HISTORY / DEVICES / RESPONSIVENESS

# The History of Device Lag

*Why faster devices can still feel slow*

PUBLICATION REVISION • 11 SEPTEMBER 2026

## Abstract

A fast device can still feel late because its headline specifications describe only parts of an interaction. This paper traces that problem from shared computers and graphical interfaces to networked games, smartphones, televisions, cameras, browsers, and musical instruments. The recurring question is what the person is waiting for: immediate feedback, evenly timed updates, new information, or confirmed completion. Historical records show several routes to improvement, including less work, shorter queues, earlier preparation, and changed input rules. They also show why a useful old recommendation can become inappropriate for a different implementation. Dated cases, attributed experiments, and labelled calculations lead to a practical conclusion: define the delayed event, choose evidence that can observe it, and judge a change against that event rather than an unrelated speed score.

## Scope and evidence

Literature-based, AI-assisted research and synthesis prepared for lag.repair. No original device benchmarks, human expert review, or live-site audit is claimed. This selective history distinguishes contemporary records, later recollections, maintained documentation, and reported experiments. Research cut-off: 11 September 2026.

For a present problem, begin with chapter 13. The other chapters supply historical context, not a prerequisite reading list.

## Reading map

# 1. Faster machines, different clocks

## What feels slow—and what the number measures

Consider three complaints. A game moves smoothly but responds late to a button. Another reacts promptly but pauses when a new room appears. A video call breaks up whenever someone starts an upload. Calling all three “lag” describes the frustration; it does not identify the cause. This history begins with that distinction, because the timing of a useful response is not the same property as a processor's capacity, an average frame rate, or a connection's transfer speed. [25, 35, 20]

The problem is not confined to inexpensive or old equipment. In an independent 2017 terminal key-response experiment, Dan Luu reported roughly 30 milliseconds for an Apple IIe and 100 milliseconds for a 2014 MacBook Pro. He timed from the beginning of key movement to the screen finishing its update. This limited configuration sample supplies a counterexample to the assumption that a newer machine must deliver every interaction sooner; it does not rank entire computer generations. [51]

Within a listed Haswell-E system, the same investigation reported approximately 50 milliseconds at 165 Hz, 80 at 60 Hz, and 140 at 24 Hz. Presentation conditions changed the measured response without replacing the processor. The rounded, historical results are useful here because their conditions are visible, not because they constitute a current buying guide. [51]

Engineering can genuinely remove waiting. In 1998, the IETF explained why normally small standing queues could support both lower end-to-end delay and higher throughput by leaving room for bursts. A useful design change need not exchange every saved millisecond for an equivalent penalty elsewhere. The relevant question is which work or dependency actually changed. [60]

The chapters follow that question across different activities. Time-sharing made a continuing exchange with a shared machine possible. Touch interfaces linked movement to visible feedback. Networked games separated local response from confirmed shared state. Cameras changed when information was acquired relative to a request. The historical connection is the timing problem, not an assertion that these systems all used the same implementation. [3, 29, 18, 65]

“Lag” is used retrospectively as a reader's umbrella term. Batch turnaround, network delay, a missed display deadline, and late sound remain distinct phenomena. When a chapter explains a mechanism using maintained documentation, that documentation is not treated as the date the mechanism was invented. Measurements retain their original task and endpoints.

For a present-day problem, the useful starting question is therefore specific: **What did I do, what response was expected, and which part arrived late?** The history provides explanations for those answers. The comparisons in chapter 13 help turn them into a next test without requiring every preceding chapter to be read first.

## Throughput, latency, continuity, and completion

Latency is elapsed time between defined events. Throughput is the amount of work completed per unit of time. For a constructed example, imagine ten consecutive processing stages, each taking ten milliseconds. Once the pipeline is full, it may finish an item every ten milliseconds: 100 items per second. Yet each item spends 100 milliseconds traversing the stages. More frequent output does not, by itself, mean younger information.

The example is deliberately simplified. Real operations can overlap, branch, or wait for several dependencies. An end-to-end account should follow the sequence that determines completion rather than add overlapping intervals twice. A number without endpoints cannot tell a reader whether it covers input handling, application execution, a network round trip, or final presentation.

| Measurement | What it describes | What it does not establish |
|---|---|---|
| Network round-trip time | A defined outward and return exchange. [15] | The complete response to a game input. |
| Frame rate | Frames counted over an interval. | Even pacing or fresh input in every image. |
| Input-device latency | A specified input event to a specified report. [53] | The later rendering and display stages. |
| Audio round-trip latency | A defined input-to-output audio path. [41] | Every playback or wireless-audio configuration. |
| Browser interaction timing | Events and presentation within the API's scope. [57] | A physical switch-to-light measurement. |

Continuity introduces another clock. Consider two hypothetical four-frame sequences taking 40 milliseconds overall: 10–10–10–10 and 5–5–5–25 milliseconds. Their average interval matches. The second contains a much longer interruption. Similarly, a stream can play continuously while staying well behind its source. Smoothness, responsiveness, and synchronization can coexist, but none proves the others.

A specification may describe only one opportunity to act. A 1,000 Hz polling interval is one millisecond by arithmetic; it is not a guarantee of a one-millisecond button-to-screen response. Wimmer, Schmid, and Bockes measured 36 USB-connected input devices in a 2019 study. Latency distributions varied, and forced 1,000 Hz polling helped some devices but not all. Their electrical-event method did not include the whole mechanical action and final display path. [53]

Finally, acknowledgement and completion must remain separate. Suppose a save operation takes several seconds. An immediate “Saving” message confirms receipt; a later “Saved” message confirms success. The earlier acknowledgement improves the exchange without necessarily accelerating the storage operation. Calling both events “response” would hide the difference. This is an analytical example, not a benchmark of a particular application.

These distinctions give the history a consistent vocabulary. **Input delay** concerns late feedback to an action; **stutter** concerns uneven progression; **network delay** concerns communication; **completion time** concerns the finished operation. Everyday complaints can include several at once. The historical achievement of a design should be evaluated against the particular interval it changes, and a repair should be judged the same way.

# 2. From submitted jobs to continuous interaction

## 1956–1968: access, shared attention, and direct manipulation

IBM's 1956 RAMAC system made stored records directly accessible using magnetic disks and a moving read/write mechanism. The change concerned the operation a user could request, not simply how much data a machine could hold. Direct access was a major capability even though retrieval still required physical movement. IBM's institutional history documents the system; it does not establish that the history of waiting began with it. [1]

Access to the computer itself posed another problem. John McCarthy's participant recollection describes time-sharing as a way to divide a computer's attention among people engaged in continuing exchanges. The fiftieth-anniversary CTSS history dates MIT's first demonstration to November 1961 and operational use to 1963. These are historical accounts written later, not performance logs collected during those events. [2, 3]

Reducing an overnight calculation and preserving a conversation at a terminal serve different activities. In the first, the person waits for a completed job. In the second, each result helps determine the next action. Scheduling now affects whether a user can continue thinking and working with the machine, even when its total capacity is shared. This is the interpretation linking the records, rather than a claim that every early system used the same scheduling policy.

Ivan Sutherland's 1963 Sketchpad thesis introduced another relationship. Using a light pen, the user could construct and change drawings whose elements had maintained relationships. Its accessible Cambridge edition was published in 2003; the underlying thesis dates to 1963. Instead of submitting a drawing for later output, the person manipulated a representation and used its changes as feedback. [4]

Douglas Engelbart and his colleagues' NLS demonstration on 9 December 1968 brought mouse-driven editing, links, and remote collaboration into one public presentation. It demonstrated the results of an existing research program, not the simultaneous invention of every feature shown. The institute's archive preserves the event and supporting records. [5]

Together these developments changed what responsiveness had to support: obtaining a record, sustaining an exchange, and maintaining a useful relationship between hand and screen. None of those tasks can be evaluated solely by how quickly the computer finishes an unrelated calculation. The distinction is visible in a simple thought experiment: a dragged object that trails the hand is showing a position the user has already left, even if other calculations run quickly.

