Total Blocking Time

Author: dproy@chromium.org, tdresser@chromium.org
Created: July 29, 2019
[Lighthouse implementation and config to run it]

Total Blocking Time is a companion metric to Time to Interactive. This document introduces the definition and detailed analysis of its behavior.

Definition

- For each task, consider all time except for the first 50ms to be "blocking time".
- Total Blocking Time := Sum of all the blocking time that fall between FCP and TTI (see below for justification of these bounds).

Caveat: Total Blocking Time is intended to be a lab metric, where there is no user input. In the wild, user input will generate extra tasks that may not part of loading, and it's better to track input delay directly by instrumenting the site.

Deciding time interval under consideration

Start Point

Options:

- Navigation start
- First Contentful Paint (FCP)

Counting long tasks between navigation start and FCP is may provide the wrong incentive, because

(a) until FCP has happened the user is unlikely to perform any interaction on the page, so long javascript tasks at this point do not negatively affect user experience.
(b) the site should be focused on getting to FCP as quickly as possible, and may be able to do so quicker without having to worry about breaking up long app initialization tasks.

We therefore **pick FCP as the start point of the interval.**

End Point

Options:

- Time to Interactive:
- Time to Interactive + n seconds: we looked at n = 10
- Infinity, or end of tracing.

The ideal end point would be infinity, but we cannot keep tracing indefinitely. In reality, infinity means whenever we stopped tracing, but then the metric is dependent on how long we decided to trace which is not a property of the web page; this is undesirable.

To decide between TTI and TTI + 10s, we looked at the data to see how often it made a difference. On mobile, out of 7380 pages we looked at, when we changed our end point from TTI to TTI + 10s,

- Only on 436 pages (5.9% of pages), TBT value changed at all.
- Only on 79 pages (1.1% of pages), TBT value changed by more than 300ms.

Phrasing another way, on mobile, **94%** of the time tracing for 10 more seconds made **no difference**, and **99%** of cases it didn't make a difference bigger than **300ms**. On desktop, these numbers are **95% and 99.8%** respectively.

Tracing for longer than needed has multiple downsides, including increased infrastructure costs and longer delays in computing metrics, and the benefits here clearly do not outweigh the costs. **We therefore pick TTI as the end point**.

Metric Analysis: Key results

To understand this metric better, we

- looked into the behavior of this metric across top 10k sites on both desktop and mobile,
- analyzed variability of the metric on top 1k sites on mobile.

Key results:

- **There are lots of sites with very low TBT.**
  - 25% of sites on desktop, and 7% sites on mobile have zero TBT. Some of these are 404 pages, but also a lot of these are landing pages with very little javascript like wikiquote.org.
- **There are lots of sites with very high TBT.**
  - 99th percentile TBT on Nexus 5X mobile is 15.5 ***seconds*****!** When we looked at examples of sites we high values , they turned out to be either synchronous XHRs or layout thrashing.
- **TBT distribution is largely independent of TTI, so TBT provides additional value in representing interactivity of a site.**
  - We grouped sites into TTI buckets of 10 second width, and in each bucket there was a *very* wide range of TBT (for example, 500ms to 6000ms at the 10th to 90th percentile, when TTI was between 50-60s)
- **TBT is more noisy for fast sites, but less noisy for slow sites.**
  - TBT has high relative standard deviation when TBT values are low, but low relative variance when TBT values are high.
  - TBT relative standard deviation is higher than TTI when TTI < 10s, but lower when TTI > 10s.
  - In general, TBT becomes less noisy as the metric value increases, which is the opposite behavior of TTI.
- **Variability gets worse if we raise the power of blocking time to 1.5/2.0 before summing up.**

Web page behaviour across TBT value range

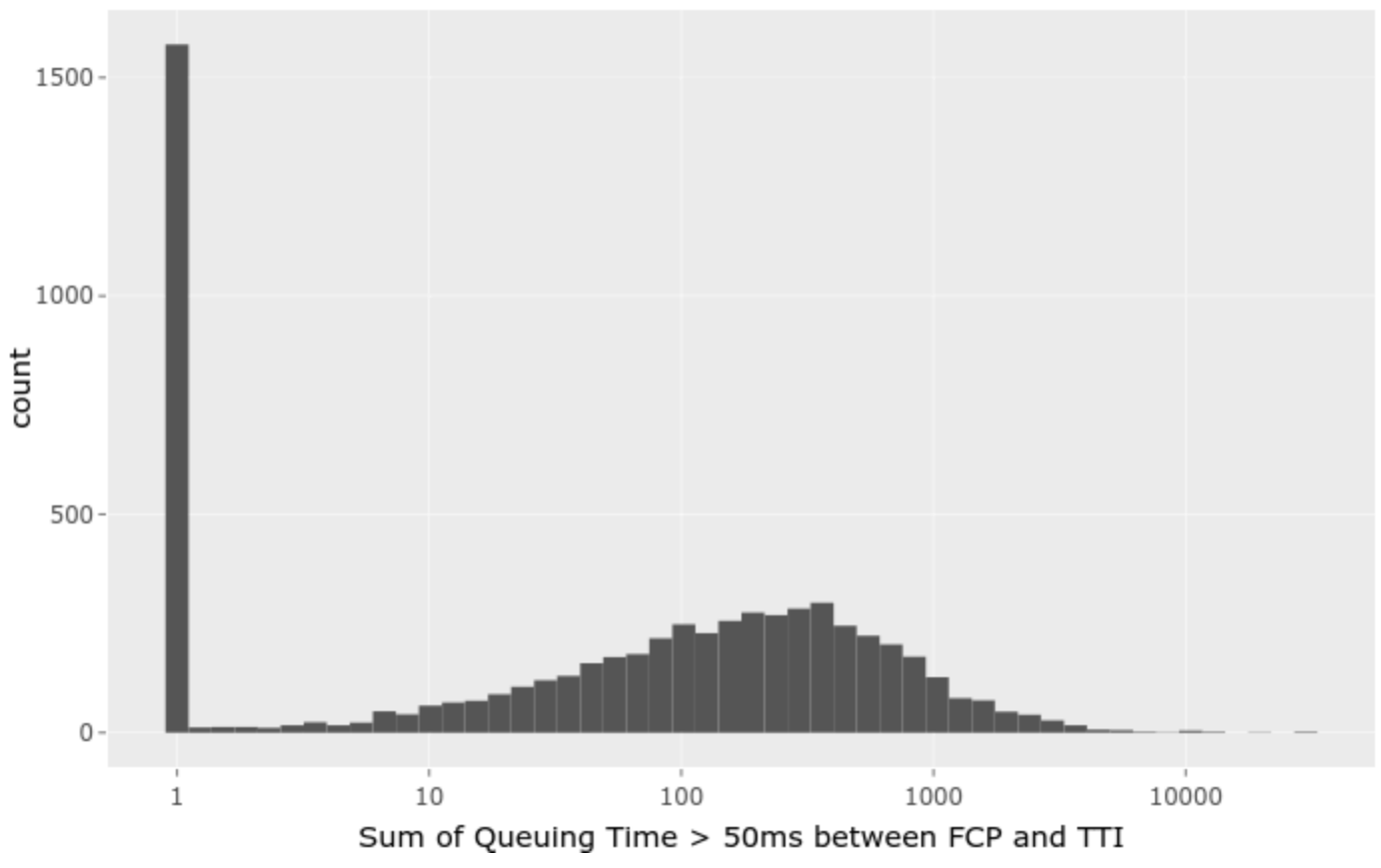
We wanted to understand TBT behavior across a wide range of sites.

