

ELE3305 Computer Systems & Communications Protocols

# Module 4: Data Transmission

1

## Data Transmission

- Electrical Interconnections
- Baseband & Broadband Modulation
- Timing and Clock Recovery
- Standards

2

## Fundamental Problems

- Efficient use of bandwidth
- Radiation: EMI/RFI
- Noise
- Clocking of receiver
- Flow control (fast sender/slow receiver)
- Medium Access (MAC) – see next module
- Multiplexing – Internet TCP/IP protocols

3

## Baseband vs Broadband

- Baseband: direct digital encoding onto line.
- Typical examples: RS232, USB, Ethernet
- Broadband: where the digital data must be modulated somehow.
- Typical examples: voice-band modem, ADSL, wireless

4

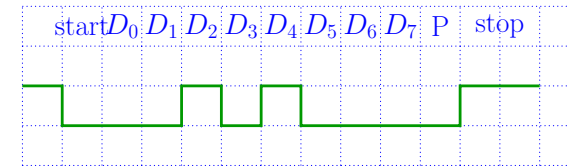
## Serial Connections

- Original was RS232. Now largely superseded.
- USB: Universal Serial Bus – automatic configuration, no user settings, hot pluggable, no external power required (mostly)
- USB much faster: high-speed 480 Mbps, full-speed 12 Mbps, low speed 1.5Mbps (high speed in version 2.0)
- Actual data rates approx 53 Mbyte/s, 1.2 Mbyte/s, 800 byte/sec. (IEEE1392 firewire 400 Mbps, 1392b 3.2 Gbps).
- Ethernet: 10Mbps everywhere, 100Mbps becoming common.
- Voice-band modems, Asymmetric Digital Subscriber Line (ADSL)

5

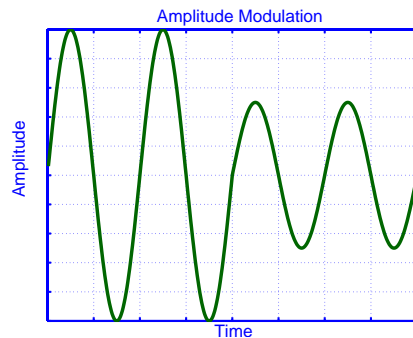
## Baseband Modulation

“start-stop” asynchronous transmission:



6

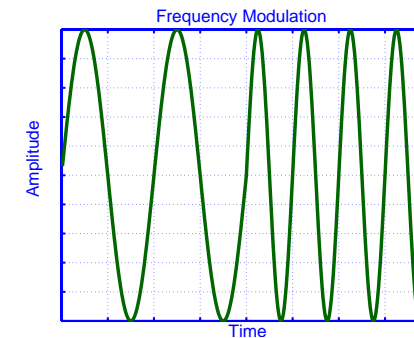
## Broadband Modulation



*Encode a 0 bit by a low amplitude, a 1 bit by a high amplitude (or vice-versa).  
Keep the carrier frequency constant*

7

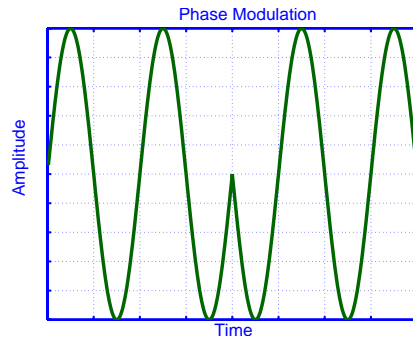
## Broadband Modulation



*Encode a 0 bit by a low frequency, a 1 bit by a high frequency (or vice-versa).  
Keep the amplitude constant.*

8

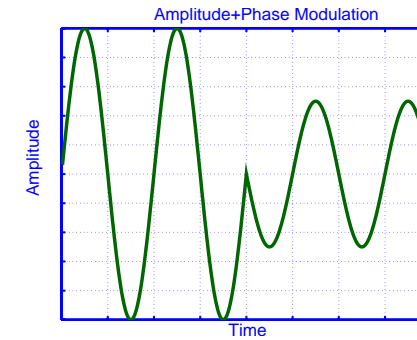
## Broadband Modulation



Encode a 0 bit by a carrier shift of  $0^\circ$ , a 1 bit by a carrier shift of  $180^\circ$  (or vice-versa).  
Keep the amplitude and frequency constant.

