Analysis of 23C3 Sputnik data

Tomasz Rybak
tomasz.rybak@post.pl

Table of content

Sputnik system

- System for tracing movement of people in closed space
- Each person is wearing tag sending signal
- Readers receive signals and send it to aggregating server
- Server stores all packets and tries to calculate position of each tag, using triangulation
- It was used during 23C3 and CCC2007
- This presentation describes data gathered during 23C3

Web pages

- Main page of project http://www.openbeacon.org/
- Description of released data http://wiki.openbeacon.org/wiki/Datamining
- Page of Peter Meerwald with some analysis http://pmeerw.net/23C3_Sputnik/
- Parser of log files http://cakelab.org/ kaner/sputnik_01/
- My page http://www.bogomips.w.tkb.pl/sputnik.html

Transmission details

- Usually RFID is passive; tag receives power from reader and is active only during reading process
- This solution limits range of transmission; also transmission occurs only if reader is present
- Sputnik uses active tags (each has own battery) with range up to 10m in buildings
- Sputnik uses 2.4GHz range; human body cuts down signal by 50%
- Tags send signals with varying strength. It allows for estimating how far from the reader tag is
- 25 readers deployed through entire BCC, placed in such way that in most cases at least two readers see every tag
- Having data from few readers allow for estimating of location of tag

USB reader

- Device connected to USB; can receive and send Sputnik signals
- Described in http://wiki.openbeacon.org/wiki/OpenBeacon_USB
- Smaller and more useful for individual user that Ethernet reader
- Creates device /dev/ttyACM* and behaves as terminal; one can connect to it ("cu -lttyACM0 -s 115200") and read received data
- Text lines with received data is send to terminal
- Format: "ID,Sequence,strength,flags"
- Can be controlled by sending commands using the same terminal
- Can receive and send RFID packets

USB reader usage

- USB reader can be used for controlling access to computer
- The easies solution is custom script checking presence of tag with particular ID
- More sophisticated uses may require PAM module
- Parameters of such module: ReaderID (or device file name), TagID, UserName
- Module returns true if there is tag, false if not
- Configuration in /etc/pam.d/login
- Sputnik module put into section describing authentication
- If “requisite” option is used, no one will login without tag detected by reader
- If module put as “required” and placed after unix module, particular users login ability may be blocked without tag
- Similar to USB PAM example

Function for checking presence of tag

```
station = 1
id = int(raw_input("Give me your ID: "))
c.execute("""SELECT time, strength
FROM sputnik.reader WHERE station = %s
AND id = %s AND time > now()-'5 second'::interval""", (station, id,))
found = False strength = 255
for i in c.fetchall():
    found = True
    if strength > i[1]: strength = i[1]
if not found: print "You are not visible at the computer"
if strength > 0x55: print "You are too far away from the computer"

b = random.randrange(1, 6)
print "Press button exactly %i times." % (b, )
time.sleep(5)

c.execute("""SELECT time, strength, tags
FROM sputnik.reader WHERE station = %s
AND id = %s AND time > now()-'5 second'::interval""", (station, id,))
found = False strength = 255 pressed = False
for i in c.fetchall():
    found = True
    if strength > i[1]: strength = i[1]
    if i[2] != None and len(i[2]) > 1:
        if i[2][0] == 'button0' and int(i[2][1]) == b:
            pressed = True
if not found: print "You are not visible at the computer"
if strength > 0x55: print "You are too far away from the computer"
if not pressed: print "You have not pressed button exactly %i times." % (b, )
```

Data from 23C3

- Data gathered during 23C3 was made available as both XML and binary files
- Both files have own problems

XML file

Consisted of "observation" tags with following attributes:

id	ID of tag
time	
position	position of tag; (0, 0, 0) if unknown
direction	always (0, 0, 0)
priority	always the same value 24
min-distance	always 0.0
max-distance	always 255.0
observer	URL of aggregating station; only one value present in file
observed-object	URL of station together with tag ID

XML data

- Many data entries contain (0, 0, 0) as position; those come from tags which positions could not be determined
- 357974 entries, 248426 with known position
- Positions of people on all floors are present; but no description of reading stations
- Many periods without data; only bursts of activity logged, for only few hours a day
- According to presentation from 23C3 Sputnik server can have more than one module processing data
- So it is possible that XML data comes from such module, which was no active all the time
- In this case, URL is address of server with this module; so there was only one module or data from only one module was saved, as there is only one URL in XML file
- I did not use XML file for analysis

Binary format according to source code

0-4	timestamp
5-8	reader station IP
9	size of frame (0x10)
10	protocol (0x17)
11	flags (0x02 — button pressed)
12	strength of signal
12-16	sequence number
17-20	Tag ID
21-24	check sum

Binary format present in file

0-4	timestamp
5-8	reader station IP
9-12	garbage (used by me to write ID)
13-16	garbage, reversed IP of reader station
17	size of frame (0x10)
18	protocol (0x17)
19	flags (0x02 — button pressed)
20	strength of signal
21-24	sequence number

Binary data

- 11144232 data samples
- Missing IDs of tags.
- Binary data has extra 8 bytes in file 2006-12-27-13, at byte 0xC4FF8 (806904); those bytes are beginning of some frame; it can be removed and then data can be processed normally
- Repetition in binary data: 65792 repeated readings

Query used to count repeated readings

```
SELECT time, sequence, station, COUNT(*)
FROM sputnik.sputnik
GROUP BY time, sequence, station
HAVING COUNT(*) > 1;
```

Database introduction

- Database is intended for holding and operating on large amounts of data
- Created database can be seen as temporal and spatial database
- Such databases store information about presence of phenomenas in space and time
- Here I store information about presence of person in place at particular moment; if this person presses button, it is also saved
- Additional characteristics, but connected to space position, as they are used to calculate it, are strength of signal and readers that noticed it
- For additional analysis it could be useful to have table with information about which readers see which rooms, possibly in more detailed form, like which reader is inside room, and which is only in proximity
- Scripts for creating tables and loading data into database are on my web page

Created tables

- I started from one table for both XML and binary data
- As it was not suitable, I divided data into more tables

station	Describes readers
sputnik	base table for storing data; tables with data inherit from it
ccc23	contains binary data from 23C3
ccc23xml	contains XML data from 23C3; has additional columns containing values of attributes from XML file
reader	table used to store data received by USB reader
adjacency	stores count of readings seen by pairs of readers
room	describes lecture rooms
event	describes events that took place during 23C3; taken from Schedule XML file

Base table for holding data from tags

id	
time	
sequence	value of sequence counter
strength	strength of signal
station	id of station that received this signal
tags	array of data, like pressed button

XML data table

is like raw data table and also contains:

position	position of tag
plane	position on the floor
direction	direction; currently only (0. 0, 0)
observer	
observedobject	
priority	
mindistance	
maxdistance	

Table of rooms

Describes room in which events (lectures) were taking places.

- **id** identifier of room
- **name** name of room: “Saal 1”, “Shelter foo”, . . .
- **shape** path describing room shape. Currently empty column; data to fill it could be taken from GPS data or from building plans
- **ymin**
- **ymax**
- **bbox** Is it necessary, or better use geometry calculations or PostGIS?

Event table

Describes information about events. Populated using XML schedules published on http://www.ccc.de/

id	identifier of event
organizerid	
name	name of event
place	identifier of room event is taking place
description	human-readable description
address	URL of description of event
start	timestamp of beginning moment of event
finish	timestamp of end moment of event

Database size

- Table with data from binary file takes 700MB
- Three indexes were created: for ID, for time, and for sequence
- Values stored in time and sequence columns have size of 8 bytes
- Each index takes about 250MB
- ID is ordinary integer, and takes 4 bytes
- Index for it takes about 130MB
- Indexes were necessary to have working database
- So in result it is big database which grows when changed

Database maintenance

- Finding all possible sequences changes large amount of rows in database
- Estimators statistics soon go out of sync with real data
- Vacuuming is not crucial if we have enough hard drive
- However statistics are, and they are made by VACUUM ANALYZE
- So autovacuum can be used to clean up changed tables
- It can analyse very frequently, and vacuum not so often, and can be set for every table on different parameters (table pg_catalog.pg_autovacuum)
- As sputnik table is large (11.1M rows) we need to analyse it often. Analyse ofter 0.5% rows been changed and vacuum after 10% rows been changed.
- Set autovacuum to more aggressive for this table — cost 500, delay 0

Exporting data

- libpq tries to fetch entire result into RAM
- When exporting Sputnik data this can be a problem
- I was getting “Out of memory” error
- Cursor can be used to fetch only few rows at the time
- Not fetching entire result is on ToDo list of PostgreSQL in libpq section

Behaviour of system

- Tags sends its ID and signal strength
- To avoid analysis of signal, transmission is encrypted using XXTEA
- If only ID and strength was send, there would be not many different packets, so it would be easy to disturb or replay transmission
- The easiest solution is to add time to signal
- Instead of complicating tag with real-time clock, ever increasing 32-bit counter ticking once about every 1.5s to 2.5s was added
- This will provide variability of send packets
- Server discards packets with sequence number smaller than what was seen
- It will take care of replay attacks
- To avoid problems with counter starting from 0 when battery is changed, each reset (battery out) increments larger word of counter (adds 0x10000)
- So database contains monotonic sequences of counter values

Graphs

- Graphs like those made by Peter Meerwald
- Mine are done using data imported to database, and present slightly higher numbers, probably because of repeated pings that were in database
- Peter's programs used hashes, so he had not repetitions

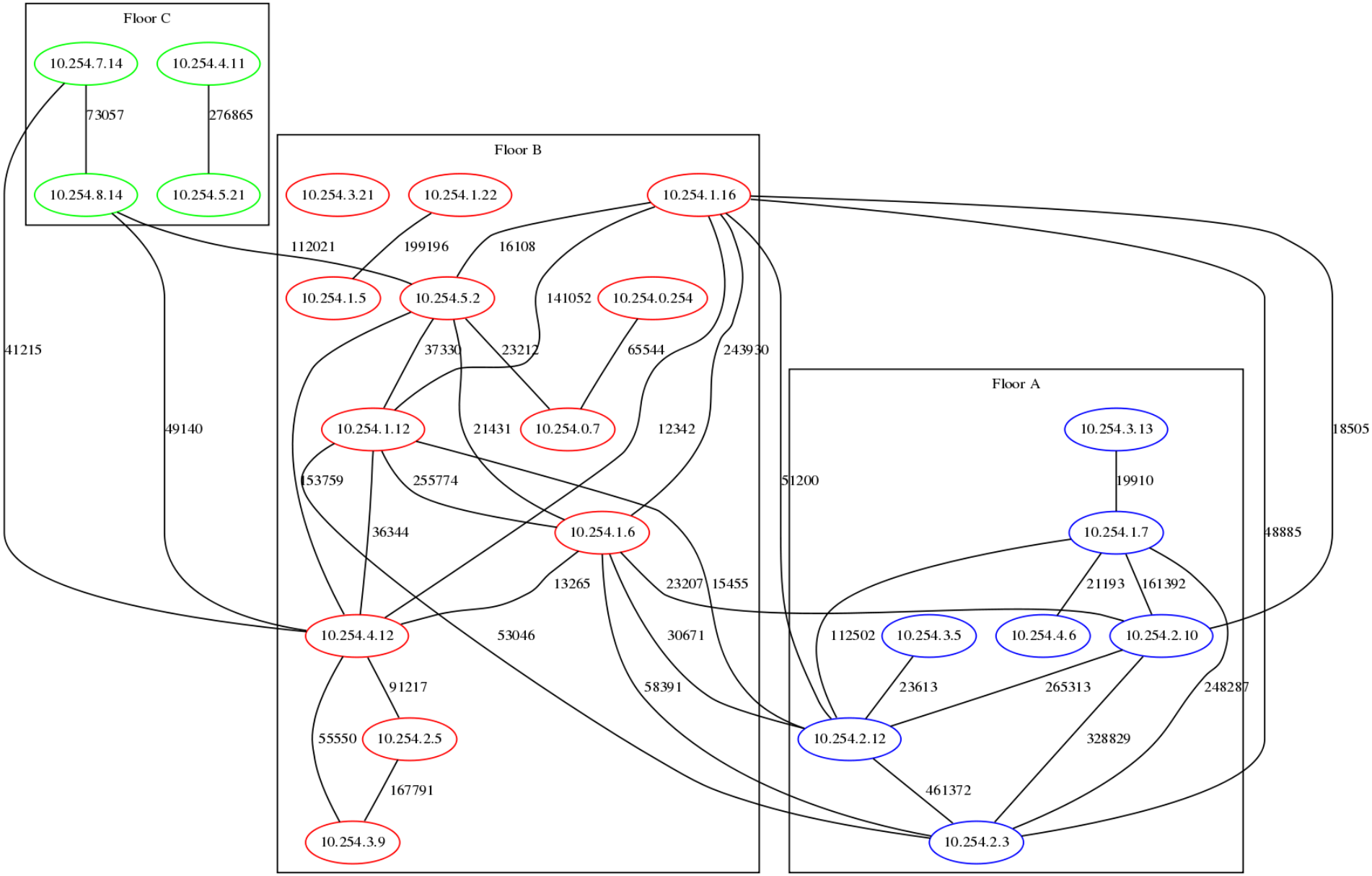


Figure: Pings read by more than one station (> 1000)

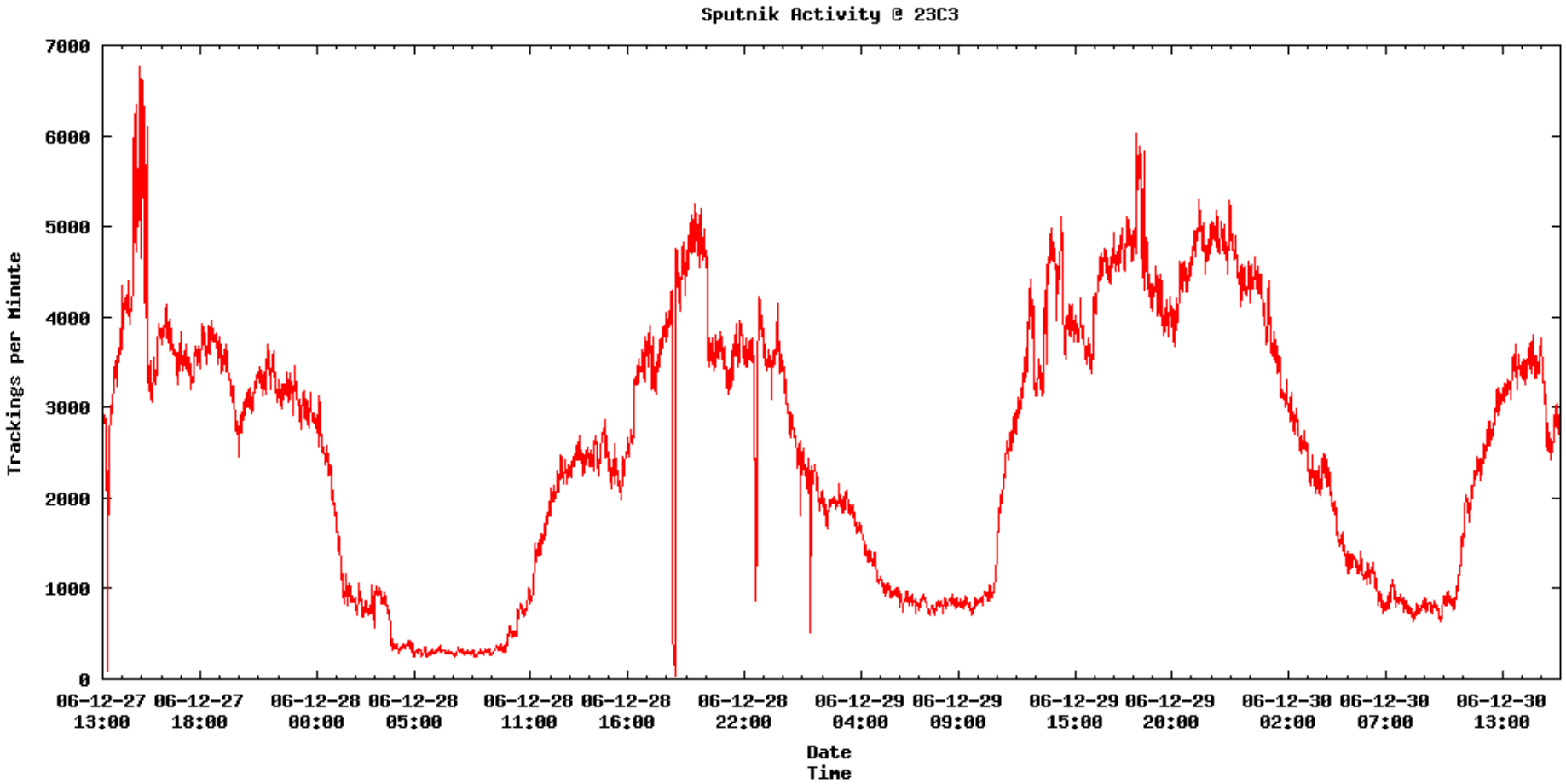
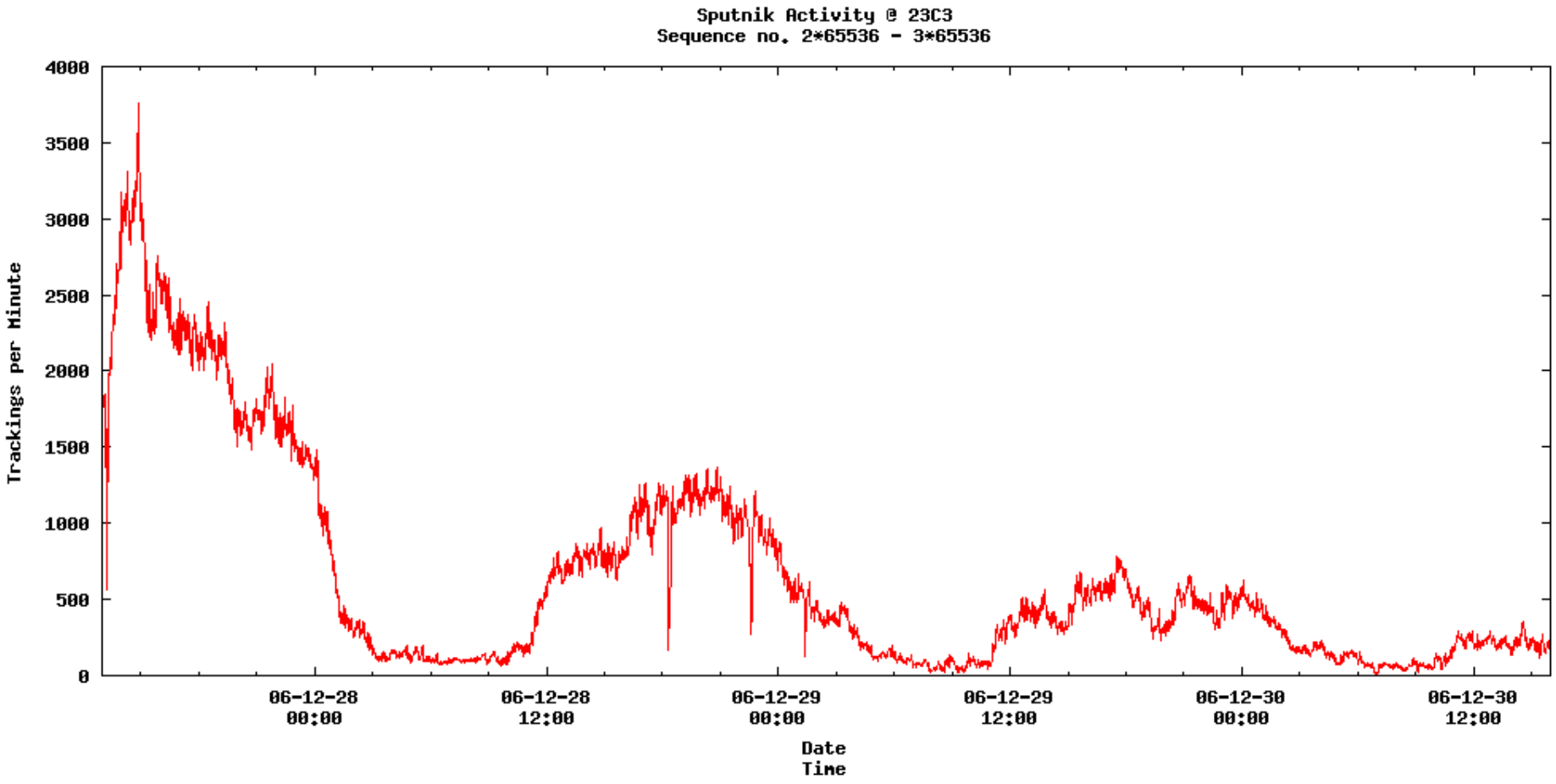
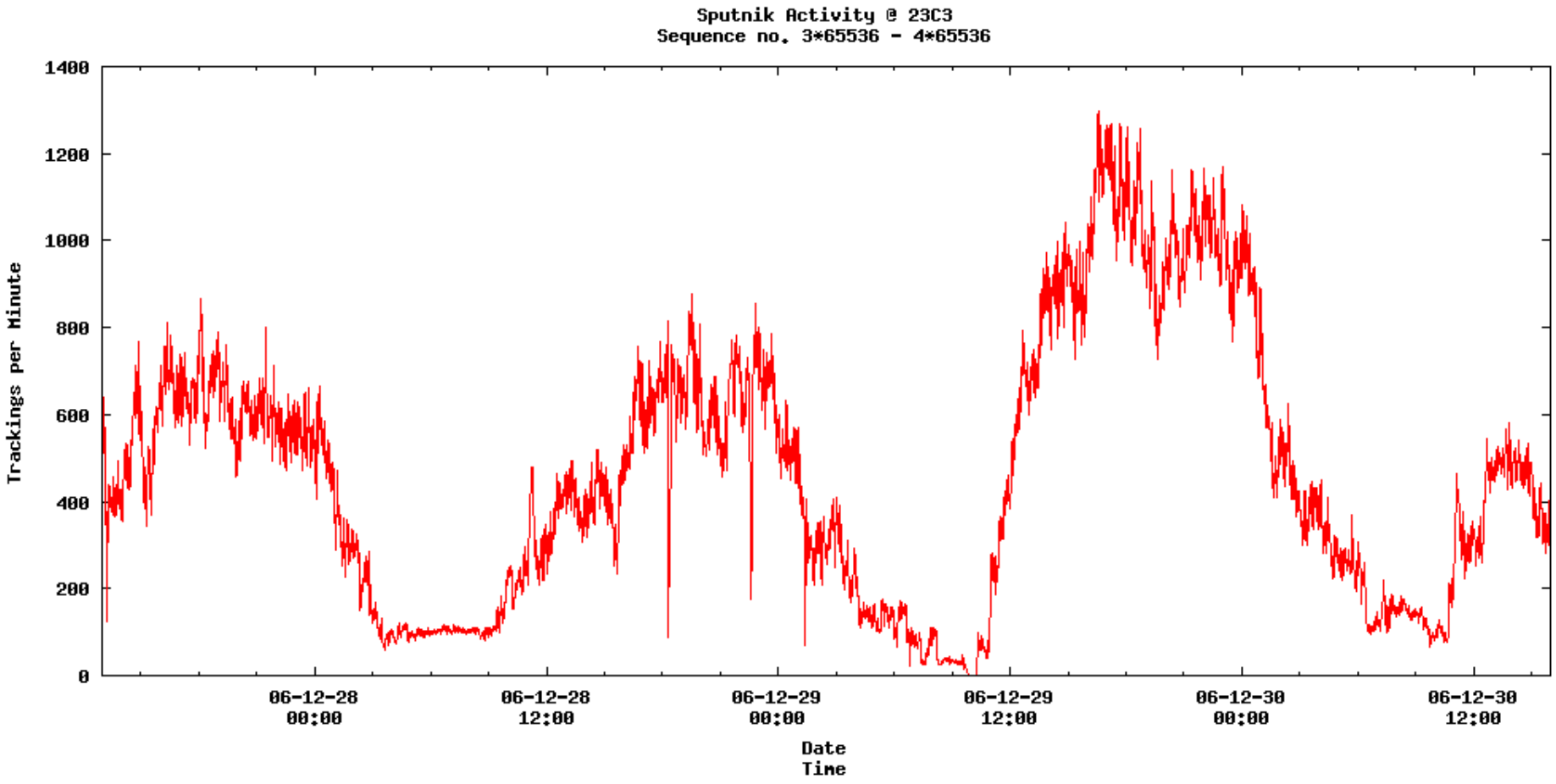


Figure: Number of packets read during one minute

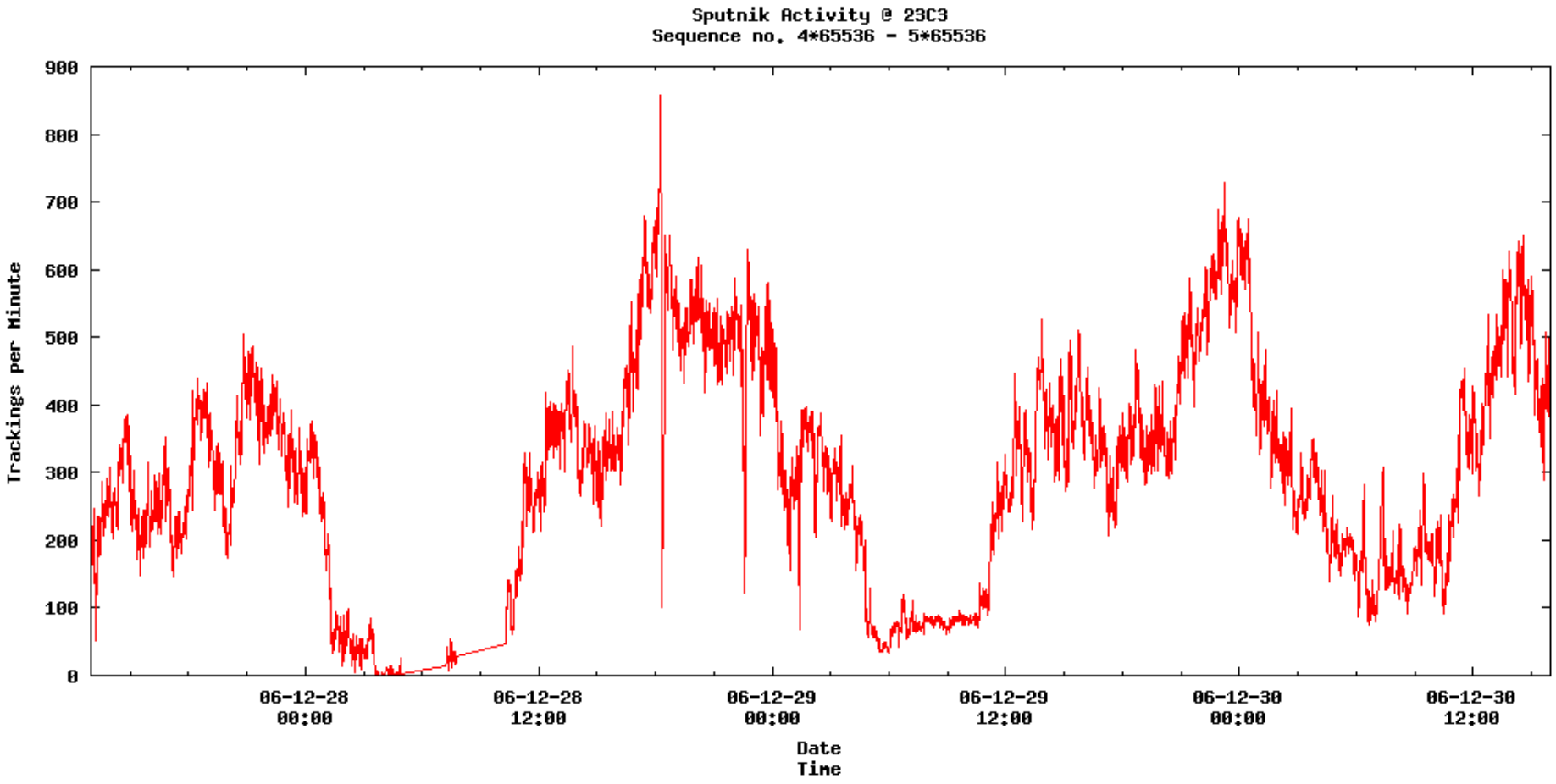
Number of packets read during one minute for sequence $\in < 2 * 65536; 3 * 65536 >$



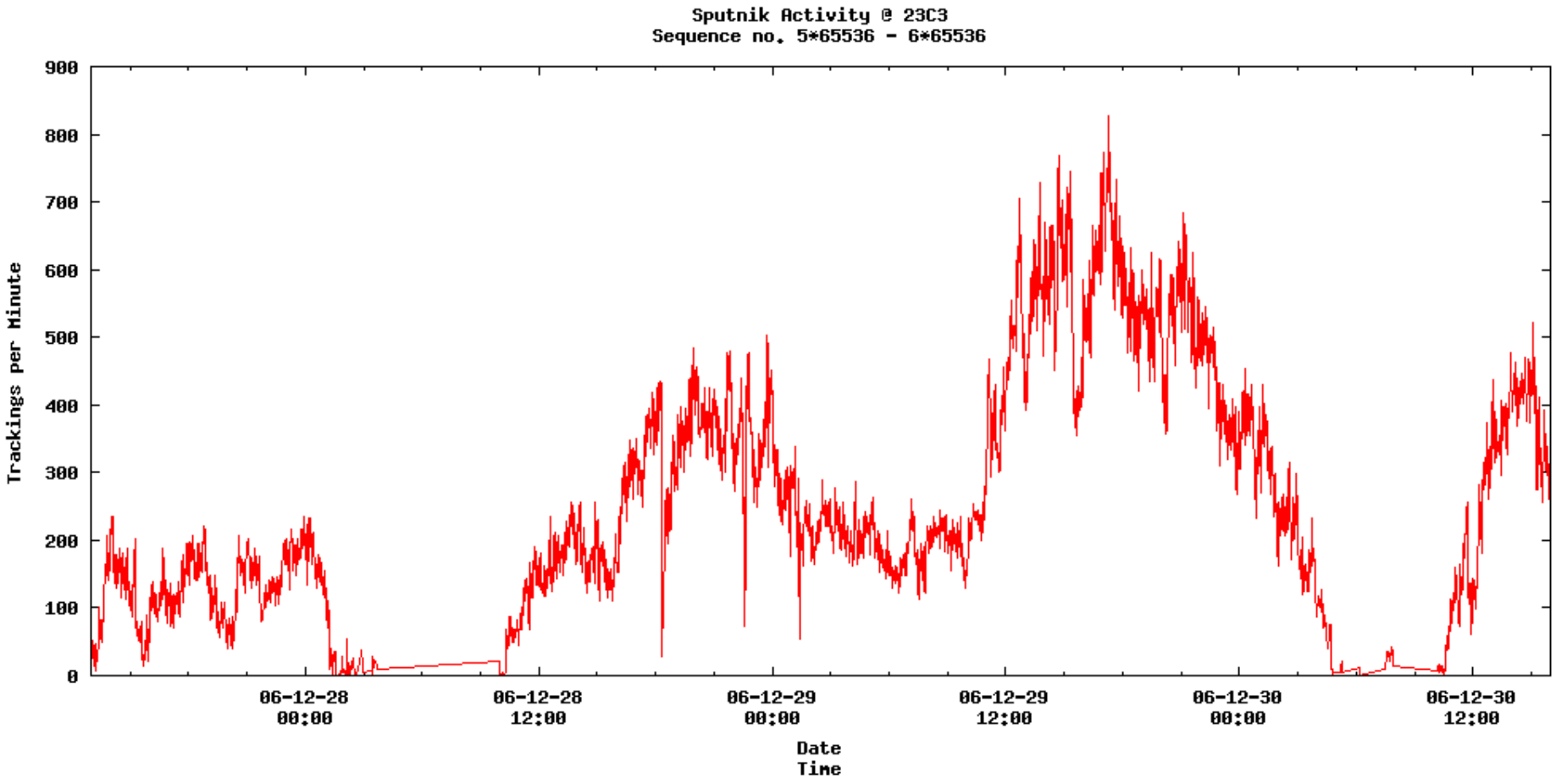
Number of packets read during one minute for sequence $\in < 3 * 65536; 4 * 65536 >$