Data collection setup

- We loaded Alexa top 10k pages on both mobile and desktop.
- We filtered out data that errored out for various infrastructure/network issues (it wasn't worth it to invest the time in debugging these failures.) Ended up with 6.3k desktop sites, and 7.3k mobile sites.
- Mobile runs were performed on Nexus 5X devices, and under a 4G-ish network throttling (4mbps down, 3mbps up, 20ms round trip latency.)
- Desktop runs were performed on linux desktops, under an average wifi speed network throttling (30mbps down, 15mbps up, 2ms round trip latency)

Desktop

Distribution (x-axis is log scaled. Added 1ms to TBT values so 0 values won't be undefined):



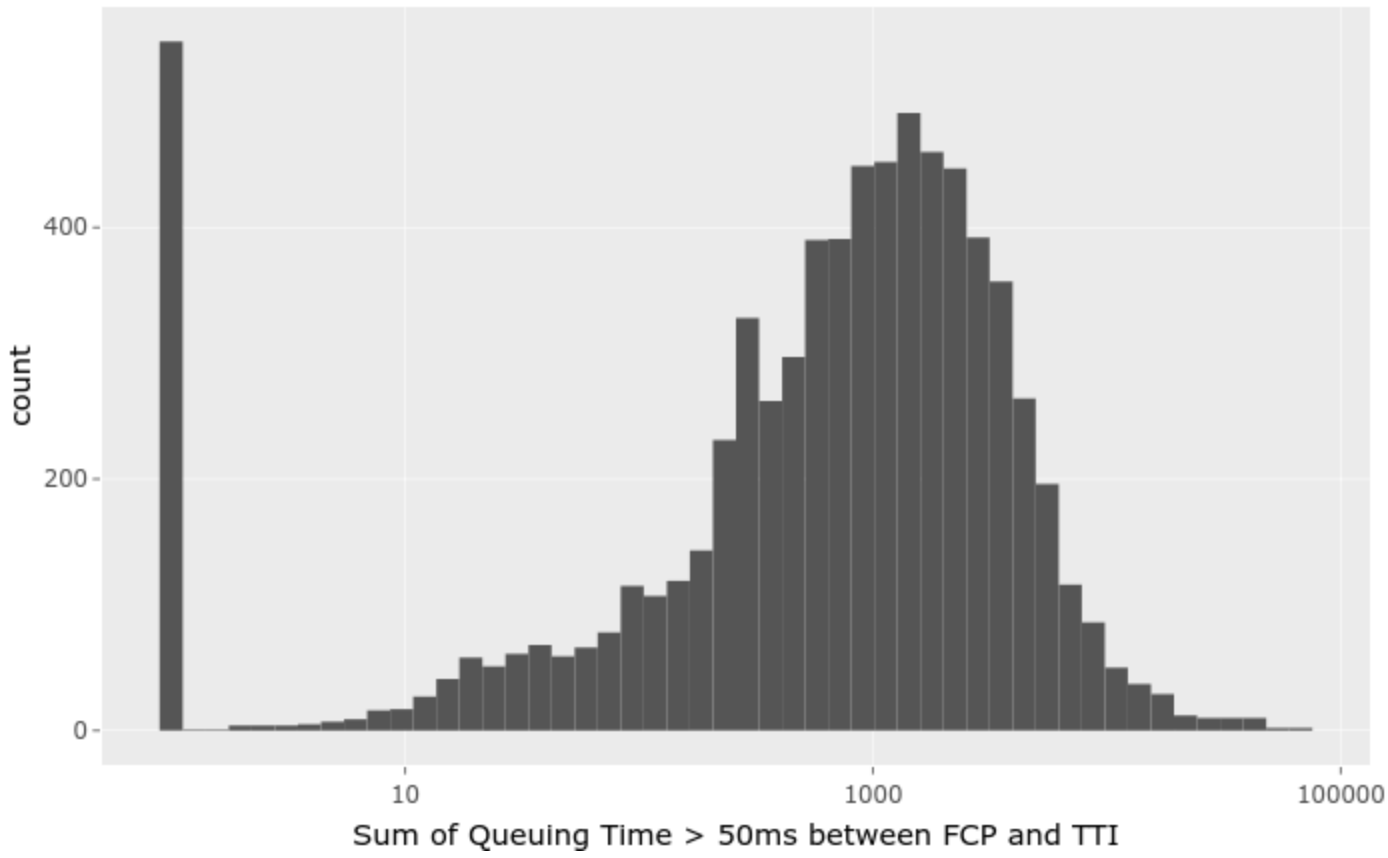
- Total number of rows: 6312
- **Number of zero values: 1572 (24.9%)**.

Mean and various percentiles:

mean(tbt) <dbl>	quantile(tbt, 0.1) <dbl>	quantile(tbt, 0.5) <dbl>	quantile(tbt, 0.9) <dbl>	quantile(tbt, 0.99) <dbl>
301.4675	0	89.5855	741.2776	2737.538

Mobile

Distribution (x-axis is log scaled. Added 1ms to TBT values so 0 values won't be undefined):



- Total number of rows: 7380
- Number of zero values: 1572 (7.3%)

Mean and various percentiles including the zero values:

mean(tbt) <dbl>	quantile(tbt, 0.1) <dbl>	quantile(tbt, 0.5) <dbl>	quantile(tbt, 0.9) <dbl>	quantile(tbt, 0.99) <dbl>
1848.143	20.2393	881.9075	4217.268	**15479.64**

See appendix for distributions excluding zero values:

Desktop distribution excluding zero values
Mobile distribution excluding zero values

Digging more into examples

What are all the zero values?

Any cases where TTI happens at FCP will have zero value, and there are quite a few of them. I expected these pages to be mostly 404 pages, but the vast majority of them are simple pages with very small amount of javascript. Examples:

- [Mobile] www.virtualbox.org . Single 119ms long task before FCP.
- [Desktop] https://www.wikiquote.org/ Super simple landing page. Two 173 and 76ms task, last one finishing at 540ms after navigation start. FCP happens at 566ms, and there are no long tasks after that, leading to a 566ms TTI.

Cases closer to the median

[Mobile] www.coursera.org

FCP: 6.2s, TTI: 38.3s. TBT: 1490ms. Long tasks: (Tasks between FCP and TTI highlighted orange):

start	end	dur
5860.661000000313	6053.320000000313	192.659
6059.542000000365	6202.9140000003645	143.372
7205.733000000007	7314.847000000009	109.114
23581.422000000253	24563.774000000252	982.352
28750.425999999978	29154.800999999978	404.375
29987.907000000123	30051.839000000124	63.932
30749.974000000395	30809.770000000397	59.796
30839.918000000063	30895.254000000066	55.336
30898.89900000021	30961.611000000208	62.712
34247.67299999995	34314.309999999954	66.637

35756.10499999998	35849.11499999998	93.01
38216.63500000024	38309.842000000244	93.21

The smaller long tasks are fine, but what's interesting is the 982ms long task, which is an EvaluateScript task, and by looking at their live site, it seems to be the webpack entry point.

[Desktop] www.shaw.ca

FCP: 9.3s, TTI: 12.5s. TBT: 214ms. Long tasks:

start▽	end▽	dur▽
7772.749000001699	8012.334000001698	239.585
8950.929999999702	9020.568999999701	69.639
9086.834000002593	9302.889000002593	216.055
9311.488000001758	9371.73300000176	60.245000000000005
9387.495999999344	9453.308999999344	65.813
9453.457000002265	9615.641000002264	162.184
9616.25	9672.4	56.15
10404.067000001669	10455.835000001669	51.768
10641.258999999613	10720.821999999613	79.563
11547.12200000137	11632.63700000137	85.515
12497.217000000179	12556.713000000178	59.496

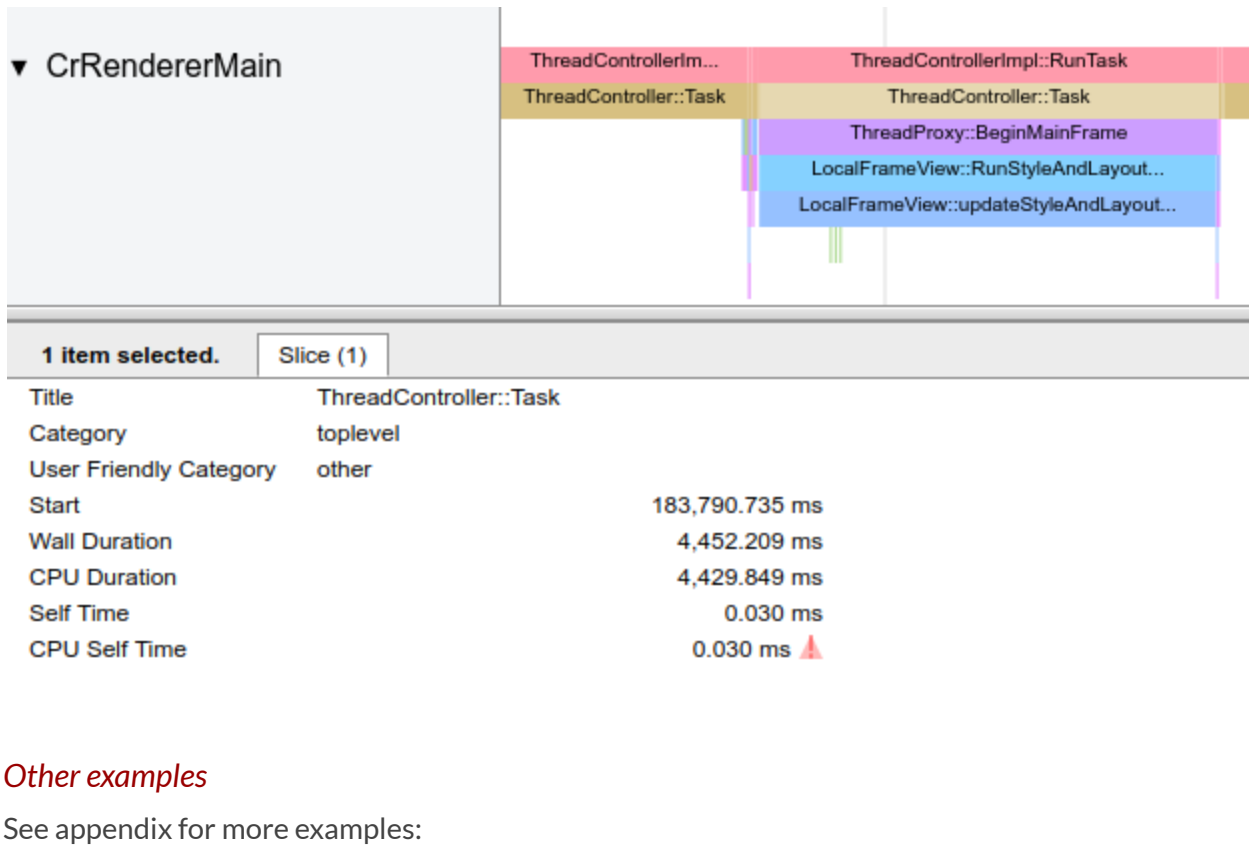
This is a very typical case on desktop, where there are some long tasks after FCP, but none of them too egregious.

