Solution :

Data Frames : number the Frames, (sequence number).
Include the sequence number in the Frame.

Acknowledgements :
Include sequence number of Frame being acknowledged.

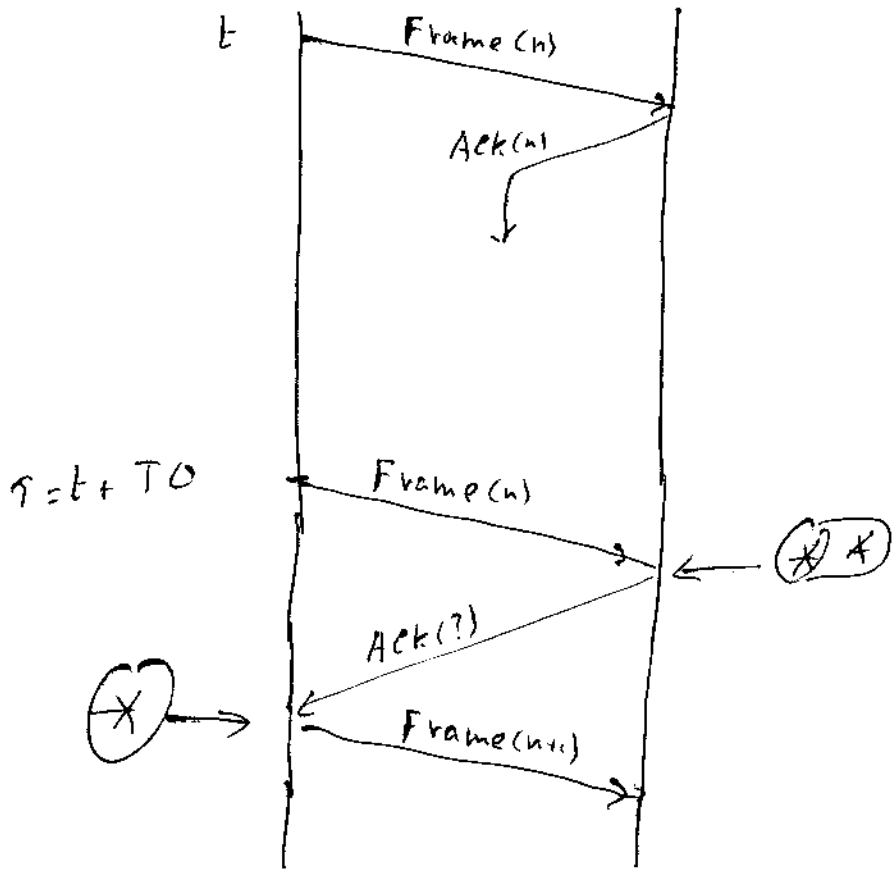
NACK : better not include sequence number : could be wrong !

Better not include sequence number : could be wrong !

To here 09/30/2003

In the situation of p 97,
Mr Haynes observed the same effect results if an Ack gets lost:

There is No p 96



(**) : The dest "thinks" it receives Frame (n+1), but it actually is a Re-Xmt of Frame (n).

Result: Worse than deadlock!
Loss of data reliability.

Stop & Wait Implementation.

Frames <u>numbered</u>. Sequence Number.
(Somewhere in Frame: part of standard).
Frame (seq)

Source maintains: source_hacked
(Highest sequence number for which an ACK has been received).

Destination maintains: dest_hacked
(Highest sequence number for which an acknowledgement has been sent.

ACK (seq) acknowledges Frame (seq).

NACK does <u>not</u> contain a sequence number.

The protocol we discuss is based on the assumption that the source <u>always</u> has a Frame to send.

(Actually: dest_hacked is <u>not</u> needed:
Exercise.)

Stop & Wait,

Correct Implementation.

(Not necessarily efficient)

Source maintains a variable

source_acked

(higest sequence number for which an ACK has been received.)

Destination mainteins a variable

dest_acked

(highest sequence number for which it has sent an ACK.

Actually: dest_acked is not needed,

(Excercise),

But serves to illustrate an issue.

