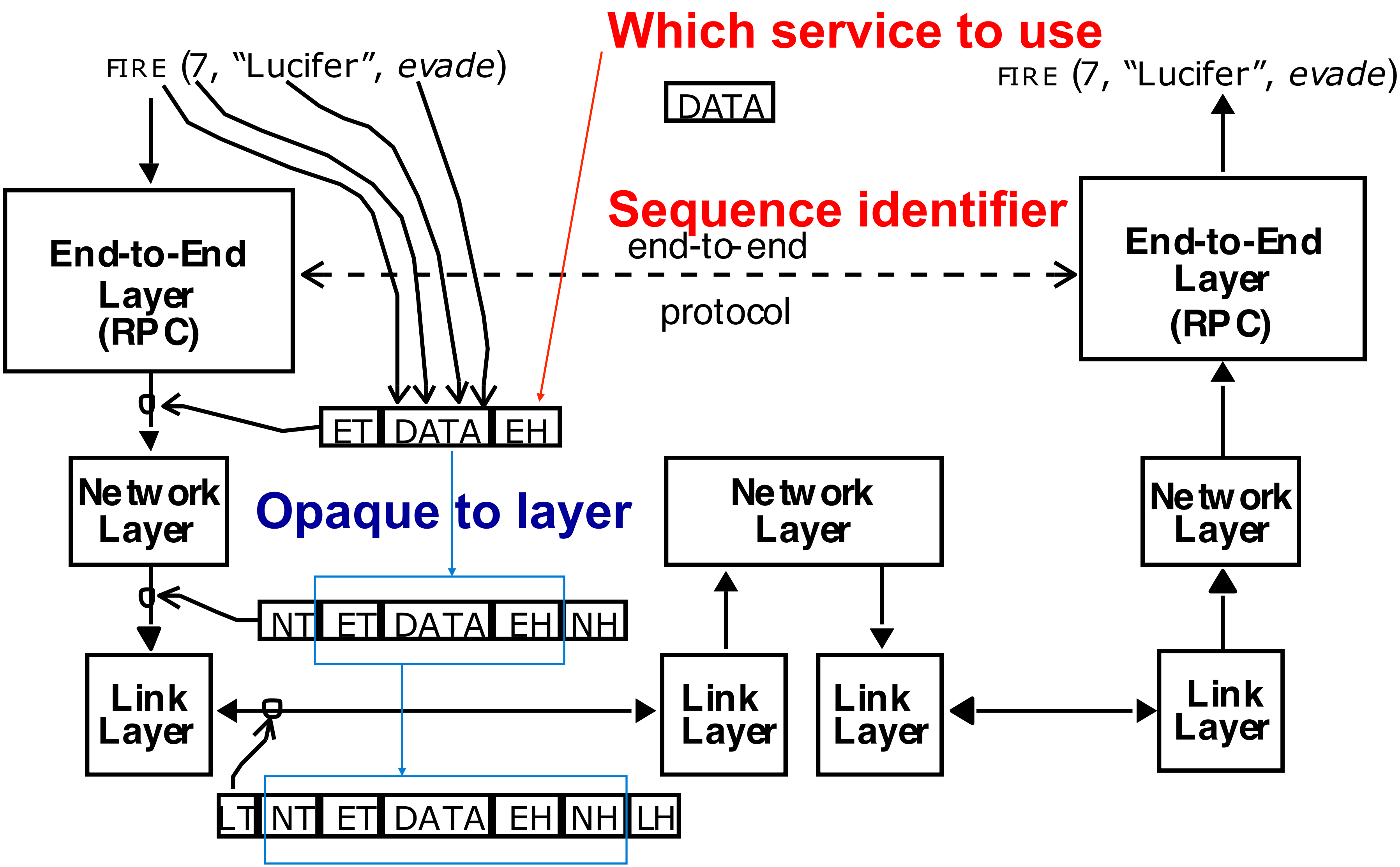
Example – IP packet header



End-to-end Layer

- Network, link layers: best-effort contract
  - Packet loss, duplicates, out-of-order, errors
- End-End layer provides an easier to use interface for applications
- Example - RPC provides:
  - Presentation: Translates data formats, emulating semantics of procedure call
  - Session: Negotiating discovery and binding to locate and prepare to use service
  - Transport: Dividing streams/messages into segments, dealing with packet loss, duplicates, out-of-order

