Information Theory and Networks

Lecture 25: Coding with Noise

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/Lecture_notes/InformationTheory/

School of Mathematical Sciences,
University of Adelaide

October 10, 2013

Part I

Coding with Noise

> 6. ERROR: A really big FUCK UP has been detected !!
>
> *189 Funny UNIX Error Messages*

http://www.fsckin.com/2007/09/24/189-humorous-unix-errors/

Section 1

Error Correcting Codes (ECC)

Redundancy is your friend

- We've talked about redundancy as a method to help avoid errors
  - English does this as a matter of course
  - Not well enough for really noisy mediums (e.g., some radio)
    - e.g. its hard to tell 'N' from 'M'
  - Nato Phonetic Alphabet

letter	code
A	alpha
B	bravo
C	charlie
D	delta
F	foxtrot
⋮	⋮

  - Or we could use repetition.
- What we need is a way to do this efficiently.

Complete alphabet (also know as International Radiotelephony Spelling Alphabet or the ICAO phonetic) is at `http://en.wikipedia.org/wiki/NATO_phonetic_alphabet`.

Note that words come with expected pronunciation, e.g. "nine" is said "NIN-ER".

It has problems

Bob:	What's that again?
Alice:	I said, I have to tell you about Mike.
Bob:	Didn't get the last word Alice, can you spell it out?
Alice:	Mike India Kilo Echo.
Bob:	Got India Kilo Echo, what was the first word?
Alice:	Mike
Bob:	Can you spell that?
Alice:	Mike India Kilo Echo
	⋮

The Alice and Bob After Dinner Speech,
John Gordon, 1984

`http://downlode.org/Etext/alicebob.html`

Error Detection and Correction

Two main approaches:

- Error Detection
  - use a few extra bits to detect when there is an error
  - retransmit errored data
- FEC - Forward Error Correction (sometimes called channel coding)
  - use coding to create unambiguous codewords that can be transmitted with arbitrarily small errors

We'll see that they are pretty closely related.

Error Detection and Correction
Two main approaches:
Error Detection
use a few extra bits to detect when there is an error
retransmit errored data
FEC - Forward Error Correction (sometimes called channel coding)
use coding to create unambiguous codewords that can be transmitted with arbitrarily small errors
We'll see that they are pretty closely related.

Section 2

Error Detection

Section 2
Error Detection

Error Detection

Include some extra bits to check for errors

Various approaches.

- Parity bits
- Checksums
- Hash functions

Error Detection
Include some extra bits to check for errors
Various approaches.
- Parity bits
- Checksums
- Hash functions

Earliest example: the Torah. Jewish scribes copying the Torah tried to make copies perfect. Used techniques such as summing numbers of words per line, and per page, and checking certain words. A page was thrown out if a single mistake was found, and three mistakes on a single page invalidated the entire text.

Parity Bits

- write data in binary: 10101011
- calculate the parity, i.e., is the sum even or odd, using addition modulo 2

  1 + 0 + 1 + 0 + 1 + 0 + 1 + 1 = 1 (mod 2)
- add the extra parity bit on the end, e.g.

  101010111
- if parity is wrong after trans., we know there was at least one error
  - but two errors may cancel each other out
- we could add extra parity bits to detect other errors
  - e.g., take a parity bit for whole sequence, and also for blocks length 4

    1 + 0 + 1 + 0 + 1 + 0 + 1 + 1 = 1 (mod 2)

    1 + 0 + 1 + 0 = 0 (mod 2)

    1 + 0 + 1 + 1 = 1 (mod 2)

    and the new sequence is

    10101011101
  - Now we can detect 2 bit errors, as long as they don't happen in the same sub-block

Parity Bits
- write data in binary: 10101011
- calculate the parity, i.e., is the sum even or odd, using addition modulo 2
  1 + 0 + 1 + 0 + 1 + 0 + 1 + 1 = 1 (mod 2)
- add the extra parity bit on the end, e.g.
  101010111
- if parity is wrong after trans., we know there was at least one error
  - but two errors may cancel each other out
- we could add extra parity bits to detect other errors
  - e.g., take a parity bit for whole sequence, and also for blocks length 4
    1 + 0 + 1 + 0 + 1 + 0 + 1 + 1 = 1 (mod 2)
    1 + 0 + 1 + 0 = 0 (mod 2)
    1 + 0 + 1 + 1 = 1 (mod 2)
    and the new sequence is
    10101011101
  - Now we can detect 2 bit errors, as long as they don't happen in the same sub-block

General form of parity bits from [Gal68, p.197] is that we take a binary signal $\mathbf{u}$ of length K and create a new signal $\mathbf{x}$ of length $N > K$ by taking

$$x_n = \begin{cases} u_n, & \text{for } 1 \leq n \leq K, \\ \sum_{i=1}^{L} u_i g_i^{(n)}, & \text{for } K+1 \leq n \leq N. \end{cases}$$

where summation is modulo 2.

