Implementing the LonTalk Protocol for Intelligent Distributed Control

Harold Rabbie Ph.D.

Embedded Systems Conference

http://www.lonworks.org.cn

EIA STANDARD

Control Network Protocol Specification

EIA-709.1

MARCH 1998

ELECTRONIC INDUSTRIES ALLIANCE
ENGINEERING DEPARTMENT

Electronic Industries Alliance

What is the LonTalk Protocol?

- A mechanism for intelligent devices to exchange sense and control information
- Developed by Echelon Corp.
- Standard: EIA-709.1 Electronic Industries Alliance Control Network Specification
  - Global Engineering Documents 1-800-854-7179
  - LonMark Interoperability Association **http://www.lonmark.org/PRESS/spec_3_0.PDF**

Design Goals for the LonTalk Protocol

- Media independence
  - Wide range of physical environments
- Supports very large networks
  - Handful of nodes to thousands of nodes
- Low installed cost
  - Low-cost simple nodes
  - Multi-drop, not point-to-point
- Very widely applicable
  - Large volumes lead to low cost

More Design Goals for the LonTalk Protocol

- No central controller needed
  - But not precluded, either
  - No single point of failure
- Inherently peer-to-peer
  - But can support master-slave
- Protocol subsets not needed, or permitted
  - Maximum interoperability
  - Vendor independence

Examples of Media that Support the LonTalk Protocol

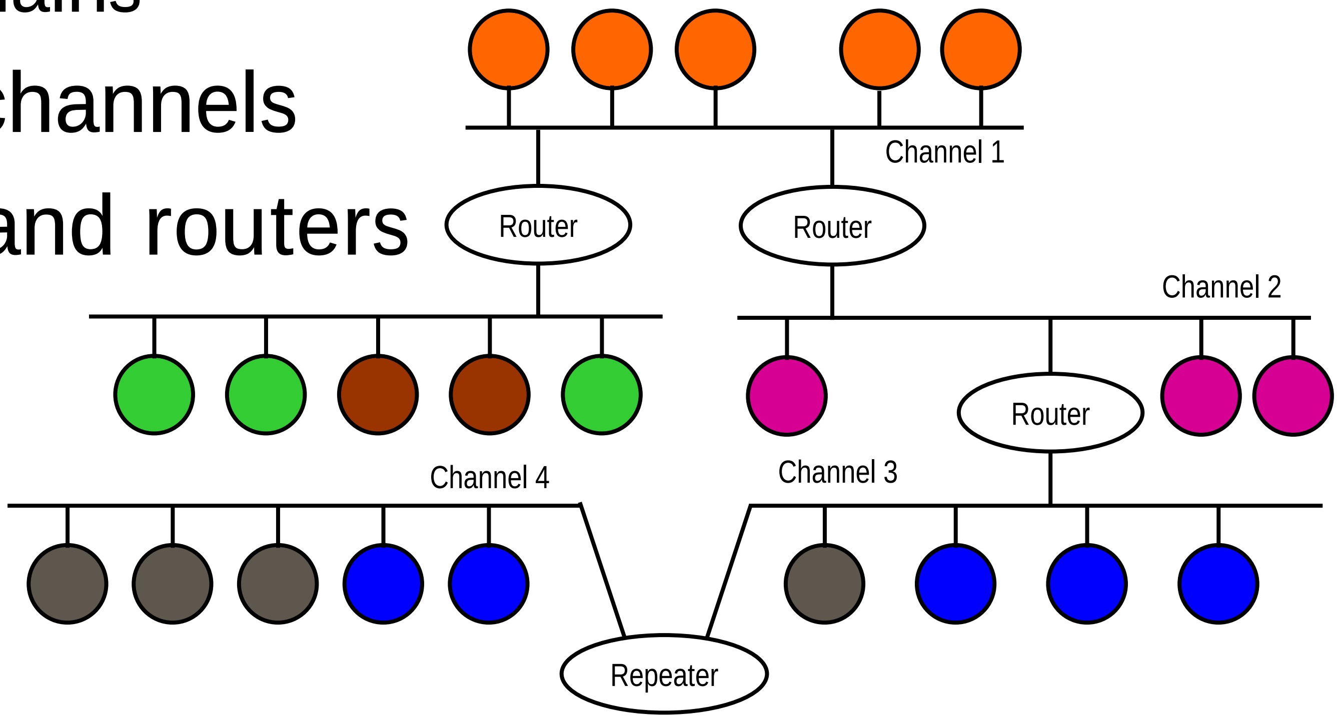
- Twisted pair wire
- Fiber optics
- Coaxial cable
- LonTalk over IP
- Radio frequency
- Power line
- Infrared
- Companion physical layer standards
  - EIA-709.2 and EIA-709.3
- Transceiver vendor list at **`http://www.echelon.com/products/contacts/tcvr.htm`**

Some Applications of the LonTalk Protocol

- Heating, ventilation and air-conditioning
- Industrial and process control
- Utility demand-side management
- Medical and scientific instrumentation
- Security and home automation
- Entertainment networks

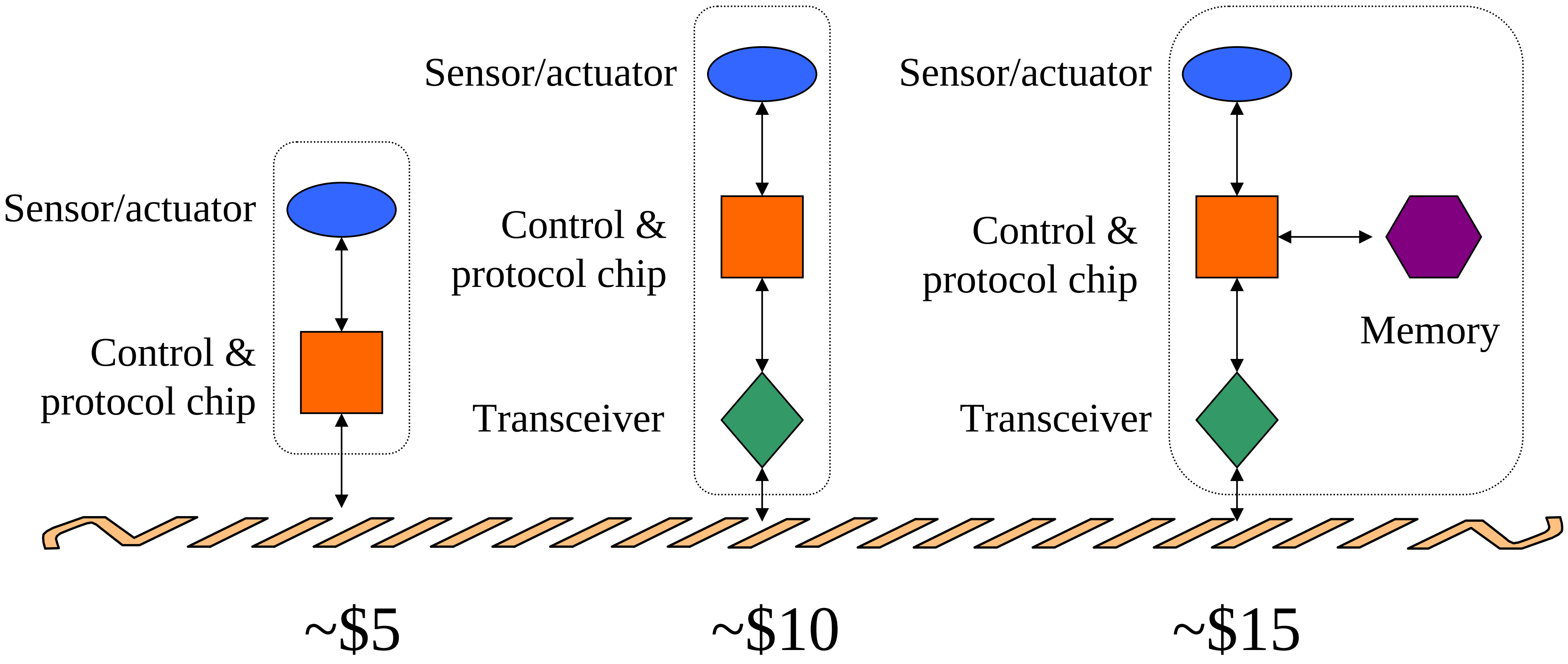
Support for Large Networks

- Very large address space
  - 32,385 nodes per domain
  - 2^{48} domains
- Multiple channels
- Subnets and routers



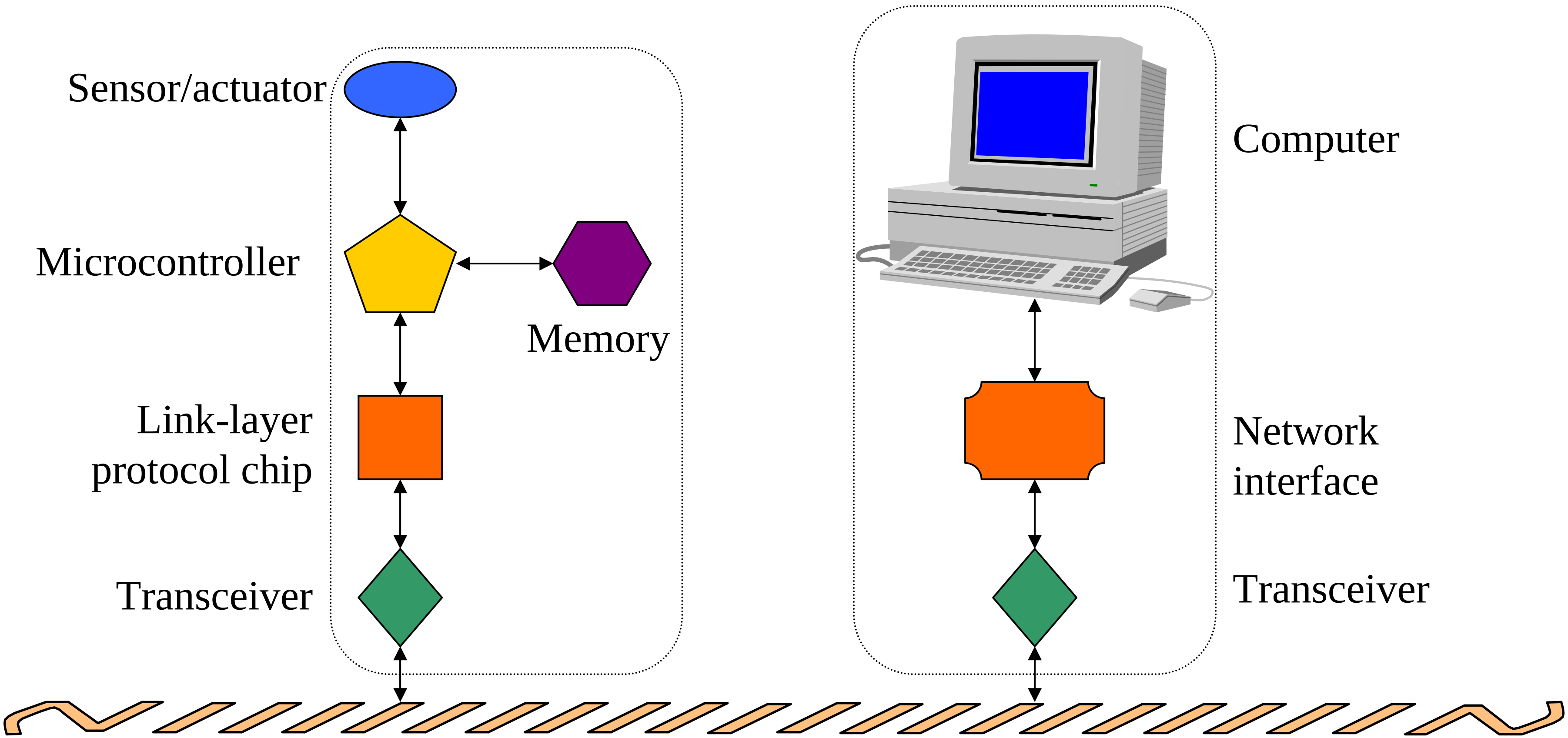
Scalability of LonTalk Implementations I

⌘High volume, low cost nodes

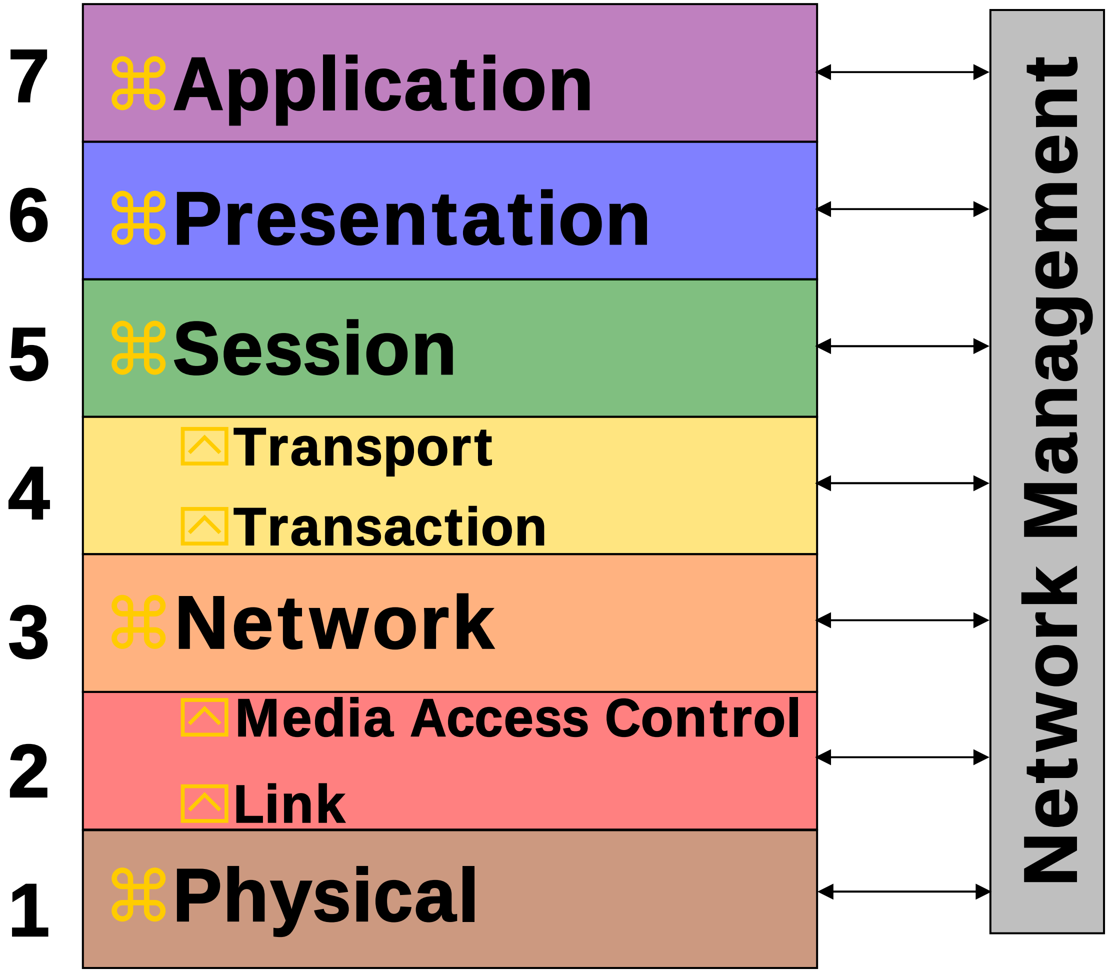


Scalability of LonTalk Implementations II

- Higher-capability nodes



Seven-Layer ISO-Model Protocol Stack



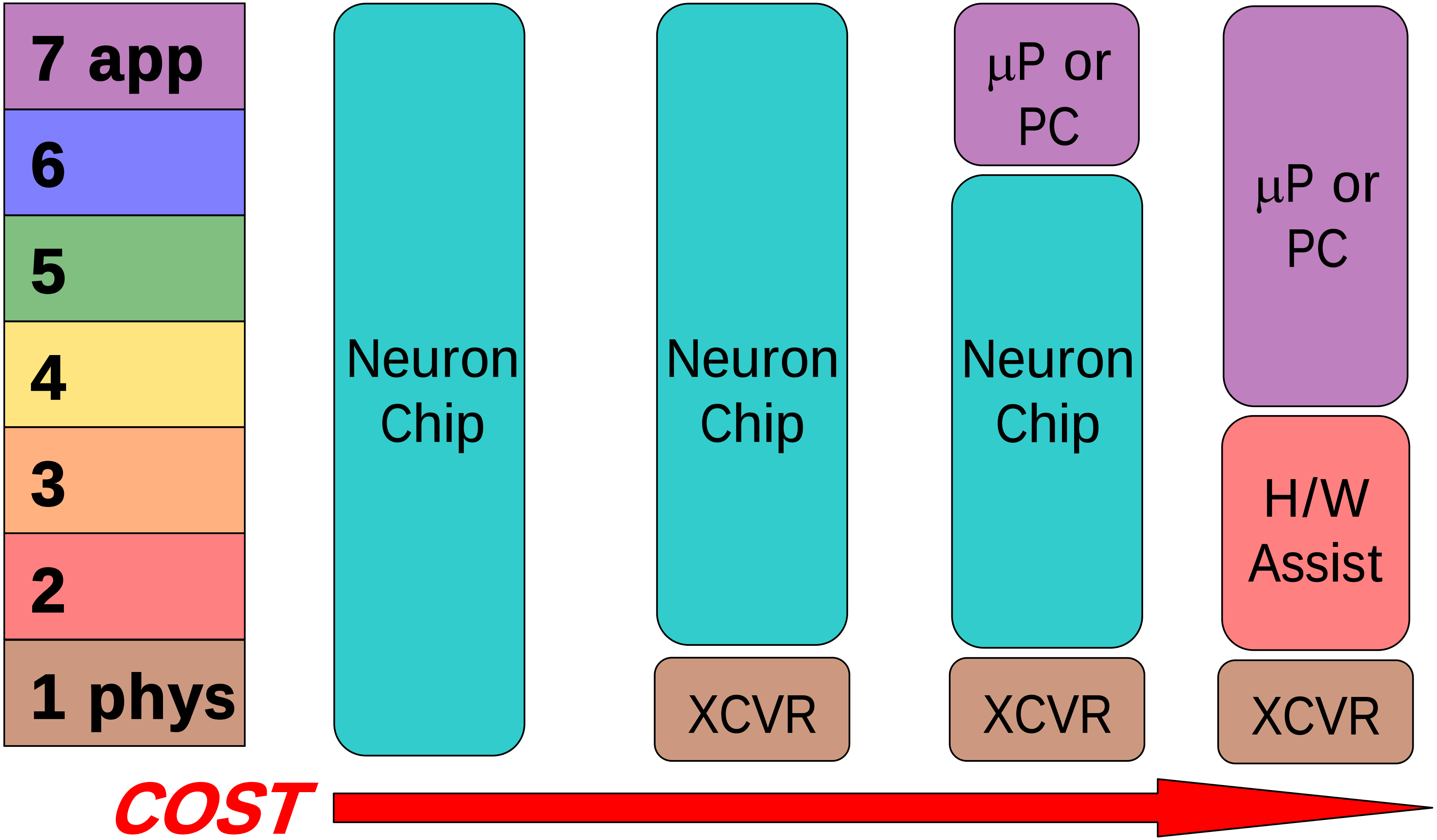
LonTalk Network Management Protocol

- Defined protocol for device configuration
  - Set application-level configuration parameters
  - Read device self-documentation data
  - Set media access layer parameters
    - Priority, timing factors
  - Set transport layer parameters
    - Service type, retry count, transaction timers
  - Load an updated application into EEPROM

LonTalk Protocol Implementation Choices

- General-purpose microprocessor
  - Upper protocol layers in user software
  - Hardware implements at least the link layer
- Neuron Chip microcontroller
  - Layers 2-6 in embedded hardware and firmware
  - App Layer in on-chip application CPU
  - *or* App Layer in separate host CPU

Node Cost Considerations

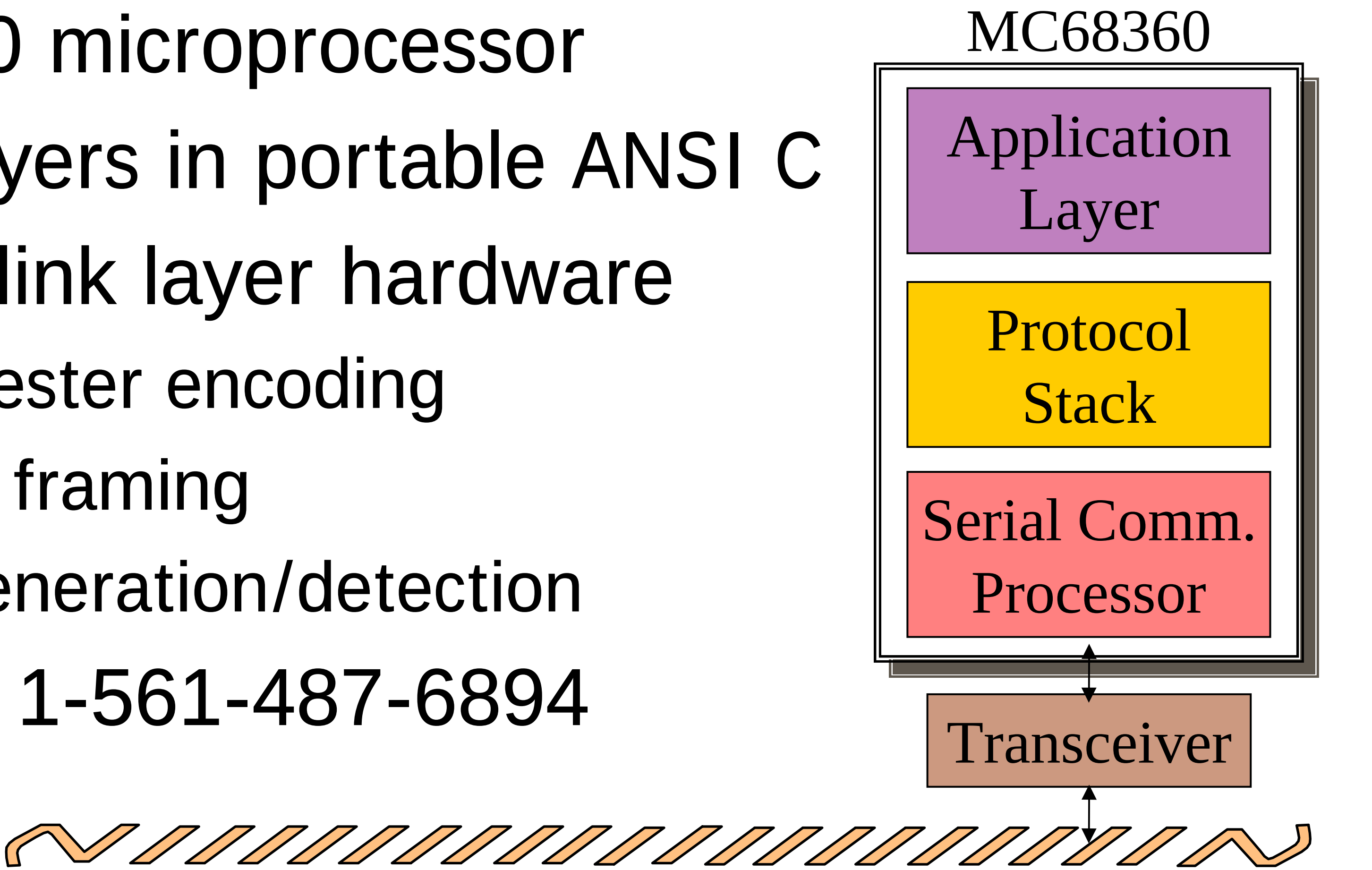


LonTalk Implementations

- Neuron Chip - the “Gold Standard”
  - Over 5,000,000 installed devices
  - Manufactured by Cypress and Toshiba
- Portable ANSI C implementation
  - Adept Systems, Boca Raton, FL
- Orion Stack, L-chip
  - Loytec Electronics, Vienna, Austria
- Process Network Computer Chip
  - Toshiba, JavaSoft