9

## Broadband Modulation



10

## Broadband Modulation

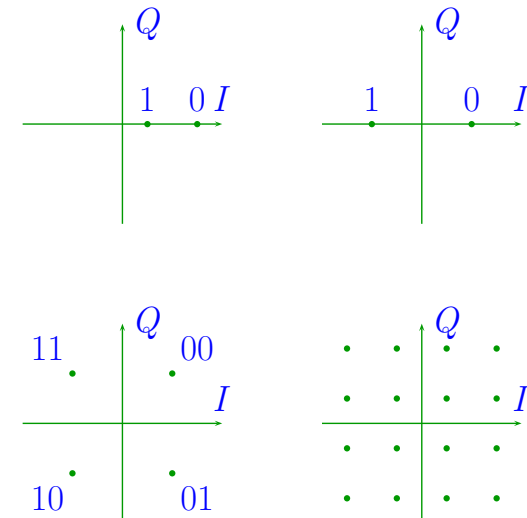
Encode:

- 00 bit pattern by a carrier shift of  $0^\circ$ , full amplitude
- 01 bit pattern by a carrier shift of  $0^\circ$ , half amplitude
- 10 bit pattern by a carrier shift of  $180^\circ$ , full amplitude
- 11 bit pattern by a carrier shift of  $180^\circ$ , half amplitude

Keep the frequency constant – easier to demodulate.

11

## Broadband Constellations



12

## Broadband Constellations

- Allows greater than one *bit per baud*
- V.22bis – 2400 bit/sec, 1200Hz/2400Hz, 600 baud modulation rate, QAM encodes 4 bits/ baud → 2400 bps
- V.32 – 9600 bit/sec, 2400 baud modulation rate, QAM encodes 4 bits/ baud → 9600 bps
- V.32bis (14k4), V.34 (28k8), V.90 (56k)
- Note term *baud*. A *baud* is the “signalling interval”. Could be more than one bit per signalling interval.
- Explanation at <http://en.wikipedia.org/wiki/Baud>
- Do not confuse with bits per second (bps)

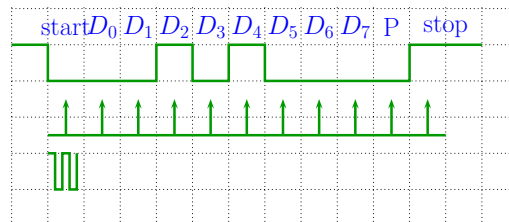
13

## Synchronization

- Also called ‘start-stop’ transmission.
- Uses oversampling clock (typically 16×)
- Start count on start-bit transition.
- Re-synchronizes on every character → inefficient.
- Overhead: start, stop, parity bits.

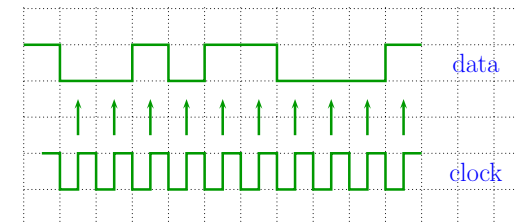
14

## Synchronization



15

## Synchronous Transmission



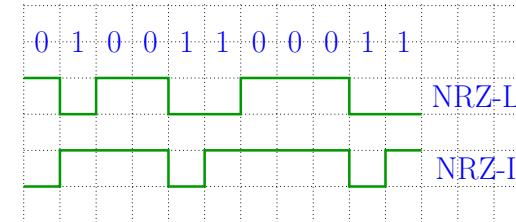
16

## Synchronous Transmission

- Continuous transmission of bit-stream.
- More efficient (no overhead)
- How to synchronize?
  - Transmit clock explicitly, or
  - Embed clock in data with special line code.