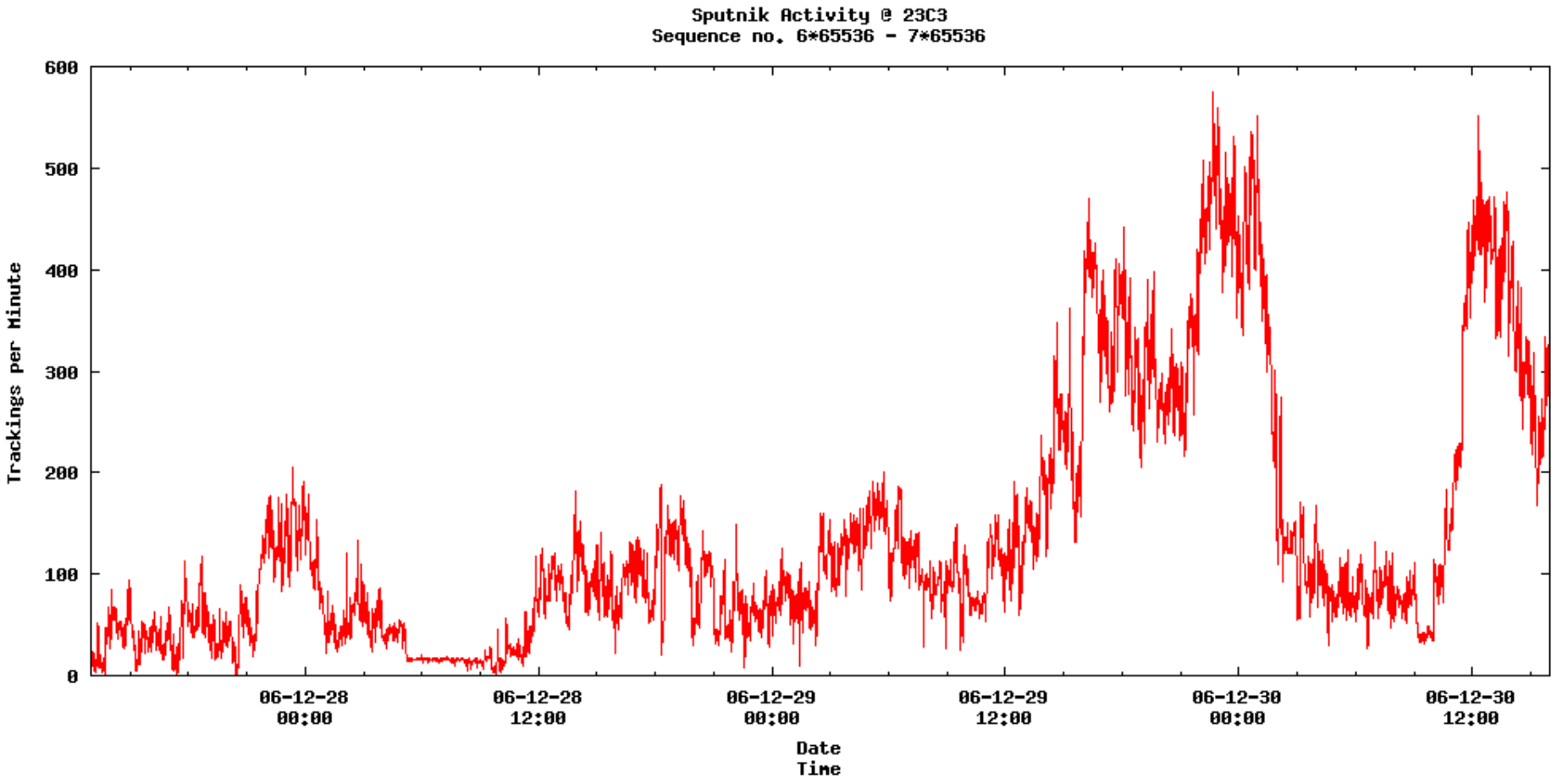
Number of packets read during one minute for sequence $\in < 4 * 65536; 5 * 65536 >$



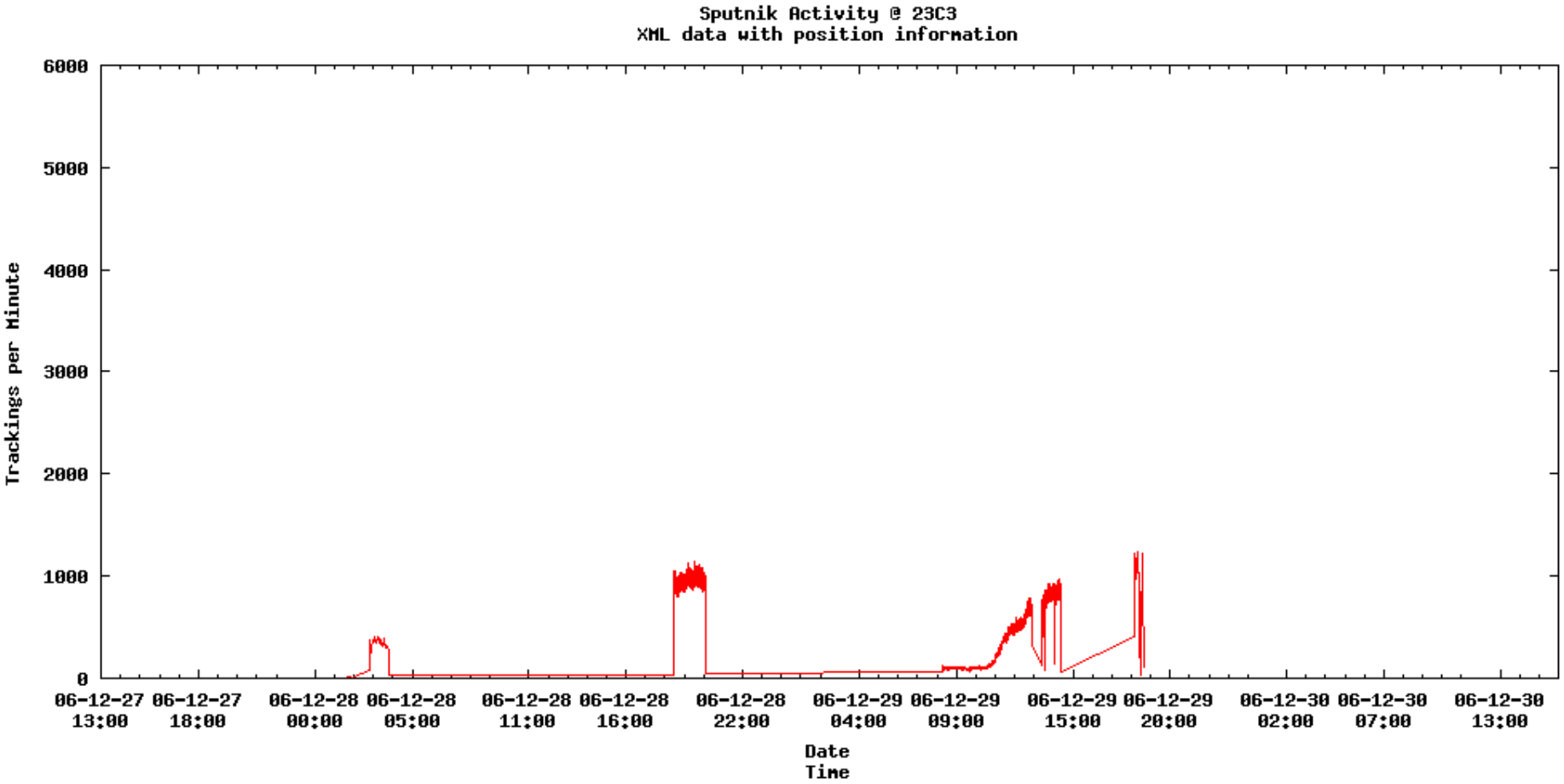
Number of packets read during one minute for sequence $\in < 5 * 65536; 6 * 65536 >$



Number of packets read during one minute for sequence $\in < 6 * 65536; 7 * 65536 >$



Number of packets read during one minute in XML data



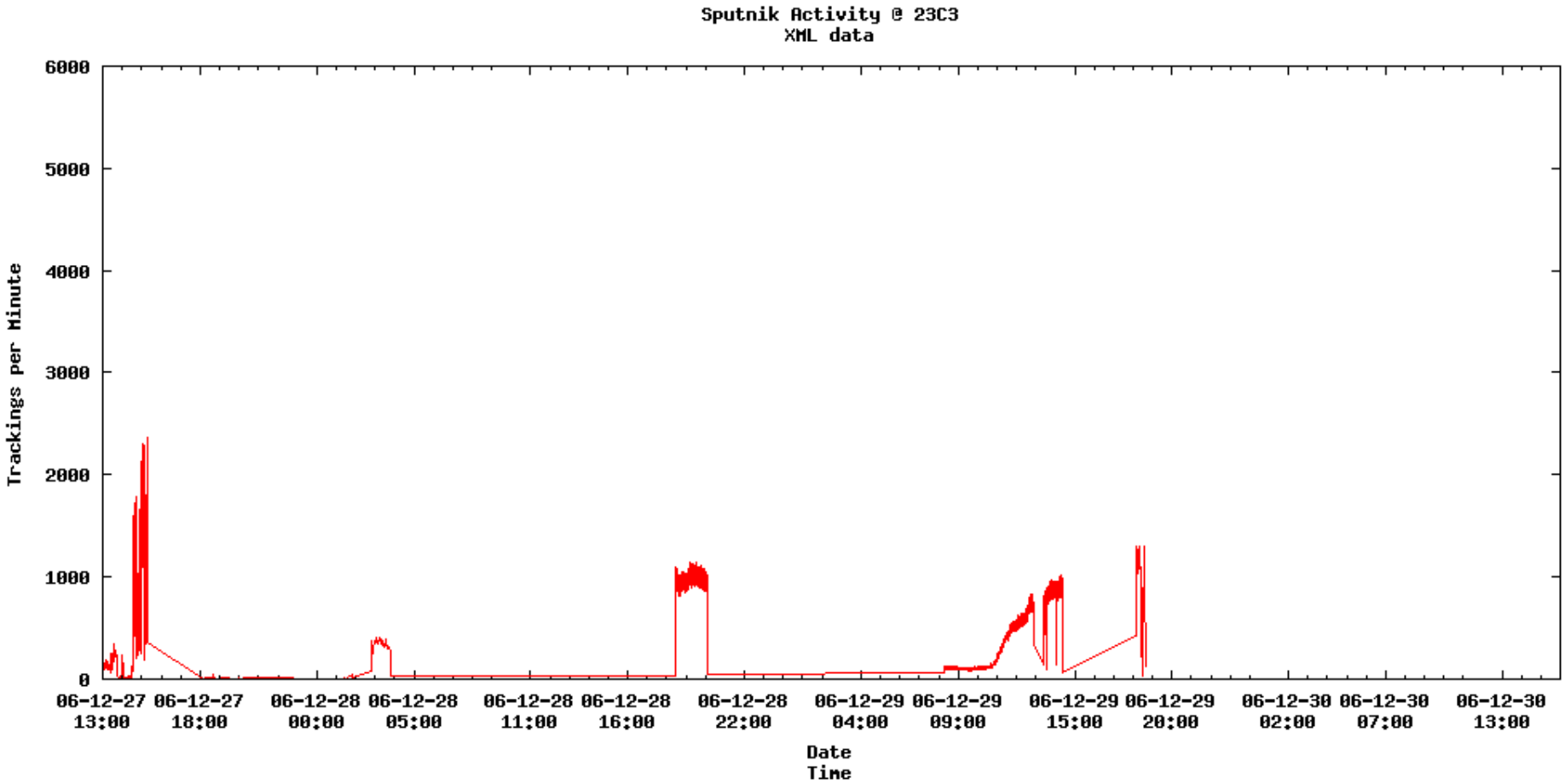


Figure: Number of packets read during one minute including unknown points

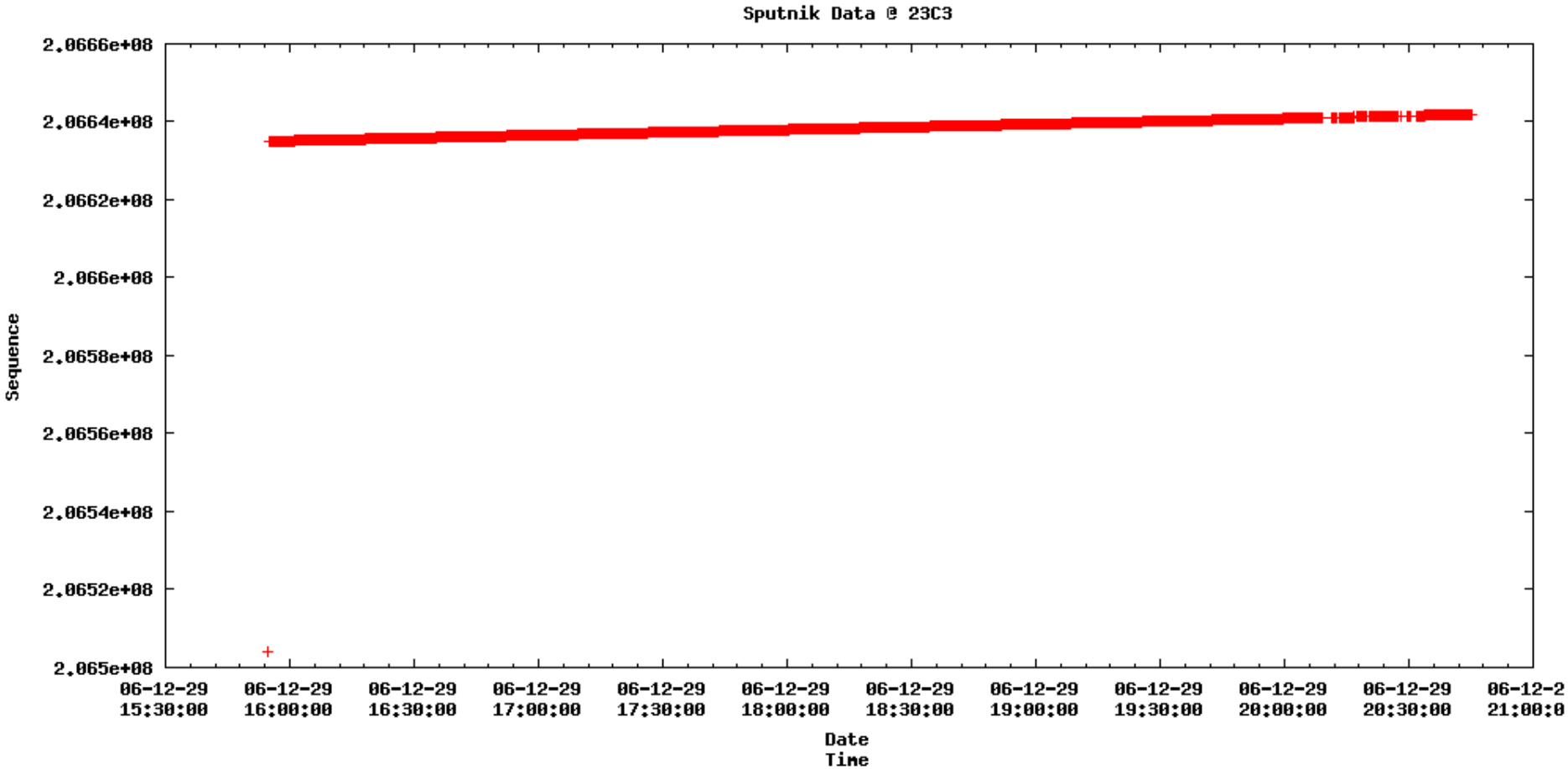
Packets read by each station

Id	IP address	count
2	10.254.2.3	1322696
21	10.254.5.21	880833
3	10.254.2.12	760606
15	10.254.1.6	758782
18	10.254.5.2	596466
14	10.254.4.12	589640
20	10.254.8.14	585443
26	10.254.1.16	570525
5	10.254.1.7	568765
4	10.254.2.10	563488
1	10.254.4.6	542657
16	10.254.1.12	532699
22	10.254.4.11	528187
11	10.254.1.22	494524
10	10.254.1.5	448760
9	10.254.2.5	428565
8	10.254.3.9	376396
24	10.254.3.5	231483
23	10.254.7.14	225075
17	10.254.0.254	187078
6	10.254.3.13	130379
13	10.254.0.7	129144
12	10.254.3.21	54863
25	10.254.0.100	8524

Strength of packets

Strength	count
0	182874
85	568413
170	1167287
255	9225658

Scatter plot of some data



Searching for sequences

- Build sequences of consecutive counter numbers, and then attach id to the each one
- Even if rebuilding sequences is possible, original numbers of tags are lost
- Either try to build some initial sequences and then join them
- Or treat each packet as separate sequence, and then join them
- In my opinion better is to try to find some base sequences and only then try to join them
- First try to search local (small) sequences, as global searching requires much CPU, RAM, and disk resources

First attempt of building sequences

```
SELECT time, extract('epoch' from time), sequence
FROM sputnik.sputnik WHERE id IS NULL AND
time BETWEEN %s::TIMESTAMP WITH TIME ZONE
AND %s::TIMESTAMP WITH TIME ZONE+%s::INTERVAL
for i in c.fetchall():
    old_e, old_s = int(i[1]), int(i[2])
    old_major = old_s/65536
    old_minor = old_s%65536
    p = []
    for j in data:
        e, s = int(j[1], int(j[1])
        major = s/65536
        minor = s%65536
        probable = (major == old_major and minor == old_minor+1)
            or (major == old_major+1 and minor == 0)
        if probable: p.append([e, s])
    if len(p) > 0:
        print old_e, old_s,
        for j in p: print j[0], j[1],
```

Finding short-term local sequences

- Choose short period of time (10s, 20s, . . .)
- Find all consecutive sequences of ticks
- Assume that each tick is about 1.5s from another, so increase of time should go along with increase of ticks
- Use linear function to predict growth; coefficient should be between 1.0 and 2.0
- So in very short term difference between two ticks will be 1 or 2 seconds
- In longer term, there should be N ticks and 1.5N seconds
- For now ignore signal strength and seen stations

Alternative points in sequences

- What if there is more than one matching tick and they cannot be used together?
- For example counter values M and N ($M \neq N$) at the same time T
- Or the same counter value N at moments T_0 and T_1, where $T_0 \neq T_1$
- We build all alternative sequences and choose the best one:
  - longest
  - with the most regular increases (no jumps)
  - with closest resemblance of original:
    - Accept with coefficient between 1.25 and 1.75
    - The longer term, the closer to 1.5 it should be
- Should we create function that gives coefficient of the best sequence? I.e. $length + \Sigma(timediff - 1.5)^2$?
- Or rather create function giving the next possible time and sequence values? But how to compute when time has started?

Distance between sequence values

```
# Assumes a <= b
# Will not work when there is more than 1 overflow
def GetTickDistance(a, b):
    majora = a/65536
    minora = a%65536
    majorb = b/65536
    minorb = b%65536
# Inside one minor, or less than minute to overflow
    if majora >= majorb or minora >= 65500:
        return b-a
    else:
        return majorb-majora + minorb+1
```

Creation of all possible alternative sequences

- Build hash of all counter values to list of all times of this value, and all time values to store all counter values at this moment
- Check in increasing order
- If found value can be added (time is more than previous, and so is counter) add to the sequence
- If not, start alternatives
- Add previous one and current one as two alternatives
- For subsequent points, add them to alternatives
- If one is found that can be added to all alternatives, add all alternatives to sequence (to choose one later)
- If no, add point to the sub-sequences where it can be added
- If there is no such sub-sequences, start new one with this point

Finding best sequences amongst all created

```
# Sequence with len >= 3
def FindBestSequence(a):
    b = max(map(len, a))
    c, a = a, []
    for i in c:
        if len(i) == b: a.append(i)
# Find minimal difference between min and max, in case of many alternative sequences
    best = i = a[0]
    ds = float(i[1][0]-i[0][0])/GetTickDistance(i[0][1], i[1][1])
    mini = maxi = ds
    for j in range(1, len(i)-1):
        ds = float(i[j+1][0]-i[j][0])/GetTickDistance(i[j][1], i[j+1][1])
        mini = min(mini, ds)
        maxi = max(maxi, ds)
    c = (mini-1.5)*(mini-1.5)+(maxi-1.5)*(maxi-1.5)
    for i in a[1:]:
        ds = float(i[1][0]-i[0][0])/GetTickDistance(i[0][1], i[1][1])
        maxi = mini = ds
        for j in range(1, len(i)-1):
            ds = float(i[j+1][0]-i[j][0])/GetTickDistance(i[j][1], i[j+1][1])
            mini = min(mini, ds)
            maxi = max(maxi, ds)
        d = (mini-1.5)*(mini-1.5)+(maxi-1.5)*(maxi-1.5)
        if d < c: best, c = i, d
    return best
```

Possible approaches in algorithm

- Either create sub-sequences by going by time or by ticks
- In my opinion those attitudes are equivalent and give the same results
- However, on some examples time gives less results
- Also, counter values give better result — longer sequences
- Should two of them be run and then the best result be chosen?

$O(N^3)$ algorithm

- Sketch of algorithm (leaves $20 \ldots 30\%$ of data without id) is:
- go through all ticks in short period of time (10s, 20s, 60s)
- for each of them, check if another can be added to it's sequence, i.e. if it can be described by $\Delta s = a\Delta t, 1.0 \leq a \leq 2.0$
- Find the best sequence by choosing the longest possible one, and with ticks generated by functions which a is the closest to 1.5
- This makes $O(N^2)$
- If any sequence is found, mark it in database and repeat entire process $\Rightarrow O(N^3)$

$O(N^3)$ algorithm

```
SELECT DISTINCT time, extract('epoch' from time), sequence
FROM sputnik.sputnik WHERE id IS NULL AND
time BETWEEN %s::TIMESTAMP WITH TIME ZONE
AND %s::TIMESTAMP WITH TIME ZONE+%s::INTERVAL
a, b, again = 0, 0, True
while again:
    again, s = False, []
    for i in data:
        majort, majors = int(i[1]), int(i[2])
        p = [[majort, majors]]
        for j in data:
            minort, minors = int(j[1]), int(j[2])
            dt = minort-majort
            ds = GetTickDistance(majors, minors)
            if dt > 0 and ds <= dt and dt <= 2*ds:
                p.append([minort, minors])
        if len(p) > 1:
            again = True
            r = CreateAllSequencesSeqs(p)
            s = FindBestSequence(r)
            a += 1
            if len(s) > b: b = len(s)
            break
    if again:
        for i in s:
            UPDATE sputnik.sputnik SET id = %s
            WHERE sequence = %s AND time = to_timestamp(%s)
            for j in data:
                if i[0] == j[1] and i[1] == j[2]:
                    data.remove(j)
                    break
        id += 1
```

$O(N^2)$ algorithm

- Gets data from database sorted by tick and time
- One step:
  - Get first tuple from database
  - Go through rest of the tuples and find all that can be described by $\Delta s = a\Delta t, 1.0 \leq a \leq 2.0$
  - Find the best sequence using found tuples as building blocks
  - If found one, remove all blocks (tuples) from data set
  - If not, remove only first tuple, as it cannot be used in any sequence
- One step is $O(N)$ as it goes through all tuples
- We do it for each tuple, so algorithm is $O(N^2)$
- It looks like it gives the same results as $O(N^3)$ one

$O(N^2)$ algorithm

```
SELECT DISTINCT time, extract('epoch' from time), sequence
FROM sputnik.sputnik WHERE id IS NULL AND
time BETWEEN %s::TIMESTAMP WITH TIME ZONE
AND %s::TIMESTAMP WITH TIME ZONE+%s::INTERVAL
ORDER BY sequence, time
a, b = 0, 0
while len(data) > 0:
    s, i = [], data[0]
    majort, majors = int(i[1]), int(i[2])
    p = [[majort, majors]]
    for j in data[1:]:
        minort, minors = int(j[1]), int(j[2])
        dt = minort-majort
        ds = GetTickDistance(majors, minors)
        if dt >= 0 and ds <= dt and dt <= 2*ds:
            p.append([minort, minors])
    if len(p) > 1:
        r = CreateAllSequencesSeqs(p)
        s = FindBestSequence(r)
        a += 1
        if len(s) > b: b = len(s)
        for j in s:
            UPDATE sputnik.sputnik SET id = %s
            WHERE sequence = %s AND time = to_timestamp(%s)
            for k in data:
                if j[0] == k[1] and j[1] == k[2]:
                    data.remove(k)
                    break
        id += 1
    else:
        data.remove(i)
```

Function trying to join found sequences

```
def JoinIDs(c, t, d, period):
    main = GetLines(c, t.strftime("%Y-%m-%d %H:%M:%S+01:00"), period)
    after = GetLines(c, (t+d).strftime("%Y-%m-%d %H:%M:%S+01:00"), period)
    before = GetLines(c, (t-d).strftime("%Y-%m-%d %H:%M:%S+01:00"), period)
    for i in sorted(main.keys()):
        majort = main[i]['max-time']
        majors = main[i]['max-seq']
        for j in sorted(after.keys()):
            minort = after[j]['min-time']
            minors = after[j]['min-seq']
            dt = minort-majort
            ds = GetTickDistance(majors, minors)
            if ds <= dt and dt <= 2*ds:
                print "Can Join"
                print "\t", main[i]['id'], main[i]['length'], main[i]['min-time'], main[i]['min-seq'],
                print main[i]['max-time'], main[i]['max-seq']
                print "with", ds, dt, float(dt)/ds
                print "\t", after[j]['id'], after[j]['length'], after[j]['min-time'], after[j]['min-seq'],
                print after[j]['max-time'], after[j]['max-seq']
```

Further improvements

- It might be possible to go even to $O(N)$
- It was not implemented so I am not sure if it would work and give the same results

Further improvements

- It might be possible to go even to $O(N)$
- It was not implemented so I am not sure if it would work and give the same results
- But pointless as this algorithm gives no useful results

Problems with local algorithms

- Initially I created function to calculate "tick distance". This function returned difference if ticks were in the same 64k block, and difference in blocks otherwise.
- However, it lead to sequences (65600, 132000, 512000, ...), so I changed it to return ordinary difference of counter values
- Unfortunately those algorithms do not create long sequences
- They produce many short sequences, 2 - 3 ticks
- Of course there are longer sequences, up to 20 ticks for 1 minute, but those are very infrequent
- There is problem with joining those short sequences — gaps are too big
- They could be changed by using knowledge gathered during developing global algorithms, but I preferred not to return to local ones

New distance in sequence counter function

```
# Assumes a <= b
# Will not work when there is more than 1 overflow
def GetTickDistance(a, b):
    majora = a/65536
    minora = a%65536
    majorb = b/65536
    minorb = b%65536
    return b-a
```

Global algorithm

- Straight lines appear when scatter plot shows about 1 hour of data
- Try to draw those lines in database
- Calculate inside blocks with the same value of high word of counter value
- Take one starting point, and try to compute slope of line from this point to all other ones

Sample query calculating slope of lines

```
SELECT sequence, sequence-65549, time,
extract('epoch' FROM time)-1167223491,
(extract('epoch' FROM time)-1167221093)::float/
(sequence-65557)::float FROM sputnik.sputnik
WHERE sequence < 70000 AND sequence!=65557
ORDER BY sequence;
```

Global algorithm

- Greedy algorithm
- Take histogram of all those slopes (in 0.1 buckets) and choose the largest slope
- Take all points that are on the line with coefficient inside ± 0.3 range of slopes
- Repeat it for all points that are not yet in some sequence
- $O(N^2)$
- As it operates on entire course of 23C3 it finds rather long sequences
- Only about 4000 points (from 11.1M) were left without sequence
- It gives interesting results
- Strange coefficients are found: 2.4, 2.5, 0.1, 0.4, 2.9, 3.2, 3.6, 0.5, 9.9, 10.0, 8.1, ...

Finding sequences in global manner

```
def FindIDs(connection, sa, sz, ta, tz, id):
    SELECT DISTINCT sequence FROM sputnik.sputnik WHERE id IS NULL
    AND sequence BETWEEN %s AND %s ORDER BY sequence
    for s in c.fetchall():
        s0 = s[0]
        SELECT DISTINCT time FROM sputnik
        WHERE id IS NULL AND sequence = %s
        for t in
            t0, hash = t[0], {}
            SELECT DISTINCT ON (sequence, time) time, sequence,
            (extract('epoch' FROM (time-%s)))::float/(sequence-%s)::float
            FROM sputnik.sputnik WHERE id IS NULL AND time > %s AND
            sequence BETWEEN %s AND %s AND sequence != %s
            ORDER BY sequence, time
            for i in  c.fetchall():
                k = int(i[2]*10)
                if 0 < k and k <= 100:
                    hash[k] = hash.get(k, 0)+1
                i = c.fetchone()
            k = -1.0
            if len(hash) > 0:
                m = max(hash.values())
                for i in sorted(hash.keys()):
                    if m == hash[i]:
                        k = float(i)/10.0
                        break
                UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL
                AND sequence = %s AND time = %s
                UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL AND
                sequence BETWEEN %s AND %s AND sequence != %s AND
                (extract('epoch' FROM (time-%s)))::float/(sequence-%s)::float
                BETWEEN %s AND %s
                id += 1
    return id
```

Calling a sequence finder

```
id = (SELECT MAX(id) FROM sputnik.sputnik WHERE id IS NOT NULL)+1
ta = '2006-12-27 12:59:19+01:00'
tz = '2006-12-30 15:59:59+01:00'
id = FindIDs(connection, 0, 2*65536, ta, tz, id)
# Very large data set, 2924448 rows
id = FindIDs(connection, 131072, 196608, '2006-12-27 12:59:19+01:00
id = FindIDs(connection, 131072, 196608, '2006-12-27 18:00:00+01:00
id = FindIDs(connection, 131072, 196608, '2006-12-28 00:00:00+01:00
id = FindIDs(connection, 131072, 196608, '2006-12-28 17:00:00+01:00
id = FindIDs(connection, 131072, 196608, '2006-12-29 00:00:00+01:00
id = FindIDs(connection, 131072, 196608, '2006-12-29 16:00:00+01:00
id = FindIDs(connection, 131072, 196608, '2006-12-30 00:00:00+01:00
# Very large data set, 2076875 rows
id = FindIDs(connection, 3*65535, 4*65536, ta, tz, id)
# Very large data set, 1277488 rows
id = FindIDs(connection, 4*65535, 5*65536, ta, tz, id)
# Very large data set, 1016195 rows
id = FindIDs(connection, 5*65535, 6*65536, ta, tz, id)
# Very large data set, 620763 rows
id = FindIDs(connection, 6*65535, 7*65536, ta, tz, id)
```

