Wash Trading Detection challenge

Ildar Khabbutdinov

4/23/2022

Contents

Introduction

This report is to propose an approach to detect possible wash trading activity given limited data, namely trade order ids, trade time, trade id, price, quantity. That is no account ids are available that perform trades. Also, no information is available about placed order ids before the trade is made.

The approach is based on assumption that wash traders don't want to make deals with other market players, otherwise they would lose money and have to employ a trading strategy, in which case they would become regular market players.

Assuming that wash traders place orders in such a way that their orders should match each other, they have to be aligned in time to be executed. Price is important as well, it shouldn't match existing orders, but time is crucial for their orders to go first.

Given this condition, a stream of trades (where we see trade order ids, trade time, price, quantity) would show transactions of our interest as trades having sequential order ids and tight in time.

A 20 hours long stream of trades of arbitrarily chosen pair ALPINEBTC from Binance exchange will be used as an example.

Data exploration and pre-processing

ALPINEBTC stream can been collected with the following python script:

```
import websocket
import json
from datetime import datetime, timedelta


def on_message(wsapp, message):
    current_datetime = datetime.now()
    if current_datetime < (start_date + timedelta(days=1)):
        arr.append(json.loads(message))
        print(f'{current_datetime}, {len(arr)}, {message}')
    else:
        print(json.dumps(arr), file=f)
        exit()


f = open('ALPINEBTC.json', 'w')
arr = []
start_date = datetime.now()

#websocket.enableTrace(True)
wsapp = websocket.WebSocketApp("wss://stream.binance.com:9443/ws/alpinebtc@trade",
                               on_message=on_message)
wsapp.run_forever()
```

The example stream can be downloaded as follows (all the rest are R scripts).

```
df <- fromJSON('https://raw.githubusercontent.com/ildkhav/Wash-Trading-Detection/main/ALPINEBTC.json')
as_tibble(df)
```

```
## # A tibble: 4,993 x 11
##    e            E s            t p      q           b      a       T m     M
##    <chr>    <dbl> <chr>    <int> <chr>  <chr>   <int>  <int>   <dbl> <lgl> <lgl>
```

```
##  1 trade  1.65e12 ALPIN~ 1264398 0.000~ 0.720~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  2 trade  1.65e12 ALPIN~ 1264399 0.000~ 0.810~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  3 trade  1.65e12 ALPIN~ 1264400 0.000~ 1.530~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  4 trade  1.65e12 ALPIN~ 1264401 0.000~ 1.530~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  5 trade  1.65e12 ALPIN~ 1264402 0.000~ 1.530~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  6 trade  1.65e12 ALPIN~ 1264403 0.000~ 1.530~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  7 trade  1.65e12 ALPIN~ 1264404 0.000~ 1.530~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  8 trade  1.65e12 ALPIN~ 1264405 0.000~ 1.490~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
##  9 trade  1.65e12 ALPIN~ 1264406 0.000~ 3.780~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
## 10 trade  1.65e12 ALPIN~ 1264407 0.000~ 10.69~ 2.50e7 2.50e7 1.65e12 TRUE  TRUE
## # ... with 4,983 more rows
```

This is a 4993x11 data frame, columns of our interest are:

- b - buyer order id
- a - seller order id
- q - quantity
- p - price
- T - trade time

See Binance Trade Streams for reference.

Time range is between

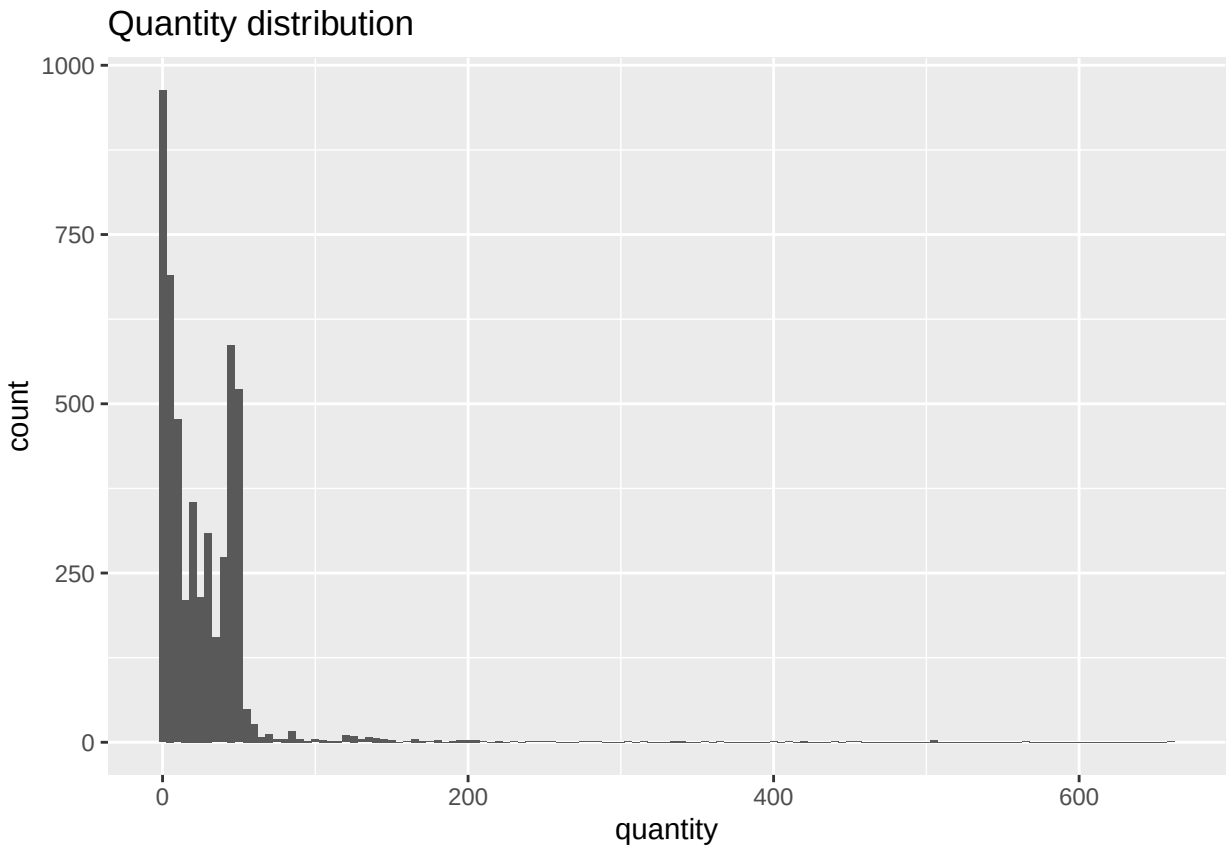
```
## [1] "2022-04-13 19:20:25 UTC"
```

