**Harry Langford**
hjel2@cam.ac.uk

## 1 2011 Paper 5 Question 5

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2011p5q5.pdf

Consider two physically-separated entities **A** and **B**. **B** has been supplied messages that will be sent to **A** following these conventions:

- **A** gets a request from the layer above to retrieve the next data ($\mathcal{D}$) message from **B**.
- **A** must send a request ($\mathcal{R}$) message to **B** on the **A**-to-**B** channel.
- Upon receipt of $\mathcal{R}$, **B** will send $\mathcal{D}$ back to **A** on the **B**-to-**A** channel.
- **A** should deliver exactly one copy of each $\mathcal{D}$ message to the layer above.
- $\mathcal{R}$ messages may be lost (but will not be corrupted) in the **A**-to-**B** channel.
- $\mathcal{D}$ messages are always delivered correctly (no loss or corruption)
- The delay alone each channel is unknown and variable.

Give the FSM describing a protocol employed by **A** and **B**.

This FSM must compensate for the loss-prone channel between **A** and **B**, as well as implementing message passing to the layer above at entity **A**. Your FSM must not use more mechanisms than necessary.

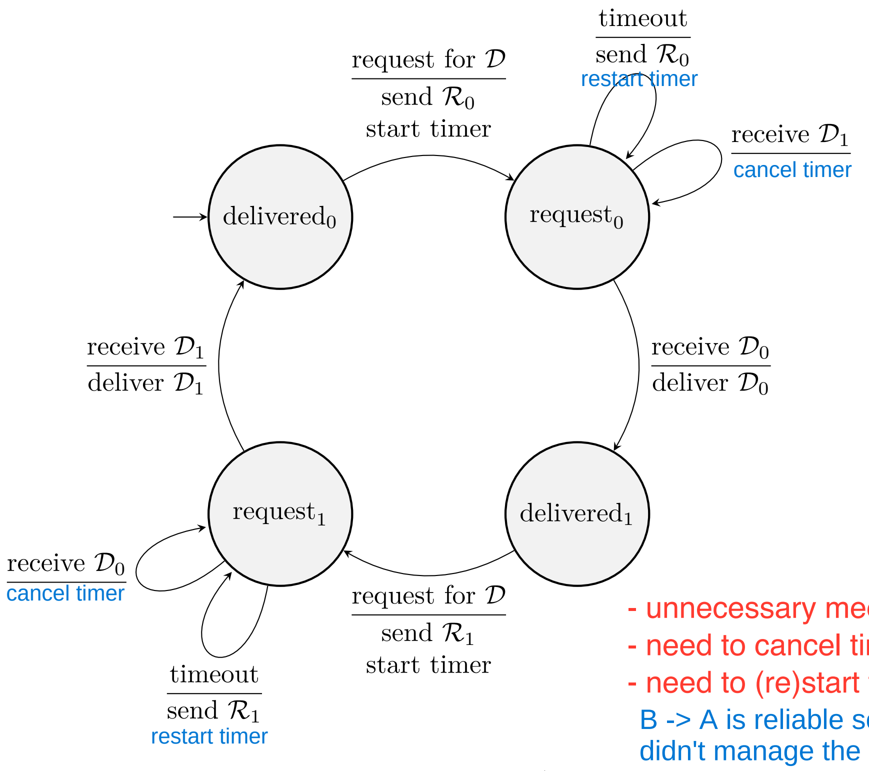

timeout / send $\mathcal{R}_0$
restart timer
request for $\mathcal{D}$ / send $\mathcal{R}_0$ start timer
receive $\mathcal{D}_1$ / cancel timer
delivered$_0$
request$_0$
receive $\mathcal{D}_1$ / deliver $\mathcal{D}_1$
receive $\mathcal{D}_0$ / deliver $\mathcal{D}_0$
request$_1$
delivered$_1$
receive $\mathcal{D}_0$ / cancel timer
request for $\mathcal{D}$ / send $\mathcal{R}_1$ start timer
timeout / send $\mathcal{R}_1$
restart timer

Figure 1: FSM for $A$

- unnecessary mechanism to number D messages
- need to cancel timer on receive expected D
- need to (re)start timer on timeout events

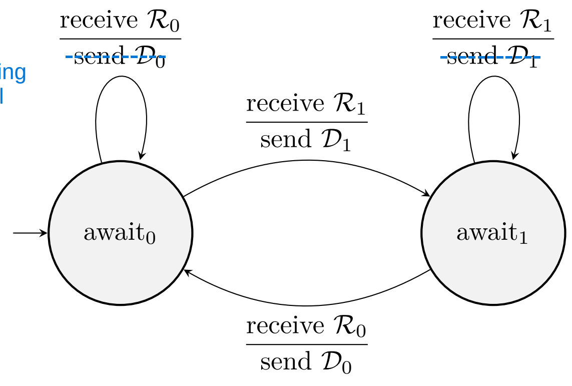
B -> A is reliable so data is always correct!
didn't manage the timer properly...

