Communications and Information Sharing
CS 118
Computer Network Fundamentals

Peter Reiher

So What?

- We can now say something about how much information you can push through a channel
- Let the source have entropy H (bits per symbol)
- Let the channel have capacity C (bits per second)
- Then we can transmit $C/H - \varepsilon$ symbols per second
  - For arbitrarily small ε
- But never more than C/H

Information

- Weighted sum of all probabilities

$$H = -\sum_i p_i \log_2(p_i)$$

Why negative?

- $p_i \leq 1$
  - Means $\log_2(p_i)$ is negative
  - But entropy should be positive
- Smaller p_i means more choice
  - More choice should mean higher entropy

What about zero probabilities?

- What if $p_i = 0$?
- *log (0)* is not defined
- But we're summing $p_i log(p_i)$

$$\lim_{p \to 0+} p \log(p) = 0$$

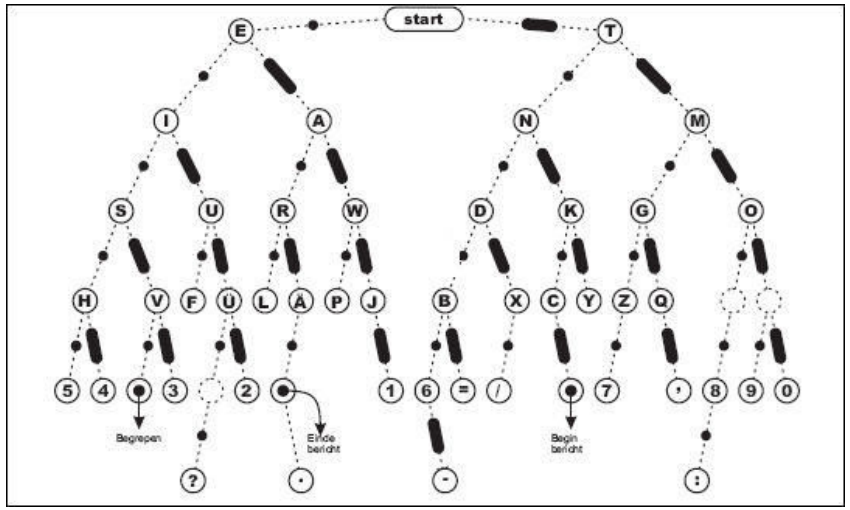
- So treat $p_i log(p_i)$ as 0 when $p_i = 0$

Predictability

- What if we're not sending random bits?
- Maybe it's English text in ASCII
- Maybe it's Morse code
- Then the $p_i(j)$'s are not uniform
  - Some symbols are more likely, given the symbols already sent
  - Entropy is lower than the max
  - Meaning we can squeeze more information through

Morse code

- Code representation:
  - E ● 1
  - T — 3
  - A ●— 5
  - O ——— 11
  - I ●● 3
  - N —● 5
  - S ●●● 5



What if choices aren't equal?

- YELLO_
  - What comes next?

We want a compact representation of characters that we would like to send

- PIT_
  - What comes next?

- "Next letter" in English isn't 1 of 26
  - Roughly 50% redundant

A look at Morse code again…

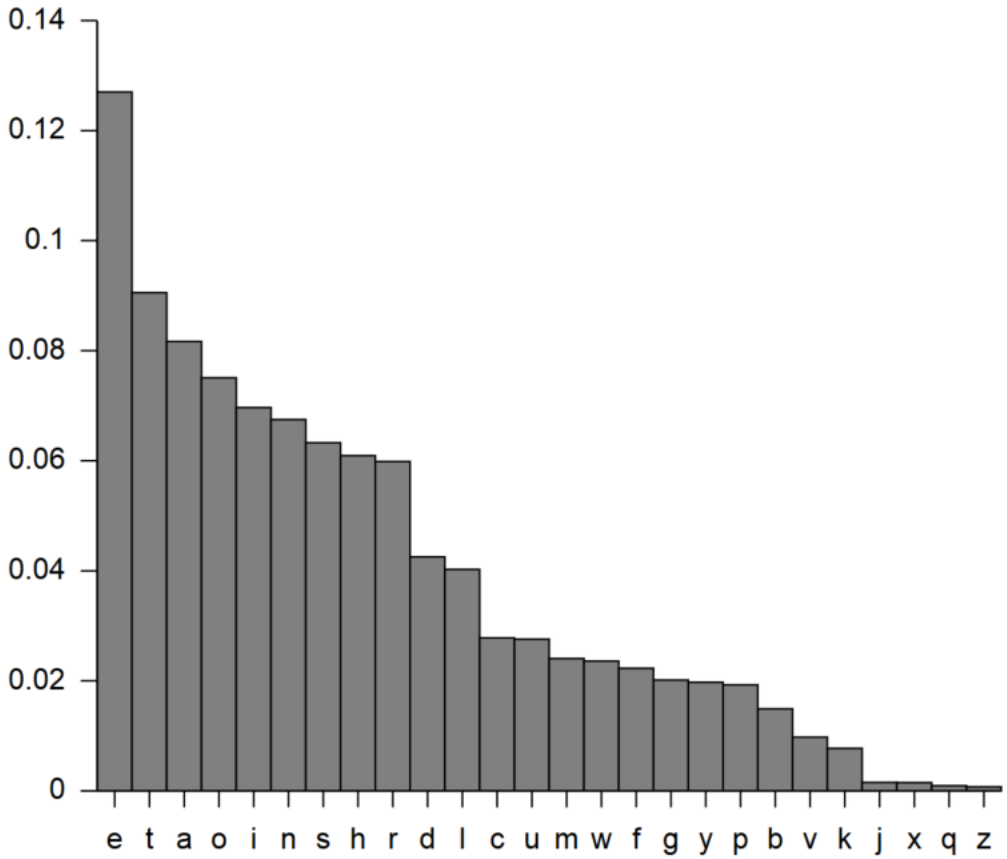
- Time units:
  - Dot = *t*
  - Dash = *3t*
  - Inter-symbol gap within a letter = *t*

