Defining 'Real-Time'

A toolkit for assessing data portability under the DMA and digital competition laws

Thomas Carey-Wilson
The Open Data Institute[1]
October 2025

Executive summary

The Digital Markets Act (DMA) and similar regulations globally mandate 'continuous and real-time' data portability to dismantle data-driven barriers to entry and lower switching costs. However, the lack of a precise definition for this requirement creates a critical ambiguity that undermines its pro-competitive potential, leaving regulators without a clear compliance benchmark and platforms without clear guidance. This report addresses this enforcement gap.

To deliver this clarity, we introduce a new taxonomy for classifying data transfer systems based on a review of **18** different implementations across 3 sectors: health, finance, and social media. Our central finding is that "real-time" is not an absolute measure but is context-dependent. We distinguish between "Absolute Real-Time" (the theoretical instant an event occurs) and "Functional Real-Time" (the point beyond which further speed provides no meaningful benefit to the user). Analysis shows that latency is often dictated not by the intrinsic cadence of the data source (eg, physiological processes in glucose monitors) or by deliberate architectural choices that trade speed for security, analytical value, or system stability.

Ultimately, this research provides policymakers and enforcers with a practical toolkit to move beyond absolutist views of speed. The taxonomy and FRT framework enable a nuanced evaluation of compliance, ensuring that data portability remedies are not just fast, but effective, pro-competitive, and fit-for-purpose.

[1] This work was supported by the Data Transfer Initiative.

Table of Contents

1. The policy and economic context behind real-time data portability....3
1.1 The economic costs of data silos: vendor lock-in and switching costs....3
1.2 Data portability as a potential pro-competitive policy remedy....4
1.3 The transition from episodic to continuous and real-time portability in policy....5
2. A taxonomy of continuous and real-time data transfers....5
2.1 Key definitions: what do we mean by “continuous” and “real-time”?....6
2.2 The taxonomy: how can we classify continuous and real-time data transfers?....6
2.3 Reflections on real-time: What do we mean?....12
3. Findings: real-time in practice....14
3.1 Use case: blood glucose monitoring....14
3.2 Use case: heart rate monitoring....15
3.3 Use case: budgeting apps....16
3.4 Use case: brokerage apps....17
3.5 Use case: messaging & social media....18
3.6 Use case: mail....19
4. Discussion: factors affecting real-time....21
4.1 Intermediary processes....21
4.2 Intrinsic cadence....22
4.3 Interoperable standards....23
5. Concluding remarks....24
5.1 Implications for competition policy and enforcement....25
Annex A. Methodology....26
Use case selection....26
Functional Real-Time (FRT) framework....27
Annex B. Taxonomy placement....31

1. The policy and economic context behind real-time data portability

The principle of data portability (an individual's right to obtain and reuse their data across different services) has emerged as a critical instrument for driving competition across the digital economy and is now a central obligation for 'gatekeeper' platforms under the EU's Digital Markets Act (DMA). Here, we trace its evolution from a nascent right into a more sophisticated policy objective mandating continuous, real-time transfers.

1.1 The economic costs of data silos: vendor lock-in and switching costs

In digital markets, the absence of effective data portability can give rise to economic friction in the form of vendor lock-in and high switching costs.[2] Vendor lock-in describes where a customer is unable to switch from one provider to another without incurring substantial costs.[3] These costs include the loss of historical data, as well as the forfeiture of established digital identities and social networks, and even the effort required to re-learn workflows on a new platform. Some studies suggest this dependency limits consumer choice and innovation by smaller market players, ultimately leading to anti-competitive market structures.[4] [5]

Yet, this potential issue can extend beyond consumers and small businesses, creating significant burdens for *larger* organisations. Evidence from major buyers shows that large and complex (particularly IT) estates become path-dependent and costly to change. The UK CMA's cloud market investigation identifies factors like egress fees, technical incompatibilities, and licensing restrictions[6] as material barriers to switching services for large firms.[7] Some regulators are treating this issue as systemic. For instance, the EU Data Act mandates the elimination of cloud switching and data-egress charges by 12 January 2027 for this reason.[8]

In the digital economy, data itself has become a primary source of lock-in.[9] The cumulative value of a user's data on a platform (be it a social network's friend connections, an e-commerce site's purchase history, or a cloud provider's stored files) creates a powerful disincentive to migrate to new services. This "data gravity"[10] effect risks trapping consumers and their data within "walled gardens", reinforcing the market power of incumbents and

---

2 https://static.aminer.org/pdf/PDF/000/326/425/information_technology_innovation_and_competition_in_the_presence_of_switching.pdf

3 https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-016-0054-z

4 https://one.oecd.org/document/DAF/COMP(2022)5/en/pdf

5 https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3083114

6 https://assets.publishing.service.gov.uk/media/67989251419bdbc8514fdee4/summary_of_provisional_decision.pdf

7

8 https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ%3AL_202302854

9 https://www.tandfonline.com/doi/full/10.1080/07370024.2024.2325347

10 https://www.techtarget.com/whatis/definition/data-gravity

raising barriers to entry for new competitors.[11]

1.2 Data portability as a potential pro-competitive policy remedy

Data portability is a direct policy response to the economic harms of vendor lock-in. The right to data portability means regulators aim to grant consumers the right to take their data with them across services. In turn, reducing the aforementioned switching costs, lock-in and dismantling the data-driven barriers that protect some incumbents from competitive pressure.[12] [13] Specifically, when users can easily move their data to a rival service, they are more likely to switch in response to differentiating factors like better prices or improved service quality. This threat of customer churn can, in theory, encourage platforms to compete more vigorously on the merits of their offerings.

