Embedding it better...

µTasker Document

µTasker – IPv4 Fragmentation

Table of Contents

1. Introduction

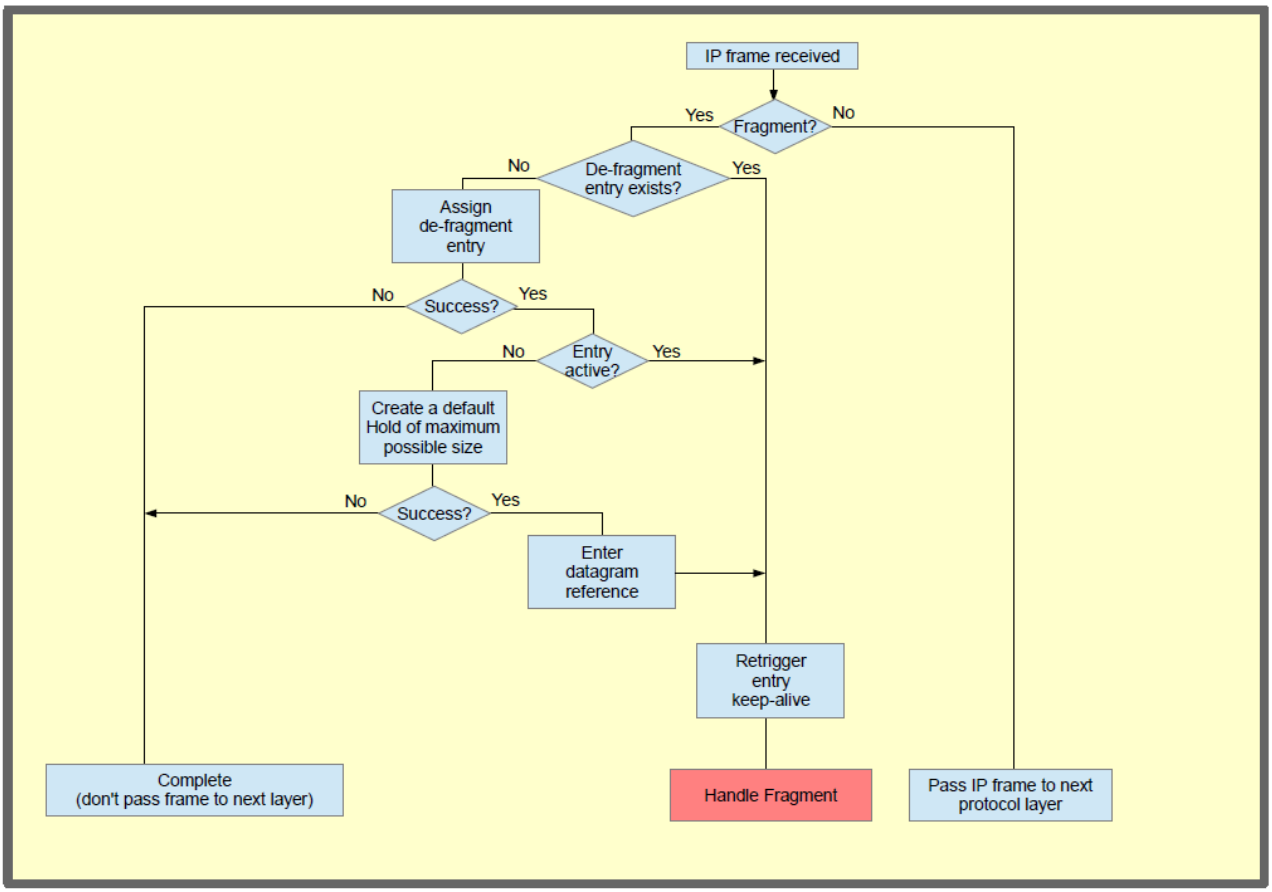
This document discusses the IP fragmentation/de-fragmentation solution optionally enabled in the IP layer of the µTasker TCP/IP stack.

RFC: 815, by David D. Clark (MIT Laboratory for Computer Science Computer Systems and Communications Group, July, 1982), describes an IP datagram reassembly algorithm that is closely followed for reassembly.

