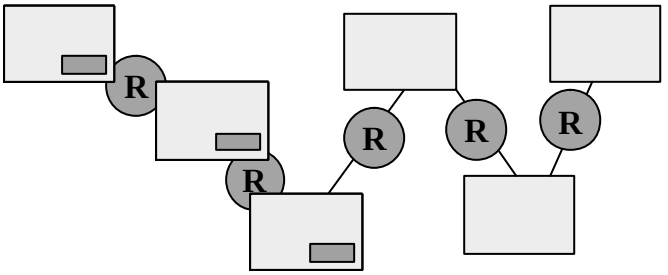
Reliability

Peter Lo

Reliability

- Progress so far
  - What (in networking terms) do we know how to do?

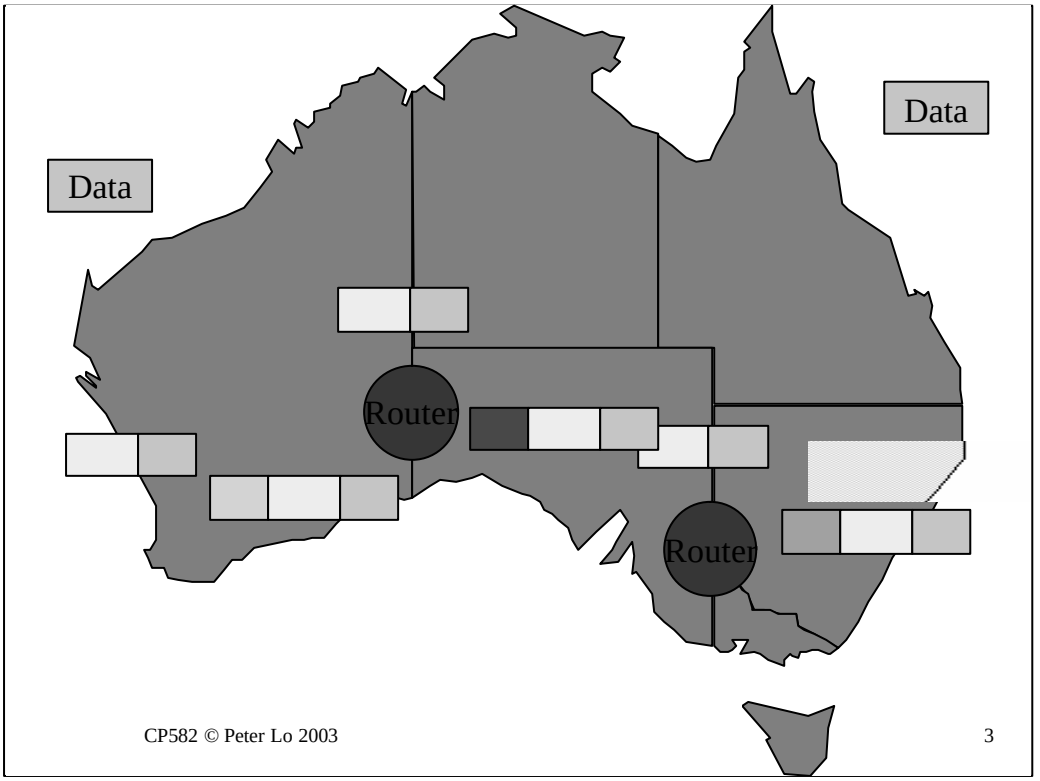


After several weeks of Networking Protocol and Services - we have really focused on how networks relay data - the explanations have led to a description of how a packet can be sent to a distant node through a collection of local networks joined together by routers.

This explanation rested on the earlier account of local delivery - the movement between each router is a local delivery issue.

Along the way we have pointed out the need to discover MAC addresses to address packets locally and we have introduced ARP as a means to do this.