Source :

1. At time t, send Frame (n).
   (necessarily : source_acked = n-1).
   (Re)-set time-out = $t + TO$.

2. Wait For ACK or NACK or time-out, whichever happens First.

3. IF ACK : at time τ, $t < \tau < t+TO$, ACK(seq) arrives.
   Check : is source_acked = seq - 1 ?
   (This is the only Legal possibility.
   Lets pretend this is always true.
   More Later ⊛).
   IF (seq = source_acked + 1)
   {
   source_acked = seq ;
   Send Frame (seq + 1);
   Set time-out = $\tau + TO$;
   Goto (2) ;
   }

Source, continued.

4. IF NACK : at time τ, $t < \tau < t + TO$, a NACK arrives.

{ Re_transmit Frame (source_backed +1);
Re_set time_out = τ + TO';
Goto (2);
}

5 IF time out at $\tau = t + TO$:

{ Re_transmit Frame (source_backed +1);
~~Collect consumption~~
Re_set time_out = τ + TO'';
Goto (2);
}

Stop & Wait,
Destination.

1. Wait For Frame.
2. IF Frame:
   Check CRC or Checksum
   IF wrong: send NACK,
   Goto (1).
   IF right:
   { Check sequence number of Frame (seq).
   iF seq = dest_ hacked +1 (good!)
   { Send ACK (seq);
   dest_ hacked = seq;
   Goto (1);
   }
   iF seq = dest_ hacked (can this happen?
   YES!)
   { Send ACK (seq);
   Goto (1);
   }
   What iF seq < dest_ hacked, or
   seq > dest_ hacked +1 ?
   "Can not happen" (?!?).
   }

"Same"!

Is it possible the receiver receives a packet with the ~~wrong~~ wrong sequence number?

Depends on what errors we assume are possible.

Simple assumption:

Source & Destination never make errors. (Do not go down.) (!?!).
Link can deliver Frame safely,
drop it,
deliver damaged & recognizably so.

No other possibility.

With these assumptions, ~~a~~ the destination never receives a Frame with wrong seq. nnmm.

Can we prove this implementation is correct? (What does correct mean?).

Finite State Machines.

Tanenbaum pp 229 - 232 : later.

For more: CIS 341.
(More on Finite State machines).

Is this "Stop & Wait" protocol efficient?

Depends.

if serialization delay >> propagation delay,
Stop & Wait is Fine.
even if "same order": Ok.

But if serialization delay << propagation delay:
Stop & Wait is Bad
(Not wrong! But inefficient).

Distance : D

speed : V (velocity)

Bandwidth : B (bits/sec)

Packet size : S (bits).

serialization delay : $\frac{S}{B}$

propagation delay : $\frac{D}{V}$

Length of packet : $\frac{S}{B} * V$ (in "meters" or km or miles)

if ~~$\frac{SV}{B} < D$~~ :

Fraction of bandwidth used :

$\sim$ ($\frac{SV}{B} < D$) :

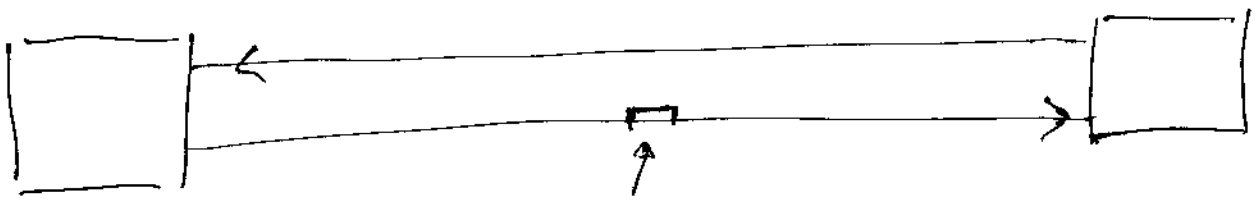
$\sim \frac{1}{2} \cdot \frac{S.V}{B.D}$ ($\frac{1}{2}$: For "ACK").