17

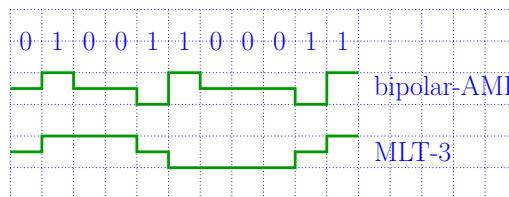
## Baseband Synchronous Codes



- NRZ-L = non return to zero, level
  - 0 = high
  - 1 = low
- NRZ-I = non return to zero, invert on ones
  - 0 = no transition at start of bit cell
  - 1 = transition at start of bit cell

18

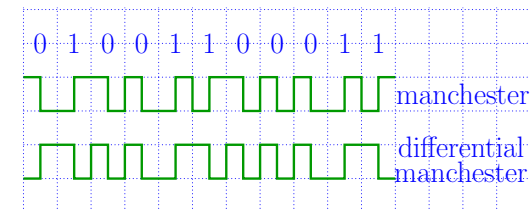
## Baseband Synchronous Codes



- Bipolar AMI (Alternate Mark Inversion)
  - 0 = no output
  - 1 = alternating  $\pm$  level
- Multilevel-3 (MLT-3)
  - 0 = no change
  - 1 = next level

19

## Baseband Synchronous Codes



- *Always a bit transition in the middle of bit cell*
- Manchester
  - 0 = high to low, 1 = low to high
- Differential Manchester
  - 0 = transition at start of bit cell
  - 1 = no transition at start of bit cell

20

## Baseband Synchronous Codes

- Problem with AMI/MLT type codes: long string of 0's results in constant level.
- Problem with constant (DC) levels.
- Problem in synchronizing phase-locked loop (PLL) in receiver to regenerate correct clock rate & phase.
- Therefore, need to induce transitions.

21

## Baseband Synchronous Codes

- 4B5B encoding
- 4 data bits mapped to 5 encoded bits
- Enough 1's & 0's to ensure transitions for receiver clocking
- Slight expansion of bandwidth

raw bits	encoded
0000	11110
0001	01001
...	...
1110	11100
1111	11101

22

## Baseband Code Characteristics

- Clocking: codes which have a transition in each bit cell may have the clock extracted from the data stream.
- DC component: Certain media cannot transmit DC (phone lines, RF channels). A long string of 1's in some codes (eg NRZ) gives rise to a DC level.
- Bandwidth: Biphase codes, although self-clocking, require effectively double the bandwidth (ie twice as many transitions on the line per bit encoded)

23

## Baseband Codes on Wired Ethernet

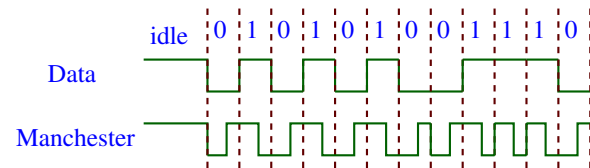
- Manchester codes used on 10Mbps 10BASE-T
- 4B5B and MLT used on 100Mbps 100BASE-TX

24

## Ethernet – Physical Layer

### 10BASE-T Ethernet:

- 10Mbps, Unshielded Twisted Pair (UTP)
- Manchester encoding
- 1=H → L, 0=L → H

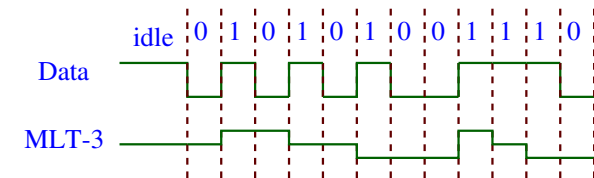


25

## Ethernet – Physical Layer

### 100BASE-TX

- 100Mbps, 2 pairs UTP
- encoded using 4B/5B encoding (4 bits in, 5 bits out)
- MLT3 line encoding - 3 levels (+, 0, -).
- Input 1 bit, transition to next level. Input 0 bit, no transition.