The information is how much randomness is in the info; same amount of information in 500 characters by doing a clever encoding

ex. Morse code does this - they encode by making the letters they see most frequency encode with less actions.

American English letter frequencies

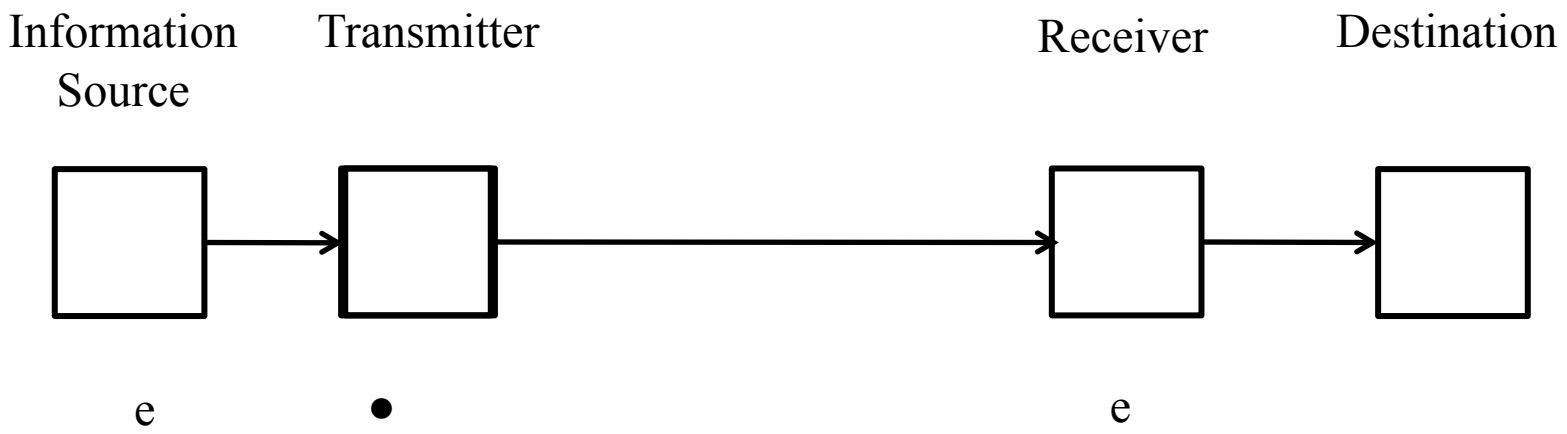
- Basic order:
  - E
  - T
  - A
  - O
  - I
  - N
  - S



How Do We Get More Through?

- Encoding it properly
- In essence, "short" signals for common things
- Long signals for uncommon things
- E.g., Morse code
  - Common letters are few dots and dashes
  - Uncommon letters require more dots and dashes
  - Each dot or dash takes up time on the line
  - They didn't do it perfectly . . .

Who Does This Coding?