Exact computation depends on sizes of Frame, ACK, ~~CTRL~~

In our case
(data one way)
$|ACK| << |Frame|$

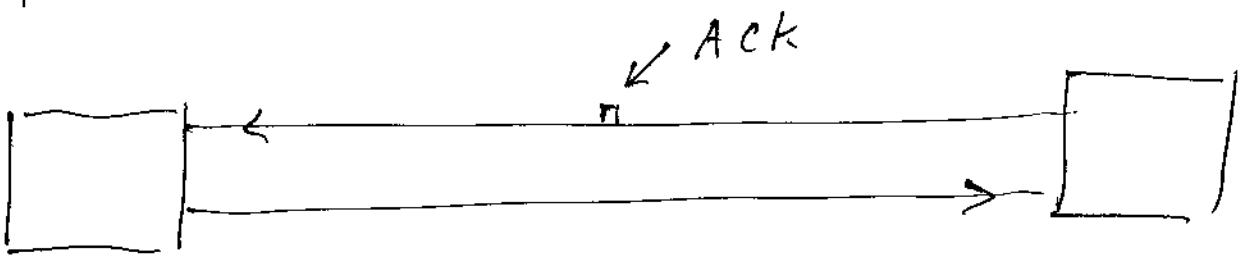
IF $\frac{S}{B} << \frac{D}{V}$

(serialization delay << propagation delay):



Frame.

Rest of Link(s) unused.



IF $\frac{S}{B} << \frac{D}{V}$:

stop & wait inefficient.

Solution ?

Sliding window.

Window Size W.

Roughly :

(at Source)

if source_backed = n,

you can send Frames up to

$\leq (n+w)$.

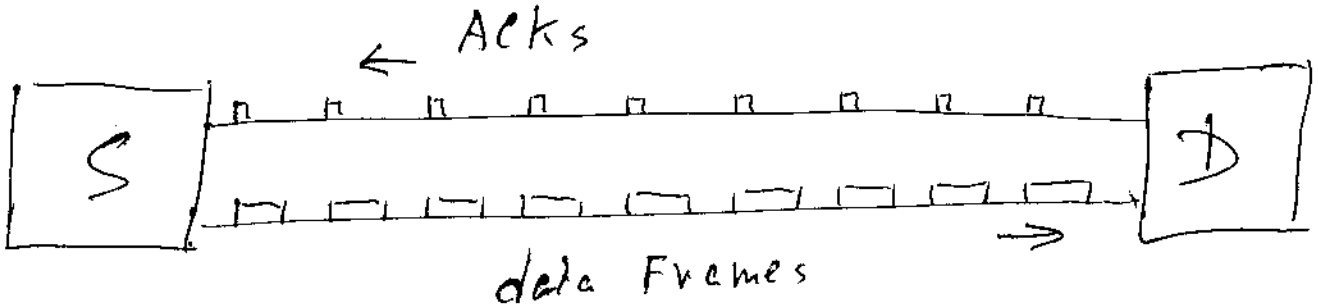
Does this define the protocol ?

No.

Devil is in the details.

Sliding Window.

Ideally:



W should be "just large enough" to make this happen.

(~~Se~~. Take CIS 456 or CIS 656).

Stop & Wait (practically) is a sliding window protocol with W = 1 (1 Frame).

Stop & Wait: also called

ARQ: Automatic Repeat reQuest

PAR: Positive Acknowledgement with Re-transmission.

Sliding Window.

Basic Pattern:

ACK(seq) means:

"I have received ~~Frame(q)~~ Frame(seq)
and all Frames Frame(k), k ≤ seq,
in good condition".

(ACKs are cumulative).

Source maintains:

source_acked (highest Ack received)
source_hsent (highest Frame "sent").

(source_hsent ≤ source_acked + W)

Typically:

IF source receives ACK(seq),
it must be true that

source_acked ⊛ ≤ seq ≤ source_hsent,

and it gives the source the right to
send packets
source_hsent +1, ..., seq + W.

And to set source_acked = seq.

⊛: think!

What does the source do when it "thinks" a packet got damaged or lost?

