# 未定时代的可信浏览器

这篇论文被包含在第25届高级计算机系统协会安全论坛的论文集中。

## 摘要 搜索

一个源的JavaScript在浏览器中可以使用定时信道来学习一个用户和其他源交互的敏感信息，这一做法违背了浏览器的分隔分布的保证。浏览器供应商已试图通过尝试重写敏感代码使它运行在一个恒定的时间和减少基准时钟的分辨率来关闭定时信道。

我们认为这些特别的努力不太可能成功。我们展示了将退化时钟的有效分辨率提高2个数量级的技术，并且还展示和评估了多个新的隐式时钟：通过这个技术JavaScript可以计时事件而无需咨询显式时钟。

通过降低所有时钟和减少所有定时信道的带宽，我们展示了在可信操作系统文献中的“模糊时间”的概念是如何被改编来构建可信的浏览器。我们描述了下一代浏览器的设计，称为Fermata，其中所有的时间源都是完全中介的。作为可行性的证明，我们展示了Fuzzyfox，它是Firefox浏览器的一个分支，在当今浏览器架构的约束下，它实现了许多Fermata原则。我们展示了Fuzzyfox在今天的部署中实现了对隐私敏感用户的足够的兼容性和性能。

总而言之：

* 我们展示了一个攻击者如何在不查询显式时钟的情况下在web浏览器中度量持续时间。
* 我们展示了“模糊时间”概念如何应用于web浏览器以减轻所有时钟。
* 我们展示了一个原型，它论证了其中一些概念的影响。

## 介绍

给通过DOM访问系统资源的代码授权，Web浏览器从用户访问的站点以及像广告网络这样的第三方站点下载并运行JavaScript代码。防止不受信任的代码控制用户的系统是限制问题。此外，浏览器必须确保在一个源中运行的代码不能学习与另一个源的用户交互的敏感信息。这就是分隔的问题。

限制的失败会导致分隔的失败。但JavaScript也可以学习敏感信息，而不必逃避它的沙箱，特别是利用侧时间信道。当攻击者可以比较一个调制的时钟时，一个计时信道是可能的——在这个时钟上，快点或慢点到取决于一个秘密-一个基准时钟-这个时钟运转的节拍到达一个一致的速度。例如，浏览器允许web页面通过CSS将SVG转换应用到页面元素，包括跨源帧。保罗·斯通(paul Stone)指出，在feMorphology滤镜上的快速路径优化创建了一个定时攻击，允许攻击者窃取像素或嗅探用户的浏览历史，使用Window.requestAnimationFrame()作为一个调节时钟。最近，Oren等人发现，在高分辨率参考时钟的出现上，就像性能一样。现在，攻击者可以使用JavaScript TypedArrays来测量lastlevel处理器缓存的瞬时负载。

浏览器供应商意识到时间信道所构成的危险，并努力解决它。

首先，他们试图通过编写任何代码，这个代码可以控制在固定时间运行的秘密值，从而消除调制时钟。例如，在100个信息的Bugzilla线程中，Mozilla

的工程师们决定通过使用常量时间的比较来重写feMorphology滤镜实现来解决斯通的像素窃取工作。

其次，他们试图降低JavaScript代码可用的参考时钟的分辨率。2015年

5月，Tor浏览器开发人员降低了性能的分辨率。现在，高分辨率计时器到100毫秒作为一种反指纹测量。在2015年晚期,一些主要的浏览器(Chrome,Firefox)应用类似的补丁(见图1),减少计时器分辨率到5µ来s击败奥伦等人的缓存时间攻击。

这些努力不太可能成功，因为他们严重低估了问题的复杂性。

首先，消除每一个潜在的调制时钟需要对整个代码库进行审计，这是一

个雄心勃勃的任务，即使是一个更小、更简单的系统，如微内核。事实上，

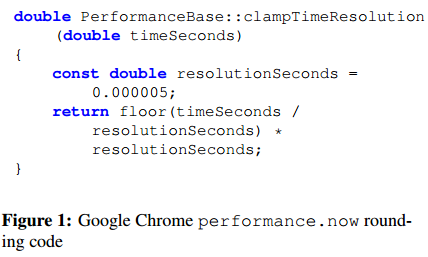
Mozilla修复feMorphology时并没有考虑到浮点指令执行的速度的快慢取决于他