and

```
## [1] "2022-04-14 15:57:07 UTC"
```

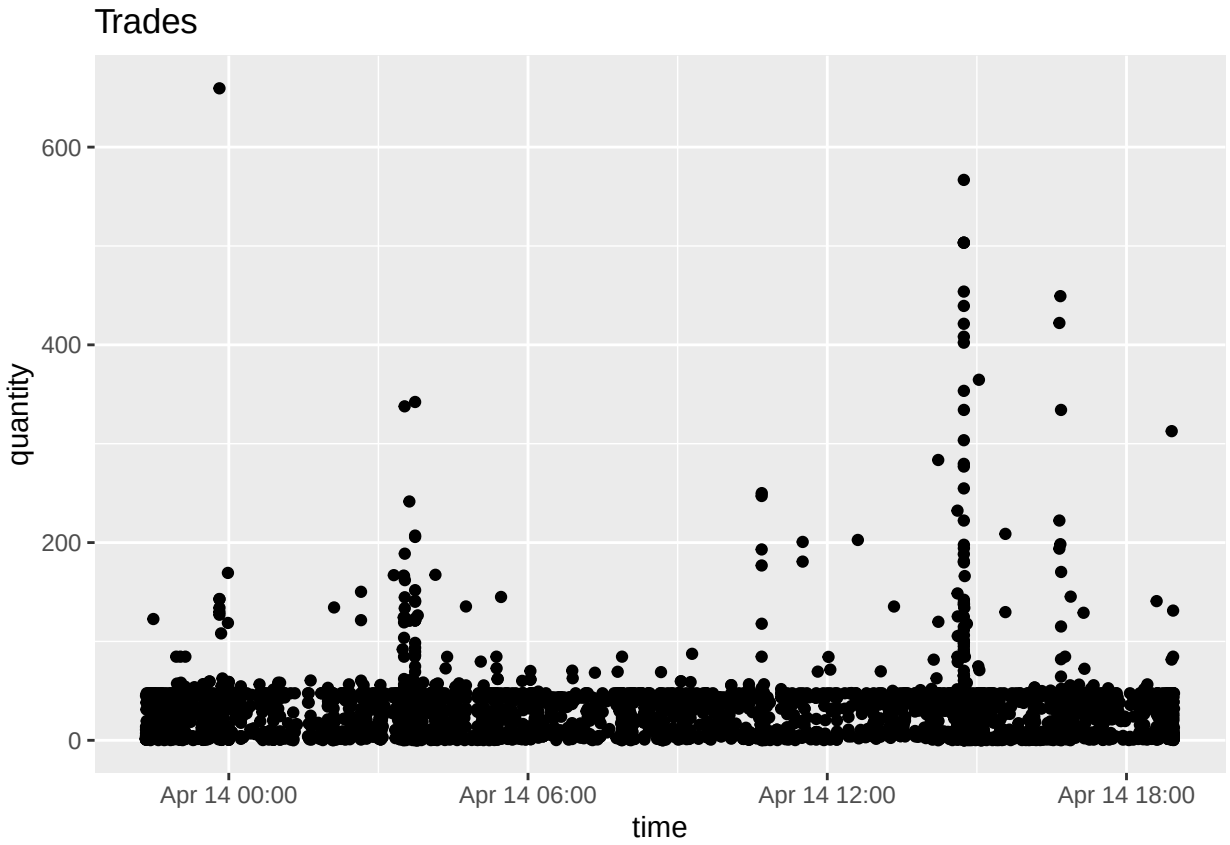
Let's check quantity distribution, trades from quantity perspective and trade time gap histogram to get an idea of the trade picture concerning volumes. And need to make p and q numeric.

```
df$p <- as.double(df$p)
df$q <- as.double(df$q)
```