26

## Ethernet – Physical Layer

### 1000BASE-TX

- 1000Mbps, 4 pairs UTP
- encoding 8B/10B
- 4D-PAM5, 5-level symbol carries 2 bits, +/- 1V
- 2 bits per transition, all 4 wires to send at a time, gives 125MBaud signalling rate

27

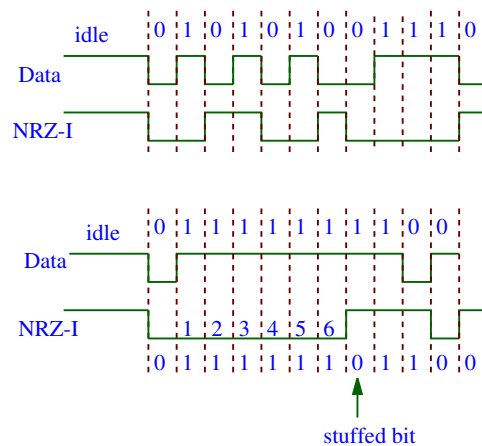
## USB – Physical Layer

- High-speed (USB2) 480 Mbps, full-speed 12 Mbps, low speed 1.5Mbps
- Two signal lines, +5 V power.
- Single differential wire
- full and high speed must use twisted pair
- shielding required on full/high speed, not required on low speed
- half-duplex (take turns to send/receive)
- Transactions in 1ms frame (125μs for high speed)

28

## USB – Physical Layer

- Encoding: NRZ-I: 0 = voltage change, 1 = no change
- Bit stuffing: long string of 1's causes no changes, hence may lose sync. After six consecutive 1's, transmitter inserts a 0



29

## Error Detection

- Idea: from a received message, we can tell if part of the information has been corrupted.
- Analogy 1: Each lighthouse on the coastline has a unique rotation speed, so ships can tell where they are. Lighthouse rotation speeds are allocated so that nearby ones have quite different speeds (in case of timing errors).
- Analogy 2: Stating a date such as Friday, Feb 22 is inherently error-checking, since (for a given year) the day/month/date must be consistent with the calendar. Also can be error-correcting, if we assume the sender meant a nearby month.

30

## Error Detection

- Two main methods: CRC or Cyclic Redundancy Check and Checksum.
- CRC usually done in hardware (Ethernet controller chips), because it operates on one bit at a time.
- Checksum usually done in software (TCP headers), because it operates on one 16-bit word at a time.
- Also newer more powerful “convolutional codes” which have even better error correction performance.

31

## Error Detection – Cyclic Redundancy Check

- Basic idea: a (type of) binary division at each end.
- Works bit-for-bit, so best done in hardware.
- Think of the message data block as a number. Divide by a known number, and you get a remainder.
- Receiver does the same operations. Different remainder signifies an error.
- Used in link protocols (eg, Ethernet)
- *precise details beyond the scope of this course*

32

## Error Detection – Checksum

- Checksum usually done in software (TCP headers), because it operates on one 16-bit word at a time.
- Basic idea: treat words of message as integers, add them up, get a result.
- Receiver does the same calculation. Different, result signifies an error.
- Complication: accumulator (running checksum) will overflow with a large message block, so have to worry about what to do with overflow.
- Used in Internet protocols.
- In practice: see assignment.

33

## Cryptography

Monoalphabetic substitution (key = 2)  
also polyalphabetic substitution

in: a b c d e f g h i j k l ...  
out: y z a b c d e f g h i j ...