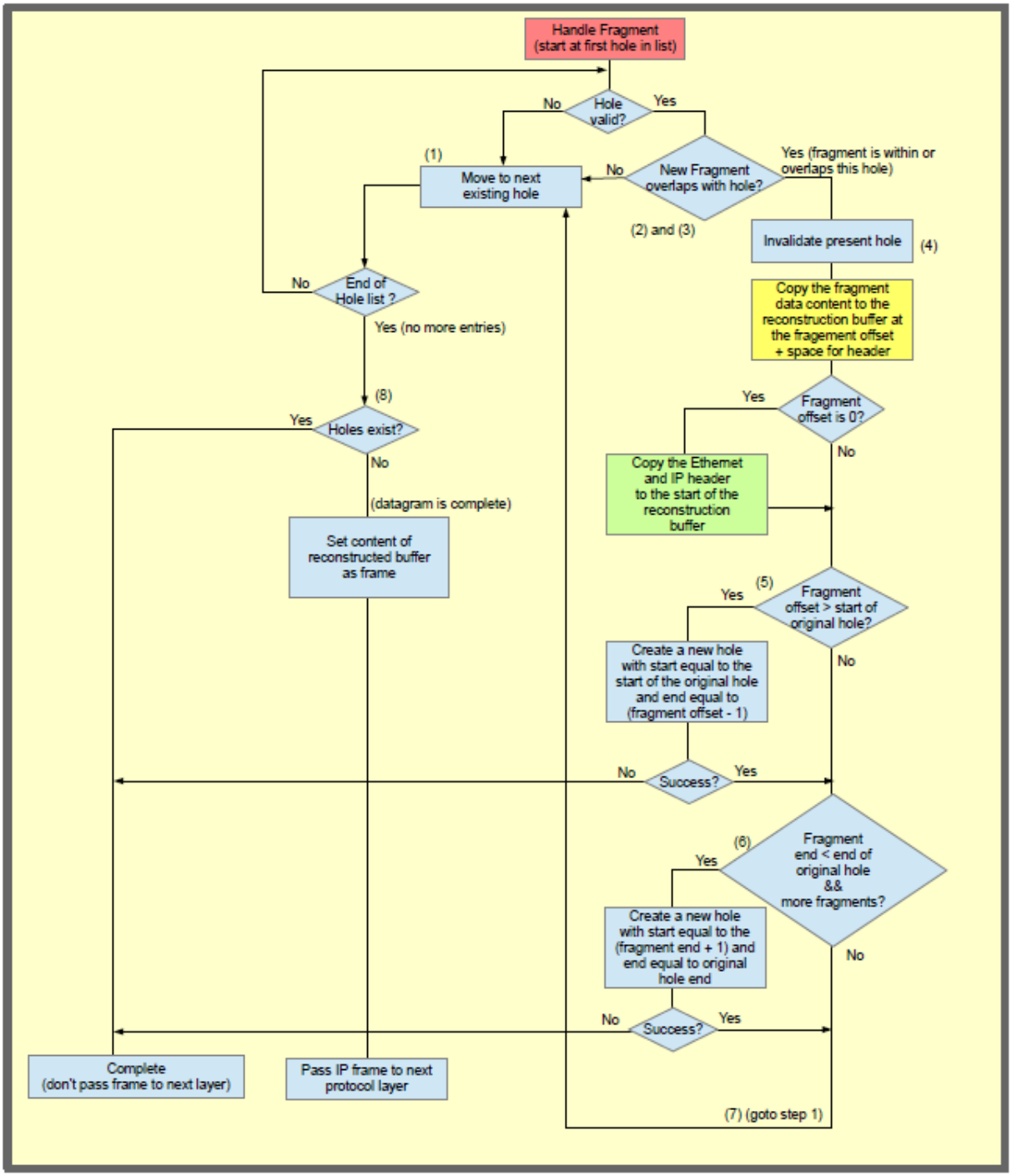
However, the memory management is based on a reconstruction data pool of a specified size (which limits the maximum size of IP content that can be received without having to drop the frame) and can be shared for reconstruction of multiple fragmented IP datagrams, whereby handling multiple such in parallel can decrease the maximum size possible for each datagram.

2. De-Fragmentation Software Flow Diagram

This is the software flow diagram showing how the recommended algorithm is integrated in the µTasker IP reception routine:



The numbers (1) to (8) represent the steps in the original recommendation.