The earlier systems should not become a nostalgic baseline for every later product. Their workloads and capabilities differed. What this period establishes is the emergence of interaction as an engineering requirement in its own right. It also supplies the central question for the device histories that follow: which part of the person's activity must the machine keep up with, and what can safely finish later?

# 3. Local computers and television deadlines

## The 1970s and 1980s: owning the machine did not remove timing

The Atari VCS's 1979 *Stella Programmer's Guide* describes a direct relationship between software and a television raster. Its NTSC-oriented account uses 262 scan lines per frame and explains how software supplies information in time for the line being drawn. The surviving source used here is an archival transcription of the manufacturer's guide, not an end-to-end test of a particular console and television. [9]

That record complicates the idea that visual timing problems arrived only with complex graphics. The challenge was not simply to calculate an image eventually. Information had to be available when the display needed it. A small scene could still impose a deadline. The scan itself also unfolded over time, so “no frame buffer” should not be transformed into “no elapsed time.”

At exactly 60 refreshes per second, the interval between refreshes is about 16.67 milliseconds. At 120 it is 8.33 milliseconds, and at 240 it is 4.17 milliseconds. These are calculations, not whole-system input-lag figures. Historical television timings and modern presentation modes vary, and different points on a scanning display need not change simultaneously.

IBM's introduction of the 5150 Personal Computer in August 1981 marks another part of the transition to local use. IBM's own history places it within an existing personal-computer market; it was not the first computer an individual could own. Processing, storage, memory, and peripherals were now assembled into a machine whose user could experience delays as problems with “my computer.” [7]

Local ownership changed some dependencies, not every access cost. Storage and resident memory still served different roles. Present-day Windows documentation offers a precise mechanism without pretending to describe every early operating system: a working set consists of pageable memory resident in physical memory. A hard page fault requires backing-store access, while a soft fault can be resolved without it. A page-fault count is consequently not identical to a count of disk reads. [8]

That mechanism explains a conditional benefit of more memory. Retaining needed data can avoid repeatedly retrieving it from storage. But adding capacity does not establish that a display will present sooner or a remote server will reply faster. A useful comparison asks which wait the added capacity removes, rather than using a larger specification as proof that all operations improved.

The two histories in this chapter illuminate different constraints. Raster programming made a presentation deadline conspicuous. Personal computing made a collection of local resource dependencies familiar. Neither can be reduced to processor speed alone. A system might complete plenty of arithmetic while waiting for data, or finish drawing work after its intended presentation opportunity.

For readers, this distinction prevents a common false choice: either the whole computer is fast or the whole computer is slow. A device has many operations with different timing paths. Identifying one slow path is more informative than attaching a single adjective to the machine.

# 4. The graphical desktop's divided attention

## The 1990s: useful software, expanding work, and frozen windows

A general-purpose interface creates an expectation that the machine will remain available while other work proceeds. Windows scheduling documentation describes priority-based selection of runnable threads and sharing among threads at the same priority. This allocates processing opportunities. It does not ensure that an application has organized its internal work so that user interaction remains available. [10]

Microsoft's Windows 7-era guidance on application hangs identifies a concrete failure boundary. A window's thread must continue processing messages, including input and redraw requests. Long calculations or blocking operations on that thread can leave it unable to respond. The processor need not be fully occupied: the application may be waiting for something else while the user is also waiting for the application. [11]

This modern documentation explains a mechanism; it does not date the invention of unresponsive interfaces to Windows 7. The historical question is how expanding desktop responsibilities made coordination increasingly important. An operating system's ability to run other work does not automatically make one blocked window useful to its owner.

Niklaus Wirth's February 1995 essay *A Plea for Lean Software* provides contemporary evidence of concern about software growth. He criticized increasing bulk without corresponding usefulness and argued for disciplined, simpler design. The essay shows that frustration with software consuming hardware gains long predates present-day phones and browser applications. It is an engineering argument, not a measured law that every new release becomes slower. [12]

A defensible history must also preserve the other side of that argument. Additional functionality is not automatically waste. In a constructed example, an editor might gain accessibility support, recoverable saves, and richer documents. These changes can require more work while making the product more useful. The relevant question is which costs are necessary, when they are paid, and whether they block an interaction that should remain available.

There are therefore two different criticisms. One concerns unnecessary work. The other concerns necessary work placed at the wrong point in the user's exchange. Removing an expensive feature and moving its processing off the immediate feedback path are not the same intervention. A benchmark of one operation should not conceal the loss of functionality elsewhere.

Nor does changing scheduling priority provide a universal answer. Higher priority changes competition for processor attention; it does not complete a blocked remote request. Microsoft's documentation warns that high and real-time priorities can interfere with important system activity. Treating such settings as consequence-free speed switches ignores what they actually control. [10]

The desktop era leaves a useful description more precise than “the computer froze”: **the machine may be working, the program may be waiting, and the user may be unable to proceed**. Those states require different evidence. Overall utilization can be part of an investigation, but it cannot explain by itself what the interface is waiting for. The next chapters extend that boundary beyond the local machine.

# 5. The Internet adds a journey

## 1981–1999: measuring communication rather than assuming it

The Internet Control Message Protocol specification of September 1981, RFC 792, includes Echo and Echo Reply messages. They provide part of the foundation for familiar round-trip probing. Their existence does not mean that a reachability test measures an entire application. It means that one communication exchange can be described using explicit start and return events. [13]

The distinction became formal in the IETF's 1999 one-way and round-trip delay metrics. RFC 2679 belongs to that historical record, but RFC 7679 superseded it in January 2016. One-way testing depends on the timing relationship between endpoints. A round trip also includes the return path; dividing it by two does not establish either direction's actual delay when conditions differ. [14, 15, 56]

These definitions changed what a useful explanation of a connected device required. The visible symptom might originate locally, in the access network, elsewhere along the route, or at the service. An attractive network number is evidence about its test, not certification that all those stages are responsive.

Bandwidth describes another property. For an illustrative packet of 1,500 bytes, ignoring overhead, placing 12,000 bits onto a ten-megabit-per-second link takes 1.2 milliseconds. At 100 megabits per second, it takes 0.12 milliseconds. This is serialization time. Neither number specifies the previous queue, physical journey, receiver processing, or response journey.

The calculation shows both why capacity matters and why it is not the entire answer. Reducing serialization can shorten an exchange. Once a different stage dominates, an equally dramatic capacity increase may have a smaller effect on the total. This is not a reason to dismiss bandwidth; it is a reason to identify the term that an upgrade changes.

Stuart Cheshire's essay *It's the Latency, Stupid*, begun in May 1996 and subsequently revised, made this distinction through interactive networking and his experience with the game Bolo. It challenged the tendency to treat higher advertised connection rates as a complete account of performance. The page explicitly discusses the continuing importance of bandwidth as well as latency. Its revision history prevents every passage from being dated confidently to the first month shown. [59]

A second complication is variation. RFC 3393, published in 2002, defines an IP packet-delay-variation metric and notes that “jitter” is used in different ways. The size, timing, and selection of packets matter to interpreting a measurement. A single average can conceal periods in which a communication-dependent activity becomes difficult. [16]

The network's historical contribution to lag is thus more than extra distance. It creates shared conditions, variable dependencies, and another set of measurement boundaries. A connected application can respond poorly even when the local processor is capable and a large transfer completes quickly. The explanation must follow the exchange rather than assign the whole experience to the device's generation or the connection's headline rate.

