ECE 435 – Network Engineering
Lecture 5

Vince Weaver

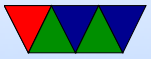
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

14 September 2016

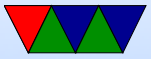
Announcements

- Grades were posted
- Don't forget HW#2



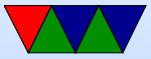
Link Layer

- All about frames.
- Transmitting values to nearby machines: ones/zeros go out to physical layer, same bits arrive back on other machine
- Design issues:
  1. Well defined interface
  2. Dealing with transmission errors
  3. Regulating flow so not overwhelmed by fast senders



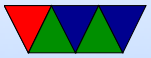
4. Propagation delay

- Packets from network layer are encapsulated into frames for transmission
- Frame has header, payload, and trailer
- Other layers also encapsulate in frames, but this is lowest level so we will talk about it here



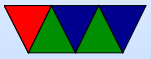
Link Layer – Issues

- Framing – split data into frames
- Addressing – specify destination
- Error control and reliability
- Flow Control – stop from sending too fast
- Medium Access Control – method to decide which host gets to transmit



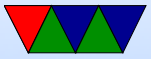
Frame Format

- Header – address? length? type?
- Data
- Tail – error detection?



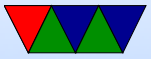
Address

- Global or local? Only few extra bits of extra overhead so often global these days (MAC address?) IEEE 802 is 48-bits. Is that enough?



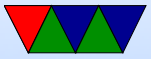
Framing

- Break up data stream into frames, checksum each on send and receive
- How do you break up into frames?
  1. Character count – send a byte describing how many chars follow, followed by that many chars
     Trouble is, what if count affected by noise. Then the data gets out of sync, no way to resync
  2. Flag bytes – special byte indicates start and stop, you can then use to find frame boundaries



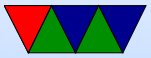
What to do if flag byte appears in data you are sending? Use escape chars (sometimes called "byte stuffing"?)

3. Bitstuffing – instead of sending multiples of 8 bits, send arbitrary bit widths, with special bit patterns as flags
4. Physical layer coding – use some of the ones we discussed, where you can 4B/5B or such where you can use the unused values as frame markers



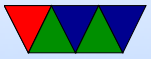
Flow Control

- What if sender tries to send faster than receiver can handle?
- Feedback based: receiver sends back info saying it is ready for more (serial with HW flow control)
- Rate-based flow control. The rate is set in the protocol. Not really used in the link layer

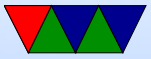


Error Control

- What can you do? Get an acknowledgement saying was correct
- What if something happens and the entire frame lost? Receiver never gets it one way or another. Sender waits forever?
- Use a timer. If no response send again
- What happens if you send multiple times and then

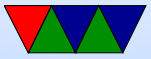


eventually both get there? Often have a sequence number to track if there are multiple.



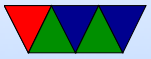
Error Detection/Correction

- Are errors a problem? If sending 1000 bit frames, and error rate is .001 per bit, then if even distributed on average each frame have an error. Are errors evenly distributed? What if 1000 in a row then none? (bursty)
- Error-Detection Codes – let you tell if an error happened what to do if error happens? resend. doable if errors infrequent (reliable connection)
- Error-Correcting Codes – let you fix an error

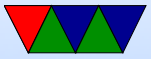


Hamming Distance

- Number of bits that differ
- Can calc by exclusive oring then counting the ones.
- 0101 1101 = 1000 = 1
- If hamming distance of N then takes N single-bit errors to convert between the two
- To detect N errors you need hamming distance of N+1 to ensure than N errors can create another valid code

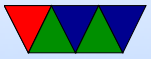


- To correct N errors you need 2N+1 distance, that way even with N errors it is still closer to changed value than any other
- parity bit. Chosen so code word is always even (or odd) can detect single bit error
- Hamming code for detecting errors



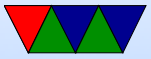
Error Detecting Codes

- One way: arrange bits in rectangle, take parity bits across both rows and columns
- Polynomial codes: CRC (cyclic redundancy check)



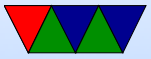
CRC check

- Polynomial, 110001 means $x^5 + x^4 + x^0$
- Agree on generator in advance. High and low bits must be 1. Checksum is calculated. Value to check must be longer than generator
- Append checksum on end, and when run through the result is zero. Any remainder means an error
- IEEE 802 uses $x^{32} + x^{26} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$



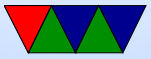
which can detect any burst error less than 32 and all odd number bits

- might seem hard, but easy to make in hardware with a shift register and some xor gates.
- CRC can find single bit errors, double bit errors, bursts of errors less than length of polynomial.
- Explaining how it works is "mathematically complex" says open source approach book