End-to-End Layer



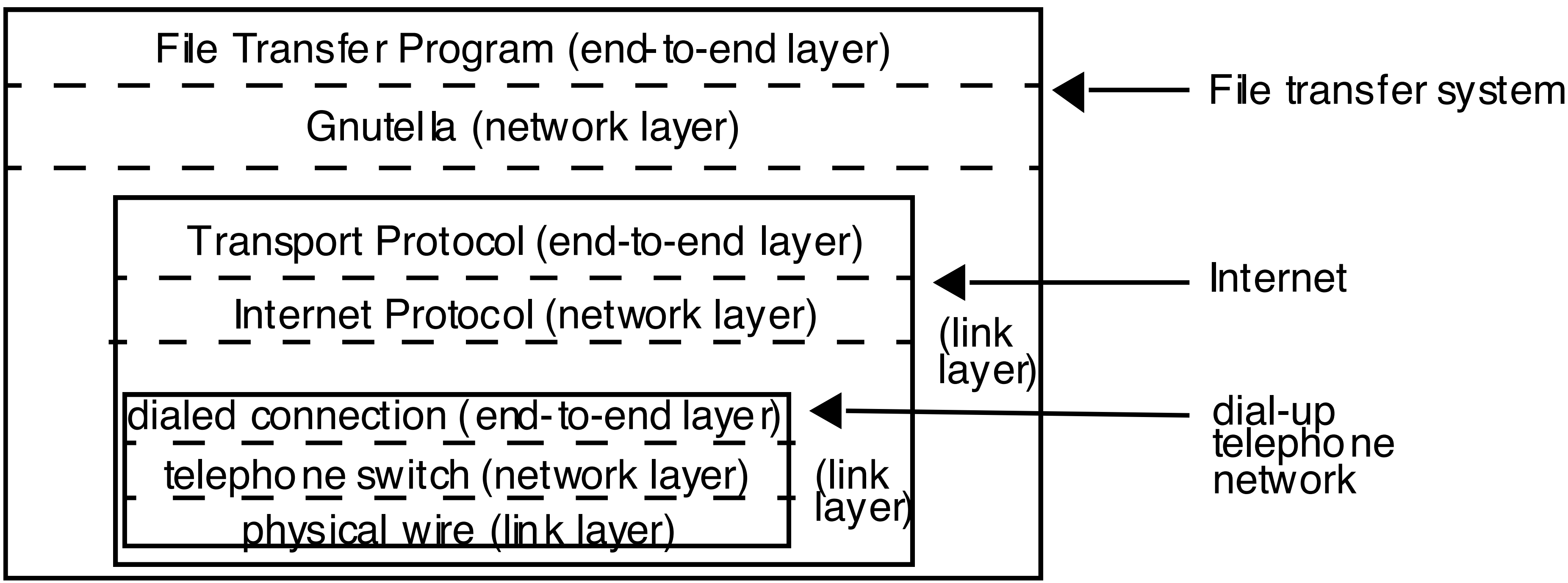
Example - Internet

- Various data links
  - E.g. dial-up, Ethernet, DSL, …
- Network layer – Internet Protocol
  - IP version 4
    - IP version 6
- End-to-end
  - Transport:
    - TCP – connection-oriented
    - UDP – datagram-oriented
    - RTP – ‘real-time’ (voice, media streaming)
      - May be layered on top of UDP
  - Application
    - File transfer protocol (FTP); HTTP; SMTP, POP

Recursive composition

- Network layer rests on a link layer that itself is a layered network
  - Dial-up phone line
    - A 3-layer network of its own
      - Used as a link layer for Internet
  - Application-level overlays
    - E.g. Gnutella
      - Use end-to-end Internet layer for its links

Recursive composition



Mapping requirements to layers

- Different applications - different requirements
  - Voice:
    - Steady stream, data rate supporting audio quality
    - Low latency/jitter
    - Ok to lose small fraction of data (lost packets, data errors)
  - Reliable transfer of large files
    - High throughput
    - Guaranteed data integrity
    - Out-of-order, duplicate packets ok
  - Financial transaction/trading application
    - Low latency – time is money
    - At-most-once semantics, in-order delivery

Mapping requirements to layers

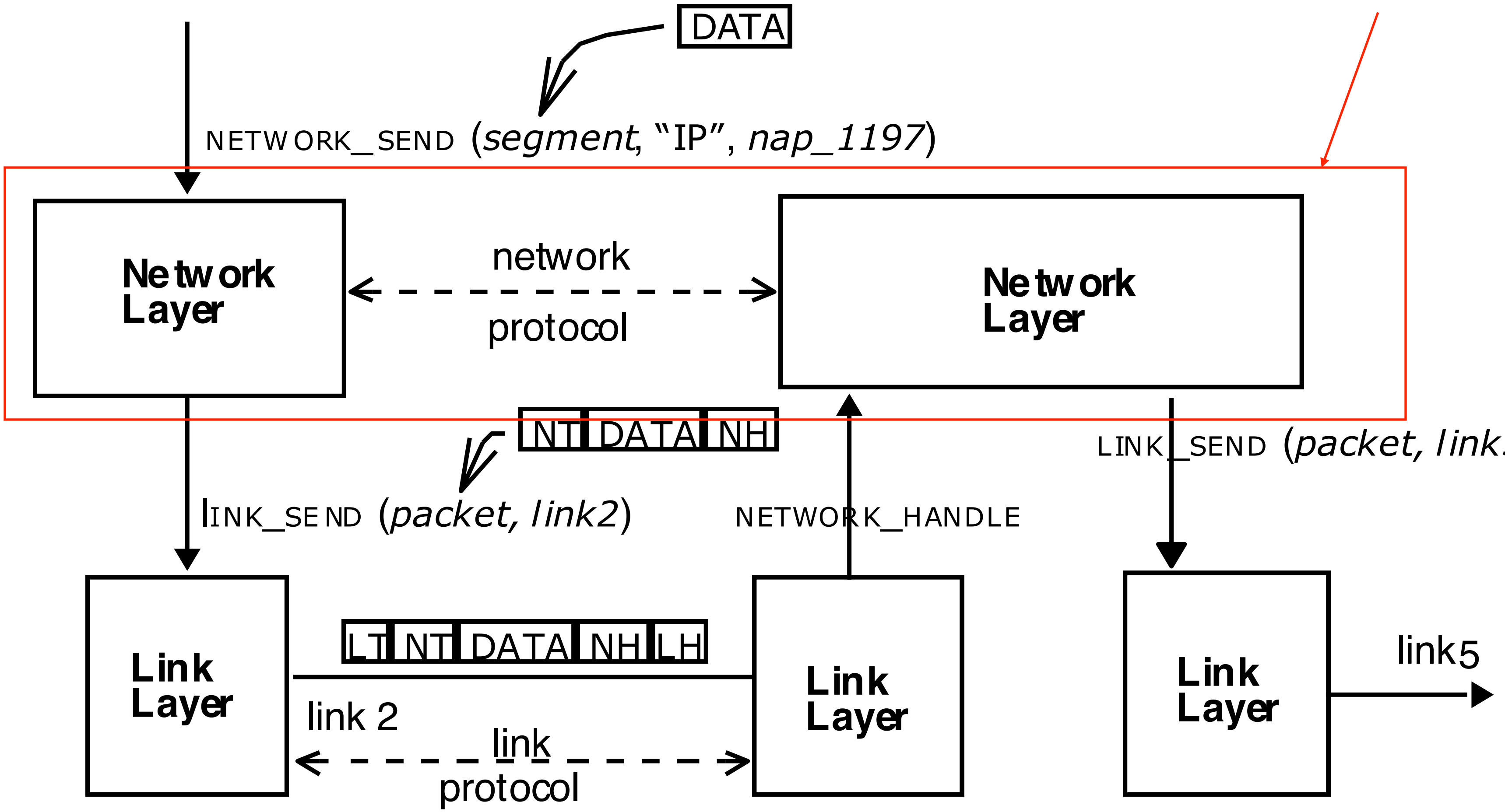
- Data link and network layers provide core functionality
  - Move data from A to B, possibly through intermediaries
- What else provided at these layers to support application requirements
  - Error detection/correction?
  - Dealing with duplicates?