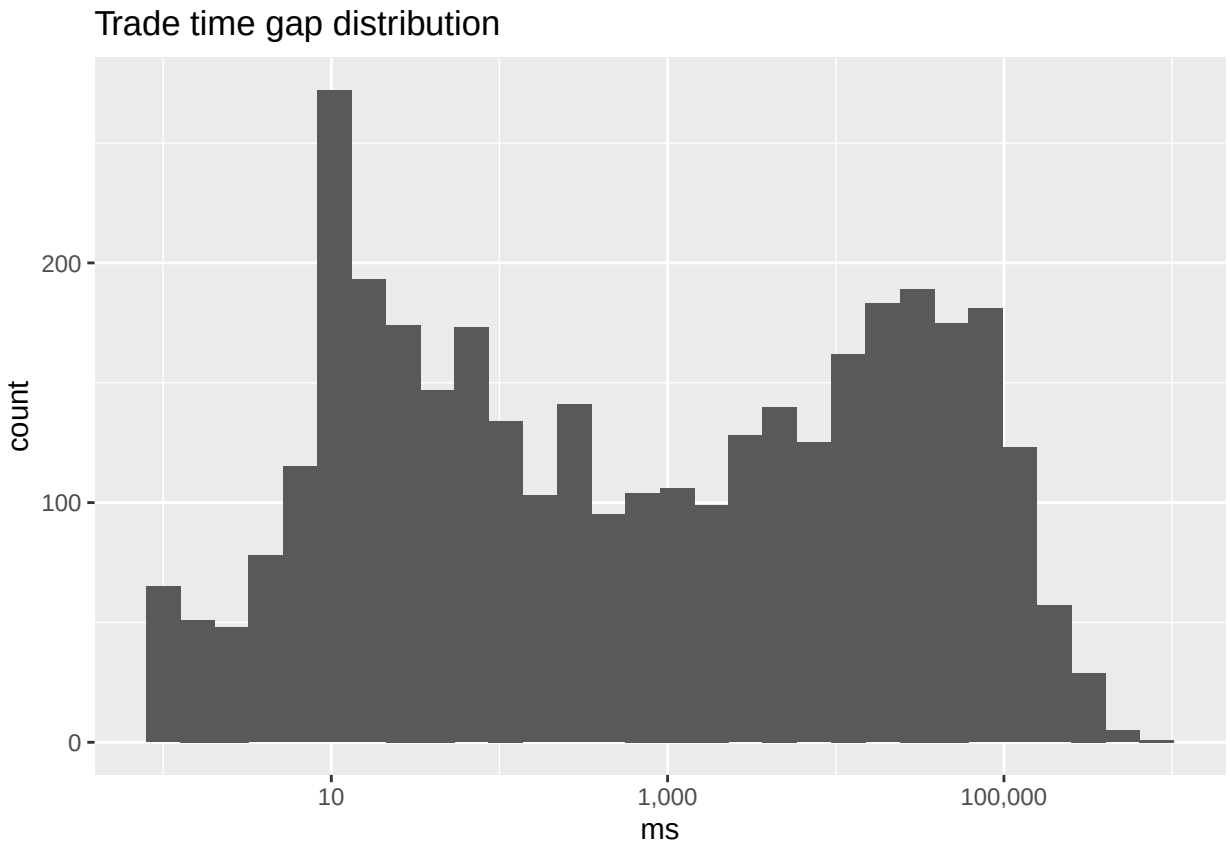


As it seen, mostly there were quantities less than 50 with spikes around 50 and 10. A sum of all:

```
## [1] 131747
```



When it comes to timing there are sharp high quantity spikes.



Time gap between subsequent trades has salient ranges. There are 10-100 and 10,000-100,000 ms groups.

Next, let's filter out market maker trades (assuming they are legit). Also we drop NA value rows if any.

```
orig_df <- df
df <- df %>%
  filter(df$m == FALSE)
nrow(df)
```

```
## [1] 2714
```

```
df <- na.omit(df)
nrow(df)
```

```
## [1] 2714
```

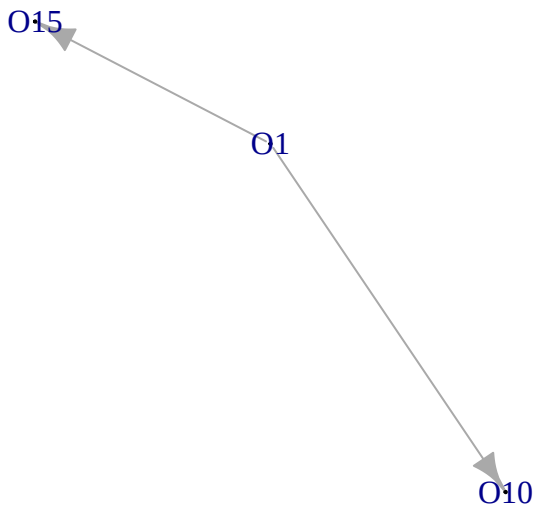
nrow shows no rows has been dropped.

Analysis

A method to solve the task - to find trades that have sequential orders, uses graph.

We will build a graph where vertices will be trade order ids and the trade relation will be the edge.

For example, each trade record has seller order id (a column in our example) and buyer order id (b). They will be vertices with an arrow pointing from seller to buyer. Trade records like [O1, O10], [O1, O15] would constitute 3 vertices graph and so on, for instance



This way we will group all orders that matched and were executed. They will be shown as connected graphs, it will be clusters in the whole graph of all trades in the stream.

Then among these clusters those will be filtered that have sequential vertix values (orders). Among them, the less the time gap between trades in the cluster (given there are more than one trade) the more suspicious it may be.

