Tutorial 9 - Reliable Transport Protocols

Gidon Rosalki

2026-01-01

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

1 Reliable Transport

The concept is to provide the upper layer a service abstraction of a reliable channel, where no bits are corrupted or lost, and all bits are delivered in order. In order for this to hold, we need to make the network reliable, and ensure reliable communication over an unreliable / noisy channel. This naturally has costs, and other requirements, which will be discussed.

2 First attempt

In order for the network to be reliable, we could add the feature that when a packet is seen by a switch, we check that it reaches the next hop without corruptions, and if it is corrupted, then we resend. This has a few questions, what if 2 switches fail? Where must we buffer packets, and for how long? What is the resultant goodput, throughput, and packet delay? Since the network is unreliable, the packets may be delayed, lost, or corrupted. We thus need a mechanism to monitor any of these.

For the purposes of today, we are going to assume that we can always detect errors, that each sender always has packets to transmit, and packets are received in the order in which they are sent, without bypassing.

3 Stop and Wait

This is split into two sides, the sender, and receiver.
The **sender** sends a packet, adn starts a timer. It waits for T_{out} (timeout) units, to receive an ACK / NACK response. If it receives NACK (not acknowledged), or the timeout is reached, then it resends, and resets the timer. If it receives ACK, then it sends the next packet, and resets the timeout.
The **receiver** will send an ACK if it receives a packet, without an error, and NACK (or ignore) if the packet is corrupted. For our purposes, we will probably ignore NACKs, since it only mildly improves the speeds offered by the sender, letting the sender reach timeout is just as effective.

This is not quite sufficient, since the receiver cannot distinguish between a new message, and a resent message. We can try and resolve this, by adding a binary counter in the sent packets.
This is also not sufficient, since now the sender does not know which packet was ACKed. To resolve this, we also add a binary counter in the ACKs sent by the receiver.

The timer is initialised after sending the last bit of the packet, but we need to decide what is the minimal timeout T_{out} needed for this protocol to work. We need to consider:

- Packet siize
- ACK / NACK size
- Transmission rate (bandwidth)
- Distance between stations
- Propagation speed
- Processing time (T_{pt}) (unless told otherwise, only the receiver can have $T_{pt} > 0$)

In order to calculate the timout, we will first note that we only care about the time from the last sent bit, and the *worst* case ACK / packet size.

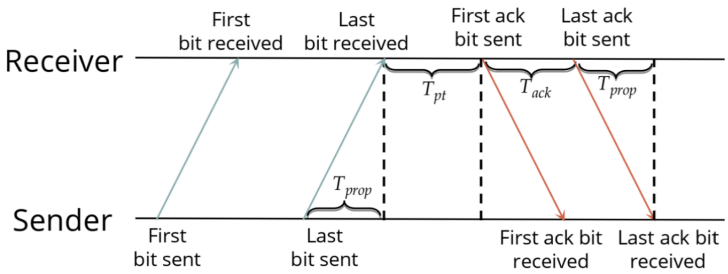


Figure 1: Timeout size

So we can see that

$$T_{out} \geq 2T_{prop} + T_{ack} + T_{pt}$$

Example 1. *Let:*

- *Packet size: 2000 bit*
- *ACK/NACK size: 200 bit*
- *Transmission rate (BW): 10 Kbps*
- *Distance between stations: 1000 km*
- *Propagation speed: 2 · 108m/s*
- *Processing time (Tpt): 0 sec*
- *What is the minimal timeout, needed for this protocol?*

Solution. We will note that the propagation speed is

$$\begin{aligned} T_{prop} &= \frac{\text{distance}}{\text{propagation speed}} \\ &= \frac{10^6 m}{2 \cdot 10^8 \frac{m}{s}} \\ &= 5 \cdot 10^{-3} s \end{aligned}$$

The ACK time is

$$\begin{aligned} T_{ack} &= \frac{\text{ACK size}}{\text{Transmission rate}} \\ &= \frac{200 bits}{10^4 \frac{b}{s}} \\ &= 2 \cdot 10^{-2} s \end{aligned}$$

So

$$\begin{aligned} T_{prop} &= 5 \cdot 10^{-3} s \\ T_{ack} &= 2 \cdot 10^{-2} s \\ T_{pt} &= 0 \\ T_{out} &\geq 2T_{prop} + T_{ack} + T_{pt} \\ &= 2 \cdot 5 \cdot 10^{-3} + 2 \cdot 10^{-2} + 0 \\ &= 3 \cdot 10^{-2} s \end{aligned}$$

