Peephole Optimizations

1 Direct Fast Forward

Pattern: `CreateInst`→`InsertSeq`→`Assemble`

Analysis: `CreateInst` creates a new instance. As such, the sequence must be empty and `InsertSeq` will insert the first block, which will be directly ejected by `Assemble`. To this end, the insertion is redundant and the assembled data is identical to the payload of this packet. To this end, the insertion and assemble instructions can be removed, and all the reference to p.`sdu` in this code block can be replaced with p.`payload`.

Output: `CreateInst`, and references to p.`sdu` in this code block can be replaced with p.`payload`, *e.g.*, `NextLayer(SDU)` should be modified to `NextLayer(Payload)`.

2 Fast Forward

Pattern: `InsertSeq`→`Assemble`

Analysis: Fast Forward optimizes the assemble operations for existing instances. To be specific, we could make a fast peek to the sequence before we insert the block: if the sequence is empty and the block's meta is aligned with the sequence's window, this block can be fast forwarded, *i.e.*, passing the payload instead of SDU; otherwise the code maintains the same.

Output: `If(IsEmpty&IsAlign)`→(replace SDU with payload) →`Else`→`InsertSeq`→`Assemble`

3 Fast Assemble

Pattern: `InsertSeq`→`Assemble` and without any `NextLayer` and `Callback`

Analysis: The `false` branch of the Fast Forward optimization means the current sequence is not empty or the current block is not aligned with the window. We can further optimize this branch, if it has no external function call, *i.e.*, `NextLayer` and `Callback`. Specifically, if the packet sequence is implemented using linked list like `libnids`, `Assemble` that collects the continuous blocks will invoke several times of data copy. However, if there is no external function that needs the assembled data, such copy is useless and can be eliminated. On the other hand, if the packet sequence is implemented using ring buffer like mOS, the data copy is still necessary when there are holes in the sequence and memory compaction is performed. In such cases, `Assemble` instruction can be eliminated. Note that we cannot eliminate `InsertSeq`, because this block may be useful for next packets' assemble in other branches.

Output: Remove `Assemble`.

4 Fast Destroy

Pattern: `CreateInst`→`DestroyInst`

Analysis: If an instance is created and destroyed by the same packet, it means that such instance will not impact any permanent data, and all sequence and PSM operations are meaningless. As a result, we can eliminate such creation and deletion as well as most of the instructions between them, except `Callback` and `NextLayer`.

Output: A mostly empty instruction block except `Callback` and `NextLayer`.