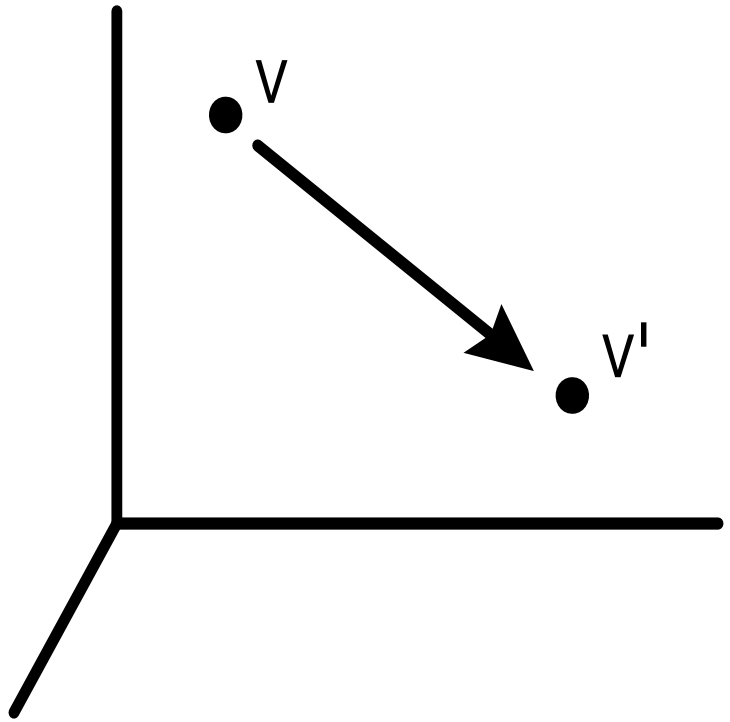
And the receiver decodes

The perils of sharing

- Shared state may be inaccurate
  - Channel errors
  - Time (i.e., 'staleness')
- Capacity is finite
  - Nobody can know everything

You have an old state; the state has changed, receiver knows that the state is in condition 'A', but when state is changed to condition 'B', receiver still feels like state is in condition 'A'. Receiver is in 'staleness'

Simple state

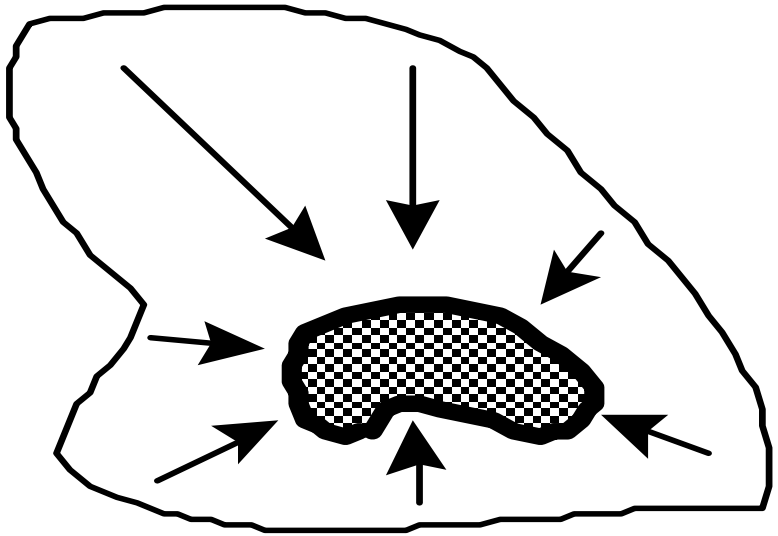


How does communication affect state?