Underlying policy, the economic dynamics are complex. An empirical study by Jeon & Menicucci (2023), using a two-period competition model, found that under certain conditions, data portability can simultaneously increase both consumer surplus *and* firm profits.[14] This suggests a potential win-win scenario where enhanced competition does not come at the expense of industry viability. However, other research cautions that data portability may not generally boost profits or welfare. Lam & Liu (2019) use a two-period entry model to show that portability has the desired "switch-facilitating" effect (ie, it lowers switching costs), but also may drive a "demand-expansion" effect (ie, users provide more data to incumbents when they know it can be ported).[15] Where a firm hosts a strong network effect, data portability may lead to more user data going their way, strengthening the incumbent's advantage, and thus the demand-expansion effect dominates.

However, Lam & Liu (2019) also point out that this demand-expansion effect can be driven by poorly designed policy that is: i) too broad in scope (ie, covering data with high marginal value to the incumbent but low relevance for switching); or, ii) delayed or subject to frictions (eg, non-real-time transfers) weaken the switch-facilitating effect, but still leave the demand-expansion effect intact. As such, the pro-competitive effect of data portability is not guaranteed. It also depends on the specifics of its implementation. Policy design flaws flagged by Reimsbach-Kounatze & Molnar (2024)[16] concur with Lam & Liu (2019: (i) scope drift (ie, lack of clarity around "what and whose data"); (ii) cumbersome modalities (eg, one-off exports and unstable APIs, among others); and (iii) expanded security attack surfaces from duplicated flows. In particular, ambiguity as to what real-time means recurs

---

[11] https://journals.sagepub.com/doi/10.1177/1024529418816525

[12] https://jolt.law.harvard.edu/assets/articlePDFs/v36/Zhang-The-Paradox-of-Data-Portability-and-Lock-In-Effects.pdf

[13] https://www.oecd.org/en/publications/the-impact-of-data-portability-on-user-empowerment-innovation-and-competition_319f420f-en.html

[14] https://www.tse-fr.eu/publications/data-portability-and-competition-can-data-portability-increase-both-consumer-surplus-and-profits

[15] https://publications.aston.ac.uk/id/eprint/41090/1/Does_data_portability_facilitate_entry.pdf

[16] https://ideas.repec.org/p/oec/stiaad/25-en.html

across work on this topic[17] [18] [19] and is a primary reason portability can underperform.

1.3 The transition from episodic to continuous and real-time portability in policy

Understanding of data portability has evolved, moving through generations of implementation. The first generation, or "Data Portability 1.0," was enshrined in Article 20 of the European Union's General Data Protection Regulation (GDPR). This landmark provision granted individuals the right to receive their data in a "structured, commonly used and machine-readable format" and to have it transmitted to another service provider.[20] While important, the GDPR's framework was hampered by practical limitations. For example, data controllers were given up to 30 days to respond to requests, which often resulted in one-off, static data dumps. This long lead time rendered the data obsolete for many dynamic use cases.[21]

These shortcomings have led to "Data Portability 3.0," a new focus on continuous, real-time, API-based data flows. This vision is articulated in Article 6(9) of the DMA, which mandates that designated "gatekeeper" platforms provide "continuous and real-time access" to data. This requirement is what gives the DMA's portability provisions their pro-competitive teeth, with continuous and real-time access being valuable here to make data portability requirements effective in fast-moving digital markets.[22] Therefore, understanding what continuous and real-time data transfers are and what it means for them to be real-time is an important question for both digital competition policy *and* the subjects of that policy.

2. A taxonomy of continuous and real-time data transfers

Some prior work have developed taxonomies focused on the strategic business value of real-time data initiatives[23], or creating highly technical, machine-readable data models for IoT

---

17 https://op.europa.eu/en/publication-detail/-/publication/21dc175c-7b76-11e9-9f05-01aa75ed71a1/language-en

18 https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3862299

19 https://www.oecd.org/en/publications/the-impact-of-data-portability-on-user-empowerment-innovation-and-competition_319f420f-en.html

20 https://www.researchgate.net/publication/351070025_Data_Portability_between_Online_Services_An_Empirical_Analysis_on_the_Effectiveness_of_GDPR_Art_20

21 https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3866495

22 https://cerre.eu/wp-content/uploads/2022/11/DMA_DataAccessProvisions-2.pdf#:~:text=of%20such%20data%20portability%2C%20and,an%20effective%20instrument%20to%20spur

23 https://management-datascience.org/articles/9967/

environments.[24] Crucially, regulations like the DMA do not define “continuous” or “real-time”, and no general taxonomy of continuous, real-time data transfers yet exists.[25] [26] [27]

The taxonomy presented here is explicitly designed for a non-technical audience (policymakers and consumer advocates). Rather than classifying business outcomes or low-level data structures, it categorises the observable mechanics of data-transfer systems to establish a common practical language for how these services work.

This paper contributes to the field not only through its findings but also through its methodology. We provide a new, replicable method for analysing the performance of data portability remedies by developing a structured taxonomy and an FRT framework from observational data of 18 in-market systems. This is a critical need as regulators worldwide implement and enforce these rules.