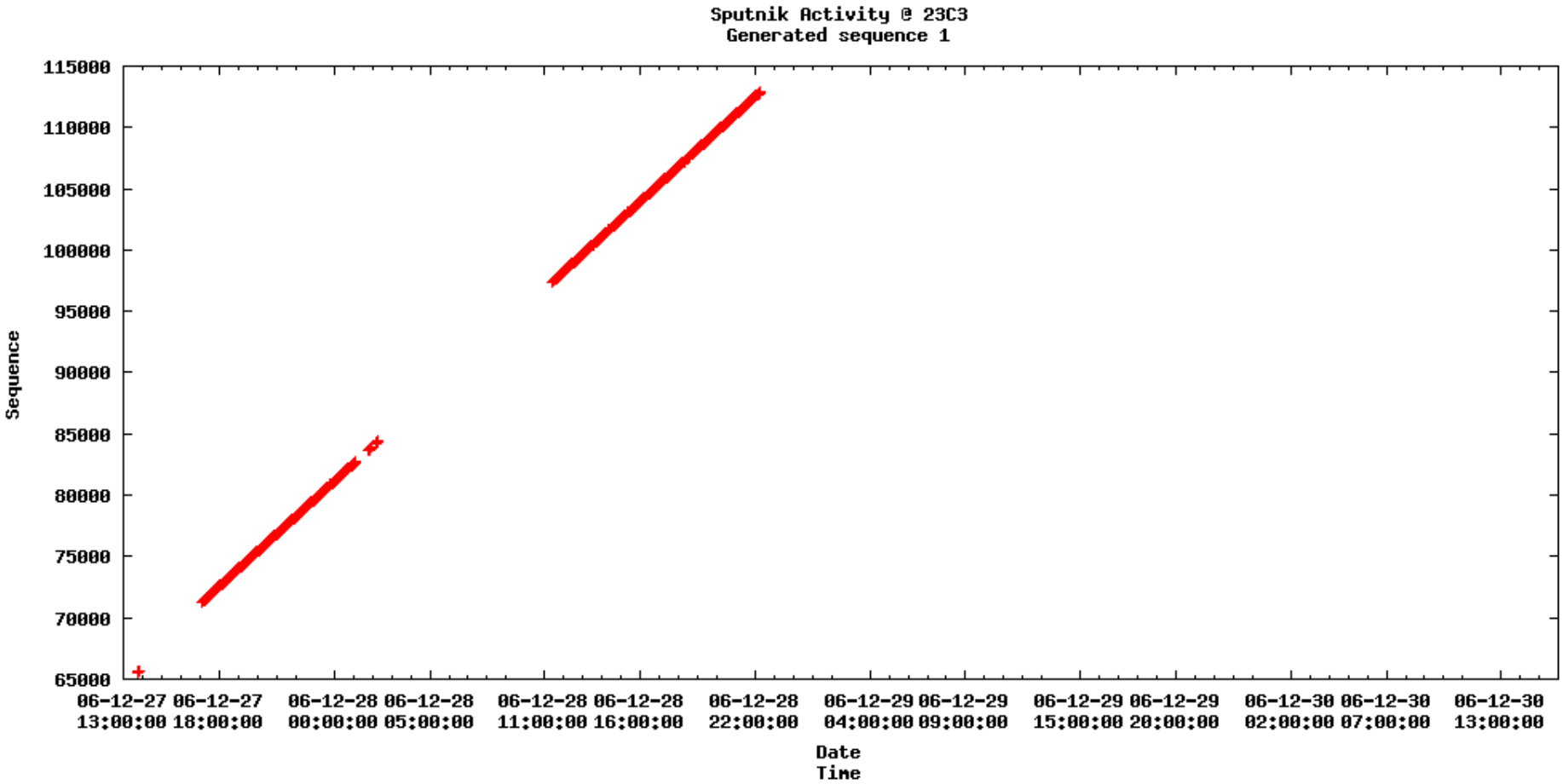


Figure: Generated sequence; first set, number 1

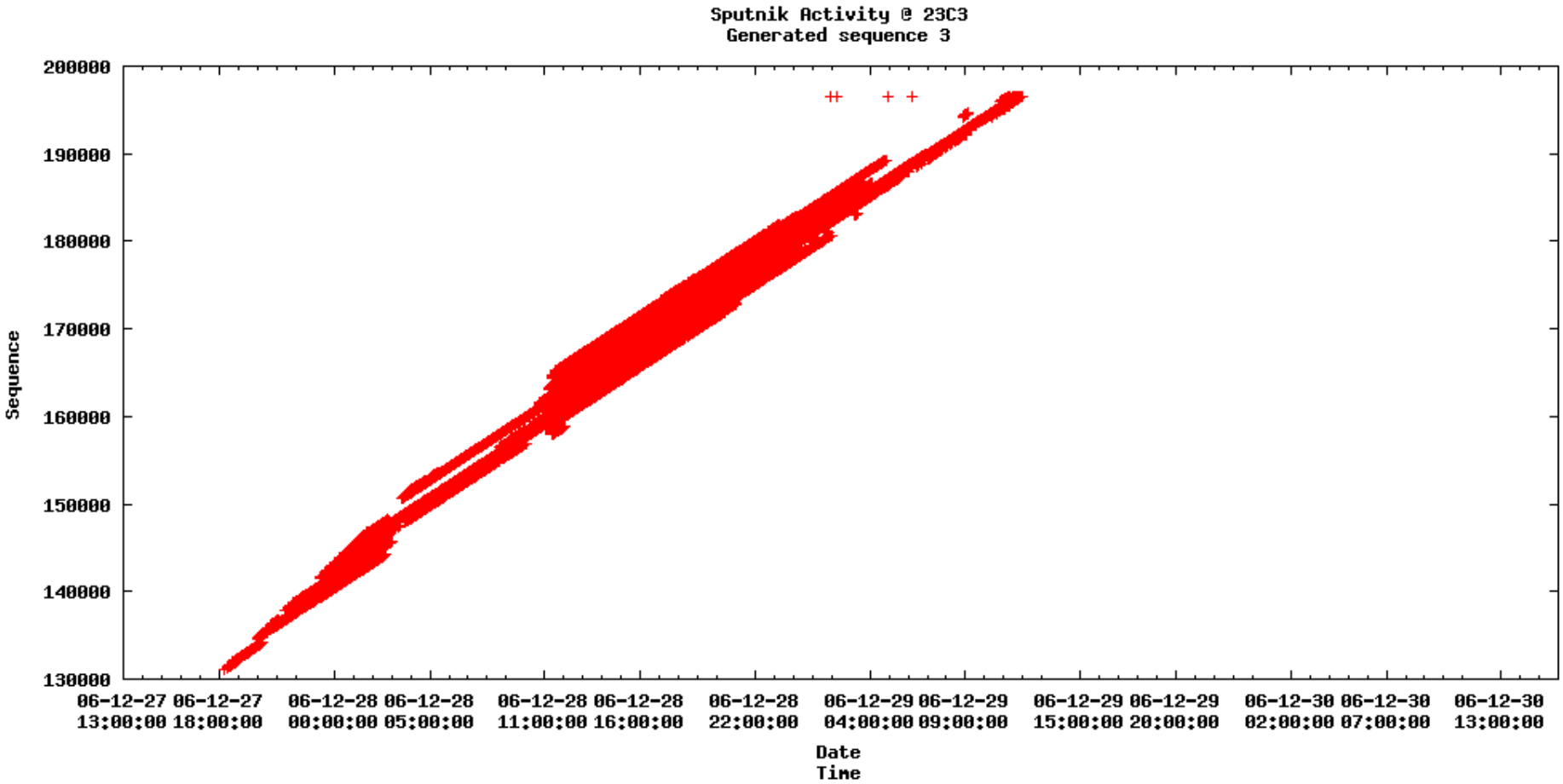
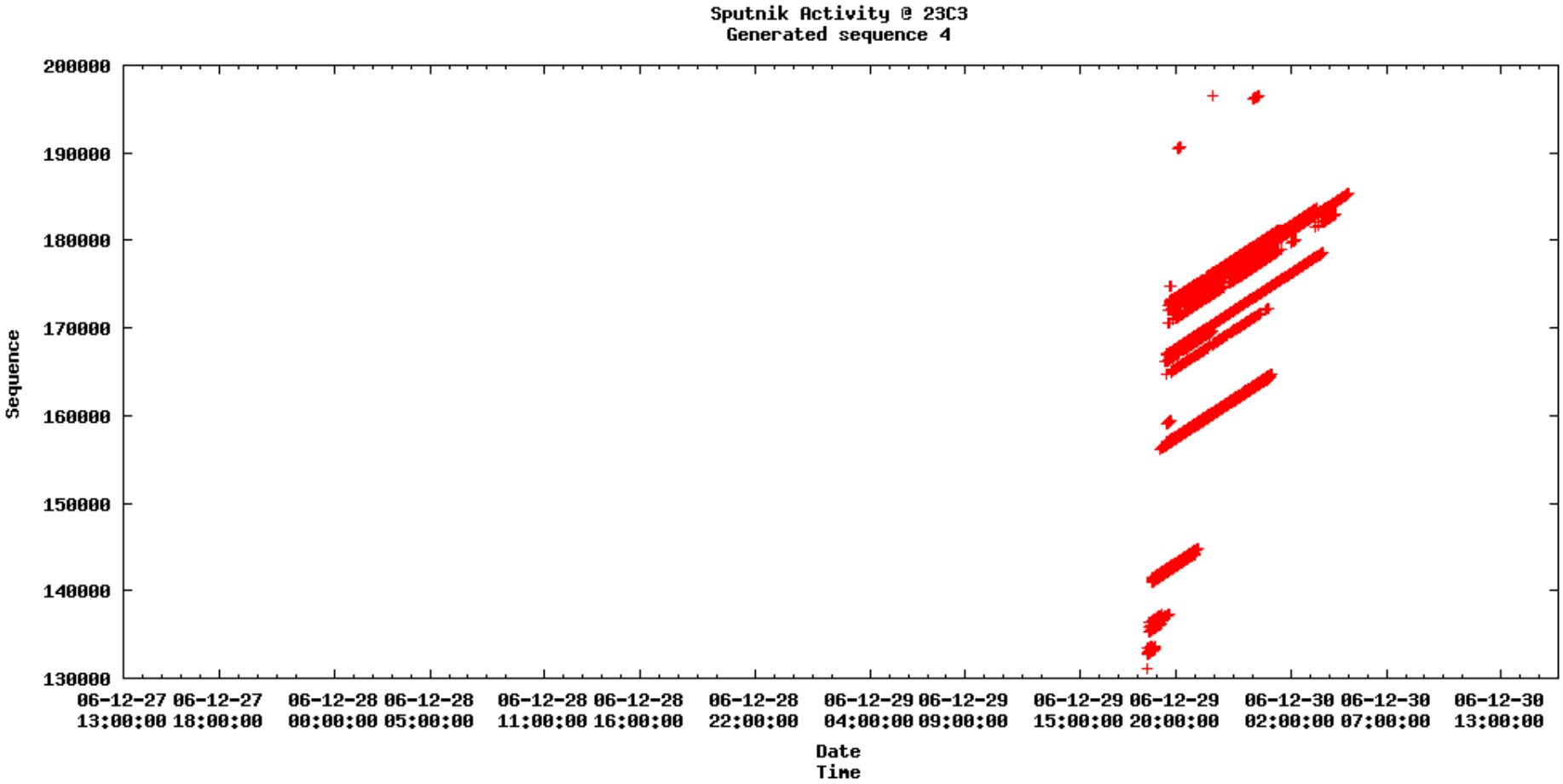


Figure: Generated sequence; first set, number 3

Generated sequence; first set, number 4



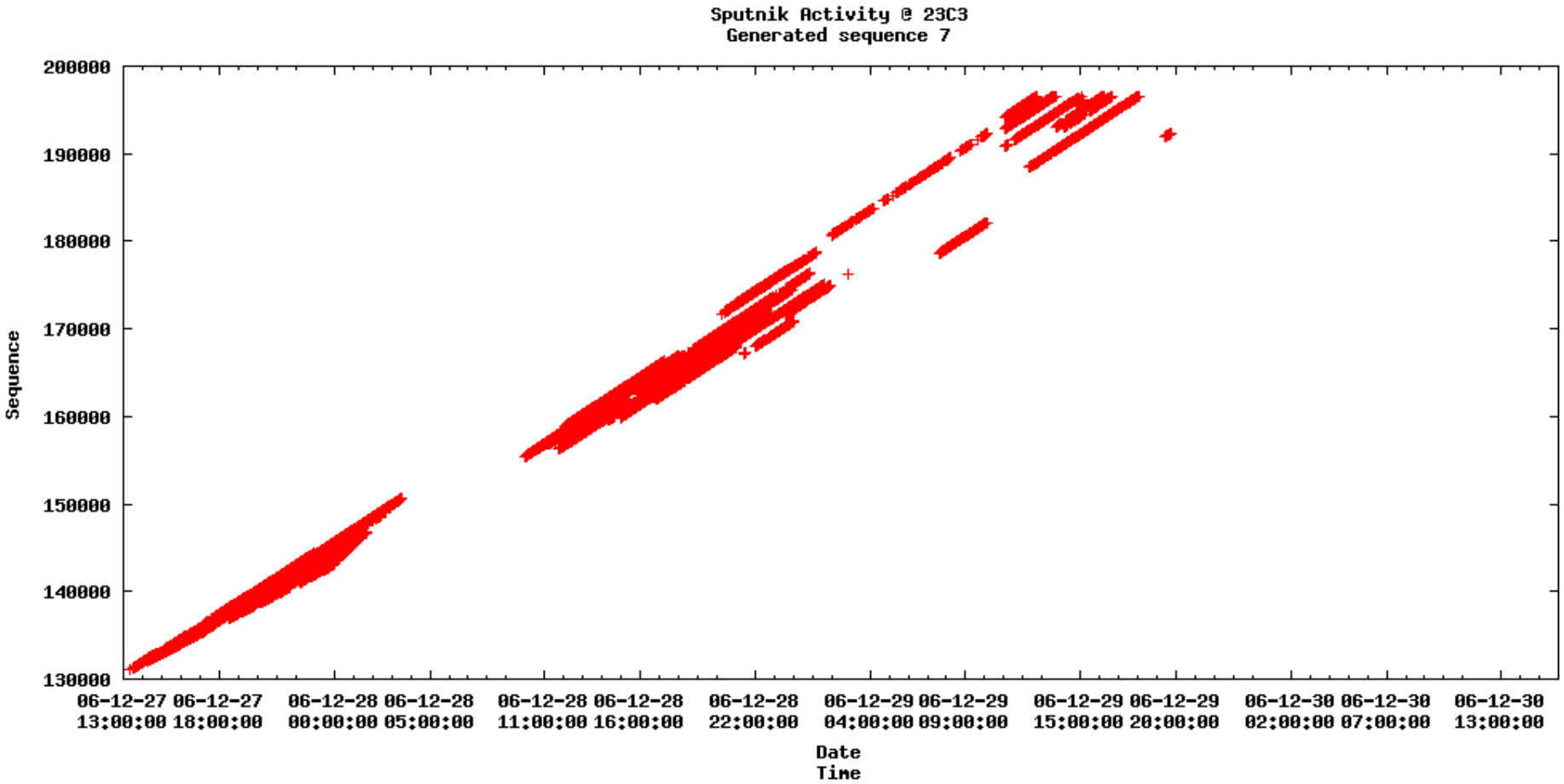


Figure: Generated sequence; first set, number 7

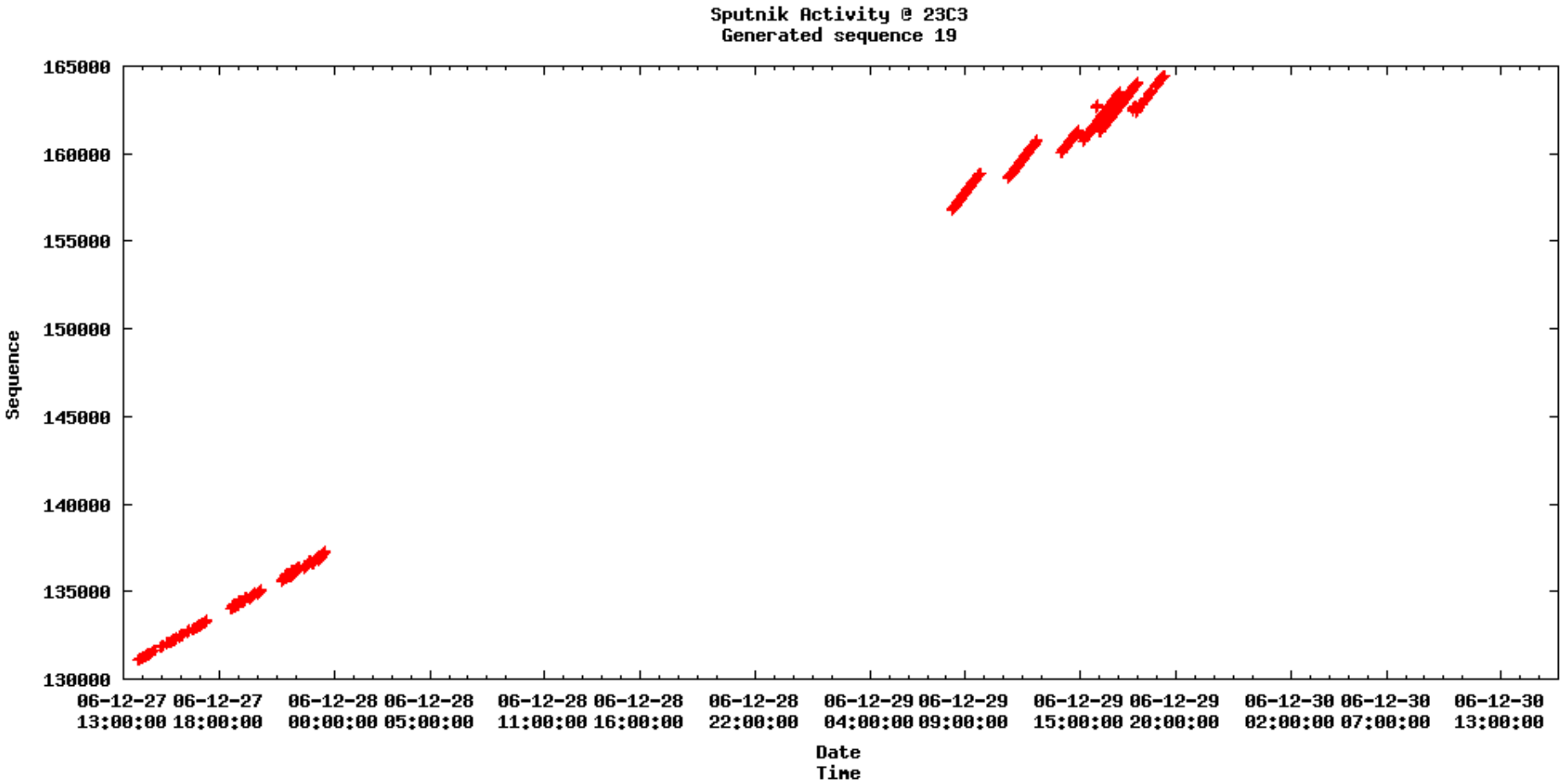
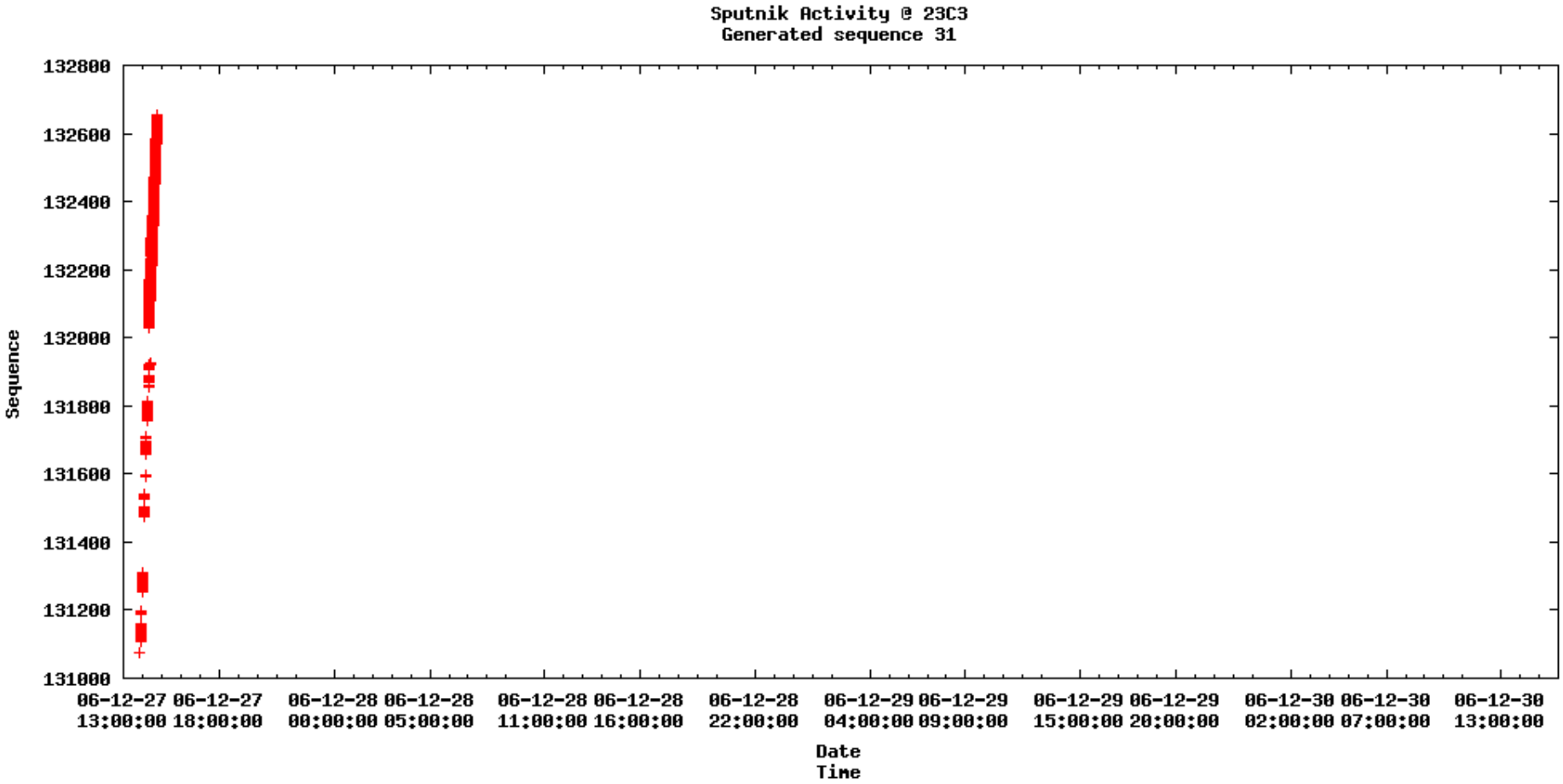


Figure: Generated sequence; first set, number 19

Generated sequence; first set, number 31



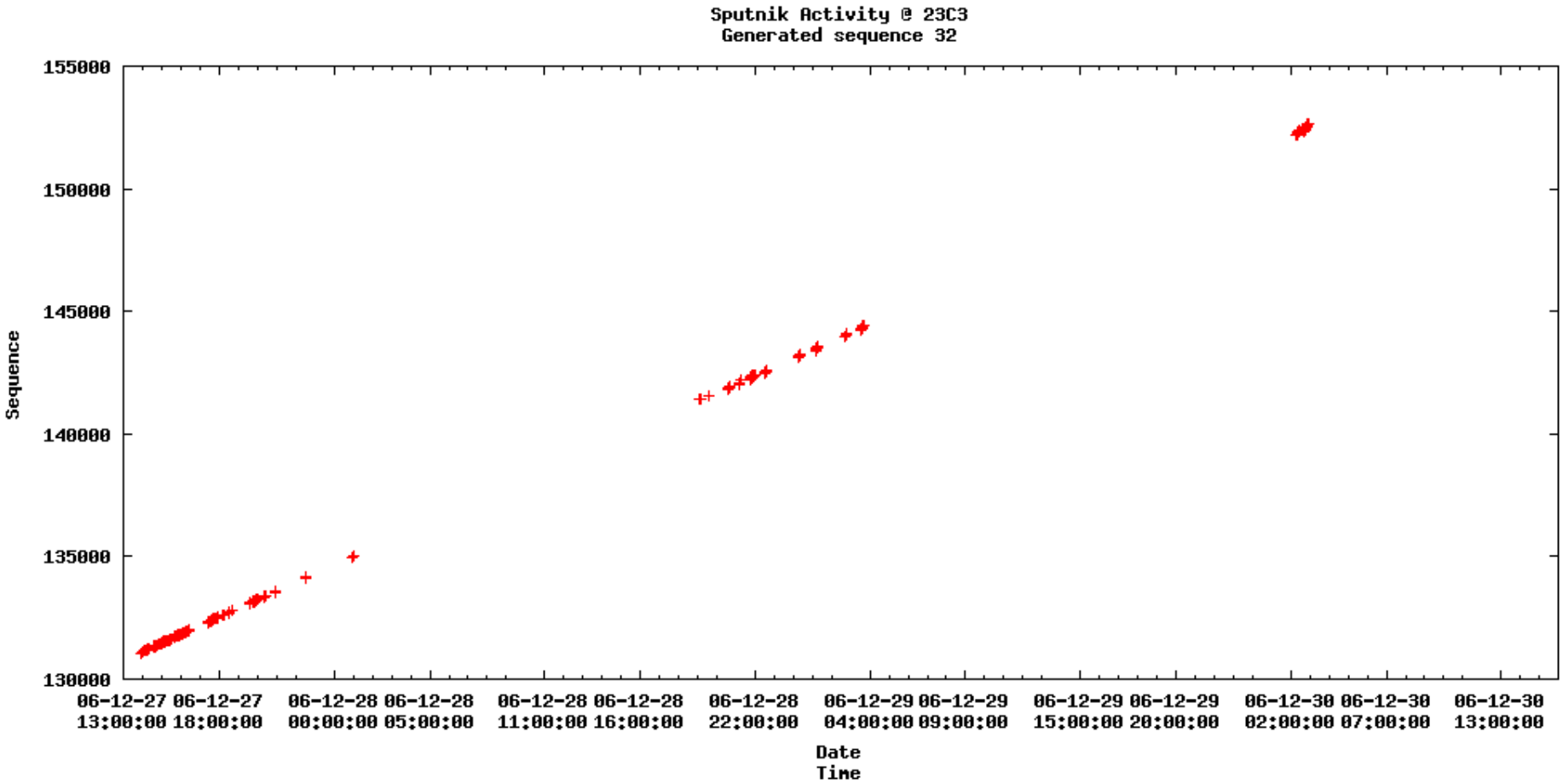
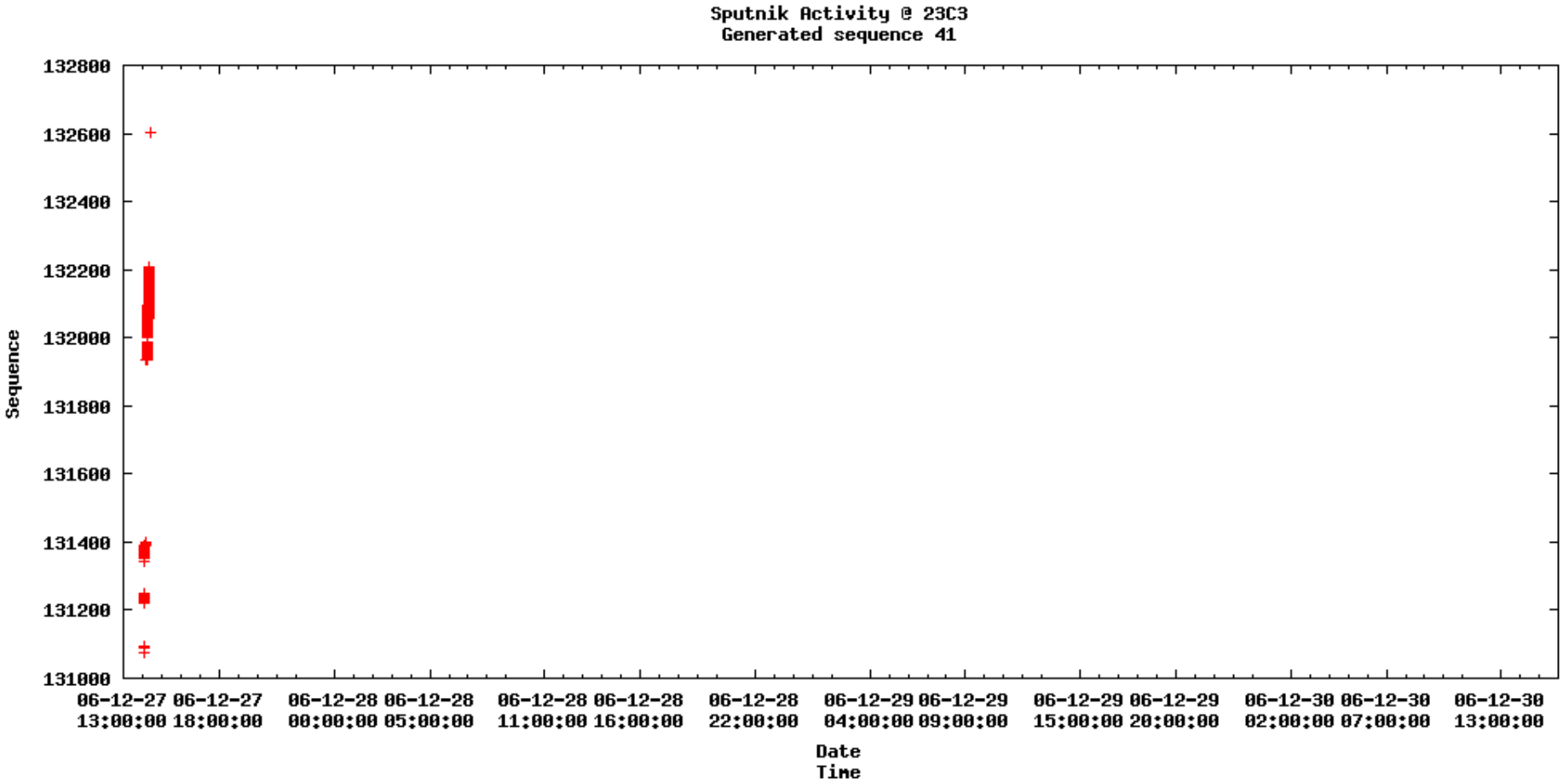
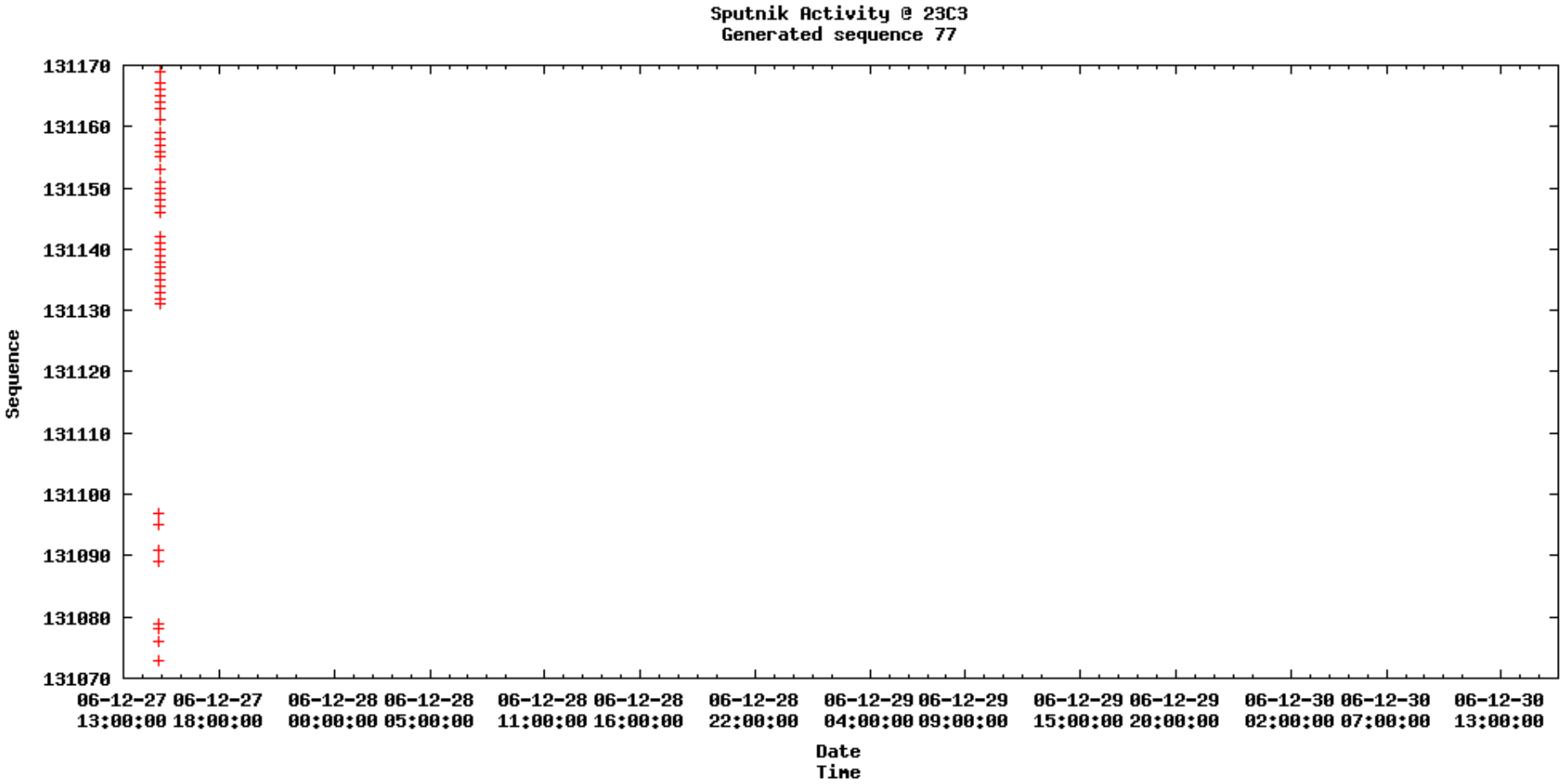