- the 1st K bits of $\mathbf{x}$ are called information bits, or the message
- the $N-K$ bits at the end are called parity bits or check digits

We will consider parity checks a little more further along. For the moment, remember that there can be errors in the parity bits as well as the message. So increasing the number of parity bits carelessly might be counter-productive, as well as inefficient. We need to be systematic about it.

Parity Bits and Errors

- Given n bits in a message block + parity bits, and a BSC (Binary Symmetric Channel) with error probability p
- Probability of m errors is binomial

$$P(\#errors = m) = \binom{n}{m} p^m (1-p)^{n-m}$$

- For small p (and reasonably large n) we can approximate this by Poisson distribution

$$P(\#errors = m) \simeq \frac{e^{-np}(np)^m}{m!}$$

  - $P(0) \simeq e^{-np} \simeq 1 - np + (np)^2/2$
  - $P(1) \simeq kpe^{-np} \simeq np(1-np)$
  - $P(>1) = 1 - P(0) - P(1) \simeq (np)^2/2$
  - $P(>2) \simeq (np)^3/6$

Imagine a message size of 4, and we will compare adding 1 parity bit (which can detect 1 errored bit) with 3 parity bits (which we will show allows us to detect when there are up to 2 errors in any of the bits). So the probability of an undetectable error for 1 and 3 extra bits is

$$\begin{aligned} P(>1|n=5) &\simeq \frac{5^2}{2}p^2 \simeq 12.5p^2 \\ P(>2|n=7) &\simeq \frac{7^3}{6}p^3 \simeq 57.16p^3 \end{aligned}$$

so clearly the decision about whether to use a single parity bit, or the three bits, depends on p. For small p, obviously the second approach wins, but for $p > 12.5/57.16 \simeq 0.21$ we might be better with just a single parity bit, because the potential for errors in the extra two bits overcomes their error checking capacity.

Parity Bits Example

- assume a typical optical fibre as a BSC with BER $\alpha = 10^{-12}$
- Max standard TCP/IP packet is 1500 bytes = 12,000 bits
- $np = 1.2 \times 10^{-8}$
- Certainly satisfies requirements for Poisson approximation

$$\begin{aligned} P(1) &\simeq np \sim & 1.2 \times 10^{-8} \\ P(>1) &\simeq (np)^2/2 \sim & 0.7 \times 10^{-16} \\ P(>2) &\simeq (np)^3/6 \sim & 0.3 \times 10^{-24} \end{aligned}$$

- So we might feel safe checking for single errors
  - but not all internet wires are that good
  - wireless certainly isn't
  - some packets are bigger
  - the Australian Internet will carry about 1 exabyte per month by 2016 – we are starting to get to the point where 2 errors could just happen

Checksums

- Checksums generalise the idea of parity bits
  - create a "sum" of message codewords
    - fixed length blocks of binary bits
  - include the checksum in the data
    - by appending, or including in a header
- Modular sum
  - break into blocks length n
  - perform sum modulo 2^n
  - detects single bit errors
  - 2 bit errors go undetected with probability $< 1/n$
  - misses correlated errors, e.g., block transposition
- Cyclic Redundancy Checksum (CRC)
  - include position as well as value
  - based on generator polynomial
  - perform arithmetic on finite field $GF(2)$
  - n-bit CRC when its check value is n bits
  - deal well with burst errors (which are common)

Galois Field $GF(2)$ is the just the finite field of single binary digits:

- addition is just the logical XOR (or addition mod 2)
- multiplication is just the logical AND (or binary multiplication)

In general we could have $GF(q)$, for an alphabet q, where the operations are addition and multiplication mod q, as long as q is prime or a prime power.

Examples

UPC barcodes: generalise parity to 10 digits

http://en.wikipedia.org/wiki/Universal_Product_Code

ISBN 978-0-521-87310-9

- UPC-A has 12 (decimal) digits
- digits represented by bar widths/patterns
- parity of patterns used to indicate left/right so can scan from either direction
- check digit UPC-A barcode "78052187310**9**"
  - add odd-numbered digits times 3, and add event digits
  - modulo 10, then complement

$$9 = 10 - \left[(7+0+2+8+3+0) \times 3 + (8+5+1+7+1+9) \mod 10 \right]$$

- there are other bar code standards, but this is a common one

- it should detect any single error because an error multiplied by three mod 10 should cause a change in the check digit
  - think of the $\times 3$ as weights on odd positioned numbers
  - we need to use weights $w = 1, 3, 7, 9$ so that $ew \neq 0 \mod 10$ for any error e
  - if we used mod 11 arithmetic (11 is prime) we could use all the weights (use X to represent 10)
- won't necessarily detect two errors, as they could easily cancel out
- different weights of the odd and even numbers is to detect a digit transposition error (where two adjacent numbers are swapped).