- Knowledge doesn't stay still…

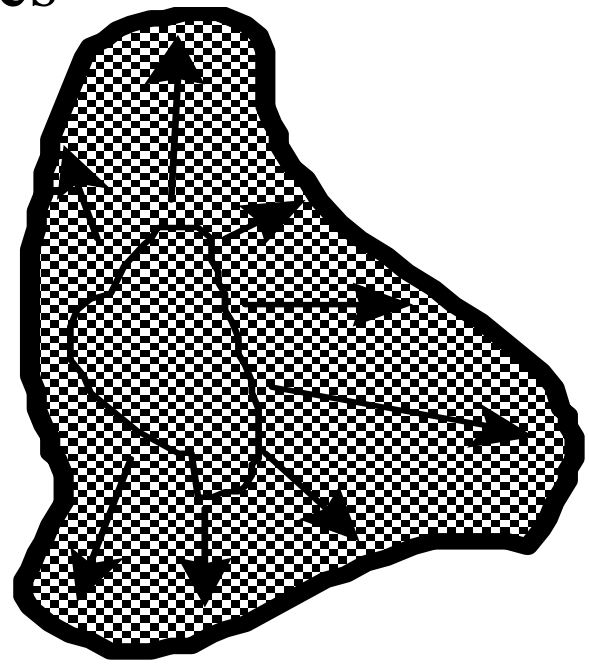
Effect of receiving

- Entropy decreases
  - Receiver knows more about the transmitter



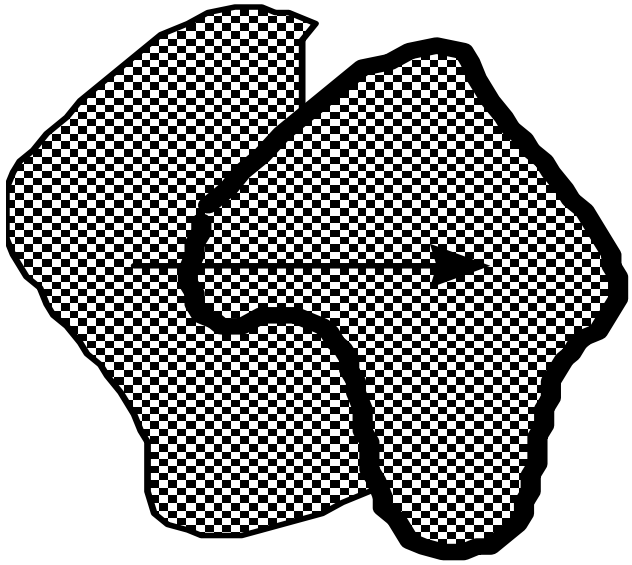
Effect of time

- Entropy never decreases over time
  - Usually increases



Effect of sending (1)

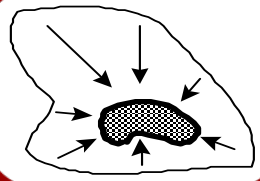
- Sending information about your state
  - Makes your view of receiver state fuzzier

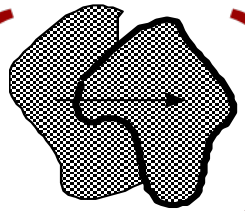


Effect of sending (2)

- Entire system entropy never decreases
  - Receiver's model of transmitter entropy decreases in entropy, so sender's model of receiver MUST increase in entropy

$$S(\quad) + S(\quad) \geq 0$$





Putting it all together – CTP

Character Transfer Protocol

- Character transfer protocol
  - (we're creating this as an example)
  - Sending a message one letter at a time
- Assume a perfect channel
  - No errors in transmission, ever

Protocol - set of rules that we use
in order to communicate with somebody

CTP events

- Starting condition
  - Both sides share rules (protocol)
  - And share endpoint info (link)

CTP rules

- Both endpoints start in the state:
  - NOT-CONNECTED
- Use phone-call protocol to get to:
  - CONNECTED
- Connected transmitter
  - sends characters one a time
- Connected receiver
  - receives characters one at a time until CLOSED
- At EOF
  - transmitter closes connection

EOF happens in a perfect connection

From not-connected to connected

- Simple phone-call protocol
  - Transmitter initiates, waits for response
  - Receiver responds when asked
  - Once confirmed, both sides enter CONNECTED state

Remember – perfect channel,
so no need for timeouts or exceptions

Simple transfer

- Character at a time
  - Transmitter sends characters in order
  - Receiver collects characters and places them in order

Perfect channel means receiver collects character in order

- What state is shared?
  - CONNECTED state
  - The character

From connected to closed

- Final change in shared state
- Lets receiver know
  - Transfer can be stopped
  - Message is complete

CTP evolution

- Successive increased shared state:
  - Agree to change from not-connected to connected
    - Using the phone-call protocol
  - Agree on each character sent
    - In a perfect channel, nothing is lost or reordered, so transmitter knows what receiver gets (i.e., agreement always happens)
  - Agree to end transfer

Limits of CTP

- Assumes a perfect channel
  - Ignores loss, reordering, flow control, etc.
- Inefficient
  - The agreed state is augmented one character at a time

Once closed, what do we know?

- The message!
  - WHY? – the succession of shared state

    we know the message because we have been building up the shared state, so now we have an idea of what the message is

Back to predictability

- We know more than we think
  - Can send groups of characters
  - Can confirm using "checksums"

build characters into a larger collection and send that

checksum is a mathematical derivation of what is suppose to be sent, and receivers can verify that has been sent

Putting it all together – FTP

File Transfer Protocol

- A more efficient version of CTP

Starting point

- Share protocol rules
- Already know each other's endpoint
- Know we're using those rules
  - TCP port 22

particular OS understanding of what happens at a network

From not-connected to connected

- Uses TCP's version of the phone-call protocol:
  - Transmitter sends SYN
  - Receiver sends SYN-ACK
  - Once confirmed, both sides enter ESTABLISHED TCP state
- What's the difference?
  - Over a perfect channel, NOTHING

What about an imperfect channel?

- Some characters might be dropped
  - "H e l l o" becomes "H e l l"
- Some characters might be changed
  - "F a r" becomes "F u r"
- Some characters might be added
  - "H e a t" becomes "H e a r t"
- Or maybe it came through unchanged
- How can the receiver know what happened?

