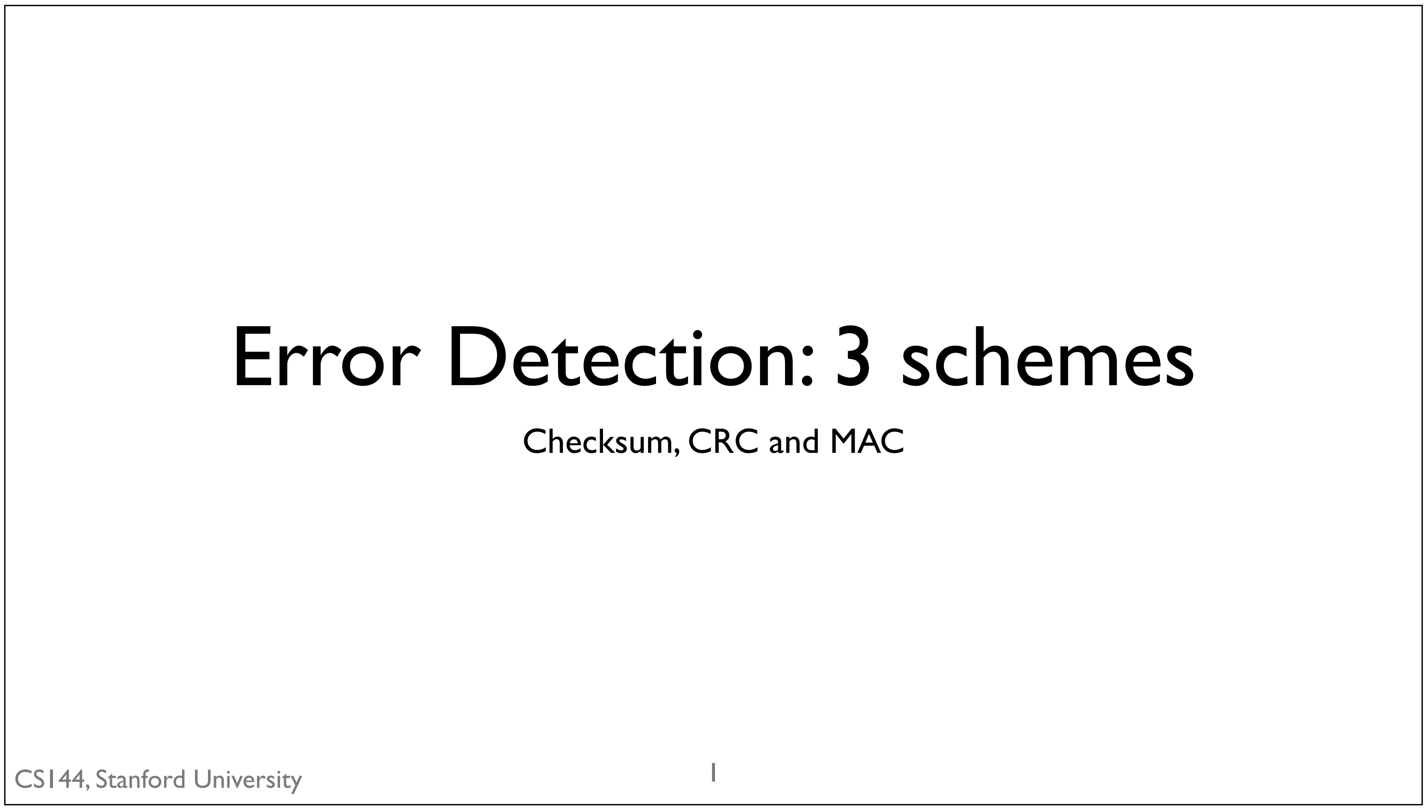
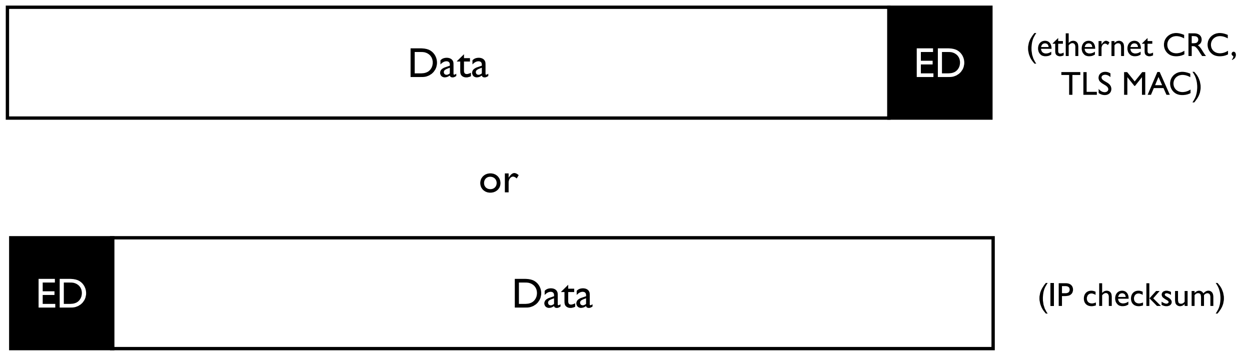