## 1984: why deliberately waiting could be a rational design

John Nagle's January 1984 RFC 896 supplies a contemporary case of efficiency and responsiveness pulling in different directions. It described a small-packet problem in interactive traffic: sending an individual character could place one byte of useful data behind forty bytes of protocol headers. Producing a packet for every tiny application action was not necessarily a sensible use of a congested network. [58]

The document discussed timer-based bundling and the difficulty of choosing one timer for very different network conditions. A delay useful on a slow path could needlessly hold up an exchange on a fast local network. Its proposed approach inhibited additional small transmissions while earlier data remained unacknowledged, allowing adaptation rather than imposing the same fixed pause everywhere. This is historical explanation, not an instruction to apply a particular modern socket setting. [58]

The important lesson is that an observed wait can be intentional without being appropriate in every setting. A design may improve the behavior of the larger shared system while creating an interaction that needs special treatment. Equally, removing an intentional wait can worsen other behavior. The correct evaluation concerns the workload and the system around it, not whether “delay” appears in the mechanism's description.

A constructed comparison clarifies the point. Suppose ten one-byte updates can be collected into one transmission, but the first must wait for the other nine. The receiver gets a more efficient bundle, while the first update is older. Whether that matters depends on whether the receiver needs each update immediately or only the final collection. Both accounts can be true without either metric being fraudulent.

Congestion control also makes the connection more than an always-open pipe running permanently at the advertised maximum. RFC 5681's 2009 specification describes TCP behavior including slow start, congestion avoidance, and loss-related recovery. These mechanisms respond to feedback from the path. They are part of the explanation for delivered service, not merely a property of the subscriber's access speed. [17]

Protocol history should therefore resist simplistic tuning folklore. A mechanism introduced to manage one class of failure is not proved harmful merely because it can add waiting. Conversely, its original rationale does not establish that it suits every later application. Nagle's memo is now classified Historic; read it as evidence of the problem addressed, not as a current tuning standard. [58]

This is also where physical and avoidable limits must be separated. More direct routing or relocating work changes the journey; removing unnecessary bundling changes a local decision; a faster link changes serialization; avoiding a repeated exchange changes the dependency structure. Calling all four “lower ping” would lose the reasons they work.

The emerging historical pattern is not that designers discovered a single correct amount of waiting. They learned to specify which information had to arrive, which uncertainty could be tolerated, and which resources had to be shared. Networked games made those choices especially visible because another person's actions could change the meaning of the information while it was still travelling.

# 6. Games respond before certainty arrives

## Prediction, correction, and the limits of “zero lag”

Internet multiplayer made a tension unavoidable: a player wants immediate local control, but several players need a coherent shared account of events. Glenn Fiedler's developer retrospective describes a contrast between Quake and QuakeWorld in 1996. Instead of waiting for authoritative server feedback before showing all local movement, a client could predict its player's movement and later reconcile differences. This is a retrospective explanation, not proof that one game invented every form of prediction. [18]

The distinction is fundamental. Prediction does not shorten the packet's physical trip. It changes which part of the experience must wait for that trip. A player can receive useful local feedback while the application remains uncertain about the final shared state. The resulting correction is a consequence of that choice, not necessarily a failure to make the local processor fast enough.

The GGPO project's description of its rollback networking SDK presents a related approach for suitable games. It identifies a 2009 SDK and describes predicting input, retaining state, and restoring and resimulating when later information differs from the prediction. That project-supplied date should not be confused with a demonstrated date for the first rollback experiment or every earlier use of the name. [19]

Rollback trades delayed knowledge for the possibility of revision. It requires a game that can support the relevant state management and simulation behavior. An application cannot gain the result simply by displaying an attractive “rollback” label, and the technique does not promise that corrections are always invisible. Its significance is that simulation can revisit a provisional account rather than always postpone the local action. [19]

A documented case shows why a plausible-looking ping number can miss the problem. On 24 May 2022, Riot Games described VALORANT movement buffers that could remain overfilled after frame-rate changes or network spikes, adding delay absent from the Network RTT graph. Its Patch 4.10 account reported faster recovery and new processing-delay diagnostics. In its example, clearing five excess moves changed from up to five seconds to less than one. These are the developer's historical findings, not a diagnosis of a current installation. [70]

It also explains why several complaints should remain separate. Controls may respond late while motion stays smooth. Motion may stutter even in an offline scene. Shared positions may jump when the application revises state. A person can reasonably call each experience “lag,” but the same evidence cannot diagnose them all.

This case adds a second lesson to the prediction story: improving observability can matter before selecting a remedy. A familiar number may omit the interval behind the complaint. Finding that omission does not prove that every similar symptom shares the same cause; it identifies what the next measurement needs to include.

Claims such as “zero added input delay” must be interpreted at that boundary. A system might deliberately avoid waiting for remote input before providing a particular local response. That is not a measurement showing that sampling, computation, transmission, and display all consume no time. The historical achievement is reducing the dependency of one response on another event. An honest performance account names that response, the information still missing, and the consequences when the provisional answer changes.

# 7. The queue is not the capacity

## 1998–2018: from full buffers to controlled waiting

A buffer's capacity and its usual occupancy are different properties. Capacity can absorb a burst; a persistently occupied queue means that newly arriving information waits behind older work. Confusing the two makes a larger memory allocation look automatically beneficial, even when the immediate problem is the time spent in it.

The IETF's April 1998 RFC 2309 made this distinction explicit. It argued that leaving queues normally small preserved room for bursts and could improve throughput as well as delay. A nearly full queue could cause bursts of drops and coordinated slowdowns among traffic sources. The document is now obsolete, superseded by RFC 7567, but remains important contemporary evidence against the idea that the problem was first recognized during recent broadband use. [60]

The 2015 successor, RFC 7567, recommended active queue management to control persistent waiting and improve congestion behavior. The reader-facing problem commonly called bufferbloat concerns excessive queueing, not the mere existence of a buffer. It is possible to move a large amount of data while an interactive exchange waits too long behind other traffic. [20]

The scale follows simple arithmetic. In an illustrative first-in, first-out queue, 1,000,000 bytes ahead of a packet represent 8,000,000 bits. At a constant drain rate of 10,000,000 bits per second, they represent 0.8 seconds of waiting, ignoring overhead and other effects. This is not a measurement of any household router. It shows why the amount of stored work must be interpreted relative to the service rate.

CoDel's January 2018 RFC describes controlled delay using the time packets spend in a queue and distinguishing persistent waiting from transient bursts. Its bibliography records the authors' earlier 2012 work; the RFC publication should not be mistaken for the algorithm's first appearance. The document was published as Experimental, not as a guarantee that every deployed network implements it successfully. [21]

FQ-CoDel's companion RFC combines queue management with flow scheduling. Those functions answer related questions: how much persistent waiting to allow, and which traffic gets served next. The specification is also Experimental. Its mechanism supports investigating interaction under competing traffic rather than evaluating a connection solely with an isolated bulk transfer. [22]

The historical progression is therefore more precise than “routers acquired a fix for lag.” Designers distinguished bursts from standing backlogs, recognized the role of feedback, and combined controls over waiting with controls over service order. Implementation, location, and the actual bottleneck still matter.

That final condition follows directly from the explanation. A control acting on a different queue cannot establish that the queue causing the observed delay has changed. A useful household comparison can test a fixed destination under idle and loaded conditions, but the result initially describes the tested path. Locating the consequential queue requires further isolation. The distinction between a path-level observation and a component-level diagnosis prevents an informative experiment from becoming an unjustified purchase recommendation.