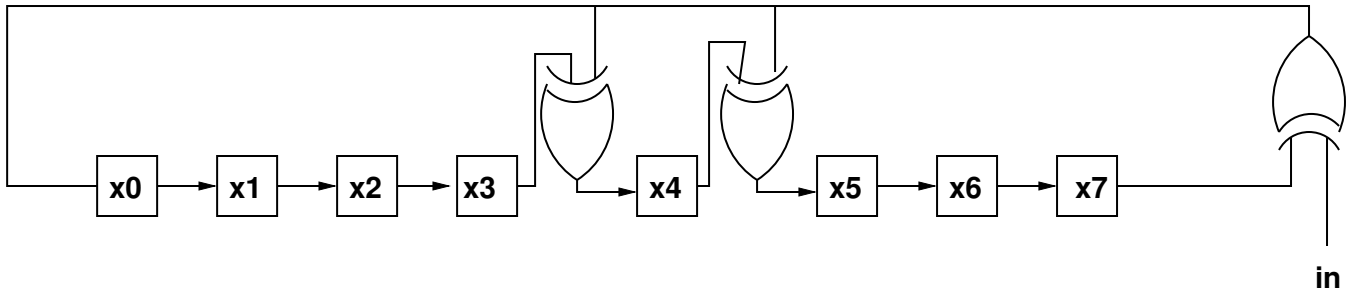


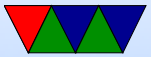
1-wire CRC check

- Usually used in hardware, harder to implement in software
- Can detect all double-bit errors, any double bit errors, any cluster within an 8-bit window
- if CRCs with itself gets 0 at the end, how hardware detects correct address.
- $X^8 + X^5 + X^4 + X^1$



- Fill with zero, shift values in.

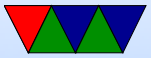




Example Data Link Protocols – Setup

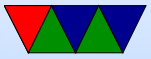
- Data Link layer accepts packet
- Encapsulates in a frame by putting on a header, the data, trailer (checksum)
- Frame info is never passed up to the higher level. If it did, could never make changes at the link layer.
- Frame header:

1. kind – what type (Control? data?)



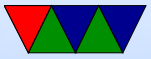
2. seq
3. ack

- Assume some time frames get lost. So when transmitting start a timer, and if no ack in enough time resend it.



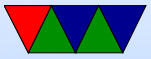
Example – Simplex

- No sequence, no ack, one way, assume error free
- sender: Infinite loop. Fetch packet, make a frame, send it
- receiver: grabs frame, removes header, passes to network layer

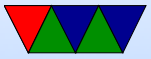


Example – Simplex/Stop&Wait

- How to prevent sender from sending faster than you can receive?
- Can you just add delay? Is how long the receiver takes deterministic?
- If you calculate worst case, do you design to always go that slow?
- Solution: provide feedback

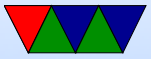


- After receive frame, send a short "ACK" acknowledge frame
- Stop and wait sends frame and waits for reply before sending next
- two-way communication, but both ends not sending at once so still can be half-duplex



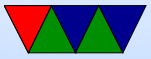
Example – Simplex/Stop&Wait with Errors

- Check checksum. Just not bother to send ACK and let it time out. Would that work?
- What happens if an ACK gets lost? Not just transmit frames get lost.
- You end up with duplicate data
- What is minimal amount of extra data you need?
- In this case, need to distinguish M from M+1. If M is



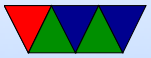
successful, it ACKS, and needs to make sure the next one it gets is the followup M+1

- In this simple case can get away with just one bit
- This sometimes called PAR (Positive Ack with retransmit) or ARQ (Automatic Repeat Request)
- Transmits packet, starts timer. Three possibilities ACK comes in (good), damaged ACK comes or no ACK (timer expires) then change sequence number and resend.

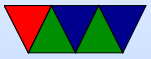


Example – Sliding Protocol Window

- Can we intermix control and data on same wire? Full duplex?
- Include bit to indicate if it's control or data frame
- Can do piggybacking, send ACK along with next data packet (a bit in the header)
- Less header overhead, less frame overhead, less overhead of waking up the system to handle extra packets



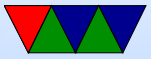
- How long should you wait for a frame to piggyback before giving up and sending a standalone ACK?
  What happens if you wait too long?
  Other side times out, resends, wasting your optimization.
- Sliding Window. Sending Window and Receiving Window. Sender has a window of frames permitted to send, receiver has window of frames can receive.
- The "sliding window": the sequence numbers in window are frames sent but have not been ACKed.
- Sender: new incoming packet assigned next free seq



number. ACKs come in the lower limit of window "slides" up.

- Needs to hold in memory a buffer of all unacked frames so can re-send if necessary. If window ever maxes out, it has to stop sending until some ACKs come in.

- Receiving window: when receive a frame any outside window is ignored. If it's the lowest edge of window, it is passed up to the network layer, an ACK is generated, and window rotated. Receiver window never grows in size



- If window size 1, then all packets received in order. Can be larger than 1, then can receive packets out-of-order
- Despite getting frames out of order, network layer only ever sees them in order