Error Detection: 3 schemes

Checksum, CRC and MAC

Networks aren't perfect, and neither are the hosts that run on them. They can introduce errors, and for a network to be able to run properly it needs to be able to detect these errors. For example, let's say that a router along our path has a bad memory cell, such that sometimes flips a bit in a packet. Imagine, for example, if the bit flipped is the most significant bit of the amount to charge a credit card. We need to be able to detect that error occurred so we don't accept the corrupted data as correct data. Networks today generally use three different error detection algorithms: checksums, cyclic redundancy codes, CRCs, and message authentication codes, MACs. Each of them has very different characteristics. Understanding their differences is important! I've actually been at meetings in the IETF where a few people weren't aware of the differences! If you don't know, you might make a bad protocol decision or analysis.

Error Detection



At a high level, error detection looks like this. We have a payload of data. We calculate some error detection bits over that data and either append it or prepend it to the payload. For example, Ethernet appends a cyclic redundancy code, Transport Layer Security appends a message authentication code, and IP prepends a checksum, which it places in the IP header. TLS and Ethernet have a footer, protocol information which follows the payload, which is where they put the CRC and MAC.

Three Error Detection Schemes

- Checksum adds up values in packet (IP, TCP)
  - Very fast, cheap to compute even in software
  - Not very robust
- Cyclic redundancy code computes remainder of a polynomial (Ethernet)
  - More expensive than checksum (easy today, easy in hardware)
  - Protects against any 2 bit error, any burst $\leq c$ bits long, any odd number of errors
- Message authentication code: cryptographic transformation of data (TLS)
  - Robust to malicious modifications, but not errors
  - If strong, any 2 messages have a 2^{-c} chance of having the same code

The first of the three commonly used error detection algorithms is a checksum. You just add all of the data in the packet. It's what TCP and IP use. Checksums are nice because they are very fast and cheap to compute, even in software. Back when the Internet started and everything was in software, this was valuable. Their major drawback is that they have pretty weak error detection guarantees. While they can catch a lot of random errors, it's easy to fool a checksum with as few as 2 bit errors, if the two bit errors cancel each other out. For example, if one bit error adds 32 and another bit error subtracts 32, then a checksum won't catch the error. So a checksum can catch a lot of errors, but it turns out to have very weak guarantees on what errors it will catch.

The second of the three commonly used error detection algorithms is a cyclic redundancy code, or CRC. A CRC is much more computationally expensive than a checksum, but also much more robust. It computes the remainder of a polynomial -- I'll show what this means and how it works in a few minutes. With today's processors, it's easy to do, and it's easy to do in hardware. It's what Ethernet and many link layers use. In some ways, TCP and IP can get away with checksums because the link layers use CRCs. If you have a CRC that's c bits long, a CRC can detect any 1 bit error, any 2 bit error, and single burst of errors less than or equal to c bits long, and any odd number of errors. So it can provide much stronger guarantees that a checksum.

