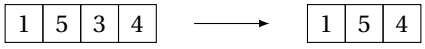
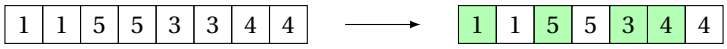
1 Erasure Errors

Suppose we have a message $(1,5,3,4)$ that we want to send, but one of those packets could be erased—only 3 packets would make it through.



If we just send the message as is, we'd have permanently lost one packet, and we won't be able to know what the original message was.

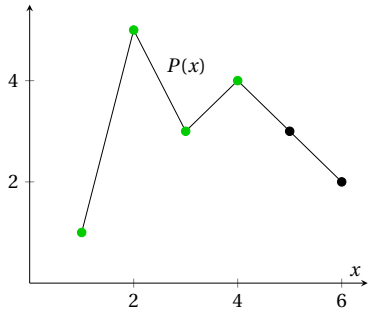
The naive solution to this problem would just be to send each packet twice:



No matter which packet gets corrupted, we'd be able to recover our original message. However, this is a really inefficient solution; with a message of length n and with k erasure errors, we'd need a total of nk packets in order to ensure we can recover the original message. This gets *really* big when n and k get larger.

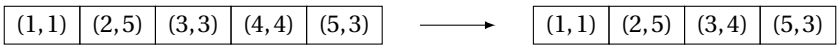
We can do better if we use a *polynomial encoding* of our message. Recall that a degree d polynomial is uniquely defined with $d+1$ points.

In this new scheme, we send points on a polynomial rather than the raw message contents. This means that we have a polynomial of degree $n-1$, uniquely defined by our original n points, and we send points along this polynomial $P(x)$. A typical encoding is with the message contents along $x = 1,2,\ldots,n$, working in a Galois field $GF(p)$ (i.e. in mod p) to make calculations easier.



Suppose k points get erased. Since n points uniquely define our polynomial $P(x)$ of degree $n-1$, our receiver needs to get n points in order to recover our original message. This means that we need to send $n+k$ points on the polynomial; k are lost, and we're left with n points, which will uniquely define $P(x)$.

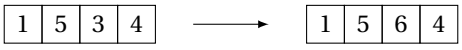
In our original scenario, we'd need to send $n+k = 4+1 = 5$ points:



We can now use Lagrange interpolation with these received points to reconstruct our original polynomial. Evaluating $P(x)$ at $x = 1$ through 4 gives us our original message.

2 General Errors

Suppose instead that when we send a message $(1,5,3,4)$, one packet is *modified* in transmission:



Sending the message directly, we won't be able to tell which of these packets are actually correct—because of this, we can't recover the original message.

In order to get around this, we can use a similar polynomial encoding as before. If we want to send a message of length n, with k possible general errors, we need to send $n+2k$ coordinates in order to recover the original polynomial and message. (Why? See section 4.)

Here, since we have a message of length 4, and 1 general error, we need to send $4+2(1)=6$ packets.

Note that we don't need to send the x-values here; we'd be able to recover the entire sent string without it. Realistically, we'd need to know which packet indices contain the actual message, though.

Now, how can we recover the original message? This is what the Berlekamp–Welch algorithm does.

3 Berlekamp–Welch

Let's define $E(x) = (x-e_1)(x-e_2)\cdots(x-e_k)$ to be a polynomial with roots at the locations of the errors. For example, if we ended up with errors at $x=1$ and $x=3$, we'd have $E(x)=(x-1)(x-3)$. This polynomial is also called the *error locator polynomial*.

Notice that if we take the product $P(x)E(x)$, we have the following equations (where r_i are the received packets):

$$P(i)E(i) = r_iE(i) \quad \text{for } 1 \le i \le n+2k.$$

The error locator polynomial is defined to be 0 wherever the errors are, so any errors would be erased when multiplying by $E(x)$; all the correct values are left intact, but multiplied by some scalar $E(i)$.

Suppose we define $Q(x) = P(x)E(x)$ to be another unknown polynomial. If we evaluate that equation at each of the $n+2k$ points, we can set up a system of linear equations $Q(i) = r_iE(i)$ to solve (in a given Galois field $GF(p)$).