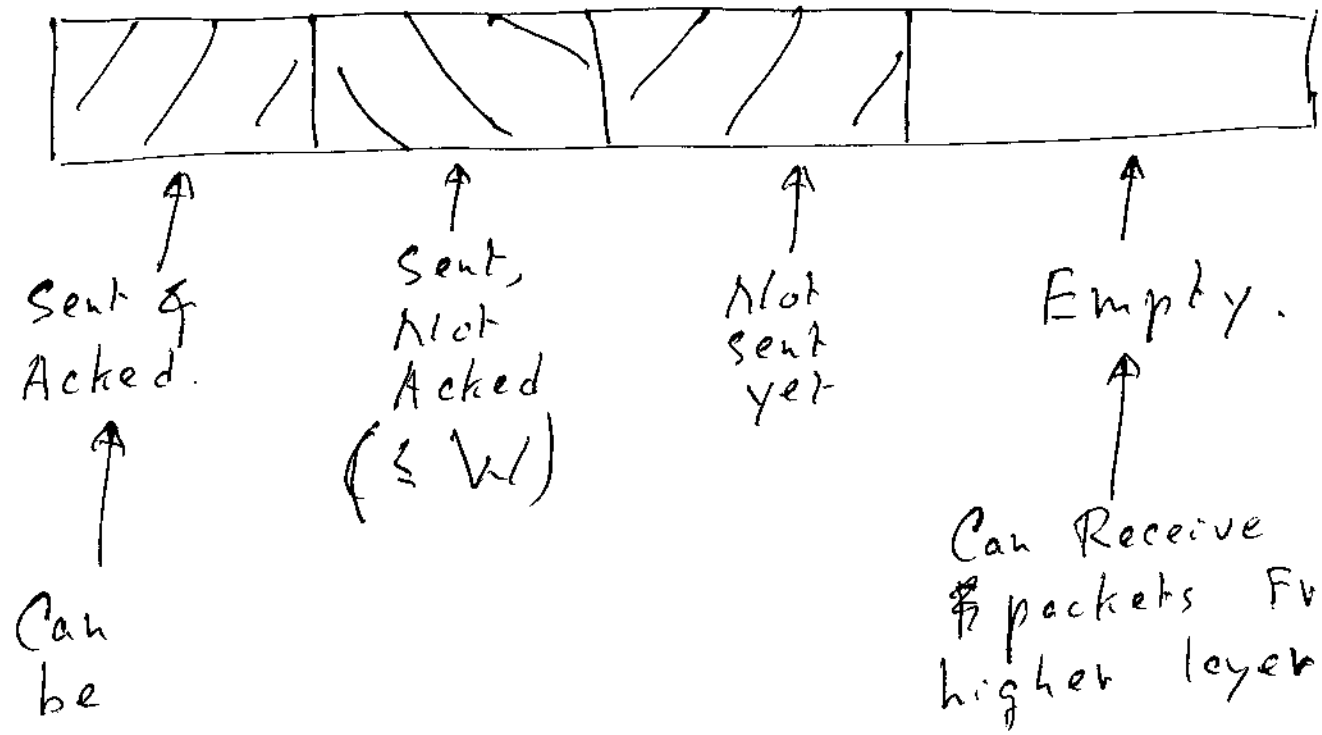
Two extremes:

(1) Start Re-transmitting From source_hacked +1 on.
Called: "Go back n".

(2) Try to Find out which Frames the dest does not have, Re-transmit <u>those</u>.
Called: Selective Repeat.

"Go back n" may (will) re-transmit more Frames, but is simpler, in particular For the <u>dest</u>.

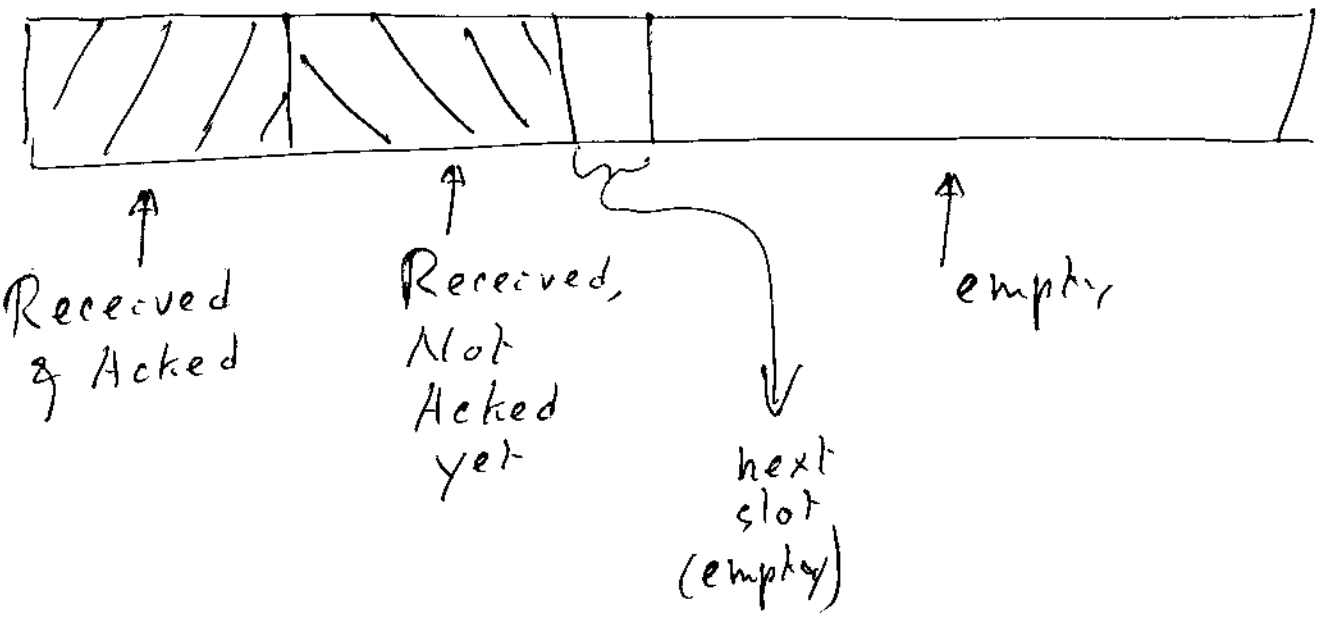
Source:
Transmit Buffer
(Send Buffer):



