Congestion Control in TCP

Outline

- Overview of RENO TCP
- Reacting to Congestion
- SS/AIMD example

TCP Congestion Control

- The idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit.
  - Once a given source has this many packets in transit, it uses the arrival of an ACK as a signal that one of its packets has left the network, and that it is therefore safe to insert a new packet into the network without adding to the level of congestion.
  - By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking*.

TCP Congestion Control

- In more detail
  - assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity

Congestion Control Overview

- Objective: adjust to changes in the available capacity
- New state variable per connection: **`CongestionWindow`**
  - limits how much data source has in transit

```
MaxWin = MIN(CongestionWindow,
                    AdvertisedWindow)
EffWin = MaxWin - (LastByteSent -

LastByteAcked)
```

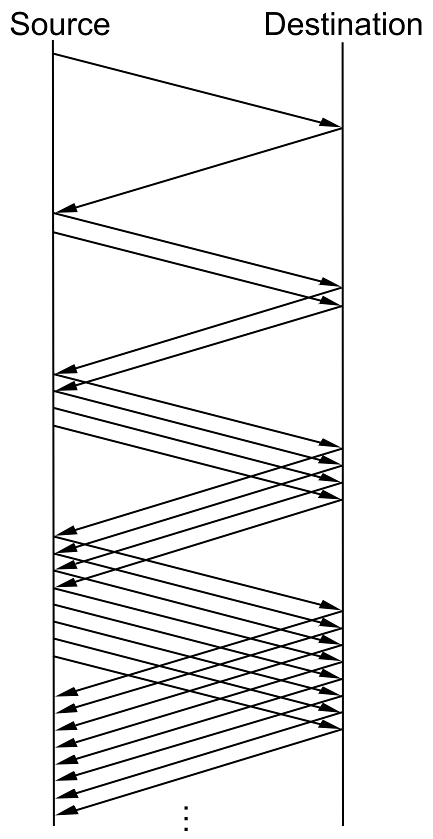
- Idea:
  - increase **`CongestionWindow`** when congestion goes down
  - decrease **`CongestionWindow`** when congestion goes up

TCP RENO Overview

- Standard TCP functions
  - Listed in last lecture: connections, reliability, etc.
- Jacobson/Karels RTT/RTO calculation
- Slow Start phase
- Congestion avoidance phase
  - Additive Increase/ Multiplicative Decrease (AIMD)
  - Fast Retransmit/Fast Recovery

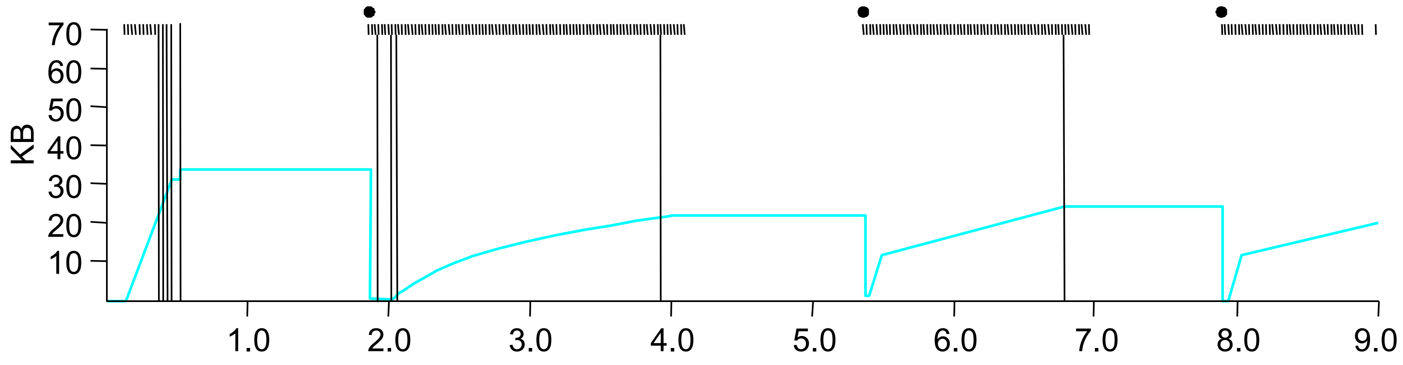
Slow Start

- Objective: determine the available capacity in the first
  - Additive increase is too slow
    - One additional packet per RTT
- Idea:
  - begin with **`CongestionWindow`** = 1 packet
  - double **`CongestionWindow`** each RTT (increment by 1 packet for each ACK)
  - This is exponential increase to probe for available bandwidth
- SSTHRESH indicates when to begin Congestion Avoidance phase



Slow Start contd.

- Exponential growth, but slower than all at once
- Used…
  - when first starting connection
  - when connection goes dead waiting for timeout
- Trace



- Problem: lose up to half a **`CongestionWindow`**'s worth of data

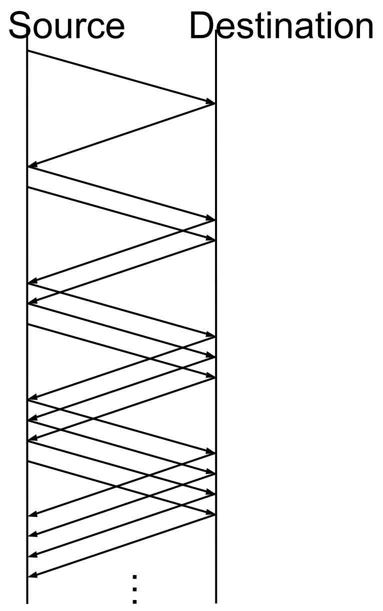
SSTHRESH and CWND

- SSTHRESH called `CongestionThreshold` in book
- Typically set to very large value on connection setup
- Set to one half of `CongestionWindow` on packet loss
  - So, SSTHRESH goes through multiplicative decrease for each packet loss
  - If loss is indicated by timeout, set `CongestionWindow = 1`
    - SSTHRESH and `CongestionWindow` always >= 1 MSS
- After loss, when new data is ACKed, increase CWND
  - Manner depends on whether we're in slow start or congestion avoidance

Congestion avoidance

- How does the source determine whether or not the network is congested?
- Answer: a packet loss is detected
  - Either through a timeout
    - timeout signals that a packet was lost
    - packets are seldom lost due to transmission error
    - lost packet implies congestion
    - RTO calculation is critical
  - Or through duplicate acks (triple)

Congestion Avoidance



- Algorithm
  - increment **`CongestionWindow`** by one packet per RTT (*linear increase*)
  - divide **`CongestionWindow`** by two whenever a timeout occurs (*multiplicative decrease – fast!!*)
  - **`CongestionWindow`** always >= 1 MSS
- In practice: increment a little for each ACK

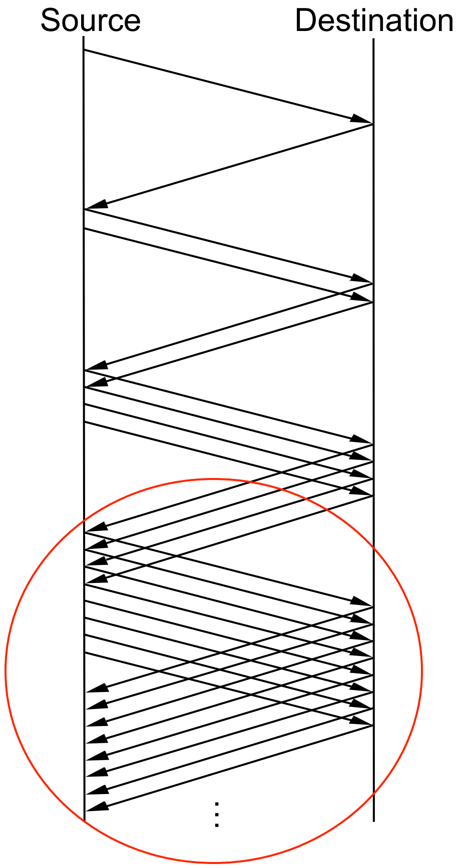
```
Increment = 1/CongestionWindow
CongestionWindow += Increment
```

MSS = max segment size = size of a single packet

What is TCP's sending rate?