By using 1-bit signals, we assume in-order transmission!

Can solve this by enforcing in-order transmission
or have a wall-clock TTL
or by never re-using sequence numbers
(monotonically increasing variable-length sequence numbers or using a session -- multiple TCP connections)

Note "Receive R_0 / " is important! If you're ever given a stimulus, you must have a transition to respond to it!
Note if you don't explicitly ignore then the protocol terminates! i.e. a timeout without a handler kills the protocol!

You can't tell whether this is blocking or non-blocking -- this is a protocol but the implementation could be blocking or non-blocking.



Unnecessary b/w use to resend sent Ds

Since B -> A is reliable!

Figure 2: FSM for $B$

My protocol uses two features to implement this protocol:

- Sequence numbers

  Each request $\mathcal{R}$ and data $\mathcal{D}$ is sent with an associated sequence number. $\mathcal{R}_i$ demotes a request with sequence number $i$. $A$ only delivers data with the correct sequence number. Under the assumption that messages cannot be reordered, this guarantees the message is the correct one. $B$ sends the data which is requested and moves onto the next data message if and only if it receives a request for the next sequence number. ✓

- Timeouts

  $A$ implements timeouts. If it does not receive a response from $B$ within a specified time, it resends its request. Note that the length of the timeout is *not* necessary for correctness – only for efficiency. A short timeout will lead to unnecessary retransmission and waste bandwidth, while a long timeout will lead to long waiting time. ✓

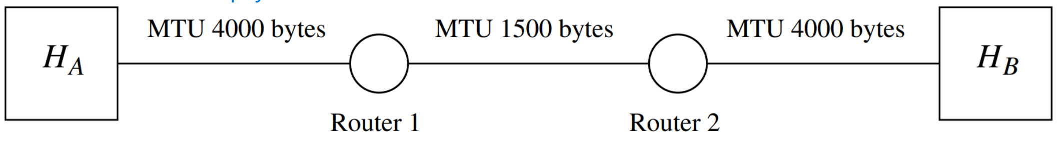
  A more efficient implementation could use the Karn/Partridge algorithm (exponential weighted average of $RTT$) to estimate $RTT$ and take the timeout to be $2 \times RTT$. However, this does not take account of the deviation in $RTT$. An even better algorithm would use the Jacobson / Karels algorithm and estimate the standard deviation (once again using an exponentially weighted average), taking the timeout to be $RTT + 4 \times \sigma$. ✓

## 2 2011 Paper 5 Question 6

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2011p5q6.pdf

(a) The diagram below shows a TCP connection between Hosts $H_A$ and $H_B$ passing through networks with different values of Maximum Transmission Unit (MTU) shown. Version 4 of the Internet Protocol (IPv4) is in use.

Whenever you expose a maximum transmission unit etc, you only want to expose the size of the payload. i.e. MTU is the maximum PAYLOAD size.
MSS is the maximum payload size.



$H_A$ chooses a TCP segment size of 3000 bytes of data (TCP and IP headers are each 20 bytes in size).

(i) Describe the way in which fragmentation takes place as $H_A$ sends data to $H_B$. Include the arithmetic used to calculate fragment sizes. Explain the saving that may

Segment means "part of a stream". Segment is the payload.

be made by $H_A$ choosing an optimal TCP segment size when sending 60KBytes of data.

Divide into
(20+) 1480
(20+) 1480
(20+) 80
(+496 padding)

I assume the header of the IP packet is *not* included in the TCP segment size, but the TCP header is included in the size of the TCP segment. So the IP packet sent from $H_A$ will be 3020 bytes large. TCP Segment is the payload...

ofc not! TCP is the payload inside IP.

REMEMBER: offsets are in 8 bytes!

In IPv4, packets are fragmented at intermediate nodes if the MTU for the next link is smaller than the packet size. When the packet arrives at router 1, it is larger than the MTU for the next link on the route. So the router will fragment the packet. Each will have a fragment offset. In this case, the packet will be fragmented into three smaller packets: the first two packets will contain the TCP header, an IP header and have 1460 bytes of payload. The final packet will contain the TCP header, an IP header and 1060 bytes of payload.

MTF "more to follow" means "there are more packets" -- think of it as a boolean "there are more packets"

In IPv4, reconstruction is only done at the destination. IPv4 packets use a fragment offset rather than a fragment number since packets can be fragmented at multiple nodes. Note that a layering violation is required to ensure that each packet contains enough information to reconstruct it at the destination.

? not true
IP identifier (header field)

fragment payloads can only be multiples of 8! i.e. not 3980 -- 3976

The downside of en-route fragmentation is that if even one of the fragments is lost, then the TCP segment cannot be delivered. Hence the sender will have to retransmit the entire TCP segment. This wastes network bandwidth and increases latency.