CRCs

Generator Polynomials

- Generator polynomial is n-degree polynomial over $GF(2)$

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

- over finite field $GF(2)$ so
  - coefficients $a_i \in \{0,1\}$
  - addition means XOR
  - multiplication means AND
- Representation can be binary numbers $a_n a_{n-1} \ldots a_0$
  - $n+1$ coefficients, but a_n must be one, so is omitted
  - same confusions as for binary, e.g., least-sig. vs most-sig. bit first
  - can be written as hexadecimal
- Selection of generator is critical
  - good examples exist for $n = 8, 16, 32, 64$
- e.g., CRC-4-ITU = 10011 $= 1x^4 + 0x^3 + 0x^2 + 1.x + 1 = x^4 + x + 1$

$$\begin{aligned} x(0) &= 0+0+1 = 1 \\ x(1) &= 1+1+1 = 1 \end{aligned}$$

e.g., CRC-16-CCITT, used in many transmission protocols

$$x^{16} + x^{12} + x^5 + 1$$

In hex: 0x1021 (NB: the first bit must be 1, so don't need to represent it)
In binary: 1 0001 0000 0010 0001

e.g., CRC-32 used for Ethernet (and many others)

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$$

CRCs

Polynomial division

- Computation of CRC corresponds to division (in $GF(2)$), so we are trying to determine $Q(X)$ and $R(X)$ such that

$$M(x) \cdot x^n = Q(x) \cdot G(x) + R(x)$$

where
  - $M(x) \cdot x^n$ is the original message (as a polynomial) with n zeros added at the end.
  - $G(x)$ is the generator polynomial
  - $Q(x)$ is the "quotient" polynomial, which we don't care about
  - $R(x)$ us the "remainder" polynomial, whose binary representation is the CRC
- Calculation is like doing long-division
  - except no carries

One nice thing about CRCs is that they can easily be implemented in hardware, so, e.g., routers that have only nanoseconds to process a packet can compute a CRC for every packet.

n-bit CRC applied to a data block of arbitrary length will detect any single error burst not longer than n bits and will detect a fraction 12^n of all longer error bursts. They have other complicated error detection capabilities, beyond our scope.

CRCs

```
    Q(x)  01010110
         _____________________Q(x)_____________
x^n M(x)  0101011100000000
         -000000000          0
         = 101011100000000
          -100000111         1
         =  01011011000000
           -000000000        0
         =   1011011000000
            -100000111       1
         =    011010110000
             -000000000      0
         =     11010110000
              -100000111     1
         =      1010101100
               -100000111    1
         =       010100010
                -000000000   0
    R(x) =        10100010 = CRC
```


8 bit CRC-8-ATM of the 8 bit ASCII code for 'W' $= 87_{10} = 01010111_2$
Input message = 01010111
Polynomial = 100000111 $= x^8 + x^2 + x + 1$

We just subtract multiples of 1 or 0 times the polynomial from the message, but we don't carry, so we effectively "borrow from infinity"

Assignment

1. Implement a CRC calculator that allows you to input an arbitrary message and polynomial.
2. Calculate the CRC-8-ATM of the 8 bit ASCII codes for the following message:

   The rain in spain falls mainly on the plane.

NB: Matlab has its own CRC routines, but I want you to write your own. You can, however, use Matlab's to check your work.

Hash Functions

- Hash Functions are used in
  - creating hash tables (associative arrays implemented this way)
  - cryptography
  - to create checksums
- Random hash functions have several properties:
  - ideally they would be random oracles
    - their output for a given input is always the same
    - it is ideally random, uniformly over output domain
    - you can't tell the input from the output, so its one way
  - collision resistance
    - hard to find two inputs that hash to the same output
    - can provide protection against intentionally changed data
  - fixed length output for input message of arbitrary length
- CRCs don't achieve these properties
- Examples
  - MD5 (message-digest) [Riv92]
  - SHA-2 (set of hash functions)
  - lots of others

No real function is really an random oracle, but the concept is often used in proofs of security when these functions are used cryptographically.

See the birthday-paradox for the problem of collisions.

Incidentally, think about the entropy of the resulting hash if included in a message as a checksum?

Hash Function Example: MD5

MD5 (message-digest) [Riv92]

- input: message of arbitrary length
- output: 128 bits
  - digest (or summary) of the message
- basic
  - designed for 32 bit machines
  - uses 4 x 32 bit buffers
  - defines 4 x bitwise-parallel operations (functions)
  - uses 64 element table: $T[i] = \lfloor 2^{32} \times |sin(i)| \rfloor$, for i in radians
  - repeat a complex set of the above on 16 x 32 bit word blocks of the input
- not cryptographically secure, but often used as check
  - chance two messages produce same output is small, but not as small as we would like (conjecture was 1 in 2^{64} but actually there is a flaw)

Retransmission

Two main ways of dealing with detected errors

- Positive ACKs of all correct data
  - all data must be ACK'd, or is assumed incorrect
  - sender waits for a timer, and if doesn't get an ACK back in that time, automatically resends
  - Example: TCP (Transmission Control Protocol) on the Internet
- Negative ACK of errored data
  - when receiver detects and error, it asks sender for a new copy
  - makes more sense if delay is an issue (don't have to wait for timeout)