Destination.

Receive Buffer.

In case "Go back n":



First priority: Acknowledge Frames Received, Not Acked.

Second priority: Send all Received Packets to higher layer.

Destination,
Go back n.

IF Frame (seq) arrives:

(1) IF it Fits in "next slot":
put it in.
(& do your thing: ACK etc.)

(2) IF it <u>does not</u> Fit in
"next slot": <u>Discard</u>.
(& do your thing: ??)

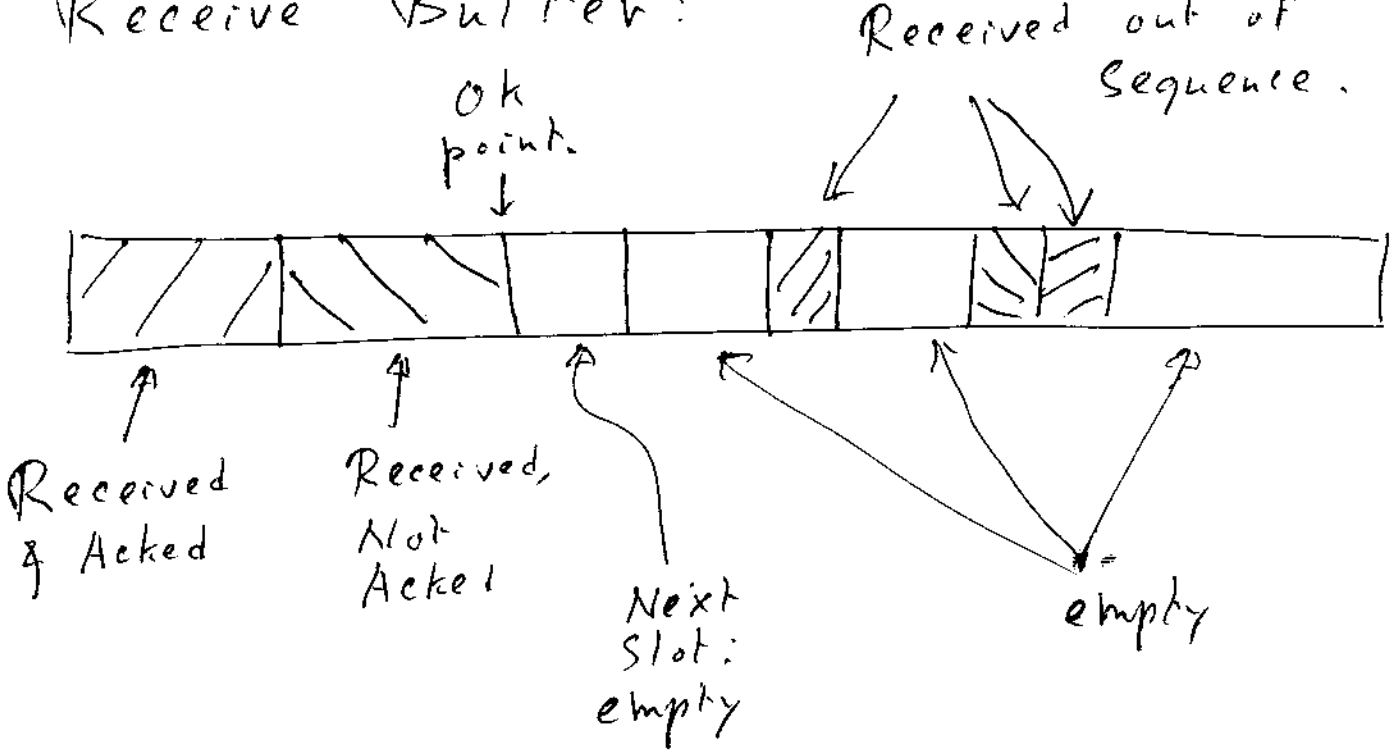
<u>What</u> if a packet arrives
out of sequence?
You <u>can not</u> ACK that packet!