What are we solving for, exactly? The unknowns are from the coefficients of $Q(x)$ and the roots of $E(x)$. Since $P(x)$ is of degree $n-1$ and $E(x)$ is of degree k, $Q(x)$ must be of degree $n+k-1$, with $n+k$ coefficients including the constant.

This means we have a total of $n+2k$ unknowns—with the $n+2k$ equations from evaluating $Q(i) = r_iE(i)$ for each packet, we are guaranteed a solution to the system.

Once we find a solution to this system, we can construct $Q(x)$ and $E(x)$, and get $P(x)$ by simply dividing the two: $P(x) = \frac{Q(x)}{E(x)}$. Evaluating $P(x)$ at $1 \le i \le n$ gives us the original message.

One tip for this last evaluation is that we don't actually need to evaluate all n points—we know where the errors are, and every other value must be correct. This means we only need to re-evaluate $P(x)$ at the locations of errors, saving some work.

In summary,

Polynomial	Expression	Degree	Unknowns
$P(x)$	$\frac{Q(x)}{E(x)}$	$n-1$	—
$E(x)$	$(x-e_1)(x-e_2)\cdots(x-e_k)$	k	k
$Q(x)$	$a_0+a_1x+a_2x^2+\cdots+a_{n+k-1}x^{n+k-1}$	$n+k-1$	$n+k$

4 Why $n+2k$?

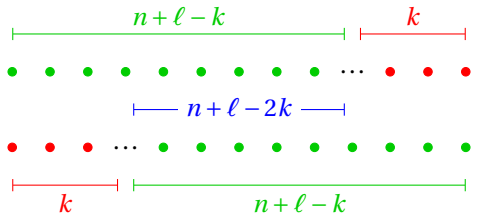
One possible way to see why $n+2k$ packets are needed is to look at the Berlekamp–Welch algorithm. We'll always have $n+2k$ unknowns—no matter how many packets we send, $P(x)$ is always of degree $n-1$ and there are always k general errors, so $Q(x)$ will always be of degree $n+k-1$ and have $n+k$ unknown coefficients. However, the number of packets we send affects the number of equations we have.

If we have any less than $n+2k$ equations, we'd have less equations than unknowns, and will be unable to solve the system of equations. But this raises another question—what if we just need a different algorithm? Can there exist an algorithm that *is* able to recover the original polynomial and message?

Suppose we have a message of length n, and we send $n+\ell$ packets. The only scenario in which we won't be able to recover the original polynomial is if we can find two distinct polynomials of degree $n-1$ that go through $n+\ell-k$ of the points (i.e. all of the presumably uncorrupted points).

Suppose we picked two subsets containing $n+\ell-k$ points, and were able to construct the polynomials $A(x)$ and $B(x)$ of degree $n-1$ from these subsets of points using Lagrange interpolation.

Without loss of generality, suppose we chose the first $n+\ell-k$ and the last $n+\ell-k$ points. Let's look at the common $n+\ell-2k$ points between these two subsets.



If we have $n+\ell-2k \geq n$, then these overlapping points would uniquely define a single polynomial of degree $n-1$ by themselves. Because of this, we must have $A(x)=B(x)$ because they share enough points to define a degree $n-1$ polynomial. This means that *all* possible choices of $n+\ell-2k$ points will recover the exact same polynomial, if Lagrange interpolation succeeds in recovering a degree $n-1$ polynomial.

If we instead have $n+\ell-2k<n$, then we have some problems: these overlapping points aren't enough to uniquely define a single polynomial. This means that if some of the points in our subsets are corrupted, we could end up with two distinct polynomials $A(x)\neq B(x)$. Since both of these polynomials go through $n+\ell-k$ points, we don't know which one is the original $P(x)$—we're unable to recover the original message.

Putting this all together, we are only able to recover the original message if $n+\ell-2k\geq n$, i.e. if $\ell\geq 2k$. This means that we must send at least $n+2k$ packets in order to guarantee that we can uniquely recover the original message.