2.1 Key definitions: what do we mean by “continuous” and “real-time”?

Data transfer is generally defined as the movement (copying, streaming or migrating) of digital information from one point, system or location to another.[28] [29] To understand the benefits of *real-time* and *continuous* data transfers as a key part of data portability 3.0, it is first important to understand what we mean by these qualifiers. NIST’s glossary frames real-time as a latency threshold dictated by the external event, not by the network alone.[30]

2.2 The taxonomy: how can we classify continuous and real-time data transfers?

Defining ‘continuous’ and ‘real-time’ in practice is challenging due to diverse implementations. We reviewed 18 implementations across six use cases and developed a framework to classify the different approaches. **Table 1** summarises the selected use cases and implementations (see **Annex A** for methodology).

Table 1. Use case areas, use cases and implementations

Use case area	Use case	Implementation
	a) "A user wants to see their current glucose level right away and also review their broader trends automatically throughout the day."	i) Dexcom G7 (Dexcom API V3) [31] [32]

[24] https://www.sciencedirect.com/science/article/pii/S254266052400101X
[25] https://www.tu-ilmenau.de/fileadmin/Bereiche/WM/wth/Diskussionspapier_Nr_192.pdf
[26] https://dtinit.org/assets/DTI-Data-Portability-Compendium.pdf
[27] https://cerre.eu/wp-content/uploads/2023/03/230327_Data-Act-Book.pdf
[28] https://www.itu.int/dms_pubrec/itu-r/rec/v/R-REC-V.662-2-199304-S%21%21PDF-E.pdf
[29] https://csrc.nist.gov/glossary/term/data_transfer_solution
[30] https://csrc.nist.gov/glossary/term/real_time
[31] https://s3.us-west-2.amazonaws.com/dexcompdf/G7/AW00046-05_UG_G7_OUS_en_MMOL.pdf
[32] https://developer.dexcom.com/docs/dexcomv3/endpoint-overview/

1) Iot Wearables		ii) FreeStyle Libre 3 Sensor[33]
		iii) Medtronic Guardian Connect[34]
	b) "A user wants to watch their heart rate in real time during exercise and check later how their body recovered afterward."	i) Polar H10 (via BLE)[35]
		ii) Garmin The Vivosmart 5 API[36]
		iii) Fitbit (Charge 5/Sense) API[37]
2) Personal Banking & Finance	a) "A user wants their budgeting app to update transactions immediately after they happen, no matter which bank they used."	i) Plaid Webhooks[38]
		ii) Yodlee Webhooks[39]
		iii) TrueLayer[40]
	b) "A user wants one platform to give live stock prices and let them trade automatically, even if they use different brokerages."	i) Charles Schwab Trader API[41]
		ii) Alpaca Markets API[42]
		iii) Tradier API[43]
3) Messaging & Social Media	a) "A user wants to receive alerts instantly when they're mentioned in a message or post, no matter where it's posted."	i) Discord API[44]
		ii) X Filtered Stream API[45]
		iii) Slack Events API[46]
	b) "A user wants to manage their email from multiple accounts (eg, work, personal) in a single application, with new messages, event invitations, and updates appearing instantly, regardless of which account they are sent to."	i) Fastmail (JMAP)[47]
		ii) Dovecot (IMAP)[48]
		iii) Gmail (Google Workspace

[33] https://freestyleserver.com/payloads/ifu/2023/q3/ART41641-001_rev-A-web.pdf

[34] https://resources.cloud.medtronic-diabetes.com/sites/prd/files/documents/2022-04/guardian_connect_-_getting_started_guide.pdf

[35] https://bjsm.bmj.com/content/50/7/441

[36] https://developer.garmin.com/gc-developer-program/health-api/#:~:text=Health%20API%20Features

[37] https://dev.fitbit.com/build/reference/web-api/intraday/

[38] https://plaid.com/docs/identity-verification/webhooks/

[39] https://developer.yodlee.com/resources/yodlee/webhooks/docs/overview

[40] https://developer.yodlee.com/resources/yodlee/json-web-token-authorization/docs#:~:text=You%20use%20an%20application%20token,header%20for%20the%20RESTful%20APIs

[41] https://developer.schwab.com/user-guides/get-started/introduction

[42] https://docs.alpaca.markets/docs/streaming-market-data#:~:text=Connection

[43] https://documentation.tradier.com/brokerage-api/streaming

[44] https://github.com/meew0/discord-api-docs-1/blob/master/docs/topics/GATEWAY.md

[45] https://docs.x.com/x-api/posts/filtered-stream/introduction

[46] https://docs.slack.dev/apis/events-api/

[47] https://www.fastmail.help/hc/en-us/articles/1500000278382-Email-standards

[48] https://doc.dovecot.org/2.4.1/

		API)[49] [50]

Examining each implementation's technical documentation (referenced in **Table 1**.) revealed several key factors that affect how 'real-time' a transfer can be. We describe these factors in the taxonomy below, and **Table 2** in **Annex A** maps each implementation to the taxonomy dimensions.

2.2.1 Where is data sharing initiated by the user?

2.2.1.1 Initiated from the source platform

In this model, the user initiates the data sharing process from within the source platform's own application or website. The user navigates through the source platform's settings or sharing menus to select a third party and authorise the connection. This flow keeps the user within a familiar environment, giving the source platform significant control over the user experience and the presentation of consent.

2.2.1.2 Initiated from the third-party application

Here, the data sharing journey begins within the third-party (recipient) application. The user, seeking to import data, is prompted by the third-party app to connect to an external source. This typically triggers a redirect (eg, via an OAuth flow) to the source platform for authentication and consent before returning the user to the third-party app. This model is common for data aggregators and services that integrate with multiple sources.