Better transfer

a whole set of bits that represents many chars

- Block at a time collection of chars, and send
  - Transmitter sends characters in order inside blocks, labeled with a checksum
  - Receiver collects blocks, verifies checksums, and places them in order
- What state is shared?
  - Connection state
  - The CHECKSUMS!

checksum is some number of bits;
we are sending more than our expected number of bits

checksum is not fresh information,
if it were a perfect channel, checksum is a useless overhead

however, if it is not a perfect channel, the checksum will allow us to determine if the transfer is good

in other words, checksum made the channel more perfect

longer checksums decrease the probability for error in the channel

FTP evolution

- Successive increased shared state:
  - Agree to change from IDLE to ESTABLISHED
    - Using the TCP protocol
  - Agree on each checksum sent
  - Agree to end transfer

From connected to closed

- Final change in shared state
- Lets receiver know
  - Transfer can be stopped
  - Message is complete – WHY?

FTP's leap of faith

- Correct file transfer IFF:
  - Sequence of correct blocks
  - Each block is correct IFF checksum is correct

Examples of FTP-like exchanges

- File transfer
- Web request/response
- Netflix

checksum - use sequence to order blocks,
ex. there are 13 blocks sent, you got 12 and connection lost; you just have to know how many blocks were sent before you lost the connection

What other states can we add?

- Pacing match speed of sender and receiver so there's no error there
  - How fast does the transfer go?
  - At constant or changing tempo?
- Number of outstanding messages
  - Messages sent but not yet received
  - Or perhaps received, but not yet acknowledged
- Amount of reordering

when system get overloaded, things don't go wel;

What should you assume?

- As little as possible!

Postel Principle

John Postel - development of the internet from UCLA

- Be conservative in what you do…
  - Transmit only what you think will help
  - Transmit only what you expect will be understood
- Be liberal in what you tolerate.
  - If a message could mean multiple things, allow as many as possible
  - Do not assume malice where incompetence will suffice (errors happen)

Information

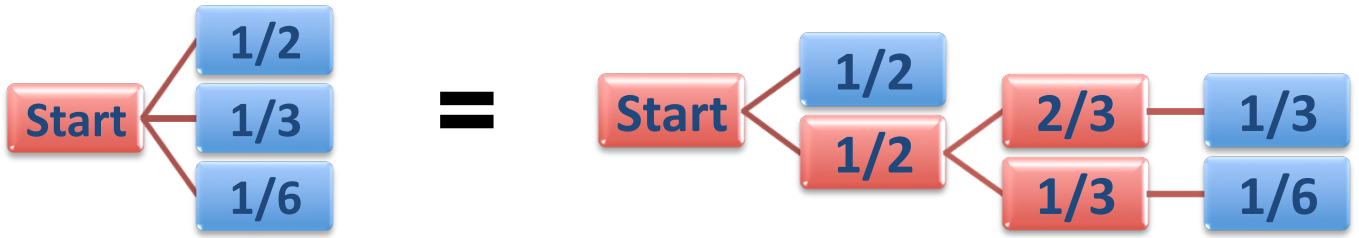
- Weighted sum of all probabilities

$$H = -\sum_i p_i \log_2(p_i)$$

Why exactly this function?

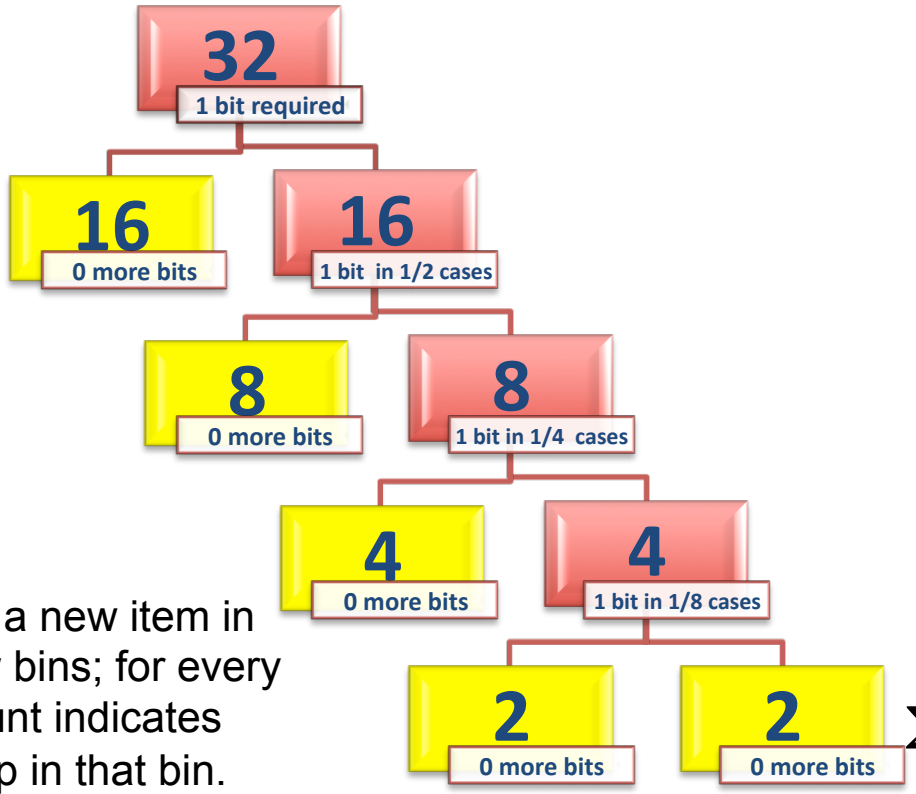
- H is continuous
- If p_i are equal $\left(\frac{1}{n}\right)$, then H increases with n
  - More choice should mean more entropy
- Cascaded choices act like weighted sums