Let's build the Graph *g*, find its components and get groups of it. Components *c* and Groups *grps* are just intermediate objects that will help. Components stores membership of every vertix, Groups provides list of vertices of the given membership. Preserving trade id *t* here in order to use join operations later.

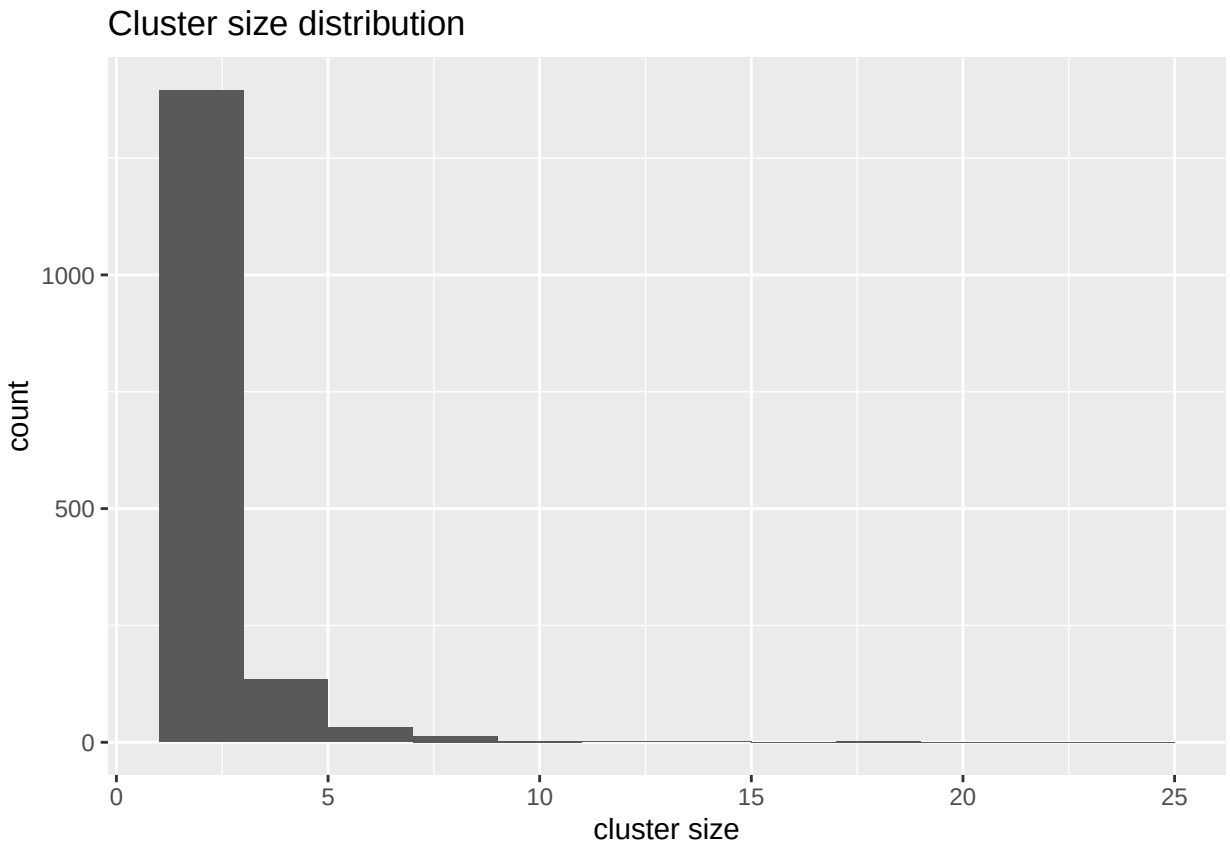
```
g <- df %>%
  select(a, b, t) %>%
  graph_from_data_frame()

c <- components(g, mode = "weak")

grps <- igraph::groups(c)
```

We can check by the way how many components

```
c$no
```

```
## [1] 1588
```

and see their distribution by number of members.



Here we get a list of memberships, members of which have sequential vertices (orders ids).

```
l <- lapply(grps, function(i) {
  if(all(abs(diff(sort(unique(as.numeric(i))))) == 1)) {
    return(c[['membership']][[i[1]]])
  } else {
    return(NA)
  }
})
```

Then we create a data frame off the graph and using Components object assign membership.

```
xdf <- igraph::as_data_frame(g)
as_tibble(xdf)
```

```
## # A tibble: 2,714 x 3
##    from     to             t
##    <chr>    <chr>      <int>
##  1 25029978 25030062 1264423
##  2 25029957 25030064 1264424
##  3 25030066 25030067 1264425
##  4 25030106 25030124 1264426
##  5 25030105 25030127 1264427
##  6 25030113 25030130 1264428
##  7 25030095 25030132 1264429
##  8 25030103 25030136 1264430
##  9 25030352 25030416 1264433
## 10 25030364 25030417 1264434
```

```
## # ... with 2,704 more rows
```

```
msh <- sapply(xdf$from, function(x) {
  c[["membership"]][[x]]
})
xdf$membership <- msh
as_tibble(xdf)
```