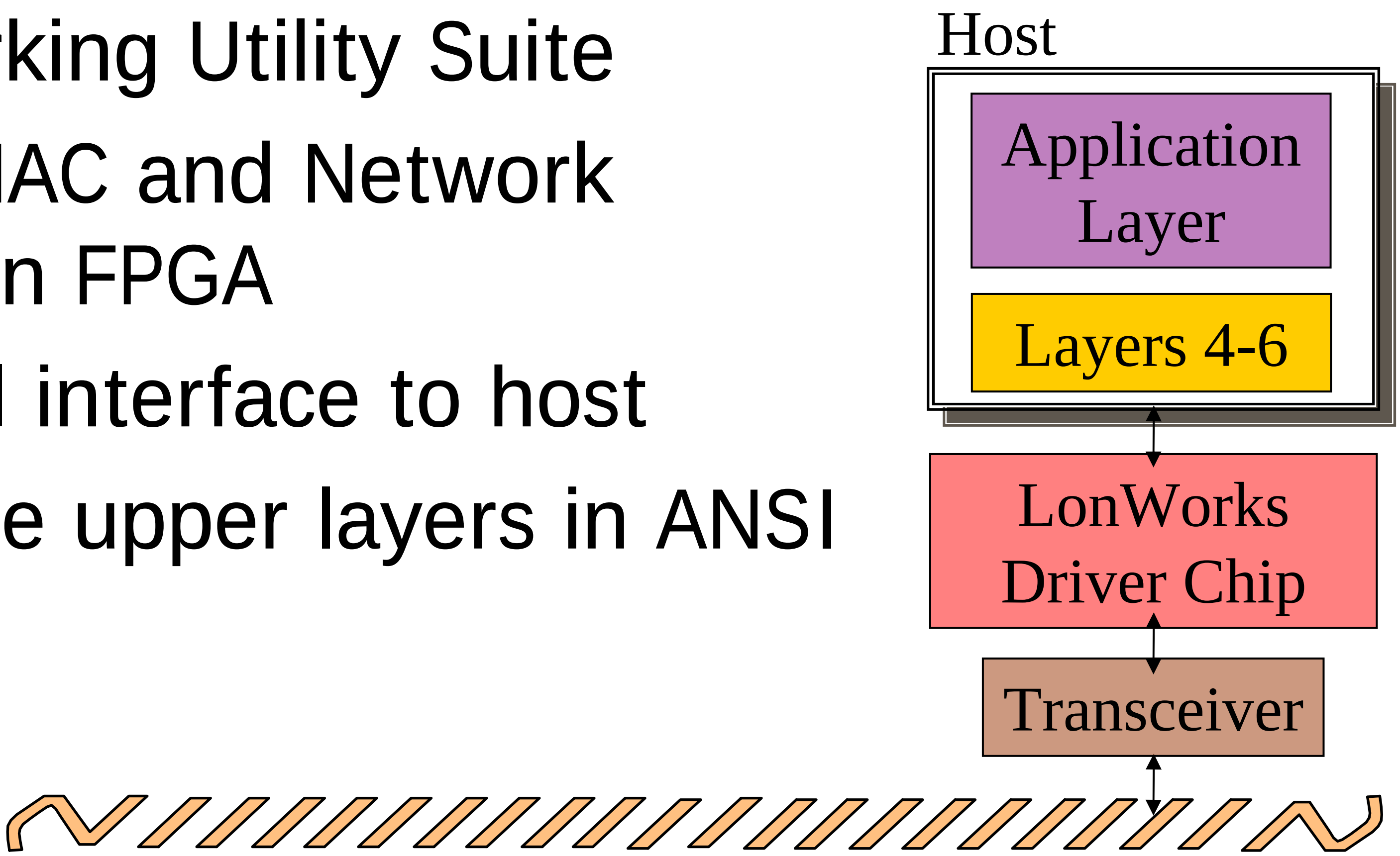
Adept Systems

- MC68360 microprocessor
- Upper layers in portable ANSI C
- On-chip link layer hardware
  - Manchester encoding
  - Packet framing
  - CRC generation/detection
- Contact: 1-561-487-6894



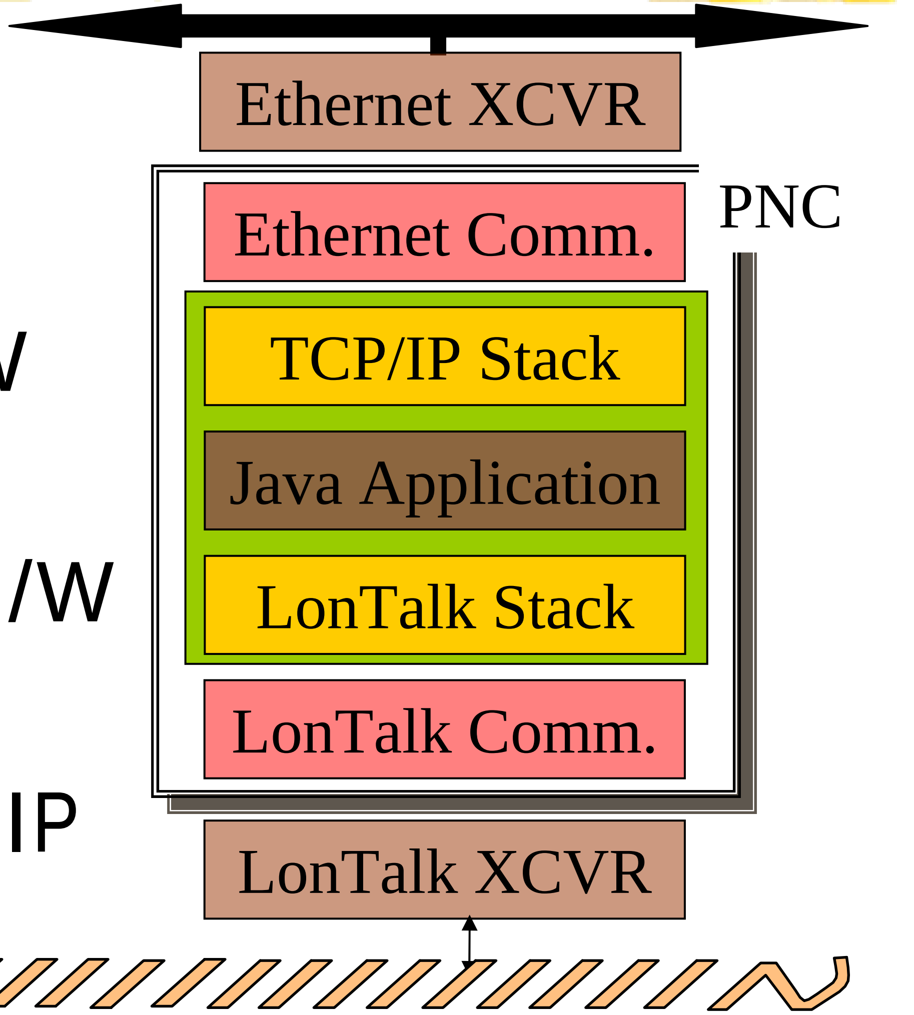
Loytec Electronics

- VENUS - Vienna Embedded Networking Utility Suite
- Link, MAC and Network layers in FPGA
- Parallel interface to host
- Portable upper layers in ANSI C



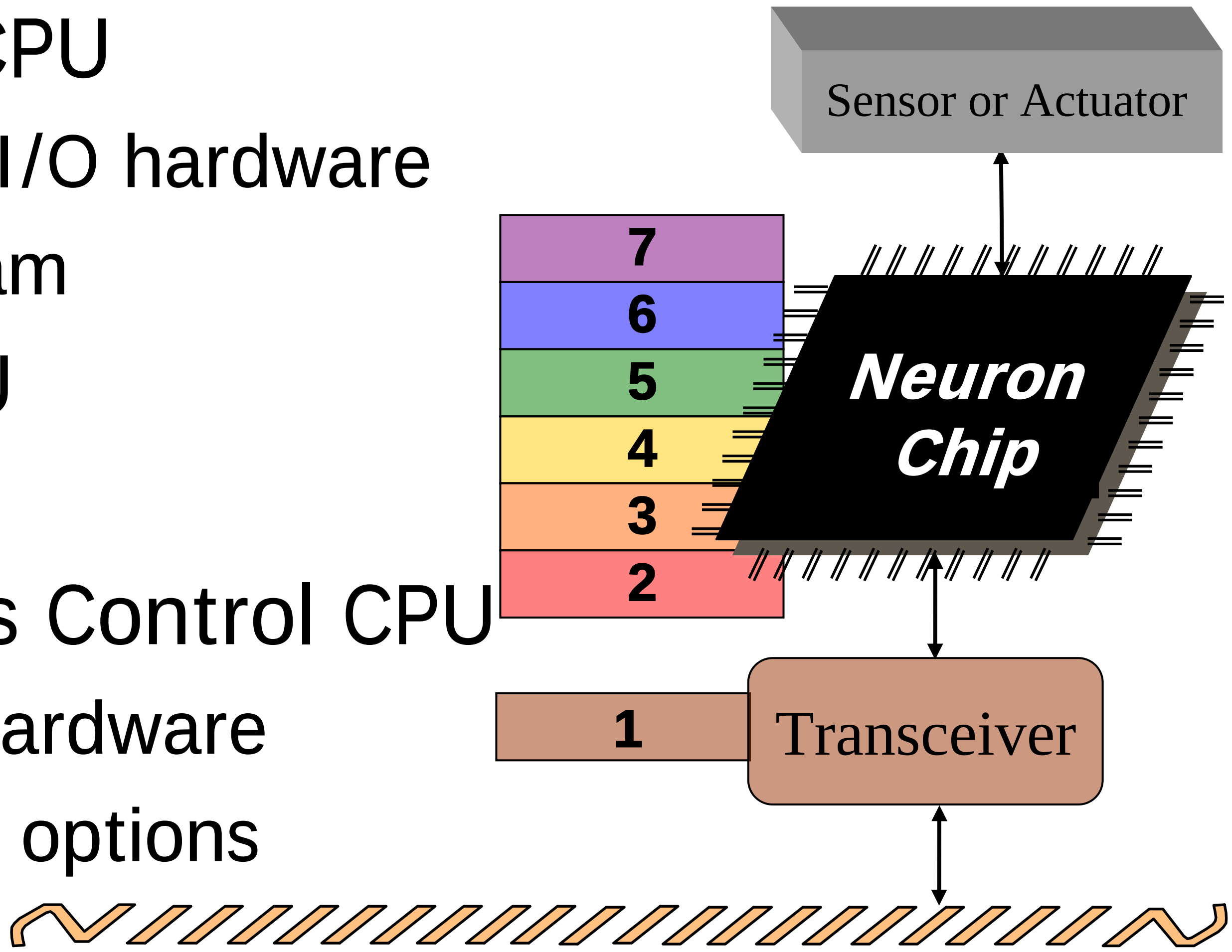
Toshiba Process Network Computer Chip

- MIPS RISC core
  - Running Java OS
- LonTalk Link Layer H/W
  - LonTalk stack in Java
- 10-base-T Link Layer H/W
  - TCP/IP stack in Java
- Supports LonTalk over IP



Cypress / Toshiba Neuron Chip

- Application CPU
  - Embedded I/O hardware
  - User program
- Network CPU
  - Layers 3-6
- Media Access Control CPU
  - Link layer hardware
  - Transceiver options



Interoperability is Primary

- Implementations include all defined layers
  - No advantage in implementing a subset
- Application layer interoperability
  - LonMark Interoperability Association
    - **`http://www.lonmark.org`**
  - Industry-specific working groups
    - HVAC, industrial, security, lighting, etc.
  - Application-layer object definitions for higher-level interoperability

Node Cost is Also Primary

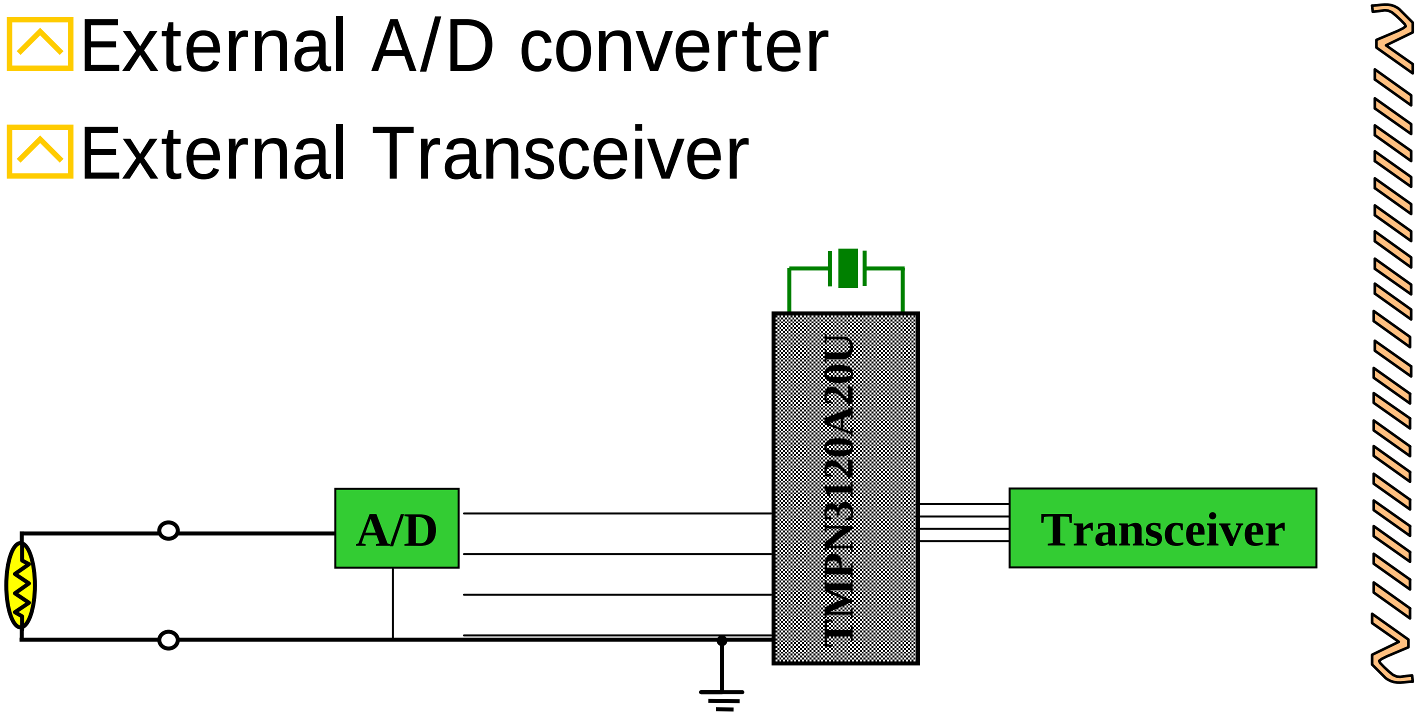
- Widest possible adoption
  - High volume silicon ➔ lowest cost
  - Neuron Chip currently *under $3.00*
- Generic control networking solution
- Industry-specific extensions
  - Transceivers with price/performance tradeoffs
  - Packaging and wiring specifications
  - Application objects for functional interoperability

Why Define All Protocol Layers?

- It guarantees interoperability across device manufacturers
- It *greatly* simplifies node design
  - Developer writes to high-level API
- In a $5 device, memory is not "free"
  - 1-2KB of EEPROM
  - 1-2KB RAM
  - 10-16KB ROM with system firmware

Temperature Sensor Example

- LonMark-compliant device
- Implemented with Neuron 3120 Chip
  - External A/D converter
  - External Transceiver



Memory Budget for Temperature Sensor

⌘EEPROM Usage	344 bytes
System Data & Parameters	69 bytes
Network Configuration Data	117 bytes
Application EEPROM Variables	7 bytes
Application Code	**91 bytes**
Self-Identification Data	60 bytes
⌘RAM Usage	841 bytes
System Data & Parameters	457 bytes
Protocol Buffers	382 bytes
Application RAM Variables	2 bytes

⌘System firmware in on-chip ROM does all the hard stuff

This is Not a Toy Device!

- Sensor sample rate, calibration, and offset settable over network
- Device is self-identifying
  - Documentation may be read over the network
- Domain, subnet, and node address settable over the network
- Data destination settable over the network
  - One, many or all other nodes may receive the temperature reading

User Code Architecture of Temperature Sensor

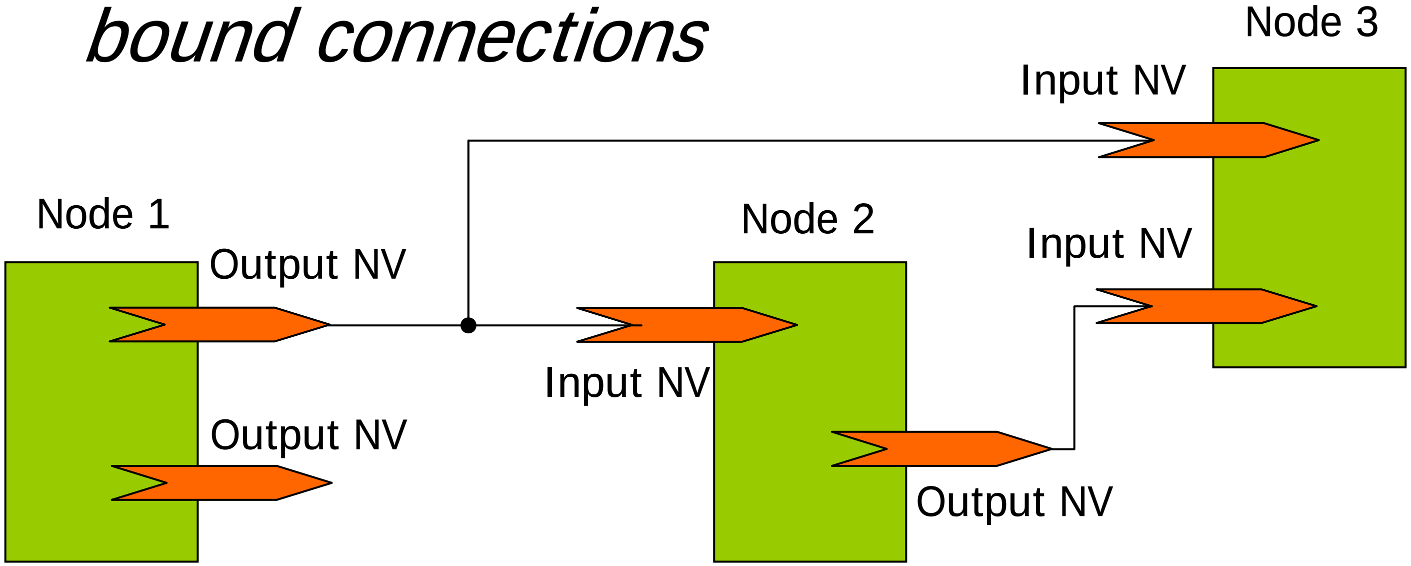
- Declare configuration parameters for sample rate, calibration, and offset
- Declare output *Network Variable* for temperature measurement
- Executable code:
  - On reset, start the A/D converter
  - When A/D conversion is done, scale reading and propagate to output network variable.

Neuron C Code for Temperature Sensor

```
#include <stdlib.h>
#include "a2d.h"

//   Declare node-level self-documentation

#pragma set_node_sd_string "&3.0@1Temp Snsr"

//   Declare sensor output network variable

network output sd_string("@1|1") SNVT_temp nvoValue;

//   Declare sensor configuration parameters

config network input sd_string("&0,5,0\x80,26") SNVT_temp nciOffset;
config network input sd_string("&0,1,0\x80,31") SNVT_muldiv nciGain;
config network input int nciSampleRate;

//   Reset task - initialize A/D converter

when( reset ) {
    a2d_enable(nciSampleRate);
    a2d_mux(0);
}

//   A/D conversion complete task - propagate network variable

when( a2d_done() ) {              // fixed point linear scaling
    nvoValue = muldiv(a2d_read(), nciGain.multiplier, nciGain.divisor) + nciOffset;
}
```

Network Variables - NVs

- Application layer abstraction for data sharing
  - Multiple addressable data entities per device
  - Implicitly addressed updates delivered via *bound connections*



Application Layer API for NVs

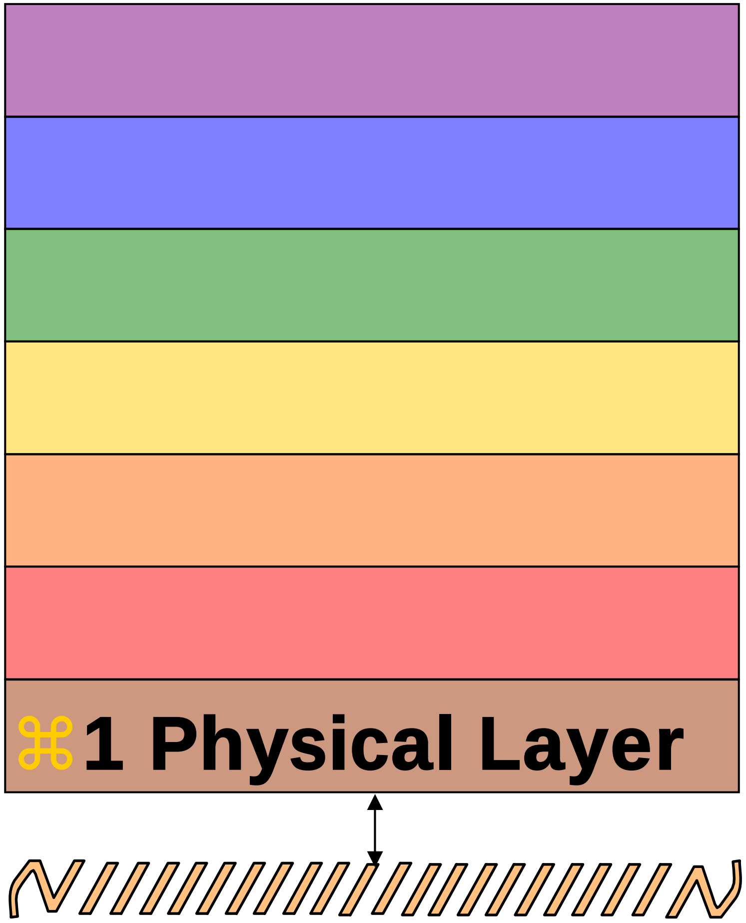
- Output network variables
  - Function
    - `_nv_update(int index, void *pValue, int len);`
  - Event handler
    - `_nv_completes(int index, boolean status);`
- Input network variables
  - Event handler
    - `_nv_update_occurs(int index, void *pValue, int len);`
  - Function
    - `_nv_poll(int index);`

Top-Down View of LonTalk

- High-level network abstraction permits very small, simple application programs
- All layers of protocol may be configured at installation time via network management protocol
- Application may override implicit configuration
  - At the cost of program size and complexity

Starting at the Bottom.....