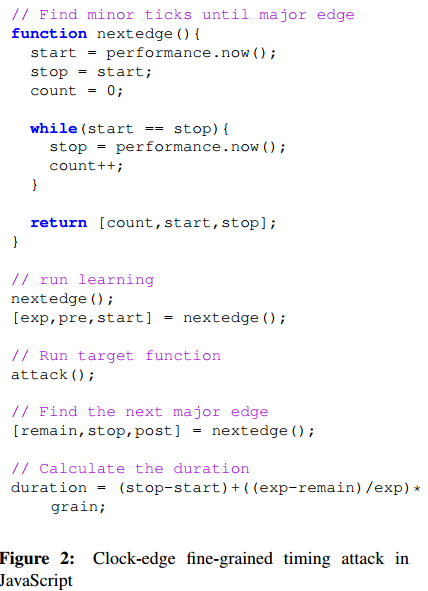
们的输入，即使在所谓的“常量时间”代码中，也允许像素窃取攻击。

其次，除了天真地查询performance.now，JavaScript代码可以通过许多方式合成一个参考时钟。在本文中，我们展示了clock - edge检测允许JavaScript将一个退化的performance.now时钟的有效分辨率提高两个数量级的时钟。我们还展示并评估了多个新的隐式时钟:JavaScript可以在不咨询像performance.now这样的显式时钟的情况下进行时间事件。例如，HTML5中的视频<video>标签在单独的线程中被解码。JavaScript可以播放一个简单的视频，它可以通过每个帧改变颜色，并通过绘制画布来检查当前帧。这立即给出了一个包含分辨率为60hz的隐式时钟，并且通过使用我们的技术可以提高分辨率。

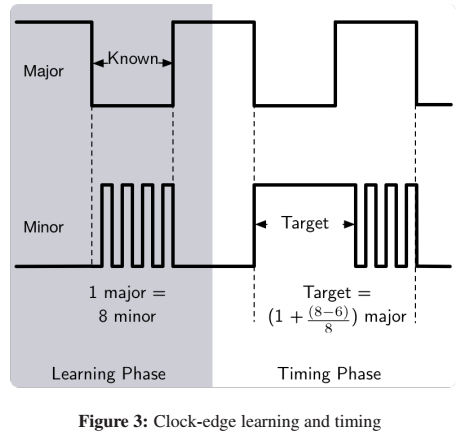
简而言之，时间信道对浏览器的分隔构成了严重的危险；浏览器供应商意识到了这个问题，并试图通过消除或降低攻击者依赖的时钟来解决这个问题，但他们的特别努力不太可能成功。本文的研究结果表明，现代浏览器中的计时信道问题与受信任操作系统的时序信道问题类似，并且可信系统文献中的思想可以为有效的浏览器防御提供信息。事实上，我们对计时频道的描述以及参考时钟和调制时钟的比较是有Wray引起的，而我们的模糊缓解策略技术直接受到了胡的启发——这两篇论文都是由VAX VMM安全内核项目引起的，目标为A1等级。

在本文中，我们展示了由Hu引发“模糊时间”概念可使用于构建值得信赖的浏览器。模糊时间会降低所有时钟，无论是隐式的还是显式的，它降低了所有时间信道的带宽。我们描述了在一个可信的浏览器中所需要的属性，所有的时间源都是完全中介的。今天的浏览器对JavaScript引擎和DOM进行了严格的耦合，需要进行广泛的重新设计，以完全协调所有的时间源。作为一个在可行性方面的证据，我们介绍Fuzzyfox，它是Firefox浏览器的一个分支，它在当今浏览器架构的约束下，使用模糊时间来降低时间源。Fuzzyfox演示了一个有原则的时钟fuzzing方案，它可以应用于主流浏览器和Tor浏览器，使用相同的机制。我们对Fuzzyfox的性能开销和兼容性进行了评估，表明它的所有理念都适合在Tor浏览器等产品中进行部署，而更温和的版本适用于Firefox。