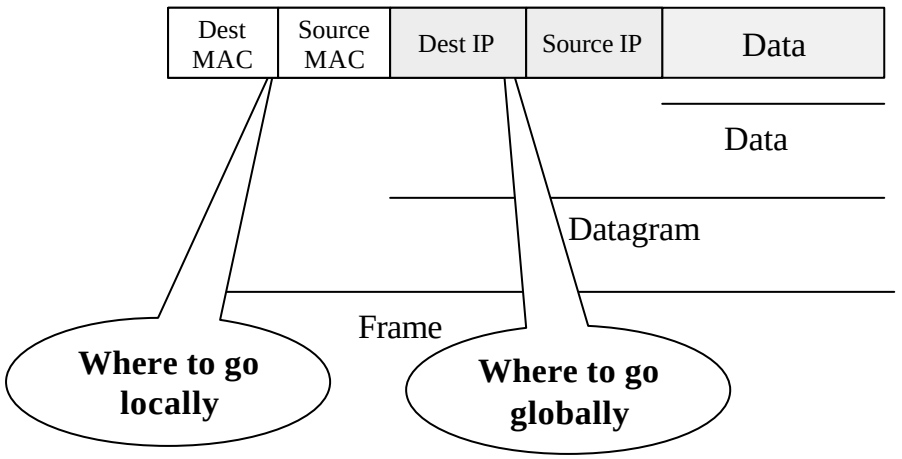
If we were to think in terms of "black boxing" we can put local delivery in a black box and then wrap that in another box named Remote Delivery. What we know how to do now is to delver a single datagram to a distant node and clearly this can be a mechanism that can be used for larger, more complex transmissions.



This animation reminds us of the detail of remote delivery.

The key concept to grasp is the way in which the frame header is discarded and replaced for each leg of the journey.

Datagrams and Frames



This slide, and the animation, is making a very fundamental point about networks.

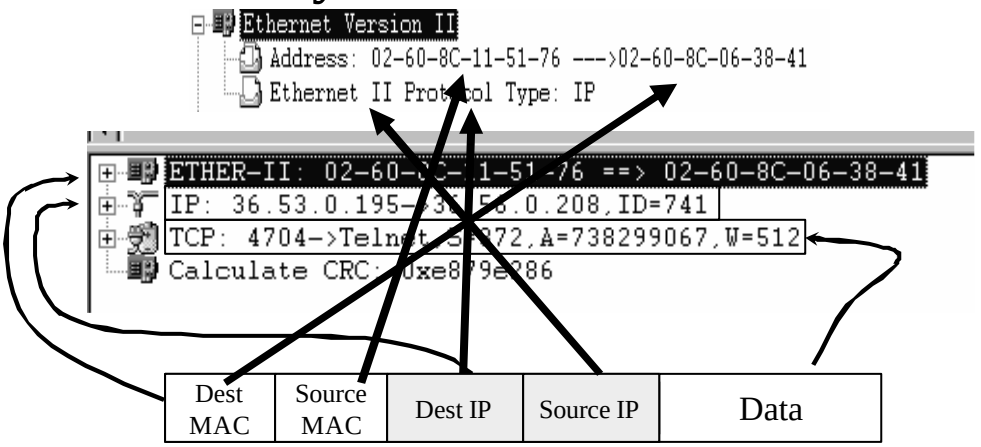
The original data is being wrapped by a series of headers that each have a specific role to play in the overall transmission.

In this example the first layer of wrapping is the datagram header. The datagram header has the job of storing the overall, long distance address (eg Father Clifford, Bally K, Ireland). We now have a datagram whereas before we just had data.

The second layer of wrapping is the frame header. The frame header **has a distinct purpose**. It has the job of storing the addressing for the particular local delivery step that we are inspecting (eg Spencer St, Tullamarine etc). It can be discarded when the local delivery is complete.

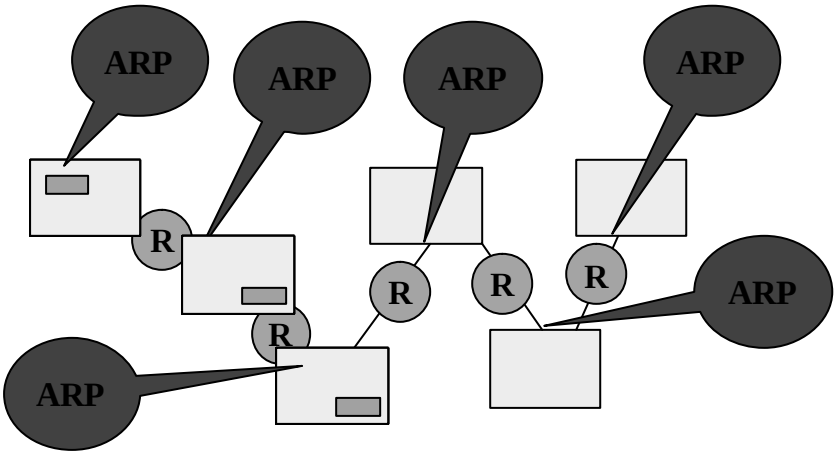
The animation in the next slide will make this clearer.