- Physical Layer Transceivers
- Interface between digital processing and analog networking medium
- Direct Mode
- Special-Purpose Mode



Direct Mode Transceivers

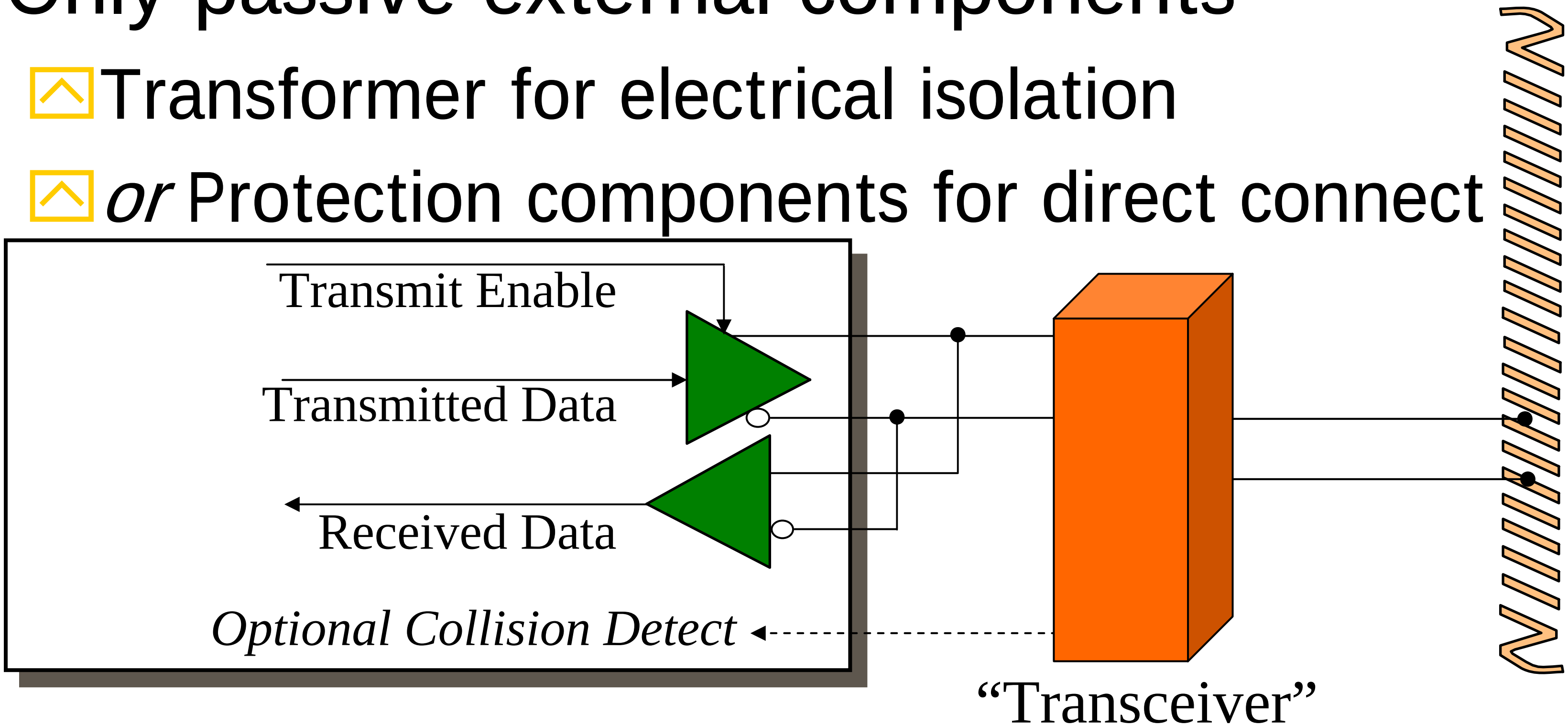
- Serial interface to Link Layer H/W
- Differential Manchester encoding
- Simple external hardware, e.g.
  - EIA-709.3 free topology TP (78kbps)
  - EIA-485 bus topology twisted pair
  - Direct connection to twisted pair
  - FSK Radio (4.9kbps)

Direct Mode Bit Rates (kbps)

2,500
1,250
625
312.5
156
78
39
19.5
9.8
4.9
2.4
1.2
0.6

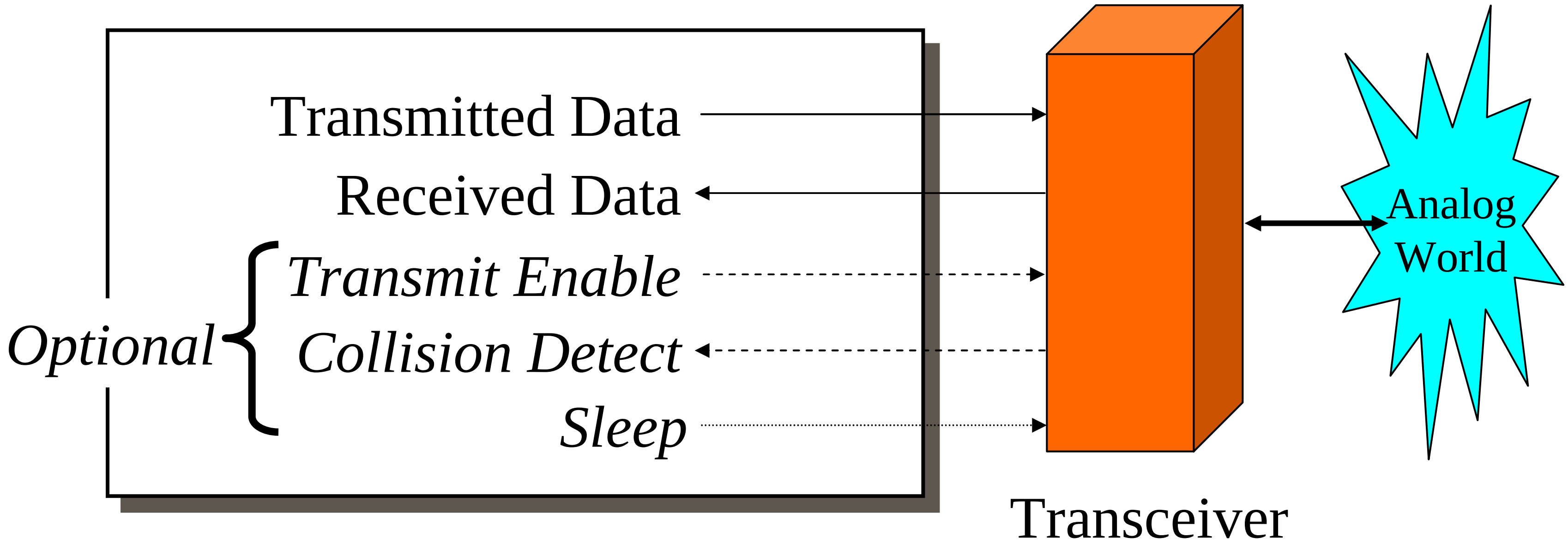
Differential Direct Mode

- Analog interface
  - Programmable hysteresis, glitch filtering
- Only passive external components
  - Transformer for electrical isolation
  - *or* Protection components for direct connect



Single-Ended Direct Mode

- Used with active external transceivers
  - e.g. EIA-709.3, RS-485, FSK radios
- Simple digital interface

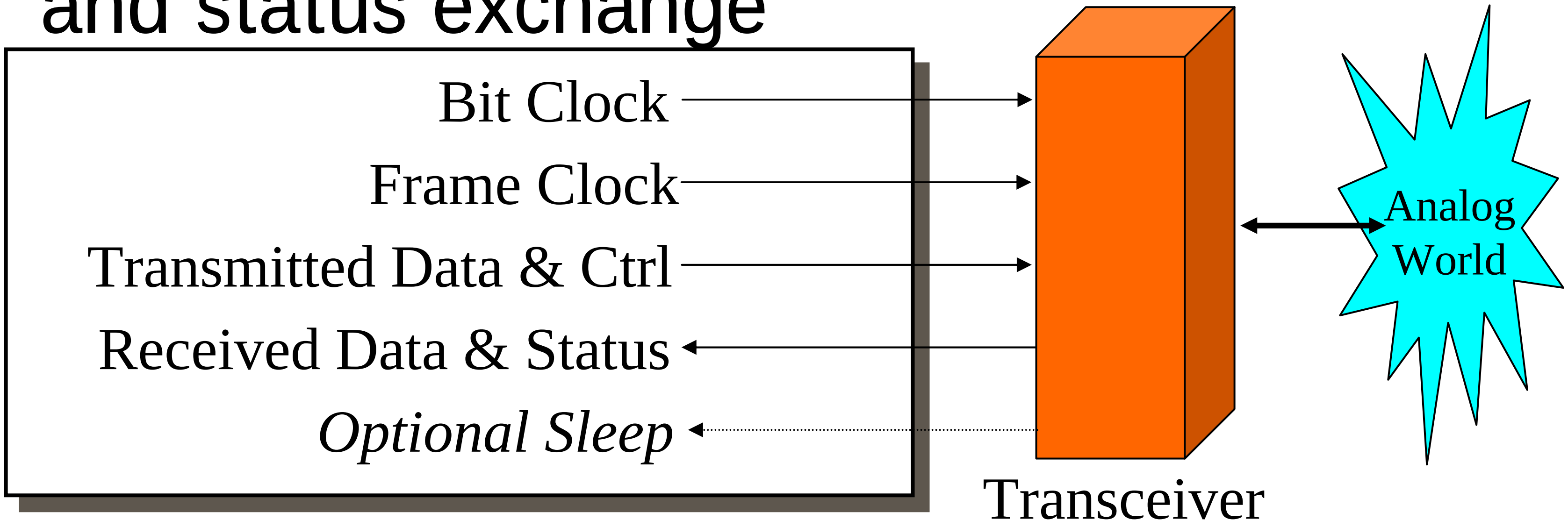


Special-Purpose Mode Transceivers

- Digital handshaking interface to MAC layer
- Transceiver provides data encoding and modulation, controls bit rate
- Possible features include
  - Error detection and correction
  - Collision detection and resolution
  - Tunneling over foreign protocols
- Example: EIA-709.2 power line

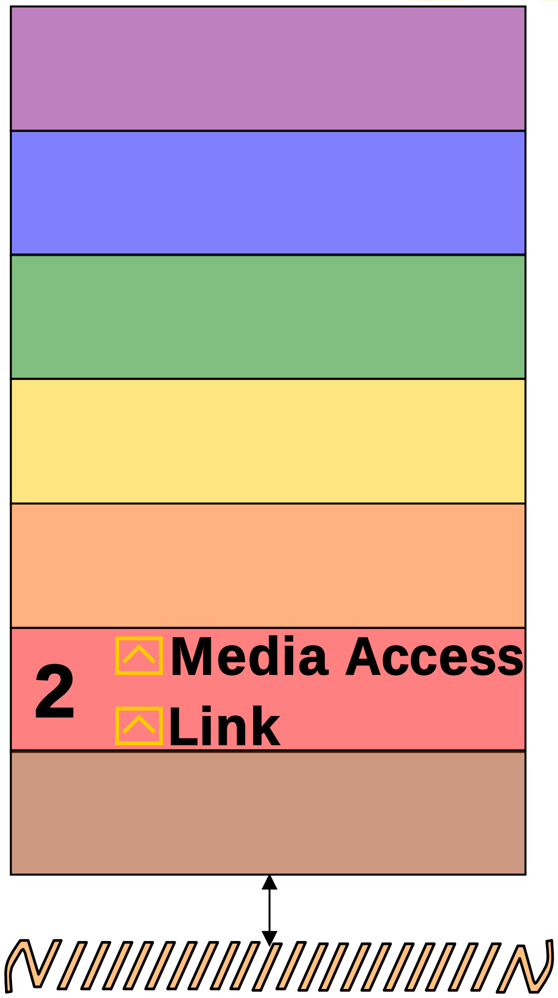
Special-Purpose Mode Transceiver Interface

- Digital handshaking interface
- Intelligent transceivers
- Defined protocols for data, control, and status exchange



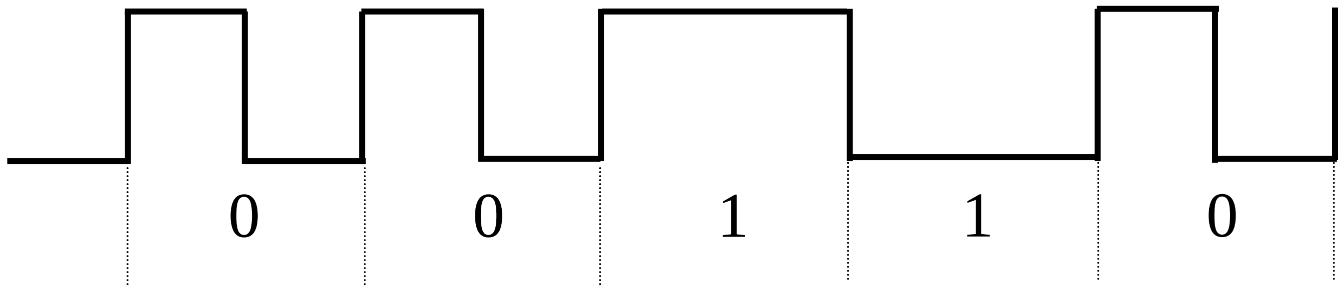
Up to the Next Layer

- Link sub-layer
  - Bit encoding
  - Packet framing
  - Packet error detection
- Media access control sub-layer
  - Sharing the bandwidth among transmitters
  - Peer-to-peer, multi-drop, priority access, collision avoidance



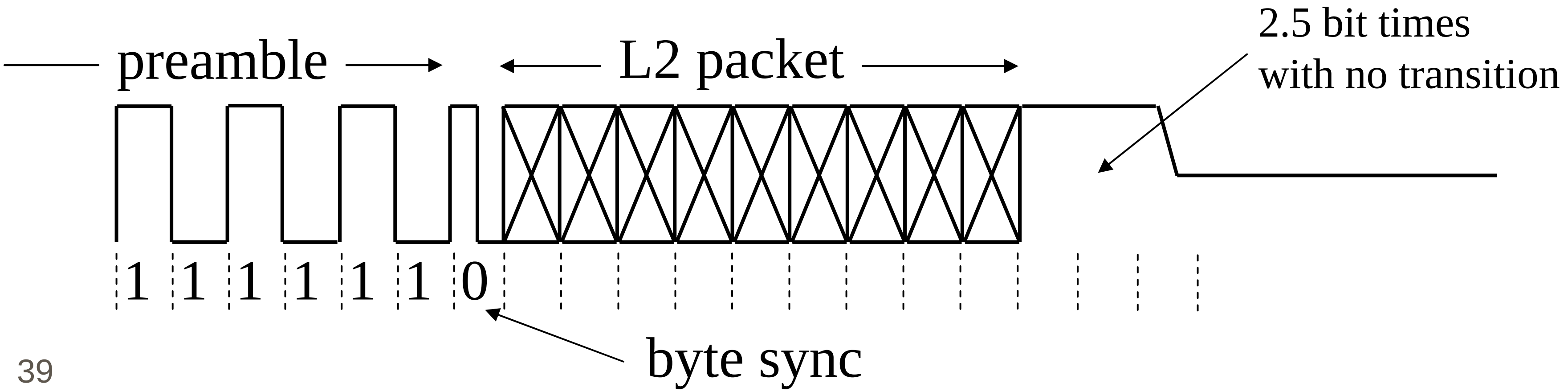
Differential Manchester Encoding for Direct Mode

- Self-clocking serial bit stream
  - Mid-bit transition indicates a “0”
- Polarity insensitive: avoids wiring problems
- Zero average DC level
  - May be transformer-coupled



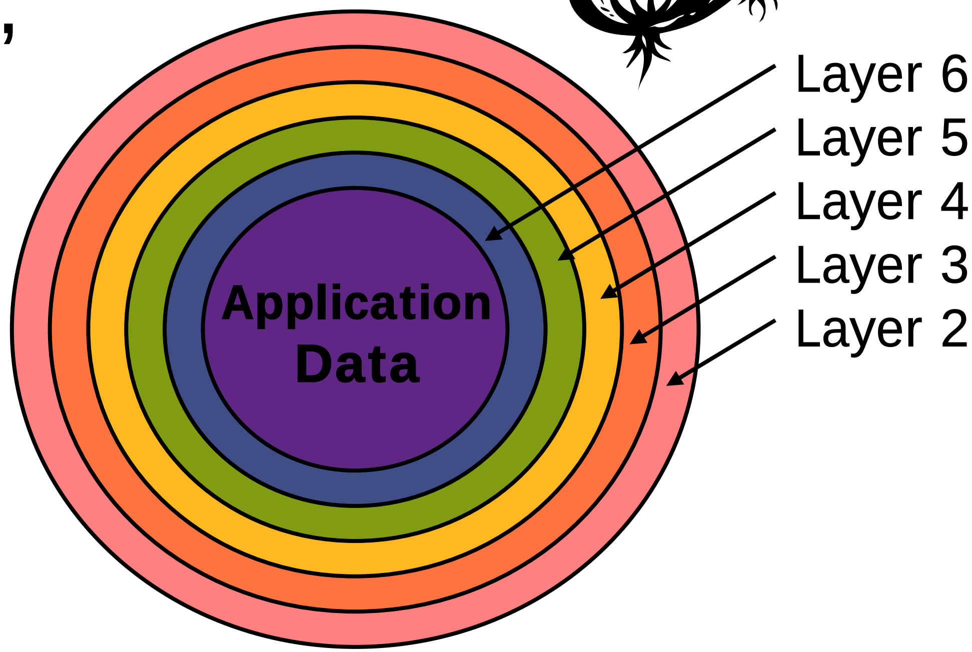
Packet Framing

- Preamble is a sequence of 1 bits
- Byte sync is a single 0 bit
- Followed by up to 256 bytes of L2 data
  - Most significant bit first
- Packet ends with Manchester code violation



Protocol Overhead

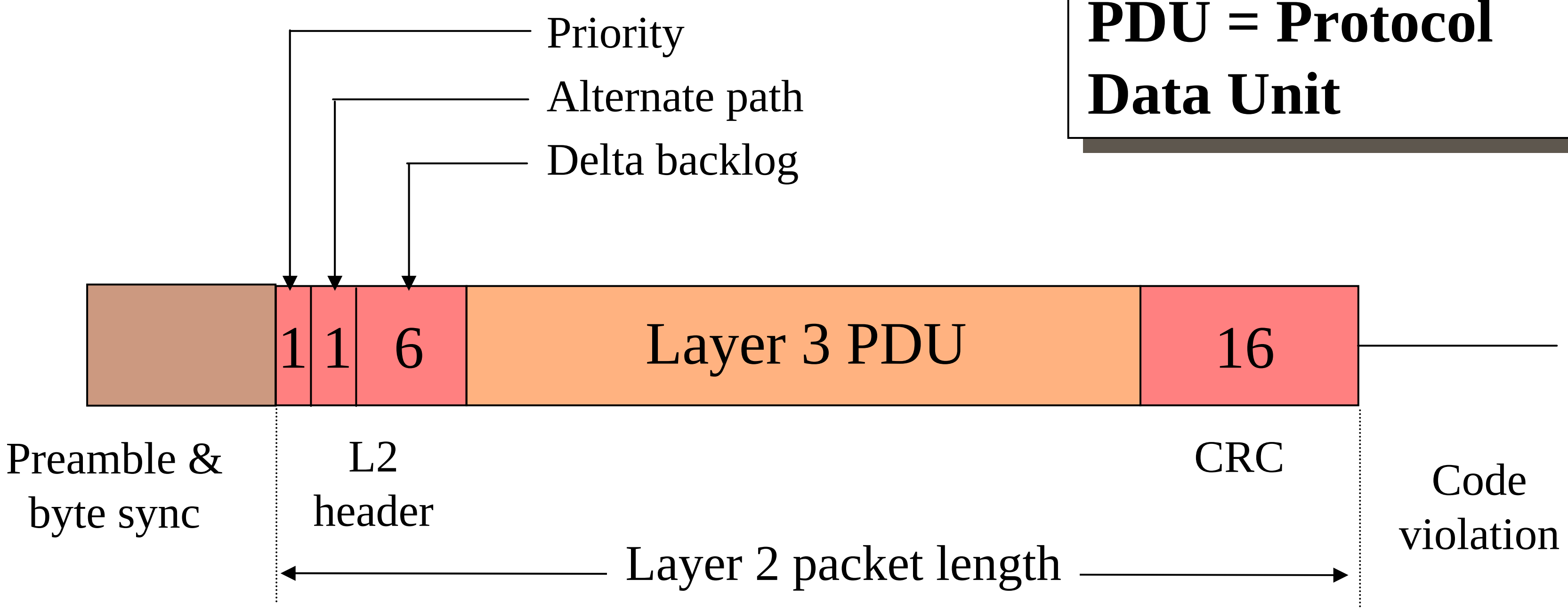
- Each layer adds its own header to the information in the packet
- In a control protocol, the application data is often small
  - e.g. on/off = 1 bit



Layer 2 Header/Trailer

- Layer 2 header is first byte of packet
- CRC is last two bytes of L2 packet
  - CCITT CRC-16 algorithm

PDU = Protocol Data Unit



Priority Bit in Layer 2 Header

- Transmitter may use priority media access algorithm
- Priority slot assignment of transmitting node is a MAC layer parameter
- Priority bit in packet ensures that routers forward the packet using the priority media access algorithm

Alternate Path Bit in Layer 2 Header

- Transaction layer sets this bit for the last two retries of an acknowledged or request message
- Informs an intelligent transceiver when to use a fallback mechanism
  - Example: Noisy power line communications
  - Quiet line: use faster data encoding
  - Noisy line: use slower, more reliable encoding if previous attempts failed

Delta Backlog Field in Layer 2 Header

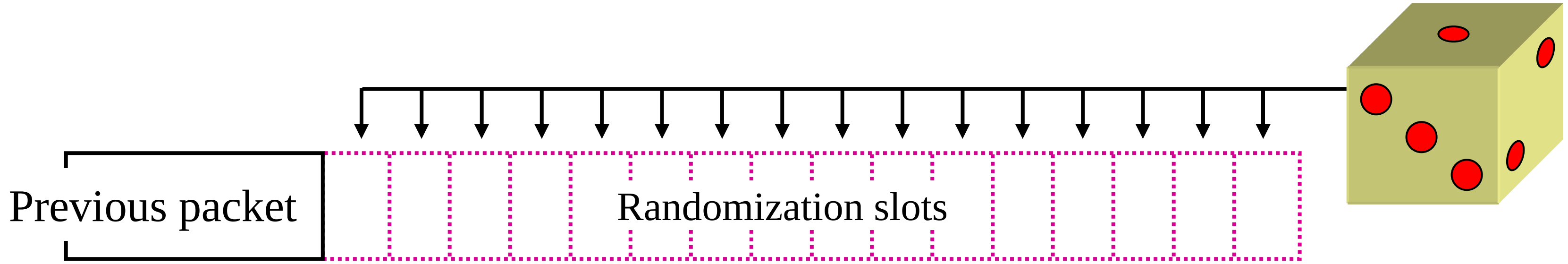
- This field informs the other nodes on the channel of the expected number of packets caused by the transmission of this packet
- Value is non-zero when sending acknowledged or request messages
- For multicast, set to *(group size - 1)*
- For broadcast, set to number of nodes in destination subnet

Media Access Algorithms Used by Other Protocols

- Algorithms with no possibility of collision use bandwidth inefficiently
- Response time increases with node count
- Master-slave polling
  - Single point of failure
- Time division multiplexing
- Token passing
  - Recovery of lost token takes time

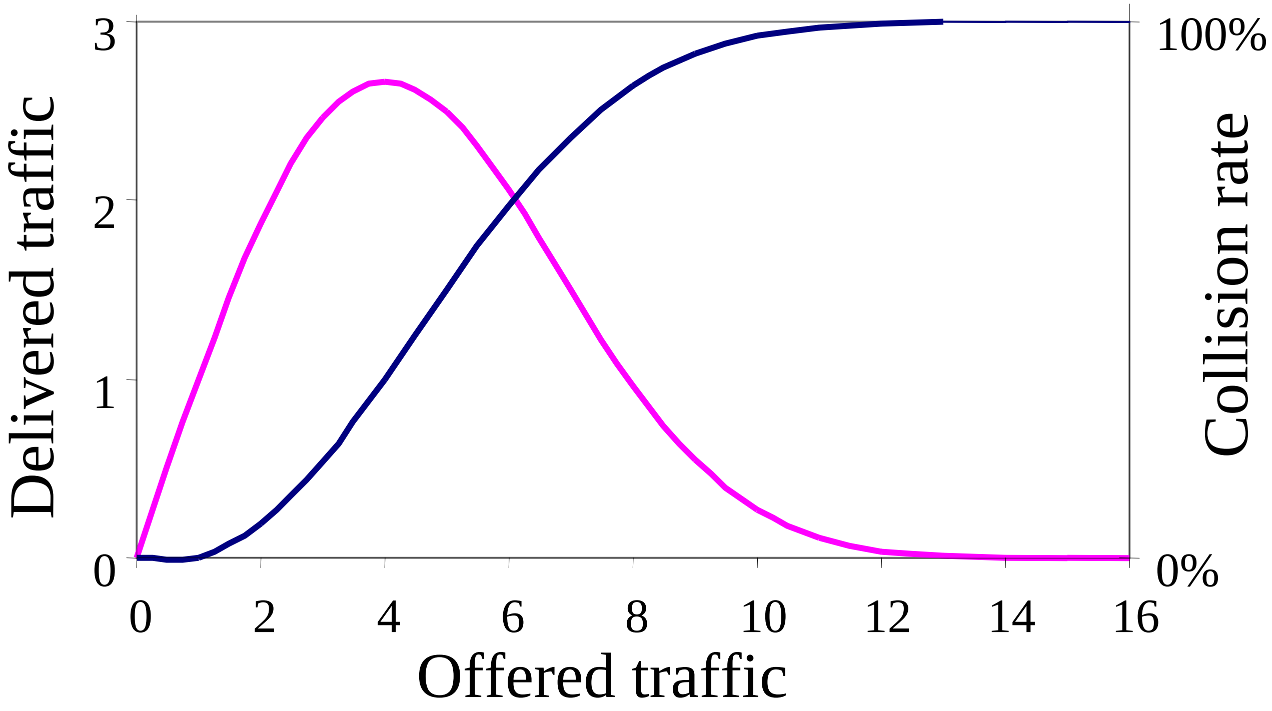
Carrier Sense Multiple Access (CSMA)

- Used by multi-drop Ethernet
- Sender waits for random number of time slots before trying to transmit
- If another node is already transmitting, sender backs off until next cycle
- Efficient use of bandwidth when traffic is low



Pure CSMA Chokes Under Load

- At high offered traffic, less gets through
- Example: 16 randomization slots

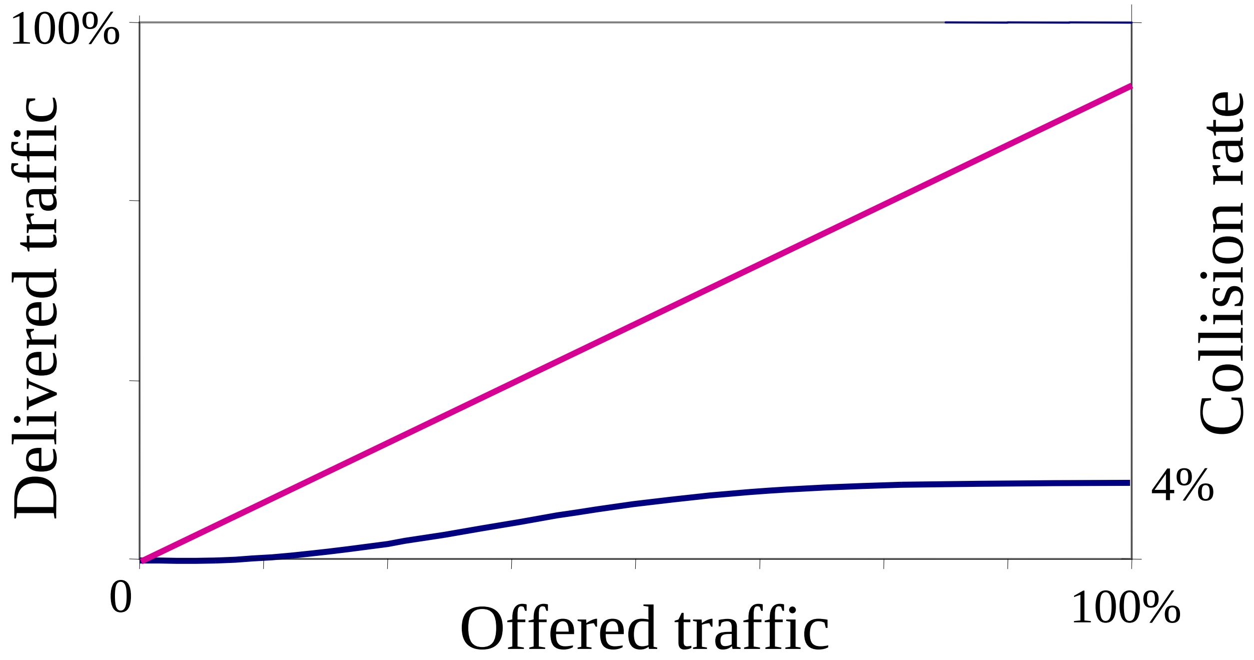


Modified p-Persistent CSMA

- Number of randomization slots is increased as traffic increases
- Delta backlog field in Layer 2 header updates offered traffic estimate on receiving nodes
  - Details of the algorithm in protocol specification document