2.2.1.3 Initiated from on-device pairing

In this model, the initial connection is established through a direct, localised action between a physical device and a user's smartphone or hub. This often involves Bluetooth pairing or a similar proximity-based protocol. While subsequent data sharing to the cloud or other apps still requires further authorisation, the foundational link is created on-device, independent of a web-based flow.

2.2.2 What is the data pathway?

2.2.2.1 On-device hub

In an on-device hub setup, the user's phone acts as the relay. The user controls when data uploads occur (eg, by opening the app or enabling Bluetooth), so update speed hinges on the phone's connectivity and power. If the phone is offline or the app is closed, data queues locally and uploads later, meaning updates are only as timely as the phone's connection permits

2.2.2.2 Direct connection

Data moves from the source's cloud servers directly to the third party's cloud servers. Once authorisation is granted, the service streams data or alerts as soon as it's available. Because

[49] https://developers.google.com/workspace/gmail/api/guides/push
[50] https://cloud.google.com/pubsub/docs/push

the data bypasses a user's device, updates can be very timely; market feeds and messaging events arrive within seconds or milliseconds of occurrence. However, sustained real-time performance depends on an efficient mechanism for notifying clients of new data. While this is often achieved with a persistent connection (eg, an open WebSocket[51] [52]), it can also be accomplished without a continuously open connection through event-driven protocols like webhooks, which avoids the resource intensity of constant polling.

2.2.2.3 Intermediary hub

Data moves through a central intermediary that can read and process the raw data. Intermediary hubs sit between the original data source (such as a bank) and your application. They aggregate data from many providers and then notify integrators via webhooks when something changes. This model offloads the complexity of polling multiple upstream sources but introduces additional latency; the intermediary must first collect data (often through scheduled polling) before pushing a notification, so updates may be "near-real-time" rather than instantaneous.

2.2.3 Whose credentials?

2.2.3.1 User credentials only

The user provides their source platform login credentials (eg, username and password) directly to the third-party application or an intermediary. The third party then uses these credentials to log in on the user's behalf, and often gathers data through methods like screen-scraping. This model places the burden of trust on the user's willingness to share their information. A better approach that still only relies on user credentials is to ask the user to generate a secret key based on their credentials, which the user can then share with the third party without sharing their password (eg, such as is the case with GitHub[53]). This is often difficult UX[54] but can work.

2.2.3.2 User and vetted 3rd party credentials

In this model, the source platform identifies a pre-registered third-party application by validating its unique credentials (eg, a Client ID and Client Secret) during an API-driven authentication flow, typically OAuth 2.0.[55] The user also authenticates directly with the source platform, which then issues a temporary, revocable access token that is linked to both user and 3rd party credentials. In some implementations of this, the 3rd party must also go through vetting processes that can examine data security practices, privacy policies, or more.

2.2.3.3 On-device pairing - proximity

Instead of using credentials, data access is established through a secure, proximity-based pairing process between a hardware device and a controlling application (eg, a smartphone

---

[51] A WebSocket is a communication protocol that allows for real-time, two-way interaction between a client (like a web browser) and a server over a single, long-lasting connection.
[52] https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
[53] https://docs.github.com/en/actions/how-tos/write-workflows/choose-what-workflows-do/use-secrets
[54] https://dev.to/msnmongare/how-to-add-github-secrets-easily-step-by-step-guide-3cmh
[55] https://oauth.net/2/

app), initiated by the user. The smartphone platform using a protocol like Bluetooth Low Energy does not allow any random nearby device to connect, but requires user approval as well as proximity. This initial trust relationship is foundational and confined to the local device ecosystem.

2.2.4 How is user consent obtained?

2.2.4.1 Explicit consent via redirect

The user grants consent through a dedicated, standardised interface, typically after being redirected from the third-party app to the source platform's domain. This flow presents the user with a "consent screen" that explicitly lists the requesting application, the data being requested, and the permissions required. The user must take an affirmative action, such as clicking an "Authorise" or "Allow" button, to grant consent and be redirected back.

2.2.4.2 Implicit consent via in-app invitation

Consent is granted through a series of actions within the source or recipient application that do not involve a standardised redirect to a consent screen. For example, a user might enter a recipient's email address within the source app to send a sharing invitation. The recipient then accepts this invitation in their own app or via email. Consent is implied by the user's deliberate actions to initiate and accept the sharing link, rather than by a single "Authorise" click on a dedicated screen.

2.2.5 What is the duration of the permission?

2.2.5.1 Time-bound

With time-bound permissions, the user’s authorisation expires after a set period. Data flows continuously for the permitted duration, but the stream will stop unless the credential is refreshed or a sensor is replaced. This introduces maintenance overhead and the risk of downtime if a renewal is missed. For example, a continuous glucose monitor only delivers real-time data until the sensor expires, and an API integration (eg, TrueLayer or Schwab) requires periodic token refreshes to keep the feed live.

2.2.5.2 Open-ended

Open-ended permissions mean that once users approve access, the service can stream data indefinitely (subject to user revocation). For real-time integrations (whether CGMs like Dexcom, messaging APIs like Slack, or trading feeds like Alpaca), this eliminates token-refresh overhead and reduces the risk of missed updates. Developers still need to manage key security, but the data flow itself can remain continuous and uninterrupted for months or years.

