ARQ techniques (window protocols)

Gruppo Reti TLC
nome.cognome@polito.it
http://www.telematica.polito.it/

Techniques to protect against transmission errors

- FEC (Forward Error Correction)
- ARQ (Automatic Retransmission reQuest)

Both need additional bits in the packet header
FEC requires more bits because correcting is more difficult than detecting
Both have limited capability (if too many errors, nothing can be done)

Block codes for error control

\[ n \text{ bit} \]
\[ k \text{ bit for user data} \quad n-k \text{ parity bit} \]

\[ 2^k \text{ possibile combinations} \]
**Examples of error protection**

- **Parity bit**
  - Detects bit errors in odd numbers
  - 1 error more likely than \( k \) errors

- **Repetition code**
  - Majority decision
  - Error correction possible
  - Requires more overhead

**Packet header**

- Parity (or error protection) bits are available in PDU header (PCI)
- Classical example of parity bit are CRC (cyclic redundancy check)
- Coding theory: select the code that given an amount of bit best suits the specific channel

**ARQ**

- Joint control over a connection of
  - Error (losses)
  - Sequence
  - Flow
- Packet numbering bit are needed in the PDU PCI
- Basic idea: send a packet and wait for a confirmation (ACK)
  - If not received, retransmit
ARQ

- Three ARQ flavours
  - Stop and wait (Alternating bit)
  - Go back N
  - Selective repeat
- We focus on a unidirectional communication
- In bidirectional connections, two independent mechanisms run in the two opposite directions

Stop and wait

- Transmitter
  - Makes a copy of the PDU (for possible retransmission) and stores it in the tx buffer
  - Sends the PDU
  - Activates a timer (timeout)
  - Waits for the ACK
  - If timeout expires before ACK reception, repeats the transmission of the PDU, reactivating the timeout

ARQ

- Acknowledgement packet header
  - Parity bit on the header only
  - N(R), next expected packet number (convention)
  - Addresses

Stop and wait

- Receiver
  - On PDU reception
  - Checks PDU correctness
    - If the PDU is correct (regardless the sequence number) sends the proper ACK
    - Check PDU sequence number
    - If the PDU has the expected number, it is moved to the higher layer protocols for further processing
### Stop and wait

**Initialization**
- Connection set up at connection opening
  - TX and RX agree on protocol parameters
  - $V(T) = 0$ at the transmitter
  - $V(R) = 0$ at the receiver

<table>
<thead>
<tr>
<th>TX</th>
<th>$V(T)=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>$V(R)=0$</td>
</tr>
</tbody>
</table>

**PDU transmission with $N(T) = V(T)$**
- Timer is started

<table>
<thead>
<tr>
<th>TX</th>
<th>$V(T)=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>$V(R)=0$</td>
</tr>
</tbody>
</table>

**PDU reception**
- Check for correctness
- Check sequence: $N(T) = V(R)$?

<table>
<thead>
<tr>
<th>TX</th>
<th>$V(T)=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>$V(R)=0$</td>
</tr>
</tbody>
</table>

**ACK reception**
- Sequence control
  - $N(R) = V(T) + 1$?
  - Stop the timer
  - Erase PDU copy from the buffer

<table>
<thead>
<tr>
<th>TX</th>
<th>$V(T)=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>$V(R)=1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TX</th>
<th>$V(T)=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>$V(R)=1$</td>
</tr>
</tbody>
</table>
Stop and wait

- $V(T) = 1$ at the transmitter
- $V(R) = 1$ at the receiver

Stop and wait

- PDU transmission with $N(T) = V(T)$
- Timer is started

Stop and wait

- PDU reception
  - Check for correctness
  - Check sequence: $N(T) = V(R)$?

