Remote Time Tracker

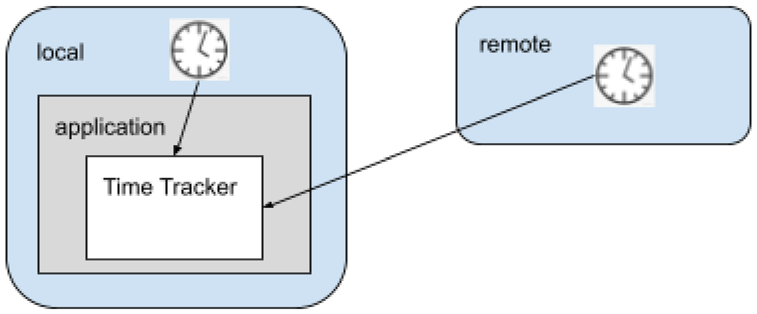
SailGP - v.2, 8 June 2021

Problem Statement

Suppose we have an application running on a computer with a readily accessible system clock that is not synchronized to any external time reference. We can query that time (the **local** time) whenever we want.

We are connected to another computer across a communication link, and that computer has a very accurate time reference. That computer periodically sends us a message telling us what its time is there (the **remote** time). We note what our clock says when we receive one of those messages.

At any given local time, we want to be able to estimate what time it is on the remote computer.



There are some complications: the communication link has a variable amount of delay; there is an offset between our clocks; our clocks may run at slightly different rates (but always within 10%); the remote clock may suddenly pause, or jump to a completely different time; and occasionally (less than 10% of the time) the remote computer reports complete garbage for the time.

The Testbed Application

The TimeTrackerTest Windows C# console app simulates this environment. It uses an instance of a class that implements the IGenerator interface to return a sequence of local + remote time pair messages, as if we were receiving messages from a remote device and timestamping them with our local time when we receive them. An instance of a class implementing the ITracker interface receives these messages and updates its state. At the same time, we periodically query the ITracker, providing a local time and requesting its estimate of the corresponding remote time.

A sample implementation - the SimpleTracker class - is provided.

Process

Test your tracker by uncommenting each of the various Generator constructors in Main() in turn and running the app:

```
...
// ----
// Here are several different types of incoming message with different problems.
// Use one of them.
// ----

// Best case - 1x speed, no delays, no jumps in time, no noise
IGenerator gen = new GenIdeal { m_bias = 0.3 };

// Add up to 0.3 secs delay in the receipt of the message
//IGenerator gen = new GenDelays { m_delay_range = 0.3 };

// The time rate is off by up to 10%
//IGenerator gen = new GenDelays { m_rate = 1.1, m_delay_range = 0.3 };
//IGenerator gen = new GenDelays { m_rate = 0.9, m_delay_range = 0.3 };
...
```

You should be able to increase the n_samples value at the beginning of Main and see how your tracker behaves over a long period of time.

Feel free to automate any part of the process if it helps.