□

Let us now consider the goodput. Let us assume that we use the whole bandwidth, and on failure, the receiver does not send NACK, but just ignores the packet. Packets /ACKs fail with probability

$$p \implies X \sim Geo\left((1-p)^2\right)$$

The expected number of tries until success (i.e. all attempts except the last fail):

$$\mathbb{E}[X] = \frac{1}{(1-p)^2}$$

The goodput is the ratio of effective transmission time, and total transmission time. Let T_{packet} be the transmission time of a data packet. So:

$$\text{Goodput} = \frac{T_{packet}}{\left(\frac{1}{(1-p)^2} - 1\right)(T_{packet} + T_{out}) + (T_{packet} + 2T_{prop} + T_{ack} + T_{pt})}$$

If T_{out} is set to the minimal value that we previously calculated $T_{out} = 2T_{prop} + T_{ack} + T_{pt}$ then

$$\text{Goodput} = \frac{T_{packet}}{\left(\frac{1}{(1-p)^2} - 1\right)(T_{packet} + T_{out})} = \frac{T_{packet}(1-p)^2}{T_{packet} + T_{out}}$$

We will note that doing this for *every packet* is expensive for the goodput, neatly bringing us on to the next section.

4 Go Back N (GBN)

In Stop and Wait we waste time during the timeout period. We can use this time to send additional packets. We will assume that the sender has a sliding window of size N. The sender will send N packets one after the other, and if it reaches a timeout, then it resends from the last known ACK. The receiver only sends ACKs for the packets within the sequence.

Sender: Window size of N packets, and send at most N unacknowledged packets. If a timeout occurs, resend all packets that have been sent but not yet acknowledged. $ACK(i)$ indicates that packets $1, 2 \ldots, i$ have been received correctly (meaning that it is cumulative). When an ACK is received, then the beginning of the window is moved to the next unacknowledged sequence number.

Receiver: Has a window size of a *single* packet, and if it receives a packet with sequence number i correctly, and in order (which is to say, that the last packet delivered to the upper layer has a sequence number $i-1$), then send $ACK(i)$, and deliver the data to the layer above. Otherwise the packet is corrupted, or not in order, so discard it, and send ACK for the most recently received in order packet.

The sender has a timer for each transmitted packet, which is started after sending the final bit of a packet.

Consider if we have a window size of 8, and use 3 bits to represent it. We can then desynchronise if all 8 packets are received, but all 8 ACKs are lost, since the sender will return to the beginning of the window, but the receiver will assume that this is new data. We can resolve this if for a window size of n, we use $\lceil \log_2(n) \rceil + 1$ bits to represent it.

4.1 Questions

Given a network with 2 nodes, A and B, we assume:

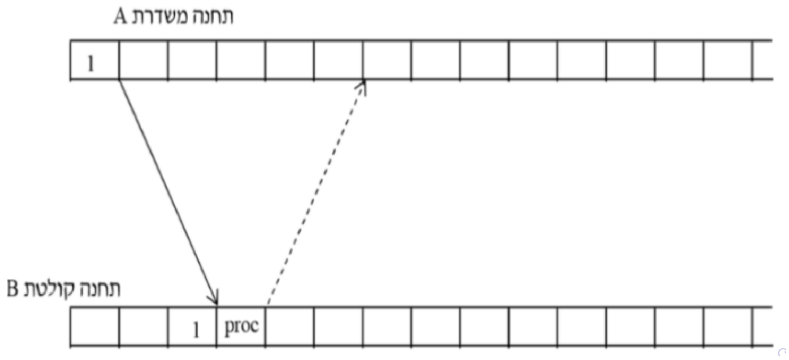
- Node A only sends data to B, B only ACKs to A
- No limit on buffers or sequence numbers
- The packet transmission (T_{packet}), propagation time (T_{prop}) and packet processing time (T_{pt}) are constant
- ACKs are delivered without any errors and $T_{ack} = 0$
- T_{out} is set to the minimal value that avoids unnecessary retransmissions
- $T_{out} = 2T_{prop} + T_{ack} + T_{pt}$
- Window size is equal to $T_{out} + 1$

4.1.1 Question 1

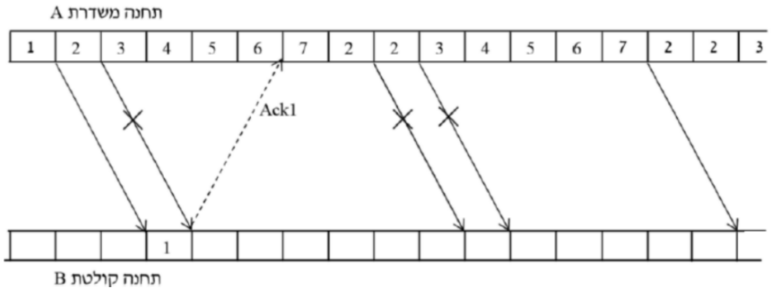
We will change the protocol as follows:

- When node A detects a failure, it re-transmits the failed packet L times
- Node A activates the timeout only after the last packet was sent
- Node A will keep transmitting the L packets even if got an ACK for one of them
- Node B will ACK the first packet and ignore all duplicates

Complete the diagram when $L = 2,\ T_{prop} = 2,\ T_{packet} = 1,\ T_{pt} = 1$, the transmissions of packet number 2 fail 3 times, and all other packets arrive successfully:



Sol. We will firstly compute $T_{out} = 2T_{prop} + T_{ack} + T_{pt} = 2 \cdot 2 + 1 + 0 = 5$. This results in a window size of 6. As can be seen, we wind up with the following answer



4.1.2 Question 2

For a given L, assume that packets fail with probability p. What is the probability to succeed with exactly k retries, where each retry consists of L packets?

Sol.

$$\text{First try: } p_0 = 1 - p \tag{1}$$
$$\text{First retry: } p_1 = p\left(1 - p^L\right) \tag{2}$$
$$\text{Second retry: } p_2 = p \cdot p^L\left(1 - p^L\right) \tag{3}$$
$$k \text{ retries: } p_k = p \cdot \left(p^L\right)^{k-1}\left(1 - p^L\right) \tag{4}$$
$$\tag{5}$$

4.1.3 Question 3

Assuming that $T_{pt} = T_{ack} = 0$, and the receiver is ready to accept the packet, what is the expected time to successfully transmit a packet?

Sol.

$$\text{First try: } t_0 = T_{packet} \tag{6}$$
$$\text{One retry: } t_1 = T_{packet} + (T_{out} + L \cdot T_{packet}) \tag{7}$$
$$k \text{ retries: } t_k = T_{packet} + (T_{out} + L \cdot T_{packet}) \cdot k \tag{8}$$
$$\tag{9}$$

The expected time is

$$\begin{aligned}
t_{avg} &= \sum_{k=0}^{\infty} t_k p_k \\
&= T_{packet} + \sum_{k=1}^{\infty} \left(T_{out} + L \cdot T_{packet}\right) k \cdot p_k \\
&= T_{packet} + pp\left(1-p^L\right)\left(T_{out} + L \cdot T_{packet}\right) \sum_{k=1}^{\infty} k\left(p^L\right)^{k-1} \\
\sum_{k=1}^{\infty} k\left(p^L\right)^{k-1} &= \frac{1}{1-p^L} \sum_{k=1}^{\infty} k\left(p^L\right)^{k-1}\left(1-p^L\right) \\
&= \frac{1}{\left(1-p^L\right)^2} \\
\implies t_{avg} &= T_{packet} + \frac{p\left(T_{out} + L \cdot T_{packet}\right)}{1-p^L}
\end{aligned}$$

4.1.4 Question 4

What is the expected goodput of the protocol?

Sol. Let $a = \dfrac{T_{out} + T_{packet}}{T_{packet}}$

$$\begin{aligned}
\text{Goodput} &= \frac{T_{packet}}{T_{avg}} \\
&= \frac{T_{packet}}{T_{packet} + \frac{p\left(T_{out} + L \cdot T_{packet}\right)}{1-p^L}} \\
&= \frac{\left(1-p^L\right) T_{packet}}{\left(1-p^L\right) T_{packet} + p\left(T_{out} + L \cdot T_{packet}\right)} \\
&= \frac{\left(1-p^L\right) T_{packet}}{\left(1-p^L\right) T_{packet} + p\left(T_{out} + T_{packet}\right) + p \cdot T_{packet}\left(L-1\right)} \\
&= \frac{1-p^L}{1-p^L + \frac{p\left(T_{out} + T_{packet}\right)}{T_{packet}} + p\left(L-1\right)} \\
&= \frac{1-p^L}{1-p^L + p \cdot a + p\left(L-1\right)} \\
&= \frac{T_{packet}}{T_{avg}} \\
&= \frac{1-p^L}{1-p^L + p\left(a + L - 1\right)}
\end{aligned}$$

In contrast, in GBN we have $L = 1$, therefore the goodput (under this question's assumptions) is

$$\text{Goodput}_{GBN} = \frac{1-p}{1-p+pa}$$

5 Selective Repeat

In GBN, we potentially retransmit many packets that were in fact received should a single packet fail. If packet 2 fails, but 3, 4, 5 all succeeded, we will still retransmit all of 3, 4, and 5, along with 2. Instead, we may retransmit selectively, and only resend the packets that failed. However, this comes at the cost of the receiver needing to store the window of messages, not just the transmitter. In the end, a minor cost.