Stop and wait

- Increase $V(R)$
- Send and ACK with $N(R) = V(R)$

Stop and wait

- ACK reception
  - Sequence control
    - $N(R) = V(T) + 1$?
  - Stop the timer
  - Erase PDU copy from the buffer

Stop and wait

- Increment $V(T)$
Stop and wait

• V(T) = 2 at the transmitter
• V(R) = 2 at the receiver

Stop and wait

• Numbering PDUs is mandatory
• Numbering scheme is cyclic
  - fixed number of bit in the header and finite counter size
• One PDU sent per cycle
  - The cycle duration is named round trip time (RTT)

Stop and wait

• One single bit for numbering: Alternating bit protocol

Alternating bit protocol

Reception of a PDU with errors

(PDU loss)

Reception of a PDU with errors

• Timeout value setting is critical
Reception of an ACK with errors

\[ V(T) = 0 \quad N(T) = 0 \quad V(R) = 0 \]
\[ V(T) = 0 \quad N(R) = 1 \quad V(R) = 1 \]
\[ V(T) = 1 \quad N(T) = 1 \quad V(R) = 1 \]
\[ V(T) = 1 \quad N(R) = 0 \quad V(R) = 0 \]

• Need to number PDUs
  → Otherwise the PDU duplication would not be detected

Non sequential channel

\[ V(T) = 0 \quad N(T) = 0 \quad V(R) = 0 \]
\[ V(T) = 0 \quad N(R) = 1 \quad V(R) = 1 \]
\[ V(T) = 1 \quad N(T) = 1 \quad V(R) = 1 \]
\[ V(T) = 0 \quad N(R) = 0 \quad V(R) = 0 \]
Non sequential channel

- Over a non sequential channel the alternating bit protocol may not be able to correctly detect:
  - PDU losses
  - PDU duplication

Non sequential channel: modulo 4 numbering

- Protocol is blocked
  - Needed mechanism to restart (reset) the connection
- Problems occurrence become unlikely if using
  - More bit to number PDUs and ACKs
  - A maximum lifetime for PDUs and ACKs

Non sequential channel: modulo 4 numbering

- Numbering:
  - TX: V(T)
  - RX: V(R)
  - Numbering is modulo 4
  - Each number represents a PDU or ACK

<table>
<thead>
<tr>
<th>V(T)</th>
<th>N(T)</th>
<th>V(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- Connections:
  - TX and RX exchanges PDUs and ACKs
  - Connection is reset if V(T) = V(R) = 0

Non sequential channel: modulo 4 numbering

<table>
<thead>
<tr>
<th>V(T)</th>
<th>N(T)</th>
<th>V(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- Example scenario:
  - TX sends PDU 1
  - RX receives PDU 1
  - TX sends PDU 2
  - RX receives PDU 2
  - TX sends PDU 3
  - RX receives PDU 3
  - TX sends PDU 1 again
  - RX receives PDU 1 again
  - TX sends PDU 2 again
  - RX receives PDU 2 again
  - TX sends PDU 3 again
  - RX receives PDU 3 again
Go Back N

- The stop and wait protocols in absence of errors permits to successfully send one packet per RTT (round trip time)
- If the RTT is large, the transmitter throughput (amount of bit/s that can be sent) is limited by the protocol
- To increase throughput, in the Go Back N version the transmitter can send up to k PDUs before stopping while waiting for ACK reception

Transmission window

- The transmission window $W_T$ is the maximum number of PDUs that can be transmitted (in sequence) without receiving any ACK
- Limited by the amount of buffer space reserved at the transmitter
- It is also the maximum number of PDUs (or ACKs or PDUs+ACKs) propagating over the channel or in the network at a given time

Receiver window

- The receiver window $W_R$ defines the sequence of PDUs that the receiver is willing to store in a given time
- Limited by the amount of memory space reserved for the connection at the receiver

Transmission window

- Correctly Acked PDUs
- PDUs waiting for an ACK
- PDUs that can be transmitted
- PDUs that cannot be transmitted

Go Back N

- The transmitter
  - Sends up to $N = W_T$ PDUs, storing a copy of them in the TX memory
  - Starts a timer (usually one timer per window)
    - The timer is reset ad any PDU transmission
    - Waits for ACK reception
  - If timeout expires before ACK reception
    - It repeats the transmission of ALL PDUs that have not been correctly ACKed
    - "Goes back" by N positions

Receiver window

Set to one in Go Back N
Go Back N

- The receiver, when a PDU is received
  - Checks PDU correctness
  - If the PDU is correct, sends the ACK
- Checks the PDU sequence number
  - If the sequence number is the number of the first PDU that has not been received yet, it moves the PDU to higher layer protocols for further processing