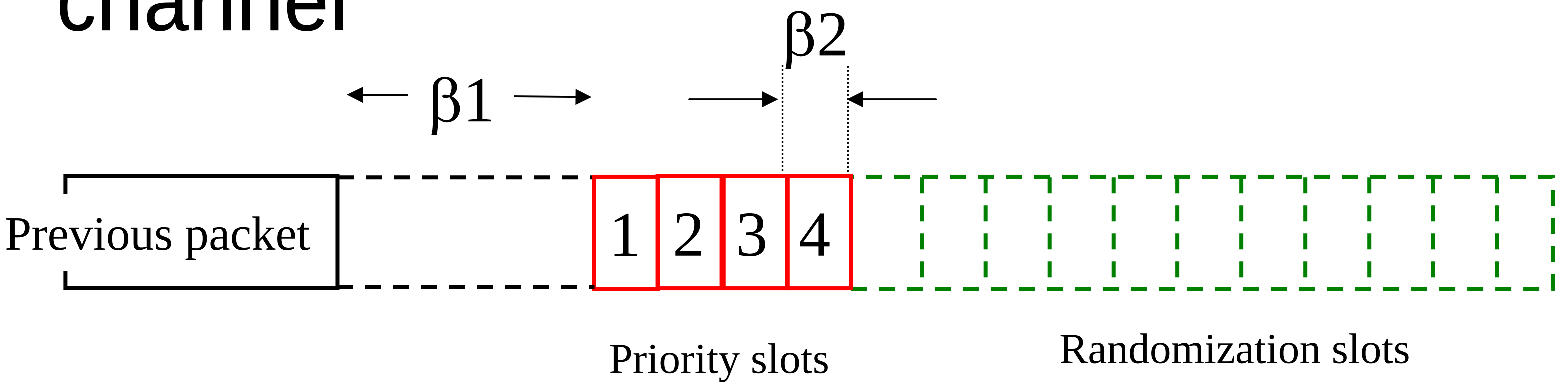
Maximum Collision Rate is Limited

- In practice, never more than 4% collisions
- Delivered traffic *increases* with offered traffic

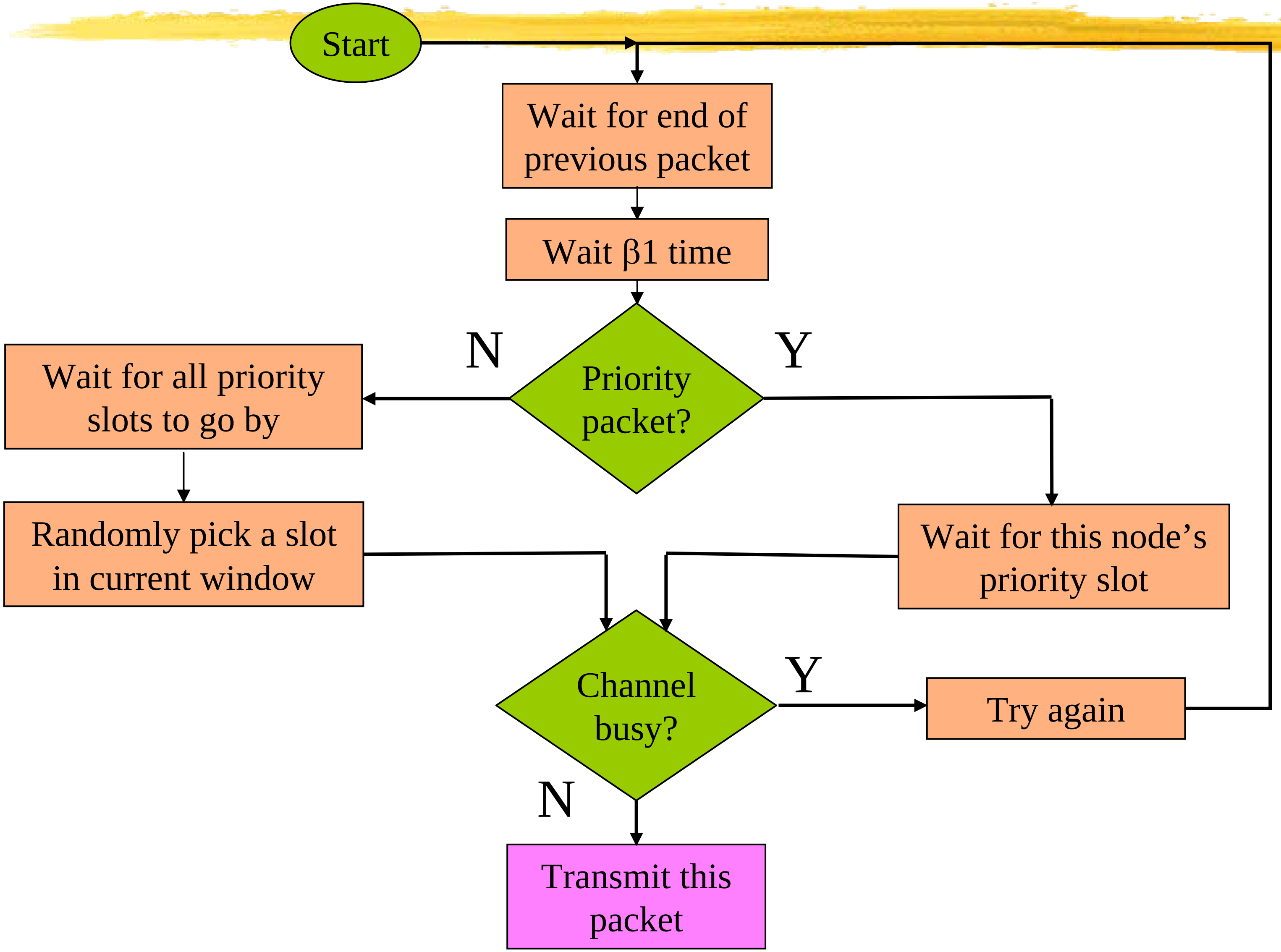


Priority Channel Access

- Media bandwidth dedicated to priority messages
- Each node that requires priority access is allocated a unique slot number on its channel



Media Access Algorithm for Transmitter



Number of Priority Slots

- Priority slots provide dedicated bandwidth for important messages
- If all packets use priority slots, there are no collisions
- The more slots there are, the wider Beta 2 time must be
- More slots ➔ worse overall bandwidth utilization

Factors Influencing Beta 1 Time

- Media propagation delays
  - 186,000 miles/second isn't just a good idea, it's the LAW
  - Physical-layer repeaters
- Transceiver turnaround delays
  - Dissipation of transmitter energy before receiver can operate
- Node response time
  - Slowest node on channel determines minimum Beta 1 time

Factors Influencing Beta 2 Time

- All nodes must have a consistent view of slot timing
- Transmit/receive clock accuracy
  - Usually requires 200 ppm crystal oscillator
- Node response time jitter
  - Jitter of slowest node on channel determines Beta 2 slot width
- Beta 2 width increases with number of priority slots

Optional Collision Detection

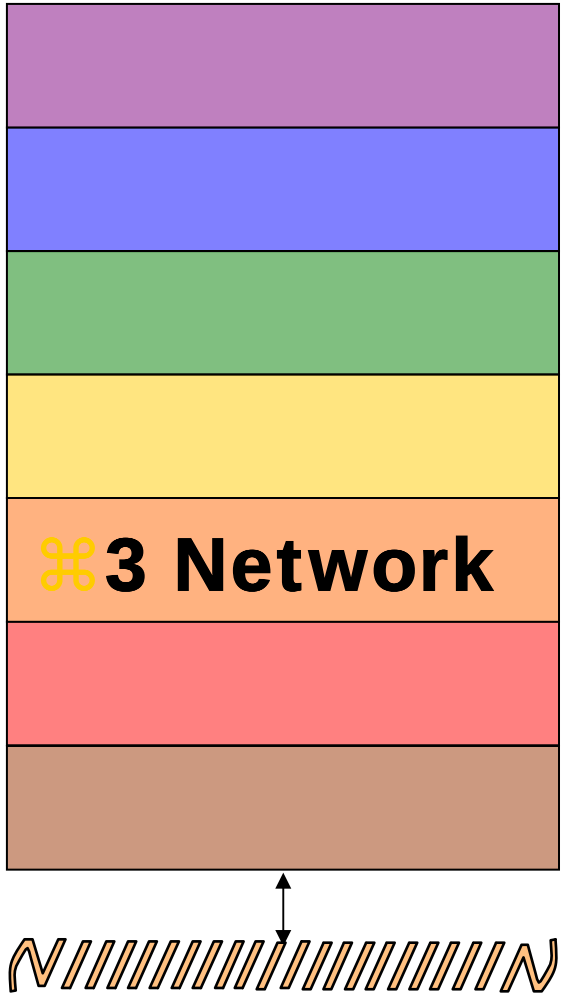
- Transceiver optionally signals transmitter when collision is detected
- MAC layer immediately ceases transmission and reschedules
- Difficult to do in a low-cost, high speed transceiver
- If *Cdet* is not implemented, transaction layer will recover from errors

LonTalk Media Access Summary

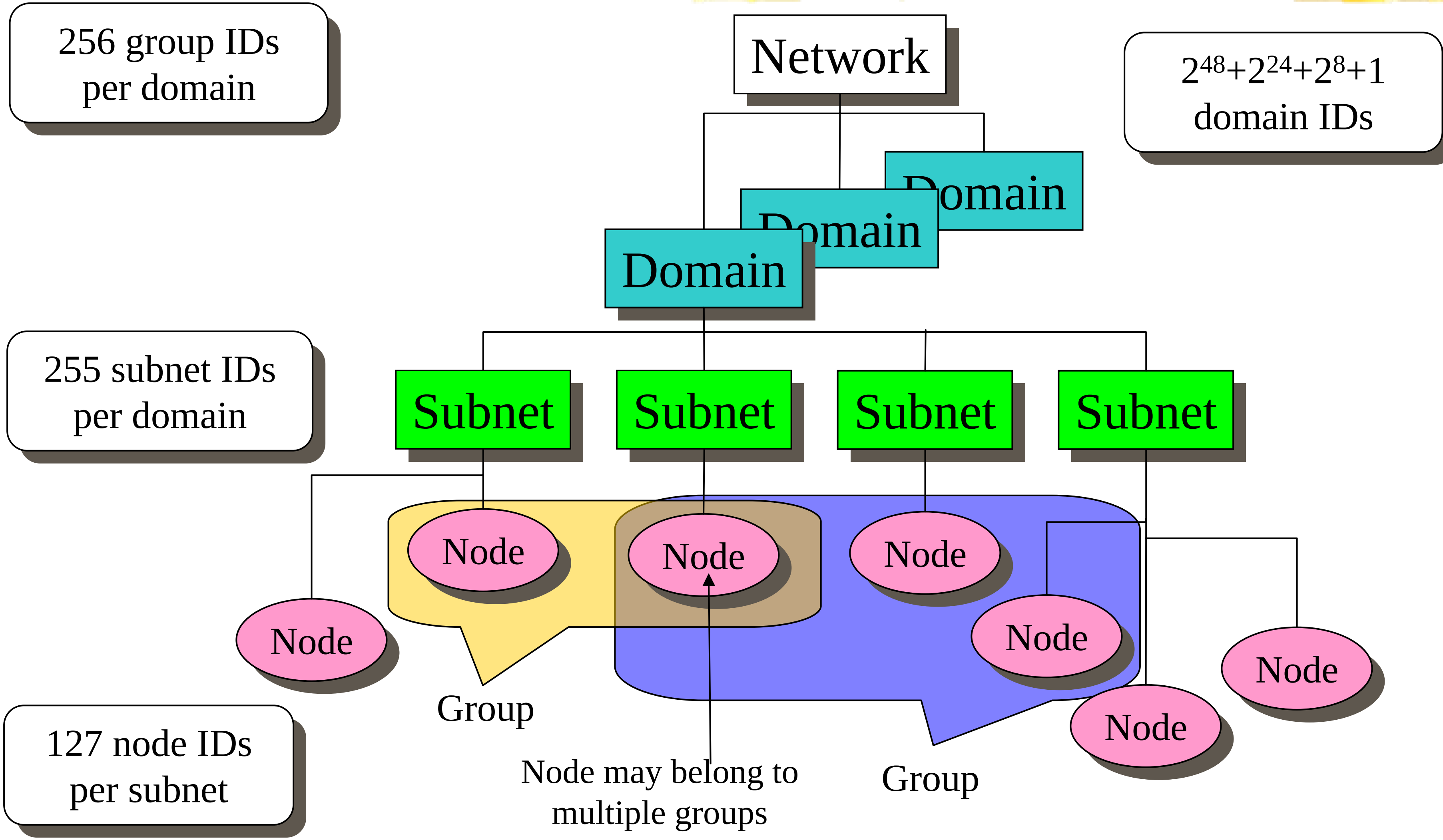
- Efficient use of available bandwidth
- Adaptive CSMA algorithm limits packet collision rate
- Priority access mechanism for alarms etc.
- Media independent
  - TP, RF, CX, FO, PL, IR etc.
- LonMark Interoperability Guidelines define a set of standard channels

Protocol Layer 3

- Message addressing
  - Unicast - single node
  - Multi-cast - group of nodes
  - Broadcast - subnet-wide or domain-wide
- Routing between subnets
  - Configured routers
  - Learning routers
  - Repeaters



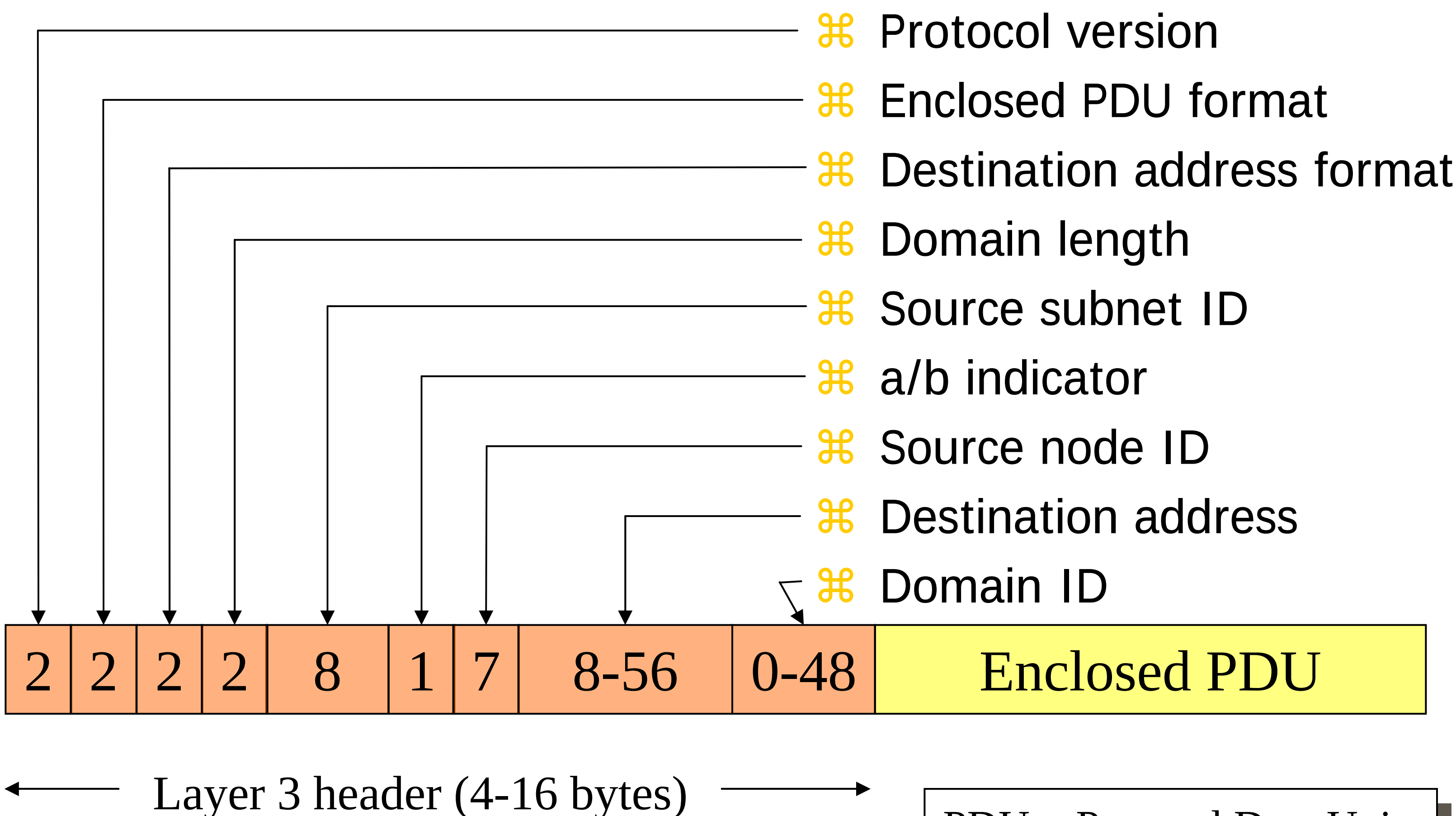
Logical Addressing Hierarchy



LonTalk Address Rules

- A configured device may belong to one or more domains
  - It sends and receives messages only in these domains, *except*
- A device can also receive a message outside of its domain if:
  - It is not configured in any domain *and* the destination address mode is broadcast
  - *Or* the destination address specifies the device's unique ID

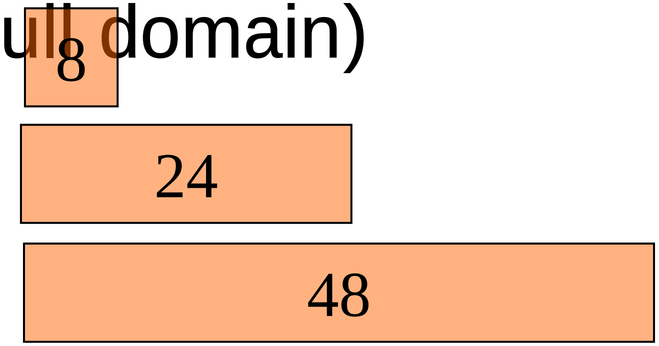
Layer 3 Protocol Data Unit



PDU = Protocol Data Unit

Network Layer Fields

- Protocol version
  - Currently at version 0
  - No revisions have been needed, or are planned
- Selectable domain ID length
  - 0 = 0 bytes (the null domain)
  - 1 = 1 byte
  - 2 = 3 bytes
  - 3 = 6 bytes



Using Domain IDs

- Domain ID identifies subsystem
- Application node may be configured in one or more domains
- Use 0 or 1 byte domain IDs for closed subsystems
  - Shorter packets
- Use unique domain IDs on open media
  - Example: power line

Using Subnet and Node IDs

- ⌘The destination subnet ID is used for routing the packet in multi-channel networks
- ⌘The destination node ID identifies the node within its subnet
- ⌘The receiving node uses the source subnet and node IDs to address any acknowledgement and response messages

Multicast and Broadcast Destination Addressing

- Address format 0 = Broadcast to Subnet
  - Destination address field is the subnet ID
  - 8 bits (1-255)

8

  - If destination subnet is 0, it means all subnets in the domain
- Address format 1 = Multicast (group)
  - Destination address field is the group ID
  - 8 bits (0-255)

8

Unicast Destination Addressing

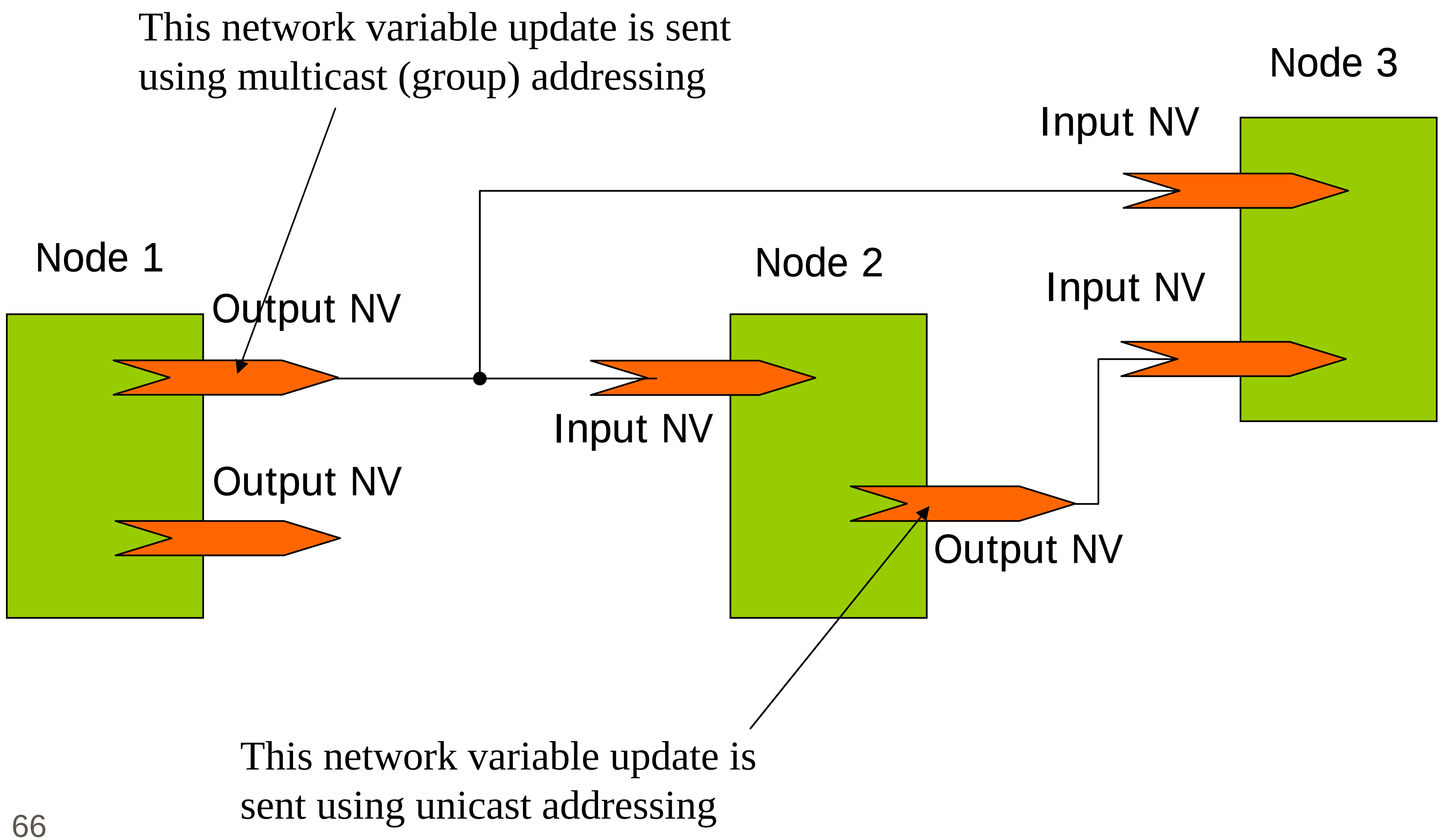
- Address format 2 = Unicast (subnet/node)
- Destination address is subnet and node IDs
  - Format 2a: 16 bits