Looking at very high values

- [Mobile] http://www.qatarairways.com TBT: 67s (yes, 67000ms)
  - ~58s synchronous XHR
  - The majority of the sites with very high TBT has synchronous XHR.
- [Mobile] www.builderall.com TBT: 63s
  - FCP: 5s, TTI: 134s

start▽	end▽	dur▽
3291.4869999999646	3404.269999999964	112.783
3766.829000000027	3835.6260000000257	68.797
3844.89300000004	4999.059000000041	1154.166
5049.9400000000605	6023.392000000062	973.452
6044.376999999979	7648.270999999979	1603.894
7670.6410000000615	8258.839000000062	588.198
34856.445999999996	38864.71399999999	4008.268
38876.898999999976	39116.67599999998	239.77700000000002
39116.687000000034	39183.176000000036	66.489
39873.55300000007	43919.53800000007	4045.985
43930.589000000036	47956.29400000004	4025.705
47961.67800000007	52000.03400000007	4038.356
52010.406000000075	56031.209000000075	4020.803
56036	60073.27099999999	4037.271
60718	64752.244999999995	4034.245
64753.042000000016	68795.20900000002	4042.167
68806.25199999998	72819.60099999998	4013.349
72829.34900000005	76841.36300000004	4012.014
76898.49500000011	80932.26500000012	4033.77
105082.679	109092.562	4009.8830000000003
122334.76099999994	126358.32099999994	4023.56
126365.48399999994	130387.98799999992	4022.504
130397.00399999996	134438.23099999997	4041.227
159540.45699999994	164219.14299999992	4678.686
164263.78000000003	164325.35400000002	61.574
164329.3870000001	168782.7620000001	4453.375
168818.206	178030.907	9212.701000000001
178031.189	178160.429	129.24
178601.89299999992	178706.94599999994	105.053
180055.80000000005	180164.39200000005	108.592
181231.7690000001	181298.3880000001	66.619

There are lots of 4s long tasks. They generally look like layout thrashing:



Other examples

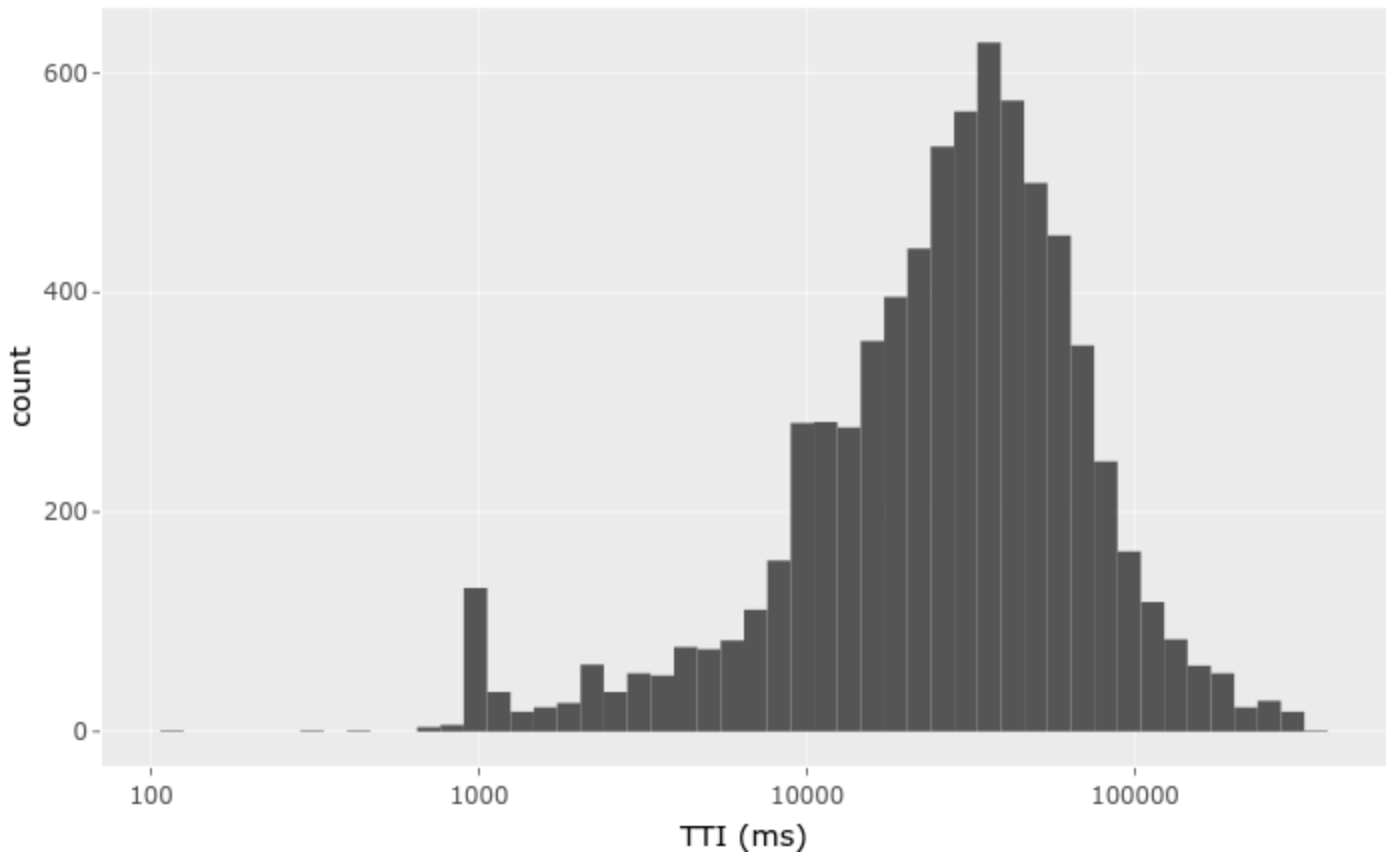
See appendix for more examples:

- High TTI and Low TBT case
  - uwaterloo.ca [trace] FCP: 23.7s, TTI: 239s, TBT: 1485ms.
- High and low TBT with Similar TTI
  - history.com
  - expedia.com
- Other high TBT cases