Transposition (key = "AKEY": order 1,11,5,25 )  
Read out columns in order...

A	K	E	Y
1	11	5	25
<hr/>			
t	h	i	s
i	s	t	h
e	m	e	s
s	a	g	e

34

## Cryptography

- Using XOR (binary exclusive-OR) function: output bit is 1 if input bits different.
- XOR each message byte (byte group) with a key to give ciphertext.
- Repeat XOR on ciphertext → plaintext.
- Hence it is invertible, if the XOR key is known.
- Multiple XOR stages usually used to increase cryptographic strength. (harder to "crack").
- Also used in combination with re-arrangement (permutation) and substitution tables (example: Data Encryption Standard DES)

35

## Exclusive-OR

Encrypt:

Message byte	1	0	1	0	1	1	0	1
Key	0	1	0	1	1	0	0	0
<hr/>								
Result (=ciphertext)	1	1	1	1	0	1	0	1

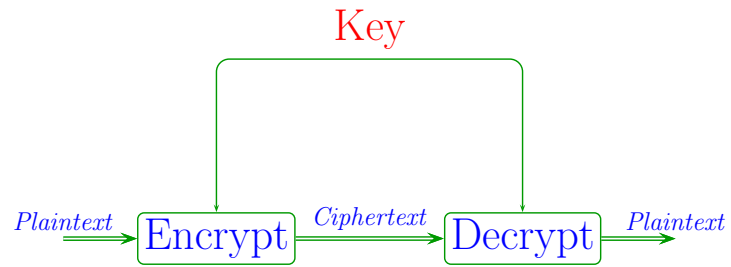
Decrypt:

Input	1	1	1	1	0	1	0	1
Key	0	1	0	1	1	0	0	0
<hr/>								
Result (=original)	1	0	1	0	1	1	0	1