Figure: Generated sequence; first set, number 32

Generated sequence; first set, number 41



Generated sequence; first set, number 77



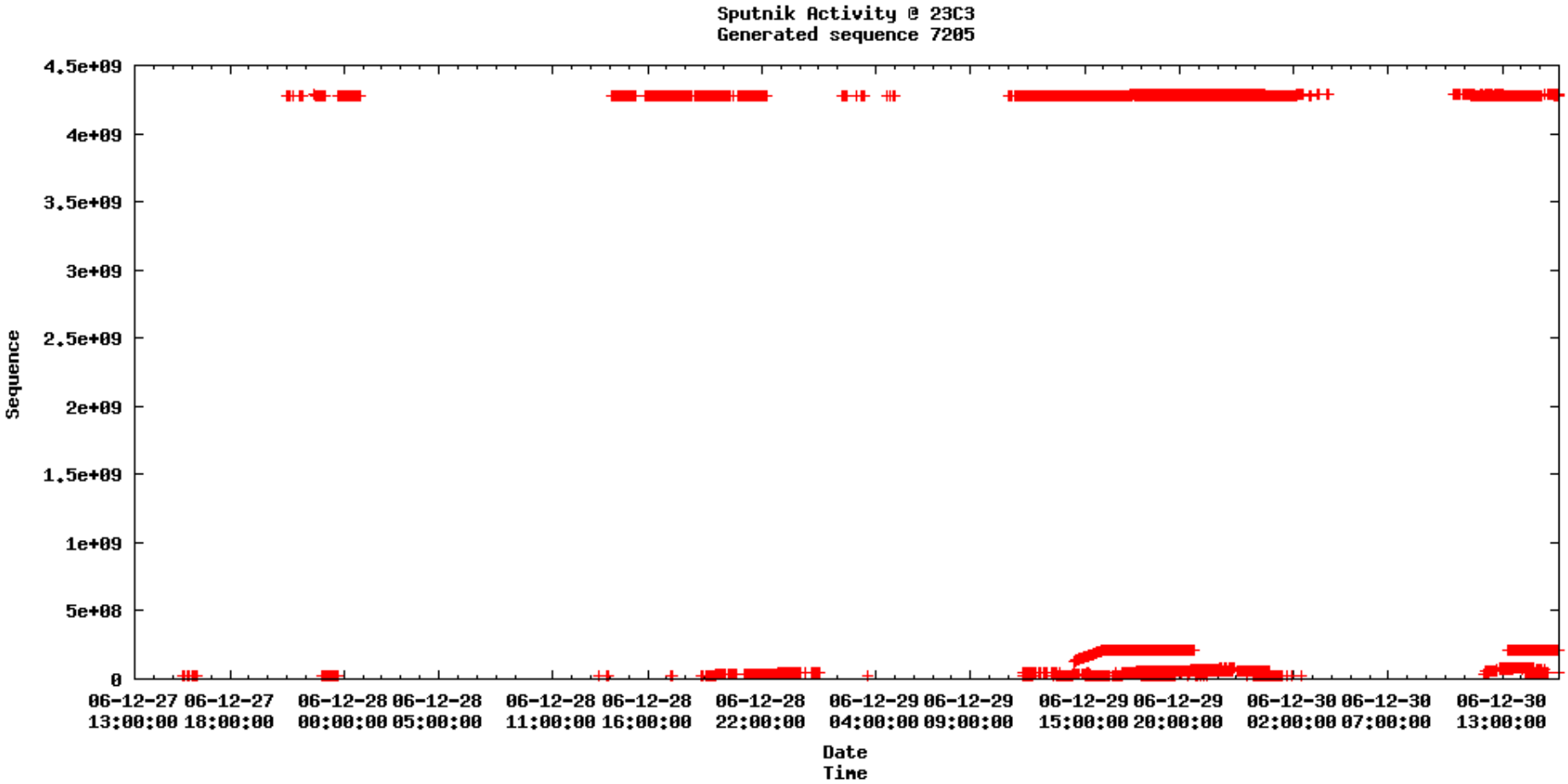


Figure: Generated sequence; first set, number 7205

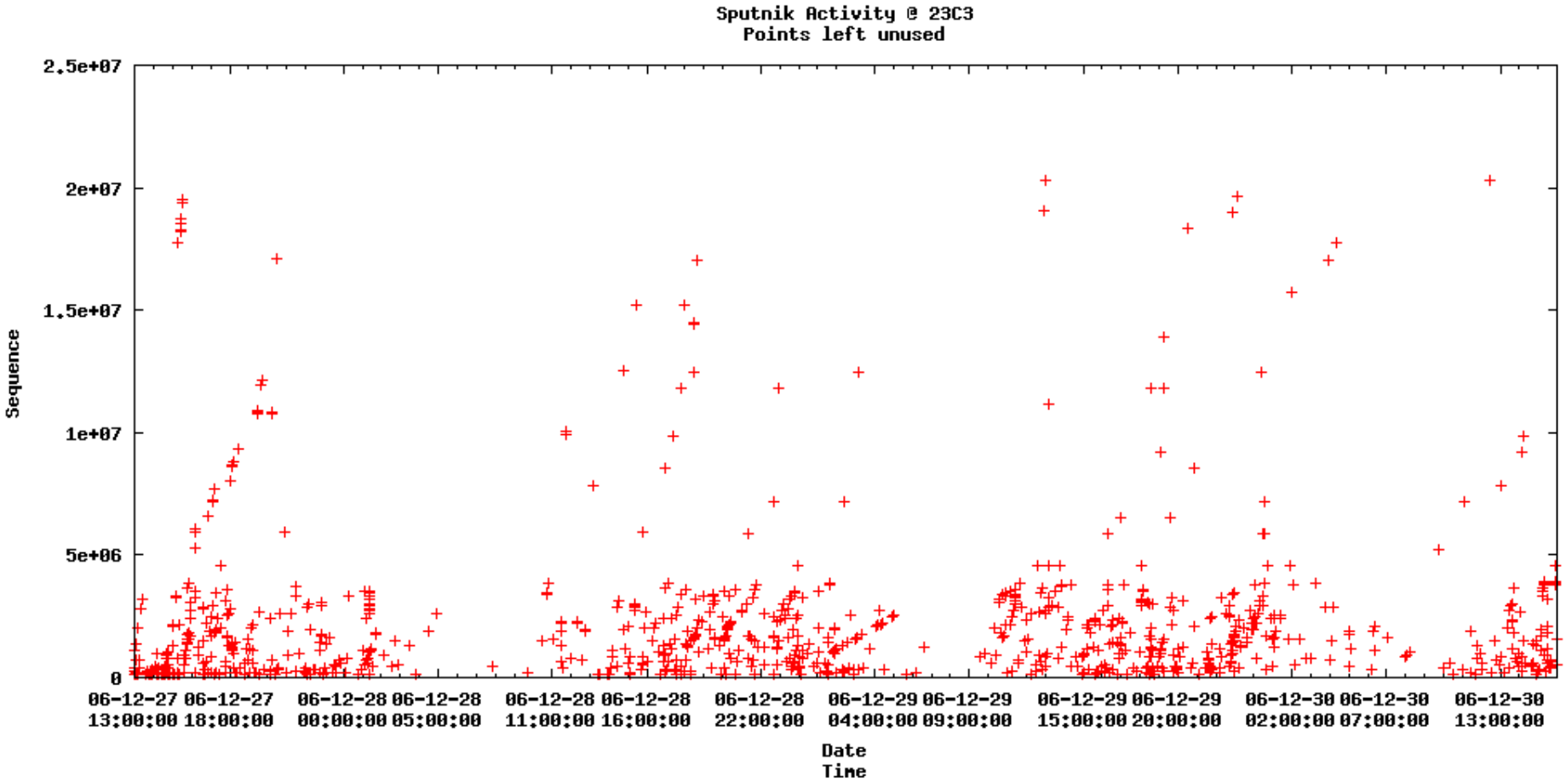


Figure: Points left without sequence; first set

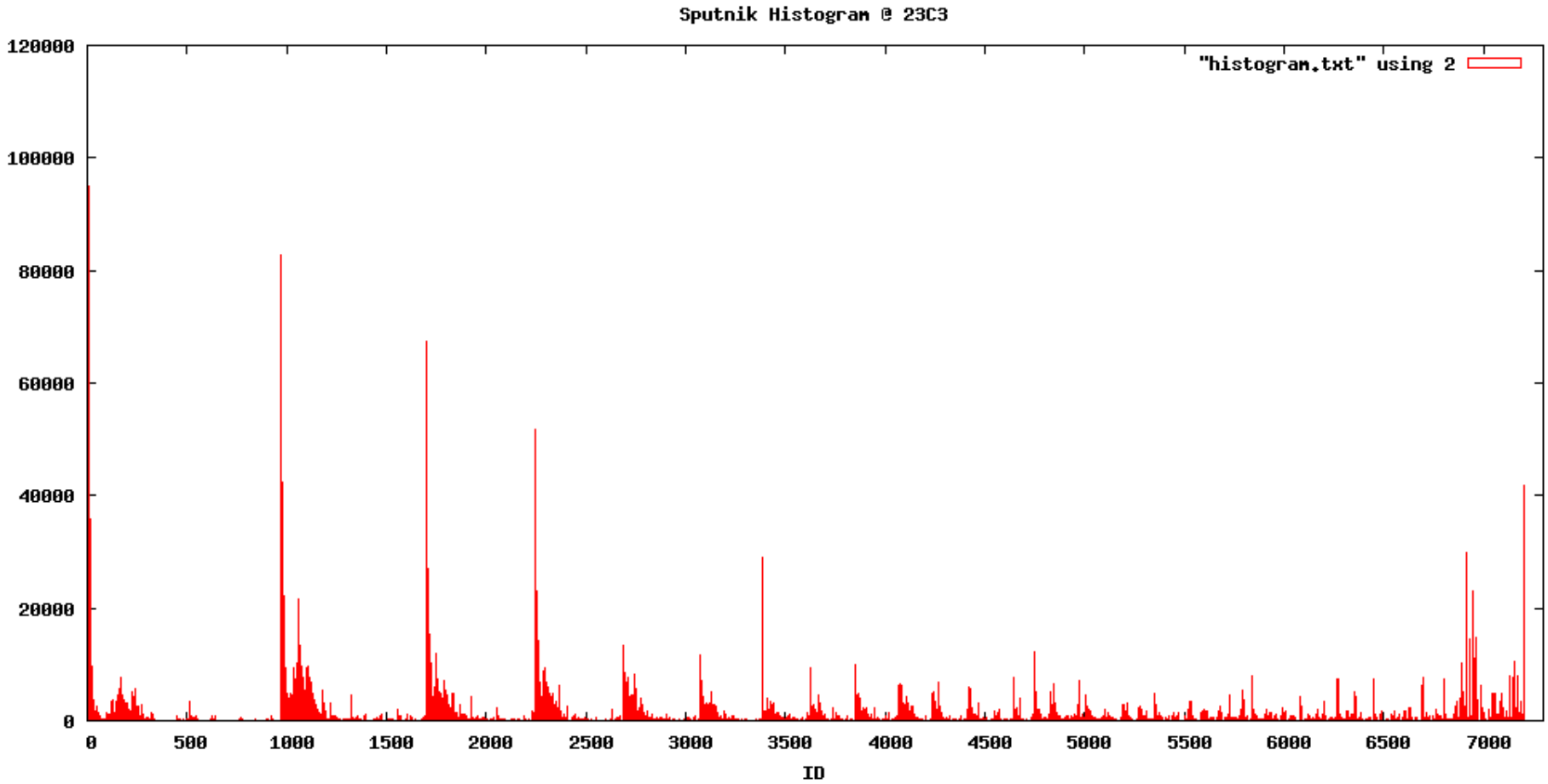


Figure: Histogram of sizes of generated sequences for the first set

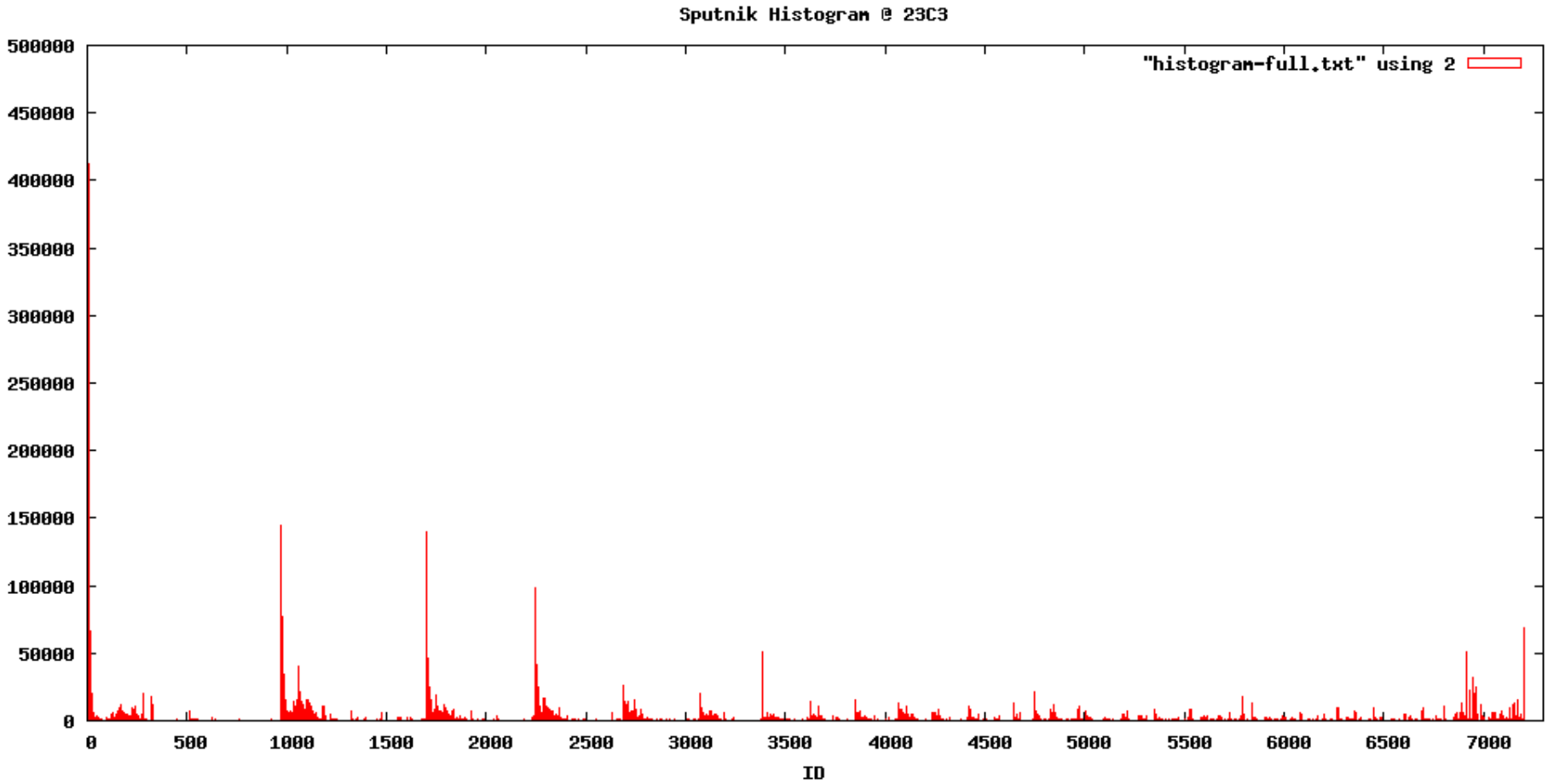


Figure: Histogram of sizes of generated sequences for the first set

Problems

- Rather long running, for entire data set was running for about 72h on Duron 1.3GHz with 768MB RAM and single HDD IDE 7200RPM
- IO-constraint (probably due to my limited RAM)
- Maybe clustering of tables could help, but it would be lost after some updates (PostgreSQL does not try to maintain clustering)
- It uses too wide ranges of allowed coefficients
- It joins more than one line into one sequence, because of big range of coefficients. The more distant from the initial point, the more visible is the problem

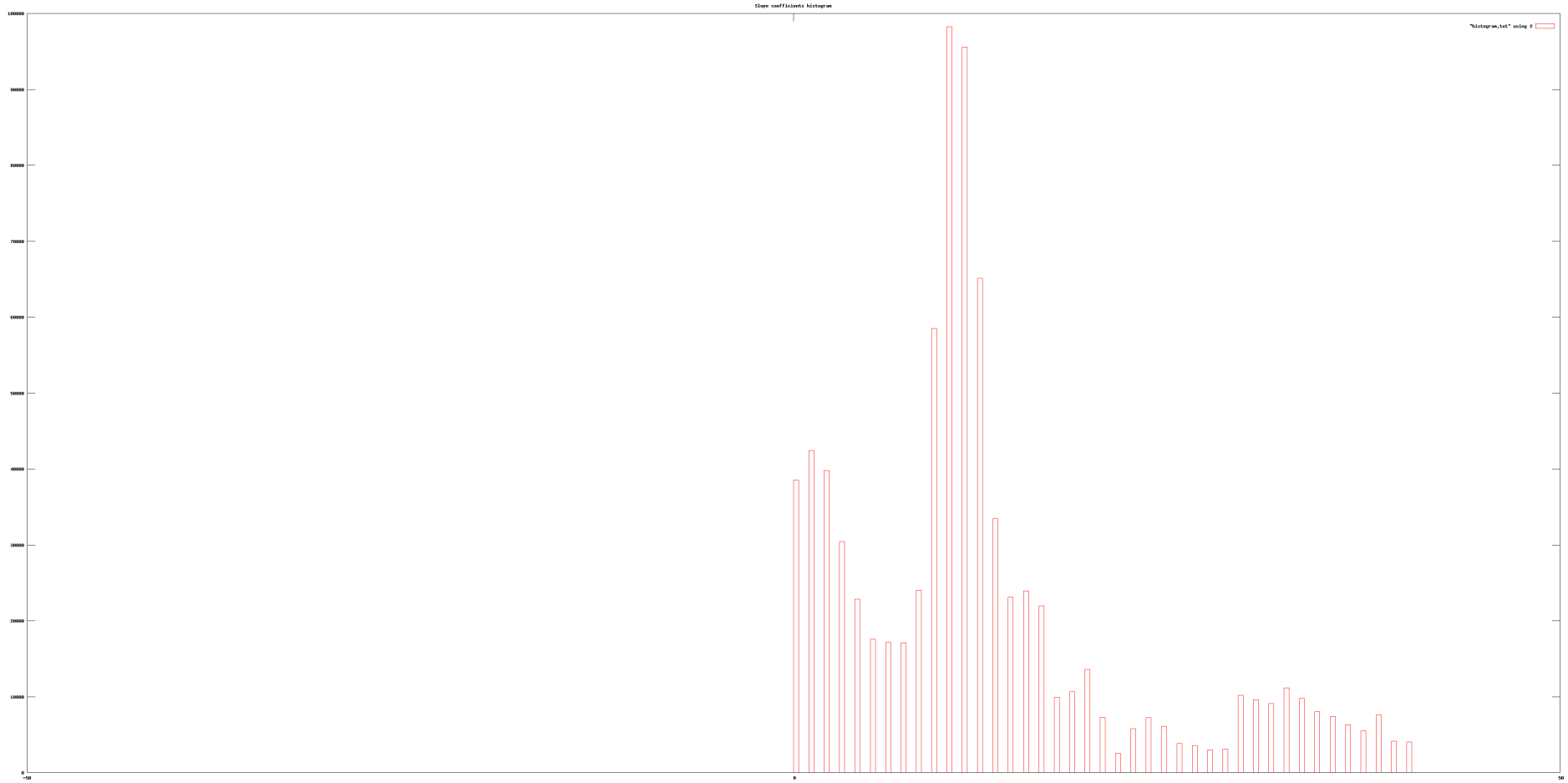


Figure: Coefficients histogram for 10 buckets

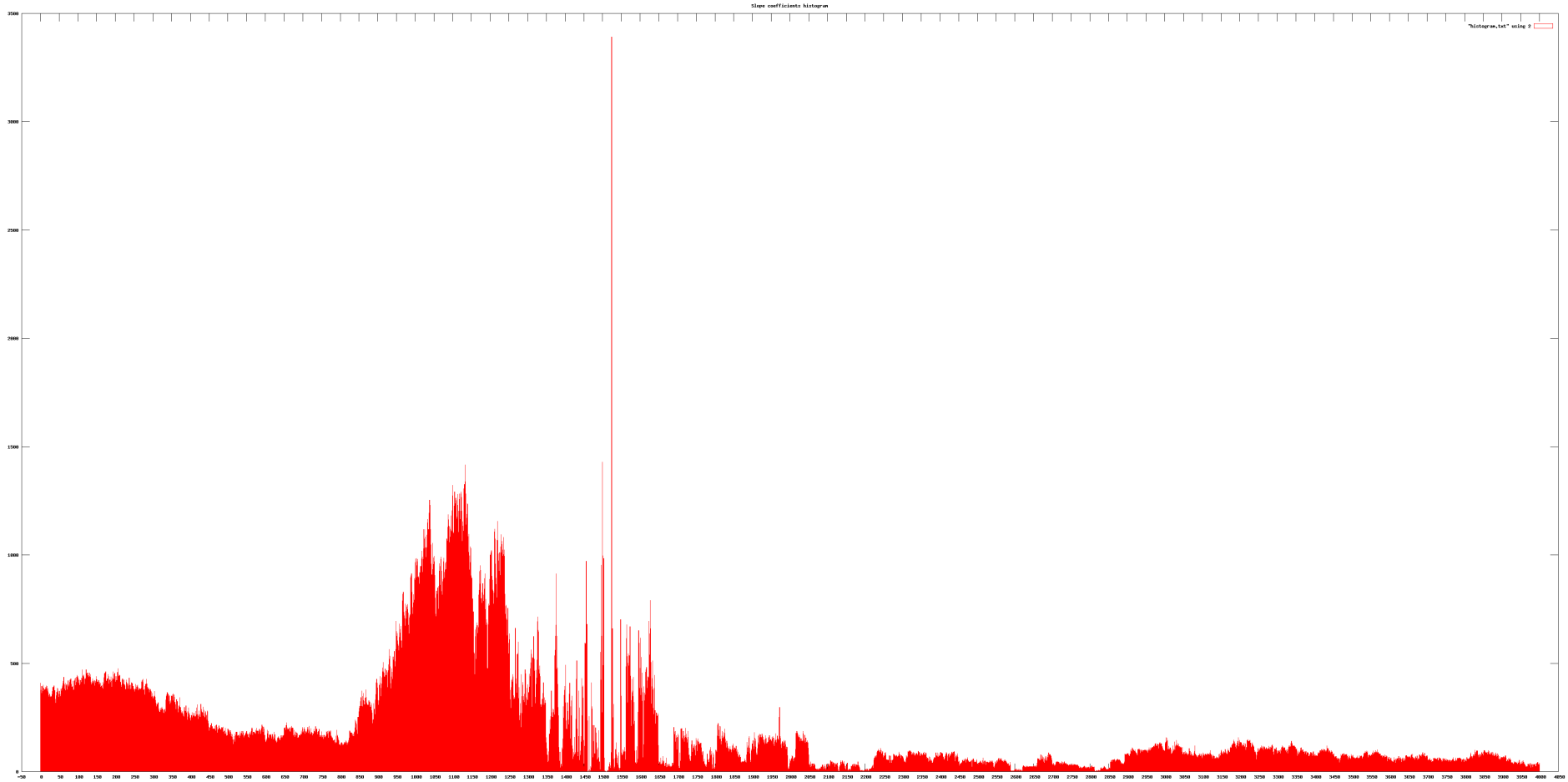


Figure: Coefficients histogram for 1000 buckets

Improvements of global algorithm

- Refactoring; splitting activities into functions
- Added aggregate function to choose only one tick if there is more than one at given second
- Chosen tick is the closest to ideal line
- More buckets in histogram; width 0.001
- Restrict histogram to range 1.0 to 5.0
- 0.001 range of coefficients
- Because of very small range of coefficients, ticks close to the starting point will not be used in generated sequence

Grouping function

```
CREATE OR REPLACE FUNCTION sputnik.guessinit(t TIMESTAMP WITH TIME ZONE, sequence BIGINT, slope DOUBLE PRECISIO
RETURNS VOID
VOLATILE RETURNS NULL ON NULL INPUT SECURITY DEFINER
LANGUAGE 'plpythonu' AS
$$
GD["time"] = t
GD["sequence"] = sequence
GD["slope"] = slope
$$;

CREATE OR REPLACE FUNCTION sputnik.guessbest(state BIGINT, t TIMESTAMP WITH TIME ZONE, sequence BIGINT)
RETURNS BIGINT
VOLATILE CALLED ON NULL INPUT SECURITY DEFINER
LANGUAGE 'plpythonu' AS
$$
if (not GD.has_key("time")) or (not GD.has_key("sequence")) or (not GD.has_key("slope")):
return None
if (t is None) or (sequence is None):
return None

plan = plpy.prepare("""
SELECT (extract('epoch' FROM ($1::TIMESTAMP WITH TIME ZONE-$2::TIMESTAMP WITH TIME ZONE)))::float/($3::BIGINT-$
""", ["timestamptz", "timestamptz", "int8", "int8"])

result = sequence
if state is not None:
r0 = plpy.execute(plan, [t, GD["time"], sequence, GD["sequence"]], 1)
r1 = plpy.execute(plan, [t, GD["time"], state, GD["sequence"]], 1)
if abs(r0[0]["slope"]-GD["slope"]) >= abs(r1[0]["slope"]-GD["slope"]):
result = sequence
else:
```

Histogram function

```
def Histogram(c, time, sequence, sa, sz):
    hash = {}
    c.execute("""SELECT DISTINCT ON (time, sequence) time, sequence,
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float
    FROM sputnik.sputnik WHERE id IS NULL AND
    sequence BETWEEN %s::BIGINT AND %s::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND
    sequence > %s::BIGINT""", (time, sequence, sa, sz, time, sequence))
    i = c.fetchone()
    while i != None:
        k = int(i[2]*1000)
        if 1000 <= k and k <= 5000:
            hash[k] = hash.get(k, 0)+1
        i = c.fetchone()
    if len(hash) > 0:
        m = max(hash.values())
        for i in xrange(1000, 5001):
# Let's take the smallest max
            if m == hash.get(i, 0):
                result = float(i)/1000.0
                break
        return result, m
    else:
        return 0.0, 0
```

Function finding points on line with given slope

```
def Line(c, time, sequence, slope, margin, sa, sz):
    result = [[time, sequence]]
    c.execute("""SELECT sputnik.guessinit(%s::TIMESTAMP WITH TIME ZONE,
    %s::BIGINT, %s::DOUBLE PRECISION)""", (time, sequence, slope))
    c.execute("""SELECT time, sputnik.guesser(time, sequence)
    FROM sputnik.sputnik WHERE id IS NULL AND
    sequence BETWEEN %s::BIGINT AND %s::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND
    sequence > %s::BIGINT AND
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float
    BETWEEN %s::float AND %s::float GROUP BY time
    ORDER BY time""", (sa, sz, time, sequence, time, sequence, slope-margin, slope+margin))
    i = c.fetchone()
    while i != None:
        result.append([i[0], i[1]])
        i = c.fetchone()
    return result
```

Function finding all lines

```
def FindIDs(connection, sa, sz, id):
    c.execute("""SELECT DISTINCT sequence
    FROM sputnik.sputnik WHERE id IS NULL AND
    sequence BETWEEN %s AND %s
    ORDER BY sequence""", (sa, sz))
    start = c.fetchall()
    for s in start:
        s0 = s[0]
        c.execute("""SELECT DISTINCT time FROM sputnik.sputnik
        WHERE id IS NULL AND sequence = %s""", (s0,))
        for t in c.fetchall():
            t0 = t[0]
            slope, count = Histogram(c, t0, s0, sa, sz)
            if slope > 0.0 and count >= 8:
                line = Line(c, t0, s0, slope, 000.1, sa, sz)
                for i in line:
                    UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL AND
                    time = %s::TIMESTAMP WITH TIME ZONE AND
                    sequence = %s::BIGINT
                id += 1
    return id
```

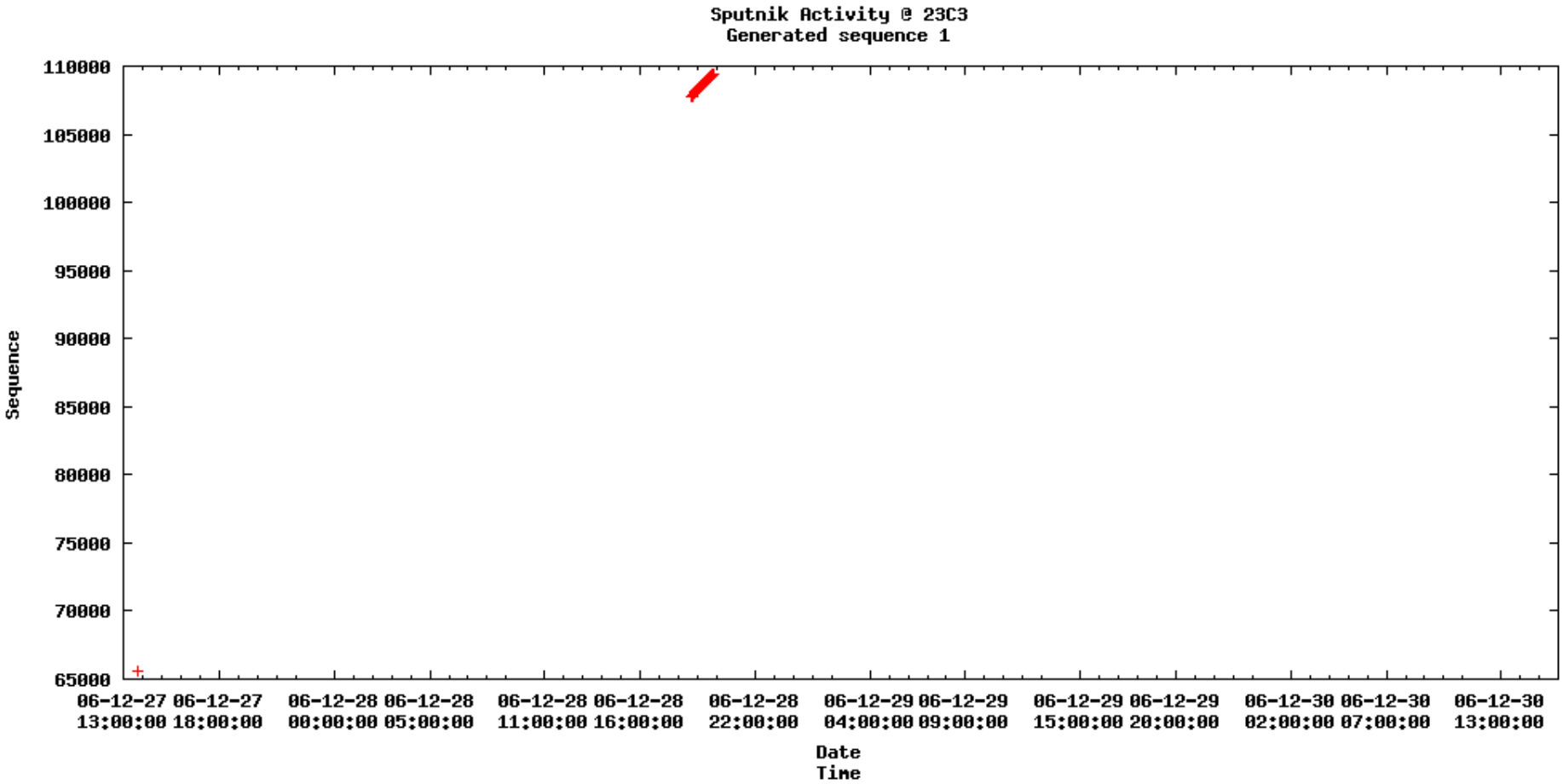
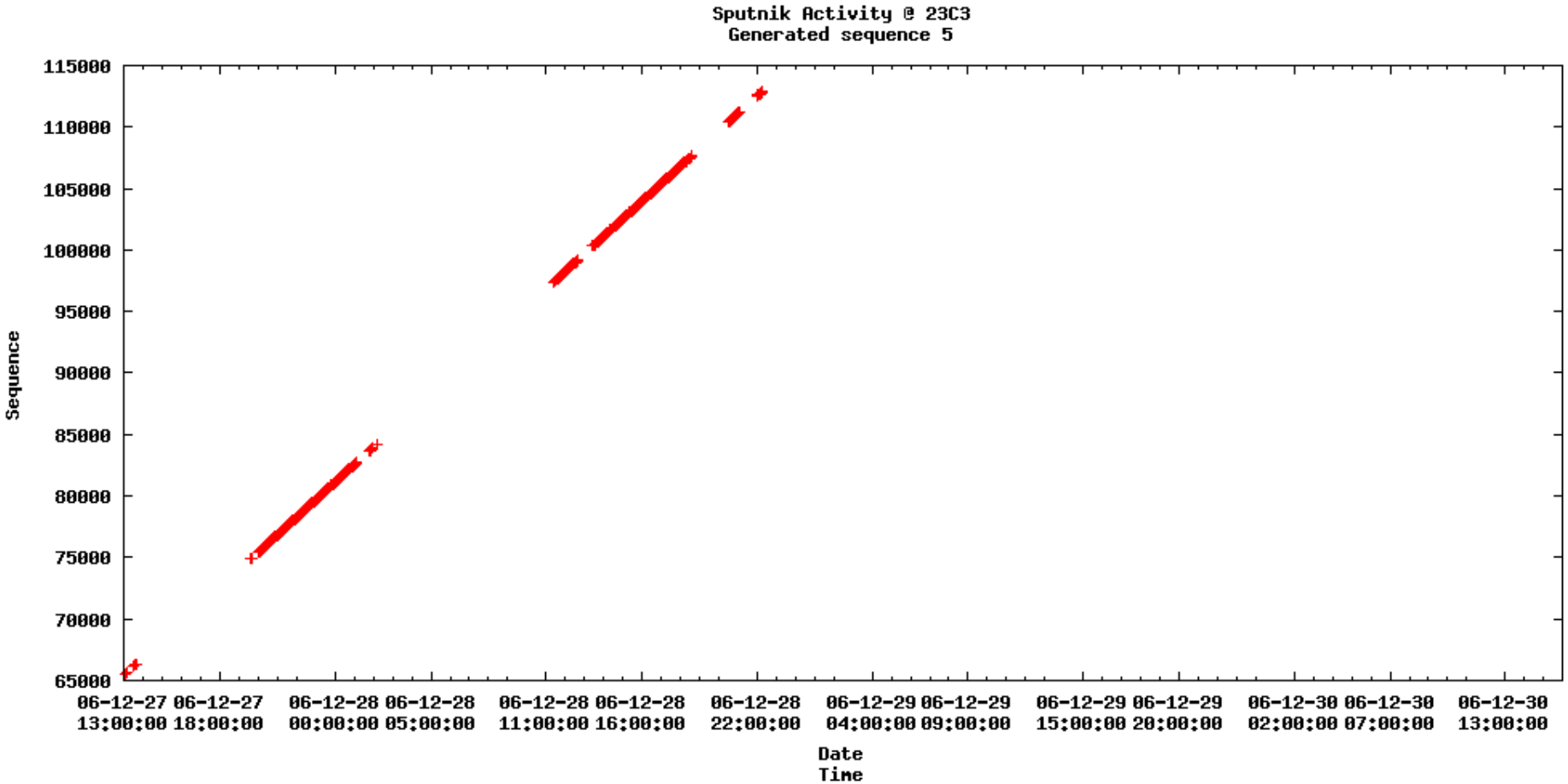
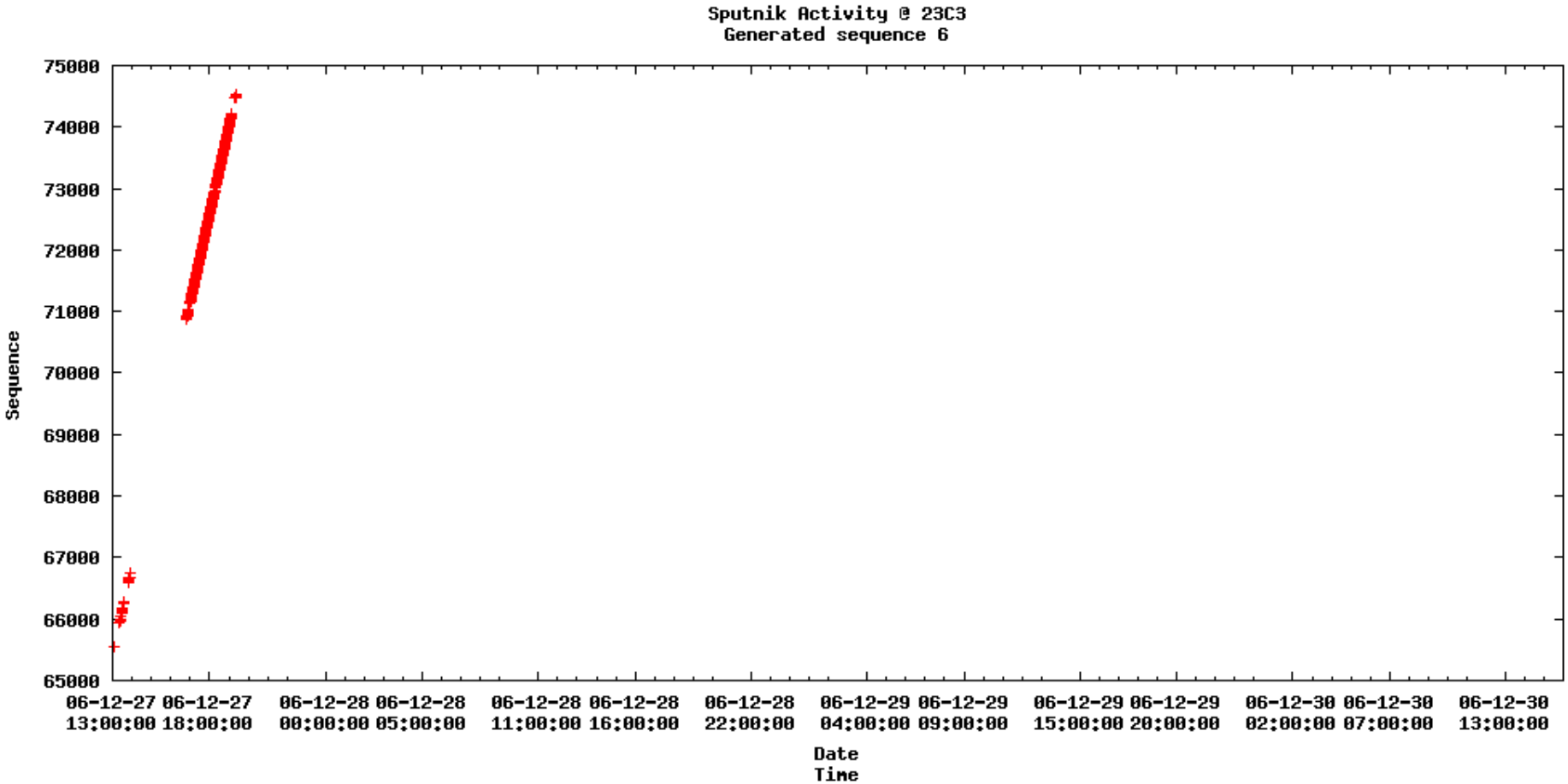