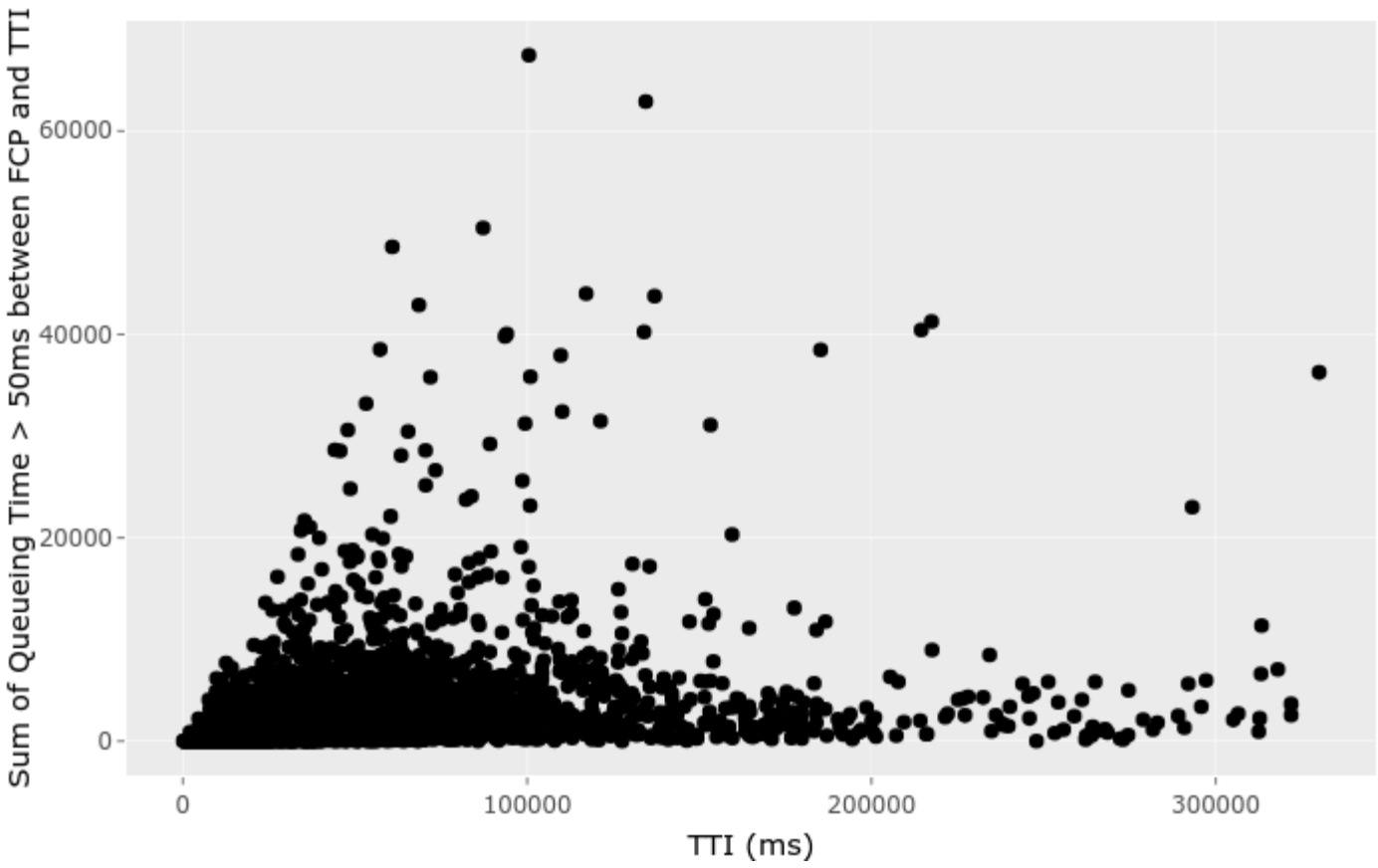
Relationship with TTI

This is the distribution of TTI over our collected data on mobile (7380 sites):

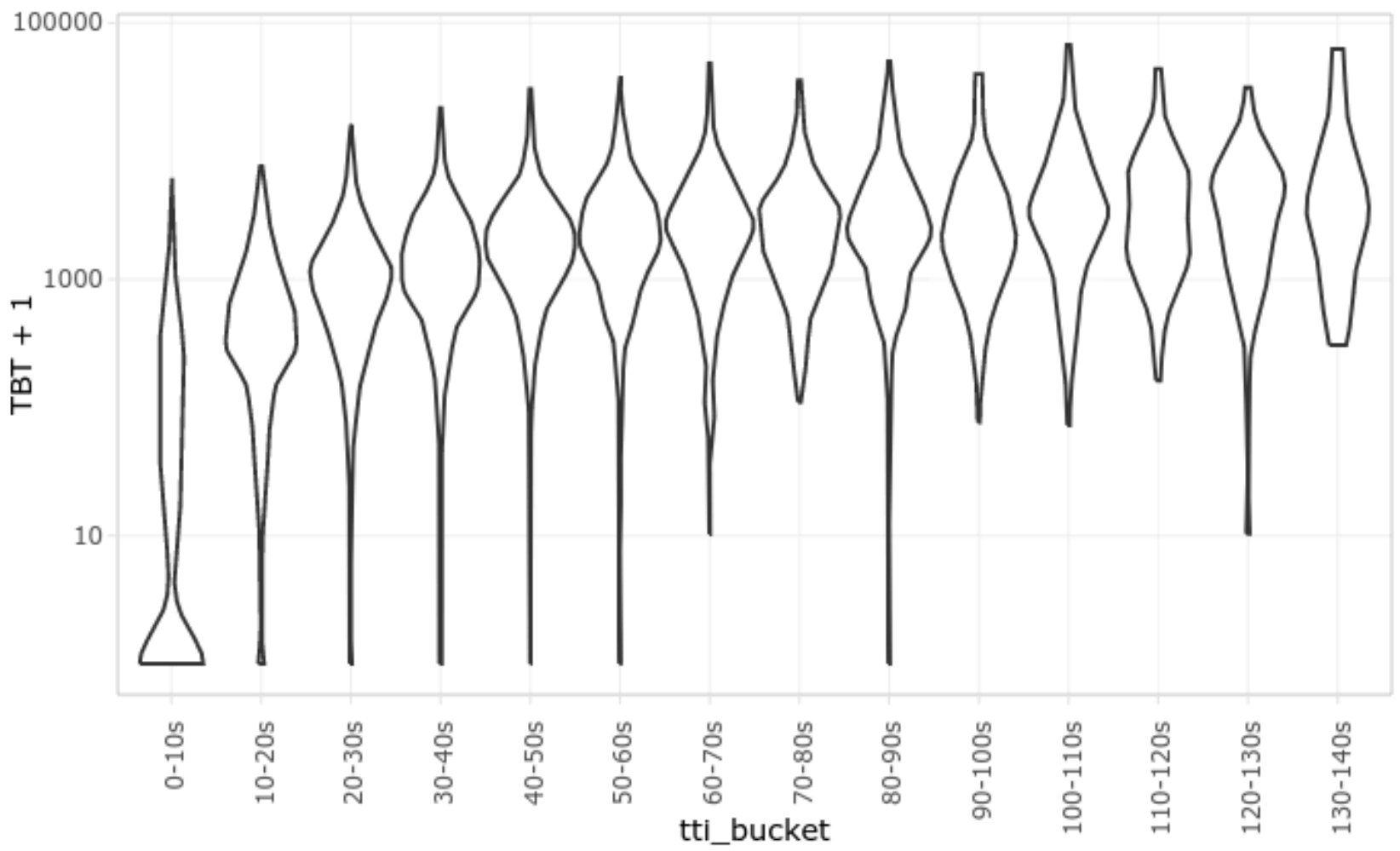
mean(TTI) <dbl>	quantile(TTI, 0.1) <dbl>	quantile(TTI, 0.5) <dbl>	quantile(TTI, 0.9) <dbl>	quantile(TTI, 0.99) <dbl>
38964.98	7137.749	29621.08	77532.3	194995.2



This is TTI plotted against TBT:



We observe that there is a subtle trend of higher TBT with higher TTI, but it's hard to gather much insight from it. To understand the relationship with TTI better, we grouped sites in TTI buckets of 10s (e.g. TTI 0-10s, 10-20s, etc.) and plotted the distribution of TBT in each bucket:



(Added +1ms to TBT again so we can represent 0 values on the y axis. TBT values in ms. Didn't show TTI buckets over 140s, as they were very sparse.)

We note:

- each TTI bucket has a TBT distribution of roughly around 500ms to around 6000ms.
- The TBT distribution shifts only slightly higher as we go up in TTI buckets.
- The vast majority of 0 TBT values happen with TTI < 10s.

Sanity checking by looking at the count, 10th percentile, and 90th percentile of each bucket:

tti_bucket <fctr>	count <int>	TBT10thPct <dbl>	TBT90thPct <dbl>
0-10s	1125	0.0000	325.2208
10-20s	1388	44.5480	1675.9364

20-30s	1213	151.7778	2779.1996
30-40s	1078	258.6729	3916.2592
40-50s	748	436.7985	4865.8566
50-60s	527	509.4666	6400.4752
60-70s	358	443.9925	6849.3637
70-80s	254	518.3366	6856.5258
80-90s	151	609.9250	8744.7740
90-100s	108	521.4631	7277.8121
100-110s	88	507.5072	12331.3819
110-120s	58	876.9134	9319.3006
120-130s	47	666.9134	8493.9368
130-140s	35	563.8622	17317.0400

Variability Analysis

TTI vs TBT noise characteristics

Metric instability has been a big complaint against TTI. Part of TTI's variability problems come from the fact that there are three hard thresholds:

- The 50ms threshold for being considered a long task
- The 2 concurrent network request threshold for network quiescence
- The 5 second threshold for sufficiently large quiet window.