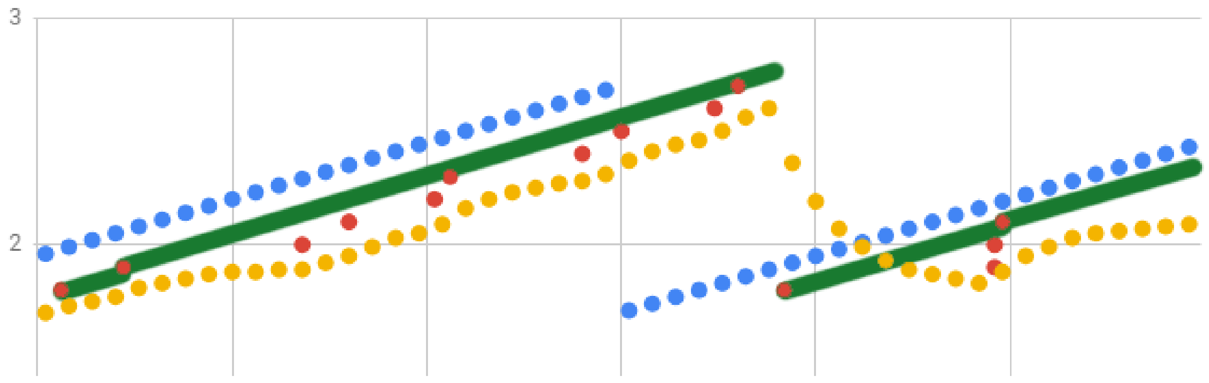
Goals

In order of decreasing importance

- **The tracker should provide a steady linear response matching the actual graph**
- **It should have minimal velocity fluctuations**
- It should ignore occasional garbage messages
- It should ignore delays and rely on those messages showing less delay
- When there are piecewise changes to the remote time, the tracker should adjust quickly and get back to linear
- If possible, it should detect mis-matched local/remote rates and adjust

A good implementation is simple, clear (verifiable as correct by inspection), efficient and fast.

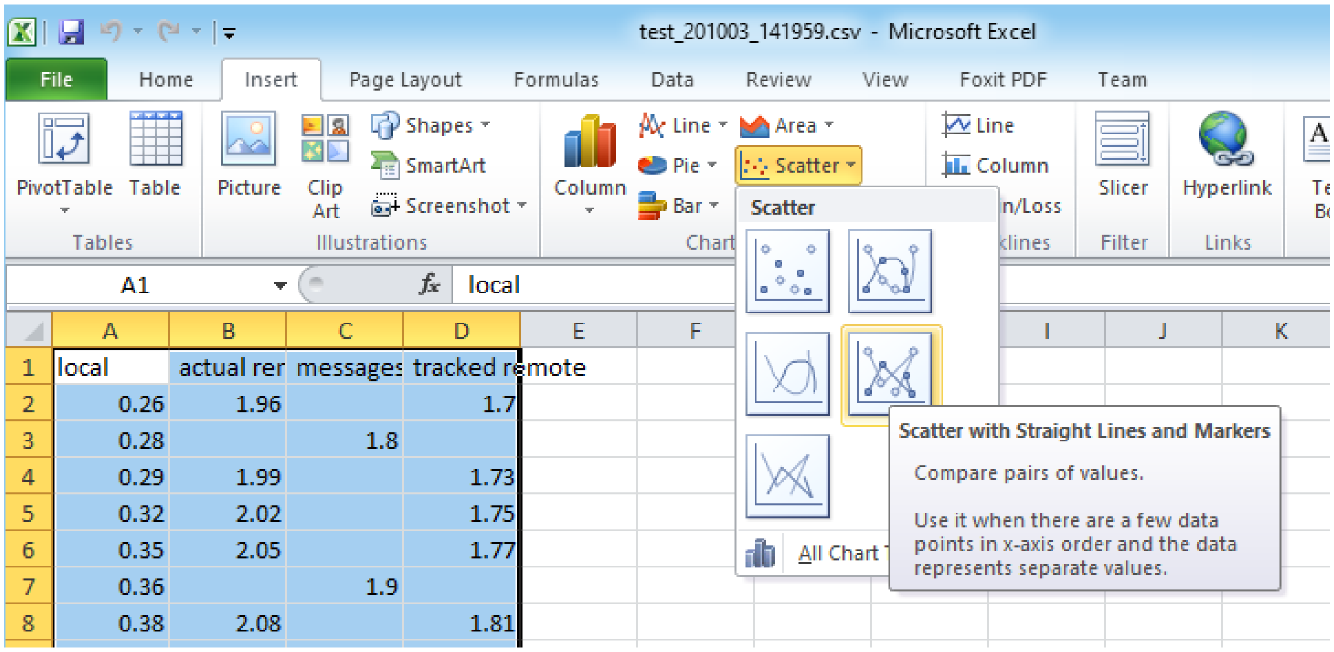
Producing the green line below would be a great result



Some parts of this are harder than others. If you can't implement a working solution to some aspects, propose how you might approach it.