Figure: Generated sequence; second set, number 1

Generated sequence; second set, number 5



Generated sequence; second set, number 6



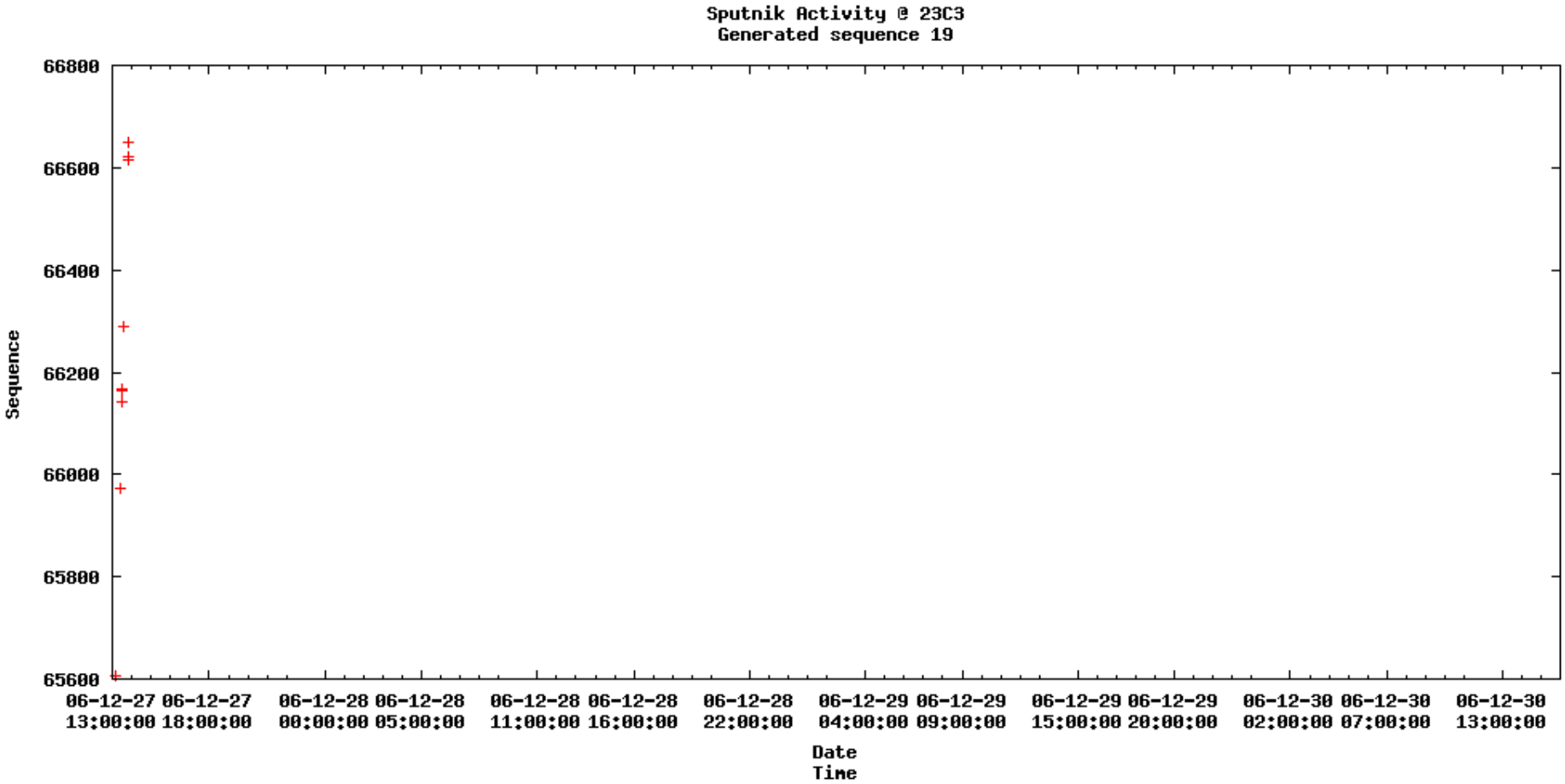
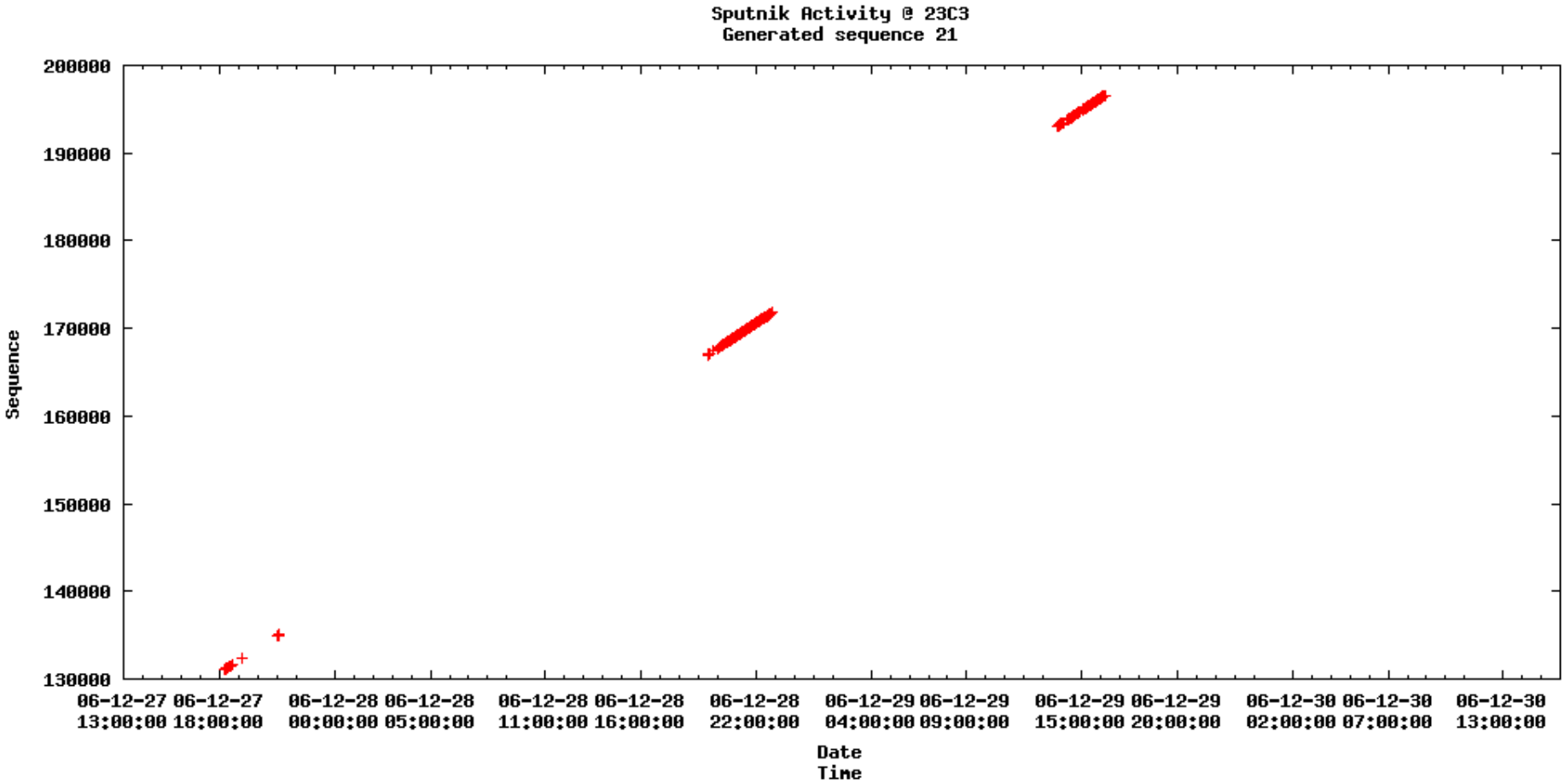
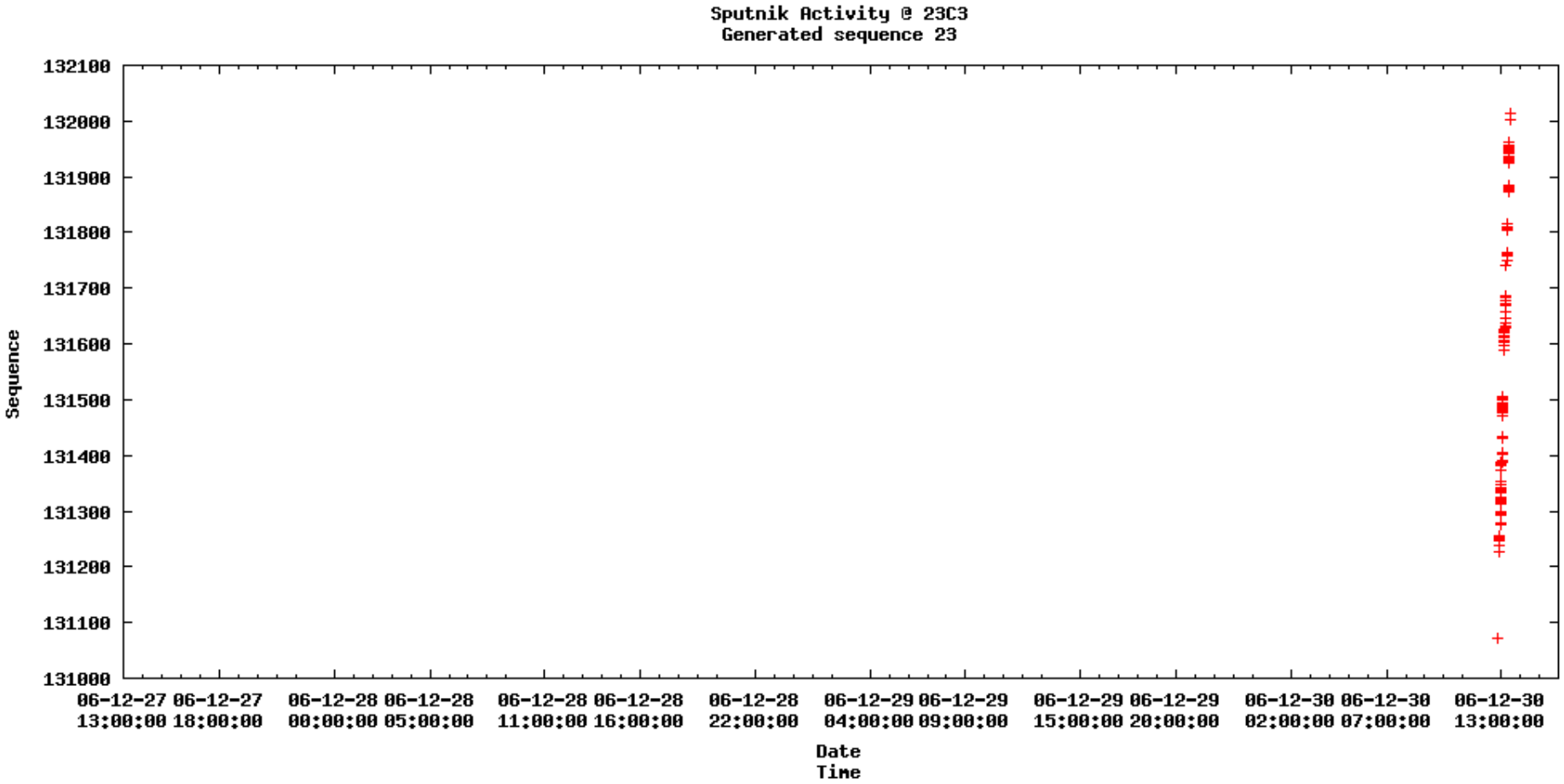


Figure: Generated sequence; second set, number 19

Generated sequence; second set, number 21



Generated sequence; second set, number 23



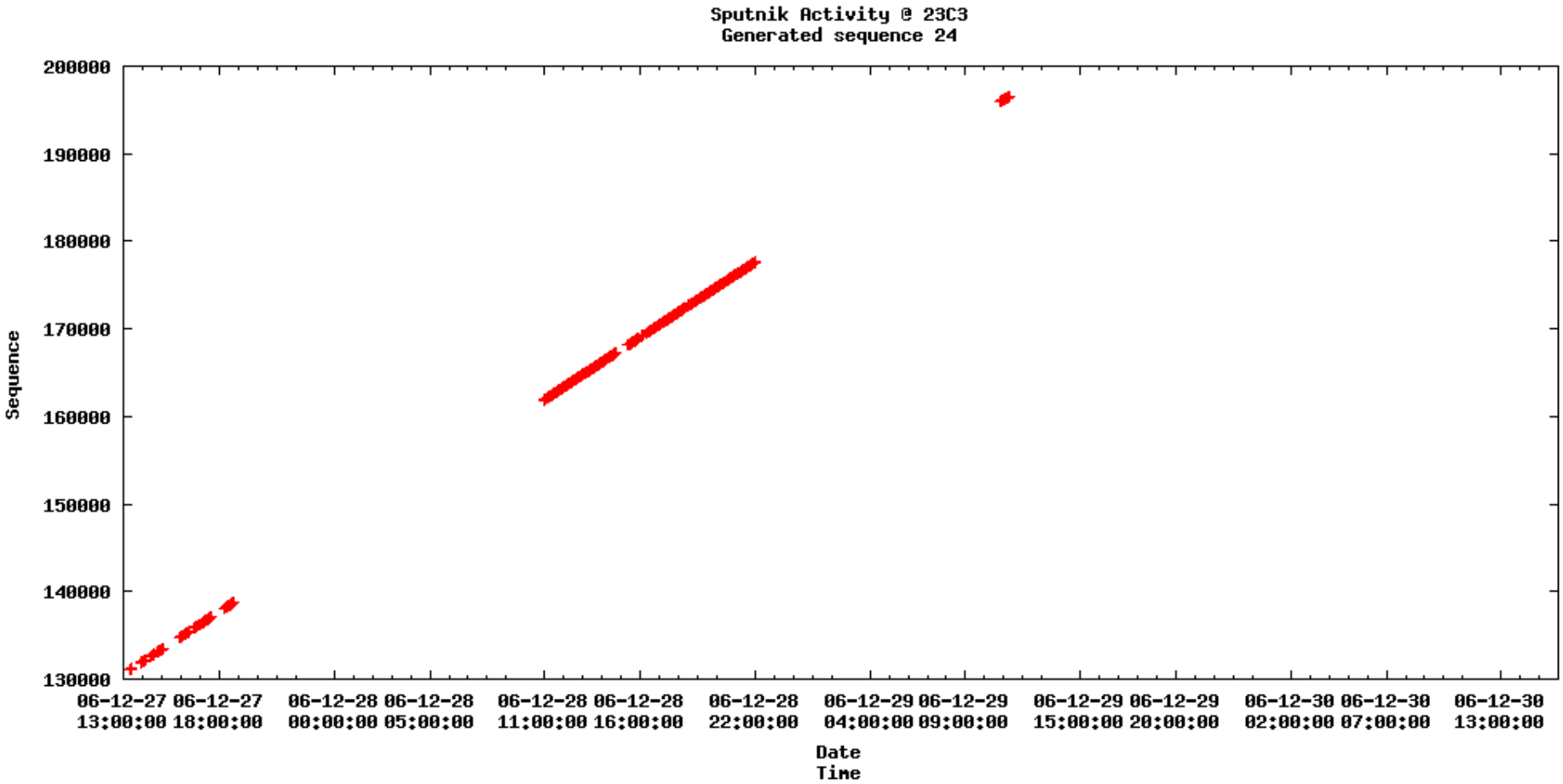
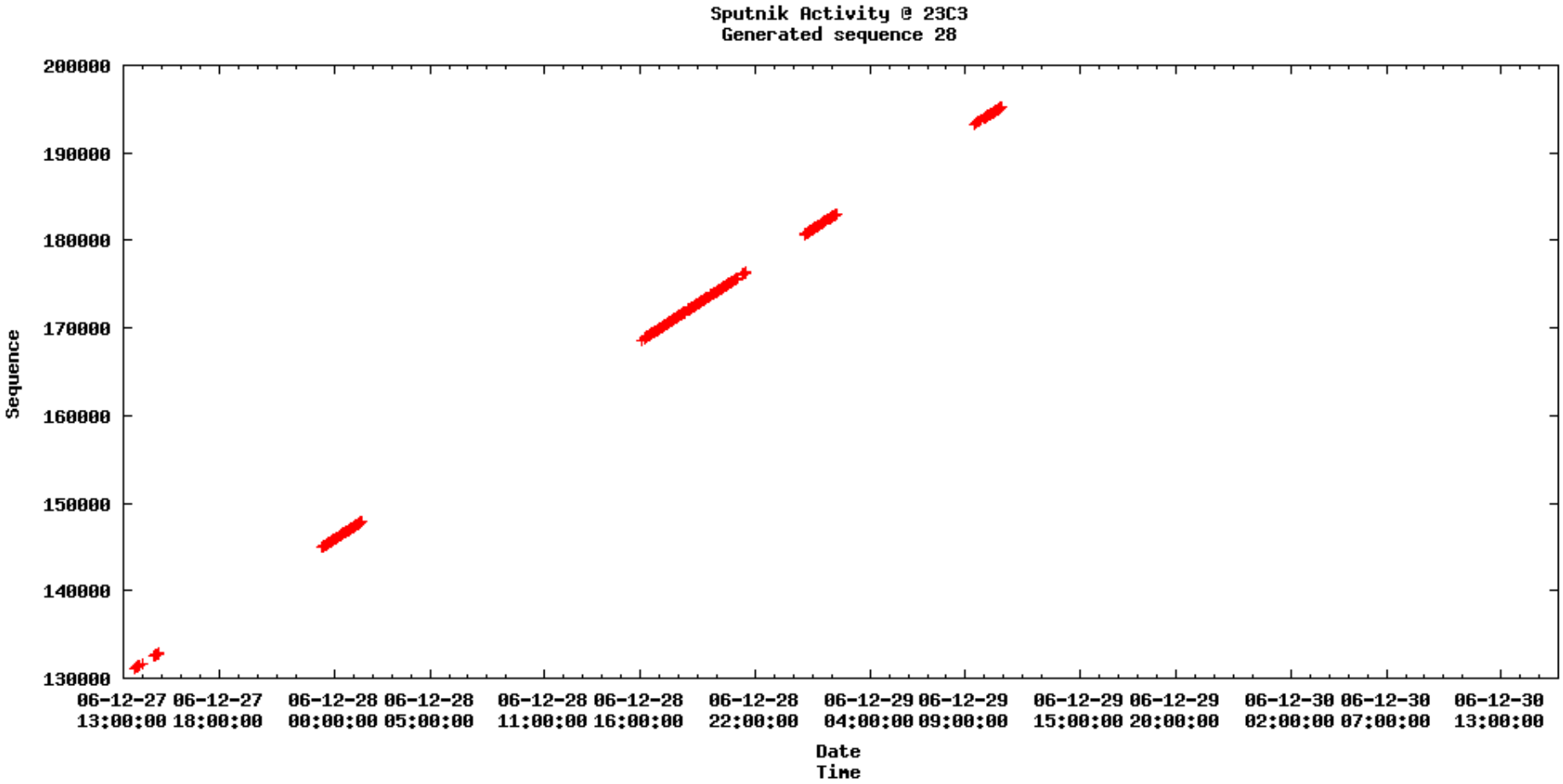
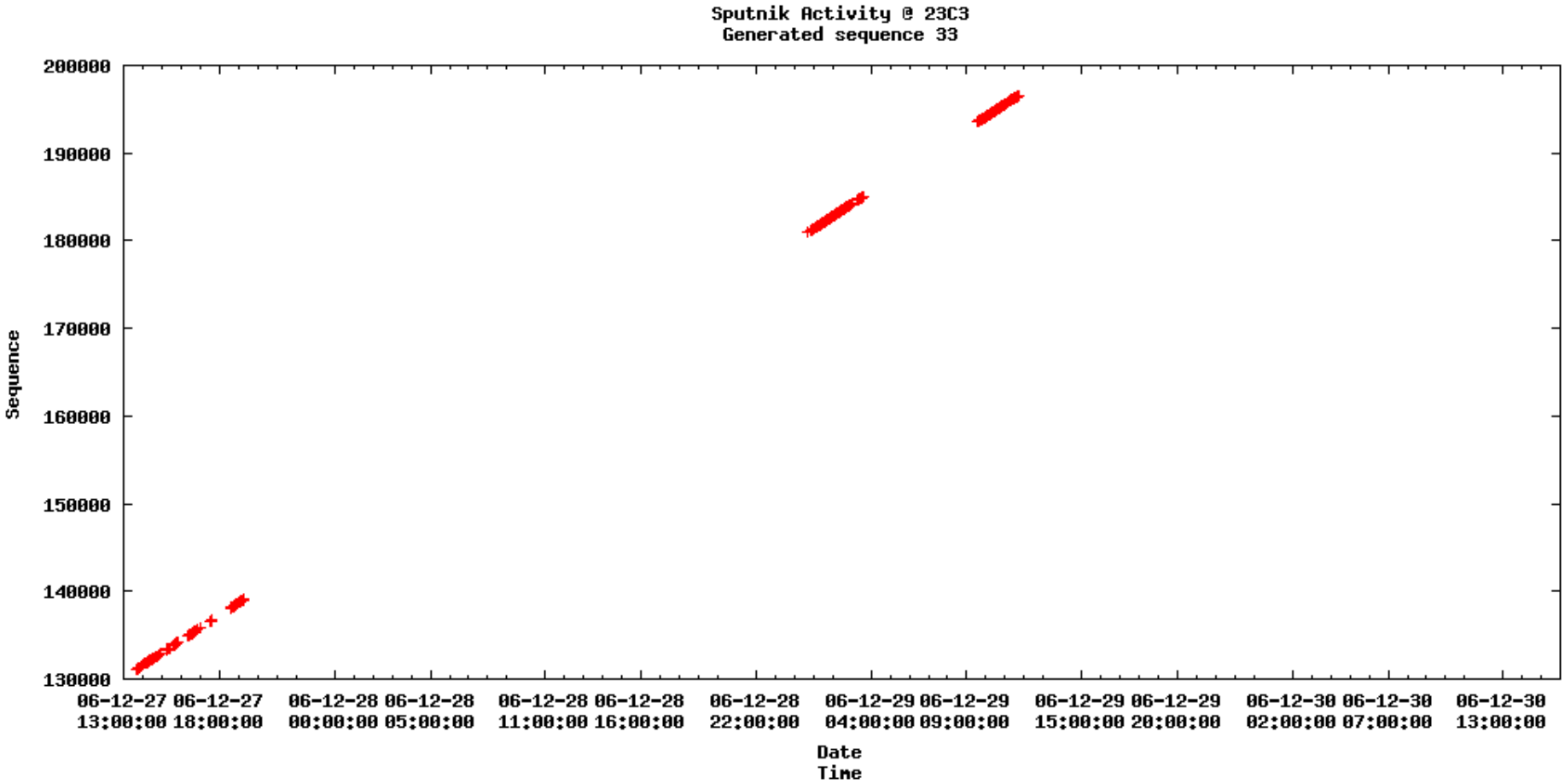


Figure: Generated sequence; second set, number 24

Generated sequence; second set, number 28



Generated sequence; second set, number 33



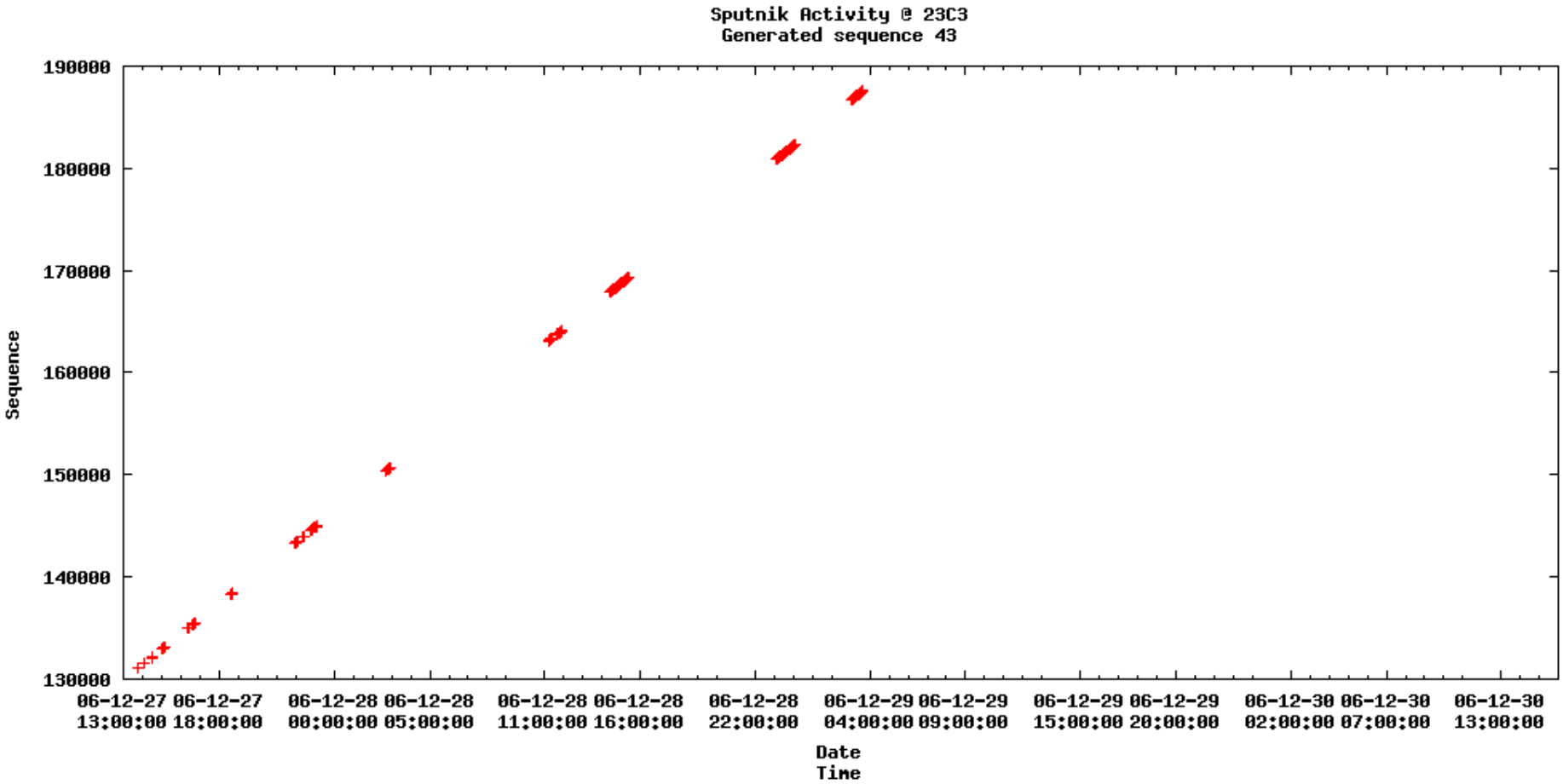


Figure: Generated sequence; second set, number 43

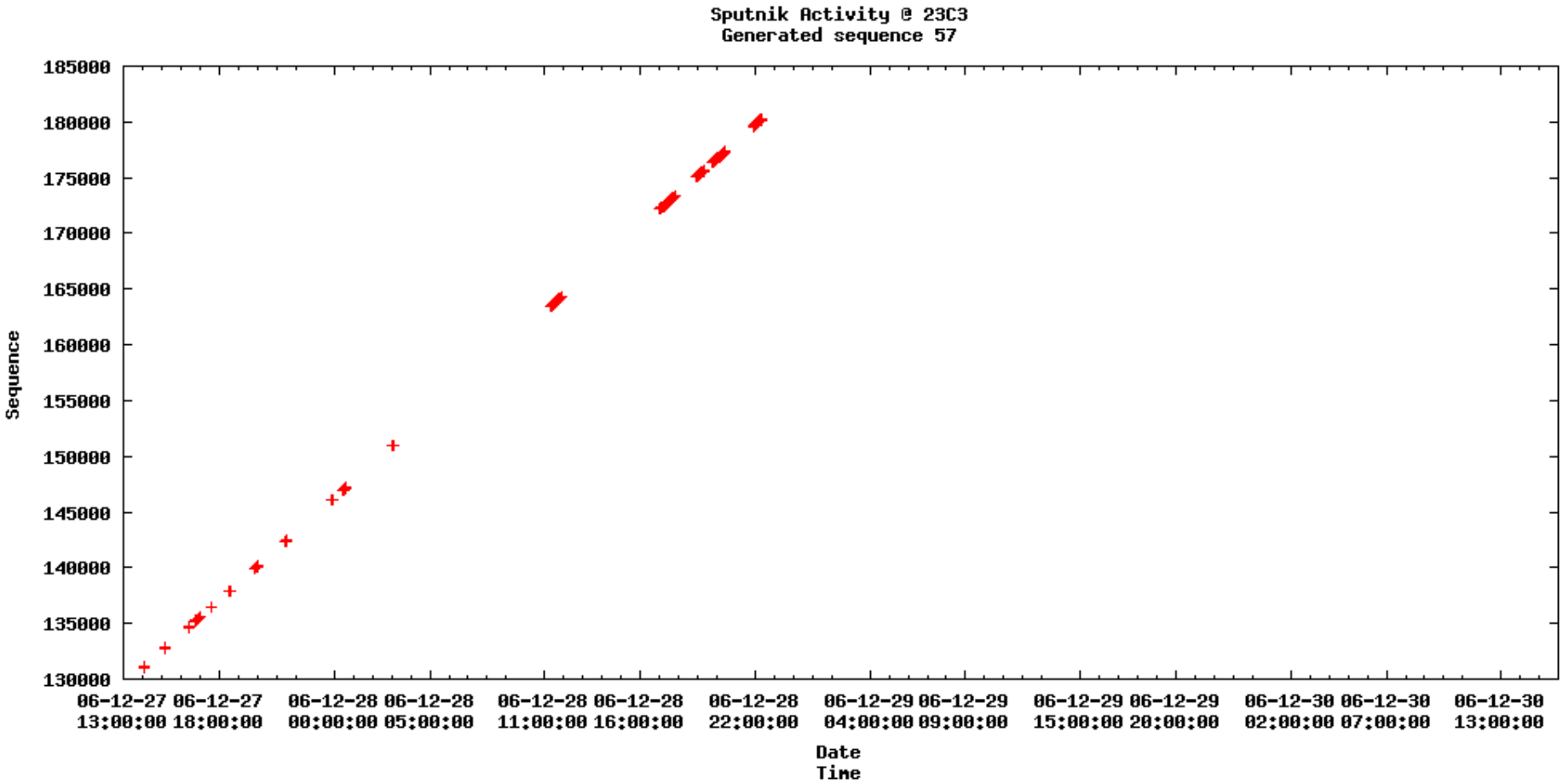
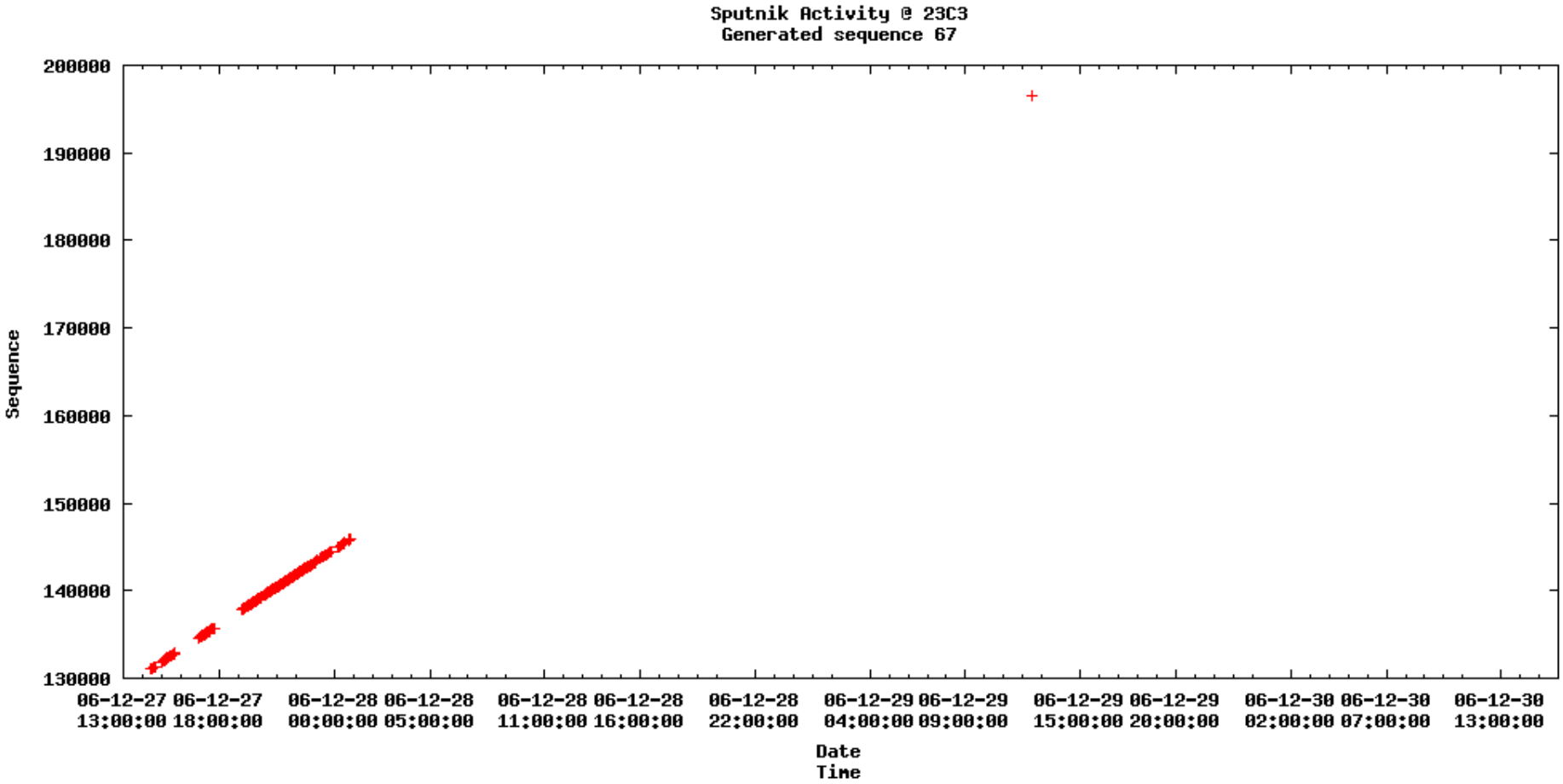