Crossing any of these thresholds cause the metric to snap to the end of a different long task, routinely changing the metric by more than five seconds.

Since TBT relies on TTI for its bound of consideration, unfortunately TBT inherits some of these problems. However, the relative magnitude of change in TBT can vary substantially depending on circumstances. Consider this scenario: A page loads most of its code with 30s TTI and 1000ms TBT. There is a 100ms analytics script at 35.1s. On some of the runs, the analytics script happens 200ms second early, and TTI gets pushed to 35s.

- Change in TTI is 5s (30s -> 35s), or 16.6%.
- Change in TBT is 100ms (1000ms -> 1100ms), or 10%.

However, consider now that the analytics task was 200ms, and shifted 300ms forward, so new TTI is still 35s.

- Change in TTI is still 5s or16.6%.
- Change in TBT is now 200ms, or 20%.

We note that:

- Even though the magnitude of change in TBT was small compared to TTI (200ms vs 5s), since TBT is generally low for websites, the *relative* change can feel quite big.
- The lower the TBT of a site, the more relative effect TTI can have on its noise.

Variability study setup

We used the following setup to run a basic variability study:

- **Page set:** Alexa top 1000 minus some sites that could not be archived using WebPageReplay for various reasons.
- **Device:** Nexus 5X devices. Each site only ran on one device - there is no noise introduced because of variation in different devices, although the device temperature or OS level cache states may have varied from experiment to experiment.
- **Repeat:** Each site was loaded 25 times. 32 sites did not report data all 25 times for various reasons (browser crash, network hangs etc.). These were discarded, leaving us with **824 total sites considered in the final dataset**.
- **Network throttling:** We used Cluster Telemetry's 'Wifi' traffic shaping, which in practice is much closer to WebPageTest's 4G settings (30mbps down, 15 mbps up, 2ms rtt)

While we took some steps like using WebPageReplay to control for external variabilities, we should point out that there are still many sources of variability like device temperature. It would be interesting to see how much the metric does with lighthouse's lantern simulation.

Results

Standard deviation and relative standard deviations

We calculated standard deviation for each site. Summary stats for standard deviation per site:

```
summary(std_df)
     url                std_fcp             std_tti              std_tbt
 Length:856         Min.   :   5.342   Min.   :    9.11   Min.   :   0.0
 Class :character   1st Qu.:  29.997   1st Qu.:   74.08   1st Qu.:  25.7
 Mode  :character   Median :  48.646   Median :  192.38   Median :  56.0
                    Mean   :  94.020   Mean   : 1001.23   Mean   : 125.2
                    3rd Qu.:  86.217   3rd Qu.:  595.77   3rd Qu.: 131.2
                    Max.   :1776.900   Max.   :51378.39   Max.   :4508.4
```

Obviously TBT's standard deviation would be a lot lower than TTIs, but it's interesting to note that it's higher than FCP's standard deviation. Now to look at relative standard deviation (std / mean) (when mean TBT is 0, we consider the relative standard deviation to also be 0):

```
> summary(rstd_df)
     url                rstd_fcp          rstd_tti          rstd_tbt
 Length:856         Min.   :0.004533   Min.   :0.002369   Min.   :0.00938
 Class :character   1st Qu.:0.028179   1st Qu.:0.021291   1st Qu.:0.03650
 Mode  :character   Median :0.039260   Median :0.037167   Median :0.06421
                    Mean   :0.058526   Mean   :0.075602   Mean   :0.12887
                    3rd Qu.:0.061339   3rd Qu.:0.076332   3rd Qu.:0.11590
                    Max.   :0.786257   Max.   :1.010243   Max.   :2.76402
```

This shows TBT is much worse than TTI!

Width of range

If we look at the absolute diff of max and min values, this is the case for TBT:

Diff range	# of sites	Percentage of total	Percentile
0-100	152	18.72%	18.72%
100-200	190	23.40%	42.12%
200-300	148	18.23%	60.34%
300-400	75	9.24%	69.58%
400-500	50	6.16%	75.74%
500-600	37	4.56%	80.30%
600-700	30	3.69%	83.99%
700-800	25	3.08%	87.07%
800-900	10	1.23%	88.30%

+ a fat tail that is not shown.

Notably, more than 42.12% of the sites had a diff between max and min values of less than 200ms.

Filtering to larger values

If we filter to cases where mean TBT (average of 25 runs) was higher than 200ms, we see on average, mean TBT standard deviation is lower than TTI's.

```
rstd_df %>% filter(mean_tbt > 200) %>% summary()
     url                rstd_fcp           rstd_tti           rstd_tbt
 Length:763         Min.   :0.004533   Min.   :0.002369   Min.   :0.009377
 Class :character   1st Qu.:0.027403   1st Qu.:0.020457   1st Qu.:0.032022
 Mode  :character   Median :0.038975   Median :0.036863   Median :0.049009
                    Mean   :0.059874   Mean   :0.076909   Mean   :0.071221
                    3rd Qu.:0.063149   3rd Qu.:0.080748   3rd Qu.:0.081550
                    Max.   :0.786257   Max.   :1.010243   Max.   :1.544617
                    NA's   :2          NA's   :2          NA's   :2
```

On the other hand, filtering to cases where TBT < 200ms, we see that it's highly noisy:

```
rstd_df %>% filter(mean_tbt < 200) %>% select(url, rstd_fcp, rstd_tti, rstd_tbt) %>% summary()
     url                rstd_fcp           rstd_tti           rstd_tbt
 Length:92          Min.   :0.009936   Min.   :0.004915   Min.   :0.0000
 Class :character   1st Qu.:0.033017   1st Qu.:0.031266   1st Qu.:0.1211
 Mode  :character   Median :0.042098   Median :0.038925   Median :0.2341
                    Mean   :0.047383   Mean   :0.065268   Mean   :0.4688
                    3rd Qu.:0.053531   3rd Qu.:0.060198   3rd Qu.:0.6558
                    Max.   :0.218366   Max.   :0.801488   Max.   :4.1332
```

Similarly, if we filter to more than 10s mean TTI, TBT has lower standard deviation.

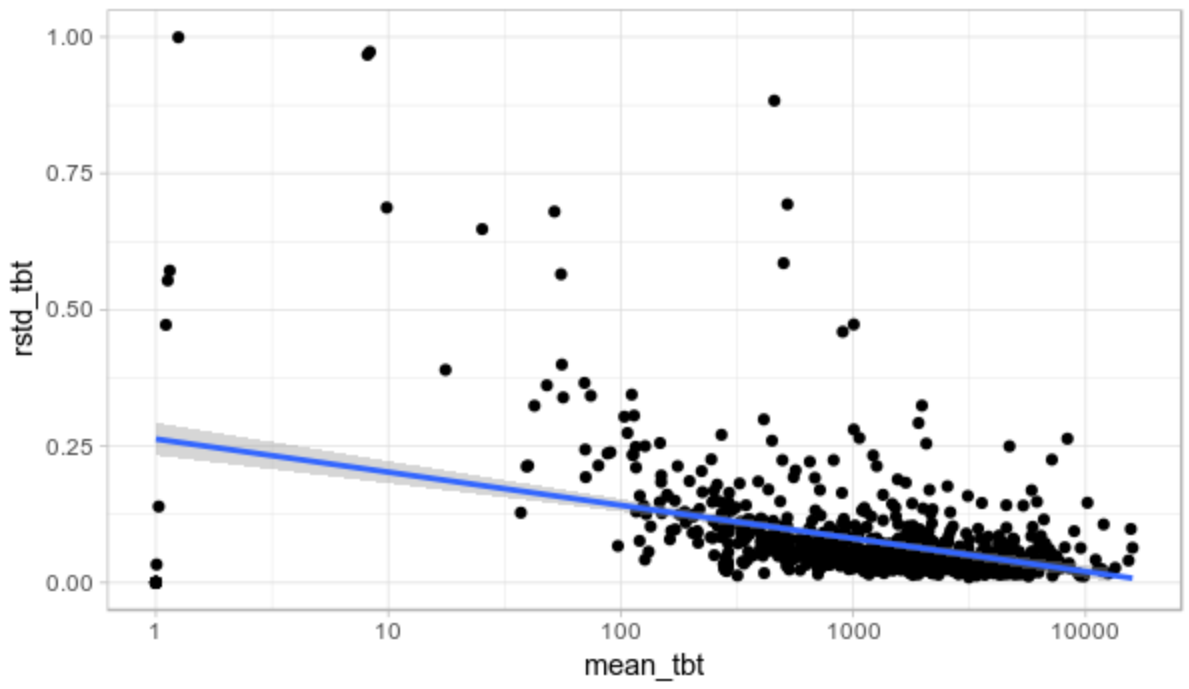
```
rstd_df %>% filter(mean_tti > 10000) %>% summary()
     url                rstd_fcp           rstd_tti           rstd_tbt
 Length:229         Min.   :0.004533   Min.   :0.002369   Min.   :0.01177
 Class :character   1st Qu.:0.023524   1st Qu.:0.023607   1st Qu.:0.02876
 Mode  :character   Median :0.035884   Median :0.045853   Median :0.04454
                    Mean   :0.066137   Mean   :0.098282   Mean   :0.06542
                    3rd Qu.:0.069137   3rd Qu.:0.103231   3rd Qu.:0.07248
                    Max.   :0.786257   Max.   :1.010243   Max.   :0.73223
                    NA's   :2          NA's   :2          NA's   :2
```