The final algorithm is something called a message authentication code, or MAC. A message authentication code combines the packet with some secret information to generate a value. In theory, someone can only generate or check the MAC if they have the secret. So if you receive a packet and its MAC is correct, then you're pretty sure the computer that computed the MAC has the secret. Unless I have the secret, it's amazingly difficult to generate the correct MAC for a packet. So a bad guy can't easily generate a new packet. In fact, if you have a strong MAC algorithm, then given one packet and its MAC, I have zero information on what the MAC will look like if I flip a single bit. Message authentication codes are therefore robust to malicious modifications. Message authentication codes are used in Transport Layer Security, TLS, which is what you use when you browse web pages securely -- https.

But they're actually not great for catching errors. If I flip a single bit in a packet, theres a 1 in 2 to the c chance that the changed packet will have the same MAC! I've seen people make this mistake when talking about error correction, thinking a MAC is just as good as a CRC. It's not! If I have a 16-bit CRC, I'm assured that I will detect a burst of errors that is 16 bits long or shorter. If I have a 16-bit MAC, I'm only assured that I'll detect bit errors with very high probability, 99.998 percent, or one in 65,536. That's high, but think about how many packets you've received just watching this video...

I'll now go into each of these algorithms in greater detail.

IP Checksum

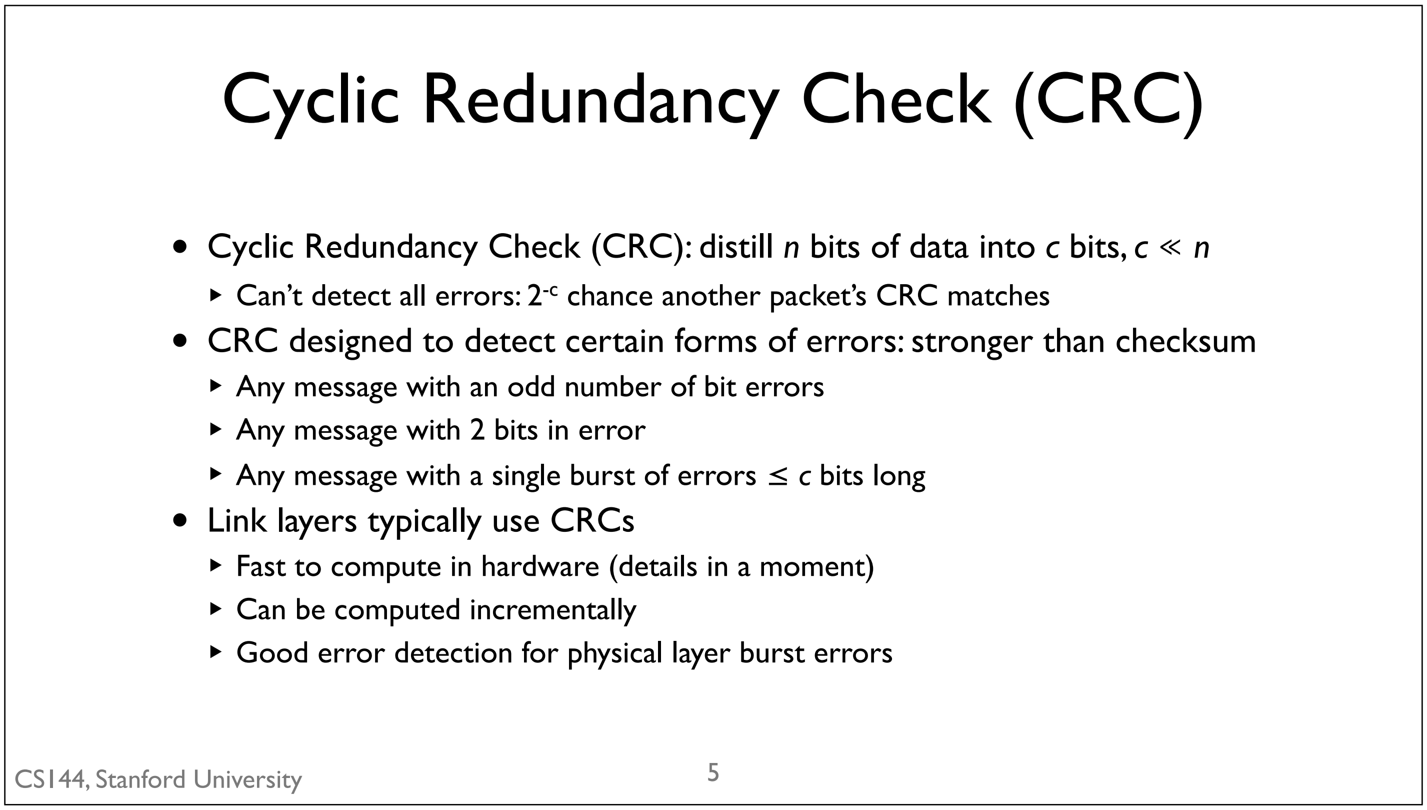
- IP, UDP, and TCP use one's complement checksum algorithm:
  - Set checksum field to 0, sum all 16-bit words in packet
  - Add any carry bits back in: 0x8000 + 0x8000 = 0x0001
  - Flip bits (0xc379 becomes 0x3c86), unless 0xffff, then checksum is 0xffff
  - To check: sum whole packet, including checksum, should be 0xffff
- Benefits: fast, easy to compute and check
  - Motivated by earliest software implementations
- Drawbacks: poor error detection
  - Only guarantees detecting a single bit error
  - Can detect other errors, but actual guarantees are both weak and complex