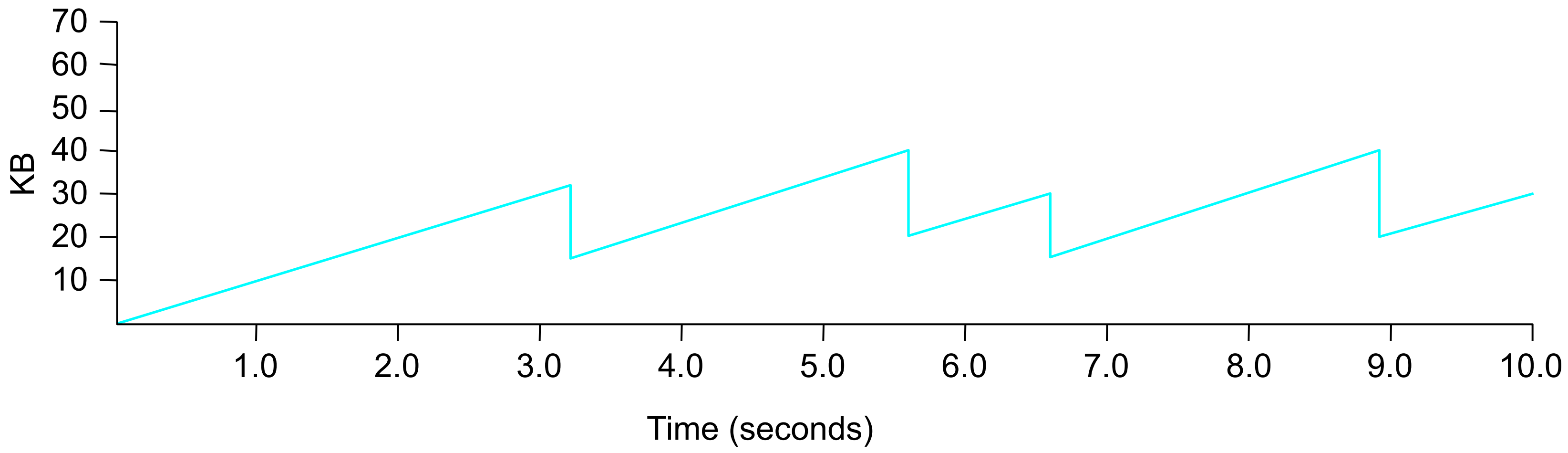
- Roughly WindowSize / RTT

- Sender can roughly send one WindowSize worth of data before and no more, until the first acks come back



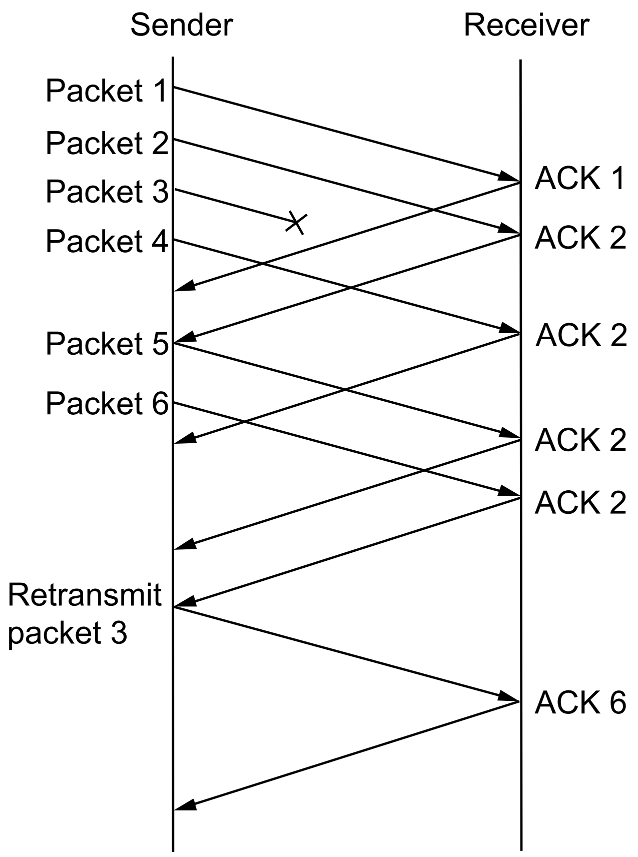
AIMD (cont)

- Trace: sawtooth behavior

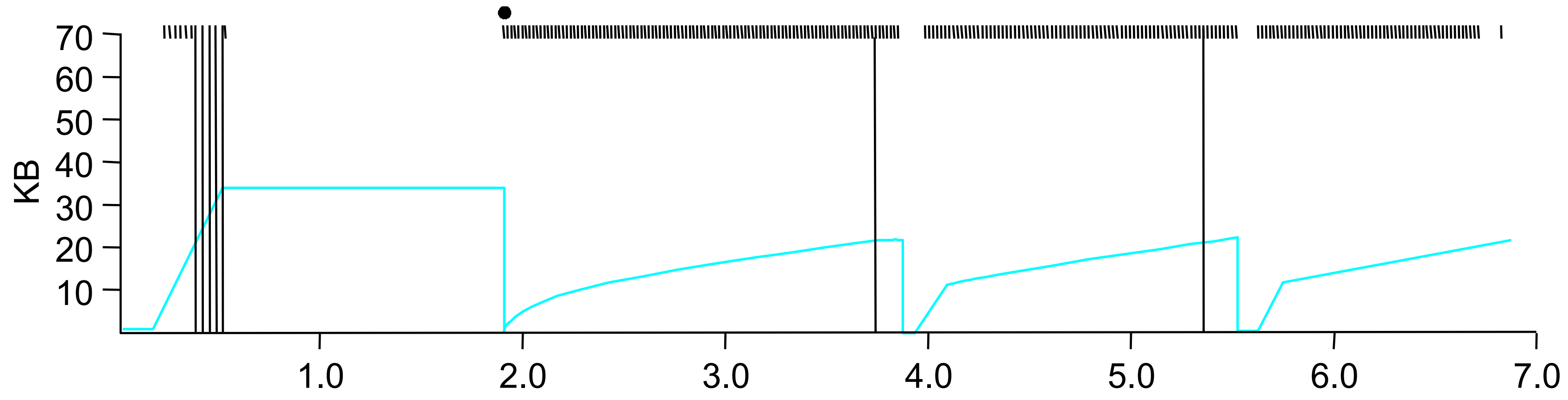


Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use 3 duplicate ACKs to trigger retransmission
- Fast recovery: start at SSTHRESH and do additive increase after fast retransmit



Fast Retransmit Results



- This is a graph of fast retransmit only
  - Avoids some of the timeout losses
- Fast recovery
  - skip the slow start phase in this graph at 3.8 and 5.5 sec
  - go directly to half the last successful **`CongestionWindow`** **`(ssthresh)`**

In summary

- TCP Slow Start:
  - On each new ack:
    - CW = CW + 1 (MSS)
- TCP Congestion Avoidance
  - On each new ack:
    - CW = CW + 1/CW (in MSS)
  - On triple dupack
    - SSThresh = CW/2
    - CW = CW + 3 (MSS)
  - On timeout
    - CW = 1 (MSS)
  - On each additional dupack
    - CW = CW + 1 (MSS)
  - On new ack
    - CW = SSThresh

More on TCP Congestion Control

TCP Congestion Control

- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops
- End-host TCP interprets drops as signs of congestion and slows down → reduces size of congestion window
- But then, periodically probes – or increases congestion window
  - To check whether more bandwidth has become available

Congestion Control Objectives

- Simple router behavior
- Distributed-ness
- Efficiency: $\Sigma x_i(t)$ close to system capacity
- Fairness: equal (or propotional) allocation
  - Metric = $(\Sigma x_i)^2/n(\Sigma x_i^2)$
- Convergence: control system must be stable

Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
    - Window(t + 1) = a + b Window(t)
    - Different a_i/b_i for increase and a_d/b_d for decrease
- Various reaction to signals possible
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?
    - Consider two end hosts vying for network bandwidth