Filtering to less than 10s mean TTI similarly show the opposite story:

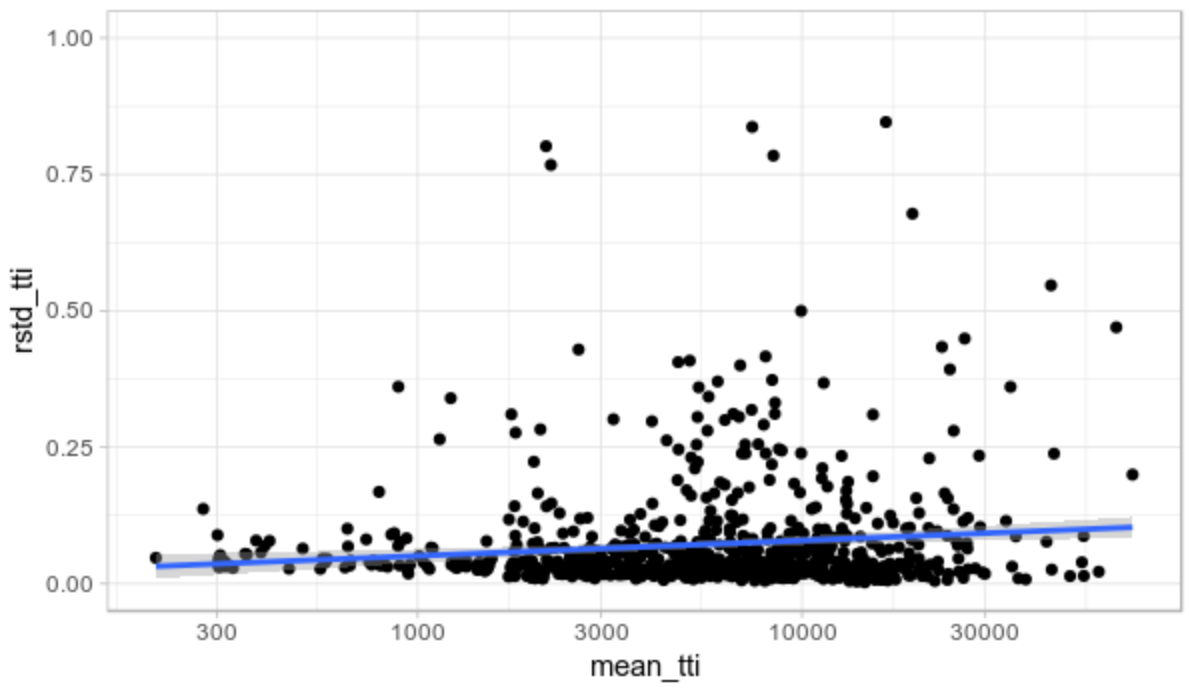
```
rstd_df %>% filter(mean_tti < 10000) %>% select(url, rstd_fcp, rstd_tti, rstd_tbt) %>% summary()
     url                rstd_fcp           rstd_tti           rstd_tbt
 Length:608         Min.   :0.005973   Min.   :0.004915   Min.   :0.00000
 Class :character   1st Qu.:0.030230   1st Qu.:0.020902   1st Qu.:0.03341
 Mode  :character   Median :0.040100   Median :0.034890   Median :0.05761
                    Mean   :0.056053   Mean   :0.067434   Mean   :0.13300
                    3rd Qu.:0.058864   3rd Qu.:0.065780   3rd Qu.:0.10312
                    Max.   :0.511886   Max.   :0.836808   Max.   :4.13323
```

Variability trend with increase in value

More generally, we can look at how variability changes as the metric value increases. This chart plots the relative standard deviation of TBT against mean TBT, which clearly shows how TBT becomes less noisy as the value increases:



Compare this with TTI's, which shows that TTI becomes slightly more noisy as the value increases:



Where does the noise come from?

The lighthouse team has done some excellent analysis of metric variability, and these are the sources of variability they list there:

- Local network variability
- Tier-1 network variability
- Web server variability
- Client hardware variability
- Client resource contention
- Browser nondeterminism
- Page nondeterminism

All of these factors will certainly be at play here in this analysis. We have not done an analysis of how much each of these factors contribute to the variability of TBT, but we can do that in the future if that appears necessary.

Higher powers of blocking time

We also considered higher powers of blocking time. For the sake of precision, we define TBT1p5 and TBT2p0 as follows:

- Similar to the original definition, define a blocking time interval to be the part of a main thread task where queuing time is > 50ms.
- TBT1p5 :=
    - Clip all blocking time intervals to be between FCP and TTI.
    - Raise each interval duration to the 1.5 power.
    - Take the sum.
- TBT2p0 := Similar, but raise each blocking time interval duration to 2.0 power.

+1 for TBT1p5/2p0

- Raising each time interval to a power penalizes longer tasks more harshly. For example, one 800ms task is likely a worse user experience than five 200ms tasks (TBT 150 * 5 = 750ms).
    - 800ms task has
        - TBT 750ms
        - TBT1p5 = 20539
        - TBT2p0 = 562500
    - Five 200ms tasks in total has:
        - TBT 150ms * 5 = 750ms
        - TBT1p5 = 9186
        - TBT2p0 = 112500
    - Higher power thus incentivizes developers to break down their longest long tasks first.

+1 for TBT

- TBT1p5/2p0 makes the narrative slightly more complicated. Total Blocking Time values are very easy to interpret; TBT of 750ms means there was total 750ms between FCP and TTI where task length exceeded by 50ms. On the other hand, TBT1p5 value of 9180 (with unit ms^1.5) is much harder to interpret directly.
- TBT1p5/2p0 suffers from worse variability problems. Here is a summary of the relative standard deviations:

```
> rstd_df %>% select(rstd_tbt, rstd_tbt1p5, rstd_tbt2p0, rstd_tti) %>% summary()
    rstd_tbt         rstd_tbt1p5       rstd_tbt2p0         rstd_tti
 Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.002369
 1st Qu.:0.03217   1st Qu.:0.04266   1st Qu.:0.05498   1st Qu.:0.020928
 Median :0.05185   Median :0.08008   Median :0.11337   Median :0.035943
 Mean   :0.11407   Mean   :0.16258   Mean   :0.21949   Mean   :0.070561
 3rd Qu.:0.09448   3rd Qu.:0.15405   3rd Qu.:0.22377   3rd Qu.:0.070884
 Max.   :4.13323   Max.   :3.79023   Max.   :4.63630   Max.   :0.845795
```

As in the case with TBT, the situation improves when we filter for TTI > 10s, but it's still markedly worse than TBT.

```
rstd_df %>% filter(mean_tti > 10e3) %>% select(rstd_tbt, rstd_tbt1p5, rstd_tbt2p0, rstd_tti)
%>% summary()
    rstd_tbt         rstd_tbt1p5       rstd_tbt2p0         rstd_tti
 Min.   :0.01177   Min.   :0.01436   Min.   :0.01491   Min.   :0.002369
 1st Qu.:0.02765   1st Qu.:0.04622   1st Qu.:0.05822   1st Qu.:0.021758
 Median :0.04177   Median :0.06990   Median :0.10326   Median :0.037567
 Mean   :0.05765   Mean   :0.09311   Mean   :0.13425   Mean   :0.079883
 3rd Qu.:0.06658   3rd Qu.:0.10428   3rd Qu.:0.15349   3rd Qu.:0.087911
 Max.   :0.32441   Max.   :0.57684   Max.   :1.16290   Max.   :0.845795
```

+0 for both

- By manually eyeballing a small set of sites, we found that TBT1p5/2p0 did slightly better than TBT at agreeing with our human evaluation of user experience on slower network, while did worse under faster networks. We can call this a tie.