## Wireless sharing and the continuing system problem

Wireless adds a shared-medium constraint. Cisco Meraki's channel-planning documentation describes listening before transmission, deferring while a channel is busy, and retransmitting when delivery fails. These mechanisms explain why strong received signal does not establish that airtime is available when an application needs it. A well-connected device may still compete with other traffic. [23]

A supported wired comparison can narrow an investigation, provided the destination, workload, and other conditions remain comparable. Improvement says that something relevant changed between the paths. It does not identify a particular neighbor's network, prove that every wireless device is unsuitable, or locate the whole problem from signal bars. This is a proposed method, not an experiment performed for this paper.

The ITU's 2017 IMT-2020 requirements show another source of confusion. Their user-plane targets include four milliseconds for enhanced mobile broadband and one millisecond for ultra-reliable low-latency communication. The report defines a radio-interface interval under specified unloaded, active-device, small-packet conditions. These are not promised round-trip times from a phone to any Internet service. [24] (section 4.7)

A large improvement in that interval can still be valuable. It simply must not be substituted for the full application path. Reducing an illustrative ten-millisecond stage to one millisecond saves nine milliseconds; it does not eliminate an unchanged 100-millisecond dependency elsewhere. A component percentage and the total experienced improvement can legitimately differ.

The network history continues beyond the early bufferbloat debate. RFC 9330, published in 2023, describes the architecture of Low Latency, Low Loss, and Scalable Throughput, or L4S. It combines changes in congestion signalling, network behavior, and sender response, with attention to coexistence with conventional traffic. The architecture RFC is Informational. Its publication establishes a documented design, not universal deployment or a guarantee that every Internet interaction has become a one-millisecond exchange. [61]

That development reinforces rather than overturns the earlier lesson. Some improvements require cooperation across boundaries. A service, endpoint, and network can participate in a lower-delay design; a faster consumer specification alone does not prove that all the required behavior is present.

The recurring temptation is to choose an easily visible proxy: signal strength, wireless generation, router memory, or connection capacity. Each can be useful information. None is the response itself. The stronger question is how the relevant exchange behaves under the conditions in which the user experiences trouble.

This distinction also protects the history from technological pessimism. Better coordination can reduce avoidable waiting rather than merely relocate it. But describing that success requires evidence at the correct boundary. A radio target, a queue-control architecture, and a measured application response should appear as separate claims, not as interchangeable proof that a new generation has solved “lag.”

# 8. Touch brings the delay under the finger

## 2007–2015: coordinated rendering meets human perception

Apple's January 2007 iPhone announcement emphasized multi-touch interaction. It was not the invention of either every touchscreen technique or the smartphone category. It is a contemporary record of a major consumer product making finger-driven manipulation central to its presentation. Content was expected to follow an action performed directly against the screen. [28]

Android 4.1's Jelly Bean documentation describes a coordinated response to that requirement. It extended vertical-synchronization timing across rendering, touch handling, composition, and display refresh; described triple buffering; predicted finger position at refresh; and boosted CPU activity following input. These were documented design measures, not proof that every device achieved the same measured result. [29]

The combination matters historically. Responsiveness was addressed through scheduling, anticipation, and coordinated presentation rather than only a faster processor. It also prevents a blanket rule that all buffering is harmful. The question is how stored work participates in a particular pipeline, not whether the word “buffer” appears in a feature description.

Human expectations require equally careful boundaries. Jakob Nielsen's 1993 discussion, later updated, presented influential approximate response-time limits of a tenth of a second, a second, and ten seconds for different interface concerns. He linked the guidance to earlier work, including Robert B. Miller's 1968 research. These are design heuristics for an exchange, not universal thresholds below which delay becomes impossible to perceive. [6]

A 2015 CHI study by Jonathan Deber and colleagues made task differences measurable. Under its experimental conditions, reported detection thresholds for dragging were approximately eleven milliseconds with direct touch and 55 with indirect touch. For tapping, the corresponding figures were 69 and 96 milliseconds. The authors also distinguished detecting a latency improvement from establishing its desirability or its effect on task performance. [52]

Those distinctions matter more than turning the four numbers into new universal targets. A study can establish that a difference was detectable in its task without proving that every user notices it during every action. It can establish perception without proving a competitive advantage or a worthwhile purchase. The appropriate question is not merely “Can a human notice delay?” but “Which delay, in which activity, judged by which outcome?”

This supports a stronger historical interpretation than the claim that expectations simply became more demanding. The activity changed. Waiting for a search result, tapping a control, and dragging an object place different demands on feedback. A device can satisfy one while disappointing another.

These experiments also separate a noticeable change from a worthwhile one. A measured improvement can be real without altering the user's task, while an apparently small difference may matter in a tightly coupled action. A comparison needs an activity and an outcome as well as a stopwatch. [52]

## The phone does not have one permanent speed

A high-refresh screen creates more presentation opportunities, not proof that an application meets every deadline. Android's rendering guidance describes missed frame deadlines and tools for investigating slow rendering. At a higher refresh rate, the available interval can be shorter. Smooth motion therefore depends on work arriving in time, not only on a panel's advertised capability. [30]

The problem also changes over a session. Android's Thermal API documentation describes monitoring thermal conditions and adjusting workload as a device approaches throttling, with support and behavior varying across devices. A short test on a cool device need not describe the performance it can sustain through a longer workload. The documentation establishes that operating conditions matter; it does not diagnose every warm phone. [31]

A useful comparison must consequently preserve or report session duration, workload, charging state, power mode, and relevant environmental information. Otherwise, an apparent setting improvement may simply coincide with a different operating state. These are proposed controls for testing, not temperature measurements collected for this history.

Battery condition introduces a separate mechanism. Apple's support documentation explains that chemical aging can reduce an iPhone battery's ability to deliver peak power. On affected devices, performance management can change system behavior to help prevent unexpected shutdowns, with possible effects including longer launches or lower frame rates under particular conditions. This is a manufacturer-described mechanism with model and condition dependencies. It does not prove that all aging phones are deliberately slowed in the same way. [32]

Heat-related limits and battery-related power limits can coexist, but they need different evidence. Replacing a battery addresses a relevant battery condition; it does not establish a remedy for a slow service, a rendering defect, or every interruption. The diagnostic error is not considering the battery. It is turning one documented explanation into a conclusion before checking whether it applies.

Some widely repeated rituals can also pay a cost they conceal. Apple's slow-device guidance warns that unnecessary force-closing can make subsequent opening slower because an app must reload data. Closing an unresponsive application is a different action from routinely discarding useful state in the belief that every cleared item produces free speed. [33]

These cases extend the history beyond “new versus old.” A single device can deliver different timing under different workloads and states. Its peak specification is not its entire operating envelope, and a short benchmark is not automatically a sustained-use result.

The appropriate response is neither to distrust every optimization nor to disable safeguards in pursuit of a favorable short run. Compare supported conditions, retain the baseline, and test the mechanism actually implicated. A claim that a device is faster should say at what, for how long, under which conditions, and with what compromise. Without that context, a temporary change can be mistaken for a durable repair.