ACK semantics

- Several ACK meaning can be used
  - Selective ACK:
    - Each ACK notifies the correct reception of a single PDU
    - $\text{ACK}(n)$ means "I received PDU n"
  - Cumulative ACK:
    - Each ACK notifies the correct reception of all PDUs with a sequence number smaller than $n$
    - $\text{ACK}(n)$ means "I received all PDUs up to n"
  - Negative ACK (NAK):
    - Request to retransmit a specific packet
    - $\text{NAK}(n)$ means "retransmit PDU n"
- TX and RX must agree on ACK semantic

Piggybacking

- For bi-directional information flows it is possible to include the ACK number on the PDU header travelling in the opposite direction
- This technique is known as piggybacking

WT and WR correct relative positions

$W_T$ and $W_R$ represent the window sizes for the transmitter and receiver, respectively. Single ACK is used.

PDU numbering

- PDU numbering is cyclic
  - With $k$ bit in the header
  - Numbering modulo $2^k$
- Example
  - 3 bit
  - $W_R = 1$
  - $W_T = 3$

Go Back N

- Transmitter is significantly more complex than the Stop and Wait transmitter
  - More memory
  - Timer management is more complex
  - PDU numbering more difficult
- The receiver is the same (window size 1)
- It is possible to group ACK (thanks to the cumulative meaning)
  - One ACK for several PDUs
  - Need a timer at the receiver
- Constraint on the maximum window size $W_T < 2^k$
We define the selective repeat (SR) as the window protocol where the receiver window is larger than one.

In Go back N, only in sequence PDUs can be received.

Accepting out of sequence PDUs may increase performance.

Usually transmitter and receiver window are of the same size.

Several options exist to implement the protocol, which differ for:
- Cumulative vs selective ACK
- Single timer per PDU or per window
- Slightly different behaviour of Tx and Rx

We focus on cumulative ACK and one timer per window.

The transmitter:
- Sends up to \( N = W_T \) PDUs, storing a copy of them in the TX memory.
- Starts a timer.
  - The timer is reset at any PDU transmission.
  - Waits for ACK reception.
- If timeout expires before ACK reception:
  - It repeats the transmission of ALL PDUs that have not been correctly ACKed.

The receiver, upon PDU reception:
- Checks PDU correctness.
- Check sequence number.
- If the PDUs is correct and in sequence:
  - Moves the PDUs (and all other PDU in sequence) to higher layer protocols.
- If the PDU is correct but not in sequence:
  - Stores the PDU if it is within the window.
  - Discards the PDU if it is out of the window.
- Sends an ACK for the last PDU correctly received in sequence.

Observations:
- If a single PDU is lost, the SR behaves as the Go back N in terms of throughput and channel occupation.
- Performance advantages if:
  - \( \text{RTT} = \text{window transmission time} \)
  - The new ACK permits to move ahead the TX window without the need to retransmit all PDUs in the transmission window.
  - For multiple PDU losses:
    - It is enough to receive a single copy of each packet.
  - By modifying the TX behavior (constraining it to retransmit only the first lost packet in the window) the channel occupation in case of losses can be reduced.

Constraint on the window size to ensure correct behaviour:
- \( W_T + W_R \leq 2^k \) where \( k \) is the number of bit used to number PDUs and ACKs.
- In the following example we highlight potential problems when this constraint is violated.

Selective repeat - Observations:

- Constraint on the window size to ensure correct behaviour:
  - \( W_T + W_R \leq 2^k \) where \( k \) is the number of bit used to number PDUs and ACKs.
  - In the following example we highlight potential problems when this constraint is violated.
    - \( W_T = 3, W_R = 2, k = 2 \)
    - Hypothesis of one timer per window.
    - We will show that:
      - 1 PDU will be accepted twice.
      - 1 PDU will be erroneously dropped.
In absence of errors, the throughput of a window protocol is ruled by
\[
\text{min}\left\{ \frac{\text{transmission\_window}}{\text{RTT}}, \frac{\text{link\_speed}}{} \right\}
\]
For a given RTT there is an optimal window size that maximize throughput
Larger windows translate in a RTT increase
“Short” connections get higher throughput for a given window size
To control TX throughput, it is possible to act on
- RTT (delaying ACK transmission)
- May induce retransmissions
- Window size

• Selective repeat

• Window protocols throughput