End-to-end principle

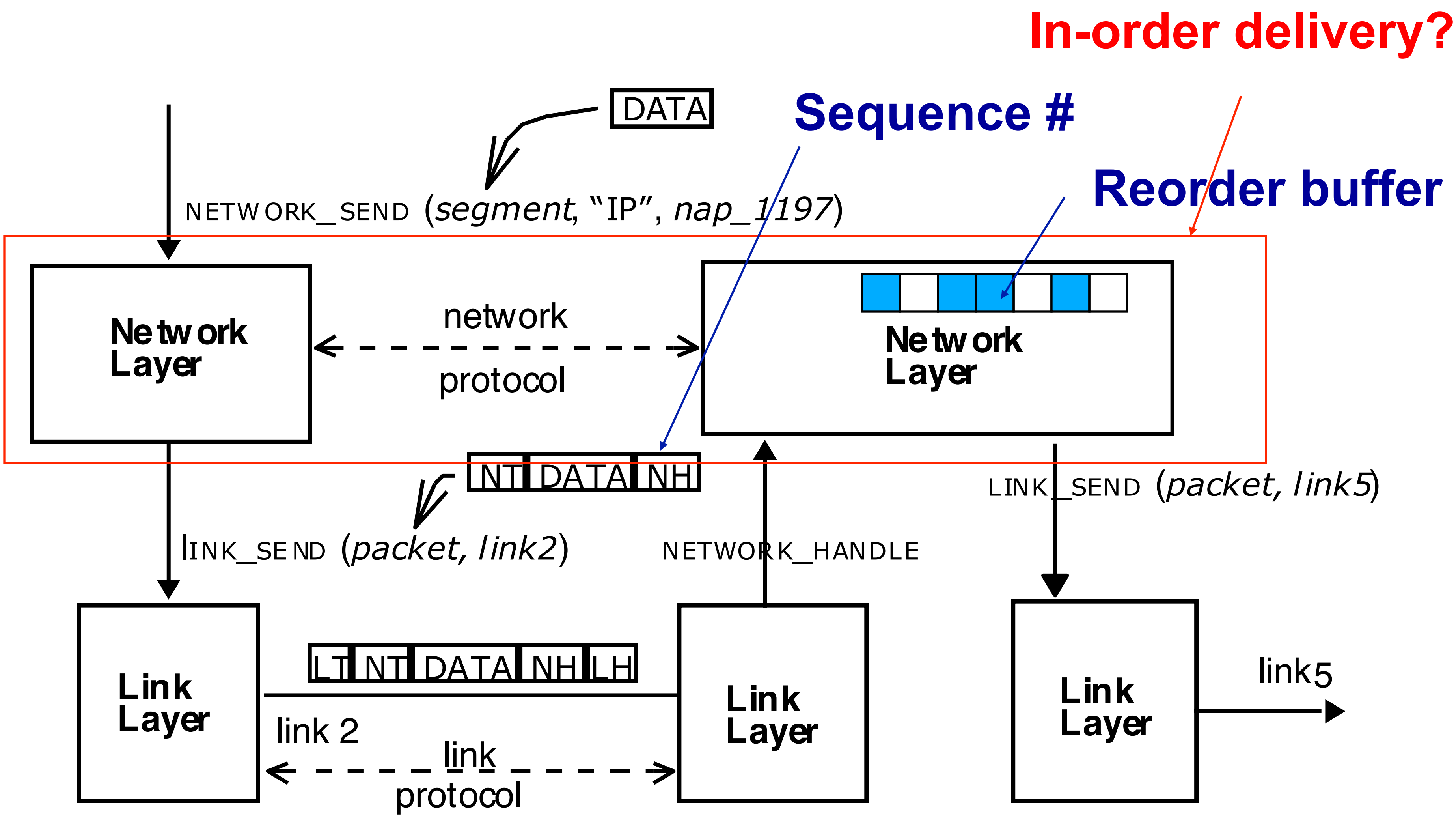
- Design principle: “the application knows best”
  - Communication requirements are application specific
    - There are many applications that have similar requirements (e.g. RPC), but no single class covers all applications
    - Implementing support for features other than data movement at lower layers may not suit what is needed by application
      - Applications that do not use feature - inefficient
      - Still require applications for which feature is not quite what they need to re-implement

Example

In-order delivery?