Let's start with a checksum. IP, UDP, and TCP use one's complement checksums. This means they add up the packet using one's complement arithmetic, a version of binary arithmetic some older computers used. Most today use two's complement.

The algorithm is pretty simple. You start by setting the checksum field of the packet to zero. Then you add every 16-bit word in the packet. Any time you have to carry, because the sum is greater than 65,535, then you carry the bit back in. So 60,000 plus 8,000 is 68,000 - 65,535 + 1, or 2,466. Once you've added up the complete packet, flip the bits in your sum and make this the checksum of the packet. Then, if you add up the complete packet, including the checksum value, you should get 0xffff. There's one edge case: if the computed checksum is 0xffff, you don't make the checksum field 0, you make it 0xffff. In IP, UDP, and TCP, a checksum field of 0 means there's no checksum.

That's it! You can write this in just a few lines of C code. It's fast, easy to compute, and easy to check. All you need to do is add the bytes of a packet. Given that most early Internet implementations were in software, this was really helpful.

The drawback is that it's really not that robust. While it definitely detects lots of random errors, the guarantees it can give on what errors it detects are really weak. In practice, it can only promise to catch single bit errors. But it works pretty well, and link layers do a lot of heavy lifting for us.


Cyclic Redundancy Check (CRC)

- Cyclic Redundancy Check (CRC): distill *n* bits of data into *c* bits, $c \ll n$
  - Can't detect all errors: 2^{-c} chance another packet's CRC matches
- CRC designed to detect certain forms of errors: stronger than checksum
  - Any message with an odd number of bit errors
  - Any message with 2 bits in error
  - Any message with a single burst of errors $\leq c$ bits long
- Link layers typically use CRCs
  - Fast to compute in hardware (details in a moment)
  - Can be computed incrementally
  - Good error detection for physical layer burst errors

Link layers do their heavy lifting with something called a cyclic redundancy check, or CRC. The idea of a CRC is that I want to take n bits of source data and somehow distill them down into c bits of error detection data, where c is much much smaller than N. For example, I might have a 1500 byte Ethernet frame with a 4 byte, 32 bit, CRC. USB and Bluetooth use 16-bit CRCs.

Of course we can't detect all errors. Given some other random packet, the chances the CRC matches is two to the minus C, or one in two to the c. For example, if I use an 8 bit CRC, then out of the space of all packets, one in two hundred and 256, or 0.4%, have the same CRC as any given packet.

But CRCs are stronger than checksums. They can detect there's an error in any packet with an odd number of errors, 2 bit errors, or any single burst of errors less than or equal to c bits long. They can't guarantee detecting errors besides these, but do a good job at it. For example, a 16-bit CRC can't guarantee it will detect two bursts of 3 bit errors spaced far apart in a packet, but it's likely it will detect the error.