<u>Later</u>.

Destination

Selective Repeat.

Receive Buffer:



First priority: Acknowledge all Frames up to Ok-point.

Second priority: Send all <u>Acked</u> packets to higher layer.

Destination.

Selective Repeat.

IF Frame (seq) arrives:

(1) IF it Fits in "next slot" (just to right of Ok point):
Put it in.
(& Do your thing: ACK etc.).

(2) IF it <u>does not</u> Fit in "next slot" (seq too high):
Put it in appropriate location.
(& Do your thing: ??)

<u>What</u> if a packet arrives out of sequence?
You <u>can not</u> ACK that packet!
Maybe much later

Or: Excercise?

Go back n.

Possible destination implementations.
(details left out.)

IF frame arrives out-of-sequence:

- Drop Frame
- { either:

1. do nothing. (rely on time-out), or
2. Send NACK. or
3. Again send Ack(dest_hacked).
   "duplicate acknowledgment".
   (TCP does this. But different.).
   }

IF the destination uses
Duplicate Acknowledgements,

This means For the source:
IF Ack(seq) is received For the
second (third, Fourth, ...) time,
it means the destination received
Frame(seq) (& all previous Frames)
in good condition,
~~R~~ Did <u>not</u> receive Frame(seq+1),
but received one (or two, or three, ...)
<u>later</u> Frames.

"Probably": Frame(seq+1) got
lost. (or was dameged)

Go Back n.

Source Behavior.

Suppose destination behavior is:

Whenever ~~a~~ Frame fails CRC,
and whenever Frame(seq) arrives, with
seq > dest_hacked + 1 ("Out of sequence")

Send NACK.

What should source behavior be?

One possibility:

(Re)transmit

Frame(source_hacked + 1), Frame(source_hacked + 2), ...
..., up to (at most)
Frame(source_hacked + W).

Is this a good idea?

Was this a good idea?

(1) It is correct: does not lead to deadlock or data integrity
(Assuming details are taken care of).
(Assuming dest can handle packets arriving multiple times, etc.)

(2) It can be very inefficient.

Suppose "everything ok" up to Frame(n).
Frame(n) Lost without trace.
Frame(n+1), Frame(n+2), ..., Frame(n-1+W)
arrive at dest while
dest_backed = n-1.

(W-1) NACKs.

Source Re-transmits W Frames,
each (W-1) times!

Waste of Bandwidth, Time.

Several Solutions.

One is :

When NACk arrives at time t
(& "First of Bunch")

Re-transmit

Frame (source_hacked +1), ...,
Frame (source_hacked + W).

Set time-out = t + TO*.

Until time t + TO* :

Disregard all other NACKs,
all time-outs.

At time $\tau = t + TO^*$:

if source_hacked
has <u>increased</u> since time t :
Back to Normal.
if source_hacked has <u>not</u> increased :
Re-transmit same bunch again.
Re-set time-out = $\tau + TO^{**}$. <u>etc</u>.

IF dest uses

"Do nothing"

or "Duplicate Acknowledgements":

Similar, (with differences).

Sequence Numbers in Frames:

- Used to recognize "holes" in the stream of Frames
- Used to recognize Re-transmissions (in Selective Repeat).
- Used to re-order Frames arriving out-of-order.

TCP (Layer 4) uses

"Duplicate Acknowledgements" &

Time-outs &

Bells & whistles.