Running a Test and Evaluating the Results

1. Write your own ITrack implementation class and assign it to the ITracker track variable in the main() method instead of the SimpleTracker instance.
2. Run the program. It generates a text file in the current directory (usually bin/debug) named something like test_201003_141959.csv, which is the local time in <yyMMdd_HHmmss> format.
3. **If you have Excel,**
    a. open the csv file in Excel,
    b. select all cells (ctrl & A),
    c. create a scatter plot by clicking Insert > Scatter > Scatter with Straight Lines and Markers



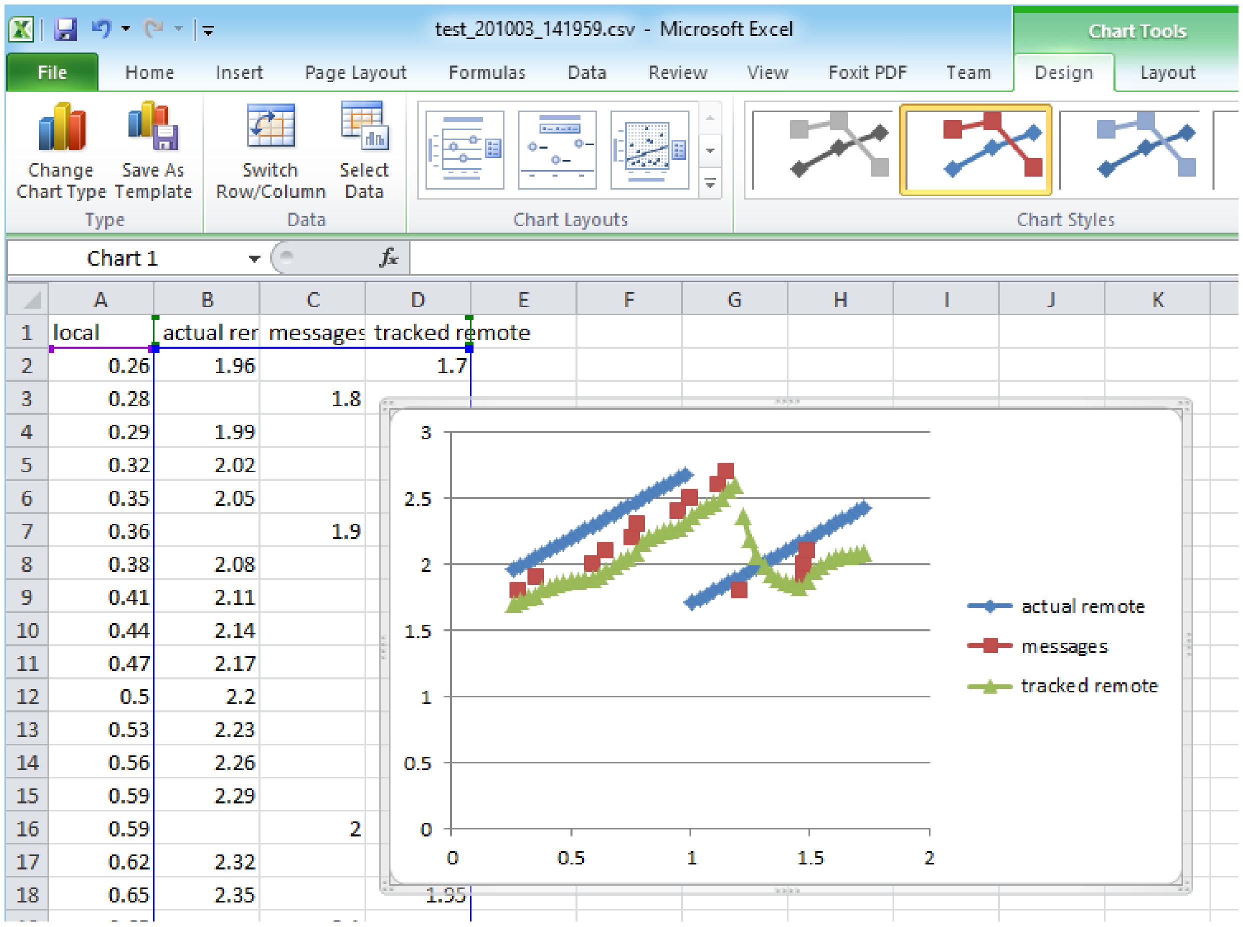
	A	B	C	D	E
1	local	actual rer	messages	tracked remote	
2	0.26	1.96		1.7	
3	0.28		1.8		
4	0.29	1.99		1.73	
5	0.32	2.02		1.75	
6	0.35	2.05		1.77	
7	0.36		1.9		
8	0.38	2.08		1.81	

Scatter with Straight Lines and Markers

Compare pairs of values.