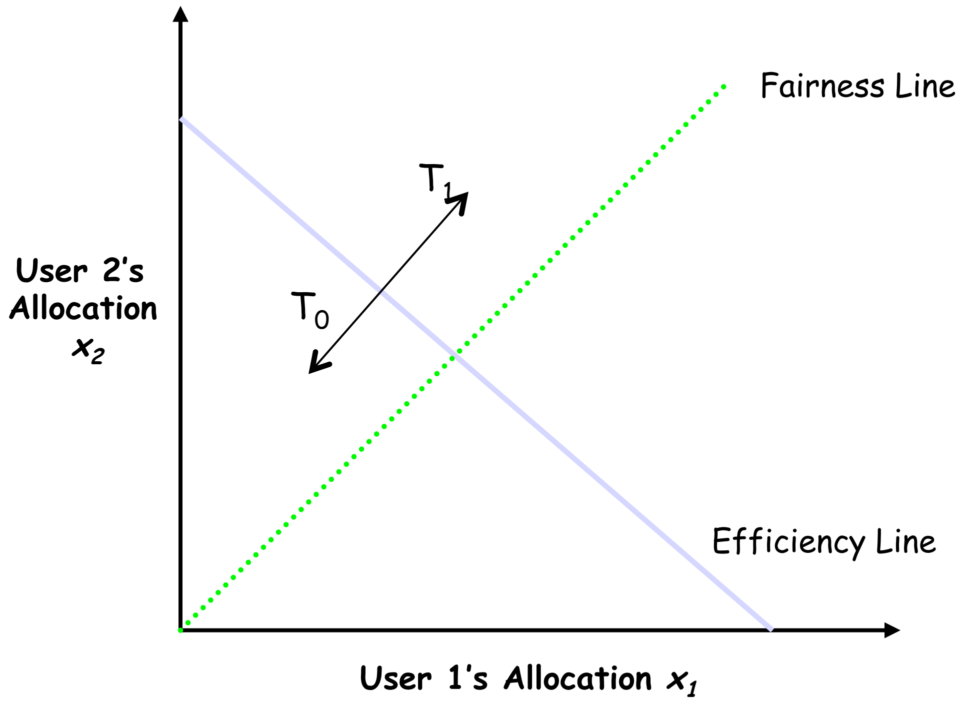
Four alternatives

- Additive Increase Additive Decrease (AIAD)
- Additive Increase Multiplicative Decrease (AIMD)
- Multiplicative Increase Additive Decrease (MIAD)
- Multiplicative Increase Multiplicative Decrease (MIMD)

- So why pick AIMD?

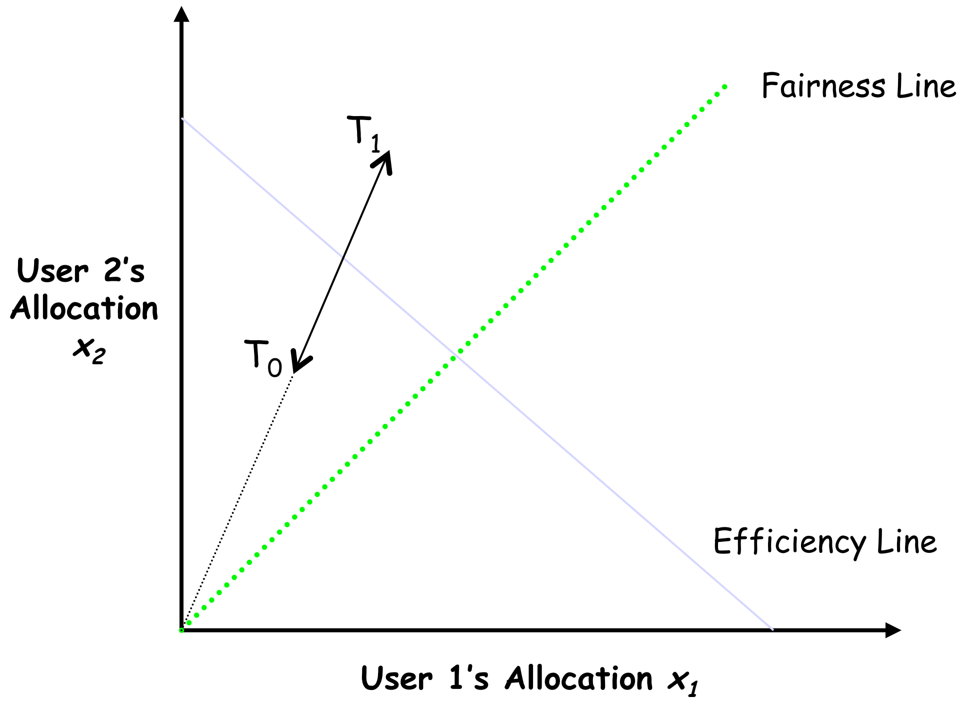
Additive Increase/Decrease

- Both X_1 and X_2 increase/ decrease by the same amount over time
  - Additive increase improves fairness and additive decrease reduces fairness

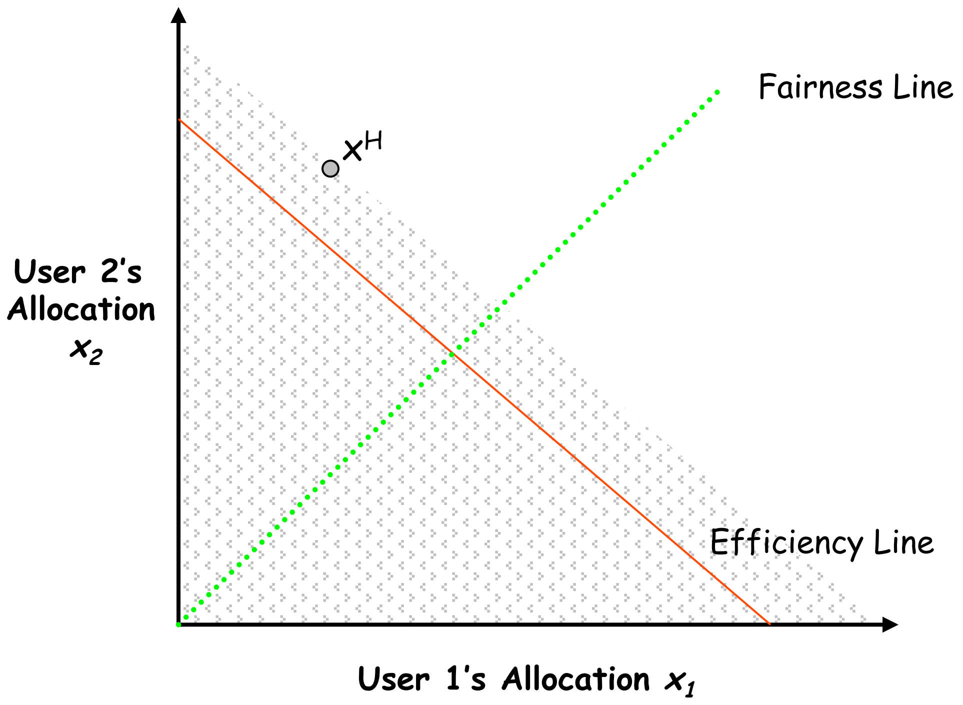


Multiplicative Increase/Decrease

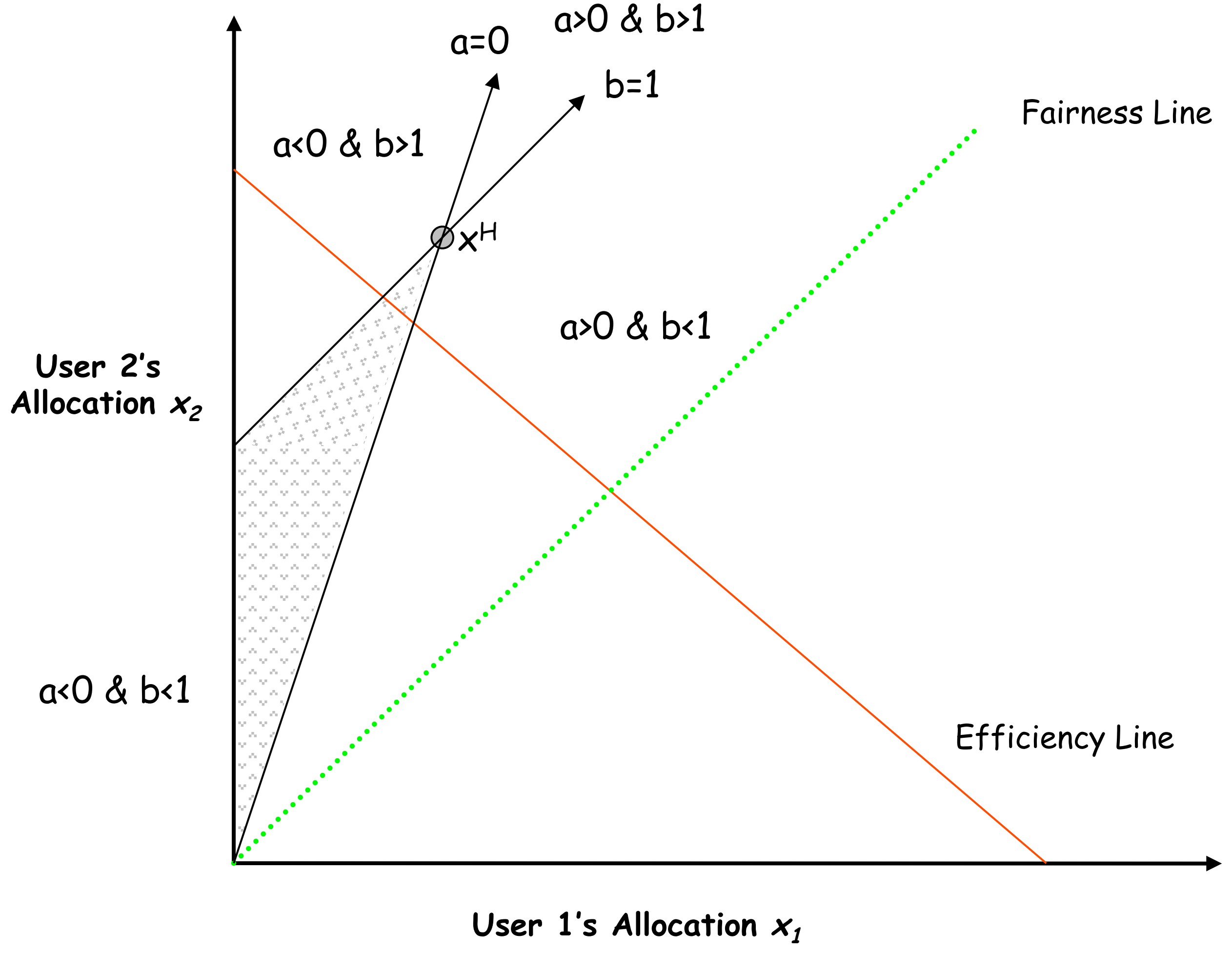
- Both X_1 and X_2 increase by the same factor over time
  - Extension from origin – constant fairness