# 9. More frames are not always fresher frames

## Television game modes, render queues, and generated frames

A display can be late after the computer has finished its work. NVIDIA's system-latency guide distinguishes display processing, scanout, and pixel response from the preceding input and rendering path. A fast pixel-transition specification therefore is not a complete button-to-screen measurement. [25]

Television operating modes make the distinction concrete. The HDMI Forum's release of HDMI 2.1 on 28 November 2017 included Auto Low Latency Mode, or ALLM. This is a dated specification milestone, not the first appearance of game modes or a record of performance on every television. [66]

HDMI's feature explanation describes a source device signalling the display to enter a low-latency mode. Some TV processing features may stop to reduce delay; when the signal is withdrawn, the display can return to its previous mode. ALLM automates a choice about processing. It does not speed up a game server or prove that every stage has become instantaneous. [67]

That is a different intervention from changing the graphics processor's work queue. NVIDIA's 2020 Reflex announcement described coordinating CPU and GPU activity to reduce system latency, including delays associated with queued rendering. Its design explanation treats responsiveness as a pipeline property; any reported performance still belongs to the tested configuration. [26]

Advice also has an implementation history. NVIDIA's 2020 guide favored exclusive fullscreen under its tested conditions. Microsoft's flip-model documentation describes efficient windowed paths, including Independent Flip. These sources do not establish a universal winner; they explain why the presentation path and software environment must accompany the recommendation. A setting name alone cannot tell a reader whether older advice applies. [25, 27]

In September 2022, NVIDIA introduced DLSS 3 as a combination of Super Resolution, Frame Generation, and Reflex. Its comparisons with native rendering changed several features together; they do not isolate the causal contribution of frame generation. Increasing the number of displayed images and shortening the measured response to new input are separate questions. [55]

A constructed example shows why. Two systems might display an image every eight milliseconds while one uses substantially older input. Their output rates match, but their responsiveness does not. Conversely, a combined optimization could improve both. The frame counter cannot decide which happened.

These cases describe three different places to intervene: the display's processing mode, the computer's work queue, and the production of images. A practical comparison should identify which changed. A supported TV game-mode test can investigate display-side delay; it cannot by itself establish the cause of online position corrections. The same screen can be where several unrelated delays become visible.

## Solid-state storage moves the investigation beyond retrieval

Solid-state storage avoids the moving read/write mechanism used by a hard disk. IBM's technical explanation supports this distinction between flash-based storage and mechanical access. It explains an important source of shorter access times, not a promise that every application pause disappears after installing an SSD. [49]

The comparison with RAMAC is historically useful because it separates obtaining bytes from completing the operation that needs them. Retrieving an asset is not the same as interpreting it, decompressing it, creating the required resources, and presenting its effect. A faster store can remove an important cost while leaving other work on the same interaction path.

Microsoft's November 2022 DirectStorage 1.1 release made that shift explicit by adding GPU decompression support. This was a developer capability for suitable applications, not a switch that retroactively rewrote every existing game's loading behavior. It illustrates optimization moving beyond transfer rate into what happens to transferred data. [34]

Epic's Unreal Engine PSO precaching documentation identifies another source of interruptions: pipeline-state preparation that has not completed before it is needed. It discusses runtime hitches and strategies for handling readiness. This establishes a documented mechanism, not a diagnosis of every pause after a driver change or on entering a new area. [35]

A game might render an ordinary scene quickly but encounter costly preparation at an inconvenient moment. An intervention affecting its normal rendering cost may not remove that separate operation. Conversely, a pause that resembles a preparation hitch could have another cause. The appropriate conclusion depends on a trace, developer information, or a comparison that distinguishes the alternatives.

The distinction suggests how to preserve evidence. In a proposed repeatable scene test, record the first traversal separately from a repeat traversal. Keep the game version, settings, workload, and relevant cache conditions visible. Combining different conditions into one favorable average can obscure whether a change affected first-use preparation, ordinary rendering, or neither.

It also explains why clearing caches is not a universal improvement. In a hypothetical workload, discarding retained results may resolve a specific invalid-state problem, but it can also require useful preparation to be performed again. The action must be connected to the evidence for the particular application. “Fresh” state and faster state are not synonymous.

This chapter supplies a correction to the idea that improvements merely fail when a complaint remains. A storage upgrade may genuinely shorten retrieval while a different stage becomes the largest remaining cost. The first improvement is real; the total interaction still contains more than one dependency.

The practical conclusion is not that readers need to become graphics-engine specialists. It is that a broad symptom should not produce an unbounded sequence of purchases. “Still stutters on an SSD” is a reason to investigate the remaining work, not proof that the SSD accomplished nothing or that the next larger component will necessarily repair the experience.

# 10. Sound can be smooth and still be late

## From MIDI instruments to buffered and wireless audio

The MIDI Association's participant-informed history describes a January 1983 NAMM demonstration connecting a Sequential Circuits Prophet-600 and a Roland Jupiter-6. Instruments from different manufacturers could exchange performance commands. MIDI is not the resulting sound recording: it tells another device what to do. The demonstration belongs to the history of responsive connected instruments, not only to music-file storage. [68, 69]

A later specification makes a timing limitation explicit. RFC 6295, published in June 2011, explains that traditional MIDI DIN transmission takes 320 microseconds per byte and does not attach timestamps to commands. Two two-byte commands that originate simultaneously arrive separated by 640 microseconds. From arrival alone, the receiver cannot know whether that separation came from the performance or the serial transmission. [69]

The distinction is between **when information arrived** and **when the action was intended**. Timestamped command formats can represent intended timing, but cannot make a late command arrive earlier. Nor does the serialization figure measure a complete key-to-ear path: the instrument still has to interpret the command and produce sound. These are limits of what the cited mechanism establishes, not a listening test of MIDI instruments. [69]

Digital audio brings a different timing bargain. Ableton's documentation explains that larger processing buffers increase latency while allowing more margin against dropouts; effects and delay compensation can add further time. At 48,000 sample frames per second, 256 frames represent 5.33 milliseconds and 1,024 represent 21.33 milliseconds. These are calculated durations of one buffer, not complete live-monitoring results. [42]

Android's audio documentation separately defines input, output, round-trip, touch, and warmup latency, with results varying by device and software build. Bluetooth SIG's distinction between Classic Audio and LE Audio describes standards capabilities, not the measured application-to-ear delay of every source and receiver. A wireless-generation label cannot replace a declared measurement path. [41, 43]

Synchronization is a further question. In an illustrative video player, delaying the picture could align it with late sound while leaving both equally far behind their source. That could help a viewer, but it would not make a live instrument respond sooner. Likewise, buffering uneven packet arrivals may protect continuity while retaining older material. RFC 3393 discusses the relevance of delay variation to buffering. [16]

Music makes the cost of imprecise language clear. A performance can be delayed, unevenly timed, interrupted, or misaligned with a picture. The next test depends on which happened. To investigate a live monitoring complaint, preserve the live input-to-output path rather than substituting a recording that merely appears synchronized. The useful comparison is tied to the musical activity, not to the smallest unrelated number in a specification.

# 11. Cameras: capturing before the shutter press

## Why zero shutter lag does not mean zero processing time

A camera makes the distinction between response and completion unusually concrete. A photographer wants to capture a moment, not merely see a button animation. The time represented in the image can differ from the time at which the finished image becomes available. A useful account must keep those events separate.

Marc Levoy's October 2014 explanation of Google's HDR+ described the Nexus 5 and Nexus 6 capturing a burst after the shutter press and combining the pictures. The documented burst took roughly one-third of a second to one second, depending on darkness; combination required additional time. Short exposures and merging were used to improve difficult photographs. This was a designed capture-and-processing sequence, not evidence that the phone spent the whole interval failing to notice the press. [64]

Google's April 2021 account of HDR+ with Bracketing describes a different capture arrangement. With zero shutter lag, viewfinder frames from before the shutter press are used in HDR+ merging. The bracketed design additionally captures a longer exposure afterward. The publication documents these techniques in that system; it does not date the invention of every form of zero shutter lag to 2021. [65]

The contrast shows a way to change the moment an image represents without making all processing instantaneous: acquire some of the useful information before the request. Exposure, selection, merging, and completion remain distinct. That is an inference from the two documented designs, not a claim that every camera app follows the same timeline. [64, 65]

Consider a hypothetical photograph of a person jumping. One delay could cause the selected frame to show the person after landing. Another could leave the correct airborne moment captured but require waiting for the finished picture. Both may be described as a slow camera; they imply different failures. A timing account that stops at the first thumbnail would not necessarily reveal either completely.

| Reader's observation | The timing question to preserve |
| --- | --- |
| The photograph shows a later moment than intended. | When were the contributing frames captured relative to the press? |
| The moment looks right, but the finished image takes time. | Which preparation, combination, or output stage is still unfinished? |
| A low-light mode behaves differently. | Did exposure or capture strategy change, rather than only processing speed? |

This comparison is an editorial framework, not a device benchmark. It prevents “zero” from silently moving between intervals. A camera may reduce shutter-related delay while still benefiting from longer acquisition or later processing in some modes. Evaluating the result also requires the photograph: reducing a waiting interval is not automatically a better outcome if the image no longer captures the intended scene.