8	1	7
subnet	1	node

  - Format 2b: 24 bits, used for group ACK's and responses

8	1	7	8	8
subnet	1	node	group	*source* group member ID

Addressing Messages and Network Variables



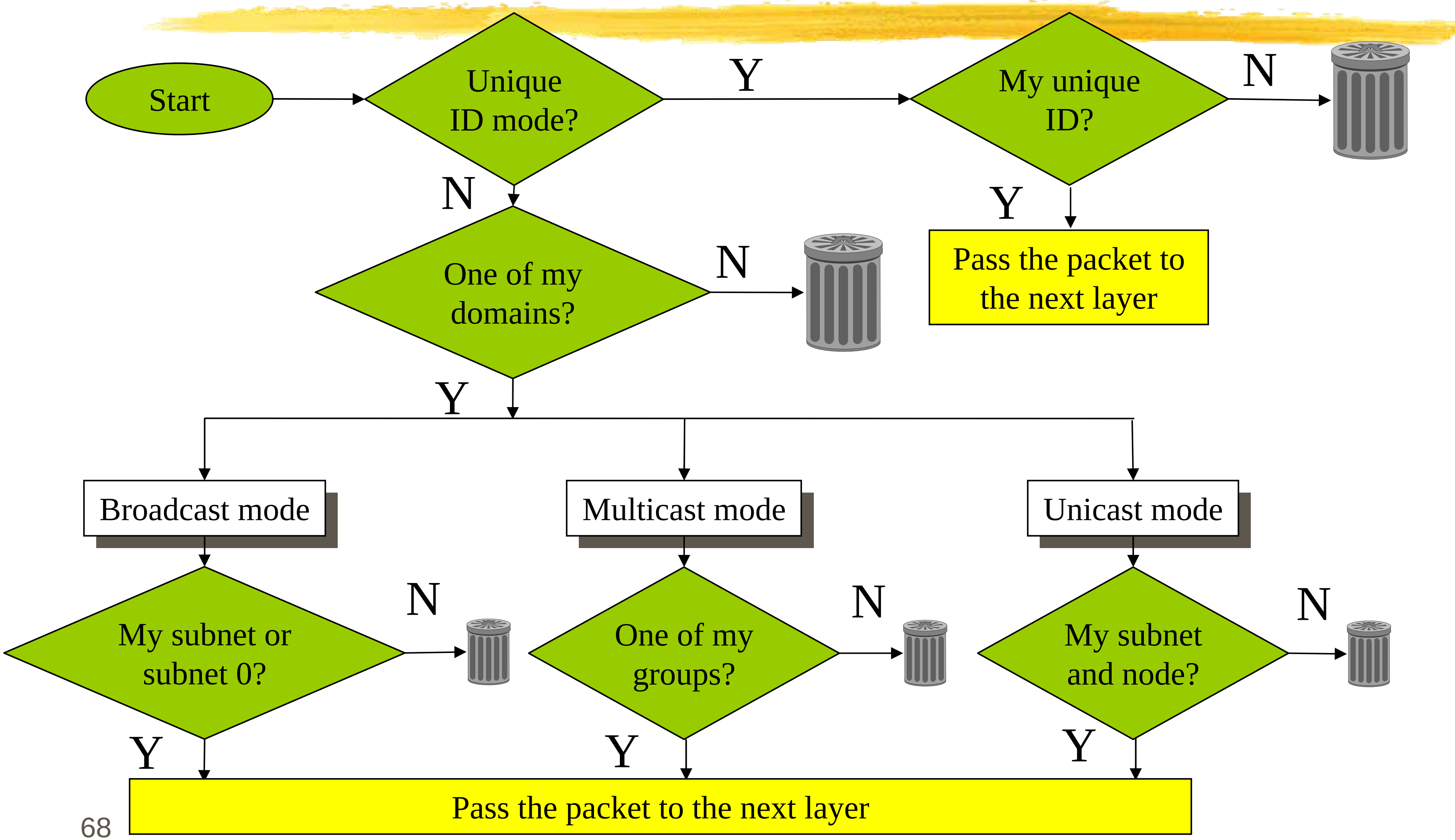
Unique ID Destination Addressing

- Address format 3
  - Destination address is subnet ID and device unique ID
  - Subnet ID used for routing

8	48
subnet	unique ID

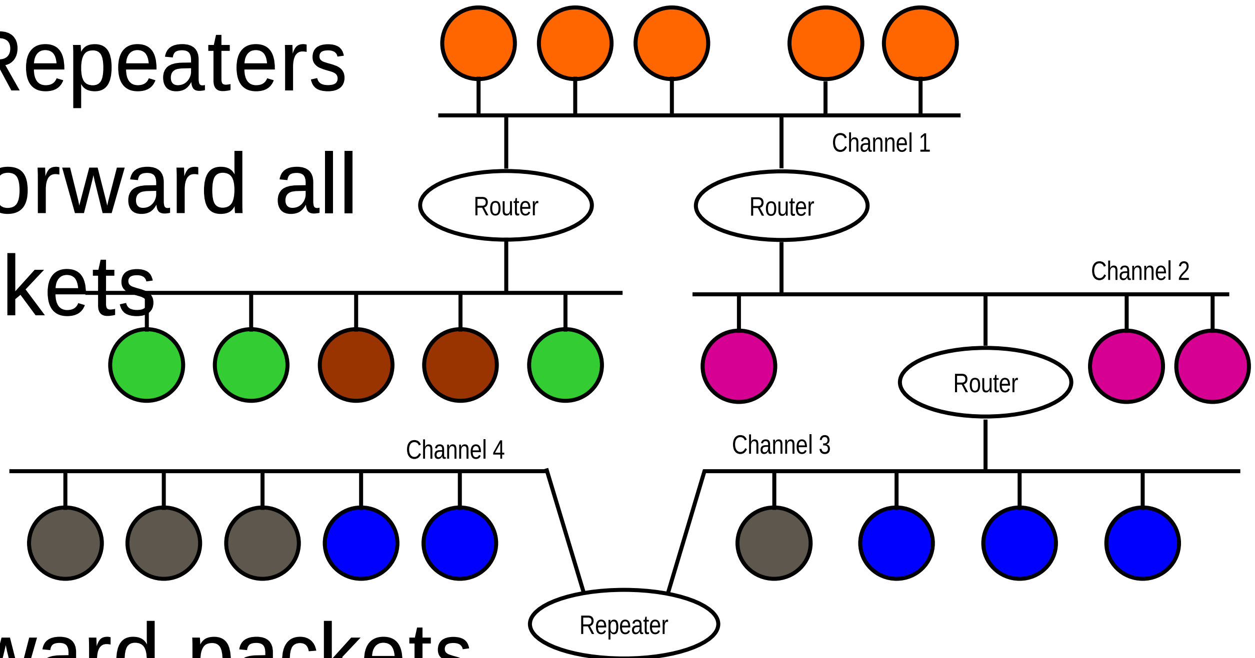
- Used to configure nodes that are not yet configured in any domain
- Not used for application messages
  - Logical rather than physical addressing allows easy device replacement

Address Recognition for a Configured Device



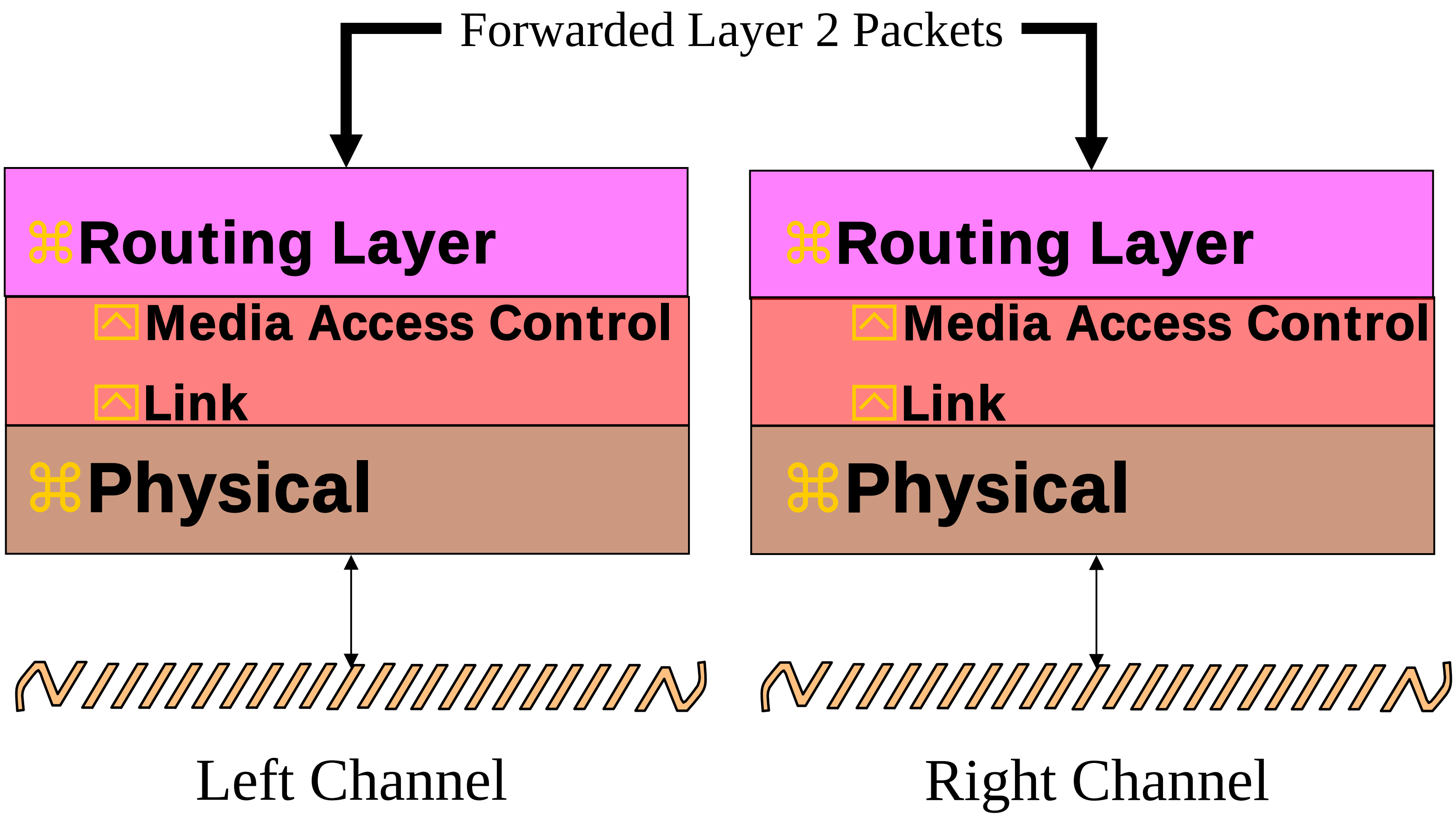
Routing in Multi-Channel Networks

- Channels connected via Routers or Repeaters
- Repeaters forward all valid packets
- Routers forward packets selectively based on destination



Architecture of a Router

- Router intelligently connects two channels



Half-Router Algorithm

- Receive layer 2 packet from channel
- If multicast, check forwarding bit for destination group
- Else check forwarding bit for destination subnet
- Is forwarding bit set?
  - Yes - forward packet to other side of router
  - No - discard this packet

Router Data Structures

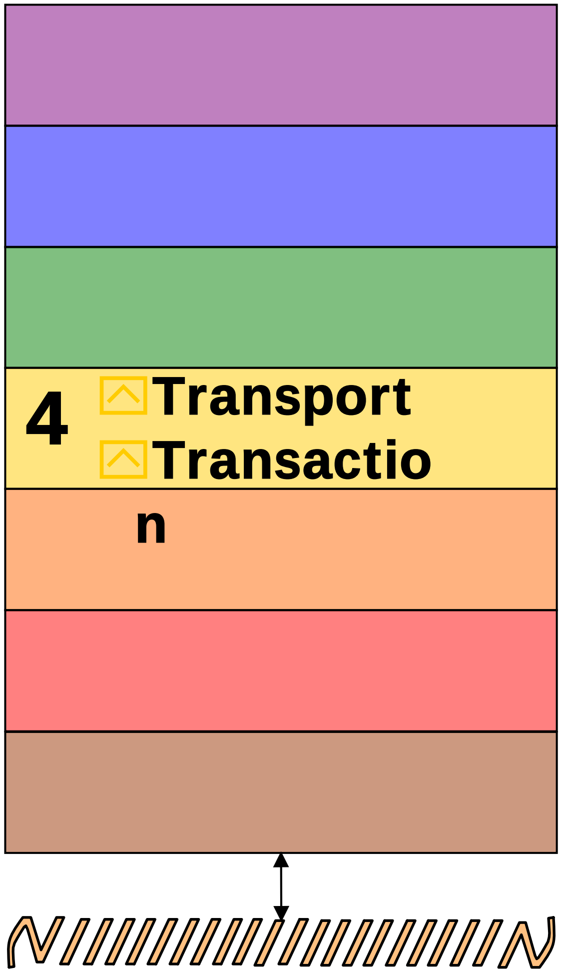
- Subnet forwarding table
  - One table per domain

- Group forwarding table
  - One table per domain

- Tables may be updated over the network by network management messages
- Learning routers build subnet forwarding table by examining *source* subnet IDs

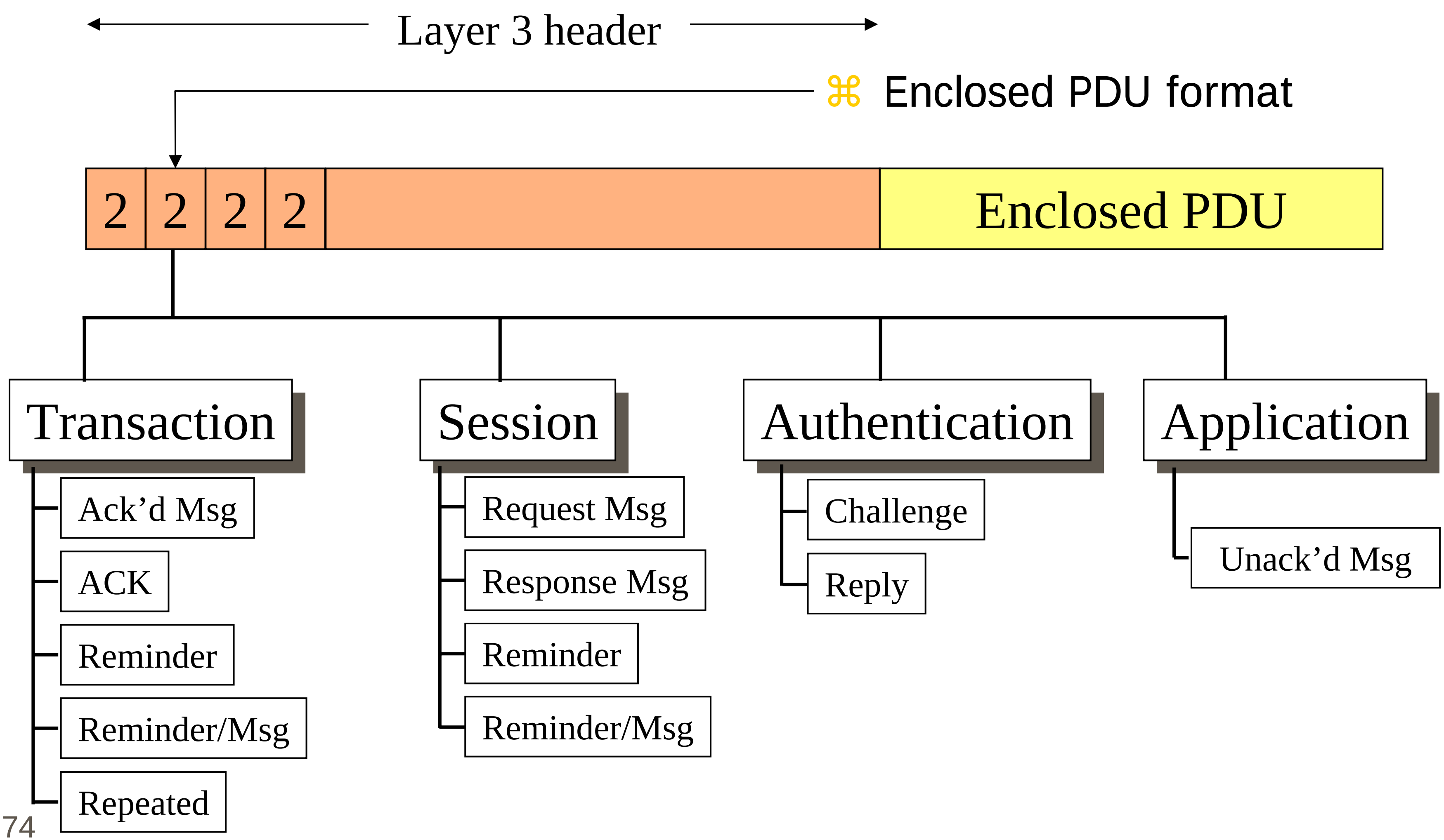
Protocol Layer 4

- Transaction control sub-layer
  - Packet ordering
  - Duplicate detection
- Authentication sub-layer
- Transport sub-layer
  - Acknowledged service
    - Unicast and multi-cast
  - Unacknowledged service
    - Repeated option



Layer 4 Packet Types

⌘ Enclosed PDU format

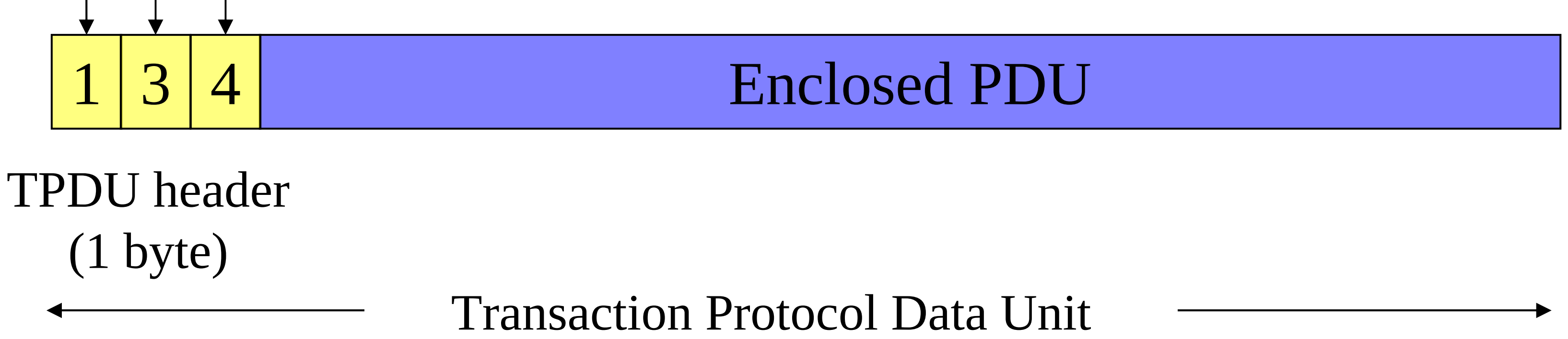


Protocol Data Unit Format

- 0 = Transaction PDU (Layer 4)
  - Acknowledged, ACK, Repeated, Reminder
- 1 = Session PDU (Layer 5)
  - Request, Response, Reminder
- 2 = Authentication PDU (Layer 4)
  - Challenge, Reply
- 3 = Application PDU (Layer 6)
  - Unacknowledged

Transaction Protocol Data Unit (PDU format 0)

- Authenticated?
- Transaction PDU type
- Transaction number

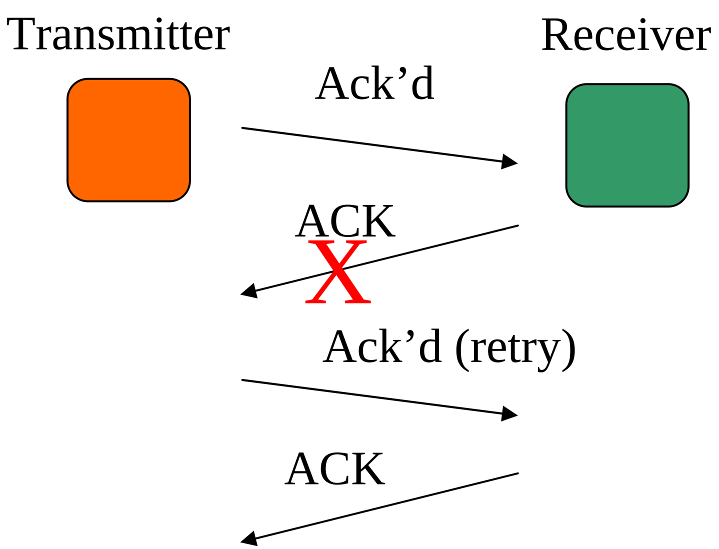


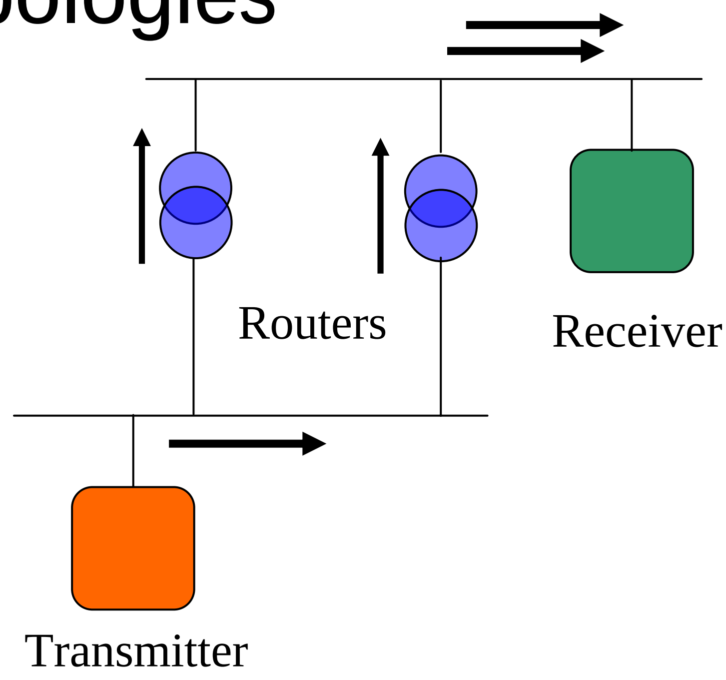
Transactions

- Preservation of ordering
  - Transmitter completes one transaction before issuing the next
  - Transaction ID is incremented by transmitter (modulo 16)
- Duplicate detection and rejection
  - Receiver checks incoming transaction ID and source address for duplicates
  - Application layer receives only one message

How Do Duplicates Occur?