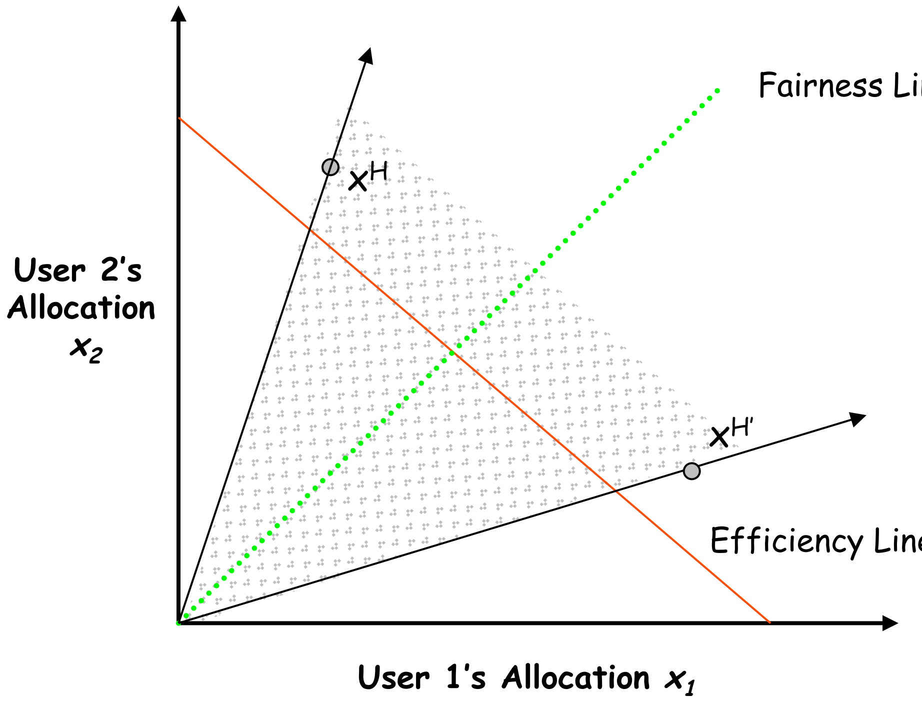
Convergence to Efficiency



Distributed Convergence to Efficiency

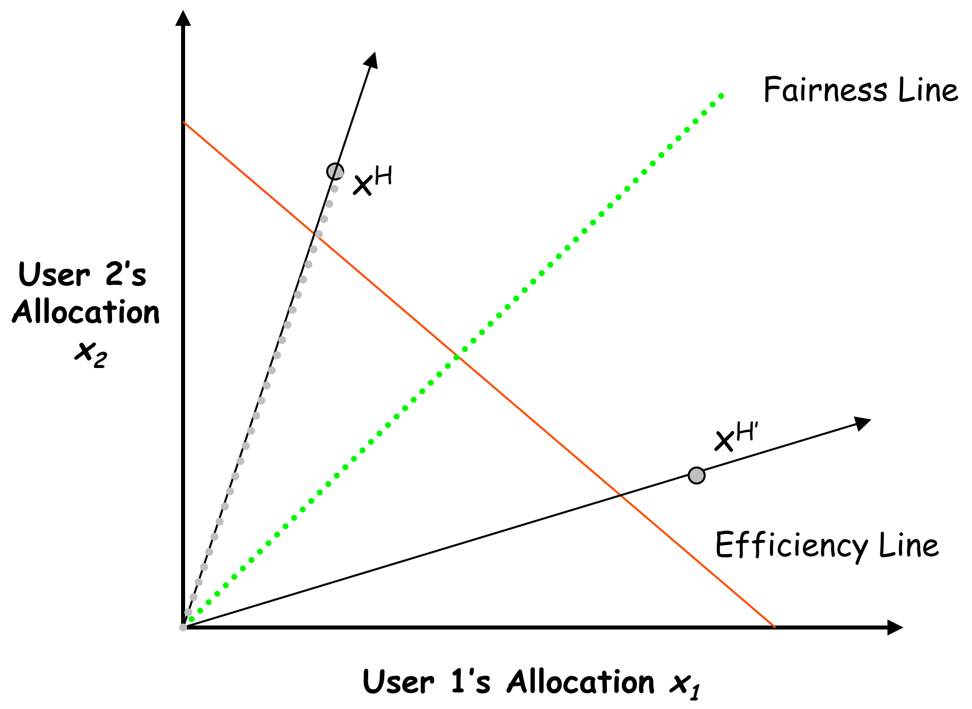


Convergence to Fairness



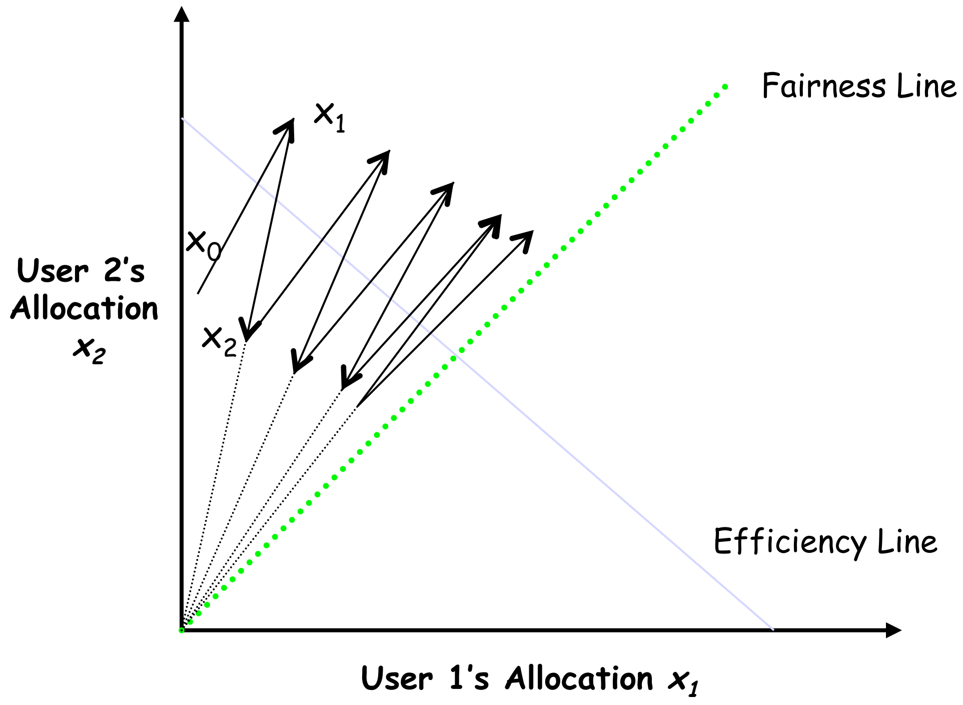
Convergence to Efficiency & Fairness

- Intersection of valid regions
- For decrease: a=0 & b < 1



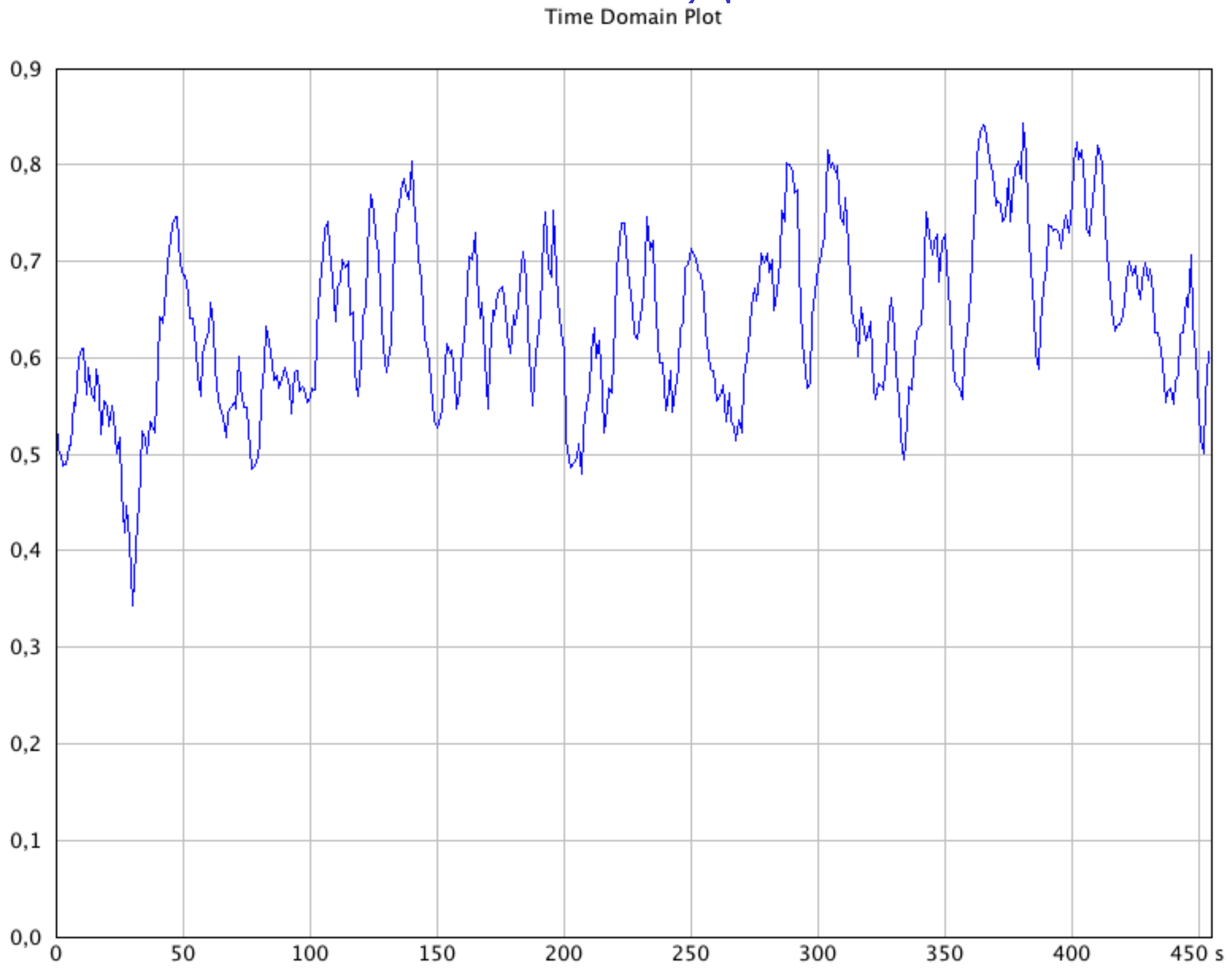
What is the Right Choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase (MAIMD)
  - AIMD moves towards optimal point



Exponentially Weighted Moving Average

Smoothing of data