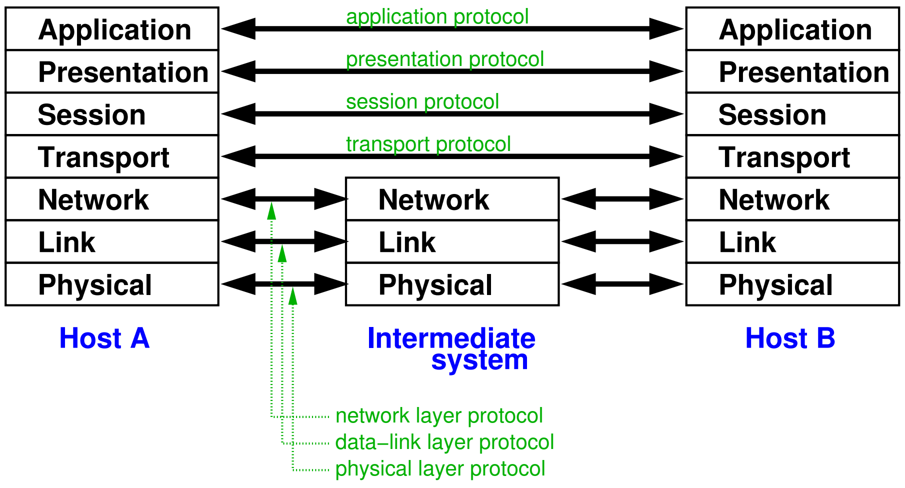
I have seen both called Automatic Repeat reQuest (ARQ), but it makes more sense to use that term for the former.

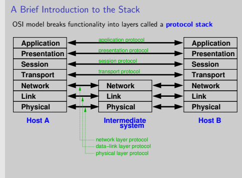
Section 3

The Stack

A Brief Introduction to the Stack

OSI model breaks functionality into layers called a **protocol stack**



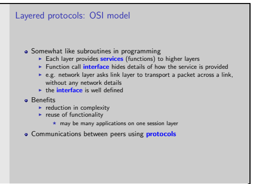


Additional references:

http://www.geocities.com/SiliconValley/Monitor/3131/ne/osimodel.html
http://www2.rad.com/networks/1994/osi/layers.htm
http://www.webopedia.com/quick_ref/OSI_Layers.asp
http://en.wikipedia.org/wiki/OSI_model

Layered protocols: OSI model

- Somewhat like subroutines in programming
  - Each layer provides **services** (functions) to higher layers
  - Function call **interface** hides details of how the service is provided
  - e.g. network layer asks link layer to transport a packet across a link, without any network details
  - the **interface** is well defined
- Benefits
  - reduction in complexity
  - reuse of functionality
    - ★ may be many applications on one session layer
- Communications between peers using **protocols**



Encapsulation

Lower layers deal with higher layer by

- treat information from higher layer as “black box”.
  - don’t look inside data
  - just treat as bunch of bits
- allowed operations on the data
  - just break data into blocks
  - **encapsulate** the blocks, by adding
    - headers (e.g. addresses)
    - trailers
- when passing back to higher
  - layers strip headers
  - join blocks back together

Encapsulation
Lower layers deal with higher layer by
treat information from higher layer as "black box".
don't look inside data
just treat as bunch of bits
allowed operations on the data
just break data into blocks
encapsulate the blocks, by adding
headers (e.g. addresses)
trailers
when passing back to higher
layers strip headers
join blocks back together

Layer 1: Physical layer

Function: Transmission of raw bit stream between devices.
Services: Physical connection, Binary modulation, frequency, ...
Issues: # pins/wires, duplex, serial/parallel, modulation, ...
Media:

- copper wire: e.g. coax, twisted pair (CAT-3/CAT-5), RS-232, USB
- lasers (fibre optics)
- lasers (free air)
- microwave, RF, satellite, ...
- infra-red
- carrier pigeons (RFC 1149) ;-)

Layer 1: Physical layer
Function: Transmission of raw bit stream between devices.
Services: Physical connection, Binary modulation, frequency, ...
Issues: # pins/wires, duplex, serial/parallel, modulation, ...
Media:
copper wire: e.g. coax, twisted pair (CAT-3/CAT-5), RS-232, USB
lasers (fibre optics)
lasers (free air)
microwave, RF, satellite, ...
infra-red
carrier pigeons (RFC 1149) ;-)

Layer 2: Data-link layer

Function: provide reliable transport of information between a pair of adjacent nodes.
Services: creates frames/packets, **error control**, flow control
Issues: Medium Access Control (MAC), headers/trailers, ...
Examples:

- **Ethernet**
- Token-ring
- IEEE 802.11 (Wi-Fi)
- FDDI (Fiber Distributed Data Interface)
- ATM (Asynchronous Transfer Mode) (also layer 3)
- POS (Packet over SONET)
- PPP (Point to Point Protocol)