Figure: Generated sequence; second set, number 57

Generated sequence; second set, number 67



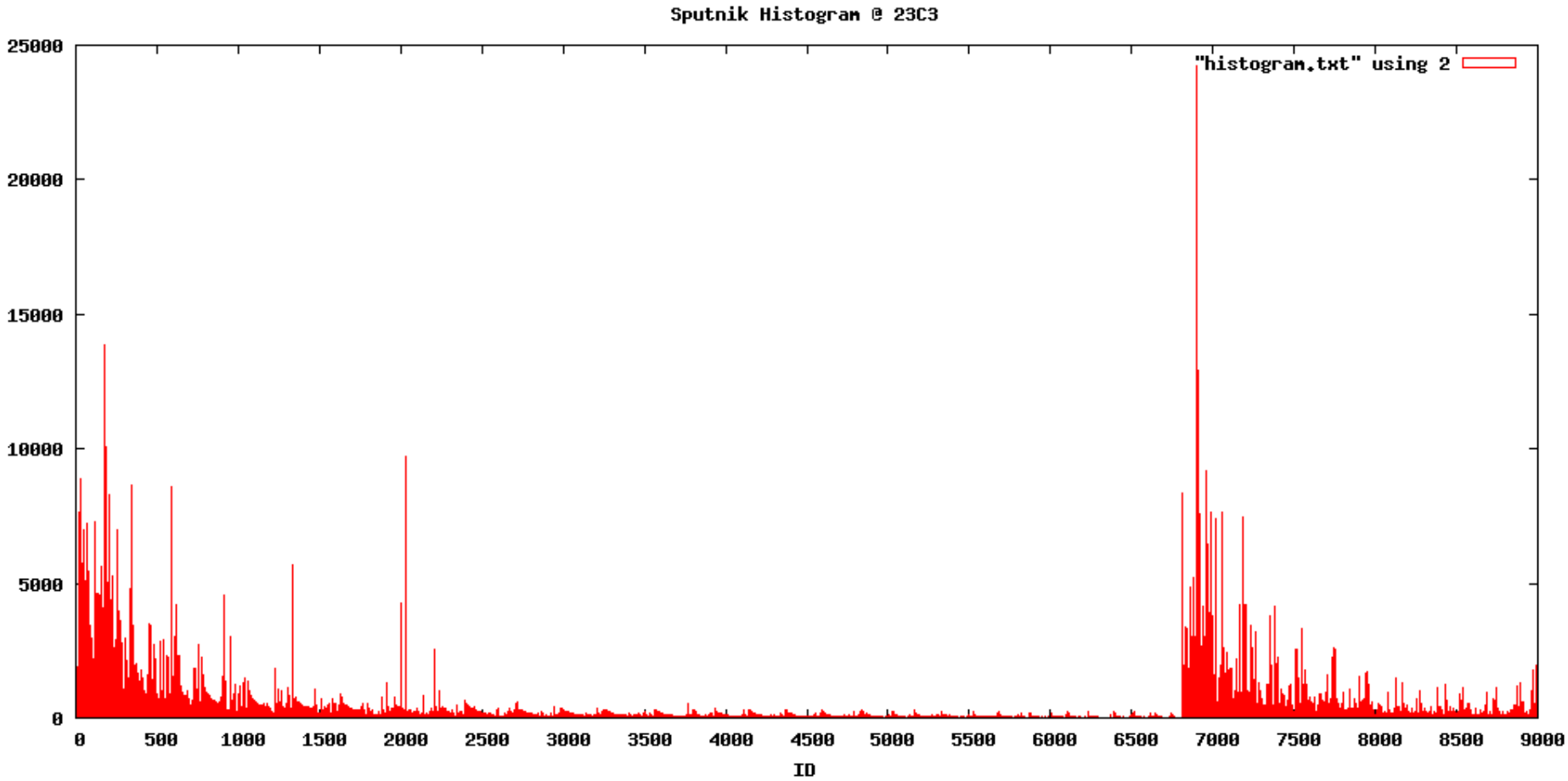


Figure: Histogram of sizes of generated sequences for the second set

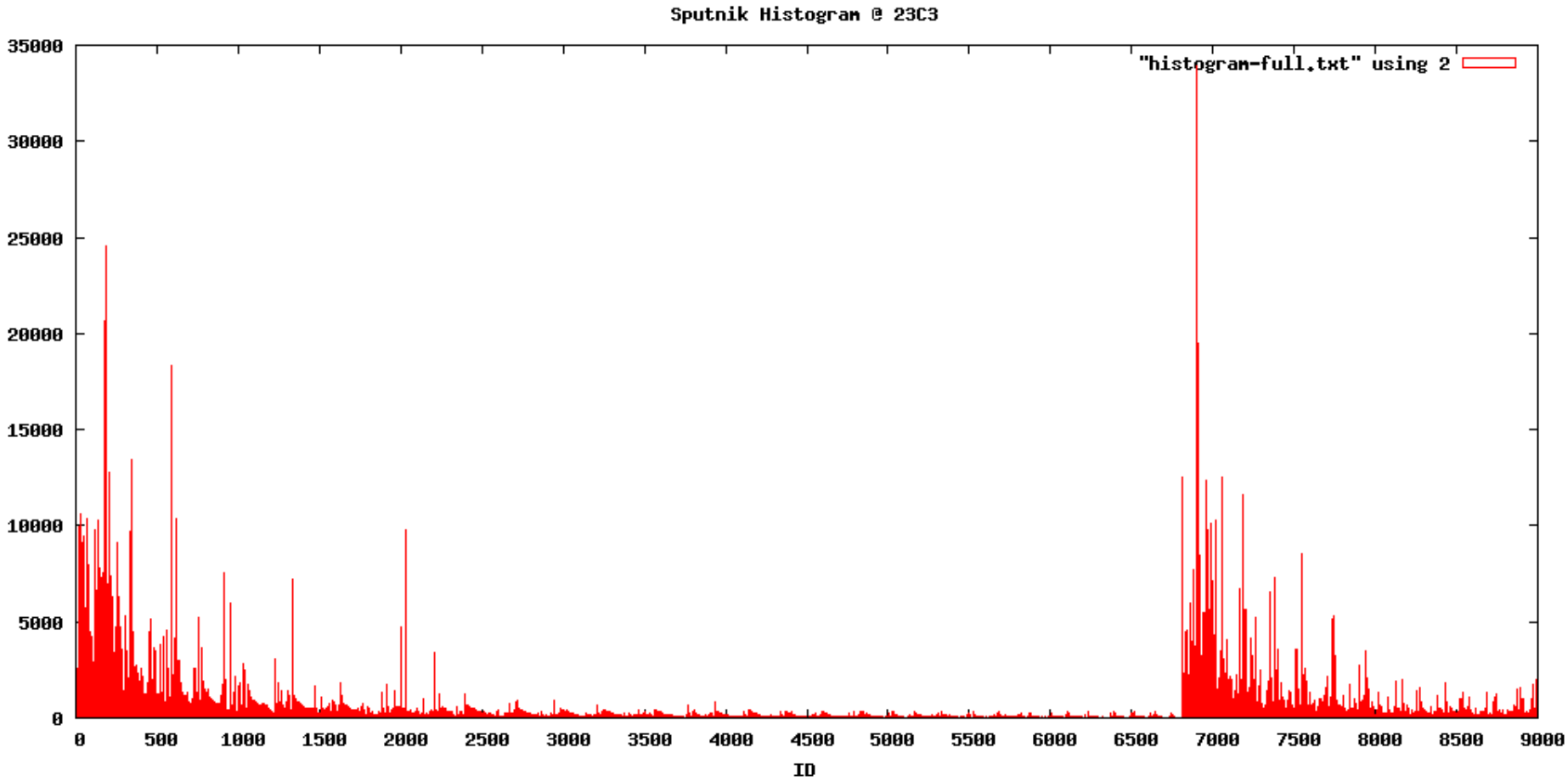


Figure: Histogram of sizes of generated sequences for the second set

Remarks on the algorithm

- Very slow; crawls after initial burst of activity
- Longer running than previous version or algorithm
  - More buckets in histogram
  - Changed code updating IDs in database; many individual operations instead of one bulk update
  - Grouping for time and using aggregate function
  - Using pl/Python function
- More equal load of CPU and IO subsystem
- First generated sequences are big
- Later are small, dozen or so points
- Maybe in case of small sequences increase width of slope range?

Problems with generated sequences

- Errors not showing on the graphs, but visible by looking at the raw data
- Problems are present even when grouping and aggregation function were used
- Collinear sequences
- Interlaced sequences

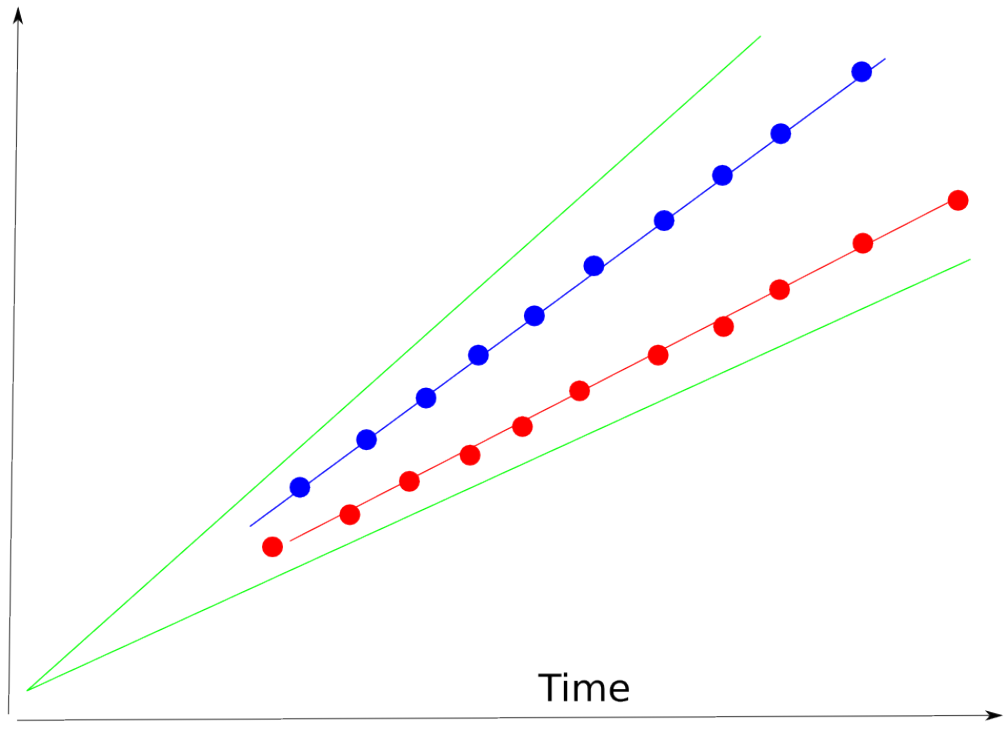


Figure: Interlaced sequences

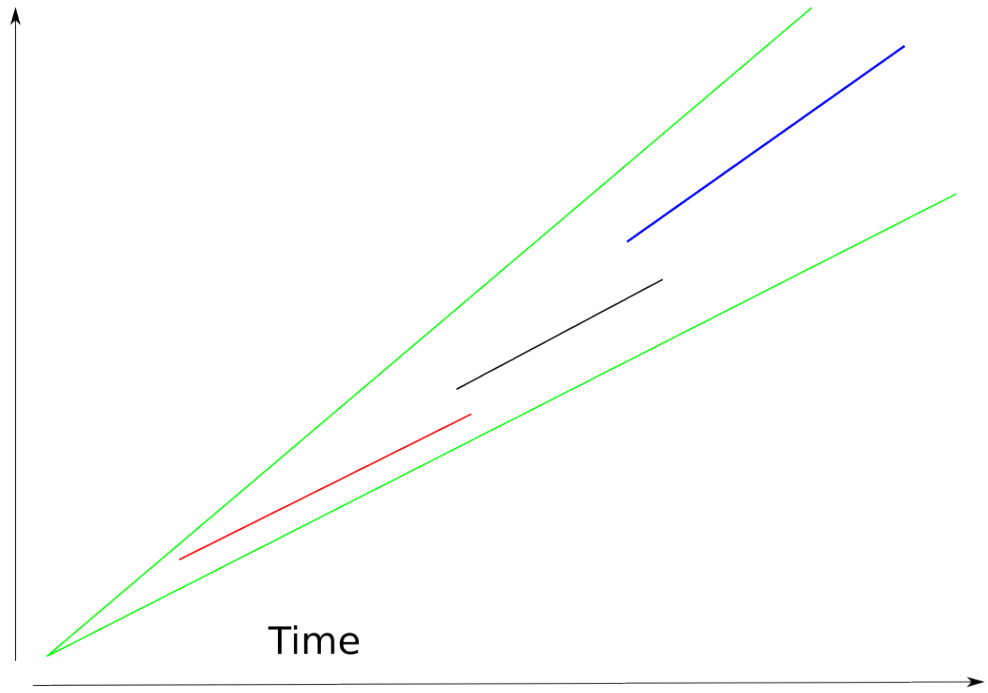


Figure: Collinear sequences

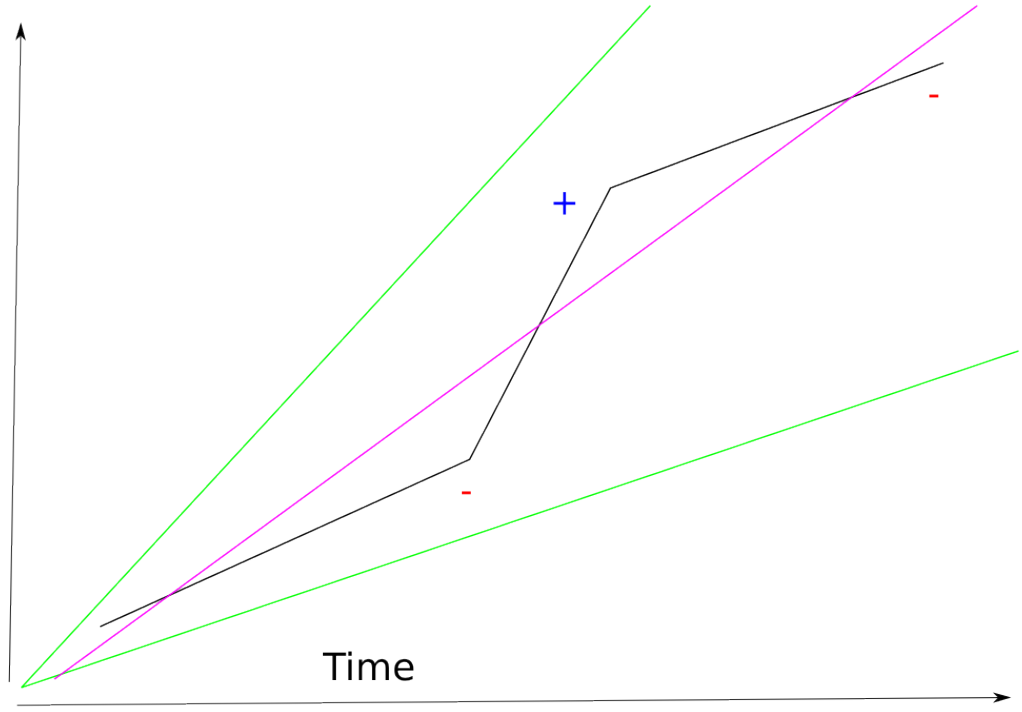


Figure: Incorrectly joined sequences

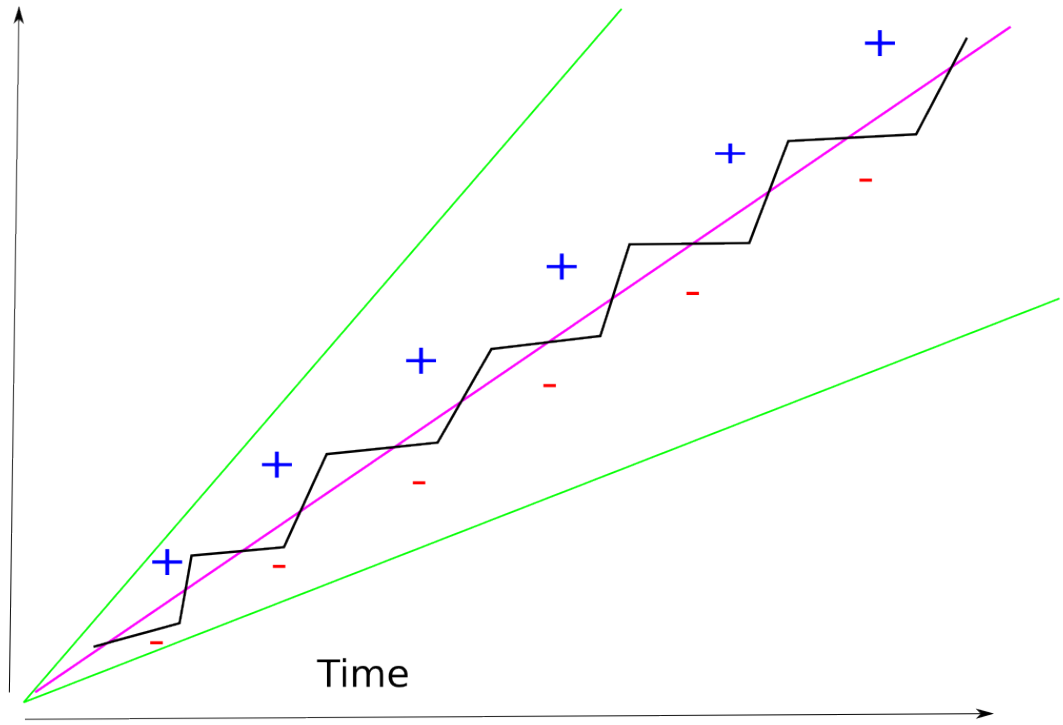


Figure: Correctly joined sequence

New firmware

- Released during CCC2007
- Ping not once per few seconds, but once per 0.1s
- Having two tags and USB reader I was able to perform experiment to check whether having only full seconds can be spoiling data
- I had two sets of data; one with discarded sub-second parts and one which uses them in calculations of slope
- First noticeable differences with slopes were on fourth place after comma
- So there is no difference whether we store only seconds, or also parts of seconds
- However first packets had very different values of slopes

Signal strength

- 4 levels: 0x00, 0x55, 0xaa, 0xff
- Cycling through those levels
- Old firmware, 23C3:
- 0x00, 0xff, 0x55, 0xff, 0xaa, 0xff, 0xff, 0xff
- New firmware, CCC2007:
- 0x00, 0x55, 0xaa, 0xff

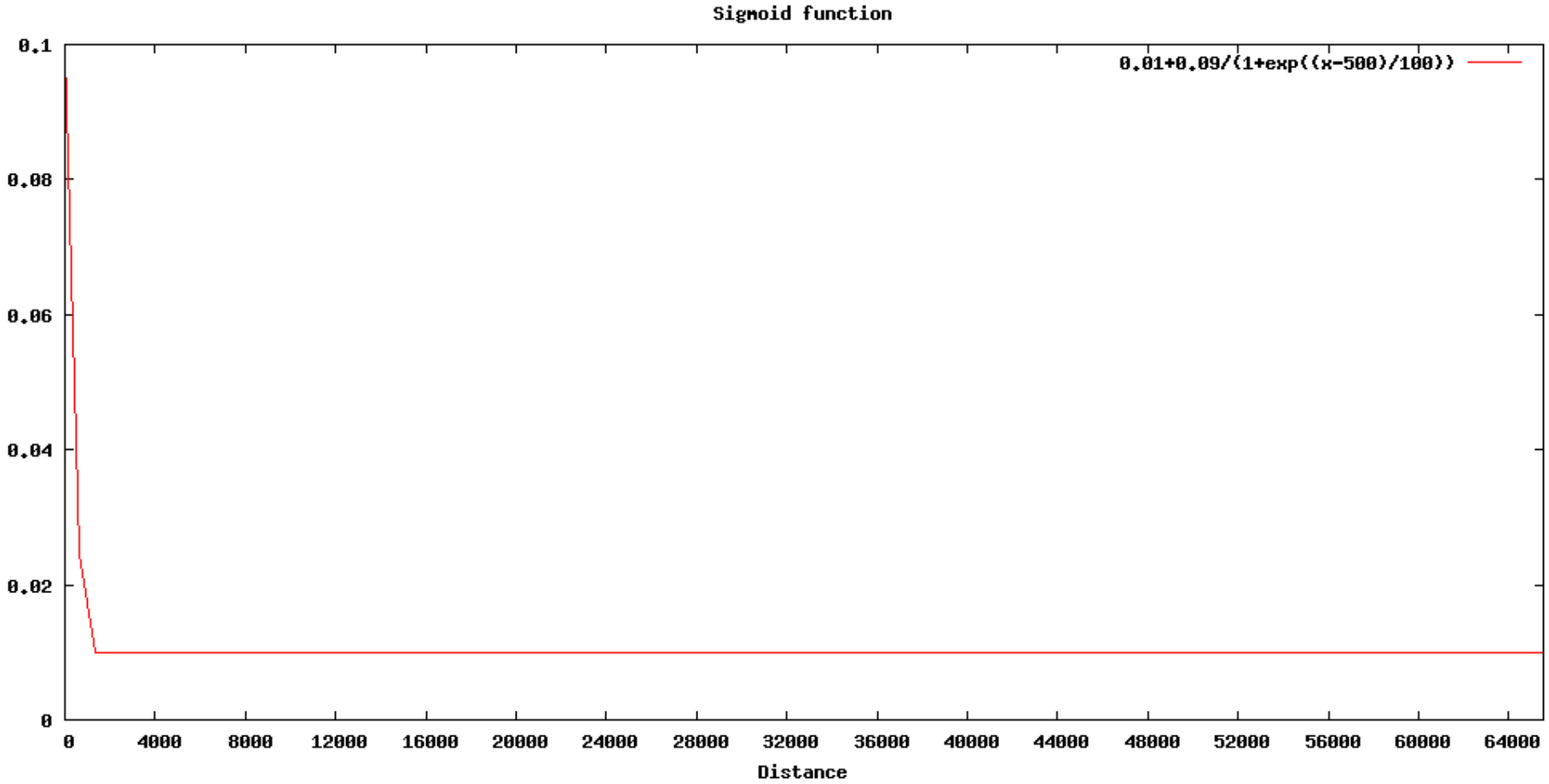
Analysis of signal strength

- I was hoping to be able to use signal's strength to check if this particular point can be put into sequence
- The problem would be in finding starting point of strength cycle for each sequence
- It would be difficult as in old firmware 5/8 (1, 3, 5, 6, 7) of values were 0xff
- But then by analysing source code I discovered that all sequences start at the same exact point
- Analysing data generated by tags, as well as looking at the single points from 23C3 confirmed this
- So only using changing strength is pointless, as it does not differ in sequences coming from different tags

Sigmoid function

- As I mentioned first packets have very different values of slopes
- This must be taken into consideration when using range to limit number points to use in line
- We need border that first is wide, and then gets narrower; this can be described by function that will give the smaller border the further from initial point it is are
- http://en.wikipedia.org/wiki/Sigmoid_function
- $\frac{1}{1+e^{-t}}$
- My version looks like: $0.01 + \frac{0.09}{1+e^{(x-500)/100}}$
- At distance 0, border is 0.1, and it is getting smaller, and at 1000 is 0.01, and stays there
- When distance is more than bout 70000 FPU exception occurs

Sigmoid function



Station list similarity

- It is time to take stations into consideration
- Using only list of seen stations, not geometrical data
- Function showing similarity between stations
- Best in range from 0 to 1 (from 0 to 100%)
- Take list of the seen stations and strengths
- Compare lists of the stations
- If the same, similarity is 1
- If different, take percentage of similar stations
- Take all stations in A and B, count distinct, and take percentage of common set
- If strengths are different and one is subset of another, assume similarity 1
- When strength is different similarity is number of stations seen in both cases divided by number of stations seen by weaker one

Generating the best sequence

- Find all candidate points
- Find all conflicting points (such that $\neg(T_1 > T_0 \wedge S_1 > S_0)$)
- Compare which one is the closest to ideal slope.
- Choose the best candidate by locally (previous and next) comparing stations seen by all points. Choose one that has the best similarity in seen stations
- If similarity is the same, or there is big jump, choose next point with help of adjacency table; not yet implemented
- Of course this will remove people that behaved differently than the rest of the crowd, nonconformists ;-)
- Put the next line into database

Changes in algorithm

- Point to generate sequences are retrieved using sigmoid border function with wider range
- All retrieved points are used to generate alternative sub-sequences, and the most similar to the main sequence are chosen
- To calculate similarity of sets of seen stations list of such stations must be received from database
- To do this I used aggregate function array_accum, from PostgreSQL documentation; grouped by sequence and time
- To avoid having joined lines, line is broken into two if there is probability that found line can be result of join of two lines

Similarity of seen stations

```
def Similarity(a, b):
    result = 0.0
    station0, strength0 = a
    station1, strength1 = b
    size0, size1 = len(station0), len(station1)
    if strength0[0] > strength1[0]:
        same = 0.0
        for i in station1:
            if i in station0: same += 1
        result = same/len(station1)
    elif strength0[0] < strength1[0]:
        same = 0.0
        for i in station0:
            if i in station1: same += 1
        result = same/len(station0)
    else:
        result = float(len(set(station0)&set(station1)))/
            float(len(set(station0)|set(station1)))
    return result
```

Getting all points that can create line

```
def Fetch(c, time, sequence, slope, sa, sz):
    result = [[time, sequence, slope, 0.0]]
    c.execute("""SELECT sputnik.array_accum(station),
    sputnik.array_accum(strength)
    FROM sputnik.ccc23 WHERE id IS NULL AND
    time = %s::TIMESTAMP WITH TIME ZONE AND
    sequence = %s::BIGINT""", (time, sequence))
        i = c.fetchone()
    if i != None:
        result[0].append(i[0])
        result[0].append(i[1])
    i = c.fetchall()
# Union of first 100s and the rest
    c.execute("""SELECT time, sequence,
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float,
    0.0, sputnik.array_accum(station), sputnik.array_accum(strength)
    FROM sputnik.ccc23 WHERE id IS NULL AND
    sequence > %s::BIGINT AND sequence <= %s::BIGINT+100::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND time <= %s::TIMESTAMP WITH TIME ZONE+'100 second'::INTERVAL
    GROUP BY time, sequence
    UNION
    SELECT time, sequence,
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float,
    0.0, sputnik.array_accum(station), sputnik.array_accum(strength)
    FROM sputnik.ccc23 WHERE id IS NULL AND
    sequence BETWEEN %s::BIGINT AND %s::BIGINT AND
    time > %s::TIMESTAMP WITH TIME ZONE AND
    sequence > %s::BIGINT AND
    (extract('epoch' FROM (time-%s::TIMESTAMP WITH TIME ZONE)))::float/(sequence-%s::BIGINT)::float
    BETWEEN %s::float-sputnik.BorderWidth(sequence-%s) AND %s::float+sputnik.BorderWidth(sequence-%s)
    GROUP BY time, sequence ORDER BY time""", (time, sequence, sequence, sequence, time, time, time, sequence,
    i = c.fetchone()
    while i != None:
        result.append([i[0], i[1], i[2], i[2]-result[-1][2], i[4], i[5]])
        i = c.fetchone()
    return result
```

Calculating all possible sequences from points

```
def Lines(data):
    result = [] candidate = []
    for i in data:
        num = 0
        for j in candidate:
            if i[0] > j[-1][0] and i[1] > j[-1][1]:
                num += 1
        if len(candidate) == num:
            if len(candidate) == 1: result.extend(candidate[0])
            elif len(candidate) > 1: result.append(candidate)
            candidate = [[i]]
        else:
            for j in candidate:
                if i[0] > j[-1][0] and i[1] > j[-1][1]:
                    j.append(i)
            if 0 == num: candidate.append([i])
# Add last alternative
    if len(candidate) == 1: result.extend(candidate[0])
    elif len(candidate) > 1: result.append(candidate)
    return result
```