Smoothing of data

- Simple moving average

$$s_t = \frac{1}{k}\sum_{n=0}^{k-1} x_{t-n} = \frac{x_t + x_{t-1} + x_{t-2} + \cdots + x_{t-k+1}}{k} = s_{t-1} + \frac{x_t - x_{t-k}}{k},$$

Smoothing of data

- Weighted moving average

$$s_t = \sum_{n=1}^{k} w_n x_{t+1-n} = w_1 x_t + w_2 x_{t-1} + \cdots + w_k x_{t-k+1}.$$

Smoothing data

- Exponentially Weighted Moving Average (EWMA)

$$s_0 = x_0$$
$$s_t = \alpha x_t + (1-\alpha)s_{t-1}, \; t > 0$$

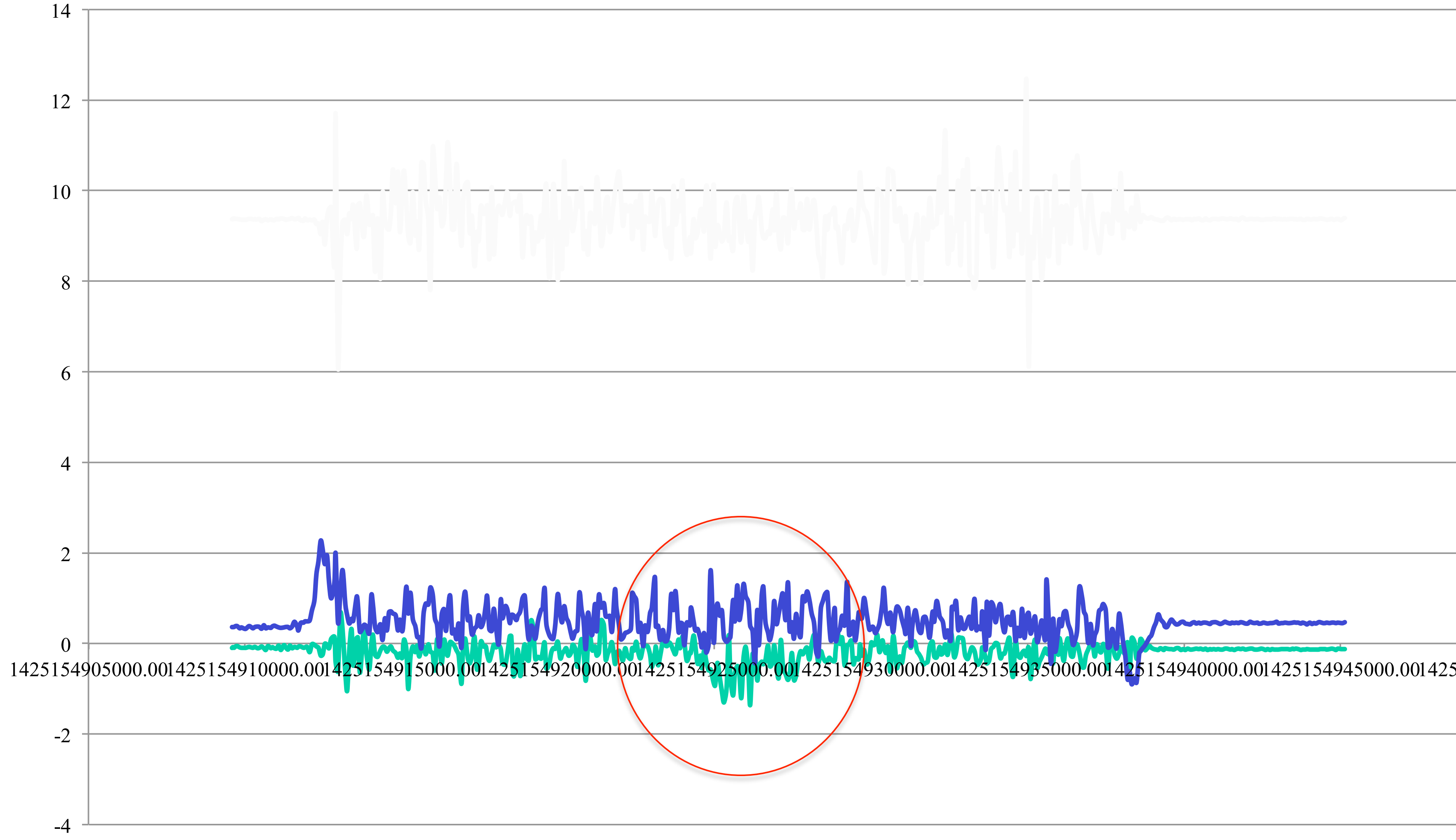
Smoothing data

- Exponentially Weighted Moving Average (EWMA)