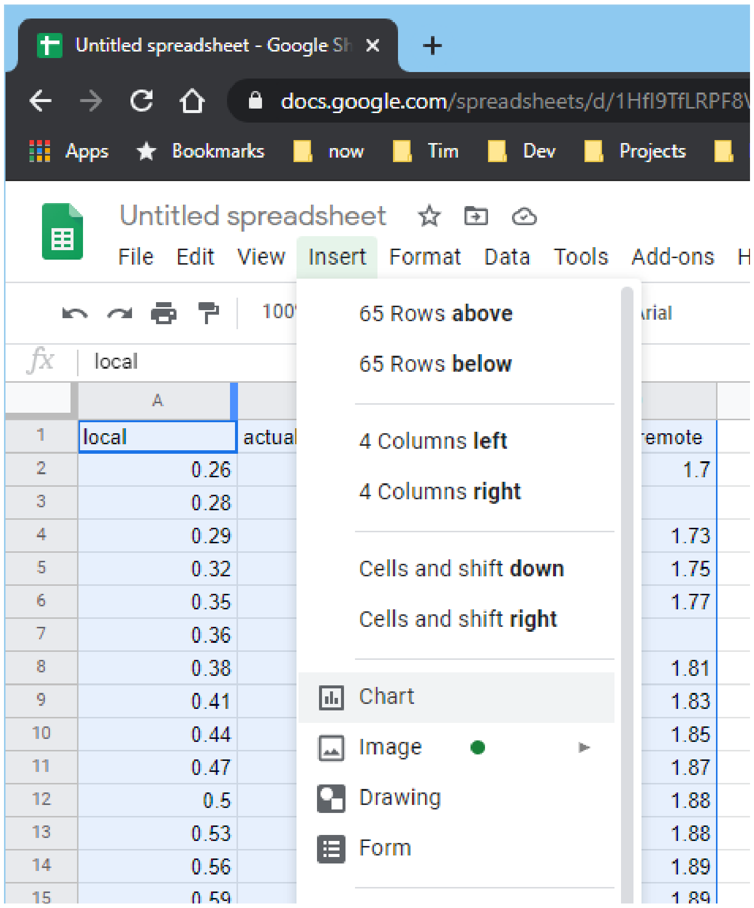
Use it when there are a few data points in x-axis order and the data represents separate values.

d. You should see a chart. The axes are **remote** time on the vertical axis as a function of **local** time on the horizontal. The Red markers are the (local,remote) time reports, and the green line is the tracker's attempt to follow that info. The blue line is the actual remote time for each estimate the tracker returned.



	A	B	C	D
1	local	actual rer	messages	tracked remote
2	0.26	1.96		1.7
3	0.28		1.8	
4	0.29	1.99		
5	0.32	2.02		
6	0.35	2.05		
7	0.36		1.9	
8	0.38	2.08		
9	0.41	2.11		
10	0.44	2.14		
11	0.47	2.17		
12	0.5	2.2		
13	0.53	2.23		
14	0.56	2.26		
15	0.59	2.29		
16	0.59		2	
17	0.62	2.32		
18	0.65	2.35		1.95