## 时钟边缘攻击



Web浏览器供应商试图通过将可用于JavaScript的显式时钟四舍五入到一些颗粒g，来缓解时间边信道攻击。例如,Google Chrome和Firefox已经实施了一个5µs颗粒。图1显示了在谷歌Chrome中的用于四舍五入performance.now的c++代码。Tor浏览器提供了一个不同的隐私和性能权衡，并实现了一个侵略性的100ms颗粒。

不幸的是，舍入并不能保证攻击者不能准确地测量小于g的时序差异。我们提出了在JavaScript时钟上下文中提高时间度量粒度的时钟边缘技术。实验结果表明，这种技术可以提高至少两个大粒度时钟的分辨率。这种技术可以推广到任何一对时钟:一个主时钟，它有一个已知的大周期，一个小时钟，它有一个短暂的未知周期。主时钟是用来建立小钟的周期的，并且它们可以一起计时，比单独的时间更精确。

考虑一个页面的情况，希望用一个比一些已知的 performance.now 更小的粒度，来定时某些JavaScript函数attack()。在这种情况下，主要的时钟是退化的performance.now，并且我们用一个紧密增长的循环来作为小时钟。图2和图3展示了页面如何执行该技术以及流程的可视化表示。

该页面首先了解主要时钟节拍Cl1和Cl2之间的平均循环迭代次数(Lexp)。在学习之后，页面将运行，直到检测到一个主要时钟边缘(Cstart)，然后执行attack()。当attack ()在主时钟时间Cstop返回时，该页面将运行小时钟(用于Lremain节拍)直到下一个主要时钟边缘(Cpost)被检测到。然后页面用(Cstop−Cstart)+ g∗(Lexp−Lremain)/(Lexp)计算attack ()的持续时间。在g不是保持不变的情况下,我们用公式 (Cpost−Cstop)/(Cl2−Cl1)测量Lexp和设置g = Cpost−Cstop。

由于(Lexp−Lremain)/(Lexp)代表g的一个小数部分, 测量时间似乎都可以获得像测量g / Lexp一样细粒。因此，只要攻击者能够获得一个合适的小时钟，一个主时钟通过舍入降级到g就不能保证攻击者不能测量一个小于g的粒度。

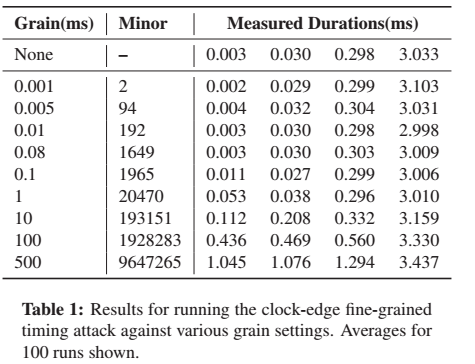


表1显示了将clock - edge技术应用于在不同的颗粒的4个不同的目标的降级performance.now的主时钟上。图2中的代码是测试代码的缩写版本。每个持续时间列表示在attack()函数中不同的迭代次数，这是一个空的for循环。The minor ticks列表示每一个主要的节拍都探测到的学习阶段的迭代次数。“None”行表示没有启用四舍五入的attack运行时间，其他行显示使用clock - edge技术在不同的谷物设置中度量的持续时间。通过修改后的Firefox的测量，可以通过JavaScript来设置任意的颗粒。

如表1所示，clock - edge攻击恢复持续时间，明显小于颗粒设置。值得注意的是，颗粒在毫秒和更高的范围仍然允许分化的持续只有数万µs事件!

如果攻击者试图区分每个持续时间小于1微秒的事件，那么仅仅只舍入可用的显式时钟就会有显著的影响，在这个级别上，clock - edge攻击通常不会对舍入时钟提供额外的分辨率。

## 在没有显式时钟的浏览器中测量时间

在本节中，我们论证了一个攻击者可以使用的不同的方法来测量在JavaScript中事件的持续时间。希望对web浏览器进行定时攻击的攻击者并不局限于使用performance.now来计时测量，本节将提供一些可用的替代方法。支持这些测量的浏览器特性是隐式时钟。取决于目标和时钟如何与JavaScript运行时进行交互，我们把他们说成是离开或无出口。我们没有给出一个隐式时钟的详尽列表。当然了，这部分应该被认为是浏览器中时钟技术的冰山一角