Layer 2: Data-link layer
Function: provide reliable transport of information between a pair of adjacent nodes.
Services: creates frames/packets, error control, flow control
Issues: Medium Access Control (MAC), headers/trailers, ...
Examples:
- Ethernet
- Token-ring
- IEEE 802.11 (Wi-Fi)
- FDDI (Fiber Distributed Data Interface)
- ATM (Asynchronous Transfer Mode) (also layer 3)
- POS (Packet over SONET)
- PPP (Point to Point Protocol)

Layer 3: Network layer

Function: forwarding packets from end-to-end
Services: packet forwarding, some congestion control
Issues: determining what routing to use
Examples:

- **IPv4 (Internet Protocol version 4)**
- IPv6 (Internet Protocol version 6)
- ARP (Address Resolution Protocol)
- ATM (Asynchronous Transfer Mode) (also layer 2)
- Routing protocols (e.g. OSPF, IS-IS, RIP, EIGRP)

Layer 3: Network layer
Function: forwarding packets from end-to-end
Services: packet forwarding, some congestion control
Issues: determining what routing to use
Examples:
- IPv4 (Internet Protocol version 4)
- IPv6 (Internet Protocol version 6)
- ARP (Address Resolution Protocol)
- ATM (Asynchronous Transfer Mode) (also layer 2)
- Routing protocols (e.g. OSPF, IS-IS, RIP, EIGRP)

Layer 4: Transport layer

Function: reliable end-to-end transport of data
Services: multiplexing, end-to-end error and flow control
Issues: congestion control algorithm
Examples:

- **TCP (Transmission Control Protocol)**
- UDP (User Datagram Protocol)
- SCTP (Stream Control Transmission Protocol)
- RTP (Real-time Transport Protocol)

Layer 4: Transport layer
Function: reliable end-to-end transport of data
Services: multiplexing, end-to-end error and flow control
Issues: congestion control algorithm
Examples:
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)
- SCTP (Stream Control Transmission Protocol)
- RTP (Real-time Transport Protocol)

Layer 5: Session layer

Function: combine logically connected transmissions
Services: group several connections into a session
Issues: what to use it for?
Examples:

- NFS = Network File System
- SMB = Server Message Block

Layer 5: Session layer
Function: combine logically connected transmissions
Services: group several connections into a session
Issues: what to use it for?
Examples:
- NFS = Network File System
- SMB = Server Message Block

Layer 6: Presentation layer

Function: specific regularly requested functions.
Services: encryption, **compression**, ...
Issues: want to do compression before encryption, but compression may be done by a lower layer (see coding theorems later on)
Examples:

- SSL (Secure Sockets Layer) (at a stretch)

Layer 6: Presentation layer
Function: specific regularly requested functions.
Services: encryption, compression, ...
Issues: want to do compression before encryption, but compression may be done by a lower layer (see coding theorems later on)
Examples:
SSL (Secure Sockets Layer) (at a stretch)

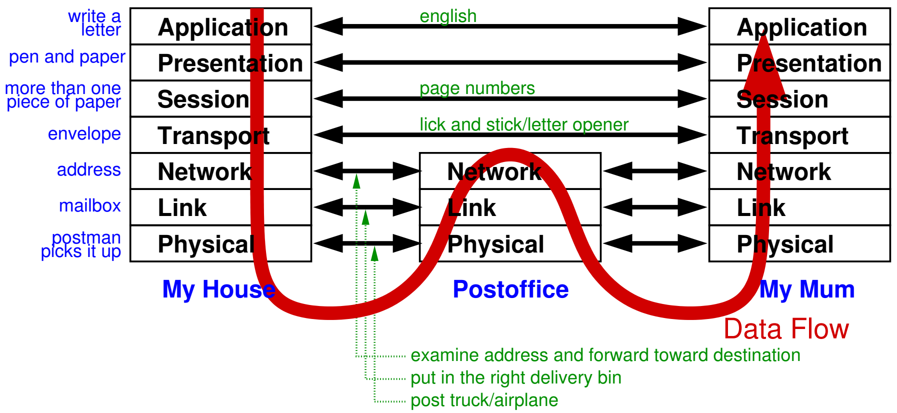
Layer 7: Application layer

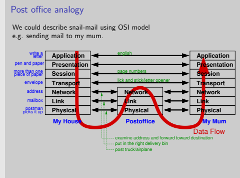
- E-mail (POP, IMAP, SMTP)
- File transfer (FTP — File Transfer Protocol)
- Remote terminal (Telnet, SSH, ...)
- WWW (HTTP — Hyper-Text Transfer Protocol)
- File sharing (Gnutella, Napster, Kazaa, ...)
- Video conferences
- Newsgroups
- NTP (Network Time Protocol)
- VoIP (Voice over IP)
- Games (Quake, MMORP, ...)
- RFC 2324: Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)

Layer 7: Application layer
E-mail (POP, IMAP, SMTP)
File transfer (FTP — File Transfer Protocol)
Remote terminal (Telnet, SSH, ...)
WWW (HTTP — Hyper-Text Transfer Protocol)
File sharing (Gnutella, Napster, Kazaa, ...)
Video conferences
Newsgroups
NTP (Network Time Protocol)
VoIP (Voice over IP)
Games (Quake, MMORP, ...)
RFC 2324: Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)