NetXRay view of this



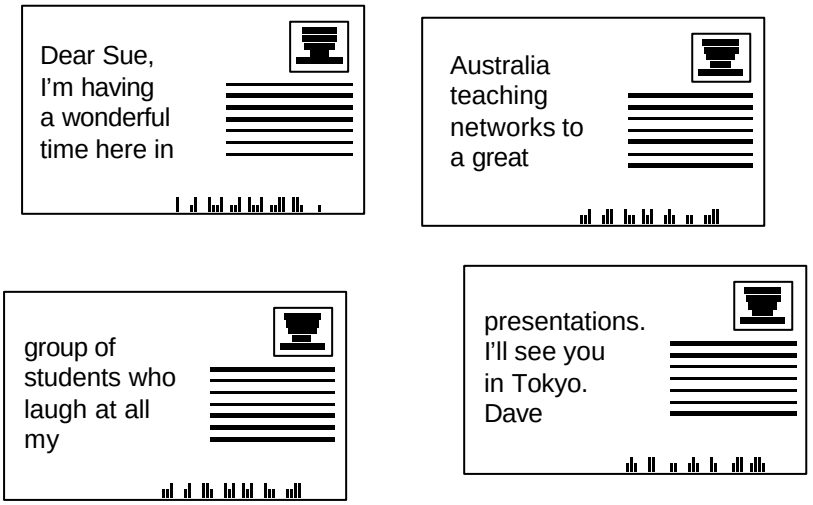
Seen from the point of view of NetXRay we can identify the series of headers within the packet and relate them to our box diagram of how a packet is built up.

The Place of ARP



This animation reinforces the observation that ARP gets used at every step along the way to discover the MAC address of the next router.

A packet is a “postcard”



The “packet” analogy that we used last week did not pay attention to the size of the packet. Really everything that was said applied to any size of data packet.

Realistically the size of packets is limited though. This can be justified when the network is considered to be a shared medium. If packets could be any size then one sender could monopolise the network by sending a never-ending packet. In practice everyone’s access to the network is broken down into small units so that each node’s traffic is interleaved with every other node’s traffic.

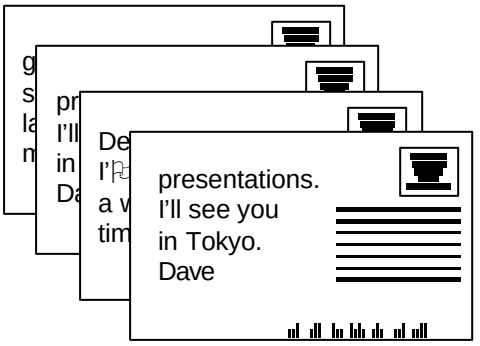
How big can packets be? It depends on the type of network being used:

- Ethernet - 1514 bytes
- Token Ring & Arcnet - 4202 bytes

The limited size of these packets is why we consider the delivery of a single datagram to be like the delivery of a postcard. Any more complex message is going to require a series of postcards and in this animation the postcards are all written and then they are all posted together. We will see what happens to them as they make their way across the world!

But what is received?

For example...



Not surprisingly the message becomes somewhat jumbled.

This example shows three types of corruption:

- One postcard did not arrive at all
- One arrived twice
- One was mangled along the way

Reliability

- What is it?
- Do we care?
- What are the threats?
- How is it preserved?
  - Detection
  - Recovery

So in terms of networking these are the questions we must ask:

Reliability Role Play

- The postcard protocol:
  - All postcards get addressed
  - A short message is broken into words, one per card
  - Delivery occurs by passing on the cards to “nodes” next to you
  - People with student numbers ending in 7 may do something to a card they receive
    - Change or erase one of the letters in a word (*sometimes*)
    - Destroy the card (*rarely*) - i.e., don’t pass it on
  - Receivers keep the cards in the order received
  - You have 2 minutes to effect message delivery!

This role play sets out to reinforce the nature of reliability:

• Students form small groups (say 5 people)

• One member becomes the originator of a message which is sent using a series of small cards that are passed from hand to hand to the furthest member of the group.

• The rules for this communication are shown in the slide

Reliability - What is it?