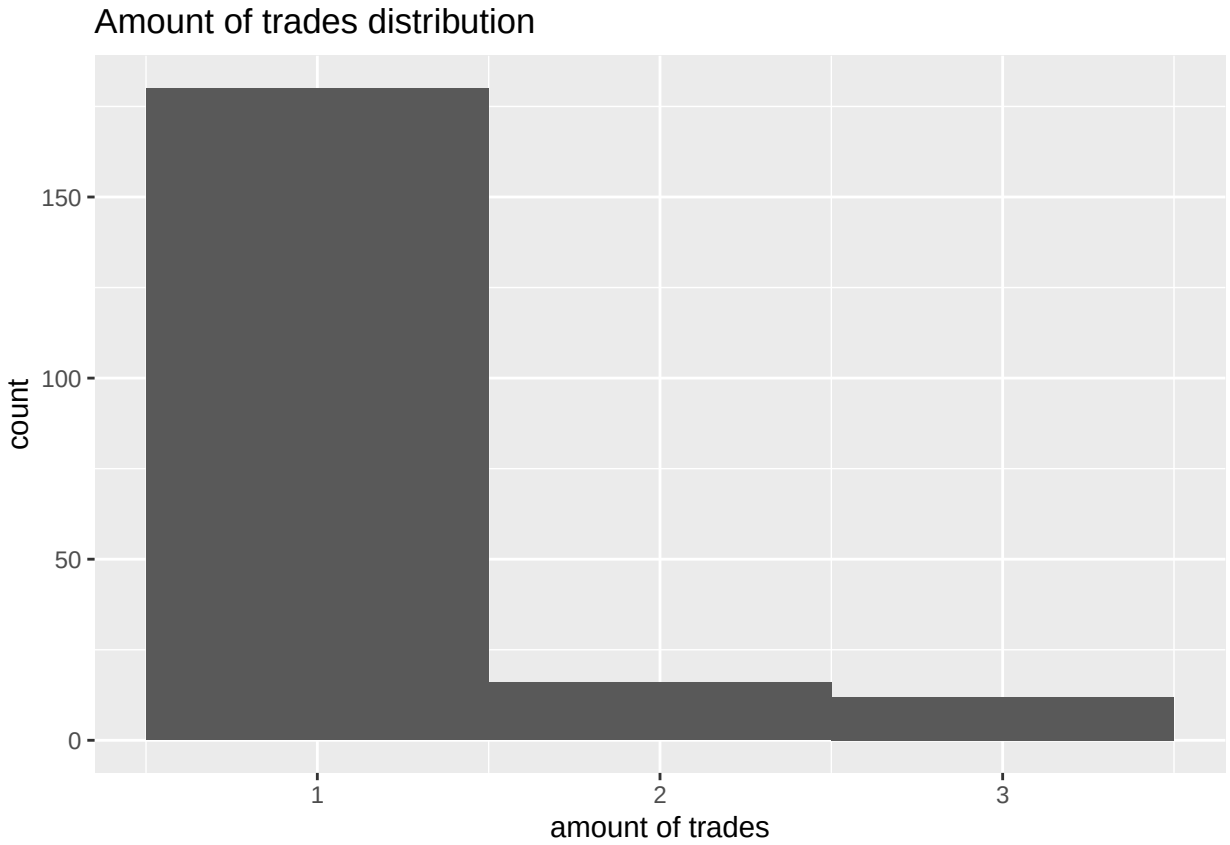
```
## # A tibble: 2,714 x 4
##    from     to             t membership
##    <chr>    <chr>      <int>      <dbl>
##  1 25029978 25030062 1264423          1
##  2 25029957 25030064 1264424          2
##  3 25030066 25030067 1264425          3
##  4 25030106 25030124 1264426          4
##  5 25030105 25030127 1264427          5
##  6 25030113 25030130 1264428          6
##  7 25030095 25030132 1264429          7
##  8 25030103 25030136 1264430          8
##  9 25030352 25030416 1264433          9
## 10 25030364 25030417 1264434         10
## # ... with 2,704 more rows
```

Now we are ready to filter trades that have sequential orders.