The camera case expands the history beyond screens that show computed state. Here the computer is choosing how to collect evidence of a changing physical world. That makes the timing of acquisition as important as the timing of display.

# 12. The device becomes part of a larger application

## The browser's 300-millisecond wait, and what replaced it

Some mobile-browser delay came from interpreting a gesture, not insufficient computing power. Chrome's revised technical account dates removal of the familiar 300–350 millisecond tap delay on mobile-optimized sites to Chrome 32 in 2014. The browser no longer had to preserve the same double-tap behavior there, while pinch zoom remained available. Its web page has mixed-era revisions, so this is an attributed historical account, not a current browser-compatibility table. [62]

WebKit supplied a contemporary explanation in December 2015. After a first tap, it waited 350 milliseconds to learn whether the user intended a second tap for zooming. Its fast-tap work removed that wait in defined mobile-page conditions. The documented `touch-action: manipulation` behavior allowed panning and pinch zoom while excluding double-tap gestures. The improvement changed an input-interpretation rule; replacing the phone's processor was not the mechanism. [63]

The lesson is not to disable zoom in pursuit of speed. These records show responsiveness improving while retaining pinch zoom. They also show why a fixed pause may require a different investigation from a long computation. A timer waiting for another possible action is not necessarily evidence that the current action is computationally expensive. [62, 63]

Other browser delays really do involve unfinished work. Google's rendering guide separates JavaScript, style calculation, layout, paint, and compositing. Its long-task guidance describes main-thread tasks exceeding fifty milliseconds and the need to yield execution, rather than merely divide one uninterrupted task into more function calls. The fifty-millisecond threshold names a performance category, not a universal perceptual limit. [36, 37]

On 12 March 2024, Interaction to Next Paint replaced First Input Delay as a Core Web Vital. INP addresses qualifying interactions beyond the first input, but does not certify completion of every asynchronous operation. The W3C's March 2026 Event Timing Working Draft also makes the instrument boundary explicit: its event-duration value is rounded to the nearest eight milliseconds. Neither is a physical switch-to-light benchmark. [38, 50, 57]

Transport changes solve another part of the path. QUIC's 2021 specification supports independent streams; HTTP/3's 2022 specification maps HTTP onto it. Avoiding a form of cross-stream blocking does not remove work inside the browser or the server. [39, 40]

A useful history therefore asks why the browser waited. Gesture recognition, queued execution, resource delivery, and confirmed remote completion are different explanations. Eliminating the old tap timer was real progress, but did not certify every later web interaction as immediate. The browser became more responsive by addressing specific dependencies, not by making the word “loaded” mean that all possible work was finished.

## Remote services and spatial displays: how old is the information?

Cloud gaming places another sequence between action and feedback: local input handling, transmission, remote application work, rendering, encoding, return delivery, decoding, and presentation. This is a conceptual decomposition; implementations can overlap stages. An honest measurement must avoid double-counting overlap while preserving the dependencies that determine when the response appears.

NVIDIA's GeForce NOW requirements distinguish bandwidth for streaming modes from network-latency conditions and client requirements. They illustrate that sufficient capacity to carry video is not a complete input-to-display measurement. Service requirements are maintained operating guidance, not universal thresholds for every remote-computing product. [44]

Moving work elsewhere can relieve a constrained local device while adding communication and delivery dependencies. It should not be assumed inherently better or worse than every local alternative. The meaningful comparison is between actual arrangements for the same intended activity, not between an ideal local machine and a poorly specified remote service.

Large services also expose the importance of exceptional delays. Jeffrey Dean and Luiz André Barroso's 2013 *The Tail at Scale* describes why occasional high-latency episodes become consequential as services grow in complexity, scale, or utilization. The source consulted here is the authors' institutional publication record and abstract; it supports that broad argument, not a reconstruction of their experiments. [54]

A constructed probability example shows the dependency problem. If a response needs all 100 independent subtasks and each has a one-percent chance of being slow, the probability of at least one slow subtask is one minus 0.99 to the hundredth power: about 63.4 percent. This is illustrative arithmetic, not Google data or a model of every service. Real tasks can be correlated, optional, or redundant.

Spatial displays make information age visible in a different way. Microsoft's HoloLens documentation describes reprojection that adjusts presentation for changes in viewpoint and explains artifacts involving depth and mismatches with the rendered scene. This is a platform-specific technique. It does not establish that every headset behaves identically or that one universal comfort threshold has been met. [45]

Imagine turning your head after a frame's viewpoint is selected but before it is displayed. The image may be delivered perfectly yet represent a direction already left. More pixels add detail without necessarily making that viewpoint newer. Reprojection and prediction address aspects of this age problem; they do not produce certainty about every future movement.

The connection to multiplayer prediction is analytical rather than identical implementation. One revises or anticipates shared state; another adjusts the view used for presentation. Both expose the possibility that information changes while computation is still being completed. Their success depends on when the response becomes useful and what errors remain possible.

The history has consequently widened the boundary of “device lag.” A local screen may be the place where a remote dependency, an old viewpoint, or a delayed decode becomes visible. The location of the symptom is not necessarily the location of its cause. Investigation has to follow the information, not stop at the object the user can touch.

# 13. What the history actually establishes

## Selected milestones, with the claims kept separate

A chronology is useful only when its entries describe comparable kinds of evidence. The following milestones identify a system, publication, or documented change. They are not a list of first inventions or a graph showing a universal decline in latency.

| Date | Record or development | Why it matters |
|---|---|---|
| 1956 | RAMAC. [1] | Access capability differs from access time. |
| 1961–1968 | CTSS, Sketchpad, NLS. [3, 4, 5] | Ongoing interaction and direct manipulation. |
| 1979 | Stella programming guide. [9] | Software meets television scan deadlines. |
| 1983 | NAMM MIDI demonstration, later institutional account. [68] | Connected instruments exchange performance commands. |
| 1984 | Nagle's RFC 896. [58] | Small-packet efficiency can involve deliberate waiting. |
| 1995–1998 | Wirth, Cheshire, RFC 2309. [12, 59, 60] | Software work, link capacity, and standing queues differ. |
| 1996 | QuakeWorld, retrospective account. [18] | Prediction provides feedback before confirmation. |
| 2011–2012 | RTP-MIDI; Android 4.1. [69, 29] | Timing semantics and coordinated presentation. |
| 2014–2015 | Chrome and WebKit fast tapping; touch study. [62, 63, 52] | Input rules and task-specific perception matter. |
| 2014 / 2021 | Google's HDR+ explanations. [64, 65] | Capture timing differs from image completion. |
| 2017 | HDMI 2.1 includes ALLM. [66] | A source can request a lower-latency display mode. |
| 2017–2019 | Historical-device and USB studies. [51, 53] | Configurations and endpoints qualify comparisons. |
| 2022–2024 | DLSS 3, DirectStorage, L4S, INP. [55, 34, 61, 50] | Different developments change different timing boundaries. |

These cases establish no single trajectory for lag. Designs can shorten actual work, remove waiting, prepare information earlier, predict a provisional result, or trade delay for continuity. A product can combine several of these changes.