- Reliability is about the integrity of the data being delivered
  - In what ways was data affected in the postcard protocol?
  - Therefore, what constitutes a set of criteria for reliability?

The role play, if everything went well, will have resulted in a variety of damage to the delivered message.

Reliability - What is it?

- In-sequence delivery of data
  - In order
  - No duplication
    - How might duplication happen anyway??
- Error-free data
- Complete data
- Reliable protocols meet all of these 3 criteria

This slide offers a positive definition of what would be needed if we were to design a reliable protocol.

Each of the types of error that we have seen needs to be taken account of, and fixed, by some mechanism.

If we can do all of this then we have designed a “reliable” protocol.

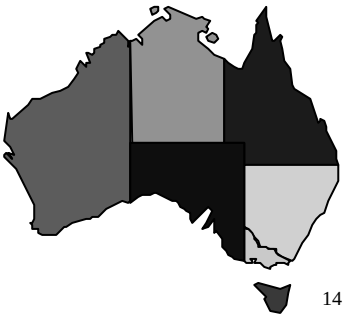
Over to you...

- Protocol design exercise
- Don’t forget to fix:
  - Corrupted packets
  - Duplicated packets
  - Missing packets
  - Out of order packets

Think about what mechanisms you would use to fix each of these problems.

Small digression - *state*

- State information
  - a record of the information relevant to the communication
    - e.g., what data has been received so far
- A protocol maintains state if it keeps such information



It is a fair bet that many of the solutions you produce will include some kind of counting of packets.

As soon as we try to impose a packet count that is shared between the sender and receiver we have a protocol that has “state”. Why is this significant?

Having “state” means that the whole series of steps has become a unit that has a beginning, a series of steps along the way and (hopefully) an end. If state is involved we cannot stop or start at any arbitrary point - we have do do the whole thing if we do any at all.

Most human communication involves state. One part of a conversation relates to the next and the previous parts. Sending a postcard is good exception though - postcard sending is an example of a **stateless** protocol.

Connections

- Connection-oriented protocols:
  - a connection is defined as existing for the period in which state information is held
  - this information may be used to ensure reliability

- ***Connectionless protocols:***
  - independent datagrams
  - may be reliable or unreliable
- ***Question:***
  - how would you classify the postcard protocol?

Once we have our heads around **state** we can use that term to define a **connection**.

In networking we speak of **connection oriented protocols**. We understand this though the telephone analogy. Each end considers itself to be part of a larger, long lasting entity - the "connection" and there are some rules ("take turns talking", "repeat if not understood by receiver") that govern the connection.

In networking terms we consider a connection to last for as long as we maintain a set of state information. To put it another way - if we set up a new set of state information (new packet count for example) then we have a new connection.

The state information in a connection is what makes the communication reliable.

Don't forget to define the opposite - **connectionless protocols**

Connectionless protocols may be reliable or unreliable - the difference is based on whether we make any effort at all to check or acknowledge the correct delivery of each packet.

Choosing Unreliability

- Why would someone design an unreliable protocol?
  - "unreliable" doesn't mean "bad"
  - Some answers:
    - Sufficiency
    - Efficiency
    - Simplicity
  - A matter of appropriateness
    - Unreliable protocols are bad only if inappropriate
    - Think of some uses….

Defining reliability naturally leads us to defining unreliability and it is important to realise that there may be times when this is exactly what we need.

There are a number of arguments that can be offered to justify using a protocol that is technically "unreliable":

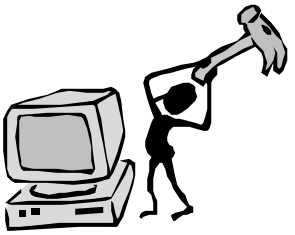
• **Sufficiency** - it does what is required. Any more effort would be overkill. The broadcast used in ARP is a good example. It would be totally beside the point to receive confirmation from all the nodes on the local network that they had received our ARP broadcast.

• **Efficiency** - it makes good use of the network bandwidth. All that error checking uses extra packets so an unreliable protocol makes more efficient use of the network.

• **Simplicity** - it is easy to understand and does not require any state information to be maintained. HTTP (Hyper Text Transfer Protocol) which is used in the WWW is "unreliable". Each page request from a client appears to the server as a unique new request. This makes servers (and clients) much simpler to design.