Post office analogy

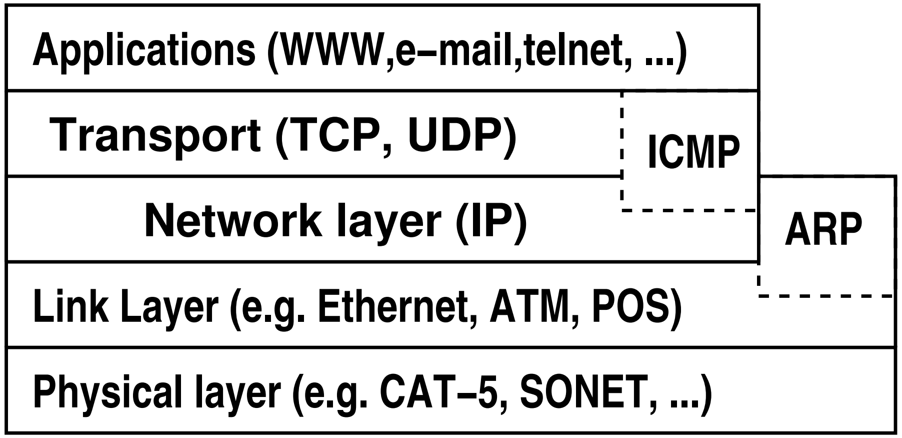
We could describe snail-mail using OSI model
e.g. sending mail to my mum.

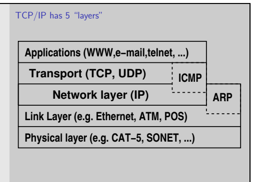




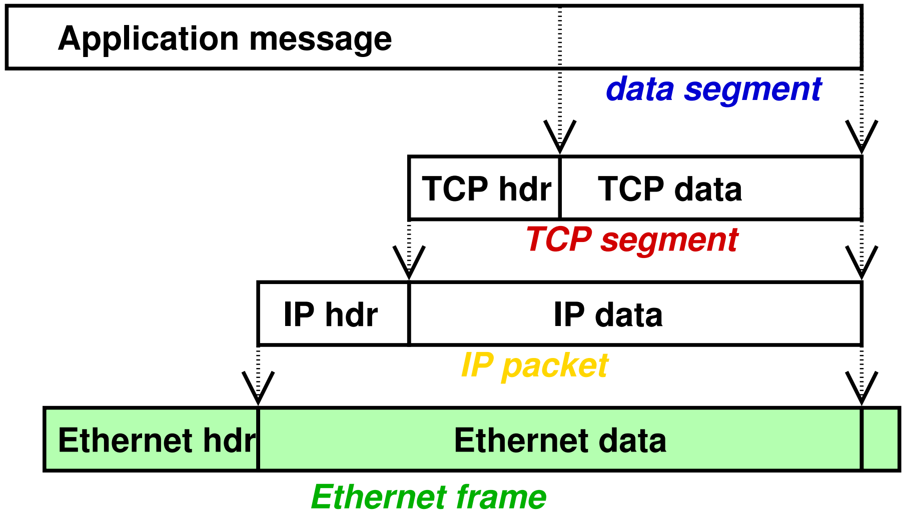
Another example "James Bond Meets The OSI Model" is given at http://www.lewistech.com/rlewis/Resources/JamesBondOSI2.aspx

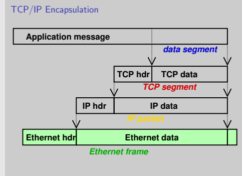
TCP/IP has 5 "layers"