Link layers typically use CRCs. They're pretty robust, and as many link layers are vulnerable to bursts of errors, the burst detection capability of CRCs is useful. It's not hard to make compute them quickly in hardware, and you can compute them incrementally, as you read or write the packet.

Diversion: CRC Mathematical Basis

- Cyclic Redundancy Check (CRC): distill *n* bits of data into *c* bits, $c << n$
- Uses polynomial long division
  - Consider the message M a polynomial with coefficients 0 or 1 (pad with *c* zeroes)
    - E.g., M = 10011101 = $x^7 + x^4 + x^3 + x^2 + 1$
  - Use a generator polynomial G of degree *c* also with coefficients 0 or 1
    - Pad first term (always 1) for frustrating historical reasons
    - E.g. G = 1011 = $x^4 + x^3 + x + 1$
    - USB (CRC-16) = 0x8005 = $x^{16} + x^{15} + x^2 + 1$
  - Divide M by G, the remainder is the CRC: pick G carefully!
- Append CRC to message M: M' = M + CRC
  - Long division of M' with G has a remainder of 0

How does a CRC work? It distills these n bits into c bits using something called polynomial long division. You take the bits of a message and use them to describe a polynomial M. Each bit in a packet is the coefficient of one term of the polynomial. If the bit is zero, the term is absent. If the bit is one, the term is present.

So, for example, a message of 10011101 is the polynomial x to the 7th plus x the fourth plus x to the third plus x squared plus one, x to the zero. This is because the 7th, 4th, 3rd, 2nd and 0th bits are set in the message.

When we calculate a CRC, we have something called a generator polynomial. This is defined by the CRC algorithm. For example, the CRC-16 algorithm used by USB has a generator polynomial of x to the 16th plus x to the 15th plus x squared plus one. For frustrating historical reasons, the generator polynomial is one term longer than its number of bits: the first term is always one. So the CRC-16 generator polynomial is written as 0x8005 even though it has an x to the 16th term.

To compute a CRC, you take the message M, pad it with zeroes equal to the CRC length and divide this padded value by G. The remainder is the CRC, which you append to the message. To check a CRC, you divide the message plus CRC by the generator polynomial G. If the remainder is zero, then the CRC passes. I won't go into the details of how this works mathematically, but it turns out it can be implemented very quickly and efficiently in hardware. The strength of your CRC algorithm depends on what generator polynomial G you pick: there's been a lot of study of this and so many good options which have the error detection properties I mentioned earlier. But you might not get the same error detection strength if you pick your own generator polynomial.