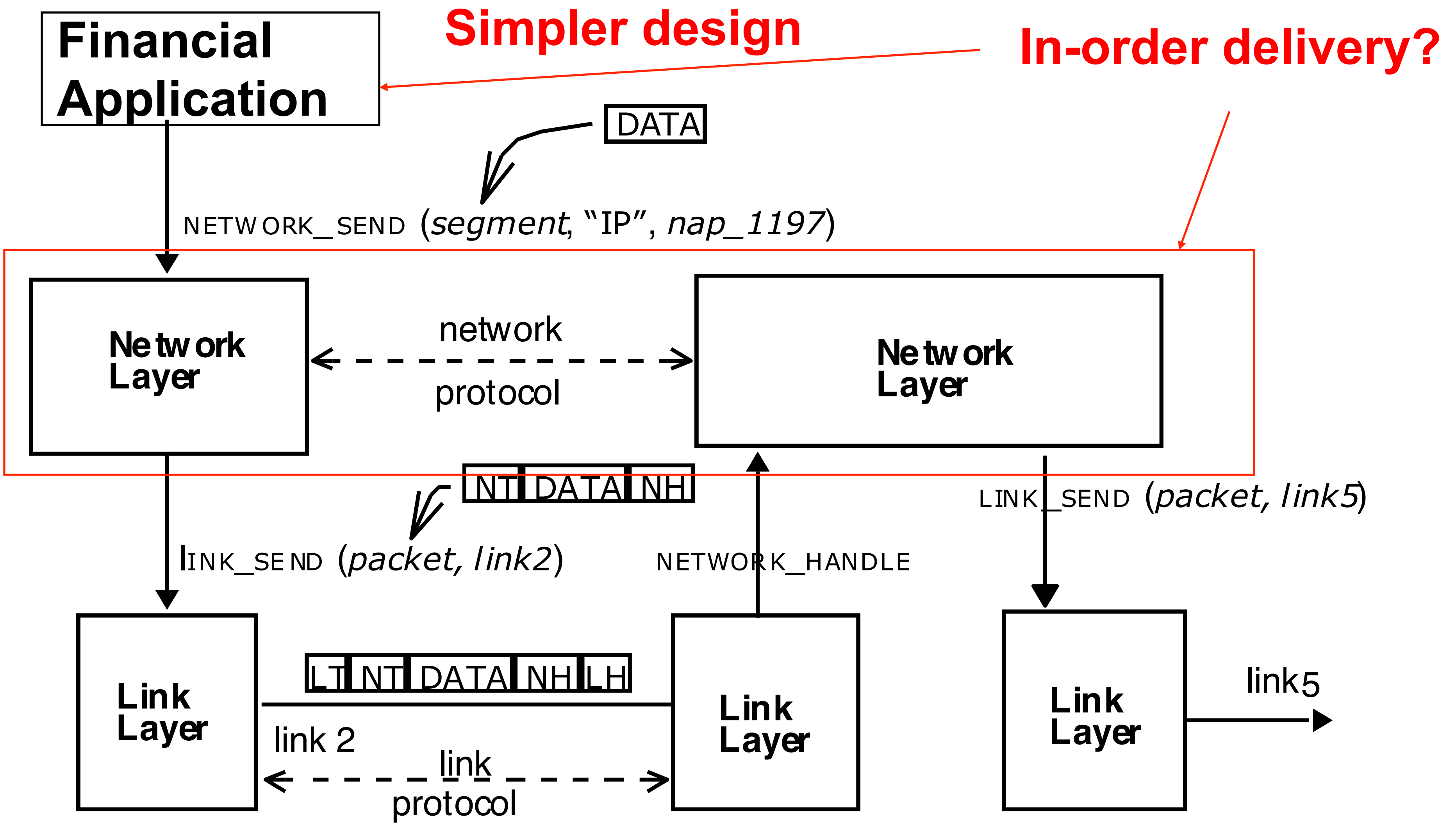
Example


In-order delivery?
DATA
Sequence #
Reorder buffer
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
LINK_SEND (packet, link5)
lINK_SEND (packet, link2)
NETWORK_HANDLE
Link Layer
LT NT DATA NH LH
link 2
link
protocol
Link Layer
Link Layer
link5

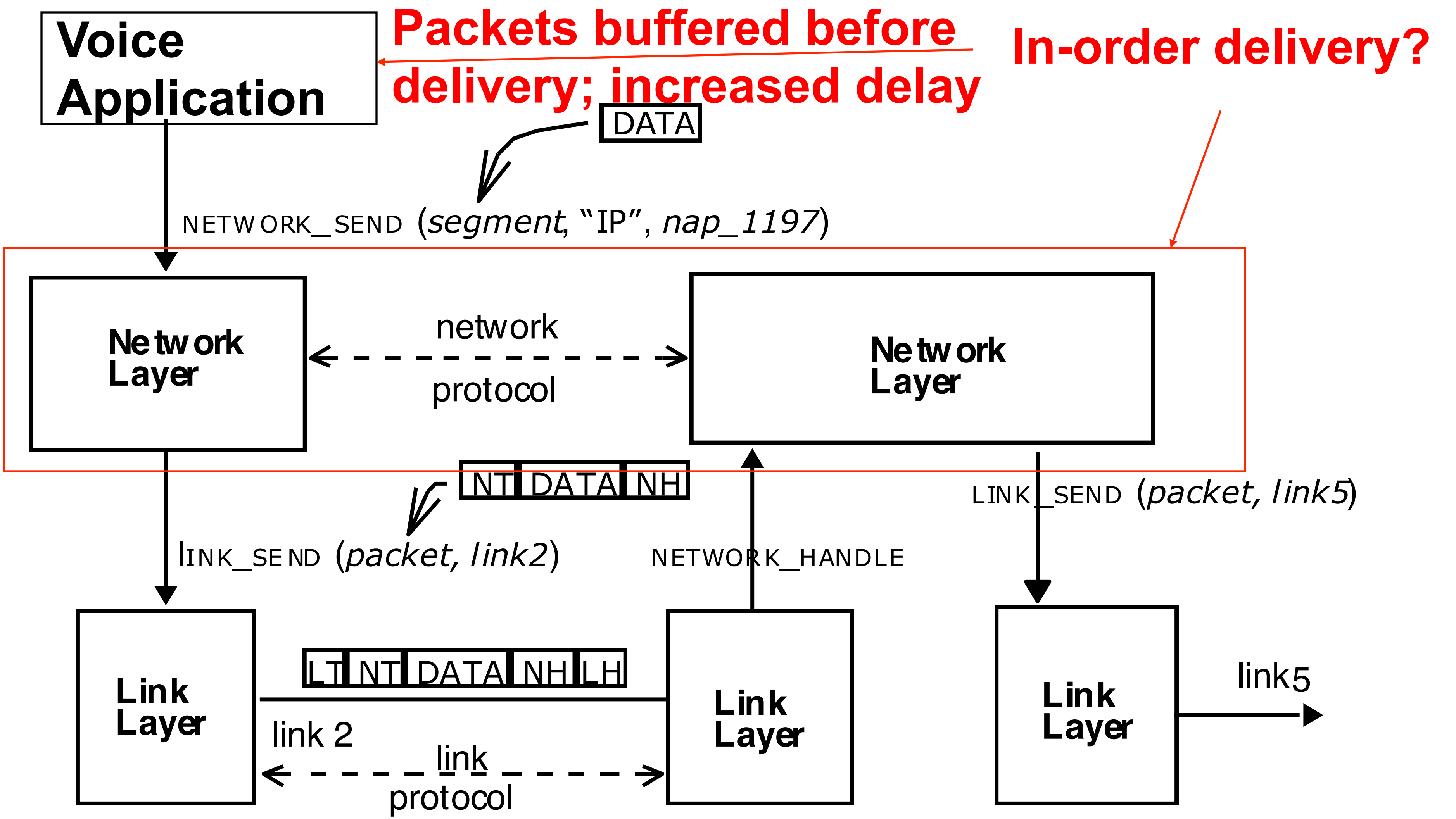
In-order delivery

- In-order delivery in network layer
- Advantage:
  - Design of modules in layer above can be simplified – assume in-order delivery
- Disadvantages:
  - Additional cost associated with every message to guarantee in-order delivery
    - Additional header information, e.g. sequence #
    - Additional code complexity, runtime
  - Additional buffering/delay for packets