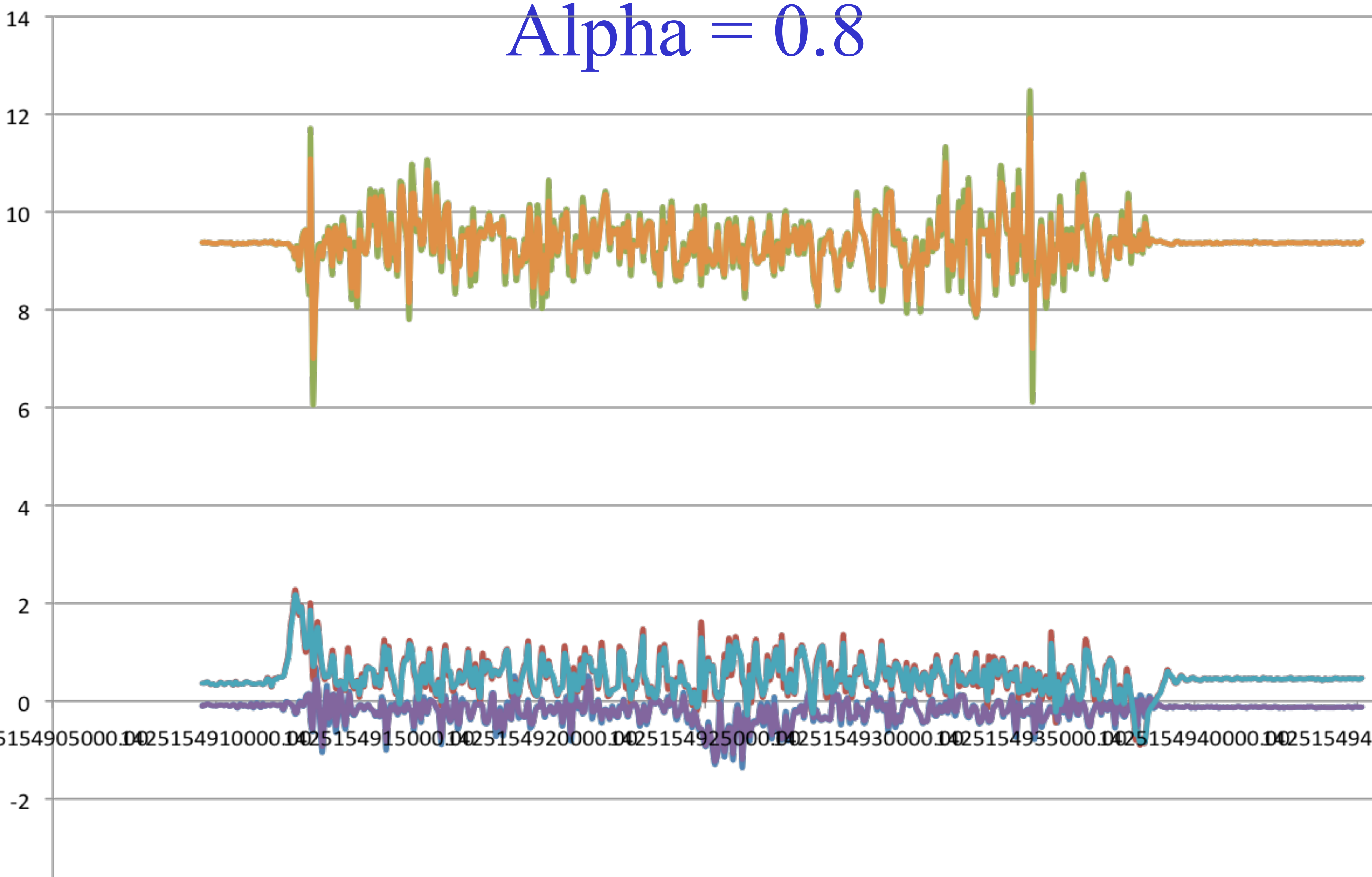
$$\begin{aligned} s_0 &= x_0 \\ s_t &= \alpha x_t + (1-\alpha)s_{t-1},\ t > 0 \end{aligned}$$

$$\begin{aligned} s_t &= \alpha x_t + (1-\alpha)s_{t-1} \\ &= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2 s_{t-2} \\ &= \alpha\left[x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + (1-\alpha)^3 x_{t-3} + \cdots + (1-\alpha)^{t-1}x_0\right] + (1-\alpha)^t s_0. \end{aligned}$$