2.2.6 How is new data delivered?

2.2.6.1 Pull

In a pull model, the client periodically asks the server for updates. This can be done on a recurring schedule or using more efficient methods like long polling. This gives developers

full control over when data is retrieved, but it also means there can be gaps between the moment data is available and when it's actually fetched. Frequent polling can approximate "real-time," but doing so increases network usage and power consumption.

2.2.6.2 Push

In pure push systems, data is delivered automatically as soon as it's generated. This can be in the form of discrete data packages (eg, a single new post) or as a continuous live stream of data over a persistent connection. This produces very fresh information with minimal delay, ideal for alerts or market feeds. However, it depends on maintaining an open connection (WebSocket or BLE link), and interruptions in connectivity will pause the stream until reconnected.

2.2.6.3 Hybrid

Hybrid systems combine push and pull methods. In most cases, we examined, the service sends a quick notification (push) to alert clients that new data is available, and the client then retrieves the full data (pull). This reduces unnecessary polling while still giving timely alerts. Either way, there can be a small delay between the initial data availability and when a client obtains the full data, but overall latency remains low and network overhead is reduced compared with constant polling.

2.2.7 Can existing data be changed?

2.2.7.1 Immutable (append-only)

For systems where data is append-only, new records are added, but old ones cannot be changed or deleted. New readings or events simply add to a growing timeline, and there is no need to handle corrections or deletions. This simplifies real-time processing because listeners can process incoming data immediately and trust that history won't change. However, if a sensor or market feed produces an erroneous value, consumers have to correct it themselves since the source never retracts or edits past data.

2.2.7.2 Mutable (append with changes)

In mutable systems, a message, transaction or event can later be edited, deleted or have its status updated. New records are added, and existing ones can be updated or deleted. Real-time consumers need to handle correction and deletion messages and reconcile updates with previously stored data. This introduces extra complexity as clients must maintain a local cache and listen for "change" notifications to keep it accurate. While updates ensure accuracy, they can also create brief discrepancies between what was seen in real time and the corrected record.

2.2.8 How frequent is the data?

2.2.8.1 Live

Live systems deliver data almost instantly as it's generated. Market feeds and sensor streams fall into this category; once a WebSocket or BLE connection is open, every new heartbeat or trade is pushed down the wire with minimal buffering. This enables true

real-time applications (there's virtually no delay), but this depends on maintaining a persistent connection and keeping devices powered and connected.

2.2.8.2 Near-Live

Near-live systems provide reasonably timely data, often every minute or every few minutes. Physiological sensors like Libre, Guardian and Lingo fall into this bucket; they send readings at one- or five-minute intervals. Many webhook-based services alert clients shortly after a change occurs, but a brief polling or processing delay means these updates are near-real-time rather than truly instantaneous. In practice, this cadence is sufficient for most applications despite not being absolutely live.

2.2.8.3 Periodic

In periodic systems, new data doesn't arrive continuously but in batches after a fixed delay. For example, Dexcom's three-hour delay means that software relying on the public API sees glucose values hours after they're recorded. This model may be acceptable for retrospective analysis or low-urgency tasks, but it precludes real-time monitoring.

All of these dimensions raise a fundamental question: what does it mean for a data transfer to be 'real-time' in practice? NIST's definition gives a clue by tying real-time to the timing of the external event (not just network speed), but it doesn't tell us how to determine if a given solution meets the real-time bar.

2.3 Reflections on real-time: What do we mean?

As discussed, this question of what constitutes "continuous and real-time" data access creates significant challenges for both policy implementation and compliance. Namely, it leaves both regulators and regulated entities without a clear benchmark for what constitutes an acceptable level of performance. This section addresses this definitional problem directly, arguing that a context-aware framework is essential for bringing clarity to the concept of real-time.

2.3.1 Absolute vs functional real-time

A continuous and real-time data transfer is not cannot be real-time in an absolute sense. Its real-time-ness is a function of latency relative to a specific context. Here we can distinguish between two key concepts:

- **Absolute real-time (ART):** The theoretical, instantaneous moment an event occurs. This is a physical limit that is impossible to achieve in practice.
- **Functional real-time (FRT):** The point of diminishing returns, beyond which further reductions in latency are no longer perceptible or meaningful to the user for a given task. For instance, a video call that already feels instantaneous may not benefit from further speed improvements.

At the most trivial level, signals cannot arrive before they are sent. Even in fibre-optic cables, light pulses travel through the fibre-optic medium much more slowly than through a

vacuum.[56] Physical limits (like the speed of light through fibre) impose a floor on latency, which is a propagation delay that cannot be eliminated.[57] Once such limits dominate the total delay, improving processor speeds or protocols yields only marginal gains in end-to-end latency. This aligns with Amdahl's Law: the speed-up from any improvement is limited by the portion of the process that can't be accelerated.[58] In other words, if an event itself only occurs every 5-10 minutes, no technology can produce the data faster than (or at the exact same time as) that natural 5-10 minute cycle, at best, the system can only approach this limit.

Together, this suggests that as you approach the moment of the event, each improvement shaves off smaller and smaller slices of latency because the remaining delay is dominated by unoptimisable components (eg, the time it takes light or glucose molecules to travel in related events). Therefore, the curve for improving the performance (ie, lowering the latency) of continuous and real-time data transfers bends towards, but never crosses, the ART boundary. But, it may cross the FRT boundary. That is, if the remaining delay falls below the threshold of human perceptual limits (or their expectations about the rate at which an event happens), additional speed adds little or no practical benefit for the experience of an end user.