### 测量目标

回想一下，对手在定时攻击中的目标是测量某些事件的持续时间，并区分两个或两个以上可能的执行。我们假定我们的对手的目标是度量一些JavaScript target ()的持续时间，或者度量一些事件目标触发回调的时间。有许多潜在的目标，例如在web浏览器上的两个不同的定时攻击。我们将目标和攻击分类为退出和无出口，并为每个对象描述一个典型示例。

#### 退出目标：requestAnimationFrame的违背保密性

以前的工作已经展示了几种不同的方法来实现历史嗅探或交叉帧像素阅读，通过对秘密数据的SVG过滤器的渲染进行计时。Andrysco等人演示了一个对隐私的定时攻击，该攻击基于一个SVG卷积过滤器的渲染时间，以区分像素。这个定时要求攻击JavaScript准确地知道何时将SVG 过滤器应用于目标，以及何时SVG过滤器完成渲染。这是通过采样一个高分辨率的时间戳(performance.now)来完成的，当应用包含了过滤器的CSS样式时，以及当触requestAnimationFrame的回调时。在这种情况下，JavaScript必须退出以允许其他的计算发生，然后通过事件完成的回调接收通知。我们将此类型的目标称为退出目标，因为它在完成之前退出了JavaScript运行时。

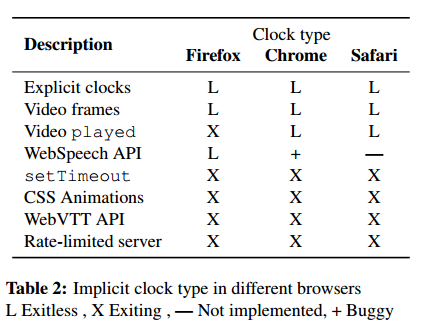
#### Exitless目标:从JavaScript缓存定时攻击

相反，也有一些exitless的目标，比如Oren等人的缓存定时攻击。此攻击不需要退出JavaScript来运行目标，相反，它们只需要执行一些同步的JavaScript函数调用，并度量它的持续时间。任何exitless目标都可以安排在回调中，从而使它成为一个退出的目标，但是一个退出的目标不能以一种exitless的方式运行。

### 浏览器中的隐式时钟

假设所有显式的时钟都从浏览器中删除，那么有动机的攻击者仍然可以度量细粒度的持续时间。攻击者不能查询一个显式的时钟，而是可以找到具有已知或可定义执行时间的浏览器的其他特性，并将其作为隐式时钟使用。

我们没有测试任何在外部观察者(比如合作服务器)上解析持续时间的时钟。例如，一个JavaScript可以生成一个网络请求，运行一个目标，然后生成另一个网络请求。这些时钟通过在section4中讨论的防御来减轻。



我们观察到，就像退出和不被执行的目标一样，也有退出和不受影响的隐式时钟。我们将参考时钟或计时方法，不需要让时钟报告的值更改为exitless。类似地，需要JavaScript执行才能在时间前退出的计时方法正在退出。

所有的exitless时钟都可以为退出和无出口目标工作。然而，一个exitless目标不能使用一个退出时钟来执行，因为目标的执行将控制主线程，停止常规回调或触发退出时钟的事件。可能有奇异的退出时钟没有这个限制，但是所有的细节都在下面做。一个exitless攻击需要同时使用一个exitless目标和时钟(例如在缓存定时攻击中)。

根据浏览器特性的实现，时钟技术可能正在退出或没有出口。一个很好的例子是对一个<audio>或<video>标记的播放信息的更新。此信息将异步更新到谷歌Chrome中的主要浏览器线程，但不会在Firefox的JavaScript执行过程中更新。因此，它可以用于在Chrome中构建一个无出口时钟，而只是在Firefox中构建一个退出时钟。