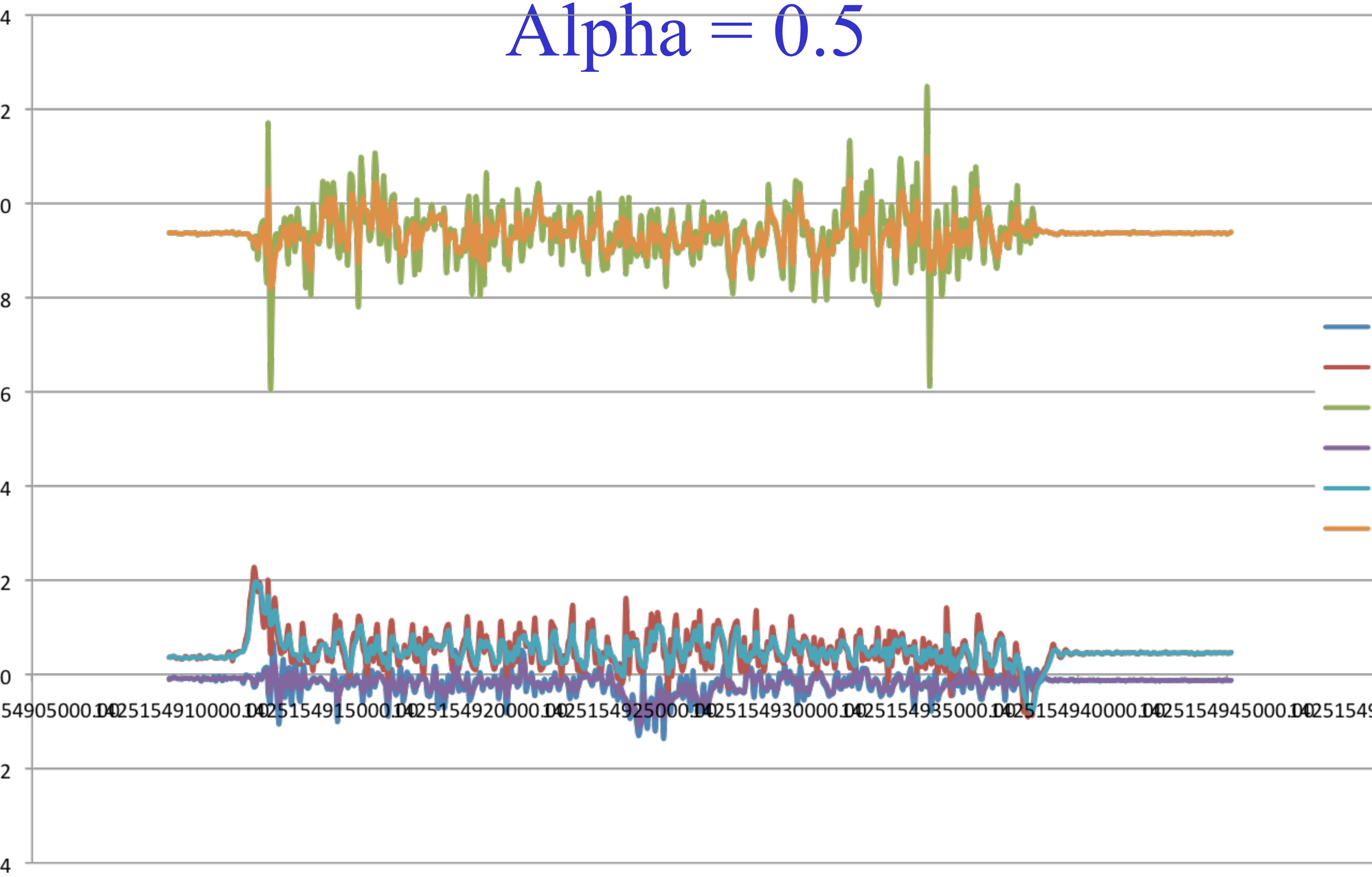
Accelerometer



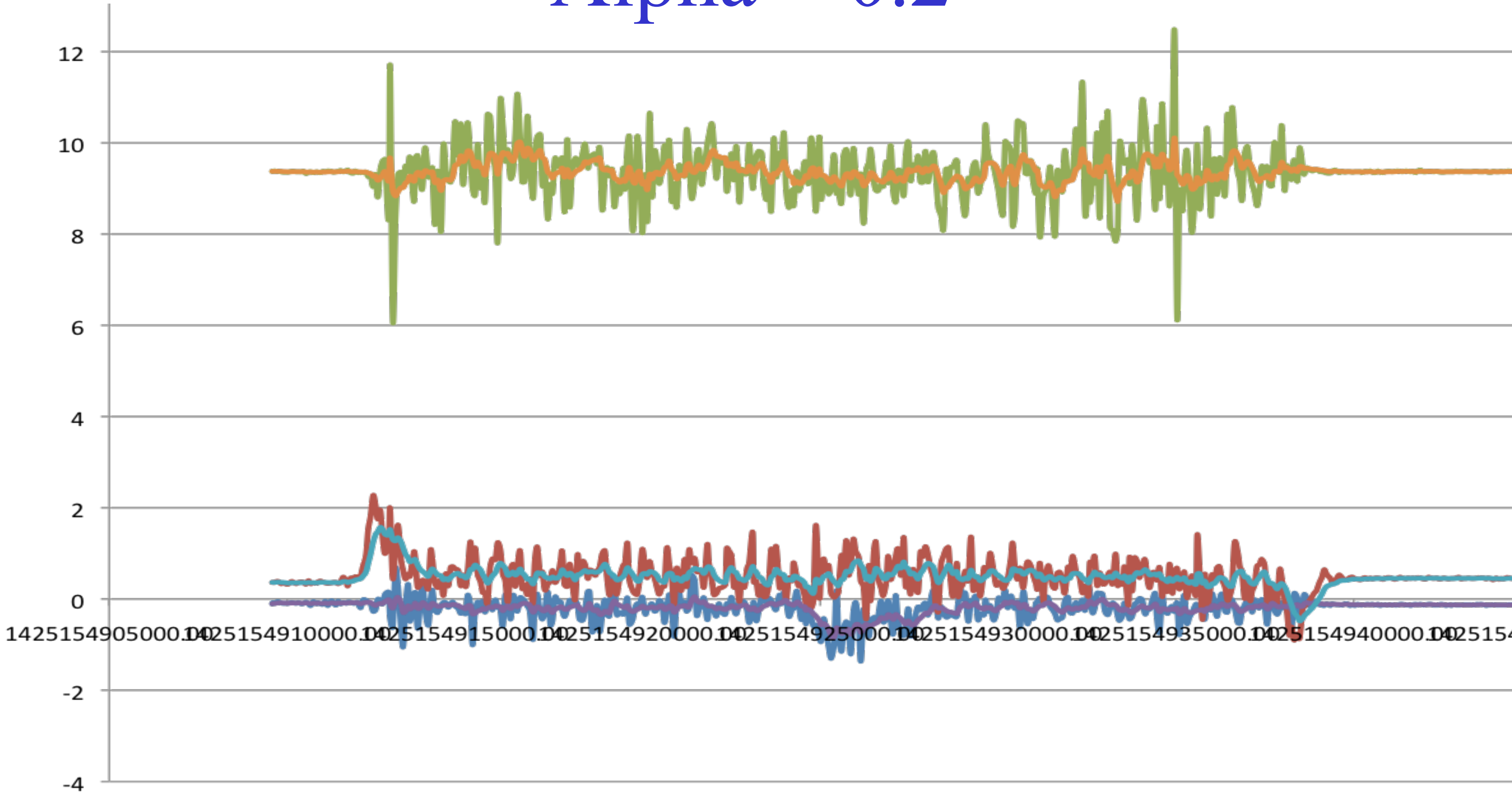
Alpha = 0.8

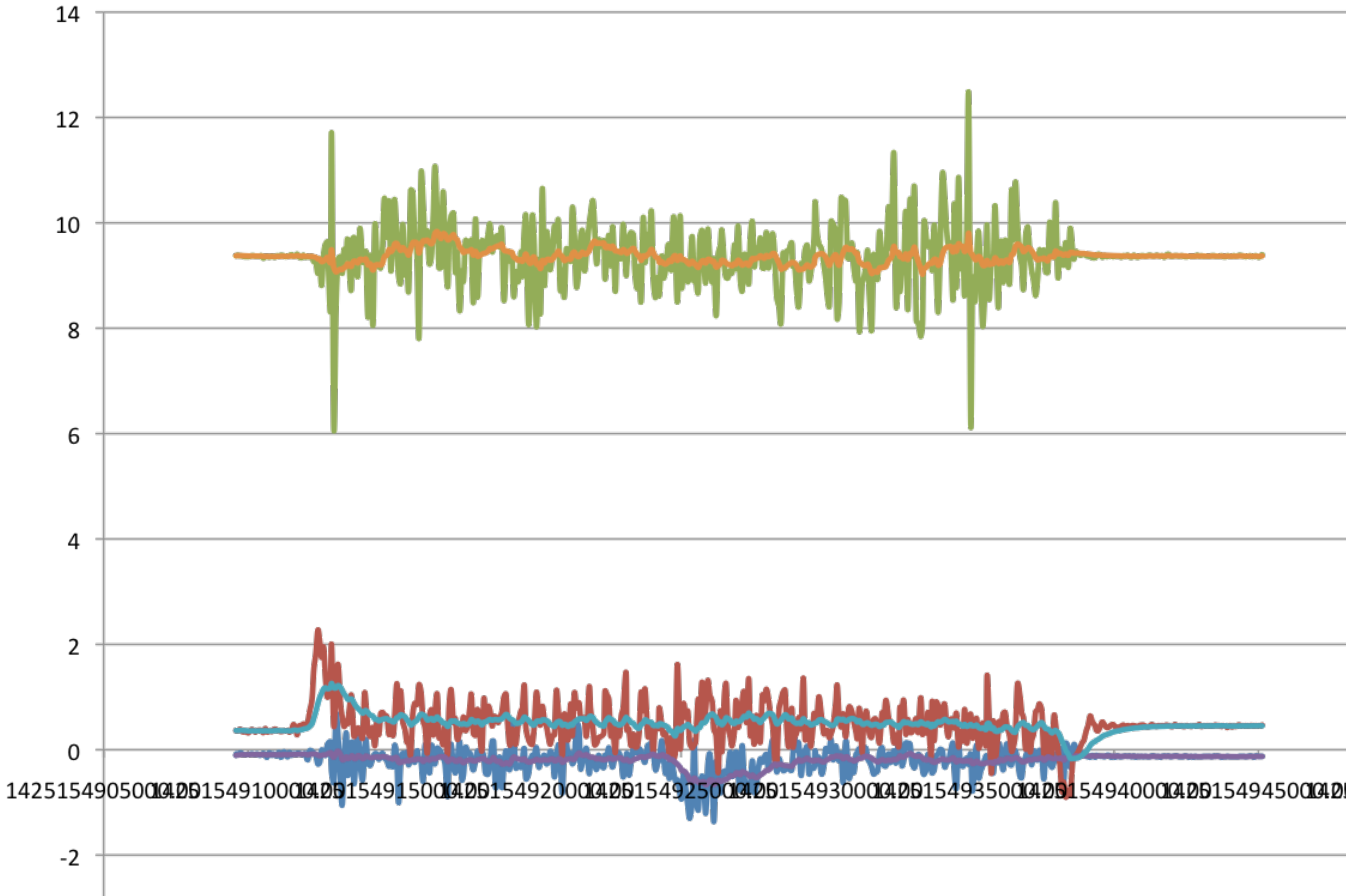

14
12
10
8
6
4
2
0
-2
5154905000.1402515491000.1402515491500.1402515492000.1402515492500.1402515493000.1402515493500.1402515494000.1402515494