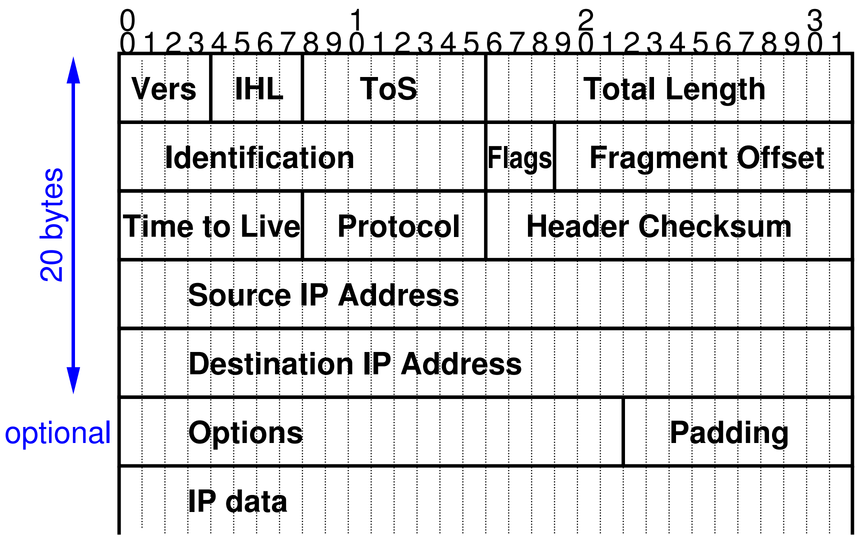


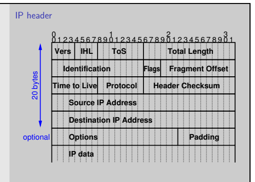
TCP/IP Encapsulation



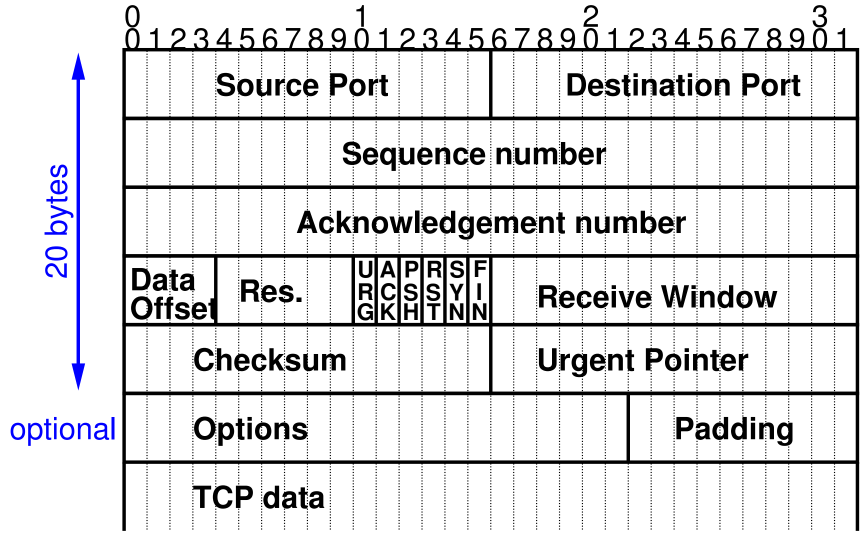


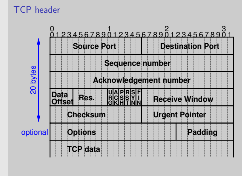
IP header



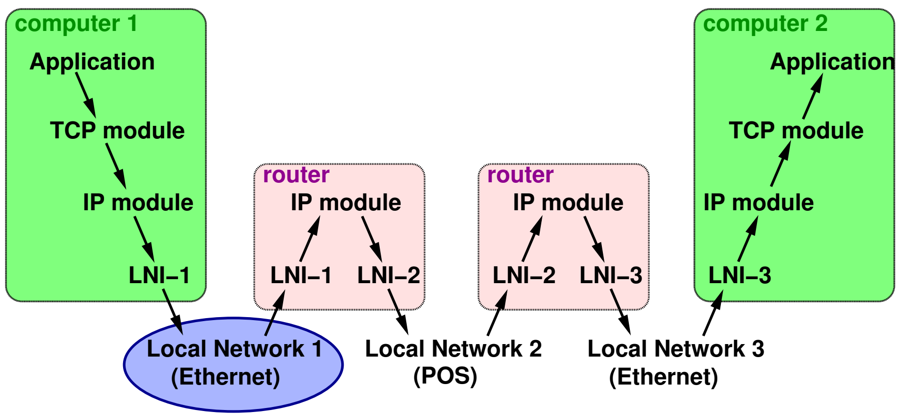


TCP header

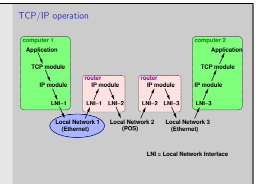




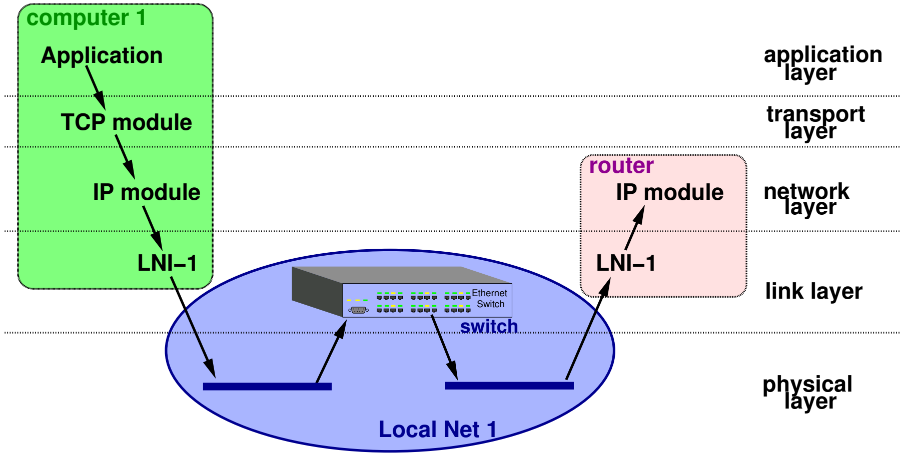
TCP/IP operation

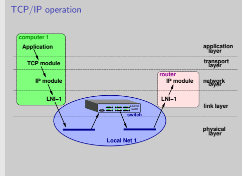


LNI = Local Network Interface



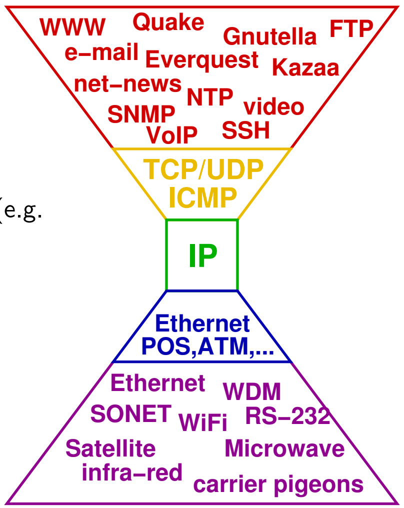
TCP/IP operation

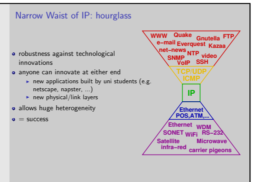




Narrow Waist of IP: hourglass

- robustness against technological innovations
- anyone can innovate at either end
  - new applications built by uni students (e.g. netscape, napster, ...)
  - new physical/link layers
- allows huge heterogeneity
- = success





Broken layering

TCP/IP layers are broken more often than not

- ICMP – uses IP, but controls its operation
- BGP is a routing protocol (IP layer), but is routed
- IP over ATM over IP over ATM over SONET
- anything involving MPLS
- often services are provided at multiple layers: error and flow control, e.g. error control in SONET (sort-of physical), link layer, IP, TCP, ...

OSI standards are too complicated

- Q: What do you get when you cross a mobster with an international standard?

- Paul Mockapetris

Broken layering

TCP/IP layers are broken more often than not

- ICMP – uses IP, but controls its operation
- BGP is a routing protocol (IP layer), but is routed
- IP over ATM over IP over ATM over SONET
- anything involving MPLS
- often services are provided at multiple layers: error and flow control, e.g. error control in SONET (sort-of physical), link layer, IP, TCP, ...

OSI standards are too complicated

- Q: What do you get when you cross a mobster with an international standard?

- Paul Mockapetris

Matthew Roughan (School of Mathematical S Information Theory October 10, 2013 41 / 45

What's best?

- The best approach depends on the "natural" block length and BER
- e.g., TCP/IP
  - many links have very low error rates (BER might be 10^{-12})
  - TCP/IP packets are 40-1500 bytes (typically)
    - checksums are in the headers (extra stuff added to data, like addresses)
  - IPv4 packets only have a checksum for header, not data
