(a) Algorithm Description

1. It extracts the SET command from the XDP payload.
2. SET command founded, try to figure the index of the Memcached key in payload.
3. If the key is found, process the key with hash function.
4. If the hash matches the calculated hash, the corresponding cache entry in the map should be invalidated

(b) REX Implementation

```
1  let set_iter = payload
2   .windows(4) // 4 chars as a slice
3   .enumerate()
4   .filter_map(|(i, v)|
5      if v == b"set " { Some(i) } else { None }
6  ); // found the SET command
7  for index in set_iter {
8    ... // set payload index via SET command
9    payload
10    .iter()
11    .take_while(|&&c| c != b' ')
12    .for_each(|&c| {
13      ... // process the key with hash func
14    });
15  ... // invalidate Memcached cache entry
16}     // if the hash matches
```

- REX utilizes the lazily evaluated `windows`, `enumerate`, and `filter_map` from slice iterators, which split the whole payload (**L1**) into chunks and collects the chunks equals the SET commands into iterators (**L7**) .
- In the original implementation, four levels of nesting (**L4,20,25,26**) is also significantly reduced by converting a `for-loop` (**L2**) with intricate conditions (**L4**) into a clean chain of higher-order functions with closures through the `take_while` (**L11**), which will filter the Memcached SET key (**L5**) from the payload with the iterator generated by `filter_map` (**L4**).

(c) Original BMC Implementation

```
1  #pragma clang loop unroll(disable)
2  for (unsigned int off = 0; off < BMC_MAX_PACKET_LENGTH &&
4   payload + off + 1 <= data_end; off++) {
5   if (set_found == 0 && payload[off] == 's' &&
6      payload + off + 3 <= data_end &&
7      payload[off + 1] == 'e' && payload[off + 2] == 't') {
9     ... // move offset after the SET command
10    set_found = 1;
11  } else if (key_found == 0 && set_found == 1 &&
12              && payload[off] != ' ') {
13    if (payload[off] == '\r') {
14      set_found = 0; key_found = 0;
16    } else {
17      ... // found the start of the key
18      key_found = 1;
19    }
20  } else if (key_found == 1) {
21    if (payload[off] == ' ') {
22      ... // found the end of the key
23      set_found = 0; key_found = 0;
25    } else {
26      if (...) {...} // process the key with hash func
27    }       // invalidate Memcached cache entry
28  }         // if the hash matches
29}
```