Alpha = 0.5

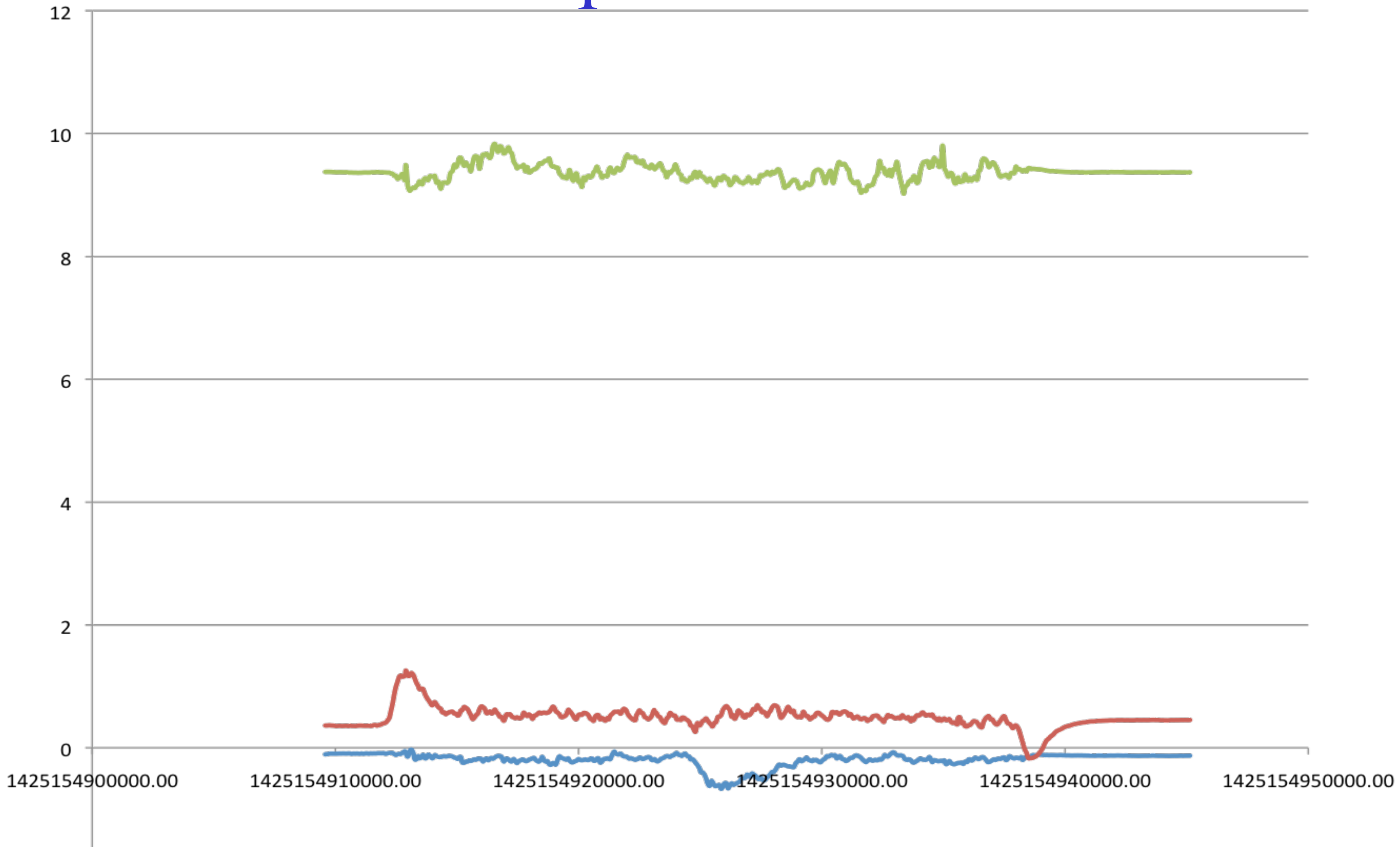

4
2
0
8
6
4
2
0
2
4
54905000.1402515491000.1402515491500.1402515492000.1402515492500.1402515493000.1402515493500.1402515494000.1402515494500.140251549

Alpha = 0.2


12
10
8
6
4
2
0
-2
-4
1425154905000.0
1425154910000.0
1425154915000.0
1425154920000.0
1425154925000.0
1425154930000.0
1425154935000.0
1425154940000.0
142515

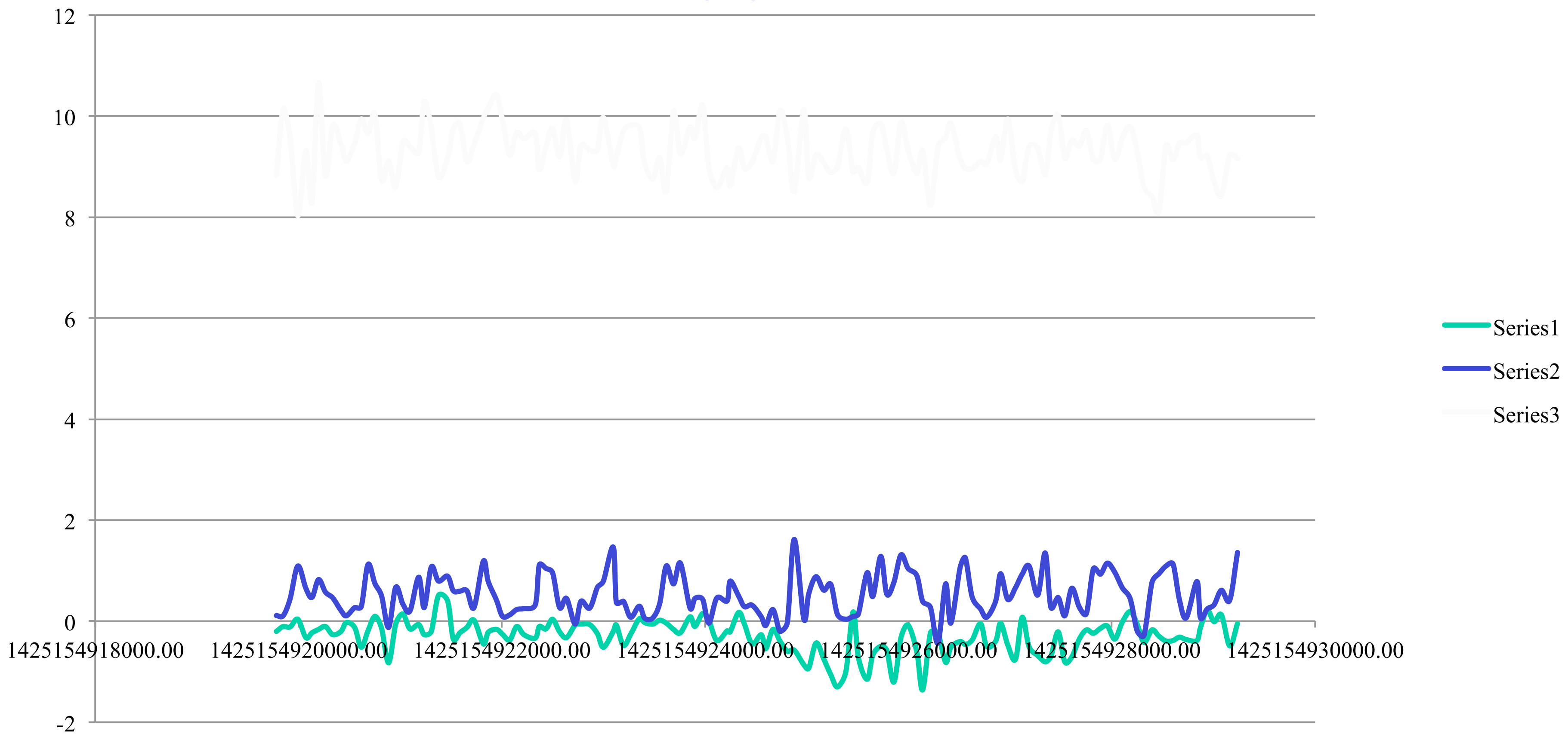

14
12
10
8
6
4
2
0
-2
1425154905000
1425154910000
1425154915000
1425154920000
1425154925000
1425154930000
1425154935000
1425154940000
1425154945000

Alpha = 0.1



12
10
8
6
4
2
0
1425154900000.00
1425154910000.00
1425154920000.00
1425154930000.00
1425154940000.00
1425154950000.00

Zoom in



12
10
8
6
4
2
0
-2
1425154918000.00
1425154920000.00
1425154922000.00
1425154924000.00
1425154926000.00
1425154928000.00
1425154930000.00
Series1
Series2
Series3