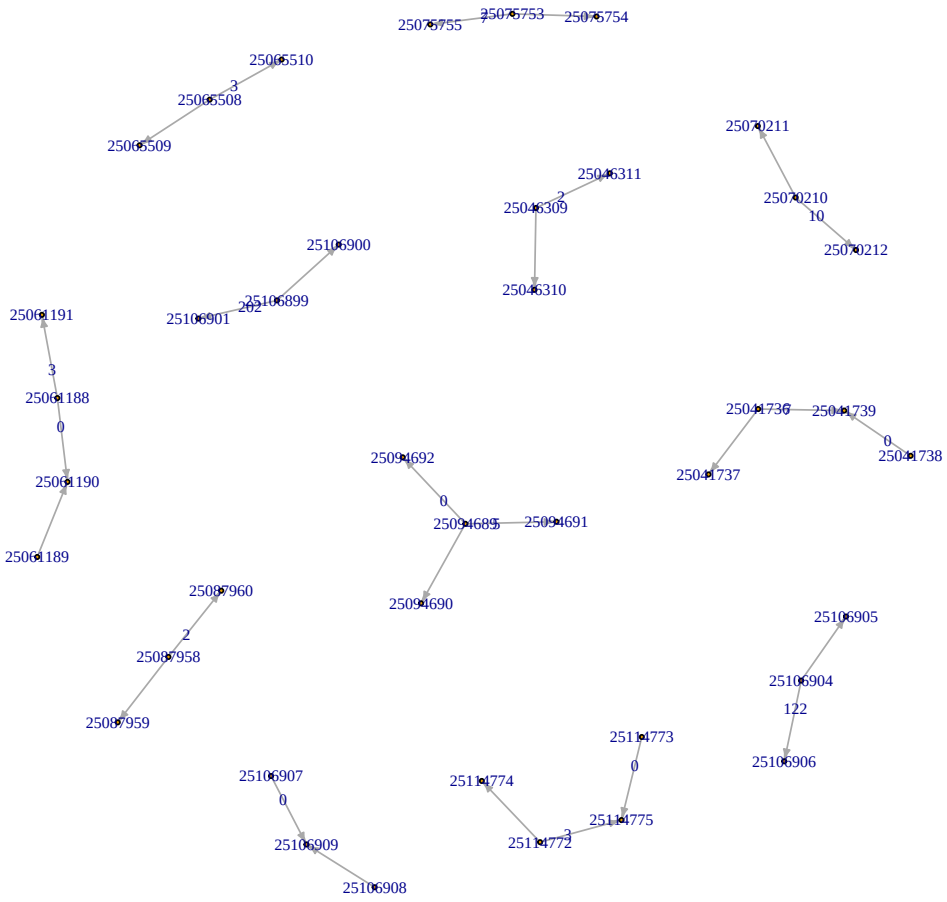
```
xdf <- xdf %>%
  filter(membership %in% l)
as_tibble(xdf)
```

```
## # A tibble: 208 x 4
##    from     to             t membership
##    <chr>    <chr>      <int>      <dbl>
##  1 25030066 25030067 1264425          3
##  2 25032402 25032403 1264519         35
##  3 25033095 25033096 1264550         49
##  4 25033629 25033630 1264560         55
##  5 25033723 25033724 1264569         60
##  6 25034699 25034700 1264618         79
##  7 25034701 25034702 1264619         80
##  8 25034718 25034719 1264620         81
##  9 25035277 25035278 1264637         86
## 10 25035997 25035998 1264679        108
## # ... with 198 more rows
```

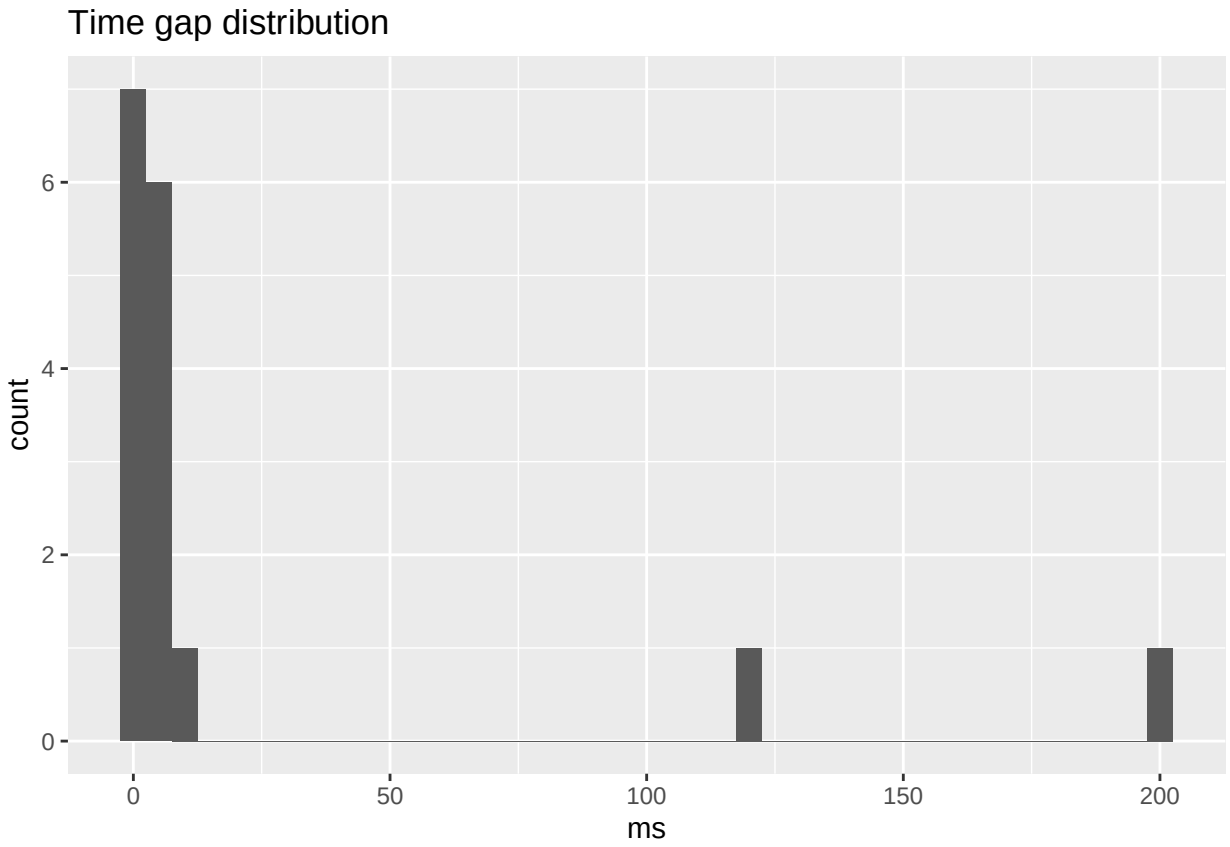
We can see distribution of amount of trades among clusters of interest.