Choosing the best line from all alternatives

```
def Line(lines):
    result = []
    for i in xrange(len(lines)):
        if type(lines[i][0]) != type([]): result.append(lines[i])
        else: alternatives = []
            if len(result) > 0:
                for j in lines[i]:
                    if j[0][0] > result[-1][0] and j[0][1] > result[-1][1]: alternatives.append(j)
            else: alternatives = lines[i]
            scores = [0] * len(alternatives)
            sizes = map(lambda x: len(x), alternatives)
            best = max(sizes)
            for j in xrange(len(alternatives)):
                if sizes[j] == best: scores[j] += 1
            stationsa = map(lambda x: Similarity((result[-1][4], result[-1][5]), (x[0][4], x[0][5])), alternati
# Find best alternative for stations in the beginning
            if i+1 < len(lines) and type(lines[i+1][0]) != type([]):
                stationsz = map(lambda x: Similarity((x[-1][4], x[-1][5]), (lines[i+1][4], lines[i+1][5])), alt
# Find best alternative for stations in the end
            slopesa = map(lambda x: abs(alternatives[x][0][3]-result[-1][3]), xrange(len(alternatives)))
# Find best alternative for slopes in the beginning
            if i+1 < len(lines) and type(lines[i+1][0]) != type([]):
                slopesz = map(lambda x: abs(alternatives[x][0][3]-lines[i+1][3]), xrange(len(alternatives)))
# Find best alternative for slopes in the end
# Find the best alternative:
            best = max(scores)
            for j in xrange(len(alternatives)):
                if scores[j] == best:
                    result.extend(alternatives[j])
                    break
# Count slope deltas once more, for final line proposal
    slope = result[0][2]
    for i in result:
        i[3] = i[2]-slope
        slope = i[2]
    return result
```

Function returning probability of break

```
def Break(a, b, c, d, slope):
    result = 0.0
    SlopeDiff = 10.0
    SlopeTrigger = 0.01
    CounterDiff = 100
    TimeDiff = datetime.timedelta(0, 120)
    StationSimilarity = 0.5
    if abs(c[3]) > SlopeTrigger:
        if abs(c[3]) > abs(b[3])*SlopeDiff: result += 1.0
        if abs(c[3]) > abs(d[3])*SlopeDiff: result += 1.0
# Time is more intuitive that sequence counter
# Also I do not have to think about line coefficient
#    if c[1] - b[1] > CounterDiff: result += 1.0
    if c[0] - b[0] > TimeDiff: result += 1.0
    if Similarity((b[4], b[5]), (c[4], c[5])) < StationSimilarity: result += 1.0
    SlopeAB = float((b[0]-a[0]).seconds)/(b[1]-a[1])
    SlopeBC = float((c[0]-b[0]).seconds)/(c[1]-b[1])
    SlopeCD = float((d[0]-c[0]).seconds)/(d[1]-c[1])
# Slopes should be similar to each other and to the main slope
    if slope-1.0 <= SlopeAB and SlopeAB <= slope+1.0 and (SlopeBC < slope-1.0 or slope+1.0 < SlopeBC):
        result += 1.0
    if slope-1.0 <= SlopeCD and SlopeCD <= slope+1.0 and (SlopeBC < slope-1.0 or slope+1.0 < SlopeBC):
        result += 1.0
    return result/6.0
```

Function creating all lines

```
def FindIDs(connection, sa, sz, id):
    c.execute("""SELECT DISTINCT sequence FROM sputnik.ccc23 WHERE id IS NULL AND
    sequence BETWEEN %s AND %s ORDER BY sequence""", (sa, sz))
    for s in c.fetchall():
        s0 = s[0]
        c.execute("""SELECT DISTINCT time FROM sputnik.ccc23
        WHERE id IS NULL AND sequence = %s""", (s0,))
        for t in c.fetchall():
            t0 = t[0]
            slope, count = Histogram(c, t0, s0, sa, sz)
            if slope > 0.0 and count >= 8:
                data = Fetch(c, t0, s0, slope, sa, sz)
                lines = Lines(data)
                line = Line(lines)
                for i in xrange(len(line)):
                    skip = False
                    if len(line[i][4]) != len(line[i][5]):
                        print "Error in size of ", line[i]
                        skip = True
                    s = line[i][5][0]
                    for j in line[i][5]:
                        if j != s:
                            print "Error in strength of ", line[i]
                            skip = True
                    if skip:
                        break
                    UPDATE sputnik.sputnik SET id = %s WHERE id IS NULL AND
                    time = %s::TIMESTAMP WITH TIME ZONE AND sequence = %s::BIGINT
                    if i > 0 and i < len(line)-2:
                        b = Break(line[i-1], line[i], line[i+1], line[i+2], slope)
                        if b > 0.5:
                            id += 1
                            print "Break here, new id ", id, b
                id += 1
    return id
```

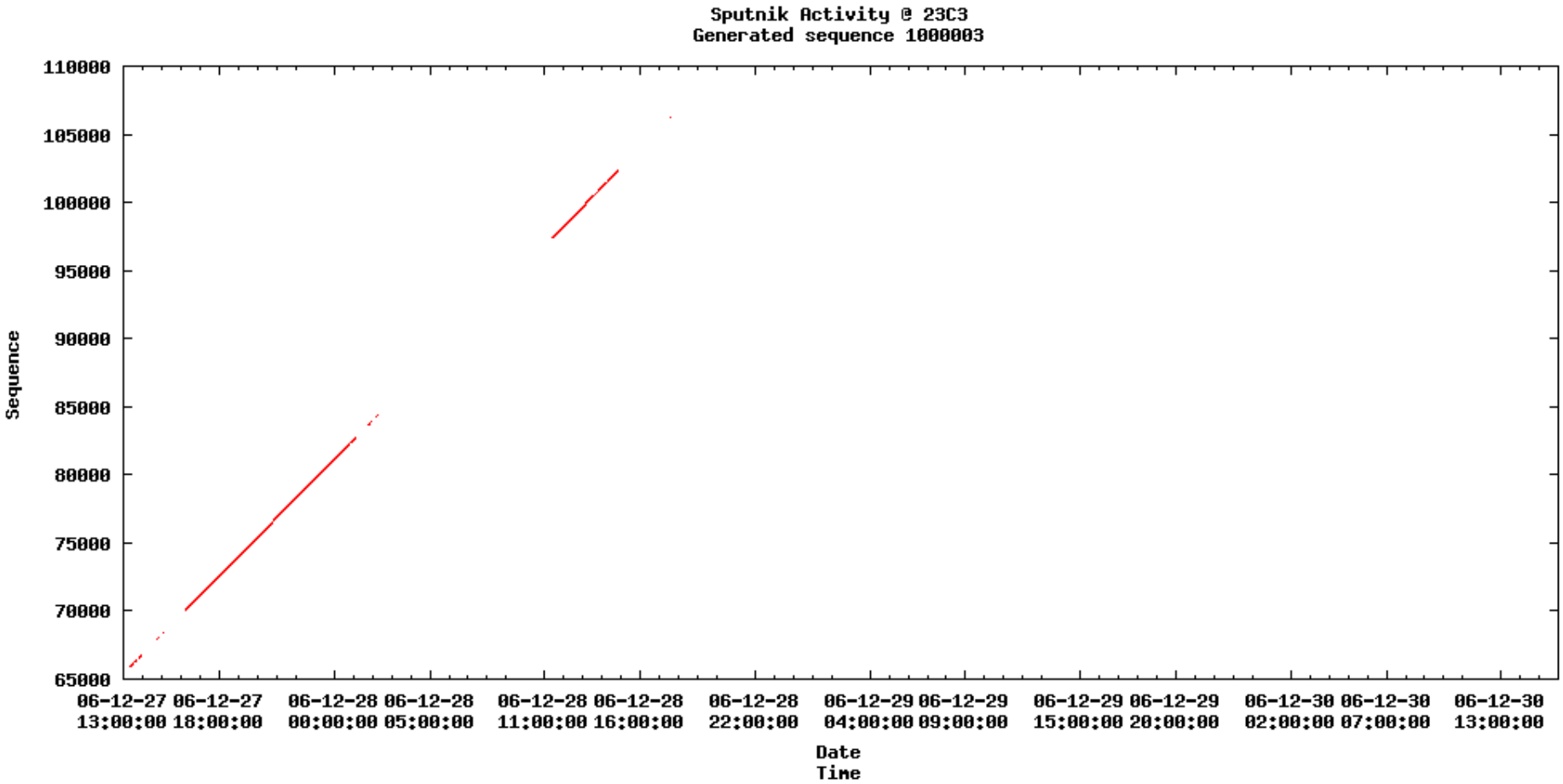


Figure: Generated sequence; third set, number 3

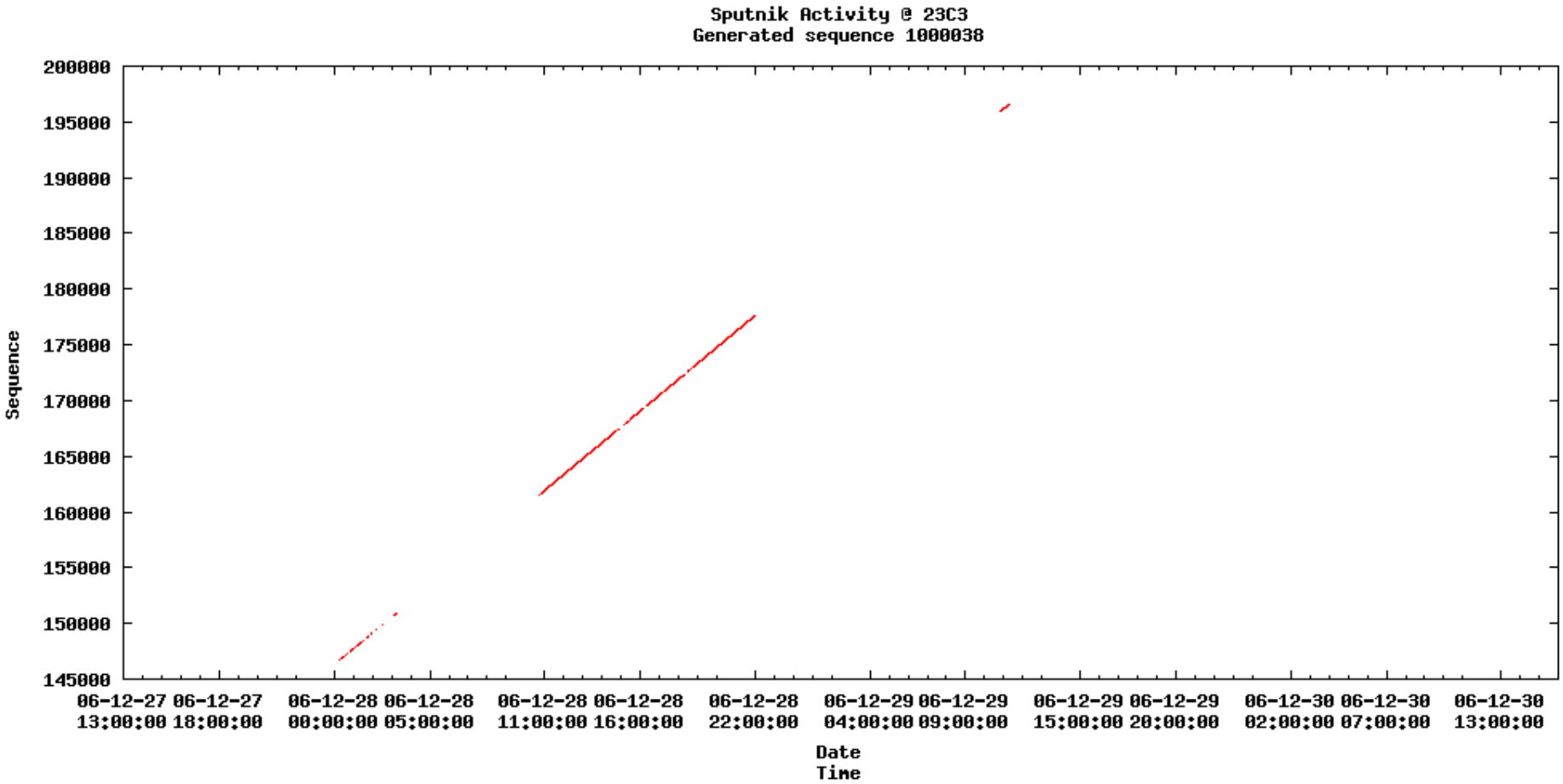
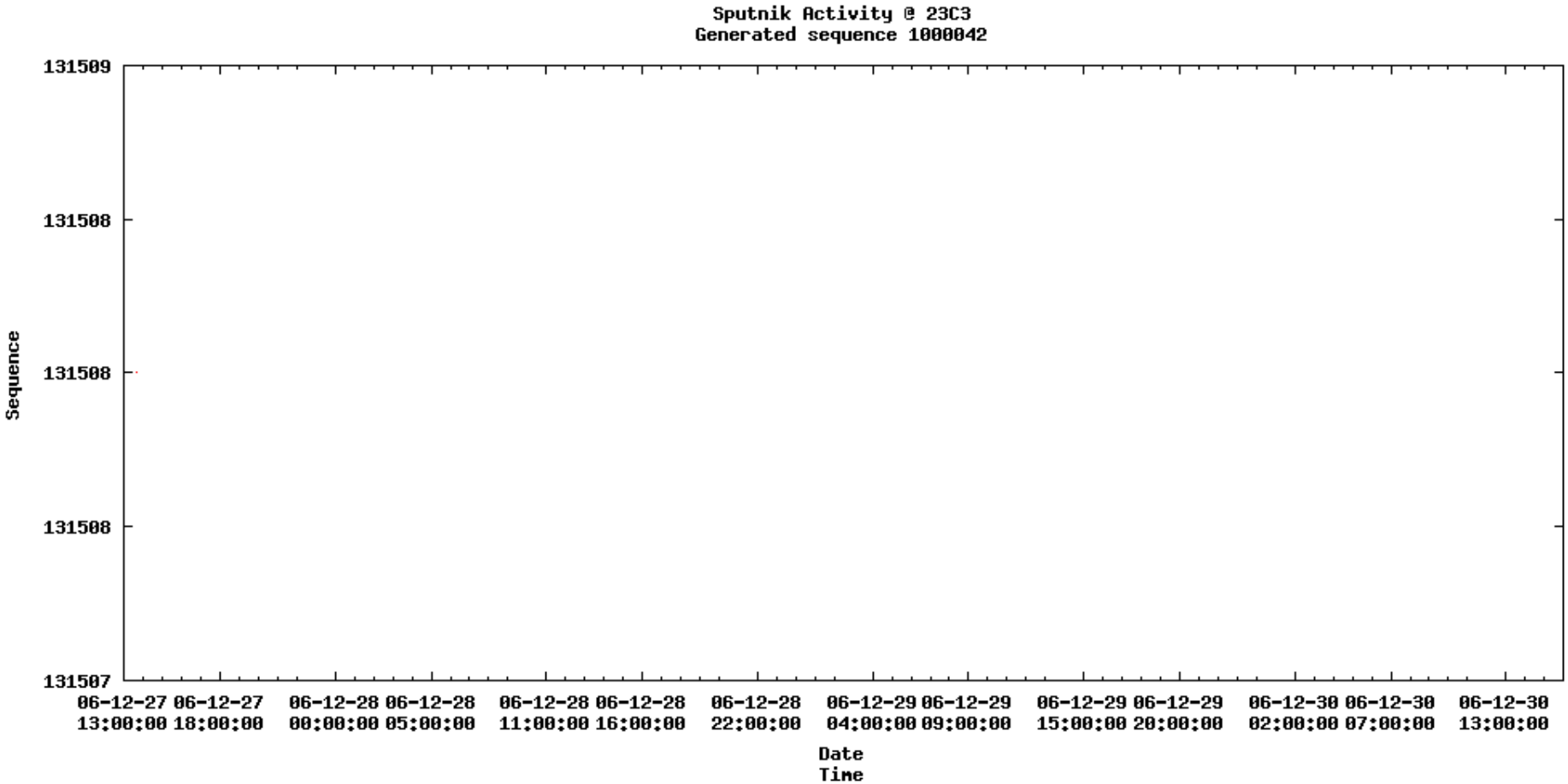
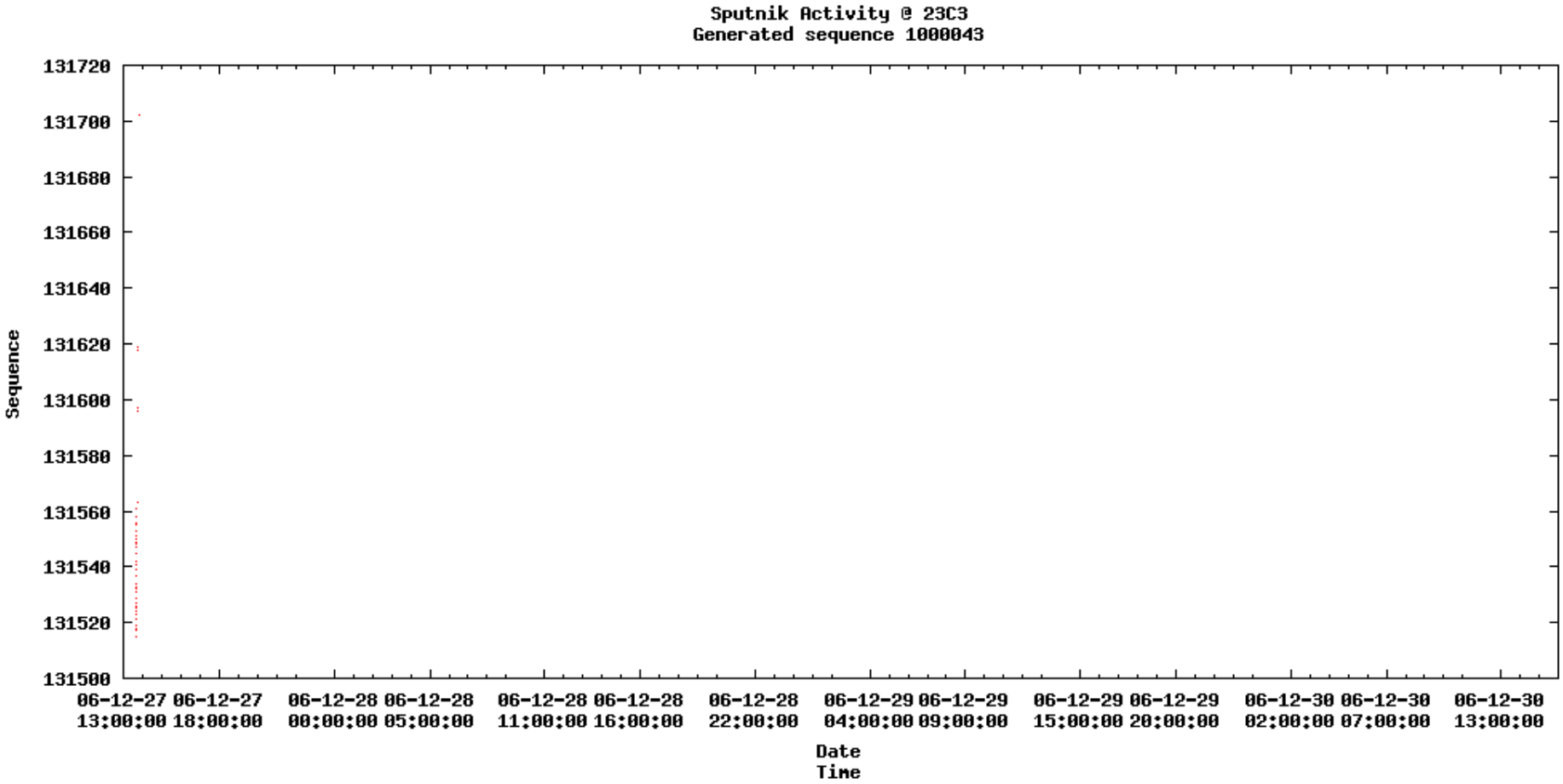


Figure: Generated sequence; third set, number 38

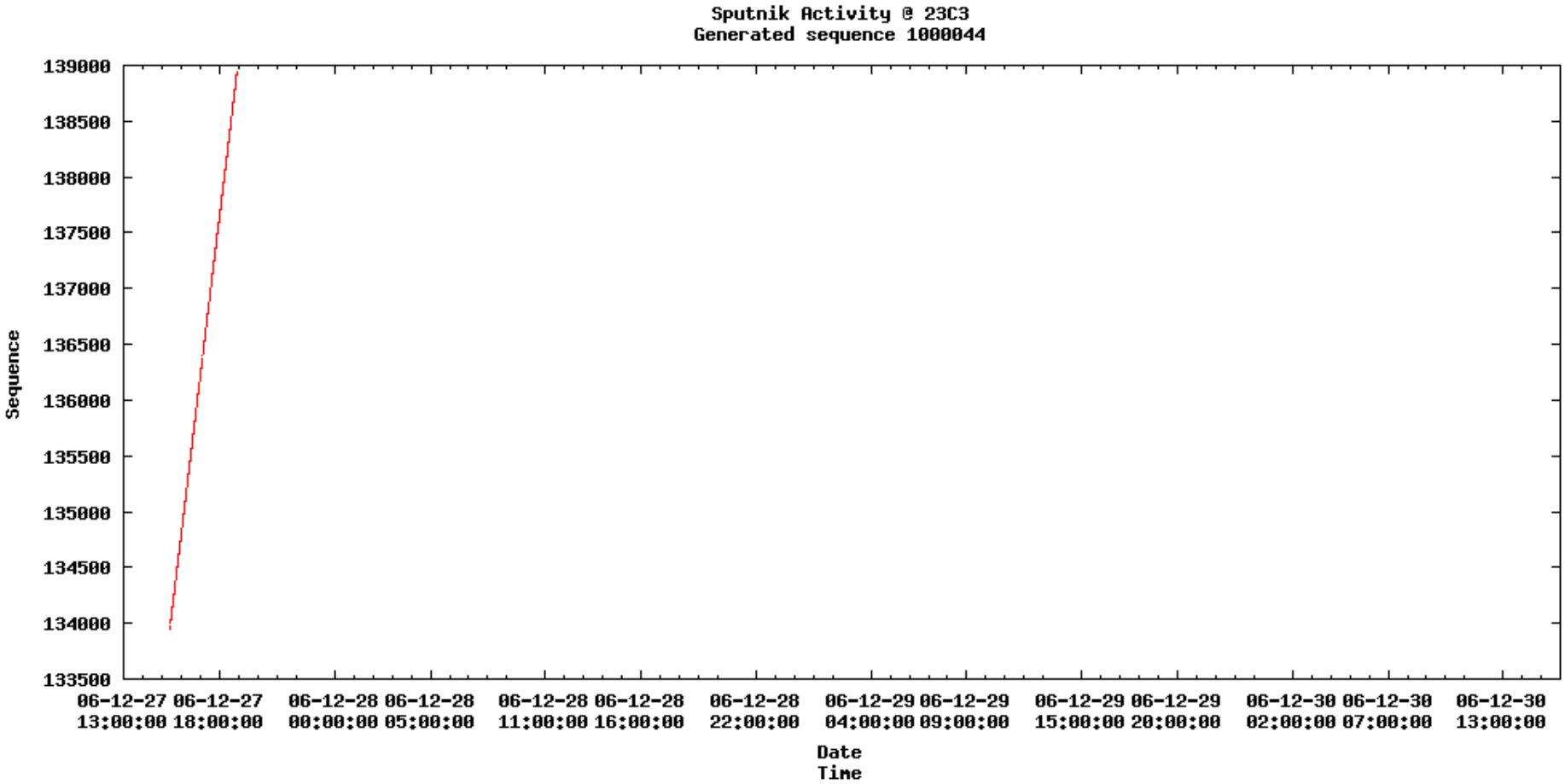
Generated sequence; third set, number 42



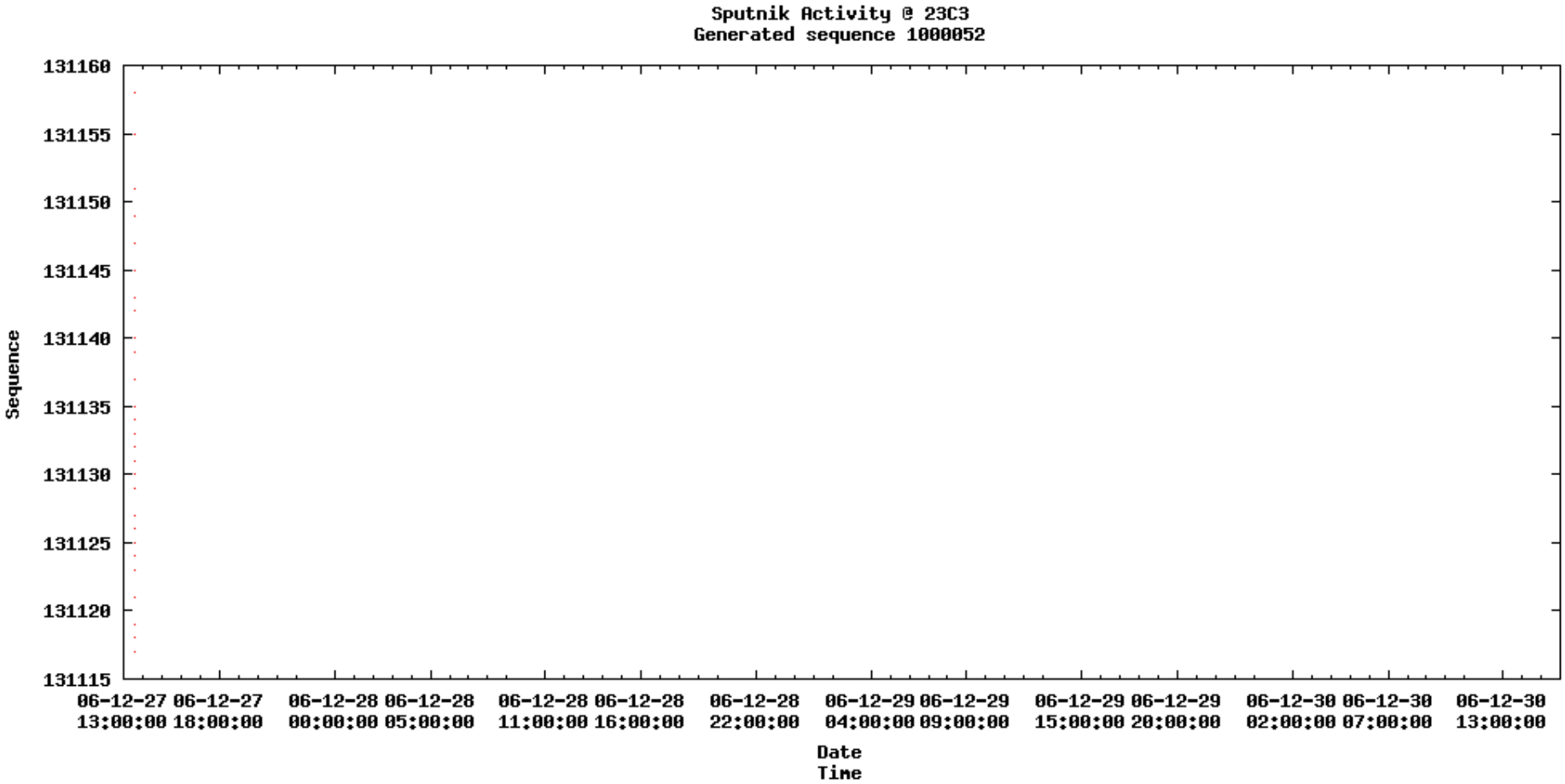
Generated sequence; third set, number 43



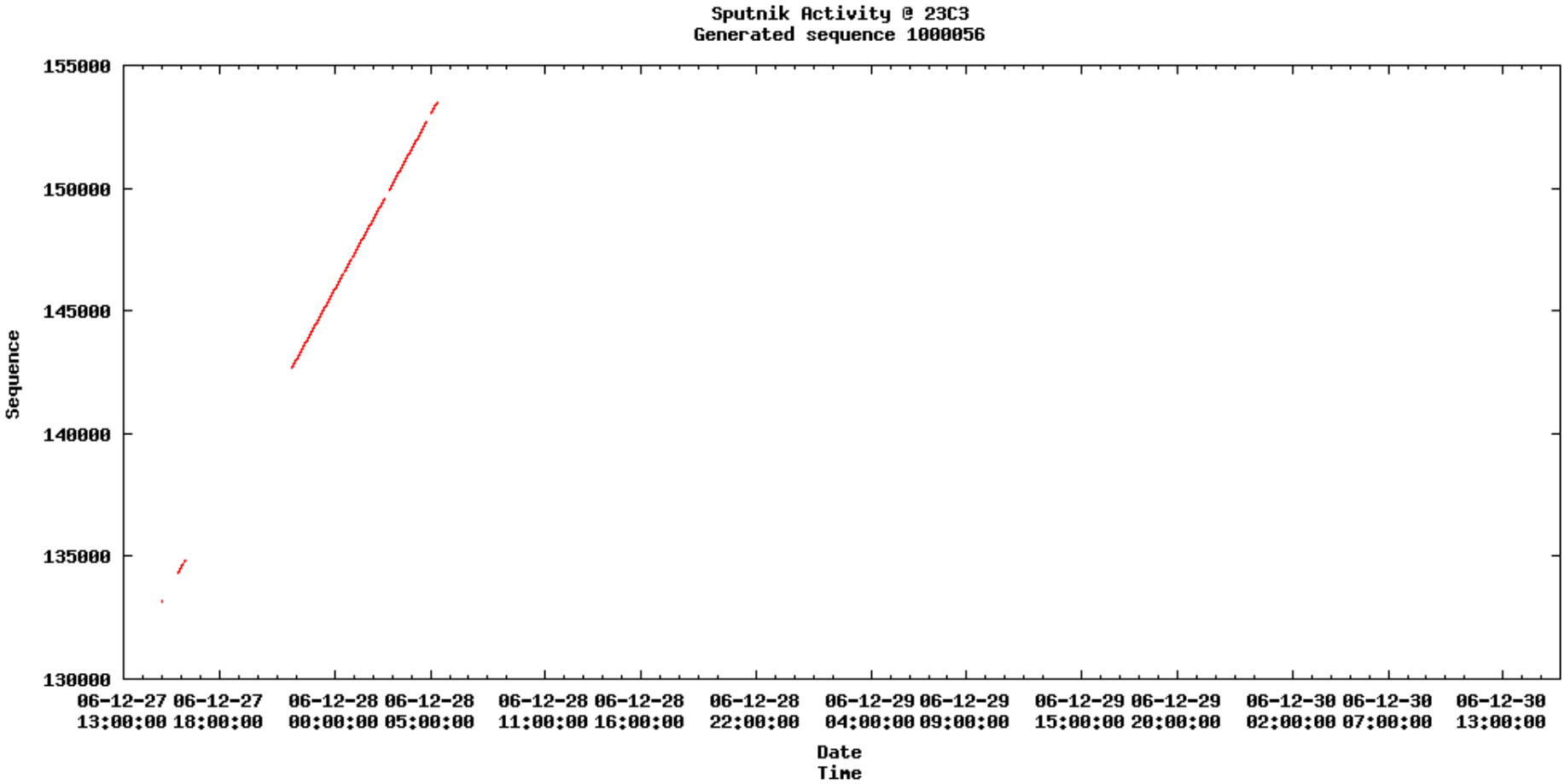
Generated sequence; third set, number 44



