0.1. algorithm

Algorithm 1 Convert text instructions to vectors

input: *command*
output: *vector*

```
1: function VECTORIZE(command)
2:     items ← command.split()
3:     result ← list()
4:     for each word ∈ items do
5:         result.append(word)
6:     end for
7:     return result
8: end function
```

Algorithm 2 Build Dictionary

input: *commands*
output: *result*

```
1: function BUILDDICT(commands)
2:     result ← {}
3:     maxIdVal ← 0
4:     for each command ∈ commands do
5:         for each word ∈ command do
6:             if !word ∈ result then
7:                 maxIdVal ← maxIdVal + 1
8:                 result[word] ← maxIdVal
9:             end if
10:        end for
11:    end for
12:    return result
13: end function
```

0.2. Tcp session

Algorithm 3 Convert text instructions to vectors

input: *command*
output: *vector*

```
1: function VECTORIZE(command)
2:     items ← command.split()
3:     result ← list()
4:     for each word ∈ items do
5:         result.append(word)
6:     end for
7:     return result
8: end function
```

Algorithm 4 relative n-gram distance

input: $bound, trainVectors, testVectors$
output: *resList*

```
1: function CALCULATERES(bound, trainVectors, testVectors)
2:     resList ← list()
3:     normalPred ← []
4:     for eachtestVector ∈ testVectors do
5:         distances ← []
6:         for each trainVec ∈ trainVectors do
7:             distances.append(nGramDistance(trainVec, testVector))
8:         end for
9:         minDis ← min(distances)
10:        predRes ← 0
11:        if minDis ≥ bound then
12:            predRes ← 1
13:        end if
14:        resList.append(predRes)
15:    end for
16:    return resList
17: end function
```

Algorithm 5 aggregate packet by src and dst

input: *packets*
output: *groups*

```
function aggregatePacket(packets)
    groups ← dict()
    for each packet ∈ packets do
        if src ≥ dst then
            key ← src + sport + dst + dport
        else
            key ← dst + dport + src + sport
        end if
        if key ∈ groups then
            groups[key].append(packet)
        else
            groups[key] = list()
            groups[key].append(packet)
        end if
    end for
    for each key ∈ groups do
        groups[key].sort()
    end for
    return groups
end function
```