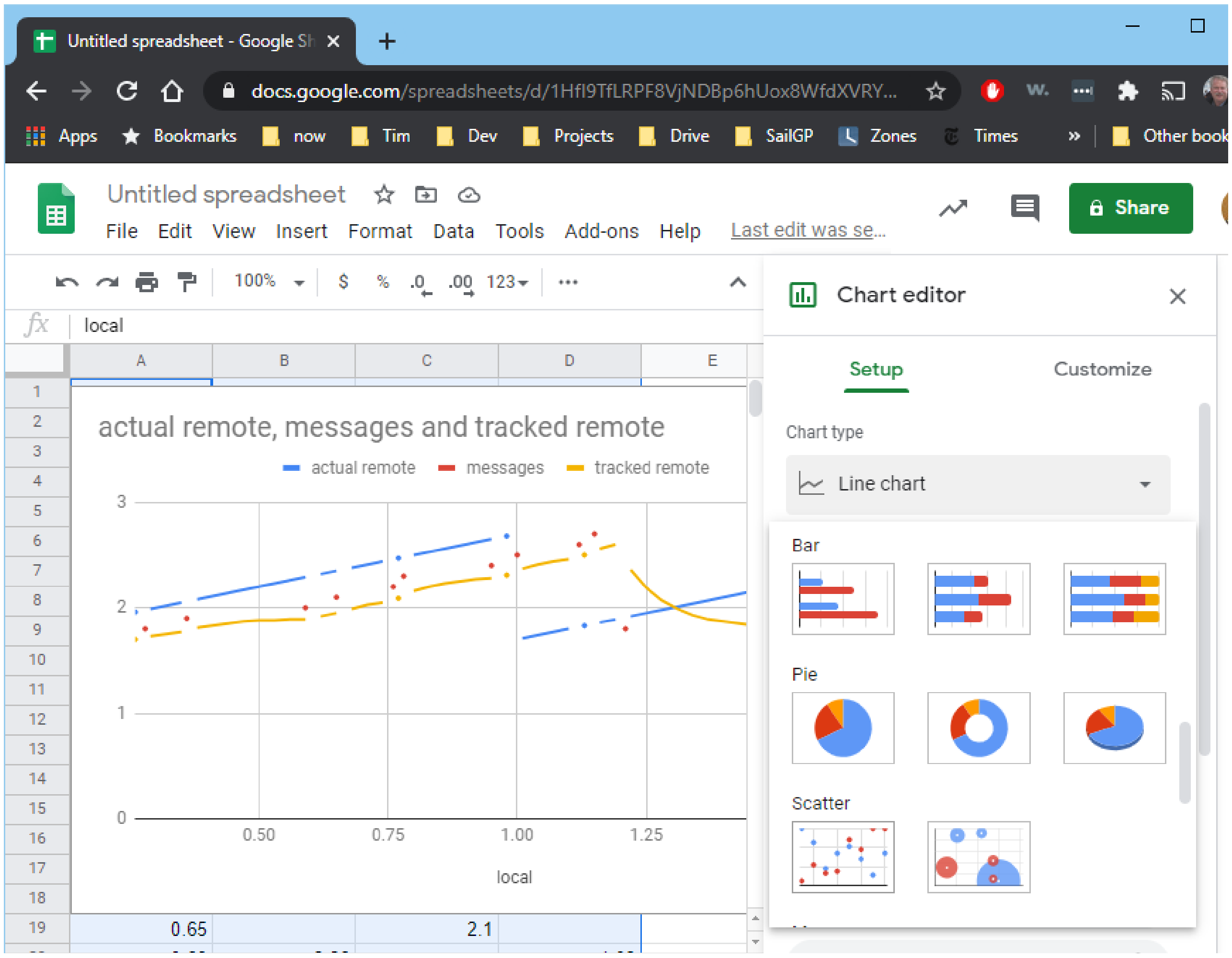
4. **If you have access to Google tools,**
    a. Create a new Google Chart
    b. Select File > Import, then click the Upload tab.
    c. Drag the csv file from Windows Explorer to Google, or click Select a file and browse to it.
    d. Click the Import data button - the default settings should work OK.
    e. Click Ctrl & A to select all cells
    f. Click Insert > Chart:



fx local

	A		
1	local	actual	remote
2	0.26		1.7
3	0.28		
4	0.29		1.73
5	0.32		1.75
6	0.35		1.77
7	0.36		
8	0.38		1.81
9	0.41		1.83
10	0.44		1.85
11	0.47		1.87
12	0.5		1.88
13	0.53		1.88
14	0.56		1.89
15	0.59		1.89

g. In the Chart Editor box that appears, use the Chart Type chooser to select Scatter:



h. Here red dots are the (local,remote) messages, yellow is the tracker's estimation for a range of local times, and blue is the actual truth at those times.