Here is how the found trades graph look like, one-trade clusters are filtered out. Vertices are labeled with order id, the arrows point from seller to buyer. Edges are labeled with time in ms that passed since previous trade. Not labeled edges are the first trade in the cluster.


25075755
25075753
25075754
25065510
3
25065508
25065509
25070211
25046311
2
25046309
25070210
10
25070212
25046310
25106900
25106899
202
25106901
25061191
3
25061188
0
25061190
25061189
25041736
25041739
0
25041738
25041737
25094692
0
25094685
25094691
25094690
25087960
2
25087958
25087959
25106905
25106904
122
25106906
25114773
0
25114774
25114775
3
25114772
25106907
0
25106909
25106908

Trade time gap distribution among clusters (obviously clusters with more than one trade).



These outliers, let's find them.

```
left_join(xdf, df, by = 't') %>%
  group_by(membership) %>%
  mutate(dT = T - lag(T)) %>%
  filter(dT > 100) %>%
  select(membership)
```

```
## # A tibble: 2 x 1
## # Groups:   membership [2]
##   membership
##        <dbl>
## 1       1012
## 2       1013
```

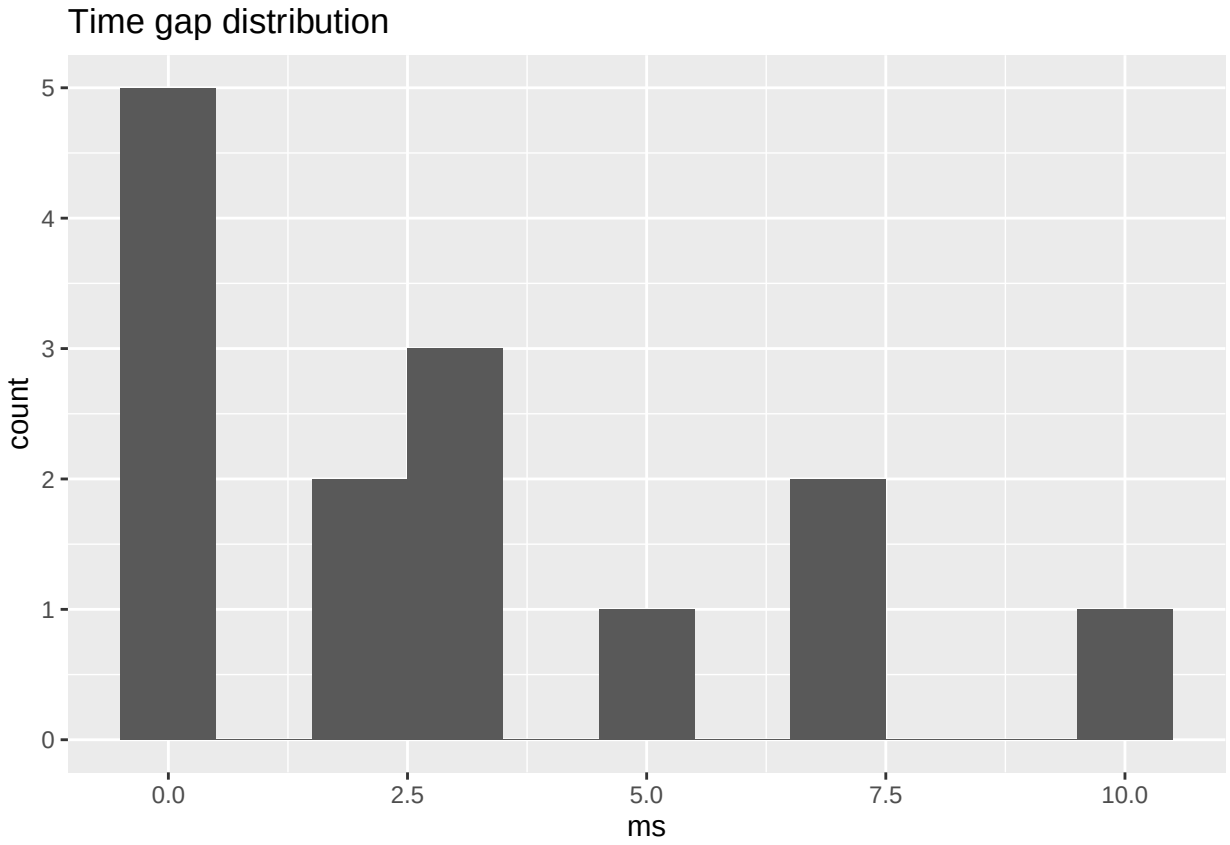
```
left_join(xdf, df, by = 't') %>%
  filter(membership %in% c(1012, 1013)) %>%
  select(from, to, membership, p, q, t, T) %>%
  as_tibble()
```

```
## # A tibble: 4 x 7
##   from     to       membership        p     q       t             T
##   <chr>    <chr>         <dbl>    <dbl> <dbl>   <int>         <dbl>
## 1 25106899 25106900       1012 0.000144 24.6  1267492 1649931375400
## 2 25106899 25106901       1012 0.000144 19.0  1267493 1649931375602
## 3 25106904 25106905       1013 0.000144 41.0  1267494 1649931376211
```

```
## 4 25106904 25106906       1013 0.000144  2.45 1267495 1649931376333
```