请参见表2，了解如何在Chrome 48(稳定)、Firefox3和Safari 9.0.3中显示下列时钟。

#### 无出口(exitless)时钟

由于JavaScript是单线程的和非抢占式的，exitless时钟不必担心其他JavaScript回调的调度，也不必担心目标和时间测量之间的其他事件。根据JavaScript的语义，一个无出口时钟被认为是一个运行到完成的违背，是一个错误。任何时候，JavaScript都可以观察到在单个回调过程中产生的变化，这被认为是一个bug；只有当他们的时间是可靠的时候，我们才能建造一个时钟。Mozilla已经明确表示他们的目标是让SpiderMonkey(Firefox JavaScript引擎)不受运行到完成的侵犯。

我们在不同的浏览器中发现了几个可以在JavaScript上使用的exitless的时钟。

1. 显式时钟查询。在预期的情况下，显式的时钟查询是运行到完成的违规行为，并暴露最精确的计时数据。performance.now是JavaScript中明确计时数据的最佳来源。
2. 视频帧数据。通过渲染一个<video>到<canvas>，JavaScript可以恢复当前的视频帧。由于该视频异步更新到浏览器的事件循环，因此可以反复地得到一个细粒度的时间- 原因 -视频-启动值。

在Firefox上，视频帧数据更新为60个FPS，粒度为17ms。我们可以在120FPS上加载一个视频，它不允许JavaScript更快地访问新帧，但JavaScript获取的帧有一个更精确的时钟。我们通过在120FPS上生成一个长时间运行的视频来证明这一点，它改变了整个视频每个帧的颜色。因此，通过将视频渲染到< canvas >来对当前颜色的采样，页面可以度量自视频开始以来的时间。视频可以在屏幕之外显示，或者对用户来说是不可见的，并且仍然会以60FPS的速度更新，使它成为隐式时钟的理想选择。我们还发现，使用多个视频并平均记录它们之间的报告时间提供了额外的准确性。

1. WebSpeech API。这可以从JavaScript开始/停止使用一个短语，并在停止时给出高分辨率的持续时间度量。WebSpeech API允许JavaScript定义SpeechSynthesisUtterance ,其中包含一个短语。这个过程可以从speak()开始，然后在任何时候以cancel()停止。取消可以触发回调，它的事件包含一个高分辨率的持续时间，这个时间是系统需要的时间。因此，攻击者可以启动一个短语，运行一些目标JavaScript函数，然后取消该短语以获得一个定时目标。注意尽管必须出发回调来获取持续时间值,持续时间度量当window.speechSynthesis.cancel()被调用时停止,而不是最终出发回调时。这使得WebSpeech API在Firefox中是一个伪exitless时钟，尽管我们必须在技术上等待回调以获得持续时间度量。时间向前推移，我们只是无法反复观察。由于我们只能通过停止时钟来测量时钟，所以不能用时钟边缘技术来提高时钟的精度。

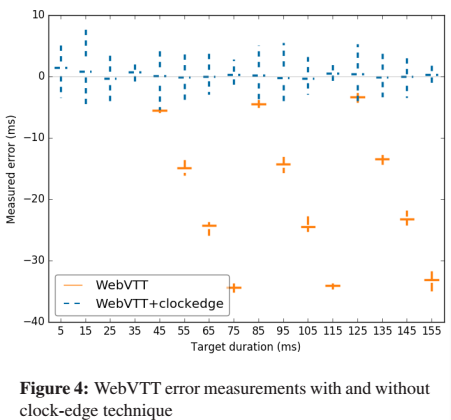
WebSpeech API只在Firefox 44 +中支持，在许多系统中需要手动启用:config。此外，除非操作系统有语音合成支持，否则时钟不能被使用，因为它永远不会开始说话。Ubuntu可以通过安装语音调度器包得到这种支持。