Example

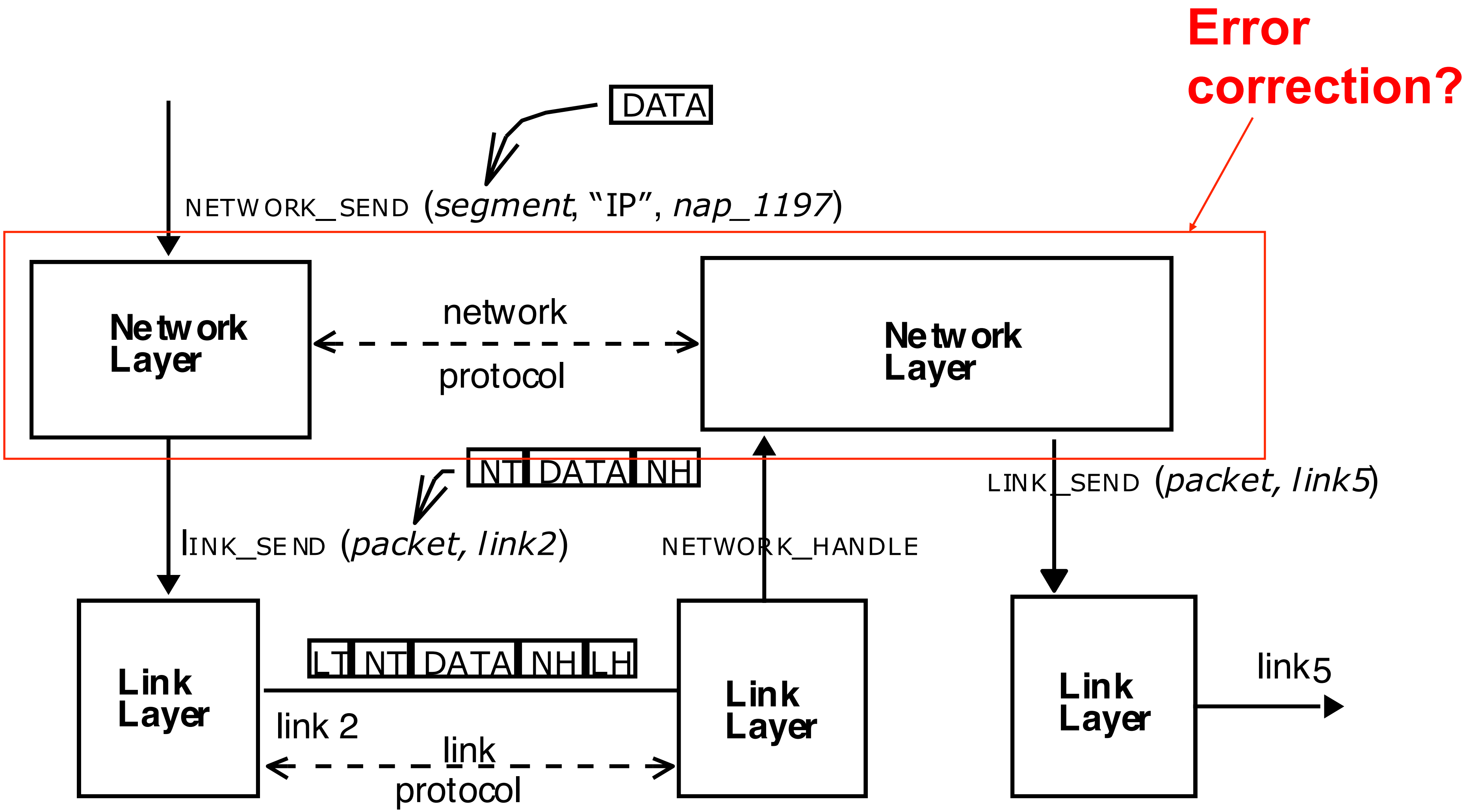

Simpler design
In-order delivery?
Financial Application
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
lINK_SEND (packet, link2)
NETWORK_HANDLE
LINK_SEND (packet, link5)
Link Layer
LT NT DATA NH LH
link 2
link
protocol
Link Layer
Link Layer
link5