**Conclusion.** Faster hardware is only one route to responsiveness. Input rules, camera acquisition, display processing, and musical timing can change when a result becomes useful. The recurring mistake is comparing different intervals as though they described the same experience. The repair lesson follows: identify the delayed event before choosing an intervention.

## From a complaint to a defensible result

Begin with an observation rather than a diagnosis: the pointer trails movement, a game pauses when entering a room, a call breaks up during uploads, or sound arrives after an action. A person may experience several symptoms at once. Recording them separately creates a better starting point than assuming they all share one cause.

| Observed problem | Useful comparison | Conclusion to avoid |
|---|---|---|
| Online play jumps; offline motion is smooth. | Fixed destination, idle versus loaded network. [20] | “The home router is definitely responsible.” |
| Motion stutters offline. | Same scene with time-ordered frame evidence. [35] | “Every hitch is shader compilation.” |
| An interface stops responding. | Trace whether relevant work is running, blocked, or waiting. [11, 47] | “Overall CPU use identifies the fault.” |
| Performance deteriorates during a session. | Comparable short and sustained workloads. [31] | “Heat is the only changed condition.” |
| Audio arrives late. | Same live path, supported wired versus wireless comparison. [41] | “Lip sync proves low live delay.” |
| TV controls feel late. | Same local scene, supported display-mode comparison. [67] | “The Internet is responsible.” |
| Camera misses the intended moment. | Separate acquisition time from finished-image readiness. [64, 65] | “Zero shutter lag means instant completion.” |

These are starting comparisons, not automatic diagnoses. More detailed instruments can help: Linux Pressure Stall Information reports time lost to CPU, memory, and input/output pressure, while Perfetto supports tracing across supported environments. Their observations still need interpretation. A trace is evidence about recorded activity, not a machine-generated verdict that every competing explanation has been eliminated. [46, 47]

Keep the method proportionate. Record the environment, software versions, workload, start and end events, instrument, repetitions, baseline, and intervention. Change one meaningful variable at a time where practical. Where safe, repeat the baseline afterward so elapsed time or a changing workload is less easily mistaken for the intervention's effect. This is the paper's proposed method, not an experiment already performed.

Preserve variation as well as an average. In a constructed set of 100 observations, ninety-nine results of ten milliseconds and one of 500 milliseconds produce a mean of 14.9 milliseconds. The average hides the half-second interruption. Report the time order and relevant distribution when available; comparisons of percentiles should state their conventions.

Supported troubleshooting also has boundaries. Google's Android guidance distinguishes checks of applications, storage, updates, and device-specific restart or recovery procedures. Destructive operations need particular care and data protection. A factory reset should not substitute for a simpler observation that can narrow the problem, and no universal speed goal justifies disabling thermal safeguards. [48, 31]

Define success before changing settings. Report improved, unchanged, worse, or inconclusive results. An unchanged result weakens an explanation only when the test could expose its expected effect. A clearer support report or avoiding an unnecessary purchase can also be useful; zero delay is not the required outcome.

# References

Stable source numbers link the narrative to this bibliography. Longer source-use and access notes are preserved in the matching Markdown edition. Historical records and later documentation have distinct evidentiary roles.

**[1] IBM (n.d.).** *RAMAC*.
https://www.ibm.com/history/ramac

**[2] John McCarthy (n.d.).** *Reminiscences on the History of Time Sharing*.
https://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html

**[3] David Walden and Tom Van Vleck, eds. (2011).** *Compatible Time-Sharing System (1961–1973): Fiftieth Anniversary Commemorative Overview*.
https://multicians.org/thvv/compatible-time-sharing-system.pdf

**[4] Ivan E. Sutherland (1963; electronic edition 2003).** *Sketchpad: A Man-Machine Graphical Communication System*.
https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf

**[5] Doug Engelbart Institute (n.d.).** *Firsts: The Demo*.
https://dougengelbart.org/content/view/209/

**[6] Jakob Nielsen (1993; updated 2014).** *Response Times: The 3 Important Limits*.
https://www.nngroup.com/articles/response-times-3-important-limits/

**[7] IBM (n.d.).** *The IBM PC*.
https://www.ibm.com/history/personal-computer

**[8] Microsoft (2021).** *Working Set*.
https://learn.microsoft.com/en-us/windows/win32/memory/working-set

**[9] Steve Wright / Atari (1979; later archival transcription).** *Stella Programmer's Guide*.
https://alienbill.com/2600/101/docs/stella.html

**[10] Microsoft (2025).** *Scheduling Priorities*.
https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling-priorities

**[11] Microsoft (Windows 7-era documentation).** *Preventing Hangs in Windows Applications*.
https://learn.microsoft.com/en-us/previous-versions/windows/win32/win7appqual/preventing-hangs-in-windows-applications

**[12] Niklaus Wirth (1995).** *A Plea for Lean Software*.
https://people.inf.ethz.ch/wirth/Articles/LeanSoftware.pdf

**[13] Jon Postel / RFC Editor (1981).** *RFC 792: Internet Control Message Protocol*.
https://www.rfc-editor.org/rfc/rfc792

**[14] IETF / RFC Editor (1999).** *RFC 2679: A One-way Delay Metric for IPPM*.
https://www.rfc-editor.org/rfc/rfc2679

**[15] IETF / RFC Editor (1999).** *RFC 2681: A Round-trip Delay Metric for IPPM*.
https://www.rfc-editor.org/rfc/rfc2681

**[16] Carlo Demichelis and Philip Chimento / RFC Editor (2002).** *RFC 3393: IP Packet Delay Variation Metric for IPPM*.
https://www.rfc-editor.org/rfc/rfc3393.txt

**[17] IETF / RFC Editor (2009).** *RFC 5681: TCP Congestion Control*.
https://www.rfc-editor.org/rfc/rfc5681

**[18] Glenn Fiedler (2010).** *What Every Programmer Needs To Know About Game Networking*.
https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/

# References — continued

**[19] GGPO project (n.d.).** *GGPO Rollback Networking SDK*.
https://www.ggpo.net/

**[20] IETF / RFC Editor (2015).** *RFC 7567: IETF Recommendations Regarding Active Queue Management*.
https://www.rfc-editor.org/rfc/rfc7567

**[21] RFC Editor (2018).** *RFC 8289: Controlled Delay Active Queue Management*.
https://www.rfc-editor.org/rfc/rfc8289

**[22] RFC Editor (2018).** *RFC 8290: The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm*.
https://www.rfc-editor.org/rfc/rfc8290.html

**[23] Cisco Meraki (n.d.).** *Channel Planning Best Practices*.
https://documentation.meraki.com/Wireless/Design_and_Configure/Architecture_and_Best_Practices/Channel_Planning_Best_Practices

**[24] ITU-R (2017).** *Report M.2410-0: Minimum Requirements Related to Technical Performance for IMT-2020 Radio Interface(s)*.
https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf

**[25] Seth Schneider / NVIDIA (2020).** *How To Reduce Lag: A Guide To Better System Latency*.
https://www.nvidia.com/en-us/geforce/guides/system-latency-optimization-guide/

**[26] NVIDIA (2020).** *Introducing NVIDIA Reflex: Optimize and Measure Latency in Competitive Games*.
https://www.nvidia.com/en-us/geforce/news/reflex-low-latency-platform/

**[27] Microsoft (n.d.).** *For Best Performance, Use DXGI Flip Model*.
https://learn.microsoft.com/en-us/windows/win32/direct3ddxgi/for-best-performance--use-dxgi-flip-model