- Unacknowledged/Repeated transactions
- Duplicate-generating topologies
- Lost acknowledgements



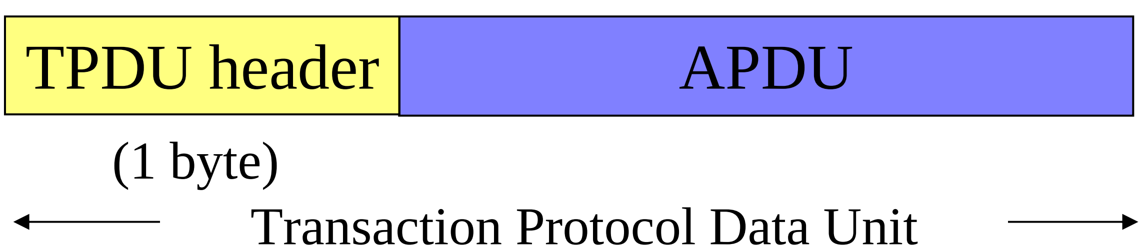


Transaction Protocol Data Unit (TPDU) Types

- Type 0: Acknowledged message
- Type 1: Unacknowledged/Repeated message
- Type 2: Acknowledgement
- Type 4: Reminder
- Type 5: Reminder with message

Unacknowledged/Repeated Message

- Transaction PDU type 1
- Enclosed PDU is Application Protocol Data Unit
- Delivered with unicast, multicast, or broadcast addressing

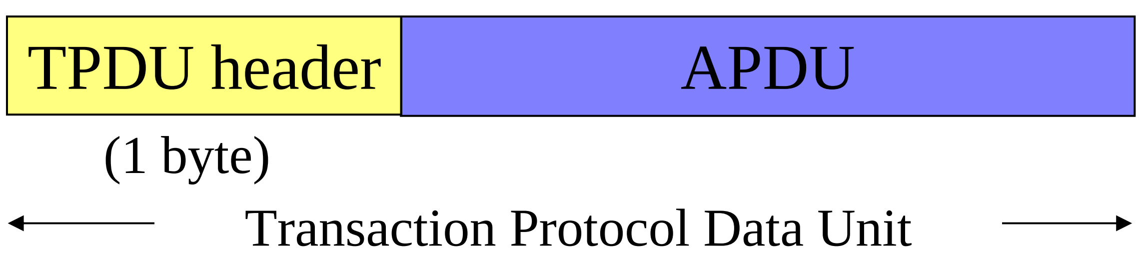


Unacknowledged/Repeated Service

- Transmitter sends application data in an Unacknowledged/Repeated packet
- Transmitter repeats the transmission a configurable number of times
- Repetition interval is configurable
- Receivers' duplicate detection ensures that application data is received at most one time

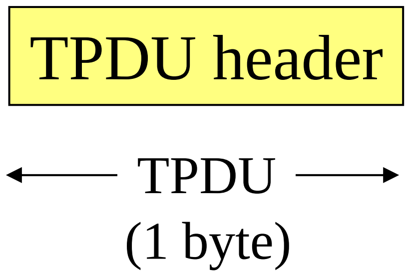
Acknowledged Message

- Transaction PDU type 0
- Enclosed PDU is Application Protocol Data Unit
- Delivered with unicast or multicast addressing



Acknowledgement Packet

- Transaction PDU Type 2
- Enclosed PDU is null
  - No data associated with ACK
- Delivered with unicast addressing
  - Address type 2a if acknowledging a unicast message
  - Address type 2b if acknowledging a multicast message
    - Includes group member number



Unicast Acknowledged Service

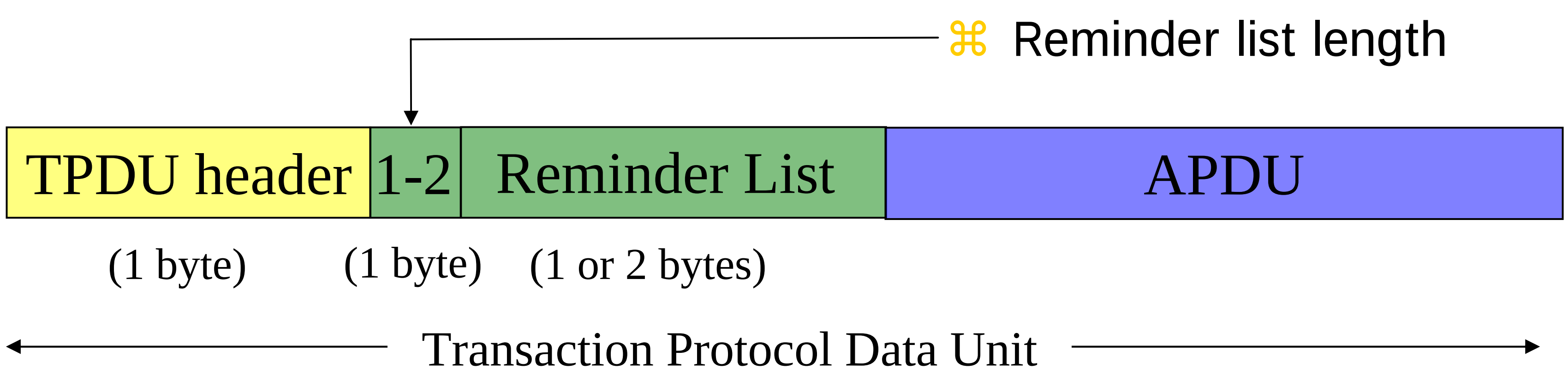
- Transmitter sends application data in an Acknowledged packet
- Receiver sends back an acknowledgement (ACK)
- If transmitter receives ACK, the transaction has succeeded
- If no ACK received within a configurable time, the transmitter retries the transaction
- If no ACK received after configurable number of retries, the transaction has failed

Multicast Acknowledged Service

- Transmitter sends application data addressed to the group in an Acknowledged packet
- Each receiver sends back an acknowledgement (ACK)
  - ACK also contains the group member number of the receiver (format 2b)
- If transmitter receives ACKs from all receivers, the transaction has succeeded
- If some ACKs are missing, the transmitter sends one or more Reminders

Reminder/Message Packet

- Transaction PDU type 5
  - Enclosed PDU is a bit map of group members who have already acknowledged, plus the application data
- Used for groups with 16 or fewer members



Reminder Packet

- Transaction PDU type 4
  - Enclosed PDU is a bit map of group members who have already acknowledged
  - For groups with more than 16 members

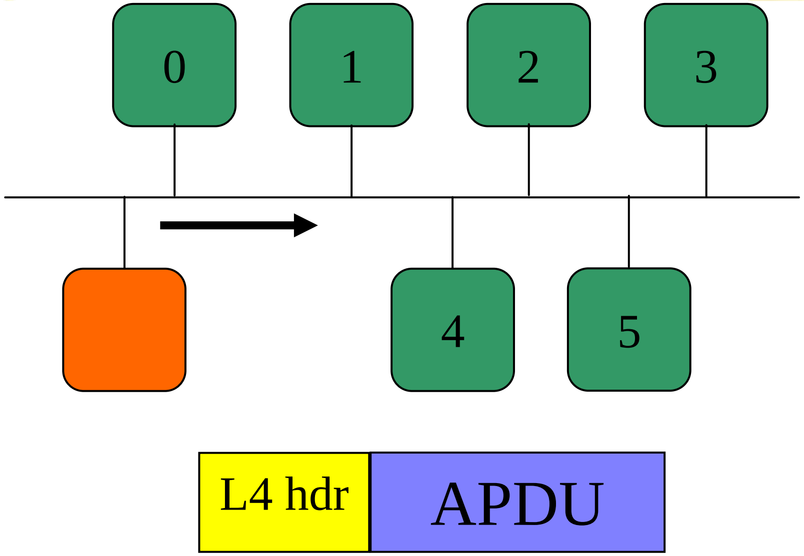
TPDU header	3-8	Reminder List
(1 byte)	(1 byte)	(3-8 bytes)

Transaction Protocol Data Unit

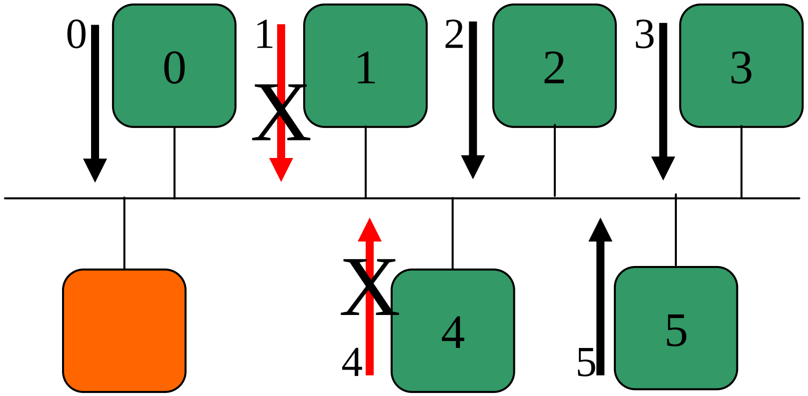
  - Followed by a separate Ack'd packet with the Application PDU

Example of Multicast Acknowledged Transaction

- Group size is 7
  - Including transmitter
- Transmitter sends Ack'd packet with App data

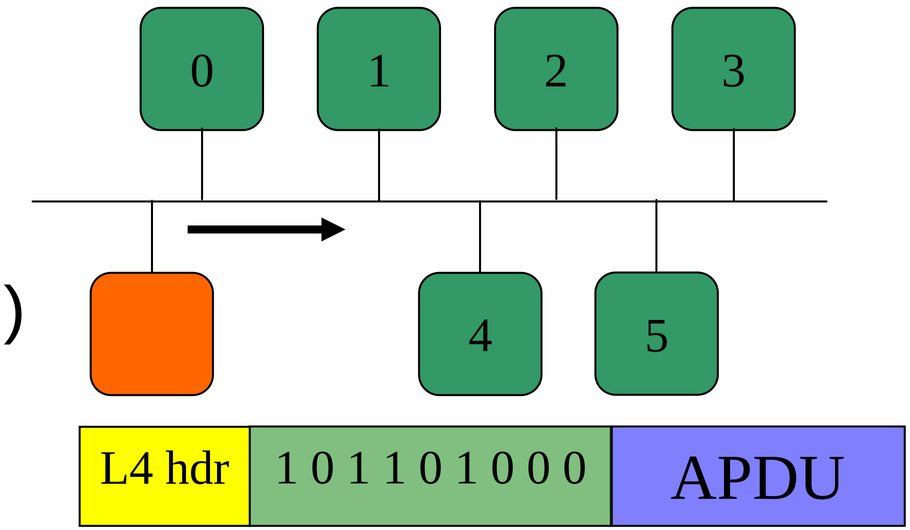


- ACKs from members 1 and 4 collide and are lost
- ACKs received OK from members 0, 2, 3, 5

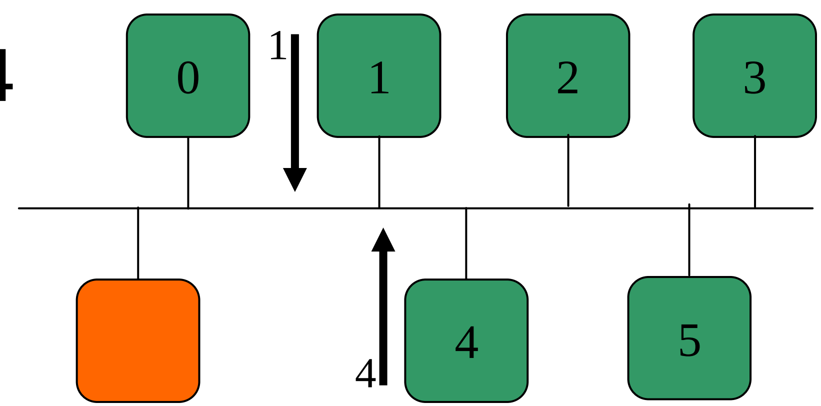


Multicast Reminder Message

- Transmitter sends Reminder/Msg packet
  - Reminder list (0, 2, 3, 5)



- Only members 1 and 4 send ACKs
- Transaction complete



Comparison of Protocol Services

- Unacknowledged (not repeated) service
- Fastest protocol service
  - No transaction processing involved
  - No Layer 4 header in packet
- Useful when the application can tolerate occasional loss of a packet
- Example: sampled analog data

Acknowledged Service

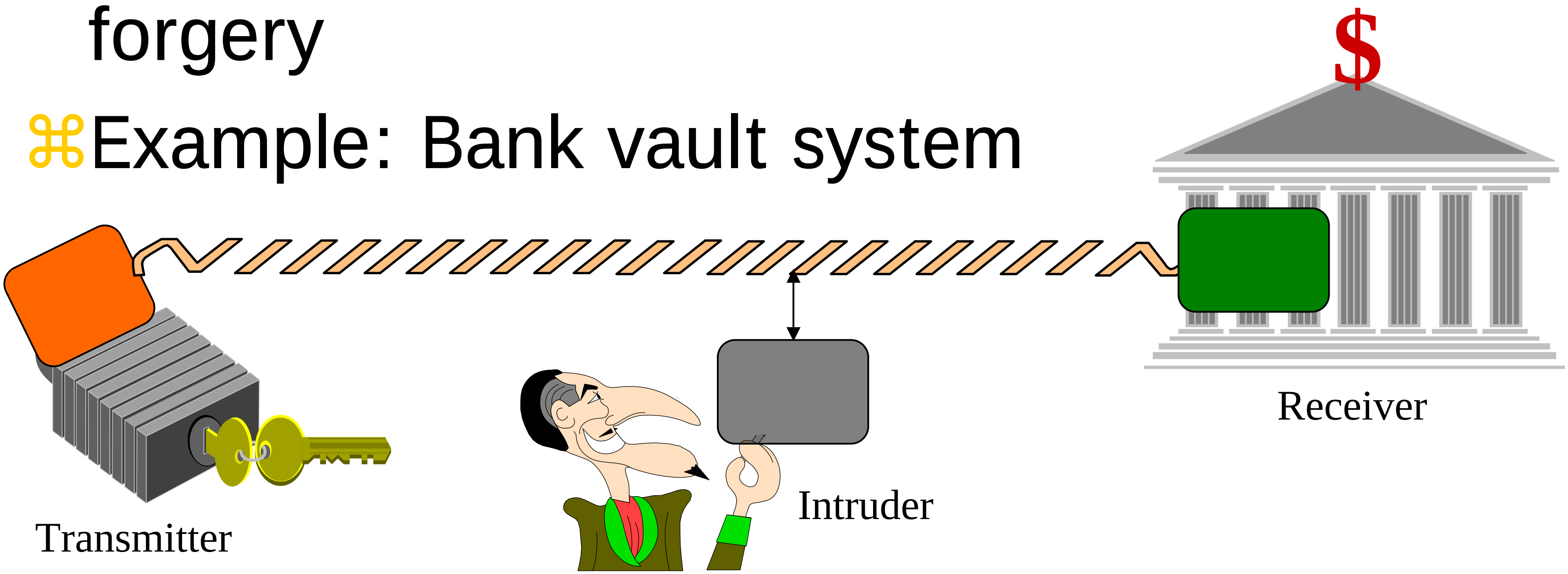
- Useful when transmitter must know if a message got through
  - Transmitter's application should handle transaction failure events that may occur
- Sending an acknowledged message to a group of N nodes causes N+1 packets on the network
  - Less efficient use of bandwidth for large groups

Unacknowledged/Repeated Service

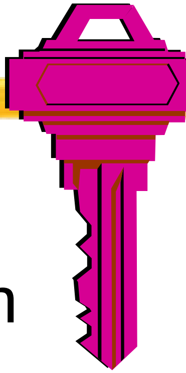
- More efficient for large groups
- No need to know the size of the group
- Failure probability example:
  - Collision rate is 4%
  - Repeated service with 4 retries
  - Probability that at least one packet will get through = $1-0.04^4 = 99.999744\%$
  - Only 4 packets instead of N+1

Authentication

- Allows the receiver of a message to know that the transmitter is genuine
- Impervious to record/replay attack & data forgery
- Example: Bank vault system



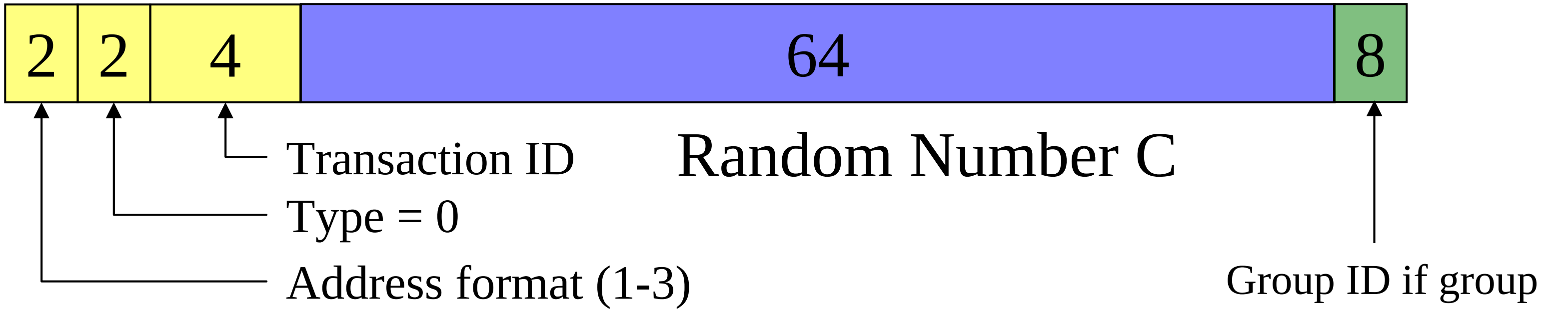
Authentication Algorithm



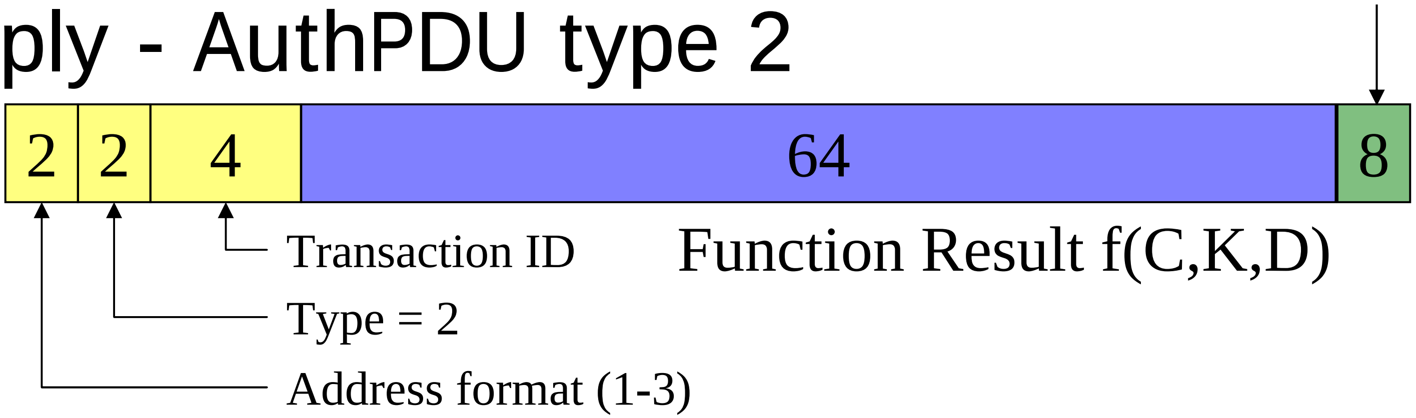
- Transmitter and receiver share a secret 48-bit Key K
- Transmitter sends a Transaction PDU with application data, and the *authenticate* bit set
- Receiver sends back a 64-bit random number Challenge C
- Transmitter computes a one-way function R=f(C,K,D) of the Challenge C, the Key K, and the Data D
- Transmitter returns result in the Reply R
- Receiver also computes f(C,K,D) and compares to R
- Receiver always ACKs even if authentication fails, to prevent brute force attacks
- Intruder only sees C, R & D; cannot deduce K, or forge D

Authentication PDUs
(PDU format 2)

- Challenge - AuthPDU type 0



- Reply - AuthPDU type 2

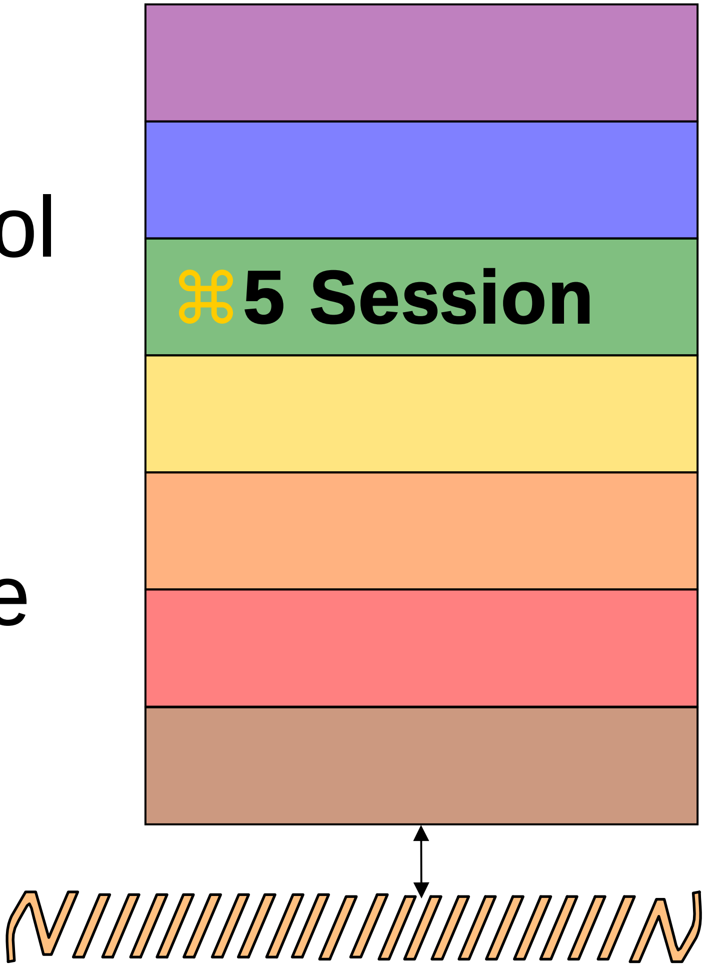


Transaction Layer Parameters

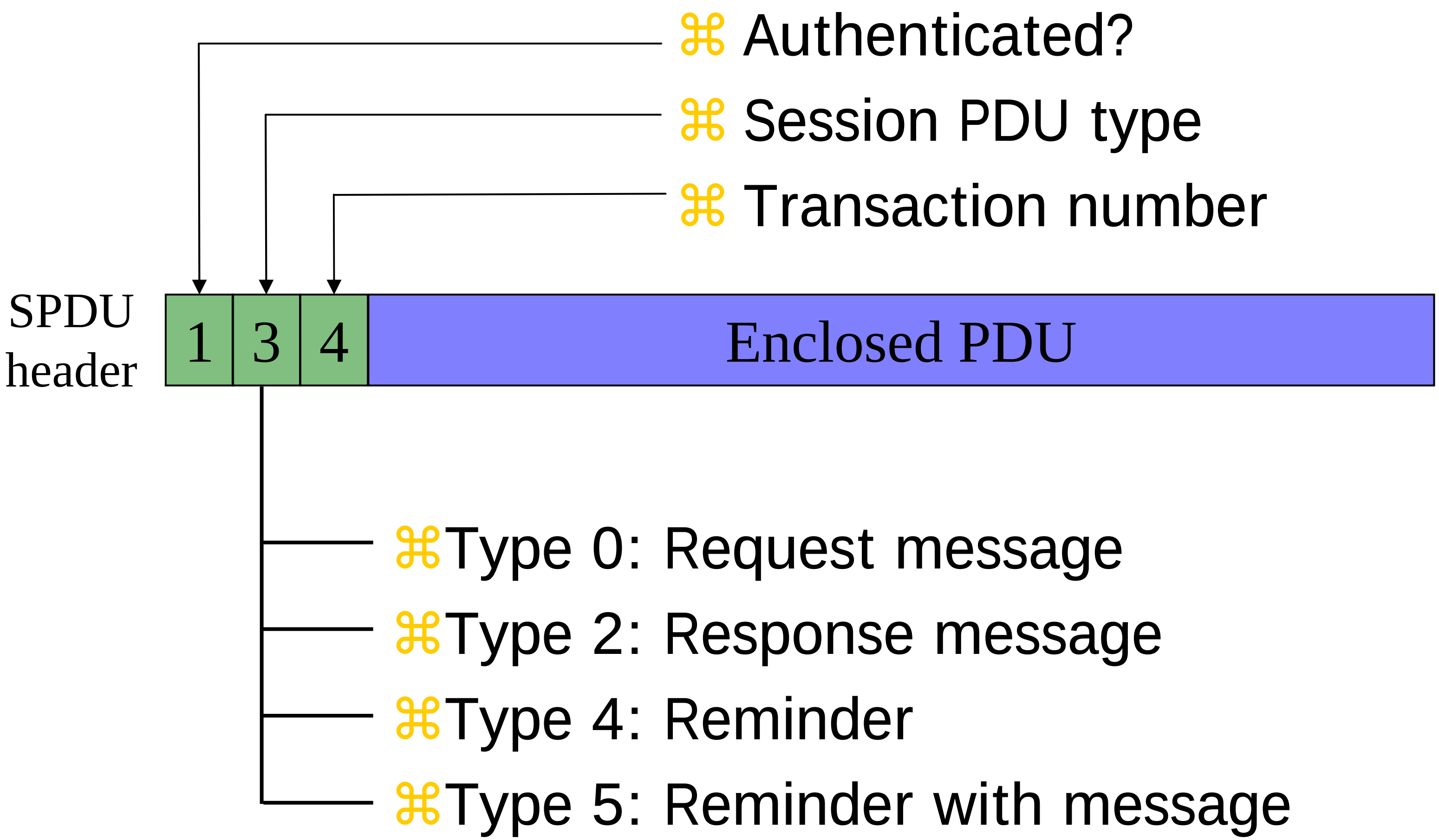
- Service type
  - Ack'd, Unack'd/Repeated, Unack'd, Auth'd
- Transmitter's transaction timer
  - Time to wait before retrying or repeating
- Transmitter's retry or repeat count
- Receiver's transaction timer
  - For duplicate detection
- Network management protocol to configure all of these

Protocol Layer 5

- Session layer
- Request/response protocol
- Remote procedure call
- Network variable polling
- Application-level response data
- May be authenticated

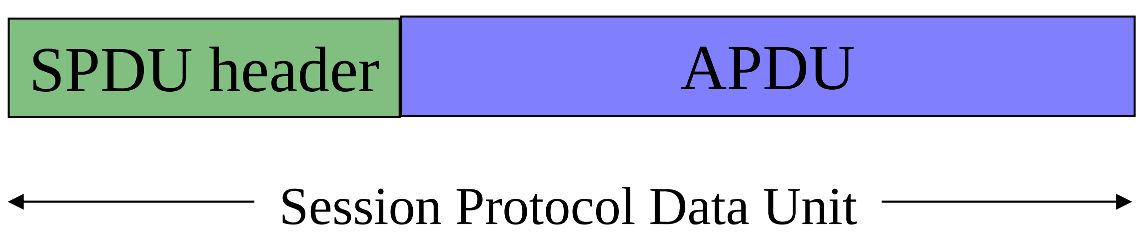


Session Protocol Data Unit
(PDU format 1)



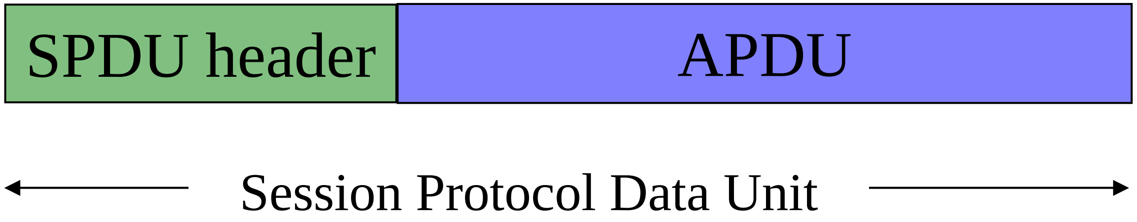
Request Message

- Session PDU type 0
- Enclosed PDU is Application Protocol Data Unit
- Delivered with unicast, multicast or broadcast addressing