Example

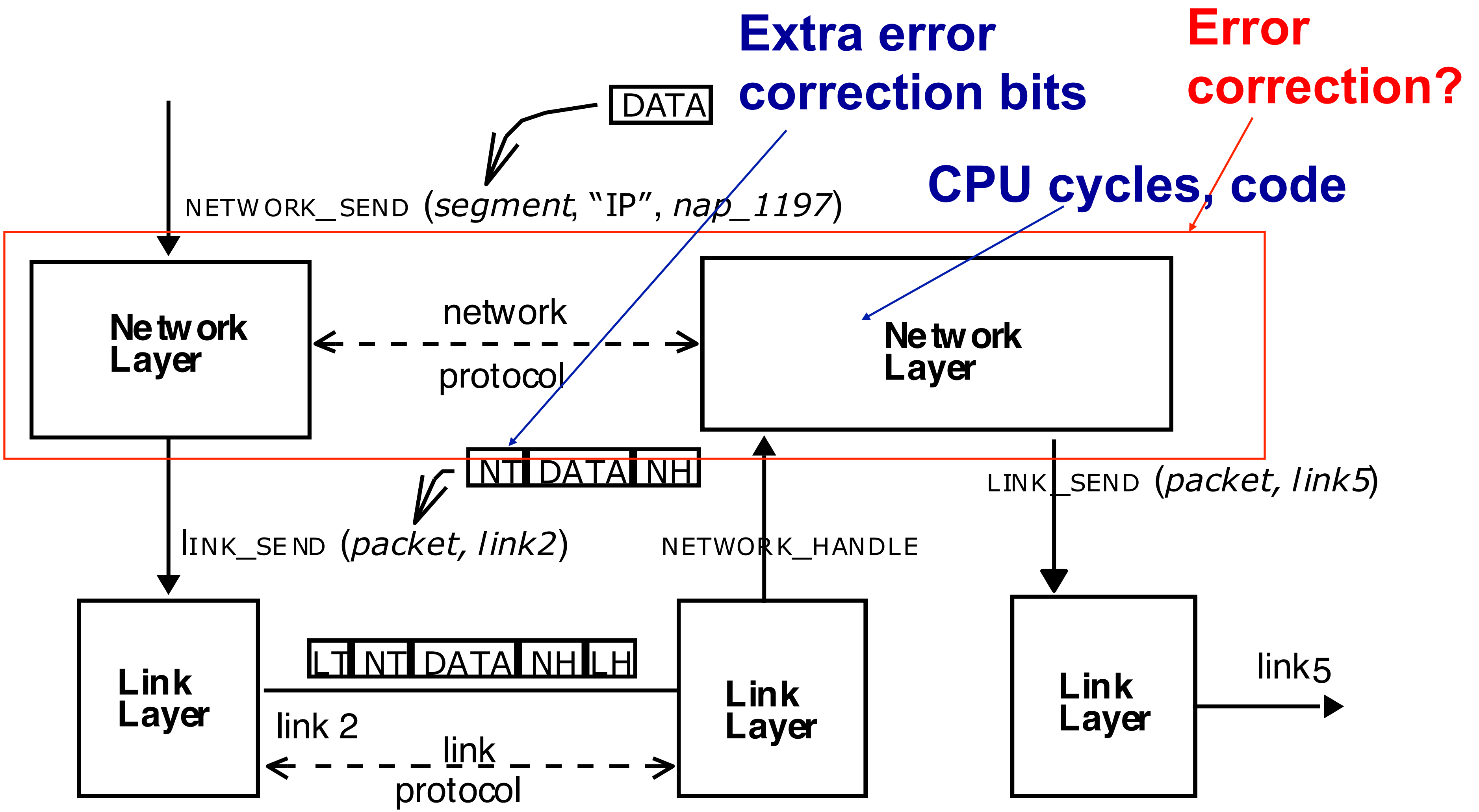


Voice Application
Packets buffered before delivery; increased delay
In-order delivery?
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
LINK_SEND (packet, link2)
NETWORK_HANDLE
LINK_SEND (packet, link5)
Link Layer
LT NT DATA NH LH
link 2
link
protocol
Link Layer
Link Layer
link5

Example


Error
correction?
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network
Layer
network
protocol
Network
Layer
NT DATA NH
lINK_SEND (packet, link2)
NETWORK_HANDLE
LINK_SEND (packet, link5)
Link
Layer
LT NT DATA NH LH
link 2
link
protocol
Link
Layer
Link
Layer
link5

Example

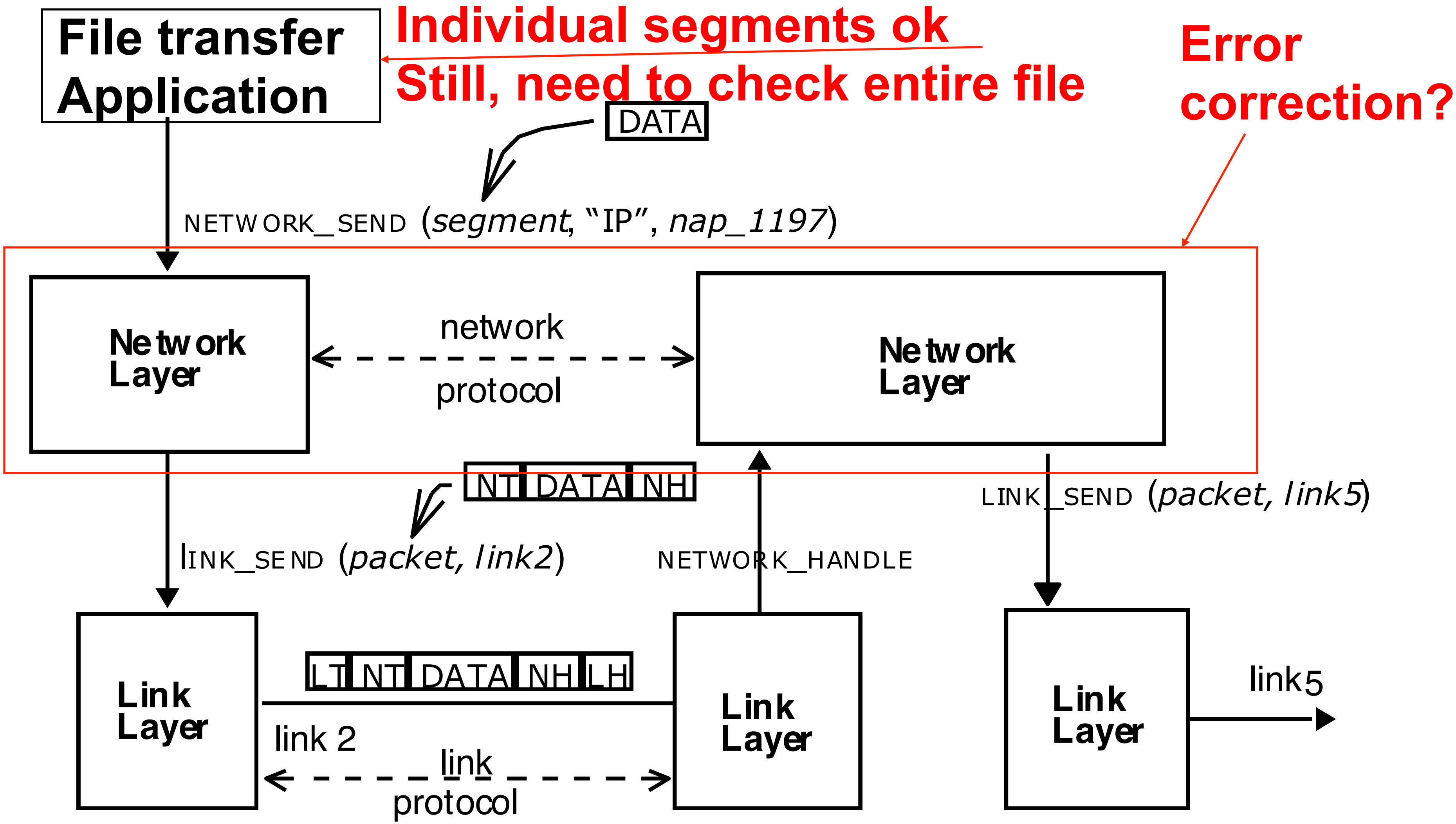


Extra error correction bits
Error correction?
CPU cycles, code
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
LINK_SEND (packet, link5)
lINK_SEND (packet, link2)
NETWORK_HANDLE
Link Layer
LT NT DATA NH LH
Link Layer
Link Layer
link5
link 2
link
protocol