MAC

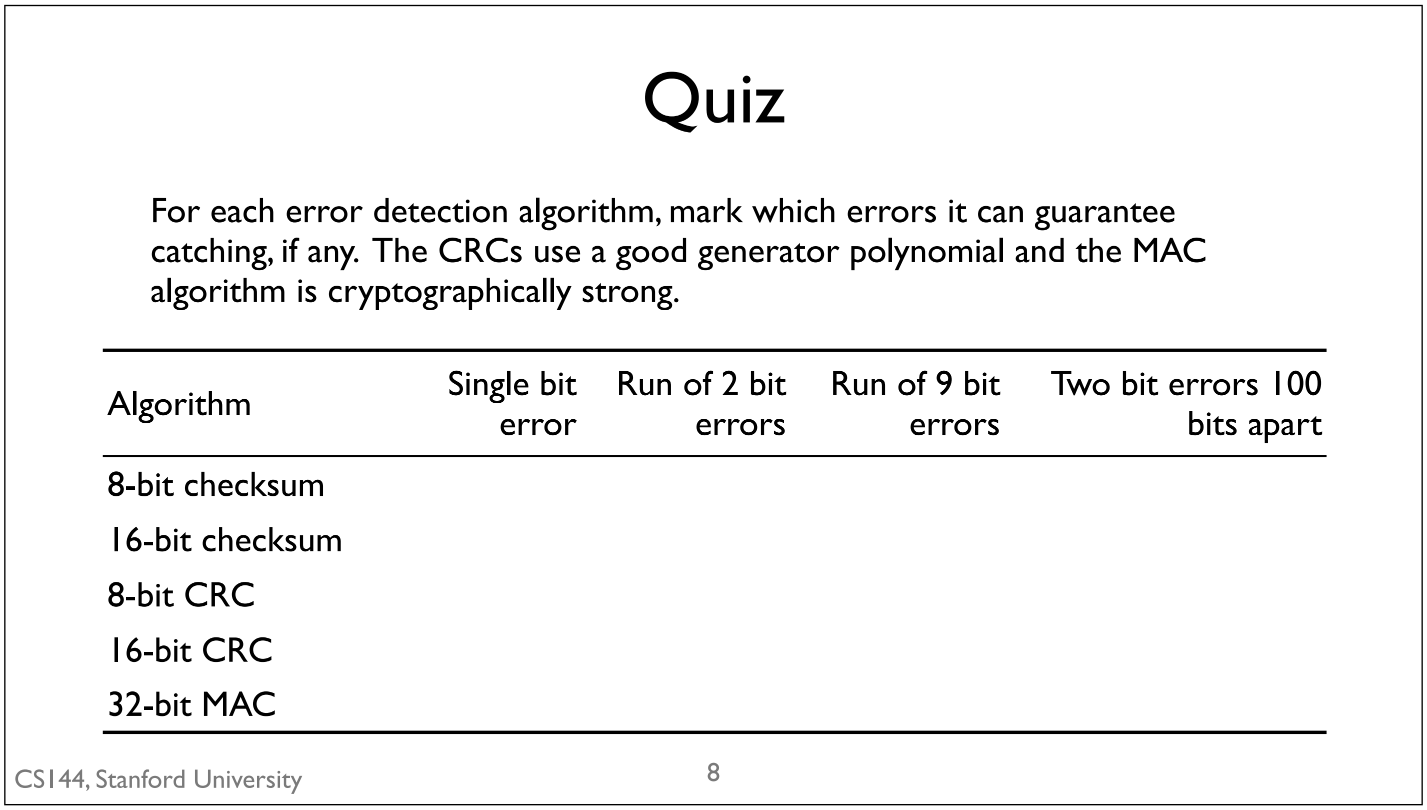
- Message Authentication Code (MAC)
  - Not to be confused with Media Access Control (MAC)!
- Uses cryptography to generate c = MAC(M, s), $|c| \ll |M|$
  - Using M and secret s, can verify c = MAC(M, s)
  - If you don't have s, very very hard to generate c
  - Very very hard to generate an M whose MAC is c
  - M + c means the other person probably has the secret (or they're replayed!)
- Cryptographically strong MAC means flipping one bit of M causes every bit in the new c to be randomly 1 or 0 (no information)
  - Not as good for error detection as a CRC!
  - But protects against adversaries

The third and final kind of error detection algorithm you commonly see in networks is a message authentication code, or MAC. Like CRCs, there's a deep and rich mathematical background on how message authentication codes work: there are good ones and bad ones. So you generally want to use an existing scheme rather than invent your own. Thankfully standards usually specify what MAC to use, and although there were some mistakes in the late 90s where standards picked poor algorithms, nowadays security is important enough that everyone relies on a small number of really well studied approaches.

Message authentication codes use cryptography, a branch of mathematics that deals with secrets. The idea behind most message authentication codes is that the two parties share a secret s. This secret is just a set of randomly generated bits (random so it's hard to guess). To calculate a message authentication code c, apply the MAC algorithm to the message M and the secret s. MAC algorithms have the property that if you don't have s, then it's really hard to generate the correct c for a message M. Furthermore, it's very hard to create a message M whose message authentication code is c. By "hard" I mean that in the best case you'd just have to exhaustively try: having M and c gives you almost no information on what s is. This means that if you receive a message M with the correct message authentication code, this means the computer that generated the message probably has the secret (or someone replayed a message generated by that computer).

Because the goal is to keep s a secret, cryptographically strong message authentication codes have an interesting property. If you change a single bit in M, then this results in a completely new c, where the probability any bit in c is zero or one is seemingly random. If this weren't the case, then someone could potentially take a message, flip a single bit (e.g., change a dollar value) and it wouldn't be that difficult to generate the correct c. This means that message authentication codes actually have no error detection guarantees. If you flip a single bit, you could end up with the same MAC!