36

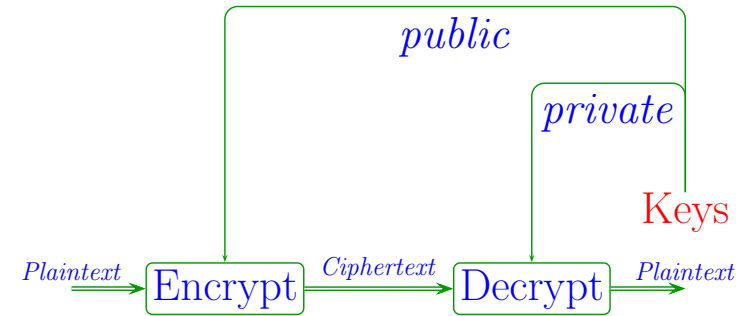
# Security

*Private (secret) key approach*



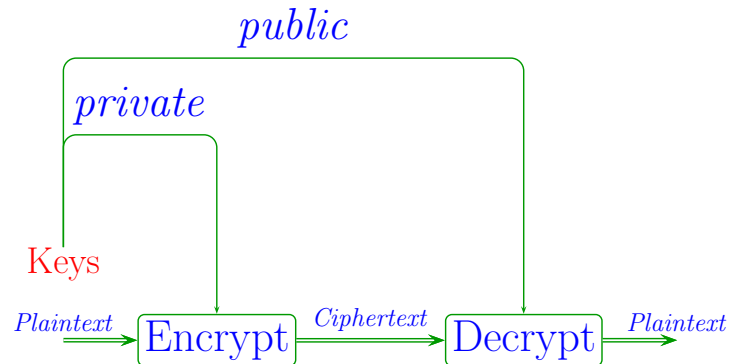
# Security

*Public key approach*



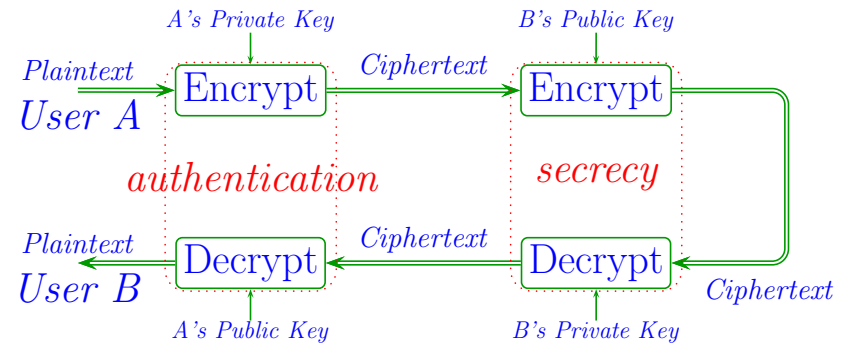
# Security

*Authentication*



# Security

*Authentication + Privacy*



## Public-Key Encryption

- Message  $m$  split into chunks.
- Ciphertext  $c$  calculated in chunks.
- $e$  = encryption key (public),  $d$  = decryption key (secret).  $n$  known to all.
- **Receiver** chooses  $p, q$  and  $e, d$  as random prime numbers such that  $n = pq$  and  $ed \bmod [(p-1)(q-1)] = 1$   
Make  $e$  (&  $n$ ) public. Keep  $d$  secret.
- To encrypt at sender:  $c = m^e \bmod n$
- To decrypt at receiver:  $m' = c^d \bmod n$
- No way to deduce  $d$  from  $e$  and  $\{ m, c \}$

41

## Public Key Summary

1. Define public encryption key  $e$ , private decryption key  $d$
2. Choose large random prime numbers  $p$  and  $q$ ,  $n = pq$ .
3. Choose  $e$  such that  $e$  ( $p-1$ ) ( $q-1$ ) are *relatively prime*.
4. Compute  $d$  such that  $ed \bmod (p-1)(q-1) = 1$ .
5. Encipher:  $c = m^e \bmod n$
6. Decipher:  $m' = c^d \bmod n$

42

## Public Key Implementation

- See following examples.
- Some problems in practice, relating to size of numbers in calculations (get numerical overflow).
- Techniques available to deal with this. But Public Key encryption is still slower than simple algorithms like XOR.

43

## Public Key Algorithm – Example 1

Setup:

1.  $p = 47, q = 79$ .
2. Then  $n = pq = 3713$  and  $(p-1)(q-1) = 3588$
3. Choose  $e$  to be 37, so that  $e$  and 3588 are relatively prime.
4. Compute  $d$  such that  $37d \bmod 3588 = 1$ , ie  $d = 97$ .

Encipher & Decipher:

1. Say the first block of the message is 58.
2. Then  $c = 58^{37} \bmod 3713 = 1671$ .
3. To decrypt, compute  $m' = 1671^{97} \bmod 3713 = 58$

44

## Public Key Algorithm – Example 2

Setup:

1.  $p = 47, q = 71$ .
2. Then  $n = pq = 3337$  and  $(p - 1)(q - 1) = 3220$
3. Choose  $e$  to be 79, so that  $e$  and 3220 are relatively prime.
4. Compute  $d$  such that  $79d \pmod{3220} = 1$ .

Encipher & Decipher:

1. Say the first block of the message is 688.
2. Then  $c = 688^{79} \pmod{3337} = 1570$ .
3. To decrypt,  $1570^{1019} \pmod{3337} = 688$

⇒ Problem is obviously the numerical overflow of these calculations.

45

## Challenge-Handshake Authentication

- CHAP Challenge-Handshake Authentication Protocol — RFC 1994.
- Idea is not to transmit password over insecure link.
- Server poses a “challenge” to the client. Client responds with the challenge hashed using the password. Password never transmitted, either in plaintext or enciphered.
- Used in dial-up serial lines (PPP, Point-to-Point protocol) and HTTP 1.1 authentication (RFC2616).