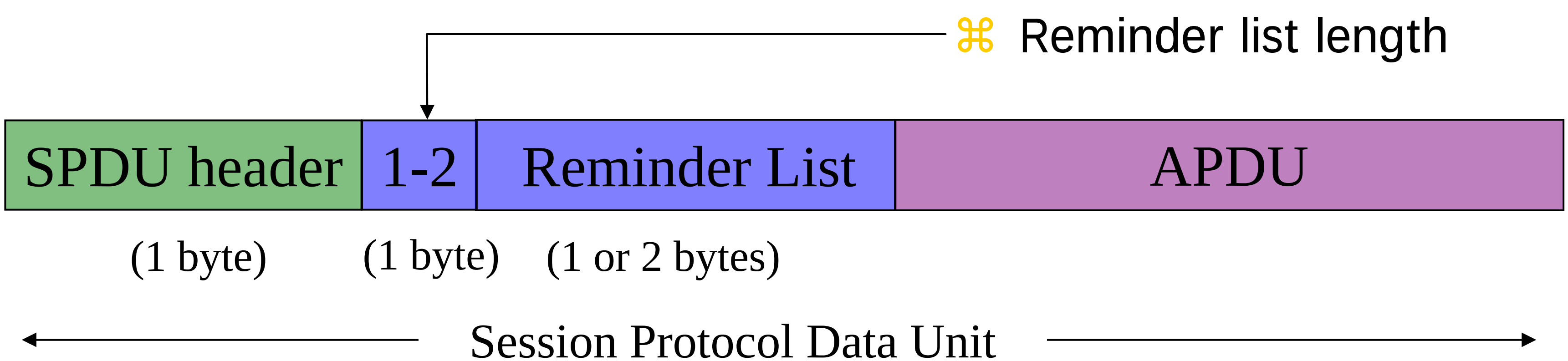
Response Message

- Session PDU type 2
- Enclosed PDU is Application Protocol Data Unit
- Delivered with unicast addressing to sender of request



Reminder/Message Packet

- Session PDU type 4
  - Enclosed PDU is a bit map of group members who have already responded, plus the application data
- Used for groups with 16 or fewer members



Reminder Packet

- Session PDU type 4
  - Enclosed PDU is a bit map of group members who have already responded
  - For groups with more than 16 members

SPDU header	3-8	Reminder List
(1 byte)	(1 byte)	(3-8 bytes)

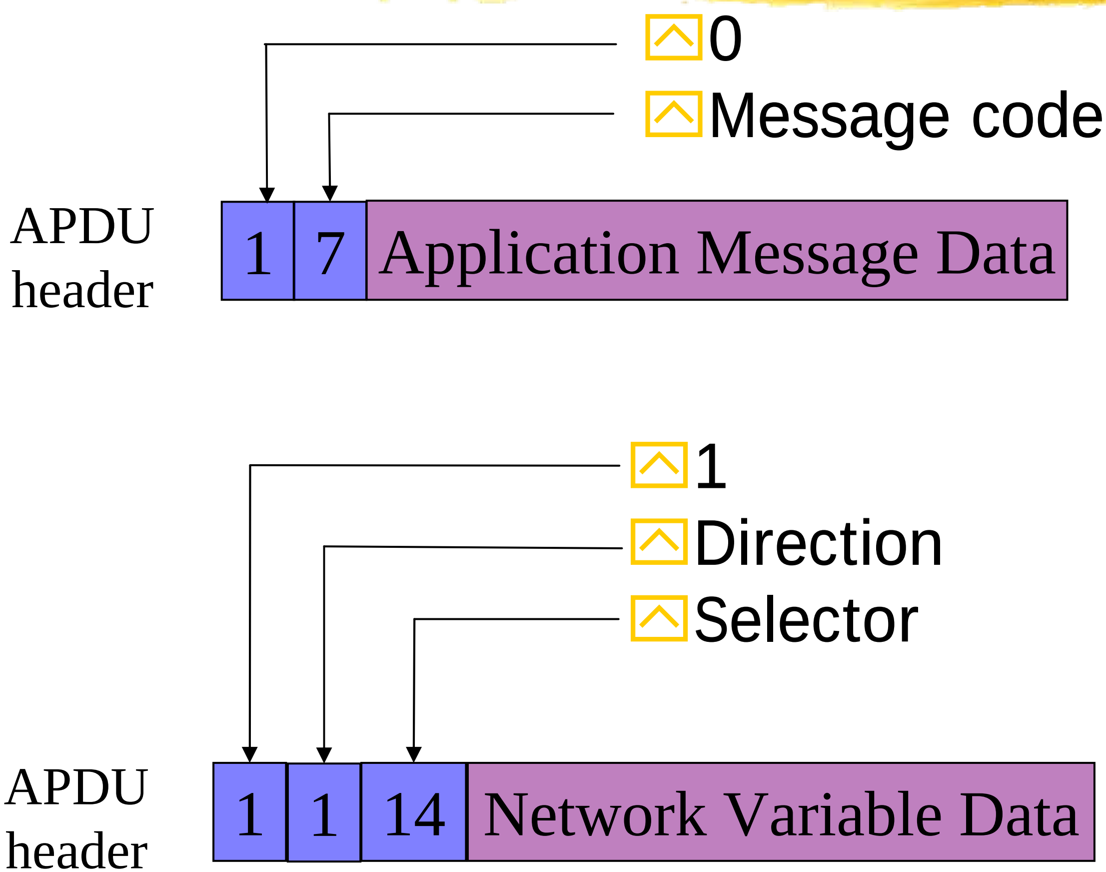
← Session Protocol Data Unit →

  - Followed by a separate Request packet with the Application PDU

Request/Response Service

- Unicast and multicast addressing
  - Same retry/reminder mechanism as acknowledged service
- Response generated by application layer
  - App layer *may* receive duplicate requests
- Broadcast addressing
  - First response from any addressed node successfully completes the transaction

The Application Protocol Data Unit



Application Message Codes*

- Hex 00 - 4F: application messages
  - Delivered to application layer
- Hex 50 - 5F: network diagnostic messages
  - Handled by network diagnostic layer
- Hex 60 - 7F: network management messages
  - Mostly handled by network management layer

* For other than response messages

Application Messages

- Interpretation of message code and data fields is up to the application
- Maximum data field length 227 bytes
  - With worst-case protocol overhead
  - For more data, use higher-level LonTalk file transfer protocol

Sending Application Messages

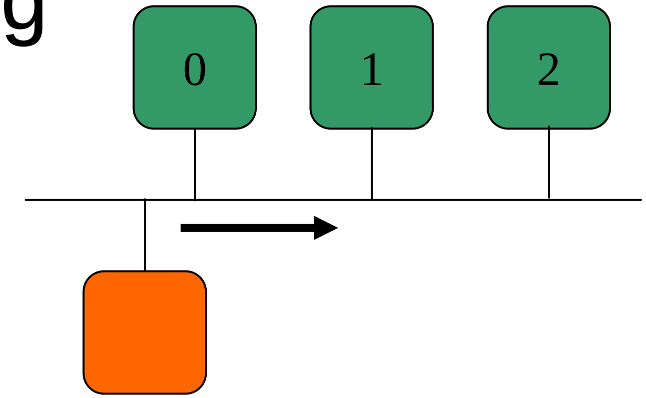
- Buffer Management API
  - Allocate / cancel output buffer
  - Priority / non-priority buffer pools
- Message parameters
  - Destination address
  - Service type
    - Unack'd, Ack'd, Unack'd/Repeated, Request, Auth'n
- Application receives completion event
  - Layer 4 success/failure indication

Receiving Application Messages

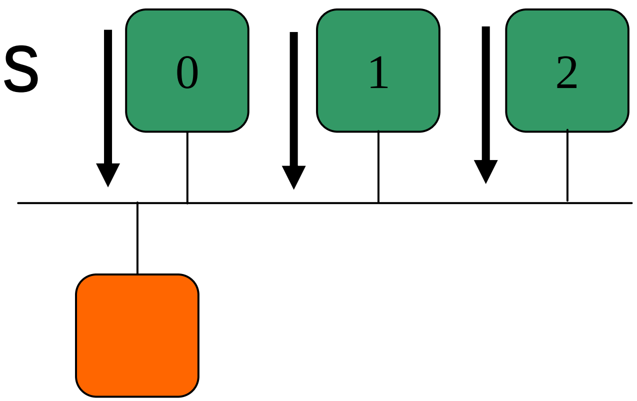
- Buffer Management API
  - Free input buffer
- Received message parameters
  - Source *and* destination address
  - Service type
    - Unack'd, Ack'd, Unack'd/Repeated, Request
  - Authentication requested but failed?
  - Priority

Request/Response Messaging

- Transmitter sends request msg
- Receivers receive request



- Receivers send response msgs
- Transmitter receives responses
  - Retries if necessary
- Transaction complete



Receiving a Request and Sending a Response

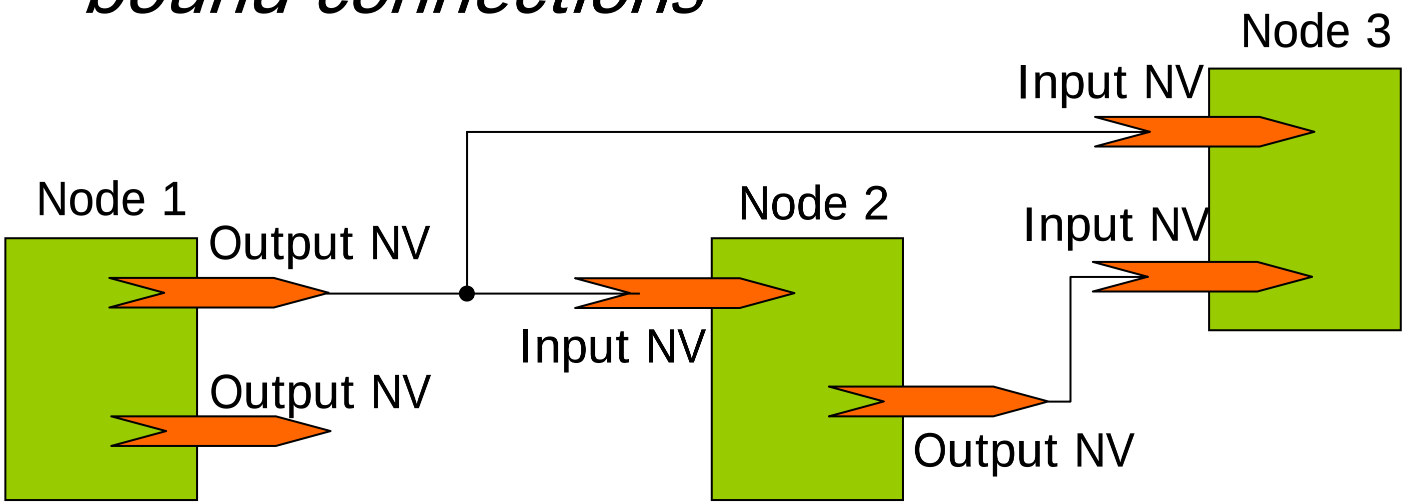
- Incoming message with Request service type
  - Retry/reminder bit set if a duplicate request
- Buffer management API
  - Allocate / cancel response buffer
  - Priority / non-priority buffer pools
- Addressing of response is always implicit
  - Response returned to sender of request

Application Messages *vs.* Network Variables

- Application messages are addressed to the node as a whole
  - Command-driven model
- Network variables provide multiple addressable entities per node
  - Shared data model
  - Higher-level semantics

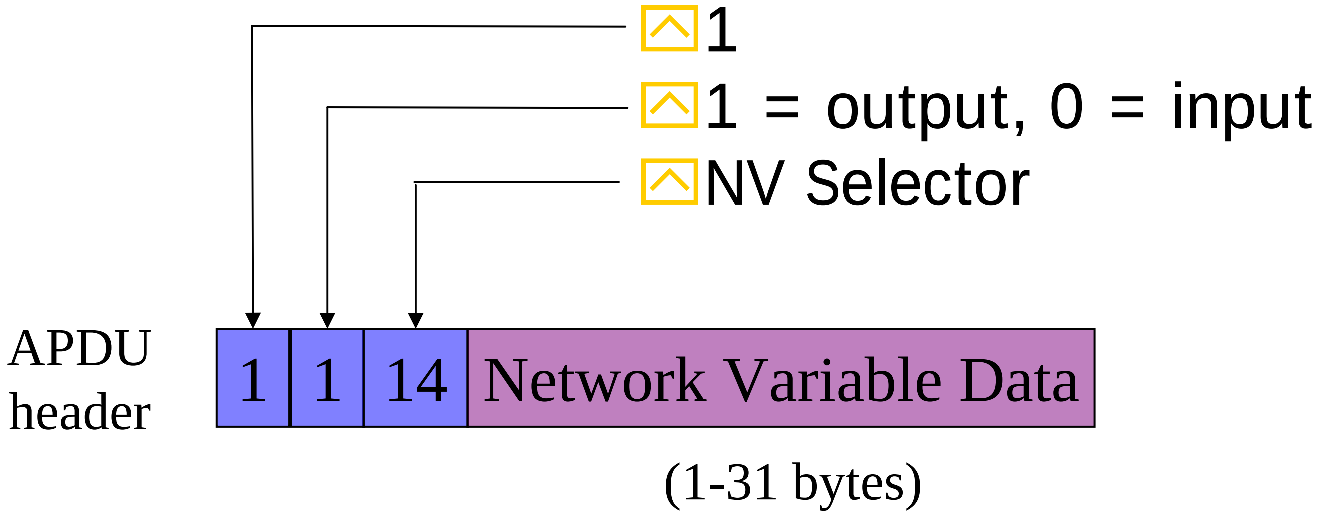
Network Variables - NVs

- Application layer abstraction for data sharing
  - Multiple addressable data entities per device
  - Implicitly addressed updates delivered via *bound connections*



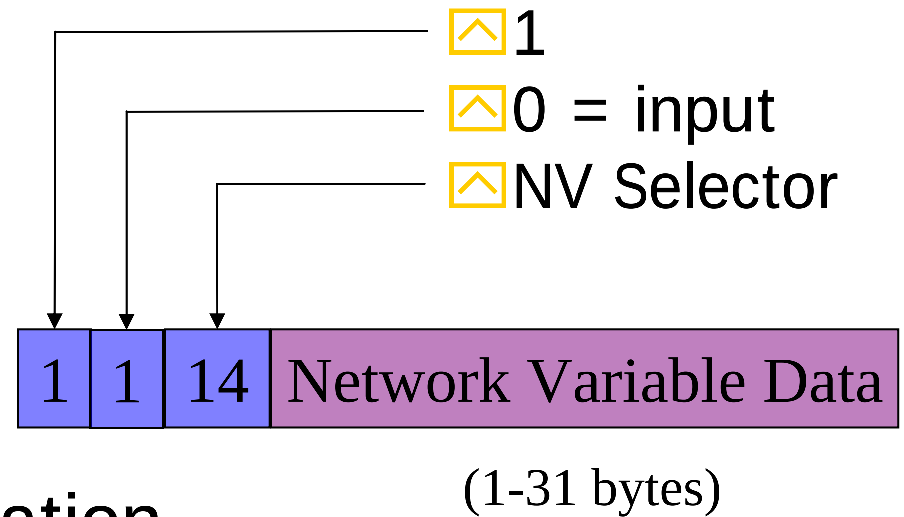
Network Variable APDU

- Selector mechanism provides associative addressing
- Bound connections may have multiple input and multiple output NVs



Network Variable Update Message

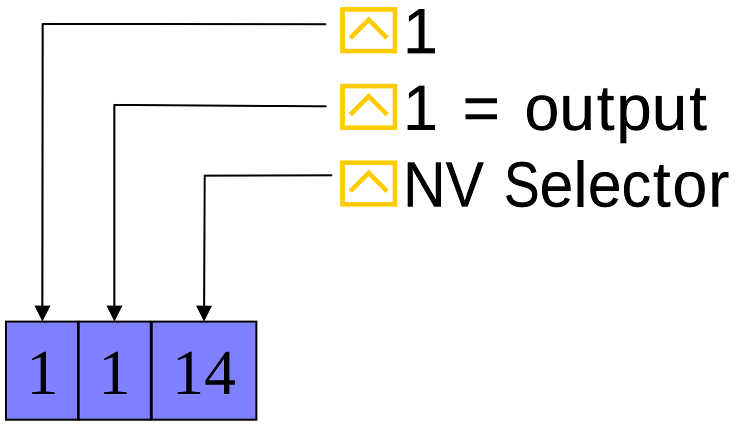
- Service type
  - Unacknowledged
  - Unack’d / Repeated
  - Acknowledged
  - Ack’d with authentication



- Receiving node(s) compare selector in message with selectors of their input NVs
- If there’s a match, NV is updated with value from message

Network Variable Poll Request Message

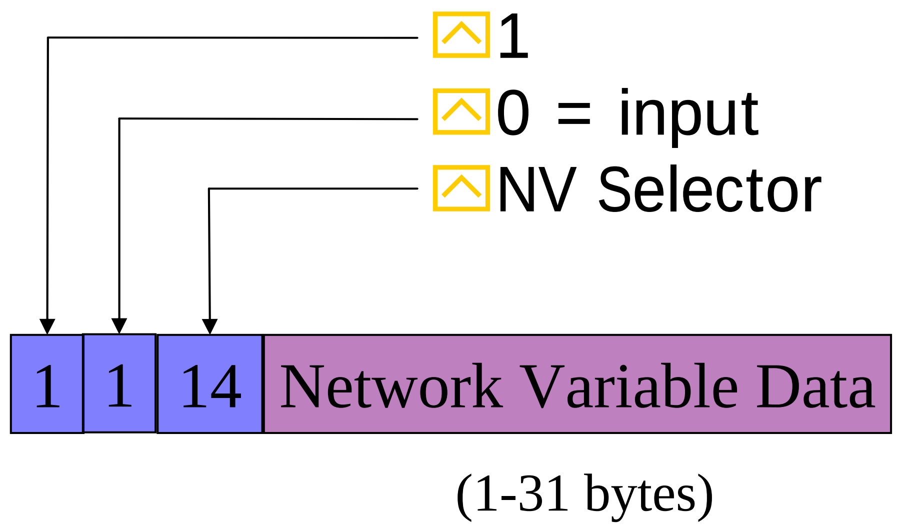
- Service type
  - Request
  - Request with authentication



- Receiving node(s) compare selector in message with selectors of their output NVs

Network Variable Poll Response Message

- If the selectors match, the value of the network variable is returned in a response message
- Otherwise a null response is returned



Application Layer API for NVs

- Output network variables
  - Function
    - `_nv_update(int index, void *pValue, int len);`
  - Event handler
    - `_nv_completes(int index, boolean status);`
- Input network variables
  - Event handler
    - `_nv_update_occurs(int index, void *pValue, int len);`
  - Function
    - `_nv_poll(int index);`

Standard Network Variable Types (SNVTs)

- Network Variables provide a convenient way to share data
- Standard Network Variable Types provide a consistent meaning for shared data
- Physical quantities
  - Mass, length, time, temperature, voltage etc.
- Fixed and floating point representations
- Enumeration states or modes
- Structured types

Examples of Standard Network Variable Types

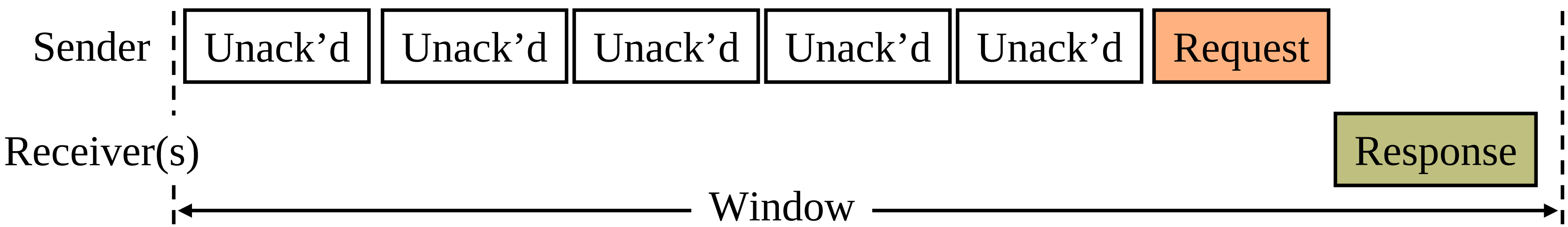
- Temperature: -273.17 .. +327.66 degrees C (0.01 deg C)
- Time Stamp:

```
typedef struct {
   uint16   year;      // 0..3000
   uint8    month;     // 1..12
   uint8    day;       // 1..31
   uint8    hour;      // 0..23
   uint8    minute;    // 0..59
   uint8    second;    // 0..59
} SNVT_time_stamp;
```

- For latest SNVT list, see **http://www.lonmark.org/PRESS/Snvt853.zip**

LonTalk File Transfer Protocol

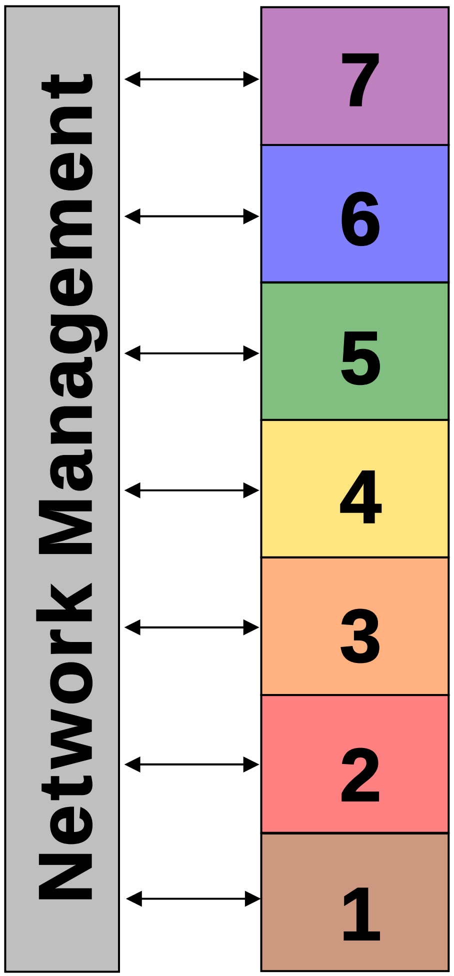
- For transferring more than a single packet
- Windowed application message protocol



- Supports single transmitter/multiple receivers
- Sequential or random access files

Network Management Layer

- Implicit addressing mechanisms
- Node address assignment
- Configuration of protocol parameters
- Application downloading
- Configuration of routers
- Network variable binding

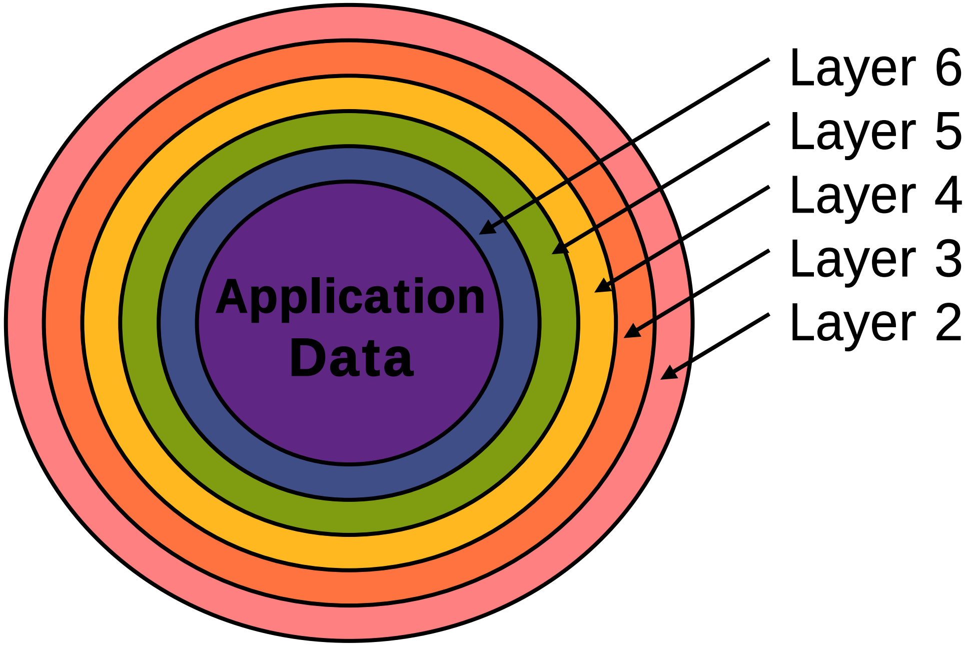


Implicit Addressing

- How to make a complete temperature sensor in 91 bytes of application code?
- Application code doesn't need to concern itself with configuring protocol parameters and device addressing
- Network configuration data structures, and mechanisms to access them, are defined as part of the standard