1. SharedArrayBuffers。虽然我们没有测试这些，但由于实现仍在进行中，JavaScript实例之间的任何共享内存都构成一个exitless时钟。正如所演示的那样，在真正的攻击这可以作为一个非常精确的时钟。

#### 退出时钟

退出时钟的数量要多得多，但对于攻击者来说也明显没有什么用处，因为他们的测量和目标执行不太可能是连续的。

1. setTimeout。设置每毫秒触发一次，然后当它们触发时设置一个全局可见的“time”变量。这是最基本的退出时钟。我们每毫秒设置超时，因为这是可以设置的最低分辨率。
2. CSS动画。设置每毫秒完成，然后在其完成回调中设置一个全局可见的“time”变量。这些行为几乎与setTimeouts完全相同，并以相同的方式度量。
3. WebVTT。这个API可以为<video>设置字幕，精确度可以高达毫秒，并检查当前显示的字幕。WebVTT接口为<video>元素提供了一种方法来添加字幕或标题，就是使用<track>元素。这些标题是由一个特殊的VTT文件装载的，它可以指定任意的字幕，以显示无限的持续时间，达到毫秒的精度。通过设置一个不同的子标题来显示每毫秒，该页面可以通过检查<track>元素的activeCues属性来确定自视频开始以来已经经过了多长时间。这仅在JavaScript未执行时才更新。
4. 下载速度有限。使用协作服务器以已知的速率向页面发送一个文件，会导致常规的进度更新在回调中排队。使用xmlhttprequest(XHRs)的onprogress事件，页面可以得到一个时钟更新函数的一致的回调流。请注意，这些回调的速率与被撤回的文件的大小有关，服务器的上载速率也一样。

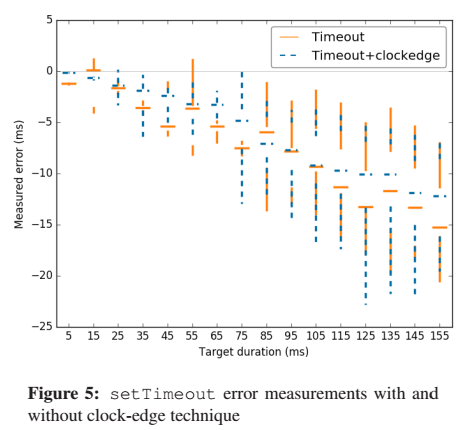


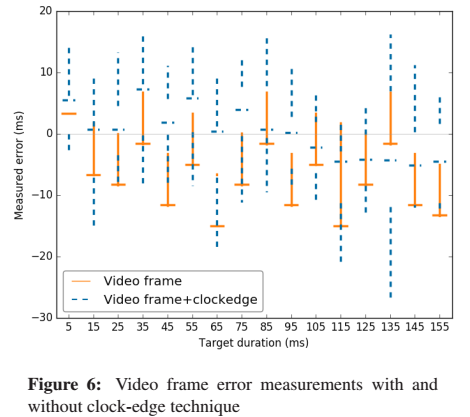
在我们的实验中，我们使用了一个大小为100mB的小工具，服务器速率限制为100kB / s，使用的是Linux实用程序细流。然后页面假设服务器发送的数据正好是100kB / s，并且有一个初始学习期，以确定onprogress触发回调的速度。完成之后，页面可以像往常一样继续运行，假定它现在有一个在计算速率的常规触发回调。请注意，onprogress事件也可以请求在加载<video>元素时触发。

1. Video/audio标签播放数据。这些包含媒体对象的间隔迄今为止都被使用。通过反复检查最远的玩点，我们可以测量事件的持续时间。在Firefox，这是JavaScript退出后的唯一更新，但在Chrome中，它会异步更新(这使它成为Chrome的一个exitless时钟)。
2. 合作的iframe / popups来自同一来源。通过在同源中创建一个弹出窗口，或者通过在源中将iframe嵌入，两页可以在相同的DOM元素上进行协作和操作。在我们的测试中，没有方法可以在这种情况下获得无出口的DOM元素操作更新。因此，这种情况减少了setTimeout案例或类似的方法。对于这些时钟，我们没有给出任何时间结果。关键的是，如果一种共享DOM元素更新的方法被发现，这将成为一个不受关注的时钟。