The optimal segment size is the MTU for the link with the least capacity. So the optimal segment size is 1480 bytes (such that with the IP header, the IP packet will be the MTU). 1460*

not true. depends on loss probability.

Sensible nums include: MSS=2940, 1460, etc

Fragmentation increases payload-to-header ratio... but more fragments decreases the prob of all getting through.

(ii) Briefly explain how the situation described in part (i) would be handled if Internet Protocol version 6 (IPv6) were used.

IPv6 does not do any en-route fragmentation. If the case above was using IPv6, the segments would be dropped at Router 1 and an ICMP error message would be sent to $H_A$.

ICMP6, a completely different protocol!

(b) The formulae below are used in TCP implementations to compute a value for the retransmission time-out ($R$). $R$ is an estimate of the round-trip time, $M$ is the most recently measured round-trip measurement, $\alpha = 0.875$ and $h = 0.25$.

Loss rate on the internet is about 5%

$$D \leftarrow D + h(|M - R| - D)$$
$$R \leftarrow \alpha R + (a - \alpha)M$$
$$\mathcal{R} = R + 4D$$

you double \mathcal R. then use real measurements. You may have to do many types of binary exponential backoff.

(i) How is $M$ measured? TCP does binary exponential backoff.

timestamp in options or ignore anything you retransmitted.

Read the clock when the message is sent, and read it when the response arrives from $H_B$. However, if we have to request a retransmission then we should not measure $M$ – since it's impossible to tell whether the response was for the original message we sent or for the retransmission.

and multiple ACKs? and what if no usable measurements?

only use the RTT if you get multiple ACKs.

(ii) Explain the principles behind the design of these formulae and the values $h$, $\alpha$ and $D$.

This is the Jacobson/Karels algorithm.

$D$ is the deviation of the round-trip time. $h$ is a constant used in the exponential weighted average of $D$ and $\alpha$ is a constant used in the exponential weighted average of the round-trip time and $\mathcal{R}$ is the timeout.

We assume that the mean and variance of the round trip time are not known and are variable. However, we can estimate both of them using an exponential-weighted average. Whatever distribution the latency of correctly delivered packets is drawn from, the probability of a packet being delayed by more than 4 about 5%

deviations is very small. So if the latency reaches this, we resent the request. This takes into account both the latency and the round trip time and so generalises well to many different types of network under many different loads.

Large timeouts occur when either the mean of the $RTT$ is very large, or the variance is very large. Either of these imply that a large delay would not be unlikely. Small timeouts occur if and only if both the mean $RTT$ and the deviation is very low.

We use $\alpha = 0.875$ because this is $\frac{7}{8}$. It can be easily calculated by bit-shifts. how? Similar reasoning for why $h = 0.25$ – it can be calculated in a single bit-shift. Since all of these calculations are carried out many times, we want the arithmetic to be as simple as possible.

Note that the expectation of $R$ is given by $\frac{a-\alpha}{1-\alpha}RTT$. Therefore, $a$ must be equal to 1 to ensure that the expectation of the estimate for the round-trip time is the round trip time.

Questions?
If client A has a connection with server B.
A sends FIN
B sends ACK
aka the connection is half-closed.
Which direction of communication did that close?
i.e. did FIN signal "I'm done" or "stop sending me messages"

If I send FIN it means "I won't send any more data"

HTTP has no way to stop transmission. HTTP is transactional. You request something and it's done.
Some protocols i.e. FTP allow you to request parts of the data.
In HTTP you've got to do it in the application itself.

How do we reassmble IPv4 fragments if the TCP window is greater than 2^16 * 576 bytes?
i.e. using TCP window scaling... or is this never done? or only done over IPv6
just don't use IPv4 if you're sending data over a fast network
IP simply can't keep up... just don't do it
TCP can go faster than IP can support

Selective acknowledgement: only acknowledge some of the packets.

You can use this with cumulative acknowledgements and it means you don't have to sent an acknowledgement for every packet.
Selective acknowledgement with cumulative acknowledgement can mean "acknowledge every k packets"
i.e. acknowledge packet 0, packet 10, packet 20, packet 30 etc

you used to be able to tell when you sent a packet out. So you can put the timeout in the packet options field and tell the receiver to include it in the ACK. So you get the RTT for all acknowledgements you get. Problem solved!

problem: if the network gets re-routed, queuing increases... so timeout incereases... so you're less likely to hit slow-start and you're going to continue using more data than you need to.

modern TCP (TCP Reno) stops exponential slow-start at ~half the point at which it timed out and goes into AIMD.

Note that the distribution of packets is not normal. It's not even CLOSE to normal!
It's got a very large positive skew.