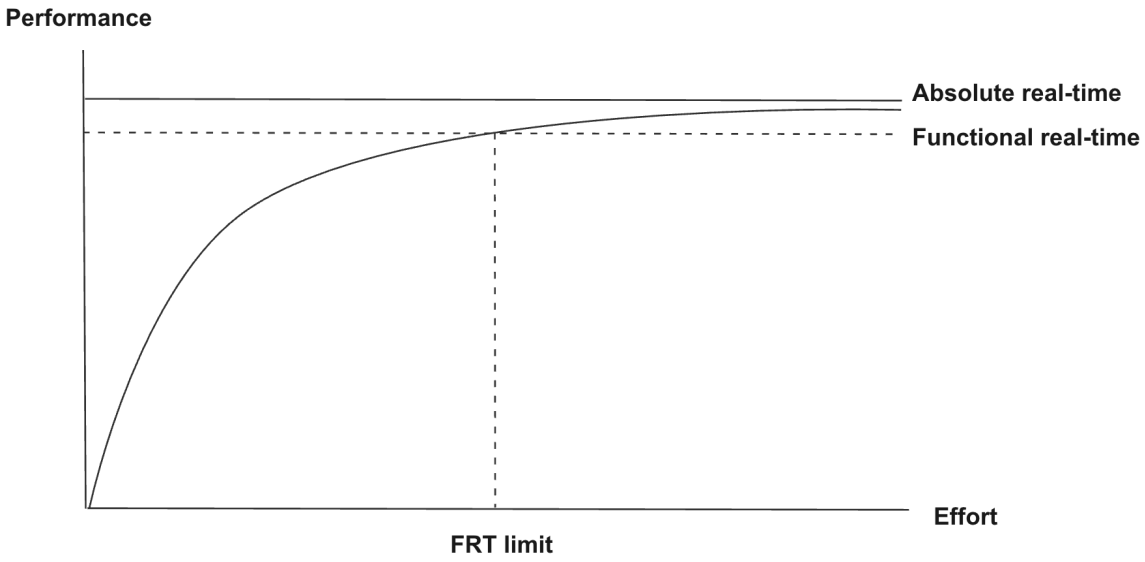
Putting these concepts together, we can compile the graph shown in **Figure 1**. This illustrates the conceptual relationship between latency and the effort required to improve it:

- The vertical Y-axis represents **Performance** in terms of *lower latency*. The top of the axis is zero latency (instantaneous), with delay increasing towards the bottom. The scale is presented conceptually like a logarithmic scale, which allows for better visual distinction between very fast response times (eg, 20 milliseconds vs. 500 milliseconds) that would otherwise be clustered together at the top of a linear scale.
- The horizontal X-axis represents **Effort**. This can be understood as the cumulative investment (eg, engineering time, financial cost, architectural complexity) applied to reduce latency.

The **curve's** shape illustrates the principle of diminishing returns, consistent with Amdahl's Law. Initial improvement efforts yield significant reductions in performance, but as the system gets faster, the same amount of effort produces smaller and smaller gains as it approaches fundamental physical or intrinsic limits.

Figure 1. FRT Framework

[56] https://www.sciencedirect.com/science/article/pii/S2666285X22000280
[57] https://mapyourtech.com/latency-in-optical-networks-principles-optimization-and-applications/
[58] https://www.staff.ncl.ac.uk/rishad.shafik/files/2020/01/IET-Amdahl-Review.pdf



3. Findings: real-time in practice

We surveyed public documentation (and developer discussions, where needed) for a range of digital services and plotted their end-to-end propagation delays on our **Figure 1** curve. From financial APIs to wearable health sensors, this lets us see what a practical functional real-time benchmark looks like across sectors.

Also, we focus on evidence for the best-case scenario, end-to-end propagation delay, assuming user data is being shared with a third party for some purpose. This is to ensure that our findings are relevant for prospective regulatory compliance purposes (eg, a private company sharing a continuous and real-time stream of *x* data for *y* purpose.)

Lastly, we focused on each system's best-case end-to-end delay (ignoring outliers and assuming optimal conditions) to align with potential regulatory compliance scenarios. Where exact data was unavailable, we chose an estimate. We ensure that even the lowest estimate puts this implementation on one side of the FRT threshold, ensuring our comparisons are meaningful. (*See* **Annex A.** *for details on determining the FRT level in each case.*)

3.1 Use case: blood glucose monitoring

In this use case, continuous glucose monitoring systems are inherently limited by a physiological delay of at least **5 minutes** between blood glucose changes and interstitial fluid measurements. We set the FRT level accordingly.