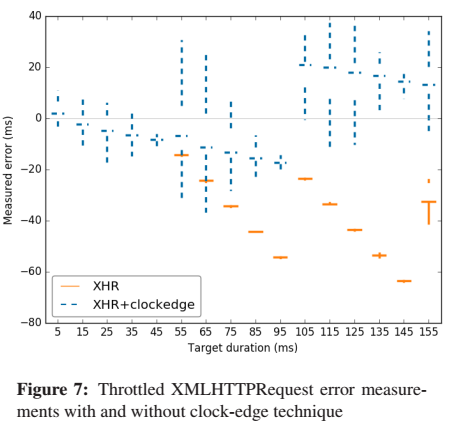
### 隐式时钟的性能

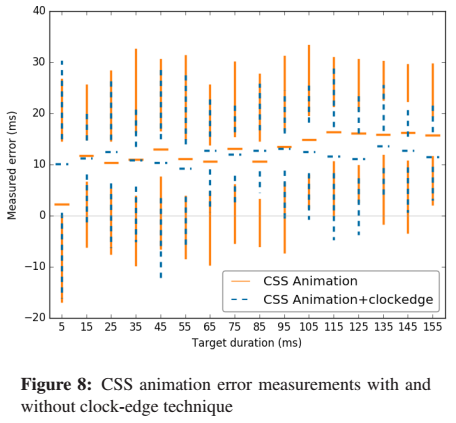
隐式时钟的粒度、精度和准确度技术上来说差异很大。我们观察到，大多数隐式时钟可以用第2节中提到的clockedge技术得到改进。通过用隐式时钟技术来替代performance.now主时钟，和使用一个合适的小时钟，大多数技术都显示出了显著的准确性提高。在这种情况下，我们想要检查区分两个不同的持续事件是多么容易。因此，一致的严格误差界限是理想的。



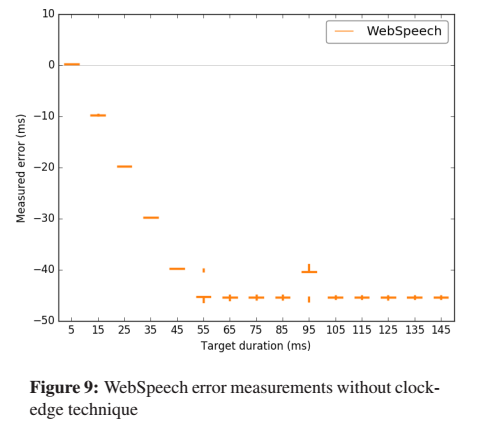


将时钟边缘技术应用于exitless时钟，只需将显式performance.now调用替换成一些其他的exitless是时钟；不需要改变小时钟。退出时钟需要一个新的小时钟技术；而不是一个紧密的循环，它必须安排定期的超时时间来检查隐式的主时钟的状态。否则，退出的主时钟在小时钟运行时不会改变状态。虽然重复的setTimeout调用会起作用，但对于每一个HTML5规范来说，setTimeout(0)实际上是一个4ms超时，使它成为一个主要的时钟。相反，我们使用重复的postMessage调用到当前窗口。这些行为的执行速度要高得多，但这段时期是未知的。因此，新的隐式主时钟现在有一个快速的、未知的小时钟，就像在exitless的情况下一样。





对和第2节中一样的Firefox进行了测量。误差(y值)被计算为时钟技术测量和被performance.now报告的实际持续时间的差值。目标时间(x值)是目标事件的预期持续时间(N毫秒)，它可能与实际持续时间稍有不同，因为系统负载，甚至是隐式时钟本身干扰退出时钟的情况。每一个目标被测量100次，测量的时间间隔为0或更少。虽然实际的持续时间与预期略有不同，但并没有相当大的噪音。



我们测量的exitless目标是一个运行时间为N毫秒的循环，由性能决定。我们的退出目标是一个N毫秒的setTimeout。

对上面描述的各种时钟技术的时钟边缘改进时钟技术，图4、5、6、7、8和9显示了在有或者没有它们情况下的错误。WebSpeech没有clockedge数据，原因3.2.1详细描述了。请注意，每个图的y轴不同，使得在时钟边缘和非时钟边缘结果之间比较更容易。正如可以在WebVTT中看到的，throttled XHRs和视频帧数据，许多时钟技术都有一个很大的他们可以操控的本机周期。这些大周期为时钟边缘留出足够的空间以提高精度。WebVTT由于其主要时钟节拍的精度，在时钟边缘情况下的显示了巨大改进；原始技术越精确，时钟边缘就越精确。

图11和图10分别显示了所有技术的平均误差的比较和clock - edge的所有技术。在这些图上，直线越接近0，平均测量的准确度就越高。同样，WebVTT对于长时间事件的clock edge的异常准确性是显而易见的。