$$H\left(\frac{1}{2},\frac{1}{3},\frac{1}{6}\right) = H\left(\frac{1}{2},\frac{1}{2}\right) + \frac{1}{2}H\left(\frac{2}{3},\frac{1}{3}\right)$$



Unbalanced tree of 32 choices

tree of 50 % redundancy



1 bit (avg)

0.5 bit (avg)

0.25 bit (avg)

0.125 bit (avg)

Σ 1.875 bits (avg)

Consider putting a new item in one of the yellow bins; for every 32 items, the count indicates how many end up in that bin. The bin use is heavily biased in this example. How many bits are needed to classify the item?

Unbalanced tree – weighted sum

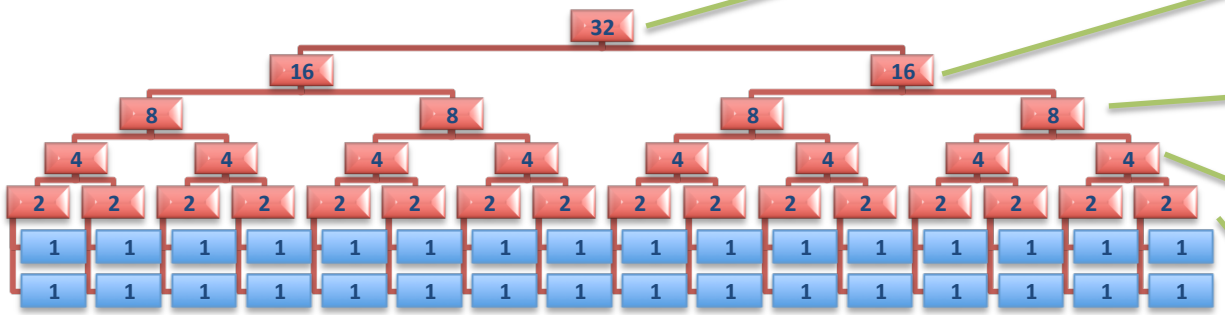
- $$H = -\left(\frac{16}{32}\log_2\left(\frac{16}{32}\right) + \frac{8}{32}\log_2\left(\frac{8}{32}\right) + \frac{4}{32}\log_2\left(\frac{4}{32}\right) + \frac{2}{32}\log_2\left(\frac{2}{32}\right) + \frac{2}{32}\log_2\left(\frac{2}{32}\right)\right)$$

$$H = -\left(0.5(-1) + 0.25(-1) + 0.125(-3) + 0.0625(-4) + 0.0625(-4)\right)$$

$$H = -(-0.5 - 0.50 - 0.375 - \ 0.25 \ - 0.25\ )$$

$$H = 1.875$$

Balanced tree of 32 choices



1 bit (avg)

1 bit (avg)

1 bit (avg)

1 bit (avg)

1 bit (avg)

Σ 5 bits (avg)

Consider putting a new item in one of the blue bins; for every 32 items, the count indicates how many end up in that bin. This time the counts per bin are the same, and the number of bins used is maximized. How many bits are needed to classify the item now?

file compression is like this - find redundancy, find that certain symbols occur more convert coding into shorter encodings

Balanced tree – weighted sum

32 different char * (1/32 * log(1/32)) = 5 as expected

- $$H = -32\left(\frac{1}{32}\log_2\left(\frac{1}{32}\right)\right)$$

$$H = -32\left(\frac{1}{32}(-5)\right)$$

$$H = 5$$

- For 32 choices, that's the maximum entropy

What if only 1 of 32 was possible?