Generated sequence; third set, number 52



Generated sequence; third set, number 56



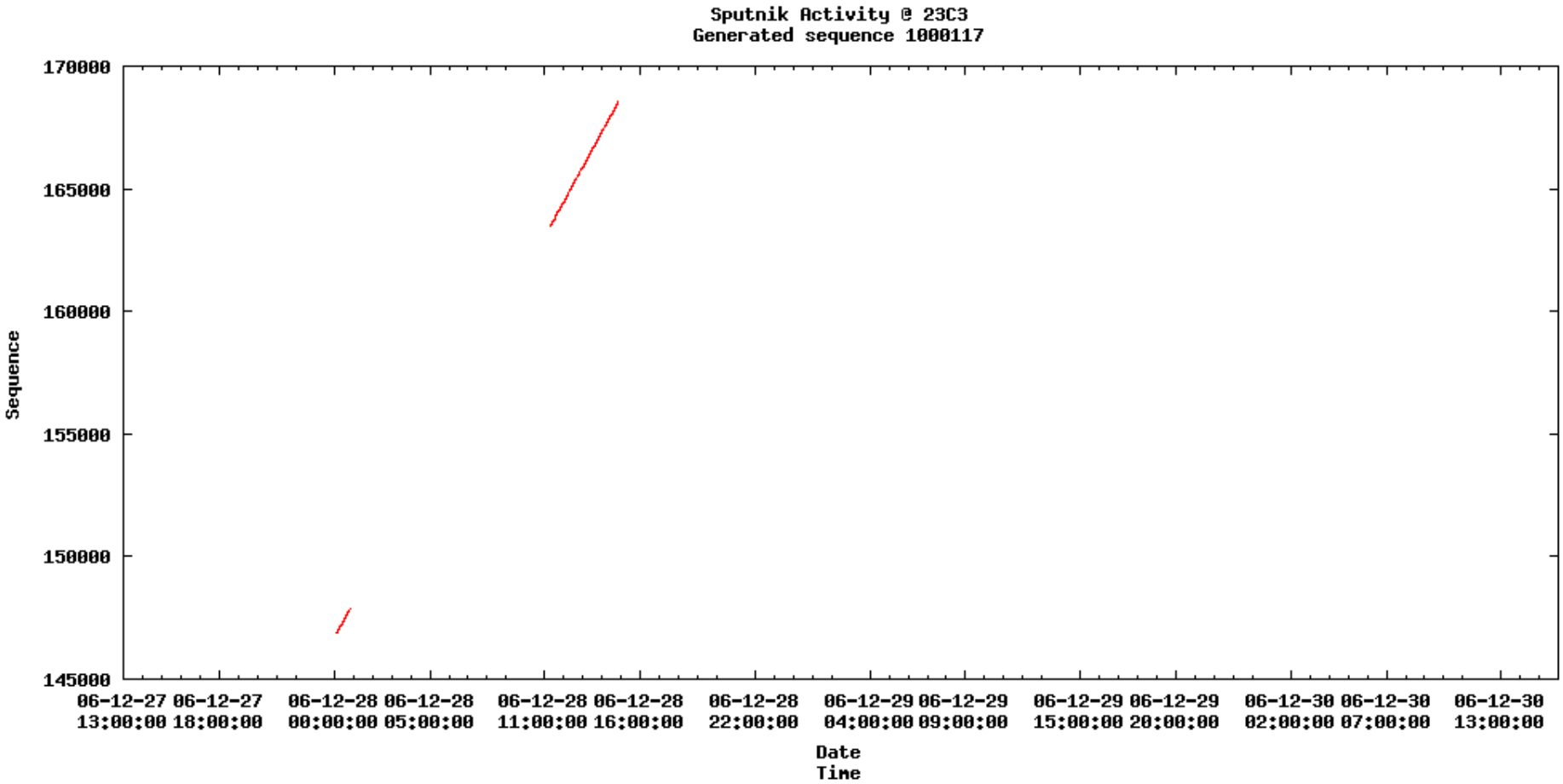
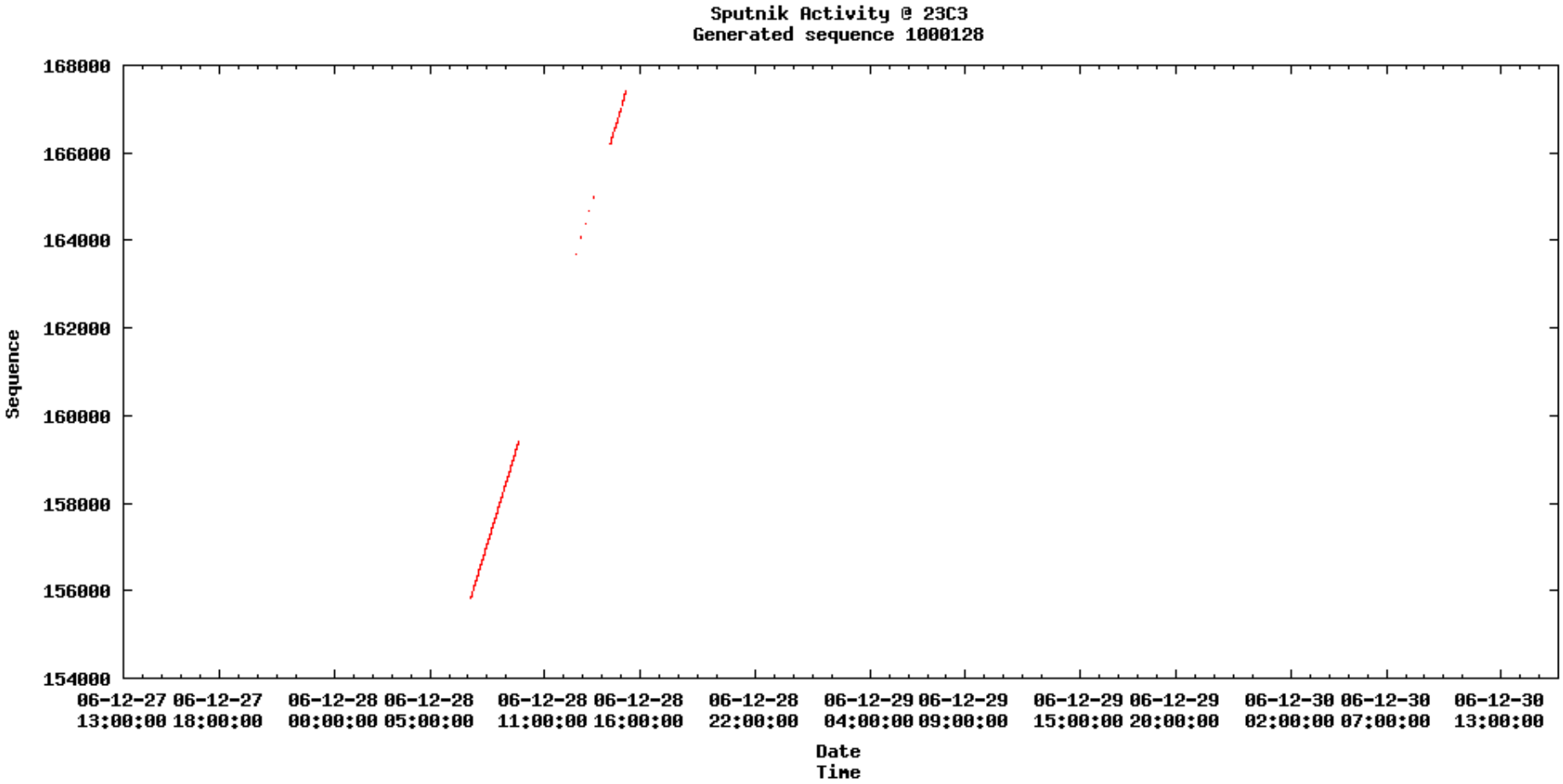


Figure: Generated sequence; third set, number 117

Generated sequence; third set, number 128



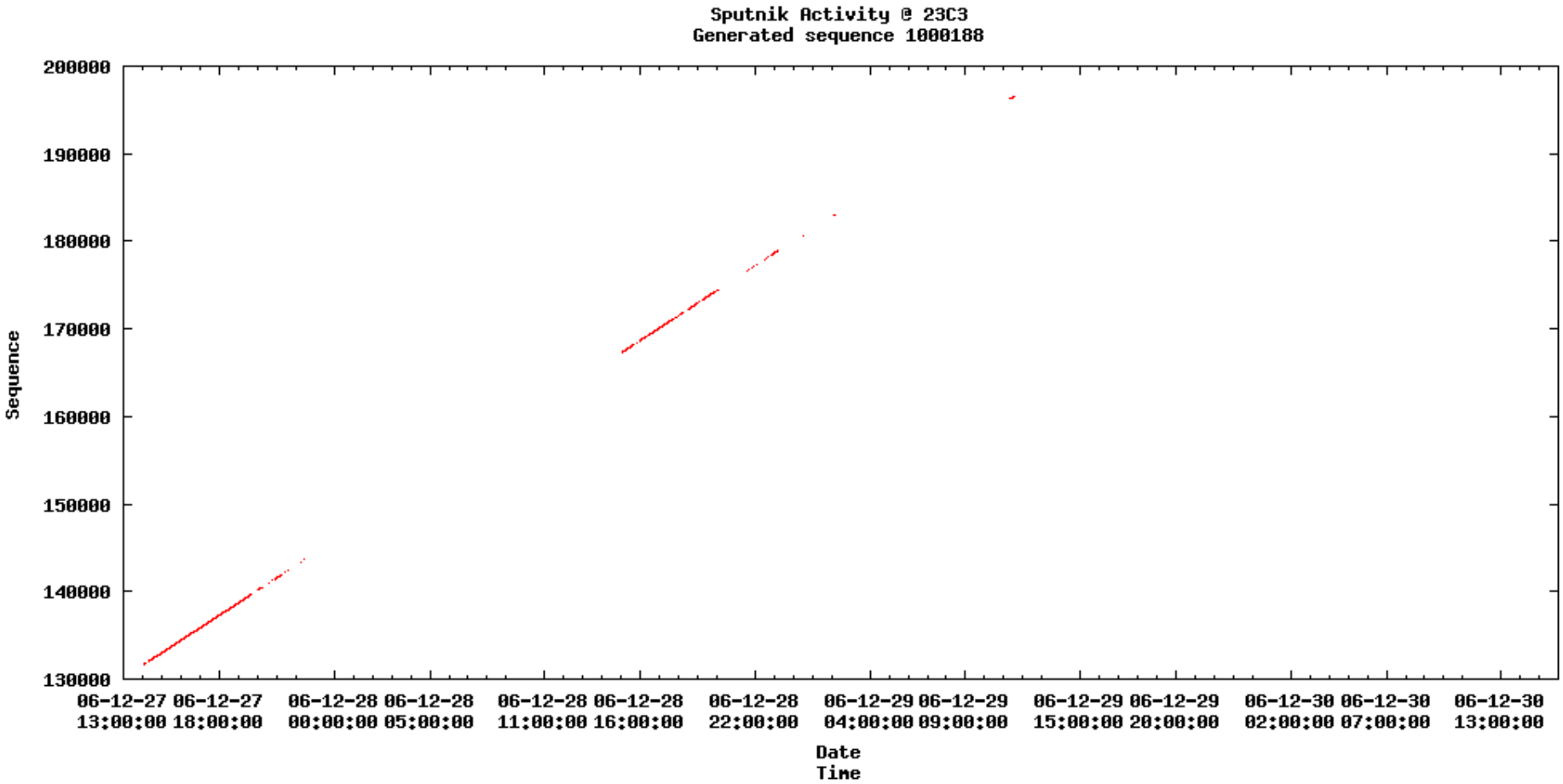


Figure: Generated sequence; third set, number 188

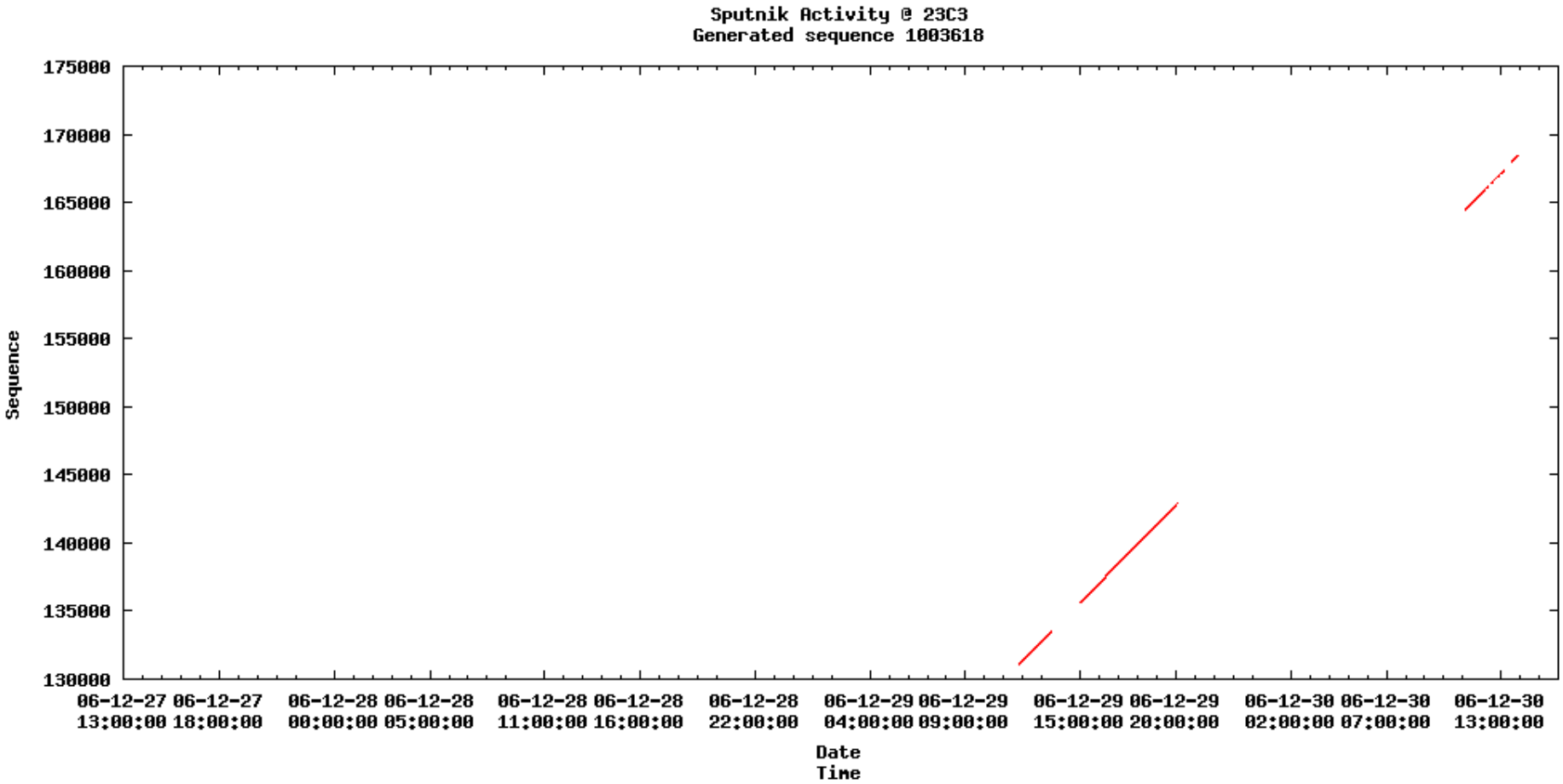
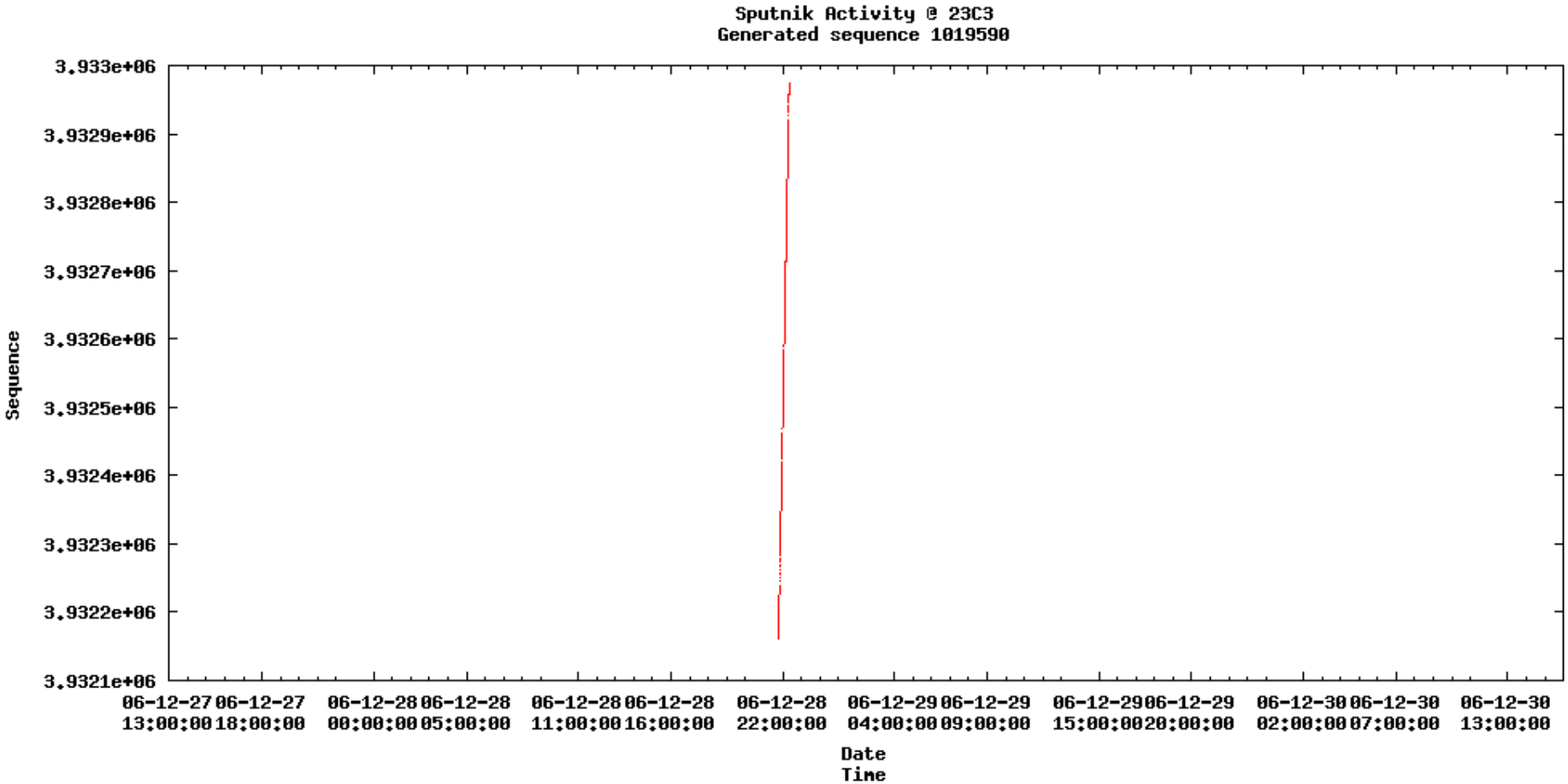


Figure: Generated sequence; third set, number 3618

Generated sequence; third set, number 19590



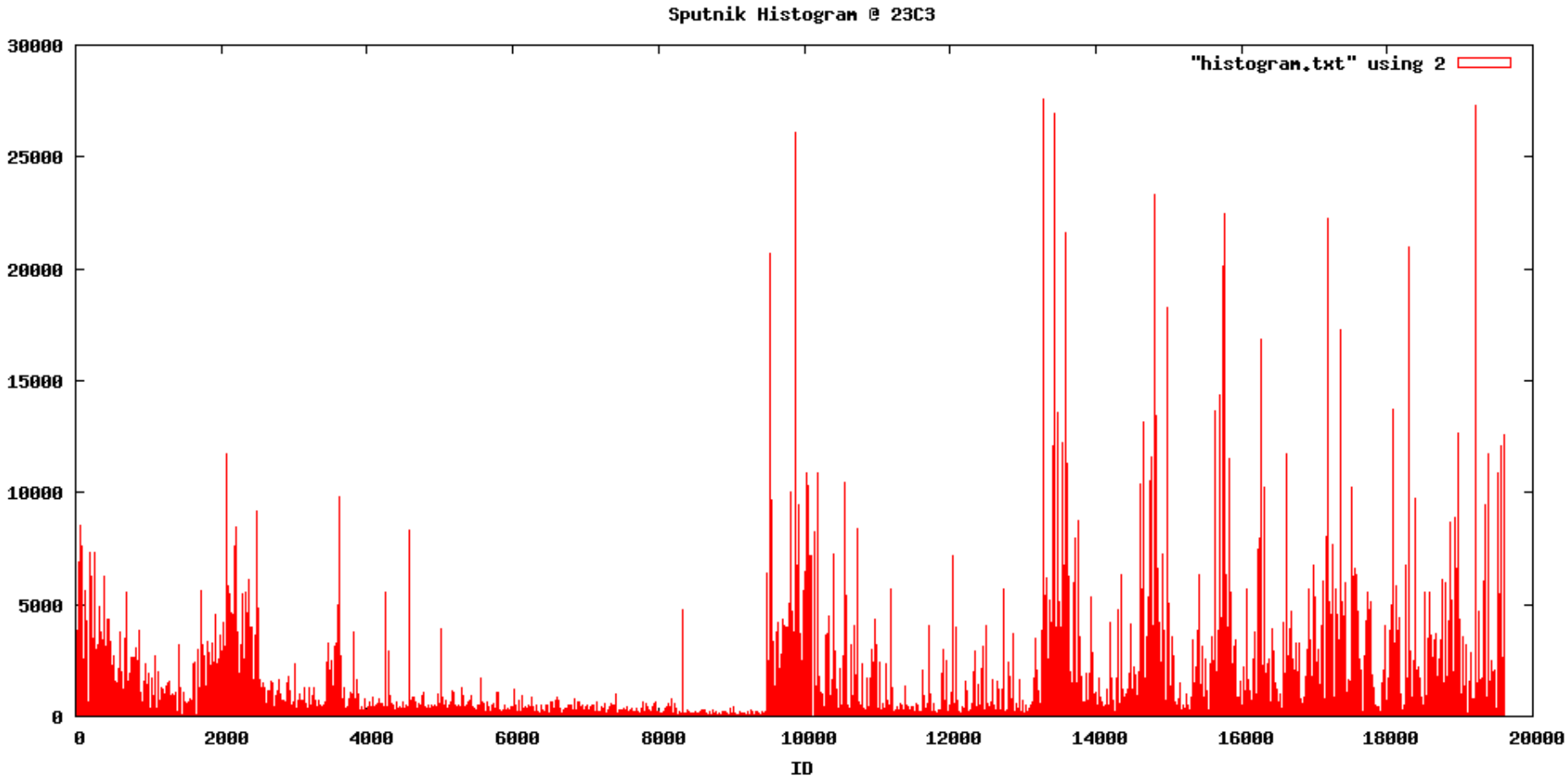


Figure: Histogram of sizes of generated sequences for the third set

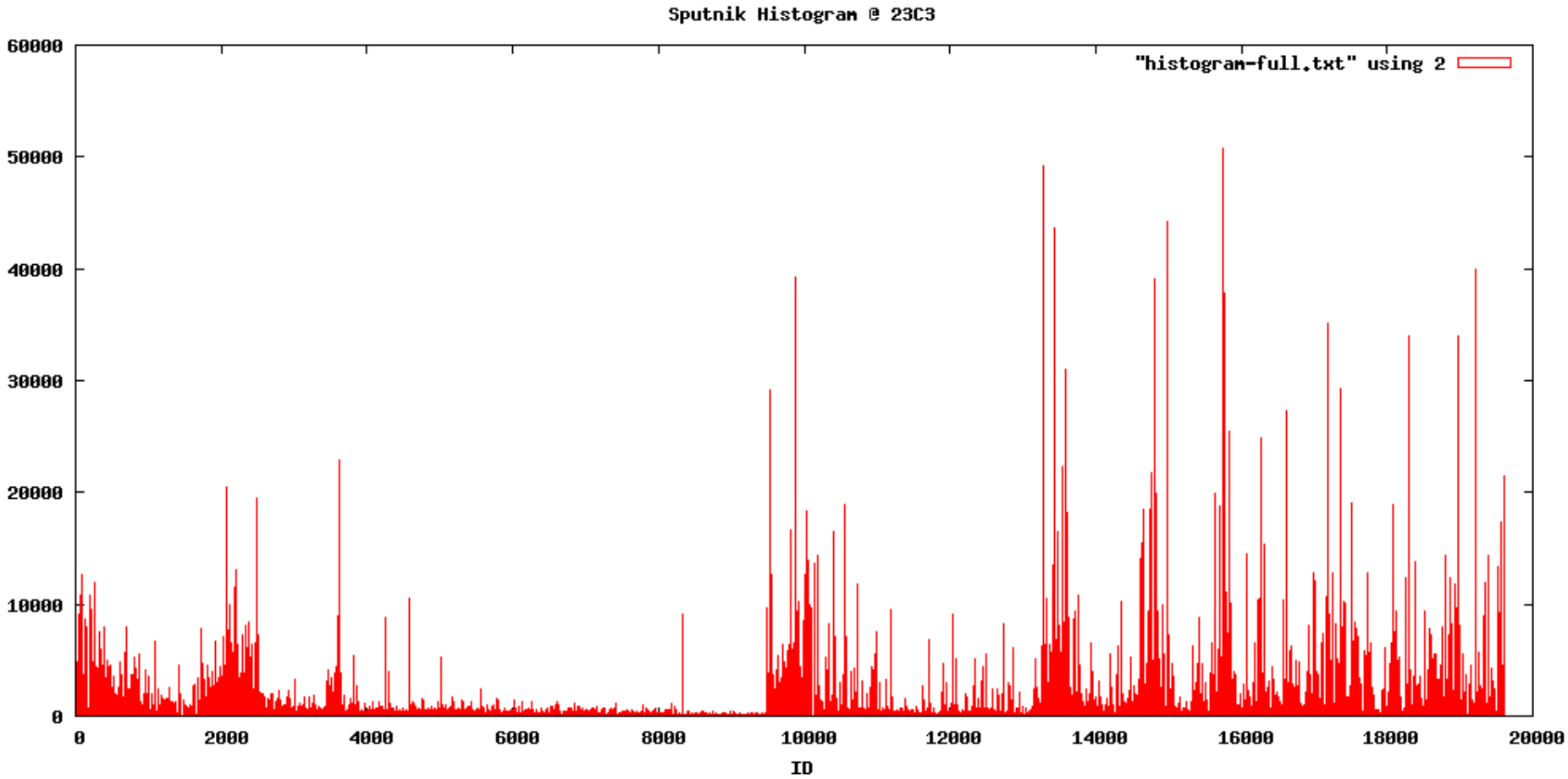


Figure: Histogram of sizes of generated sequences for the third set

Remarks on the algorithm

- Was running 5634 minutes on 64 bit AMD 3400+ with 1GB of RAM and one IDE HDD 7200RPM
- Used 10.6 million points
- Was stopped by FPU error in sigmoid function for large values of counter
- Over 1600 sequences made from more than 1000 points

Joining of sequences

- Many (too many?) sequences that are too short
- Need to start joining
- Or extend sequences by trying to use left points
- But how to join?
- For example which one of 3, 38, 117, 128, 188 should be joined to the 3618?
- Should I use manual joining?
- Manual (semi-manual) choosing of alternative in sequences?

Line slope

- I assumed that sequence has linear growth, with coefficient 1.5
- Next I assumed range of coefficients, from 1.0 to 2.0, based on presentation from 23C3, and by observing histograms
- This did not give good results in local algorithms, and was extended in global algorithms
- In Sputnik source code version 0.29 (file main.c) there are two sleep function calls
- Units are ticks, 32768Hz
- One is constant sleep_jiffies(0xffff) in line 271, and one is random sleep_jiffies(rand() in line 276)
- Difference between ticks should be between 2.0 and 4.0 seconds
- So there should be no straight line in data!
- But I can find lines
- So either there is too much data, and one can find anything he/she wants
- Or rand() was not so great, and it gave not-so-random data

Fragment of firmware of tag

```
void main (void)
{// get random seed
  ((unsigned char *) &seq)[0] = EEPROM_READ (4);
  ((unsigned char *) &seq)[1] = EEPROM_READ (5);
  ((unsigned char *) &seq)[2] = EEPROM_READ (6);
  ((unsigned char *) &seq)[3] = EEPROM_READ (7);
// increment code block after power cycle
  ((unsigned char *) &crc)[0] = EEPROM_READ (8);
  ((unsigned char *) &crc)[1] = EEPROM_READ (9);
  store_codeblock (++crc);
  seq ^= crc;
  srand (crc16 ((unsigned char *) &seq, sizeof (seq)));
// increment code blocks to make sure that seq is higher or equal after battery change
  seq = ((u_int32_t) crc) << 16;
  i = 0;
  while (1) {
// update code_block so on next power up the seq will be higher or equal
      crc = seq >> 16;
      if (crc == 0xFFFF) break;
      if (crc == code_block) store_codeblock (++crc);
// encrypt my data
      shuffle_tx_byteorder ();
      xxtea_encode ();
      shuffle_tx_byteorder ();
// send it away
      nRFCMD_Macro ((unsigned char *) &g_MacroBeacon);
      CONFIG_PIN_LED = 1; nRFCMD_Execute (); CONFIG_PIN_LED = 0;
// reset touch sensor pin
      TRISA = CONFIG_CPU_TRISA & ~0x02; CONFIG_PIN_SENSOR = 0;
sleep_jiffies (0xFFFF);
      CONFIG_PIN_SENSOR = 1; TRISA = CONFIG_CPU_TRISA;
// sleep a random time to avoid on-air collosions
sleep_jiffies (rand ());
      i++;
    }
}
```

Possible improvements

- No geometrical data was taken into consideration
- So person wearing tag could run, walk, crawl, as no speed was analysed
- As no distance between stations was calculated
- If data set with new firmware was used, its call for current mood could help with finding sequences

Disclaimer

- Without having sequences sophisticated analysis is not possible
- I was focusing on algorithms for recovering IDs
- I have yet to come with analysing of behaviour of attendees
- Following are the ideas, not something I have already done

Position estimation

- Simple way of telling difference in signal strength: tag sends different power level pings. If reader receives weakest one, it is close. If it receives only the strongest ones, it is far away.
- Tag cycles though all power levels, but every other is the strongest one.
- Estimation of position by using negative knowledge
- Take signal strength. It limits sphere where user can be.
- You get two spheres, on with minimum radius, one with maximum
- By using few of them, from different readers, you can take intersection, and have estimated position. Should be enough to get room from which users are. Then, take data from reader inside room and accept only the weakest levels from this reader — they are the ones inside room
- Use logarithmic radius — stronger signal is twice radius
- This requires knowing exact positions of readers

Calculating direction of person

- Is it possible to calculate direction basing on signal's strength?
- Let's assume that one hemisphere (slightly less) is behind body, hence signal loss
- We could calculate few possibilities in direction, and choose the most probable
- But it would also move our persona. Maybe cooperate with previous position? But it could fast become exponential algorithm.
- It can be based on time sequences; if one goes in one directions, we can assume that he/she is pointed in this direction
- Also we could check all possibilities; there will not be many permutations to check

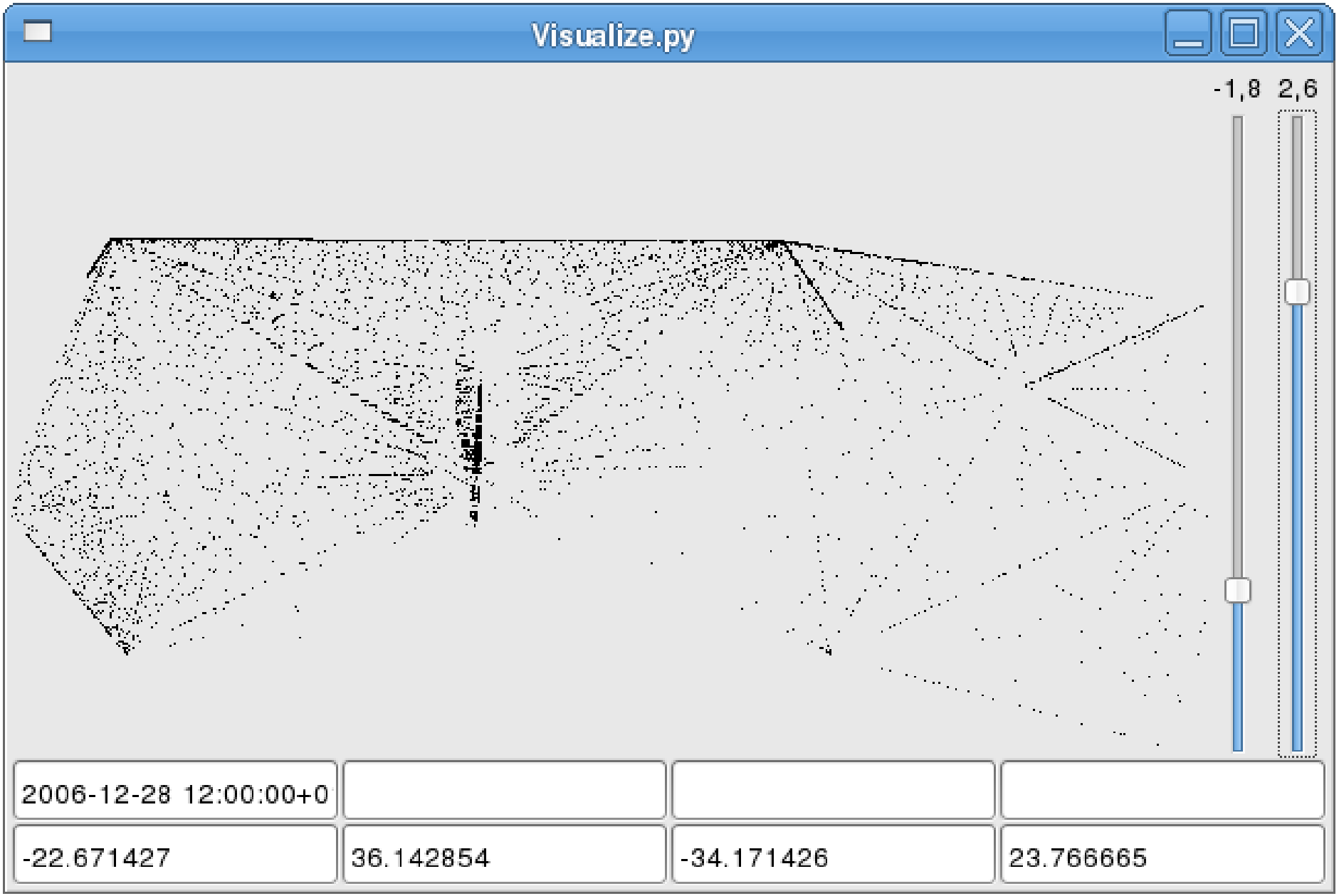
Possible ideas

- For each day and for each ID, calculate when person comes to the BCC and when leaves
- Probably there will be two groups of people — those who leave BCC and those who don't
- When there is disappearing sequence and the new one appears in the same place, probably one reset tracker (by removing battery)

Connections between people

- Analysis of data to see who attended which talks
- Groups of people that attended similar talks
- Groups of friends — people who are close together not only during talks, but also during breaks
- To find friends and colleagues, find tags that are close in space and time for more than defined period of time (so they talk or at least are close in space) more than once (so they are not just standing in a queue for tickets/papers/food)
- Conditions: $\Delta t < 10s$, $distance < 1m$, $\Delta y < 0,5$, id0 != id1, maybe group by id0
- By assuming that people are on the one surface we can ignore third dimension, and just use planar functions offered by PostgreSQL
- Of course, change of position is less important than distance between tags, because fiends can talk and walk together

Visualisation of XML data



The end

- Questions?
- Thank you for your attention