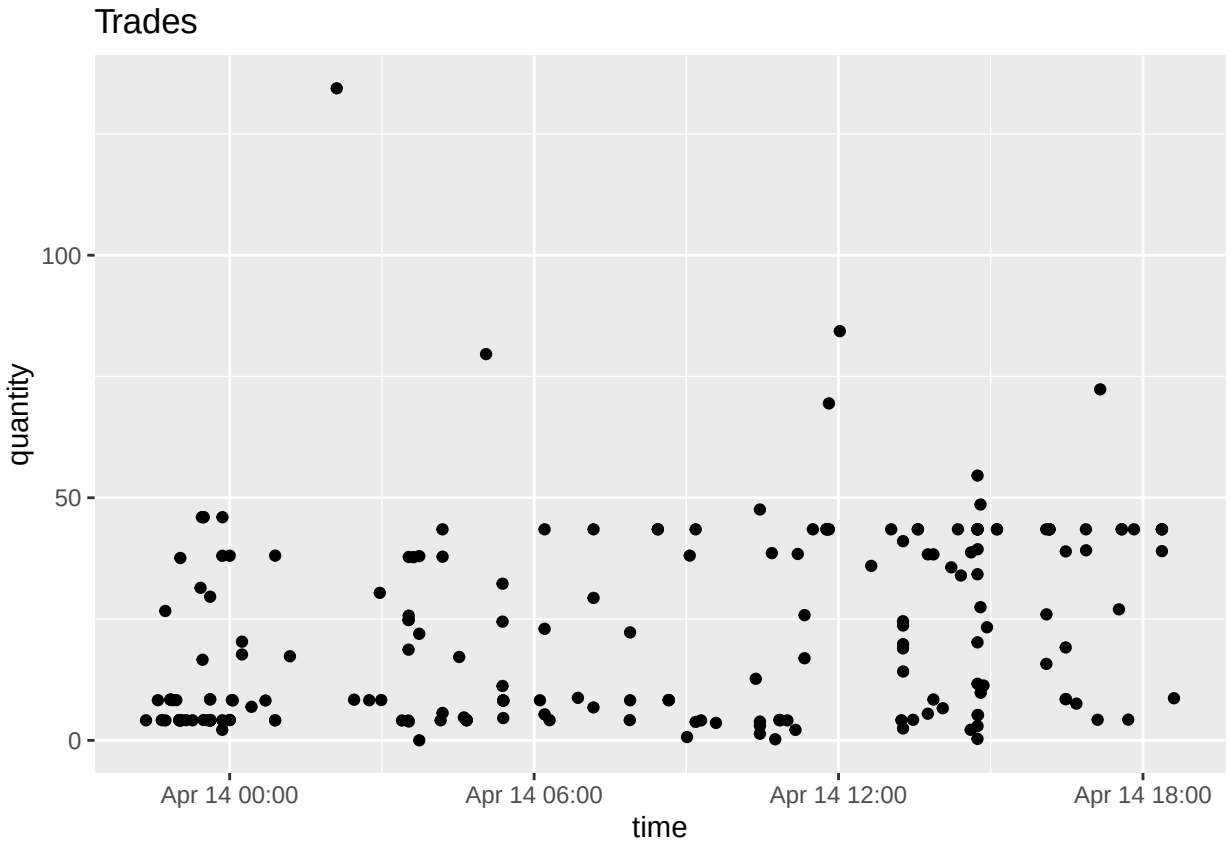
It's hard to say, looks like a glitch. *25106899* placed an order to sell, *25106900* placed an order to buy (we don't know in what time though), then in 202 ms after their deal *25106901* placed its order to buy remaining quantity. The same for the others. But their trade ids are in a row and there is no *25106902*, *25106903* even in the original data frame.

Without them it shows as



Many 0 time gaps between sequential matching orders can be seen.

All the found traded as follows



Among them these are trades that all had 0 time lag in their cluster.

```
##       from       to membership          p     q       t            T
## 1 25106908 25106909       1014 0.00014397 19.82 1267496 1.649931e+12
## 2 25106907 25106909       1014 0.00014398 23.68 1267497 1.649931e+12
```

Only one *1014* cluster has been found.

Overall volume produced could be calculated as

```
left_join(orig_df, xdf, by = 't') %>%
  mutate(v = q * p, membership = ifelse(is.na(membership), 0, 1)) %>%
  group_by(membership) %>%
  summarise(s = sum(v))
```

```
## # A tibble: 2 x 2
##   membership      s
##        <dbl>  <dbl>
## 1          0 18.2
## 2          1  0.651
```

```
100 * 0.651 / 18.249
```

```
## [1] 3.567319
```

3.6% overall.

Conclusion

A wash trading detection approach has been considered that allows to spot possible fraudulent activity.

Provided limited data - trade order ids, trade time (no order book with time and ids, no account ids), it's based on assumption that offenders make deals between their accounts, and in order to achieve that, they place sequential matching orders with minimal time gap.

A method has been proposed to build a graph from the stream with order id as vertices, trade relation as edges, and to find all clusters (connected graphs) that have sequential order ids. These clusters would be of interest.

0 time gap between found trades would indicate high probability of fraud. Whether not 0 time gaps and to what extent are legit (for the found trades) would depend on the exchange individual performance characteristics. It seems research needed in this area.

Single trades may be legit, but still look unusual. We didn't know how much time passed between the two orders, but the fact that two sequential orders matched exactly looks appealing. Order book information would help here to check how fast the orders followed.