Error correction

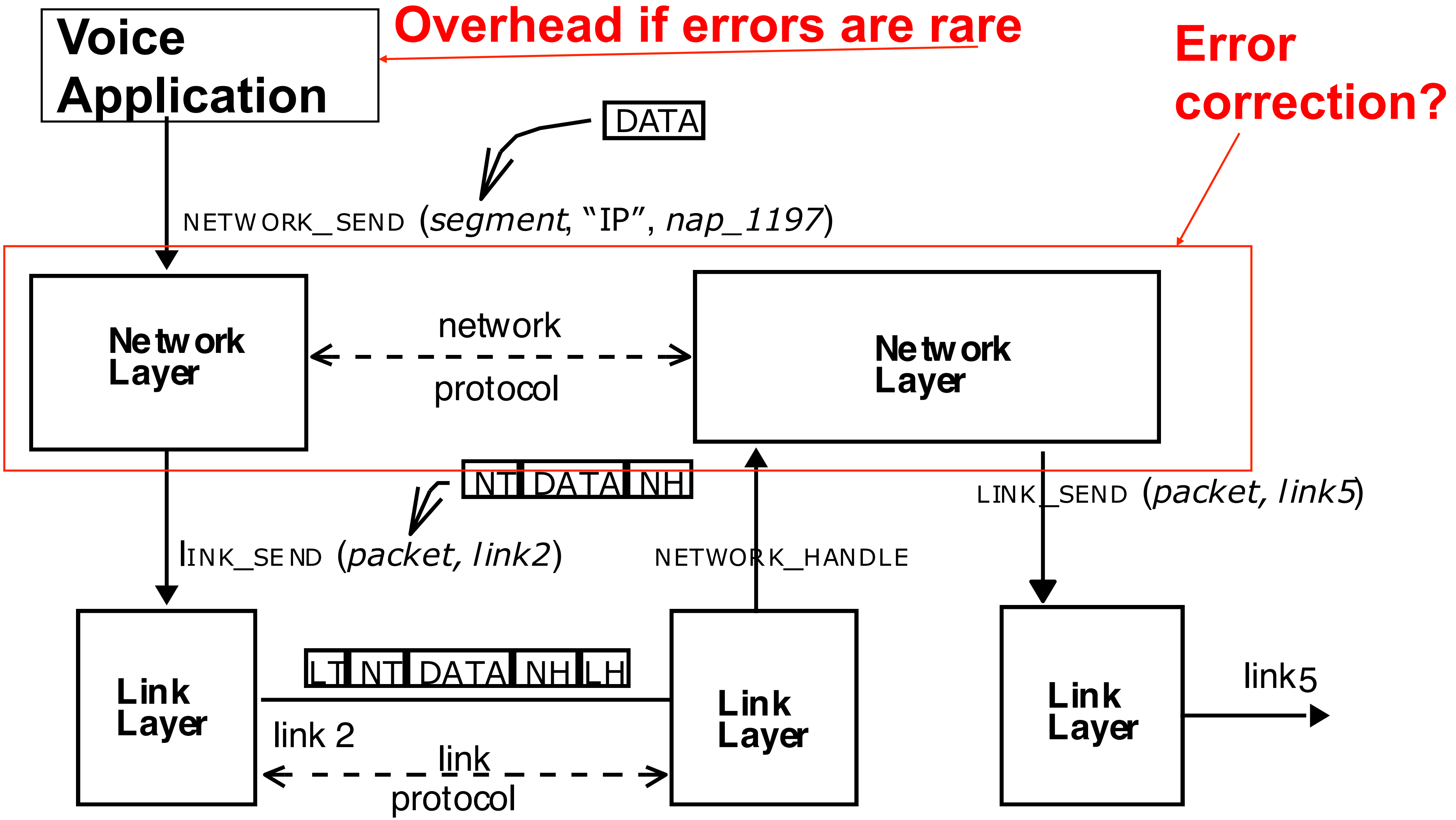
- Error correction in network layer
- Advantage:
  – Design of modules in layer above can be simplified – assume if a packet arrives, it has no errors
- Disadvantages:
  – Additional cost associated with every message to guarantee in-order delivery
    - Additional trailer information, ECC
    - Additional code complexity, runtime
  – When correction code not sufficient, must still retransmit
    - Additional code complexity; delay if slows down other packets (e.g. in-order delivery)

Example



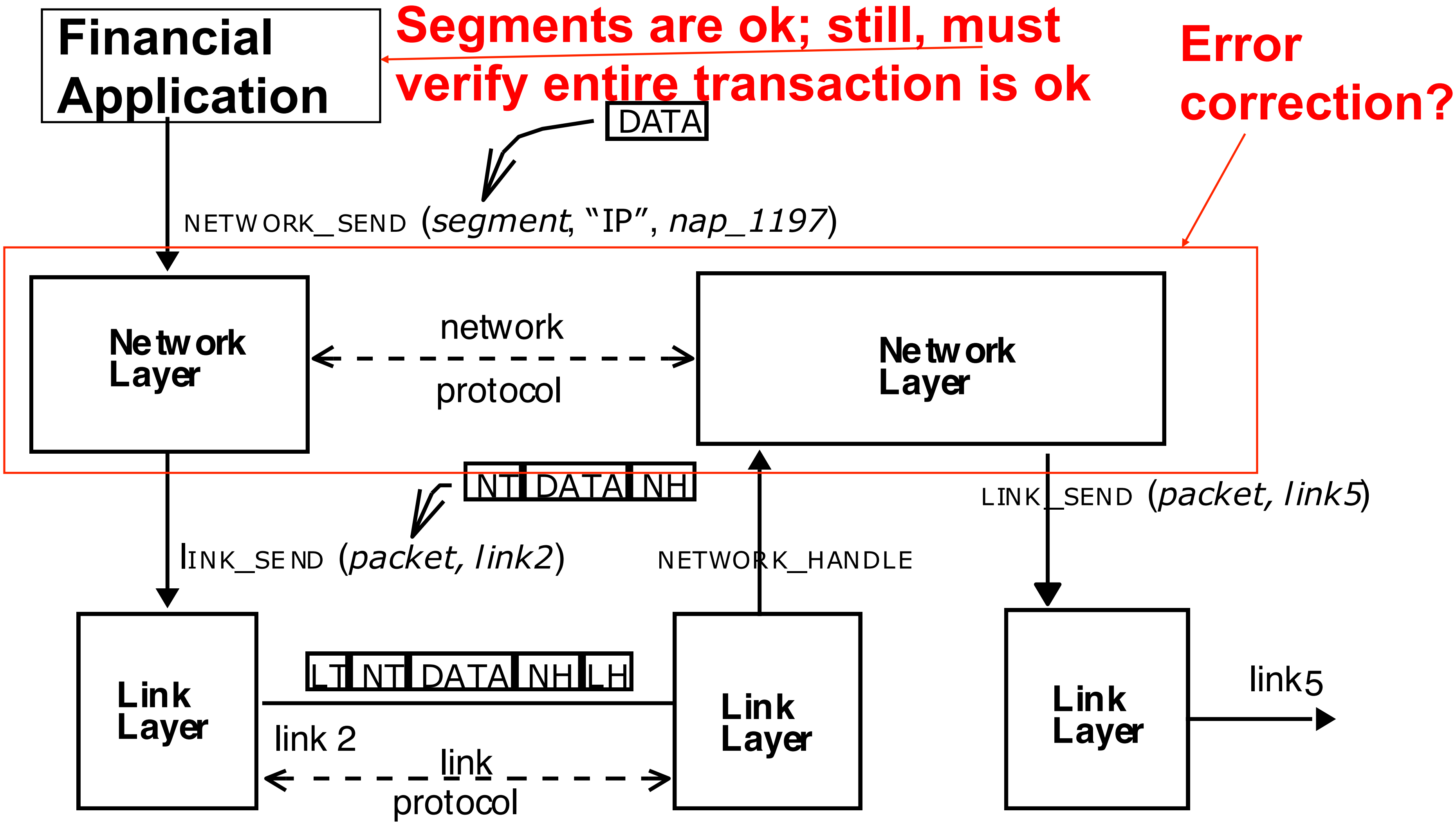
File transfer Application
Individual segments ok
Still, need to check entire file
Error correction?
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
LINK_SEND (packet, link5)
lINK_SEND (packet, link2)
NETWORK_HANDLE
LT NT DATA NH LH
Link Layer
link 2
link
protocol
Link Layer
Link Layer
link5