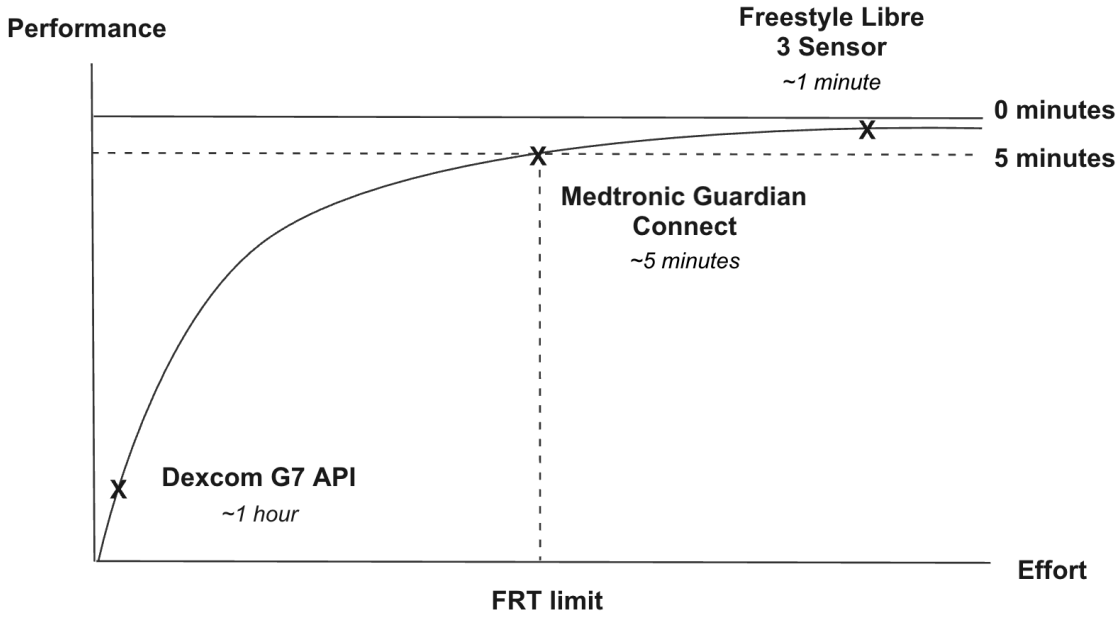
Dexcom G7 (API): The official API is intentionally delayed by ~1 hour (3 hours outside the US)[59] possibly as a policy choice to provide high quality analytics.[60] Thus, any thirdparty app using the public API sees at least a ~1 hour propagation delay, even though Dexcom's own app/receiver updates on a near-real 5-minute cycle (a latency not offered via the public API.)

[59] https://developer.dexcom.com/docs/dexcomv3/endpoint-overview/
[60] https://forum.fudiabetes.org/t/dexcom-developer-api-now-live/2059

Abbott Libre 3: Streams a new glucose reading every 1 minute to the user's phone, and integrated apps can relay that data almost immediately.[61] Under ideal conditions, the end-to-end delay to an authorised third-party service is on the order of ~1 minute (dominated by the sensor's one-minute sampling interval, with only a few seconds added for transmission).

Medtronic Guardian Connect: Generates a new reading every 5 minutes and pushes it to the phone immediately.[62] In the best case (forwarding each reading without delay), a third-party service can receive data in ~5 minutes end-to-end. As with other sensors, the frequency of measurement (5 min) dictates the latency; only a few seconds are added in transit.

Figure 2. Functional real time across CGM monitoring implementations



3.2 Use case: heart rate monitoring

In this use case, wearable heart-rate monitors must average beat-to-beat intervals over approximately one full respiratory cycle (**~8 seconds**) to produce a stable and meaningful BPM reading. So, we set the functional real time lower bound at this level.

Polar H10 (BLE): Streams heart-rate data packets covering ~0.5s of readings at a time, arriving on the phone essentially in real time.[63] In optimal conditions, the phone receives data in under 1 second, so we estimate ~0.5s as a best-case propagation to a thirdparty (if the app forwards data instantly.) **Garmin Vivosmart 5:** Provides a real-time heart-rate

---

61 https://www.freestyle.abbott/uk-en/support/faq/question-answer.html?q=UKFaqquestion-114#:~:text=The%20FreeStyle%20Libre%203%20app,hour%20glucose%20graph

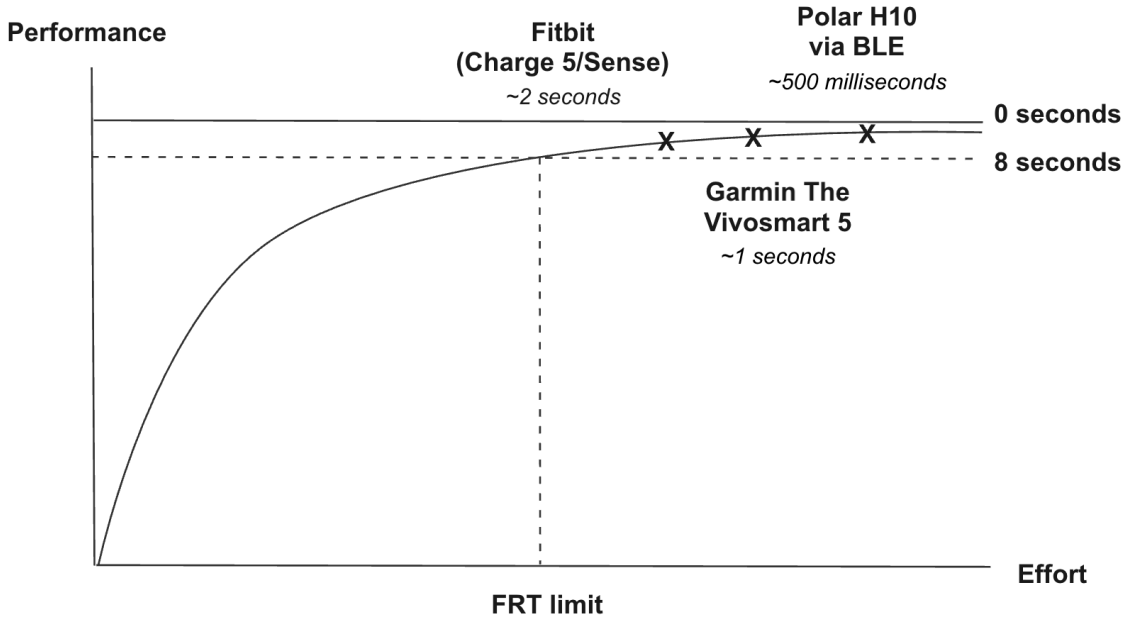
62 https://www.uspharmacist.com/article/guardian-connect-continuous-glucosemonitoring-system

63 https://github.com/polarofficial/polar-ble-sdk/issues/227

stream to the phone via its software development kit (SDK).[64] If an app immediately relays this data to a server, documentation indicates ~1 second end-to-end is achievable[65] (assuming the data isn't waiting for Garmin's cloud, which can add minutes of delay.)