Packet Structure

- Each protocol layer adds its own header to the information in the packet
- Device's network image contains all the information necessary to send and receive packets



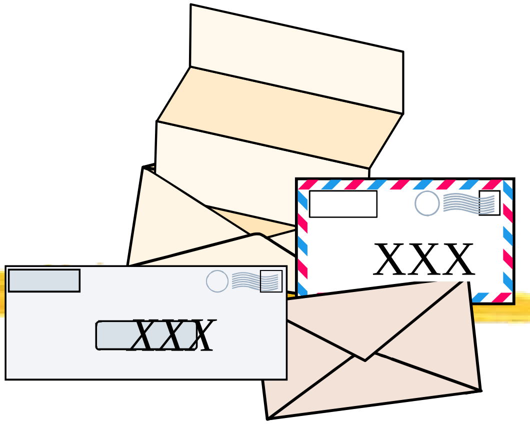
Managing the Device's Network Image

- *Network Management Protocol* is defined to Query and Update any portion of the Network Image
- For maximum flexibility, use a Network Management Tool with rich GUI
- Self-installing device may modify its own network image
  - At the cost of code size and complexity

Data Structures For Node Addressing

- Every LonTalk device contains a network configuration image
  - Domain Table
  - Address Table
    - Destination addresses
    - Group membership
  - Network Variable Configuration Table
  - Configuration Structure
- Tables in writeable non-volatile memory
  - May be updated over the network by network management messages

Explicit Addressing



- Application may specify destination address and layer 4 parameters
  - At the cost of code size and complexity
- Usually used in complex devices
  - e.g. graphical PC-based devices used for network management and user interfacing
- For low-cost sensor/actuator nodes, implicit addressing is easiest and cheapest

Example: Network Variable Update Message

LonTalk Packet

1
2
3
4
5
6
7

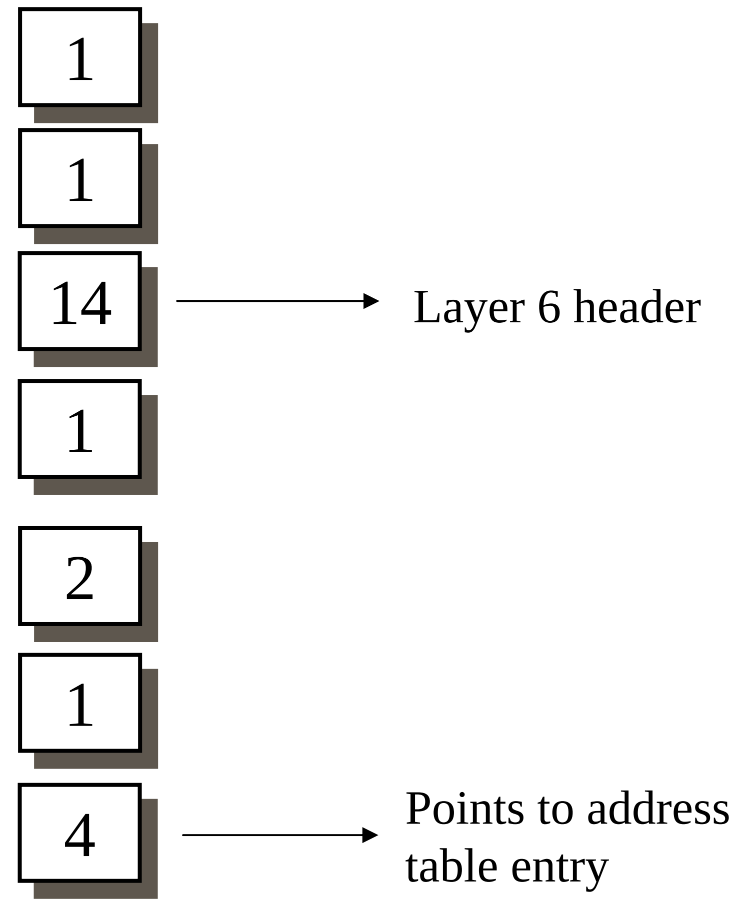
- Transmitter needs to specify:
  - Preamble, beta1, beta2
  - Priority, delta backlog
  - Source, dest address
  - Transaction type
  - NV selector
  - Application data

The Network Variable Configuration Table

- One 3-byte entry for each network variable
- Specifies network variable selector
  - Layer 6 header for outgoing messages
  - NV recognition for incoming messages
- Specifies implicit address for outgoing NV messages
  - Output NV updates
  - Input NV polls

Network Variable Configuration Table Entry

- Priority
- Direction
  - 0=in, 1=out
- NV Selector
- Turnaround
  - 1 if bound to NV on same node
- Service Type
  - 0 = Ack’d, 1=Rept’d, 2=Unack’d
- Authenticated
- Address Table Index
  - For output NVs and polled input NVs



The Address Table

- Up to 15 entries, each 5 bytes long
- May specify group membership for incoming address recognition
- May specify destination address and layer 4 parameters for outgoing messages
  - Pointed to by output or polled NV table entry
  - When application sends a message, it may specify which address table entry to use

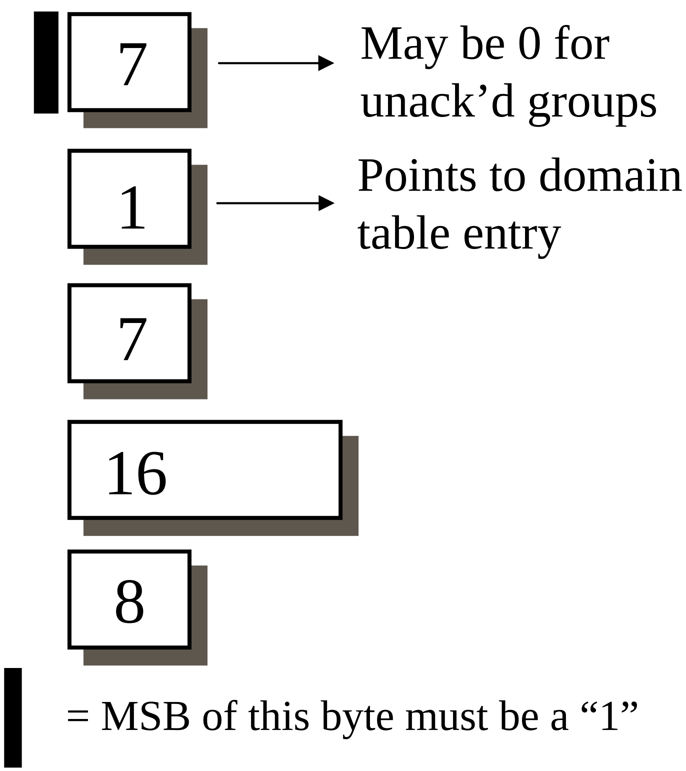
Transaction Layer Timing Parameters

- All address table entries contain two bytes to specify layer 4 timing parameters
- Repeat timer* — 4
  - For unack'd/repeated service
- Retry/repeat count — 4
- Receive timer* — 4
  - For incoming duplicate detection
- Transmit transaction timer* — 4
  - For ack'd & request svc

* For encoding, see spec.

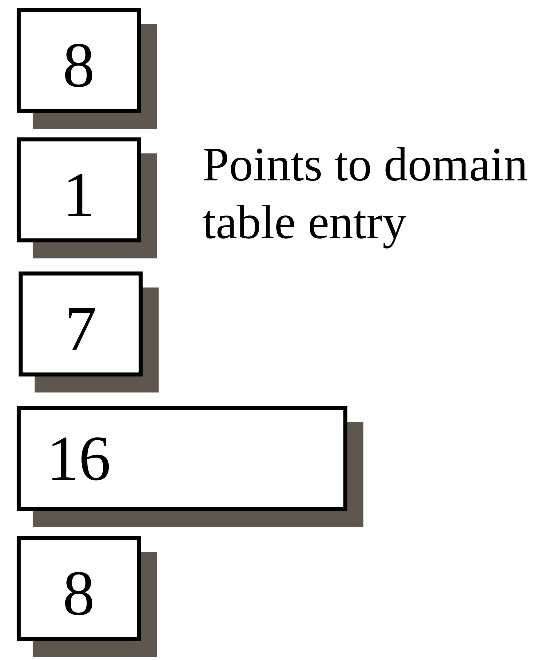
Address Table Entry: Group Membership

- Used to send to *or* receive messages from a group
- Group size (2-64)
- Domain reference (0-1)
- My member ID (0-63)
- Transaction layer params
- Group ID (0-255)



Address Table Entry: Unicast Addressing

- Used to send messages to a single node
- Type = 1
- Domain reference (0-1)
- Destination node ID (1-127)
- Transaction layer parameters
- Destination subnet ID (1-255)



Address Table Entry: Broadcast Addressing

- Used to send messages to a whole subnet or domain

⌘ Type = 3	8	
⌘ Domain reference (0-1)	1	Points to domain table entry
⌘ Delta backlog (0-63)	7	
⌘ Transaction layer parameters	16	
⌘ Destination subnet ID (1-255)	8	

  - Subnet 0 means domain-wide

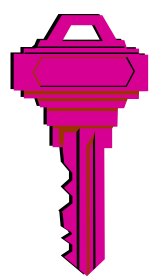
The Domain Table

Masters of their Domain

- Up to two 15-byte entries, one for each domain to which the node belongs
- Specifies this node's subnet and node IDs within the domain
- Specifies the domain ID itself
- Specifies the authentication key to be used in this domain

Domain Table Entry

- Domain ID — 0, 8, 24 or 48 bits
- My subnet ID (1-255) — 8
- My node ID (1-127) — 7
- Domain ID length — 8
  - 0, 1, 3, or 6
- Authentication key — 48 bits



= MSB of this byte must be a “1”

Configuration Structure

- Physical layer parameters
  - Transceiver type
  - Communications bit rate
  - Special-purpose transceiver parameters
- Media access layer parameters
  - Preamble length, beta1, beta2 etc.
  - Number of channel priorities
  - Priority assignment of this node on its channel

LonTalk Network Management Messages

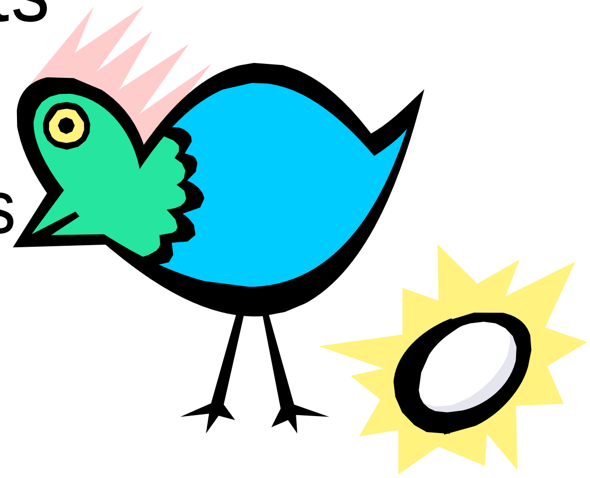
- Complete set of request/response messages defined for managing the network configuration of the device
- Query/Update Network Variable Configuration table
- Query/Update Address table
- Query/Update Domain table
- Query/Update Configuration Structure

More Network Management Messages

- Query node’s self-documentation data
  - Allows network management tool to read node’s external interface
- Download application program
  - Protocol defined for Neuron Chip implementations
- Router configuration messages
  - Query/Update routing tables, change routing algorithm

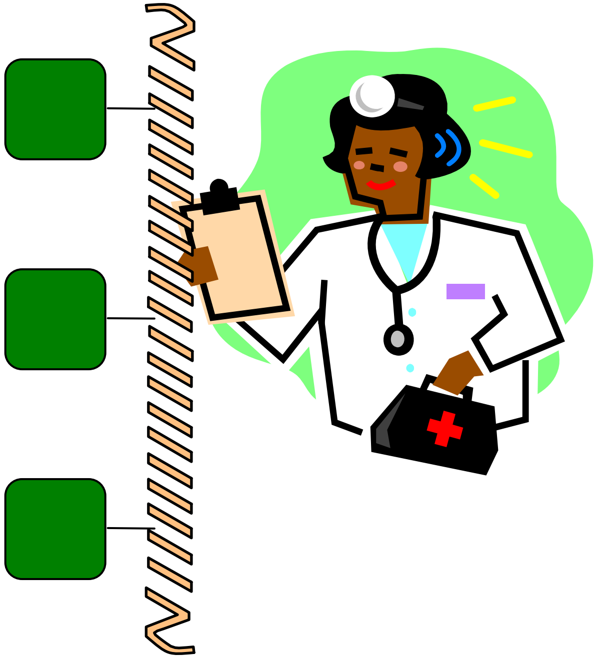
Bootstrapping the Network Configuration

- How do you tell a node what its address is?
  - If it doesn't yet have an address
  - Unique ID addressing mode



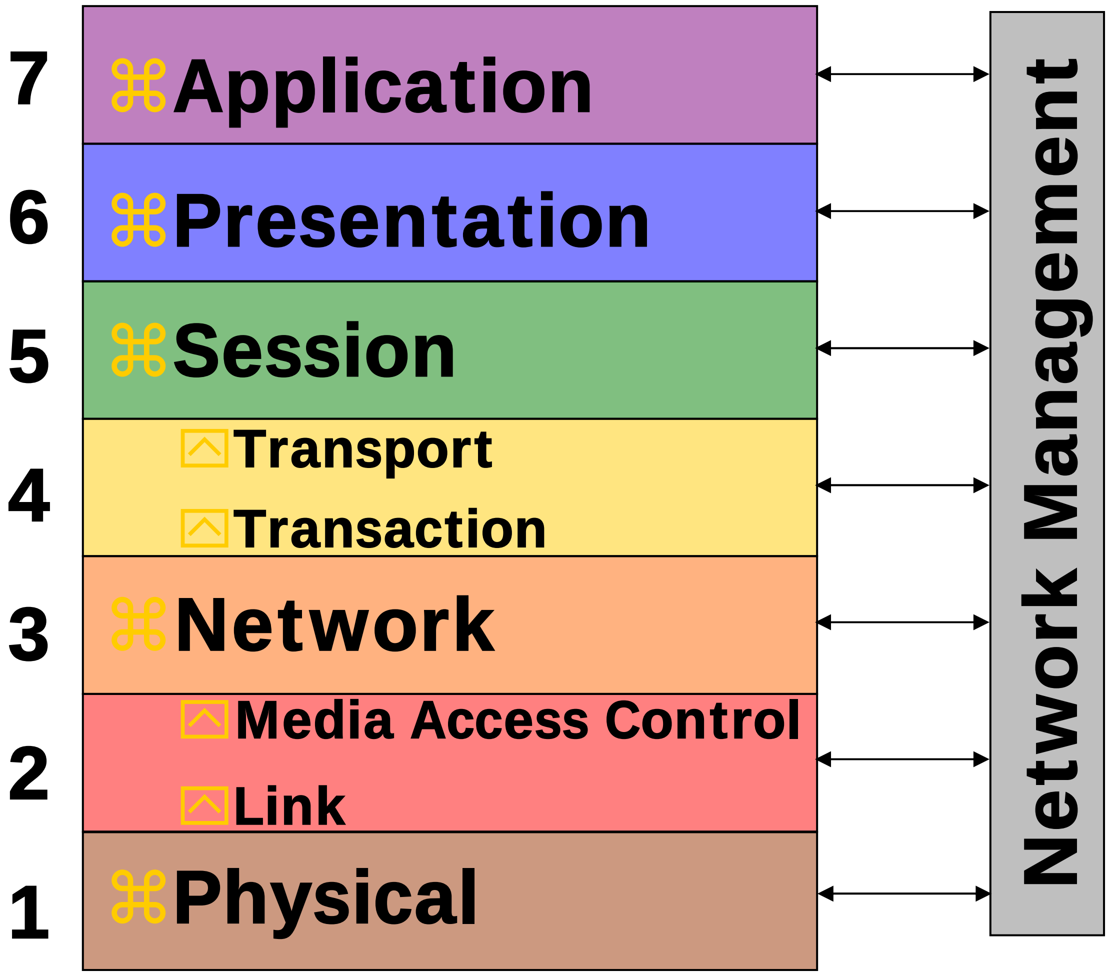
- Service Pin
  - Hardware mechanism to cause the node to broadcast a message containing its unique ID
- Broadcast a query request to nodes
  - Response message contains the unique ID
  - Install message to cause node to "wink"

Network Diagnostic Messages



- Query node's status
  - CRC errors detected
  - Buffer overruns
  - Transaction layer errors
- Query node's traffic statistics
  - Packets received with valid CRC
  - Packets received addressed to this node
  - Packets transmitted
- Proxy addressing

LonTalk Protocol Summary



Services Defined at Each Layer

- Designed for cost-sensitive implementation
- Physical layer
  - Media independent
- Media access control sub-layer
  - Modified CSMA with collision avoidance and priority access
- Addressing layer
  - Supports very large networks, multiple channels
  - Low-cost routers for multiple subnets

Transport and Session Layer Services

- Reliable transport services
  - Duplicate detection and rejection
  - Unicast and multicast acknowledged
  - Unacknowledged/repeated service
- Authentication
  - Security applications
- Session layer request/response protocol
  - Unicast and multicast
  - Authentication option

Presentation Layer Services

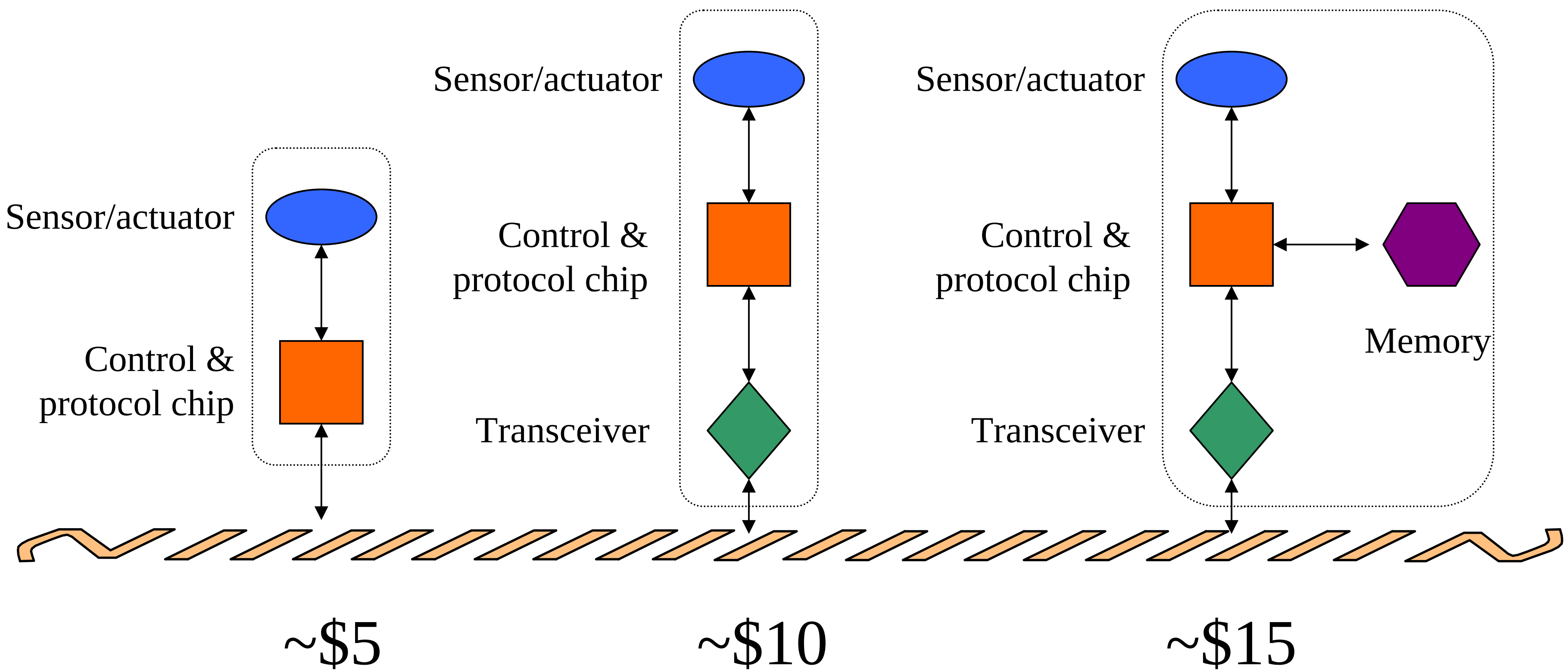
- Application Messages
  - Datagrams addressed to node
  - File transfer protocol for large data objects
- Network Variables
  - Multiple addressable entities per device
  - Shared data-driven model
  - Flexible variable binding semantics
  - Standard Network Variable Types for interoperability

Network Management and Diagnostic Protocols

- Complete access to all protocol parameters via defined management services
- Wide range of installation and maintenance scenarios
  - From self-installed to PC-based tools
- Diagnostic protocol for network troubleshooting

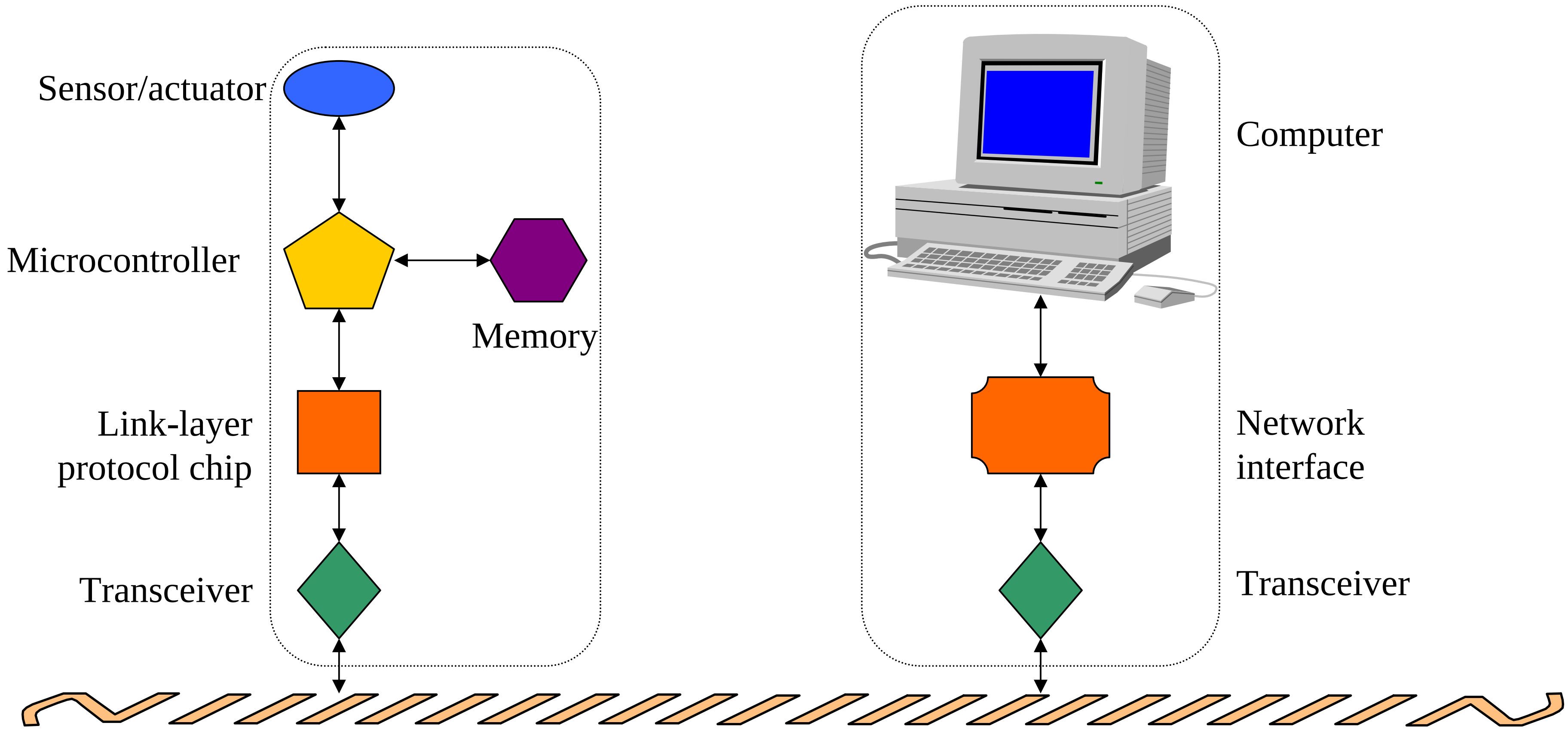
Cost Effectiveness of LonTalk Implementations

- High volume low cost nodes



Functionality of LonTalk Implementations

- Higher-capability nodes



The LonTalk Protocol is a Freely Available Open Standard

EIA STANDARD

Control Network Protocol Specification

EIA-709.1

MARCH 1998

ELECTRONIC INDUSTRIES ALLIANCE
ENGINEERING DEPARTMENT

EIA
Electronic Industries Alliance