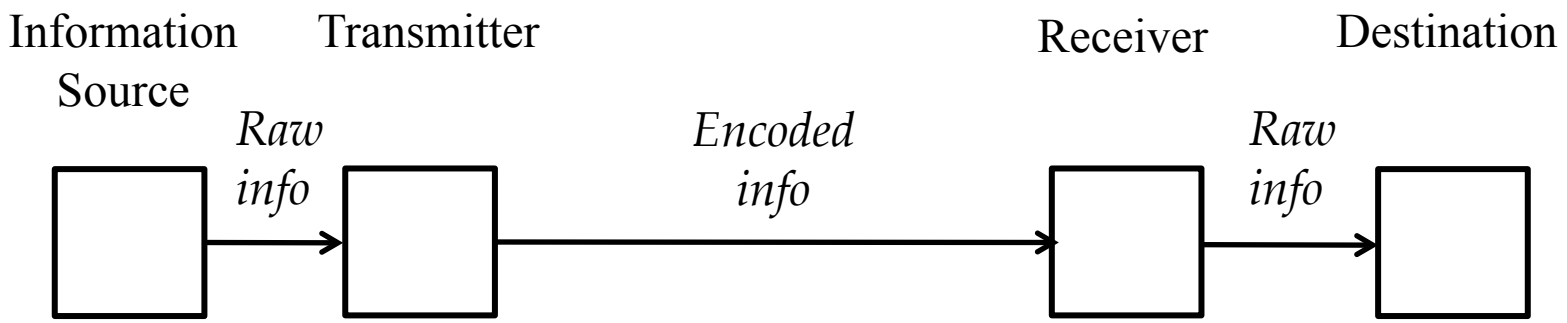
- Let's say 31 values have zero probability and one value has probability 1

$$H = -1\log_2 1 + 31 * 0\log_2 0$$

$$H = 0 + 31 * 0$$

- So H = 0
  - As expected, when there's no choice

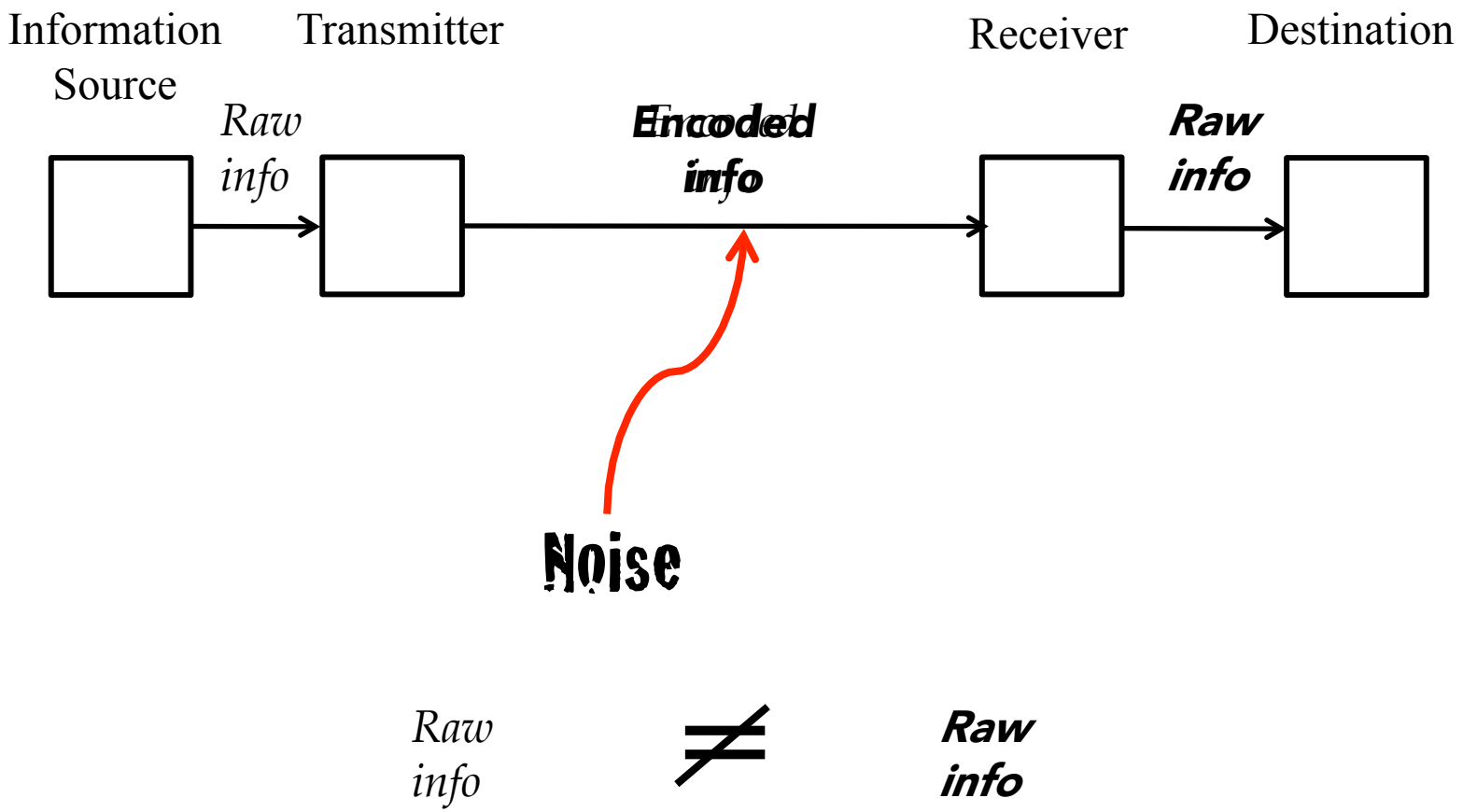
The Perfect Channel



The channel never alters the encoded information

The transmitter and receiver always convert perfectly back and forth

The Noisy Channel



What is a noisy channel?