**[28] Apple (9 January 2007).** *Apple Reinvents the Phone with iPhone*.
https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/

**[29] Android Developers (historical release documentation).** *Jelly Bean*.
https://developer.android.com/about/versions/jelly-bean

**[30] Android Developers (n.d.).** *Slow Rendering*.
https://developer.android.com/topic/performance/vitals/render

**[31] Android Developers (updated 2026).** *Thermal API*.
https://developer.android.com/games/optimize/adpf/thermal

**[32] Apple Support (1 June 2026).** *iPhone Battery and Performance*.
https://support.apple.com/en-us/101575

**[33] Apple Support (n.d.).** *If Your iPhone or iPad Is Running Slow*.
https://support.apple.com/en-us/102598

**[34] Cassie Hoef and Damyan Pepper / Microsoft (7 November 2022).** *DirectStorage 1.1 Now Available*.
https://devblogs.microsoft.com/directx/directstorage-1-1-now-available/

**[35] Epic Games (n.d.).** *PSO Precaching*.
https://dev.epicgames.com/documentation/en-us/unreal-engine/pso-precaching-for-unreal-engine

**[36] Paul Lewis / Google web.dev (updated 2023).** *Rendering Performance*.
https://web.dev/articles/rendering-performance

# References — continued

**[37] Jeremy Wagner and Brendan Kenny / Google web.dev (2022; updated 2024).** *Optimize Long Tasks*.
https://web.dev/articles/optimize-long-tasks

**[38] Google web.dev (n.d.).** *Interaction to Next Paint (INP)*.
https://web.dev/articles/inp

**[39] IETF / RFC Editor (2021).** *RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport*.
https://www.rfc-editor.org/rfc/rfc9000

**[40] IETF / RFC Editor (2022).** *RFC 9114: HTTP/3*.
https://www.rfc-editor.org/rfc/rfc9114.html

**[41] Android Developers (updated 2024).** *Audio Latency*.
https://developer.android.com/ndk/guides/audio/audio-latency

**[42] Ableton (n.d.).** *How to Reduce Latency*.
https://help.ableton.com/hc/en-us/articles/209072289-How-to-reduce-latency

**[43] Bluetooth SIG (n.d.).** *LE Audio*.
https://www.bluetooth.com/learn-about-bluetooth/feature-enhancements/le-audio/

**[44] NVIDIA (current page checked 2026).** *GeForce NOW System Requirements*.
https://www.nvidia.com/en-us/geforce-now/system-reqs/

**[45] Microsoft (n.d.).** *Hologram Stability*.
https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/hologram-stability

**[46] Johannes Weiner / Linux kernel documentation (2018; maintained documentation).** *PSI: Pressure Stall Information*.
https://docs.kernel.org/accounting/psi.html

**[47] Perfetto project (n.d.).** *What Is Perfetto?*.
https://perfetto.dev/docs/

**[48] Google Android Help (n.d.).** *Speed Up a Slow Android Device*.
https://support.google.com/android/answer/7667018?hl=en

**[49] IBM (n.d.).** *What Is a Solid-State Drive?*.
https://www.ibm.com/think/topics/solid-state-drives

**[50] Jeremy Wagner and Rick Viscomi / Google web.dev (2024).** *Interaction to Next Paint Becomes a Core Web Vital on March 12*.
https://web.dev/blog/inp-cwv-march-12

**[51] Dan Luu (2017).** *Computer latency: 1977–2017*.
https://danluu.com/input-lag/

**[52] Jonathan Deber, Ricardo Jota, Clifton Forlines, and Daniel Wigdor (2015).** *How Much Faster is Fast Enough? User Perception of Latency & Latency Improvements in Direct and Indirect Touch*. CHI 2015, 1827–1836. DOI: 10.1145/2702123.2702300.
https://www.tactuallabs.com/papers/howMuchFasterIsFastEnoughCHI15.pdf

**[53] Raphael Wimmer, Andreas Schmid, and Florian Bockes (2019).** *On the Latency of USB-Connected Input Devices*. CHI 2019. DOI: 10.1145/3290605.3300650.
https://www.researchgate.net/publication/332745060_On_the_Latency_of_USB-Connected_Input_Devices

**[54] Jeffrey Dean and Luiz André Barroso (2013).** *The Tail at Scale*. Communications of the ACM 56(2): 74–80. DOI: 10.1145/2408776.2408794.
https://research.google/pubs/the-tail-at-scale/

# References — continued

**[55] Henry Lin and Andrew Burnes / NVIDIA (20 September 2022).** *NVIDIA DLSS 3: AI-Powered Performance Multiplier Boosts Frame Rates By Up To 4X*.
https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/

**[56] Guy Almes, Sunil Kalidindi, Matt Zekauskas, and Al Morton, ed. (January 2016).** *RFC 7679: A One-Way Delay Metric for IP Performance Metrics (IPPM)*. DOI: 10.17487/RFC7679.
https://www.rfc-editor.org/info/rfc7679/

**[57] W3C (19 March 2026).** *Event Timing API*. Working Draft.
https://www.w3.org/TR/2026/WD-event-timing-20260319/

**[58] John Nagle (6 January 1984).** *RFC 896: Congestion Control in IP/TCP Internetworks*.
https://www.rfc-editor.org/rfc/rfc896

**[59] Stuart Cheshire (begun May 1996; subsequently revised).** *It's the Latency, Stupid*.
https://www.stuartcheshire.org/rants/Latency.html

**[60] Bob Braden and colleagues / RFC Editor (April 1998).** *RFC 2309: Recommendations on Queue Management and Congestion Avoidance in the Internet*.
https://www.rfc-editor.org/rfc/rfc2309

**[61] IETF / RFC Editor (January 2023).** *RFC 9330: Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture*.
https://www.rfc-editor.org/rfc/rfc9330

**[62] Jake Archibald / Chrome for Developers (revised page; mixed date metadata).** *300ms tap delay, gone away*.
https://developer.chrome.com/blog/300ms-tap-delay-gone-away/

**[63] Wenson Hsieh / WebKit (15 December 2015).** *More Responsive Tapping on iOS*.
https://webkit.org/blog/5610/more-responsive-tapping-on-ios/

**[64] Marc Levoy / Google (27 October 2014).** *HDR+: Low Light and High Dynamic Range photography in the Google Camera App*.
https://research.google/blog/hdr-low-light-and-high-dynamic-range-photography-in-the-google-camera-app/

**[65] Manfred Ernst and Bartlomiej Wronski / Google Research (23 April 2021).** *HDR+ with Bracketing on Pixel Phones*.
https://research.google/blog/hdr-with-bracketing-on-pixel-phones/

**[66] HDMI Forum (28 November 2017).** *HDMI Forum Releases Version 2.1 of the HDMI Specification*.
https://www.hdmi.org/download/pressfileid/60

**[67] HDMI Licensing Administrator (n.d.).** *Auto Low Latency Mode (ALLM)*.
https://www.hdmi.org/spec21sub/autolowlatencymode

**[68] Athan Billias / MIDI Association (maintained institutional history).** *MIDI History Chapter 6: MIDI Begins 1981–1983*.
https://midi.org/midi-history-chapter-6-midi-begins-1981-1983

**[69] John Lazzaro and John Wawrzynek / RFC Editor (June 2011).** *RFC 6295: RTP Payload Format for MIDI*.
https://www.rfc-editor.org/rfc/rfc6295

**[70] VALORANT Gameplay Technology team / Riot Games (24 May 2022).** *VALORANT Gameplay Consistency Update 2*.
https://playvalorant.com/en-us/news/game-updates/valorant-gameplay-consistency-update-2/