    - 1's complement of 1's complement of sum of heads 16 bit words (by default there are 20 octets or 10 x 16 bits words))
    - calculated as if checksum bytes were zero
    - must be recalculated at each hop as TTL changes
    - packets that fail are discarded (IP is best effort only)
    - 16 check bits out of 160 bits – fairly strong checking, but only of header
  - TCP adds 16 bit CRC checksum for header and data, and does retransmission of errored packets
    - pretty "weak" (16 bits out of potentially 12000)
    - only makes sense for low error rates
    - see link-level (e.g., Ethernet) below

What's best?

- The best approach depends on the "natural" block length and BER
- e.g., Ethernet
  - Ethernet goes over many mediums (copper, optical fibre)
  - Frames are $\leq$ 1514 bytes
  - 32-bit CRC (CRC-32)
  - generator polynomial

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$$

  - longer checksum than TCP
    - TCP/IP packets encapsulated in Ethernet (often)
    - link-level reliability is enhanced (underlying BER is still assumed to be fairly low)
    - TCP checks are mainly looking for (hopefully) rare cases where something goes wrong end-to-end, so it doesn't need to be as strict
  - just to complicate things: applications may have further checks on integrity of messages.

What's best?

- The best approach depends on the "natural" block length and BER
- e.g., Deep space communications
  - low signal to noise ratio (SNR), so there is a large BER
  - high delays (minutes, due to speed of light delay)
  - resending every time a single bit is bad would be unworkable
- in this case we need something better than error detection
- other cases:
  - when there is no reverse channel
    - e.g., storage (such as CDs)
    - e.g., broadcast or multicast (e.g. digital TV)
  - channel is expensive, e.g., physical transport

Further reading I

- Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, John Wiley and Sons, 1991.
- Robert G. Gallager, *Information theory and reliable communication*, John Wiley and Sons, 1968.
- David J. MacKay, *Information theory, inference, and learning algorithms*, Cambridge University Press, 2011.
- R. Rivest, *The MD5 message-digest algorithm*, IEFT RFC 1321, April 1992, `http://tools.ietf.org/html/rfc1321`.