- A channel that introduces errors into transmission
  - What is sent is not always what is received
- Noise can be of arbitrary nature
  - Always or sometimes
  - Affects some or all of message
- The capacity of a noisy channel is the maximum rate at which useful information can be transmitted
  - Information not altered by the noise

Equivocation

- A situation where a received symbol might have been caused by more than one sent symbol
- E.g., you got a 1, but I might have sent either 0 or 1

  what receiver got might be different from what sender sent
  ex. received 1 when .5 chance I get 1 and .5 chance I get 0
- How did that happen?
- Noise in the channel
  - Possibly altering my intended signal

Consider three cases

- 0% change
  - 0->0, 1->1
  - No noise
- 100% change
  - 0->1, 1->0
  - All the symbols change, but there's no ambiguity!
  - No noise!
- 50% change
  - 0->(0,1), 1->(0,1)
  - The symbols change with max randomness!
  - All noise – NO way to recover

max randomeness here

Consider a matrix of choices

	Receive a 0	Receive a 1
Send a 0	a	c
Send a 1	b	d

What does it mean?

pretty important slide to show how sending and receiving works

	Receive a 0	Receive a 1
Send a 0	a	c
Send a 1	b	d

- Sender transmits 0
  - (a+c)/(a+b+c+d) sent as 0
  - a/(a+c) received as 0
  - c/(a+c) received as 1
- Sender transmits 1
  - (b+d)/(a+b+c+d) sent as 1
  - b/(b+d) received as 0
  - d/(b+d) received as 1

What does it mean 2.0?

	Receive a 0	Receive a 1
Send a 0	a	c
Send a 1	b	d

- Receiver gets 0
  - (a+b)/(a+b+c+d) rec'd as 0
  - a/(a+b) sent as 0
  - b/(a+b) sent as 1
- Receiver gets 1
  - (c+d)/(a+b+c+d) rec'd as 1
  - c/(c+d) sent as 0
  - d/(c+d) sent as 1

Perfect Channel

Important slides about sending and receiving

	Receive a 0	Receive a 1
Send a 0	1	0
Send a 1	0	1

No noise

s

Swap the choices

	Receive a 0	Receive a 1
Send a 0	0	1
Send a 1	1	0

No noise

Merge things

	Receive a 0	Receive a 1
Send a 0	0	1
Send a 1	0	1

Total noise – no communication

Split things . . .

	Receive a 0	Receive a 1
Send a 0	.5	.5
Send a 1	1	0

More complex noise . . .

What is the capacity of this last channel?

- How many bits per second are we effectively communicating?
- Rate of channel = H(x) + H(y) - H(x,y)
  - Intuitively, the bits per second that the sender and receiver "share"
- Let's calculate that for our example
  - Using information from the matrix
  - Working on the assumption that the sender is equally likely to send 0 or 1

So what is the conditional entropy for this channel?

- First we need the entropy of the source
- H(x) = 1
  - Since two equally likely signals
- We also need the entropy of the receiver
  - More complex calculation, but H(y) comes out to .81 Read through the Shannon paper about the calculations
- And the joint entropy H(x,y)
  - That's 1.5 (also a bit more complex to calculate)

Where did I get those numbers?

- I hope you understand where H(x) = 1 came from
- But where did that .81 and 1.5 come from?
- From simple (if tedious) probability calculations
- Laid out in Shannon's paper
- You should try these for yourself

Working it out

- $R = H(x) + H(y) - H(x,y)$
- $R = 1 + .81 - 1.5$
- $R = .31$ Make sure to understand how to calculate this
- We're effectively communicating around 1/3 of a bit per second

But . . .

- What do I do with this result?
- It means, sort of, the bit I got is about 1/3 likely to be correct
- How can I know when it is and when it isn't?
- I can't, really
- But I can use encoding to improve my odds
- I can create a code that will help correct the errors

However, . . .

- No matter how clever my code is, I don't get to cheat on Shannon's limit
- On average, I will need to send ~3 1/3 encoded bits to transmit 1 actual bit more reliably
- When you work out my rate, it's still .31 bits per second
- And if I'm not maximally clever, it will be less
- NEVER more

Summary

- Communication is less than most think
  - Just syntax – not semantics or intent
- Information is based on states
  - Which is based on entropy (disorder)
- We can model how state evolves
  - Each side models the other
  - Successive steps in models are how we go from sharing state to transferring files
- Noise decreases the information we can pass
  - Encodings can correct errors
  - But cannot break the Shannon limit