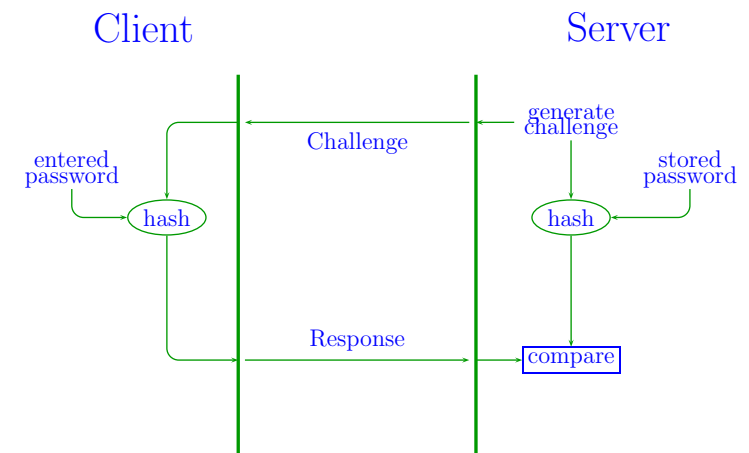
46

## Aside: Hash Function

- To calculate a small (16, 32 bit) number based on the contents of the message.
- Typically combinations of movements of bits, arithmetic operations like squaring and modulo arithmetic (division remainder)
- Aim is to calculate a binary “signature” that could only have come from that particular message. ie, it is unique.

47

## Challenge-Handshake



48

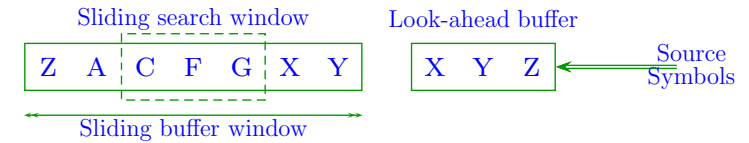
## Challenge-Handshake

- Don't want to send password (plain, or even encrypted).
- Server generates a random "challenge" string.
- Challenge is sent to client.
- Client computes hash of challenge and password. Transmits the hash result.
- Server checks against hash of stored password (probably encrypted itself)
- Actual password is never sent.
- Random challenge is to foil "replay" attacks

49

## Data Compression

- Relies on repetition: eg words of text, run of pixels in an image
- LZ77: Lempel-Ziv Algorithm
- Uses sliding window
- Transmit (offset into window, length of match)



50

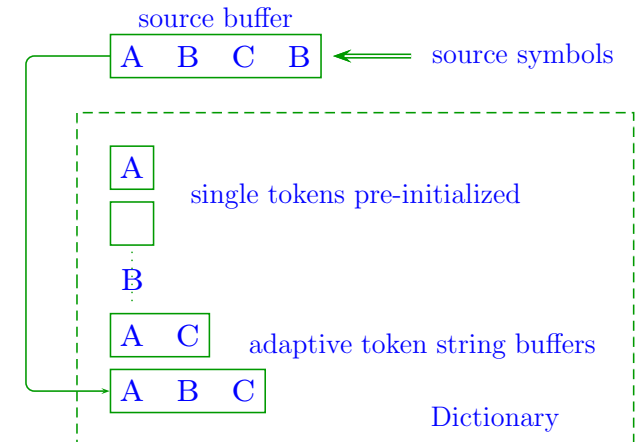
## Data Compression

- Relies on repetition: eg words text, run of pixels in an image
- LZ78: Lempel-Ziv Algorithm
- LZW: Lempel-Ziv-Welch Algorithm (pre-initialized dictionary)
- Uses adaptive dictionary
- Transmit index into dictionary, next character, update dictionary

51

## Data Compression

### LZW Algorithm



52

## Module Summary – Important Points

1. Data transmission – coding
2. Overview of Ethernet (and USB)
3. Understand need for, and basic algorithms used in, encryption, error detection, compression