Further implementation details that are not defined by the recommended algorithm:

What happens when only a part of the datagram is received, for example when one frame is lost, or the transmitting side is powered down during transmission?
In this case there may be one or more fragments collected at the receiver which will be utilising a datagram management entry. This entry could block subsequent datagrams, including a repetition of the lost datagram, if it were used to carry a protocol with retries.
To avoid an incomplete datagram from blocking subsequent operations, each management entry has a time to live entry that is decremented each time there is any IP activity. This will ensure that further usage of an entry can't indefinitely block and saves requiring a timer for its

operation. *If time based decay is required it could be called from another periodic task, such as the watchdog task.*

What about IP options?
Space for up to 40 IP options are reserved at the start of the reconstruction buffer. Since IP options are rare this space is usually not used, but the de-fragmentation could support maximum options in the IP frame if ever needed.

Identifying frames – the algorithm requires a match of the frame identification, protocol ID, foreign IP and local IP addresses.

What about fragments that are received multiple times or out of order?
The recommended algorithm should automatically handle these cases.

3. Project Defines

```
#define IPV4_SUPPORT_TX_FRAGMENTATION // support sending IPv4 fragments
                                      (for datagrams exceeding the Ethernet MTU)
#define IPV4_SUPPORT_RX_DEFRAGMENTATION // support receiving and reassembling IPv4 fragments
#define DEFRAG_MEMORY_POOL_SIZE (4 * 1024) // total de-fragmentation datagram memory to be shared
                                           by multiple datagrams if they occur at the same time
#define MAX_DEFRAG_DATAGRAMS 2       // the number of datagrams can be reassembled in parallel
                                     (sharing the de-fragmentation memory) - 1 or 2 presently possible
#define MAX_DEFRAG_HOLES     8       // the maximum number of holes that can be managed for each
                                        de-fragmented datagram
```

3.1 *Receiving Multiple Fragmented IP Datagrams at the same Time*

If `MAX_DEFRAG_DATAGRAMS` is set to 1 it is never possible to receive more than one fragmented IP datagram at the same time since the data pool and management entries are dedicated to the first de-fragmentation process that occurs and are locked by this until it has completed, or timed out.

If `MAX_DEFRAG_DATAGRAMS` is set to a larger number this many of de-fragmentation processes can take place at the same time. The data reconstruction memory (data pool size defined by `DEFRAG_MEMORY_POOL_SIZE`) is allocated in full to the first de-fragmentation process that starts so that it could reconstruct a frame up to that size but will be shared by further processes that start if possible. This means that the second datagram reconstruction process will be allocated half of the pool in case this is still free, or else it is allocated whatever remains free. Since the pool size originally allocated to the first reconstruction process has to be reduced it may stop it reconstructing large frames, which would have been possible without the second process starting before the first is complete.

4. Oversize IP Datagrams

Often fragmentation results when there is a hop in the connection routing path that is smaller that the MTU of Ethernet. In this case the resulting frame, after reconstruction, will not be larger than the original Ethernet MTU and the reconstruction data pool doesn't need to be larger than this for a single reception.

The following example shows an over-sized IP datagram, which can be easily tested with the ping utility.

In the case of over-sized datagrams, the complete IP datagram is larger than the Ethernet MTU and so the pool size also needs to be adequate in order to receive the frame.

When responding to over-sized pings it is not possible to send the response without also fragmenting the IP datagram. This is supported when `IPV4_SUPPORT_TX_FRAGMENTATION` is enabled but doesn't require additional buffering memory because IP simply sends the data content in multiple IP frames.

Example of a ping test with length exceeding the Ethernet MTU of 1500 bytes:
"`ping -l 3500 192.168.0.10`" sends a ping with 3500 byte ICMP payload that is expected to be echoed back. It is broken into three IP fragments:

First frame (example):
IP identification = 0x338b
Fragment offset = 0x2000 (b1110000000000000 are flags, where 0x20000 is the "More Fragments" flag)
IP protocol = 0x01 (ICMP)
Payload (IP payload) length = total frame length – Ethernet II and IP header = 1514 – 14 – 20 = 1480

The fragment offset bit 0x2000 means that there are more fragment belonging to this IP frame and so triggers the IP de-fragmentation process. This means that the payload needs to be saved until the complete content has been received, after which it can be passed on up to the next protocol layer.
The IP identification is also used to identify which subsequent IP frames belong to the same overall frame.