Threats to Reliability

- Threats to sequencing:
  - Independent packet routing
  - Connectionless protocols
- Threats to error-free data:
  - Corruption of the signal
    - Noise
    - Attenuation
- Threats to completeness
  - as above (what happens if address corrupted?)
  - time-out



We have seen the kinds of unpleasant things that can happen to packets but what exactly causes them?

Sequencing problems

• Sequencing, as a problem that needs to be solved, arises because each packet travels independently through the network. The routing decisions that are made by the routers along the way are, to a certain extent, load related. Routers are smart enough (due to ICMP - Internet Control Message Protocol) to route packets around congested areas. Hence each packet may take a unique route from A to B and this is how they can arrive out of order.

• Use of connectionless protocols can cause sequencing problems - there is no record of the sequence required.

Errors in data

• Noise - which is unwanted electrical signals that get mixed with the data - can corrupt the signal that is received. Crackling on radio due to an electrical appliance in operation is a good example of noise.

• Attenuation is the progress decline in a signal as it travels farther and farther. The TV signal in the little village where I live is attenuated and we watch some channels through a fuzzy haze. What has happened here is that the signal is so weak, and the TV is trying so hard to boost it, that the background noise is being boosted as well. Digital signals are less prone to this but an over-long network segment could cause data corruption.

Missing packets

• Data corruption may be so severe that packets go missing. This is especially true if the addressing of packets is damaged. The packet no longer knows where to go!

• Time out is the reaction to a missing packet. The receiving node can only wait so long for a packet that it expects to receive. When the limit of waiting the receiver **times out.**

The 3-way Handshake

- X sends an open-connection request to Y
  - How does Y know that this is not an old request packet from X?
- Y responds to X asking for another packet
  - Are you (X) still interested?
- X sends that packet
  - Which means that Y knows X is still interested
- For each of the above ask "what if it never arrives?"
- This is a 3-way "handshake"
  - Used by TCP

The trick to establishing a connection safely, in an environment where things may go wrong, is to devise a **handshake** that has sufficient checks in it.

The 3-way handshake is the classic solution adopted by TCP the **reliable** protocol on the Internet (now you can see why it is described as reliable!)

Step 1

X asks Y for a connection. At this stage. if Y receives the request, it only knows that X, at some point in the past, wanted a connection. Y does not yet know if X still wants a connection.

Step 2

Y replies. By asking for another packet in this reply Y is making sure that X is still interested in making a connection

Step 3

When X receives the second packet from Y it knows that Y wants to talk and the third packet is letting Y know that:

- The second packet got through successfully
- That the initial request from X was current

Some Detection Methods

- Sequence numbering
  - But what about sequence number starvation?
    - It's a finite world! (You've heard of Y2K?)
- Error detection schemes
  - e.g. CRC (Cyclic Redundancy Check)
    - you'll see reference to this using NetXRay, etc.
  - But what about the (rare) failure events?
- Timeouts
  - But what happens when the timeout period is…
    - Too long?
    - Too short?

This slide raises possible problems with the techniques used for detecting errors.

Sequence numbers

Numbering each packet is an obvious fix for sequencing, and completeness, problems but if the sequence number is too small (by which we mean from a selection of numbers that is too small) then numbers may begin to repeat and packets will not be uniquely identified.

This is known as sequence number starvation.

Error detection

The classic approach is to send, along with each packet, a cross-check in the form of a number that can be calculated by reading the data in the packet.

A simple minded approach would be to simply add together all the bytes in the packet but this might fail to detect multiple errors - what is one byte went down by two whilst a separate byte went up by an equal amount?

In practice more complex maths is used to derive the "check sum" - the common approach is the "Cyclic Redundancy Check"

If the receiver cannot reproduce the CRC that the sender calculated then either the data or the CRC is damaged. In either event the packet must be resent.

Timeout

Deciding how long to wait for an expected packet is one of the hardest issues.

If we have a timeout that is too short then the receiver stops waiting and requests a resend whilst the packet is still en route. The sender will resend and the receiver will then have to cope with a duplication issue!

If the timeout is too long then the whole transmission may be slowed down by the long waits that occur when there is a failure for odd packets. Asking for a resend sooner may speed things up.

Recovery

- Once an error has been detected…
  - Report it and seek retransmission
- or
  - Correct it
    - Some error detection schemes may be able to correct certain errors by the use of built-in redundancy (i.e. extra information)
    - Which of these (report/correct) would you use for your design of a protocol for a probe which will explore the atmosphere of Jupiter?
- Here, we focus on reporting...