Message authentication codes are very useful, but they're first and foremost a security mechanism. Being able to get both error detection and security with one mechanism is efficient and nice, but their security properties mean their error detection isn't as good as other approaches.



Quiz

For each error detection algorithm, mark which errors it can guarantee catching, if any. The CRCs use a good generator polynomial and the MAC algorithm is cryptographically strong.

Algorithm	Single bit error	Run of 2 bit errors	Run of 9 bit errors	Two bit errors 100 bits apart
8-bit checksum				
16-bit checksum				
8-bit CRC				
16-bit CRC				
32-bit MAC				

Here’s a quiz. For each error detection algorithm, mark which errors it can guarantee catching, if any. The CRCs use a good generator polynomial and the MAC algorithm is cryptographically strong.

Quiz

For each error detection algorithm, mark which errors it can guarantee catching, if any. The CRCs use a good generator polynomial and the MAC algorithm is cryptographically strong.

Algorithm	Single bit error	Run of 2 bit errors	Run of 9 bit errors	Two bit errors 100 bits apart
8-bit checksum				
16-bit checksum				
8-bit CRC				
16-bit CRC				
32-bit MAC				

Let's go over the answers.

Both checksums can detect a single bit error -- remember that this is one of the errors a checksum guarantees detecting.

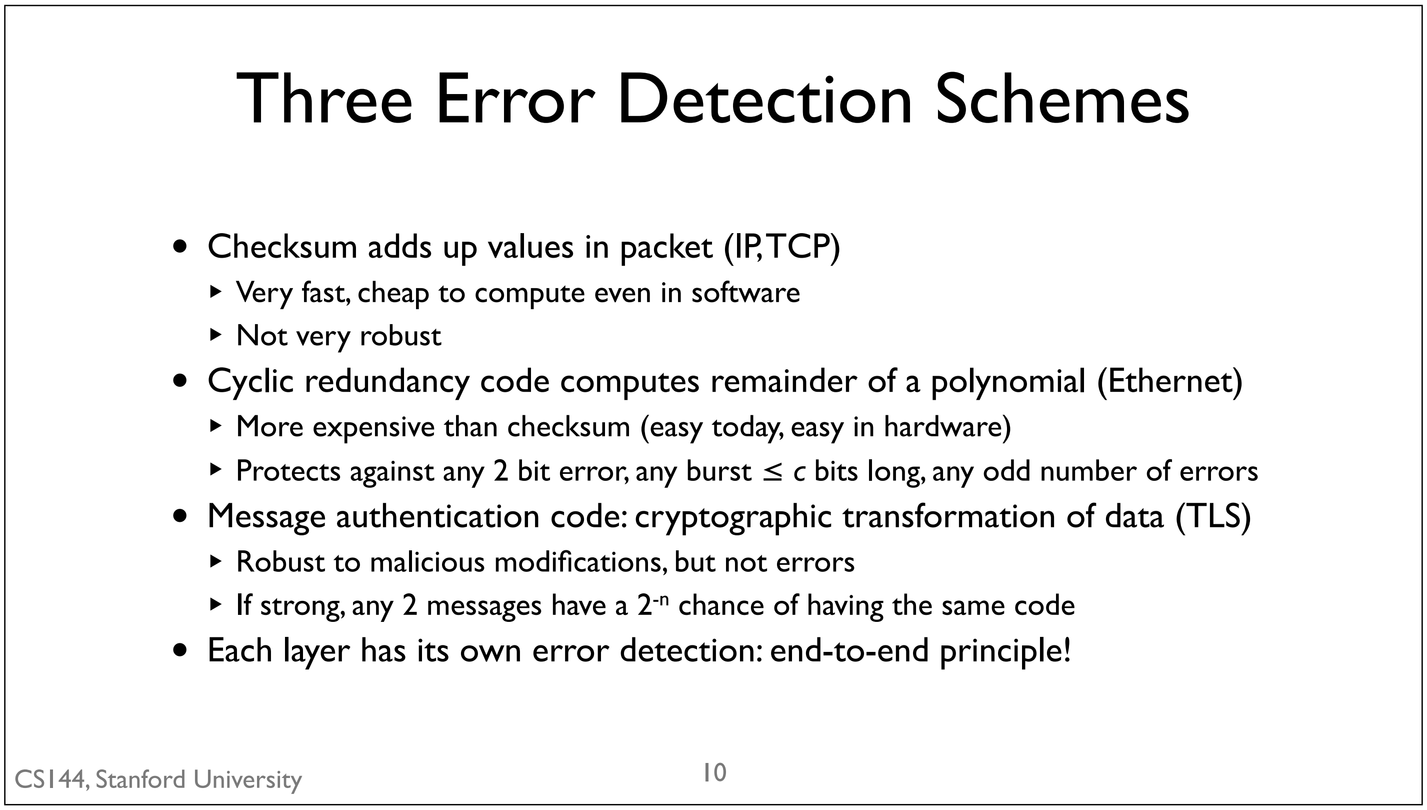
Both CRCs can also detect a single bit error.