The second frame has also the same IP identification so it is known that its payload needs to be concatenated with others with matching IDs.
The fragment offset = 0x20b9 indicating that there are still more fragments belonging to the IP complete frame. 0xb9 = 185 and 185 x 8 = 1480, meaning that the location of the payload in this individual packet belongs at the offset 1480 from the start of the overall frame.
The IP payload size is again 1480.

The third and final frame has the same IP identification, a fragment offset value of 0x0172 (equal to 370 x 8 = 2960) with the more-fragments flag set to 0, meaning that it is the final packet in the overall IP frame. The payload is 0x224 (548 bytes).
Also a non-zero fragment offset triggers the de-fragmentation process.

If these three payloads (1480 + 1480 + 548 = 3508 (including the 8 byte ICMP header) are joined together the full ICMP can then be delivered to the ICMP protocol layer.

5. Implications of Checksum Offloading

Modern Ethernet controllers often include support for IP and payload checksum offloading. This means that the Ethernet controller hardware performs calculation of checksums over certain IP hearers and certain payload that is carried by IP datagrams so that it can recognise reception errors without the IP stack needing to check this in software. In the opposite direction it can insert checksum values into the frames that the stack passes to it without the IP stack needing to calculate the values itself. The result is uselful savings in processor power/time that is otherwise needed for the IP checksum execution in code.

IP fragments however pose a difficulty to such checksum offloading engines due to the fact that the payload is distributed in multiple datagrams. When the Ethernet controller transmits single frames containing a complete IP datagram it can recognise what type of payload is being sent (eg. ICMP in the case of a ping response), calculate the ICMP checksum field across the rest of the payload and insert the resulting value at the corresponding entry. If either a fragment offset is specified or the “more fragments” flag is set it can no longer calculate the payload checksum based on on the single frame content that it has been given. In this case the ICMP routine in the IP stack has usually no option but to perform the calculation itself and insert the calculated checksum value into the relevant field, whereby it would usually not have had to to do this and typically leaves the filed at 0x0000 so that the Ethernet hardware can set it appropriately. The second implication is that the Ethernet checksum offloading operation on the payload may need to be selectively disabled for each fragment that is sent so that the hardware doesn't incorrectly overwrite the value.

The µTasker Kinetis Ethernet driver disables the payload insertion on a frame basis to avoid the engine from overwriting the checksum value that was inserted by the stack code.

The reception of fragments represents a similar case where the Ethernet hardware cannot perform a complete checksum calculation of IP payload. It can however offer some assistance to the software. Although the complete payload is not available in a single frame the engine can still calculate the checksum result of the payload in the individual frame. This result can be made available to the handling software so that it knows the value for the content. If the same is done for all frames it is faster for the stack to then use the results of each individual packet's calculation to work out the overall value than to perform the software calculation over the full payload. Depending on the method used to perform the calculation for a specific protocol there may still be a little extra post-processing involved, such as to handle the pseudo header that UDP requires, but this is also required during the software method too.

6. Conclusion

Fragmentation should generally never be found together with TCP since TCP actively avoids it. Also, since MTUs smaller that the Ethernet MTU are uncommon, it is not very likely that fragmentation is encountered in modern networks. Theoretically, routers are free to fragment as long as the don't fragment flag is not set, but it is unlikely that this is commonly performed since fragmentation is also a load for routers, which probably aim at being as efficient as possible and so presumably also avoid fragmentation unless it were for a very important reason.

Enabling de-fragmentation and fragmentation in the µTasker project doesn't cause any noticeable effect on the performance of non-fragmented IP traffic, so enabling a small reconstruction data pool in case such datagrams are experienced represents a low overhead to be sure that they could be handled if ever required.

The loss of hardware checksum offloading possibilities for fragmented payloads will tend to represent a loss in efficiency of the handling of such frames when hardware support is otherwise available for non-fragmented datagrams.

Modifications:

V0.01 04.1.2016: First draft

V1.00 11.1.2016 Added hardware checksum offloading implications