Verdict

We believe we have stronger justification for TBT over TBT1p5/2p0.

Comparing with sum of long tasks

Total Blocking Time creates better incentives than Sum of Long Task Durations. Consider this example:

- one 500ms task:
    - Total Blocking Time = 450ms
    - Sum of long tasks = 500ms
- five 100ms tasks:
    - Total Blocking Time = 250ms
    - Sum of long tasks = 500ms

Five 100ms task is better than one long 500ms task, but sum of long task durations does not capture this difference.

Conclusion

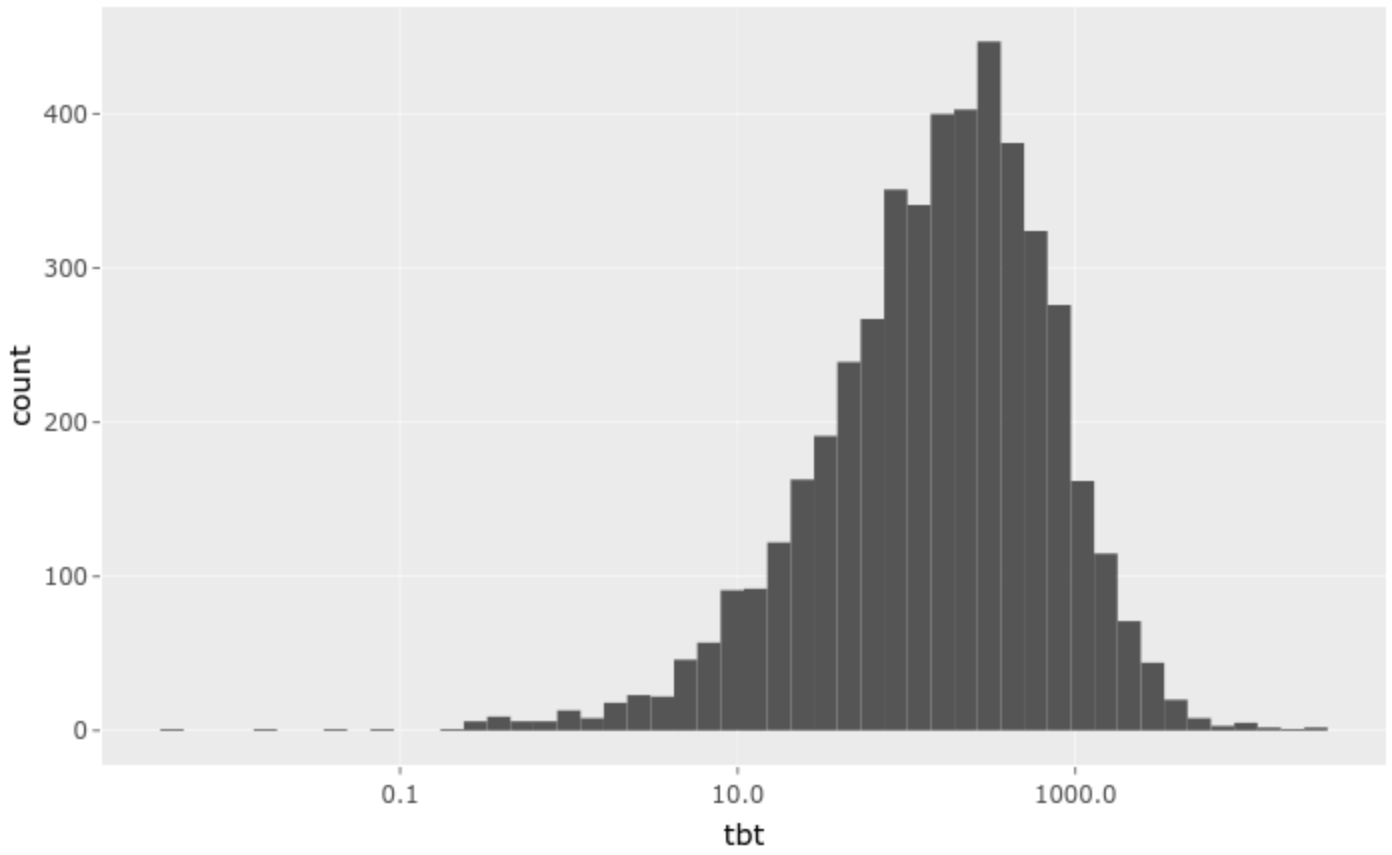
To recap the key qualitites of TBT:

- TBT is useful at differentiating two sites with similar TTI even if neither have perfect UX.
    - The wide distribution of values at each TTI bucket and some examples we looked at proves that TBT is indeed valuable in differentiating user experience when TTI cannot.
    - Also, unlike TTI where most sites on the internet do poorly on, many sites have very good TBT. It should be easier understand the value of improving TBT.
- TBT can be effective at highlighting cases where UX is bad enough to hurt user engagement.
    - A 4.2s TBT (value at the 90th percentile on mobile) sends a very clear message that there is serious room for improvement, with clear directions for how to make things better and lots of examples of sites that *are* doing better.
- A small improvement to the site results in a small improvement of TBT, unlike TTI where small improvements are often not reflected in metric value.
- TBT has reasonably low variability so it's easy to detect performance regressions and improvements.
    - 42% of the time, the metric variation is lower than 200ms.
    - The metric is lower relative standard deviation than TTI when filtered to high values (either TBT > 200ms, or TTI > 10s)
    - When metric value is low (<200ms), it has higher relative standard deviation, but 200ms can be considered a reasonable TBT budget for a site, where further improvement has low marginal benefit.

Appendix

Desktop distribution excluding zero values

Excluding zero values, so we can look at the non-zero distribution better:

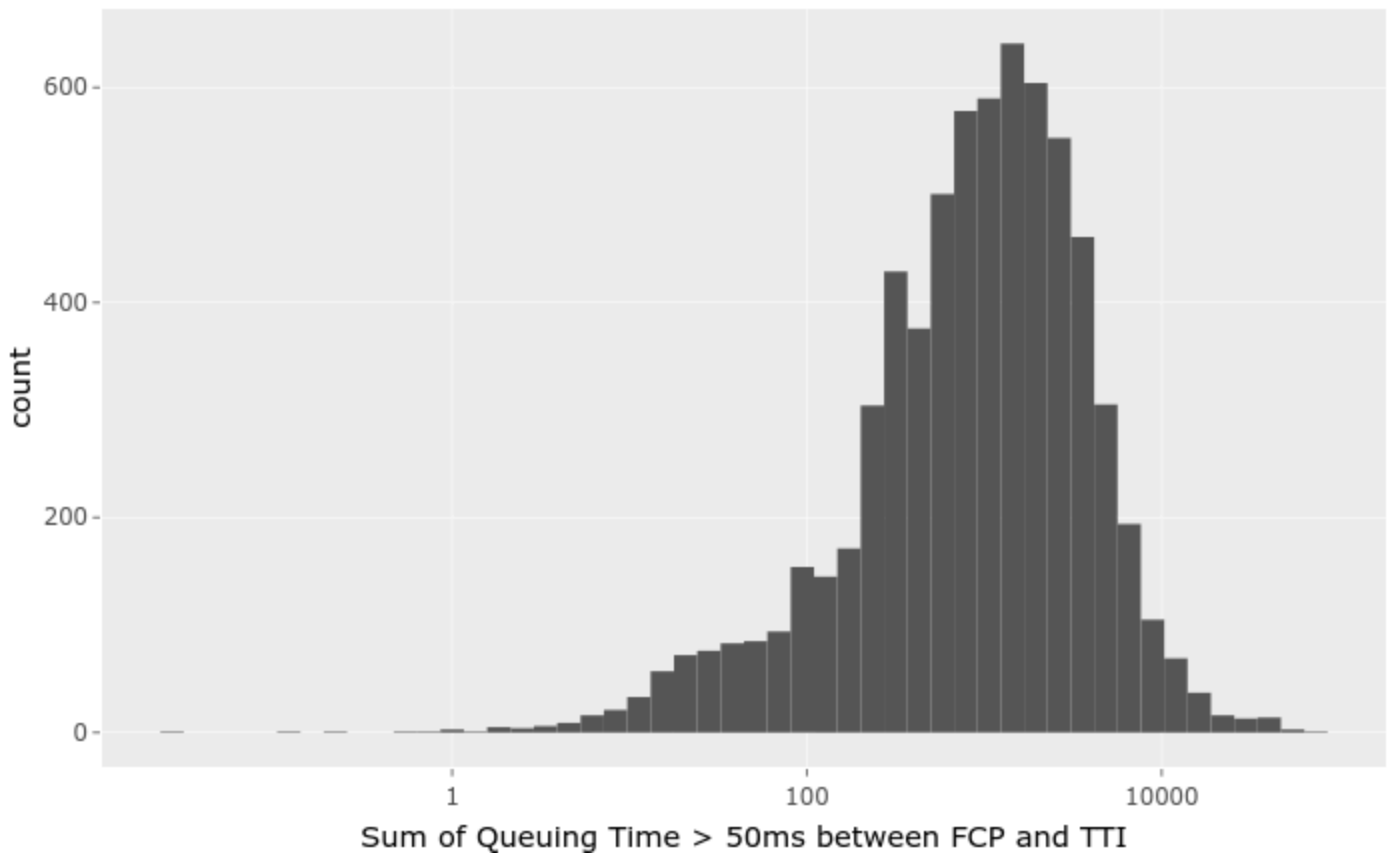


Mean and various percentiles excluding zero values:

mean(tbt) <dbl>	quantile(tbt, 0.1) <dbl>	quantile(tbt, 0.5) <dbl>	quantile(tbt, 0.9) <dbl>	quantile(tbt, 0.99) <dbl>
401.4479	18.3306	176.808	873.1359	3092.247

Mobile distribution excluding zero values

Excluding zero values, so we can look at the non-zero distribution better:



Mean and various percentiles excluding zero values:

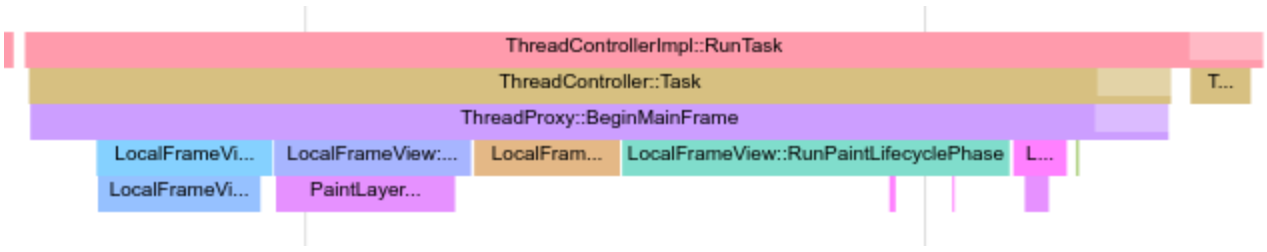
mean(tbt) <dbl>	quantile(tbt, 0.1) <dbl>	quantile(tbt, 0.5) <dbl>	quantile(tbt, 0.9) <dbl>	quantile(tbt, 0.99) <dbl>
1995.8	101.1546	1017.357	4442.485	**16095.91**

High TTI and Low TBT case

uwaterloo.ca FCP: 23.7s, TTI: 239s, TBT: 1485ms.

start▽	end▽	dur▽
11420.5270000007	11488.102000000701	67.575
14714.76900000032	14772.24700000032	57.478
23439.571000000462	23716.60800000046	277.037
24954.537000000477	25193.418000000478	238.881
26801.78200000059	26908.06500000059	106.283
29524.299999999814	29580.080999999816	55.781
32068.95100000035	32129.721000000347	60.77
39111.08100000024	39169.997000000236	58.916000000000004
46142.830000000075	46198.69100000007	55.861000000000004
74244.45999999996	74297.13699999996	52.677
81269.97900000028	81324.33900000028	54.36
109365.3459999999	109418.3269999999	52.981
116389.3490000004	116439.9140000004	50.565
186629.25	186682.642	53.392
214727.89099999983	214782.41299999983	54.522
220566.12200000044	220619.00300000043	52.881
220619.01099999994	220698.46799999994	79.45700000000001
220985.97000000067	221124.16800000067	138.198
221756.08100000024	221808.50100000025	52.42
231000.3320000004	231172.1540000004	171.822
231483.43000000063	231653.15200000064	169.722
231912.93200000003	232089.52000000002	176.588
232357.19299999997	232528.98099999997	171.788
232979.29800000042	233153.58300000042	174.285
233628.82600000035	233807.57600000035	178.75
234286.23700000066	234608.21400000068	321.97700000000003
236676.38800000027	236777.62900000028	101.241
239694.6570000006	239746.11700000058	51.46
249847.57800000068	249901.30700000067	53.729
256869.89400000032	256924.29000000033	54.396
277938.4950000001	277992.3370000001	53.842
284974.2960000001	285029.8030000001	55.507
291999.3830000004	292057.18300000037	57.800000000000004
299026.7110000001	299077.36100000015	50.65
306056.284	306106.948	50.664
320115.0650000004	320171.4730000004	56.408

There are a ton of just above 50ms tasks, and they all look like this:



This is likely a recurring animation.

The 321ms task at 234s (5 seconds before TTI at 239s) is a style/layout task that happens after a fond download finishes.

High and low TBT with Similar TTI

One example from the desktop data:

history.com

page_name	traceUrl	TTI	# of navigations	FCP	DCL	TBT
http://www.history.com	trace	8962.492	1	1960.708	2066.451	335.923

Long tasks:

start ▽	end ▽	dur ▽
1573.341000000015	1671.7290000000148	98.388
1672.680999999866	1927.6979999998657	255.017
2002.5679999999702	2072.88399999997	70.316
7792.790000000037	7858.852000000037	66.062
8804.78799999971	8872.025999999709	67.238
8883.589999999851	8962.491999999851	78.902

expedia.com

page_name	traceUrl	TTI	# of navigations	FCP	DCL	TBT
http://www.expedia.com	trace	9509.443	1	639.661	1475.036	2758.311

start ▽	end ▽	dur ▽
1271.8769999966025	1354.2189999966022	82.342
1358.3669999986887	1415.4699999986888	57.103
1618.777000002563	1720.2940000025628	101.517
1783.5239999964833	1865.7509999964832	82.227
1866.3819999992847	1995.0869999992847	128.705
2058.1960000023246	2108.421000002325	50.225
2143.1689999997616	2235.4279999997616	92.259
2259.8430000022054	2419.2000000022053	159.357
2442.347999997437	2499.1719999974366	56.824
2500.4319999963045	3209.223999996305	708.792
3247.3439999967813	3313.252999996781	65.909
3320.767999999225	3578.1799999992254	257.412
3701.9659999981523	3839.335999998152	137.37
3919.3069999963045	4692.089999996304	772.783
4830.129000000656	4992.710000000656	162.58100000000002
5101.777999997139	5216.993999997139	115.21600000000001
5928.381999999285	6137.8809999992845	209.499
6218.728000000119	6293.599000000118	74.871
6312.938000001013	6405.248000001013	92.31
6464.146999999881	6524.64399999988	60.497
6554.501000002027	6665.668000002026	111.167
6952.0839999988675	7075.549999998868	123.46600000000001
9303.56400000304	9509.44300000304	205.879

]

This is a clear case where two sites have similar TTI (8.9 and 9.5s) but very different TBT (335ms vs 2758ms.) TBT is providing meaningful differentiation here.

Other high TBT cases

- [Desktop] www.geforce.com . TBT: 20.8s.
    - 20s synchronous XHR load. This happens through document.write
- [Desktop] http://gumtree.com.au TBT: 5.5s.
    - FCP: 2.3s, TTI: 13s. All long tasks are after FCP:

[

start▽	end▽	dur▽
2408.683999999892	2853.2669999998925	444.583
3712.4849999998696	3810.8679999998694	98.383
4869.195999999996	4919.493999999997	50.298
4976.638000000035	5048.604000000036	71.96600000000001
5421.407999999821	5511.586999999821	90.179
5512.900999999838	5570.750999999838	57.85
5570.753000000026	5700.968000000026	130.215
5700.970999999903	5803.591999999902	102.621
5890.541999999899	5942.640999999898	52.099000000000004
6416.953999999911	6909.339999999909	492.386
7758.284999999916	7848.608999999917	90.324
7940.7319999998435	7991.995999999843	51.264
8164.649999999907	12518.479999999909	4353.83
12703.767999999924	12796.684999999925	92.917
13053.035999999847	13128.456999999846	75.421

The 4s long task is a timer task, and it's unclear what it is. It shows 4.3 seconds of wall time and only 47ms of CPU time, so clearly the task was blocked on something.