Example



Voice Application
Overhead if errors are rare
Error correction?
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
LINK_SEND (packet, link5)
lINK_SEND (packet, link2)
NETWORK_HANDLE
LT NT DATA NH LH
Link Layer
link 2
link
protocol
Link Layer
Link Layer
link5

Example



Financial Application
Segments are ok; still, must verify entire transaction is ok
Error correction?
DATA
NETWORK_SEND (segment, "IP", nap_1197)
Network Layer
network
protocol
Network Layer
NT DATA NH
LINK_SEND (packet, link5)
lINK_SEND (packet, link2)
NETWORK_HANDLE
LT NT DATA NH LH
Link Layer
link 2
link
protocol
Link Layer
Link Layer
link5

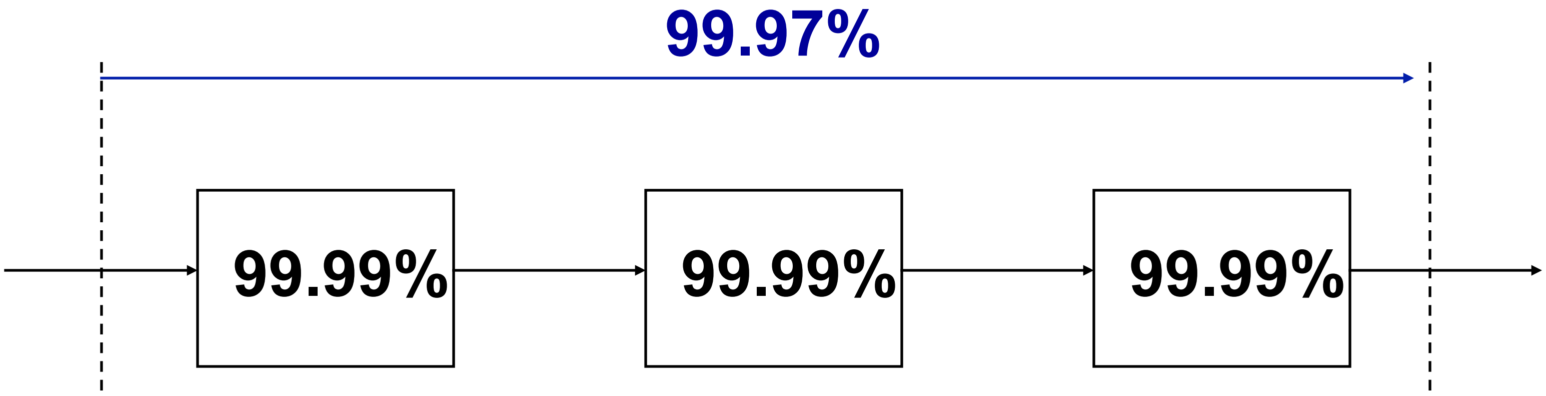
File transfer example

- Reliable file transfer application
  - Even though packets are guaranteed to be error-free by network layer, many other errors can occur as the application interacts with other systems
    - Crashes during transmission while buffers still in memory
    - Faults in hardware – e.g. bad block, bit flip
  - Important to guarantee integrity of the file stored in disk
    - E.g. compute an “end-to-end” checksum of entire file on source, verify on destination after stored in disk
    - Correcting an error may require re-transmitting the whole file, regardless of error correction support at network layer

File transfer example

- Each individual source of error might be dealt with at its own layer to lower probability of its occurrence
  - Redundancy in disks (RAID)
  - Error correction in memory
  - Error detection, retry in network
- No matter how small their probabilities, errors can still happen, outside control of application
  - If application must guarantee a target probability that errors do not happen, it needs to control this aspect

File transfer application



What if target probability of error being detected is 99.9999999% ?

End-to-end principle

- Avoid implementing functionality that is not strictly necessary at a lower layer
- Note, still may be desired for some features may be applied repeatedly across layers
  - E.g. fault-tolerance:
    - Ethernet has data link layer checksum: error detection for frames
    - TCP has transport layer checksum: error detection for messages
    - Reliable file transfer application: whole file checksum
  - Why?

End-to-end principle

- Example: errors caused by different reasons
  - Data link – noisy transmission
  - Transport – errors in intermediate devices (e.g. bit-flip in router)
  - File – disk errors, crash
- How likely are these errors?
- Different error rates call for different checksum codes
- Dealing with more common errors at lower layer
  - Simpler, faster
  - Avoid potentially expensive upper-layer operations

End-to-end principle

- Application desires to provide its own level of fault tolerance
- Support at lower layers for fault tolerance may reduce likelihood of faults that application will perceive
  - Does not eliminate need for end-to-end error correction
  - Performance enhancement, not application requirement
    - Tradeoffs:
      - Re-sending a frame vs. re-sending a file
      - More bits for all frames vs. more retransmissions

End-to-end principle

- Applies to various aspects of systems design – not just networking
  - E.g. file backups
  - Reduced instruction set (RISC) vs. CISC
- Not an absolute decision technique
  - But an argument/principle that should guide the decision-making process when you are designing modules/layers in a complex system

Reading

- Section 7.3