Sender: Window size of N packets, which are sent one by one, in order. A timer for each packet is started after the transmission of the last bit of each packet, and a timeout for packet i indicates retransmitting packet i. $ACK\left(i\right)$ indicates reception of packet i *only*. If an ACK is received for i, mark it as received. If this is the first packet of the window, then shift the beginning of the window to the first unacknowledged packet.

Receiver: Stores a window size of N packets, and sends an ACK for every correctly received packet (even if it is not in order) in the window. Out of order packets with a sequence number *within* the window range are buffered until any missing packets with lower sequence numbers are received. When all missing packets are received, then deliver all in order packets to the upper layer and move the beginning of the window to the next expected (in order) sequence number.

We once again have the problem of number of bits / window size as we discussed earlier. For a window size N, we need $\geq 2N$ sequence numbers, i.e. $\lceil \log_2 2N \rceil$ bits.

Criteria	Selective repeat	Go back N
Bandwidth utilisation	Only retransmits lost packets	Might retransmit packets that have already successfully arrived
Window sizes	Sender: N, Receiver: N	Sender: N, Receiver: 1
Sequence numbers	$2N$	$N+1$
Out of order handling	Track which packets were received (out of order is possible)	No sorting, receiver only accepts packets in order
Receiver storage	Store packets until missing / damaged packets are retransmitted	No need to keep undamaged packets, since the sender may resend

Table 1:

5.1 Questions

5.1.1 Question 1

Assume an ideal SR protocol:

- Infinite sized windows on both ends
- Infinite bits for sequence numbers
- Infinite sized buffers on both ends
- ACKs do not fail and T_{ack} is negligible
- Packets fail with probability p

What is the protocol's goodput?

$$\text{Goodput} = \frac{T_{packet}\,(1-p) + 0 \cdot p}{T_{packet}} = 1 - p$$

5.1.2 Question 2

A sender (A) and a receiver (B) works with S&W with the following modification: station A transmits a window of L packets, starts a timer, and waits for all ACKs (for every packet in the transmitted window) before transmitting the next window.
In case any of the packets are not ACKed then the entire window is considered failed and station A retransmits the entire window (after the timeout)
If all packets are ACKed, station A will transmit the next window.

T_{prop} is given, T_{pt}, T_{ack} are negligible. What is the optimal T_{out}? (optimal in the sense of avoiding unnecessary retransmissions)

Sol. $T_{out} = 2T_{prop}$

From now, let us assume that $T_{out} = 2T_{prop}$, $L = 4$, $T_{prop} = 2$[unit]

5.1.3 Question 2

Draw the packets, and the ACKs transmission until packet 4 is successfully received, where packet 3 fails in its first transmission, and all other packets in the window are successful (got ACKed).

Sol.

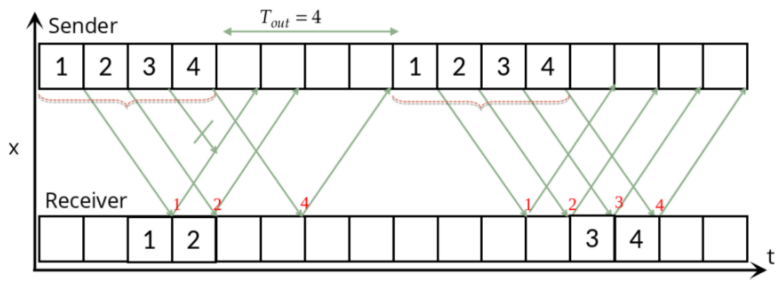


Figure 2: Solution

5.1.4 Question 3

What is the probability for an L sized window to fail?

Sol. $p' = 1 - (1-p)^L$.

5.1.5 Question 4

What is the goodput?

Sol.

$$\begin{aligned}\eta &= \frac{\mathbb{E}\left[\text{Effective sending time}\right]}{L \cdot T_{packet} + 2T_{prop}} \\ &= \frac{\Pr\left[\text{Window failed}\right] \cdot 0 + \Pr\left[\text{Window succeeded}\right] \cdot L \cdot T_{packet}}{L \cdot T_{packet} + 2T_{prop}} \\ &= \frac{L \cdot T_{packet} \left(1-p\right)^{L}}{L \cdot T_{packet} + 2T_{prop}}\end{aligned}$$