A MAC can't guarantee that it will detect a single bit error. For security reasons, could be that the new MAC is the same as the old one. So it can't guarantee detecting it. In fact, a MAC can't guarantee detecting any errors! So we can mark No for all of the columns for the message authentication code.

How about two bit errors? Checksums can't guarantee detecting two bit errors. So N for both of them. CRCs, though, can guarantee detecting bit error runs in length less than or equal to the length of the CRC. Since 2 bits is shorter than both 8 bits and 16 bits, both CRCs can detect a run of 2 bit errors. Correspondingly, an 8-bit CRC can't guarantee detecting a run of 9 bit errors, but a 16-bit CRC can. So N for the 8 bit CRC and Y for the 16-bit CRC.

How about two bit errors 100 bits apart? It turns out none of these algorithms can guarantee detecting this error. So N for all of them.

Looking at this matrix, you might think error detection is a waste -- the algorithms promise very little! But guarantee is a very strong statement. While a 16-bit checksum can't guarantee that it will catch a run of 9 bit errors, there's a high probability it will. Similarly, a 16-bit CRC has a very high probability of detecting two bit errors 100 bits apart. And in practice, high probability is often good enough. If failures are rare, then you only sometimes have to do something more expensive to recover. But it means in practice you tend to have multiple layers of error detection: the link layer detects them with CRCs, IP detects them checksums, TCP detects them with checksums, and then often the application has its own error detection. So, all put together, the changes of errors creeping through is very very low.



Three Error Detection Schemes

- Checksum adds up values in packet (IP, TCP)
  - Very fast, cheap to compute even in software
  - Not very robust
- Cyclic redundancy code computes remainder of a polynomial (Ethernet)
  - More expensive than checksum (easy today, easy in hardware)
  - Protects against any 2 bit error, any burst ≤ *c* bits long, any odd number of errors
- Message authentication code: cryptographic transformation of data (TLS)
  - Robust to malicious modifications, but not errors
  - If strong, any 2 messages have a 2^{-n} chance of having the same code
- Each layer has its own error detection: end-to-end principle!

So we've seen three error detection schemes: checksums, CRCs, and message authentication codes. Checksums are fast and cheap, but not very robust. They're used in IP and TCP. Cyclic redundancy codes are much stronger. Computing them in hardware is easy today, and they can detect a wide range of errors. Link layers, such as Ethernet, typically use CRCs. The third algorithm is a message authentication code, or MAC. Message authentication codes are designed for security, but you can also use them for error detection. However, since they're designed for security first, this means they're really good against malicious modifications, but not that greater for detecting errors.

Data error detection is a great example of the end-to-end principle. It's actually what originally motivated the principle -- the only way a layer can be sure that it communicates data correctly is to perform an end-to-end check. Ethernet needs to be sure that its frames don't have errors so it can parse them correctly, so it has a CRC. IP needs to be sure that its packets don't have errors so it can parse them correctly. IP can't depend on Ethernet doing its checking for it: the Ethernet card or driver might introduce an error after it checks the packet. So it has to do its own end-to-end check at the network layer. TLS using message authentication codes is another example. It's especially interesting because TLS has very different error detection requirements than IP or Ethernet: it wants security. So it has to provide its own, end-to-end error detection scheme, as that's the only way it can be sure its requirements are met.