When an error has been detected there are two ways to go about correcting it

Report it

Tell the sender about the damage and ask for a new version of the damaged data.

Fix it

This is only practicable if there is some redundancy in the data which enables damaged regions to be rebuilt correctly.

This is how the human body works. DNA provides, massively replicated, instructions for repairing damage that is not too extensive. More major damage, such as loss of a limb, is beyond the scope of this redundancy scheme and regrettably, the"report and request resend" technique is not an option!

The networking that we use (TCP, IPX) uses reporting but certain, specialised applications, may use correcting.

Recovery

- Acknowledgements (ACKs)
- Negative Acknowledgements (NAKs)
  - These are used in a number of ARQ (Automatic Repeat reQuest) error-reporting schemes:
    - Stop And Wait (SAW)
    - Go Back N (GBN)
    - Selective Repeat (SR)

These are some of the options and techniques for reporting errors. The basic idea is to let the sender know when a packet has been successfully received.

The acknowledgment of a successful packet is termed an ACK. The complete lack of an ACK (!) for a packet indicates that it was lost.

If a packet arrives but is damaged then the receiver replies with a 'Negative Acknowledgment" (NAK). There are a variety of Automatic Repeat Request (ARQ) schemes for NAKs that vary in the efficiency of the resending process:

Stop and wait (SAW)

The sender will not proceed with the next packet until it has an ACK for the previous one. This could be slow!

Go Back N (GBN)

If the sender is more aggressive then there is the possibility that an error may be detected in a packet that was sent some time ago. In GBN the receiver asks for all the packets since the error to be resent.

Selective Repeat (SR)

Here a mechanism is provided for the receiver to explicitly request the resend of only the data that was damaged.

The Postcard Protocol Lives!

- In the following examples:
  - Use an error detection scheme based on number of characters and the number of vertical and horizontal lines in your characters
    - “THE” -> C=3, H=5, V=4
    - “SEX” ???? Decide on something!
  - Use one-digit sequence numbers
  - Use an initial timeout period of 30 seconds
  - Receivers should therefore employ:
    - a timekeeper, error-checker, and ACK-er

This is a more complex role play in which we will try to reproduce the various schemes for reporting and correcting errors using our postcards.

Go-Back-N

- The Go-Back-n (GBN) ARQ scheme
  - The sequence is maintained "by the network":
    - When a packet arrives out of sequence, send a NAK (specifying its number)
    - The sender resends from that point
      - so needs a "good-sized" send buffer (copies of sent packets)
    - ACKs may be sent (one acknowledges all before it)
  - Cost:
    - Simple to implement
    - The receiver has a buffer of size 1
    - Extra traffic due to network ordering responsibility

This slide shows the working of the GBN scheme.

No attempt is made to fix up sequencing errors - packets are simply sent in order and we rely on the network to keep them in order. As soon as this fails, and a packet is received out-of-sequence, the receiver NAKs.

For example:

Sending: 1,2,3,4,5,6

Receiving: 1,2,3,6,4,5

Reply: 1 ACK, 2 ACK, 3 ACK, 6 NAK, 4 NAK, 5 NAK

Resend: 4, 5, 6 (3 ACK told us where to start, 6 NAK told us where to end)

Reply: 4 ACK, 5 ACK, 6 ACK (hopefully!)

GBN is simple to implement and does not require the receiver to store packets other than the latest packet.

The downside is that the resends may end up being long sequences of packets and this generates excessive network traffic.

Selective Repeat

- The Selective Repeat (SR) scheme
  - Sequence maintained by the receiver
    - When a damaged packet arrives a NAK is sent
    - The sender resends that packet
    - ACKs will acknowledge only specific packets successfully received
  - Cost:
    - The buffering and reordering at the receiver
    - Error bursts are handled inefficiently

The SR scheme supposes that the receiver has a large enough buffer to be able to "juggle" a sequence of received packets, sorting them into correct sequence. This removes the dependency on the network maintaining sequence.

An example:

Sending: 1,2,3,4,5,6

Receiving: 1,2,3,6,4,5 (6 was damaged)

Reply: 1 ACK, 2 ACK, 3 ACK, 6 NAK, 4 ACK, 5 ACK

Resend: 6