Fitbit Charge 5 / Sense: Captures heart-rate data at 1 second granularity[66], but uploads that data only when the smartphone app syncs (eg, when opened, or occasionally in background). With the app open and forwarding data immediately, a best-case end-to-end delay of only a few seconds (~2 s) is achievable.[67] However, if the app isn't actively syncing, data might not be uploaded for minutes or even hours until the next sync.

Figure 3. Functional real time across HRM monitoring implementations



3.3 Use case: budgeting apps

In this use case, financial data aggregation can, in principle, propagate new transactions within the physical network limit of ~15ms round-trip time (RTT), meaning observed delays are caused by factors other than technical constraints. Since this is below the perceptual threshold for an "instant" response, we adopt Nielsen's 100ms limit as the FRT benchmark.

Yodlee (bank webhooks): Yodlee reports that webhook events are typically received ~5 minutes after the triggering user action, so ~5 min is the best-case latency for updates.[68]

---

[64] https://developer.garmin.com/health-sdk/questions-answers/

[65] https://www.thisisant.com/APIassets/1.0.0_ANTnRFConnectDoc/samples/ant_plus/ant_hrm/hrm_tx/README.html#:~:text=Parameter,8070%20(4.06%20Hz)

[66] https://fitbit.google/enterprise/researchers-faqs/

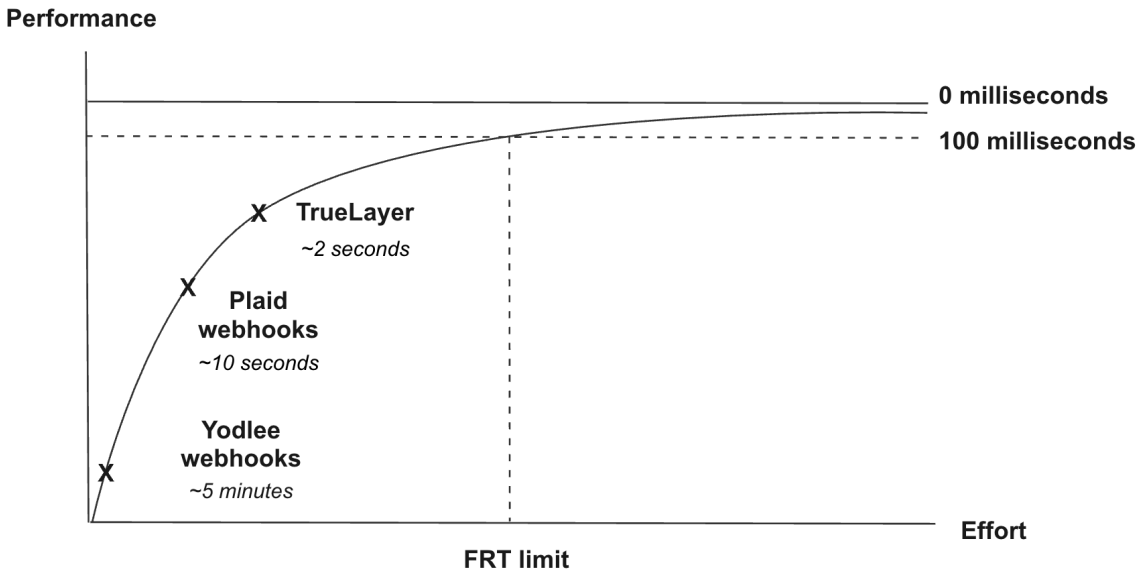
[67] https://dev.fitbit.com/build/reference/web-api/intraday/get-heartrate-intraday-by-date-range/#:~:text=Get%20Heart%20Rate%20Intraday%20by,Date%20Range

[68] https://developer.yodlee.com/resources/yodlee/webhooks/docs/consent_events

Plaid (transaction webhooks): Plaid's docs state that an "INITIAL_UPDATE" webhook usually fires ~10 s after a new transaction, and a "HISTORICAL_UPDATE" within ~1 min.[69] We take these as best-case latencies from the transaction event to the webhook. (Plaid's ongoing updates, however, only pull data a few times a day unless manually refreshed, so day-today latency depends on the refresh schedule and the bank involved.)

TrueLayer (instant payouts): An instant payout executes within a few seconds (SEPA Instant ≤10 s; Faster Payments ~instant).[70] [71] TrueLayer sends a webhook once the payment is marked "executed".[72] In ideal conditions (instant-capable banks, no risk checks, good network), we estimate ~2 s from initiation to webhook reception. (External factors like bank support or fraud checks can introduce additional delays.)

Figure 4. Functional real time across budgeting implementations



3.4 Use case: brokerage apps

Note: *While brokerage APIs are widely known for streaming public market data like stock prices in real-time, they are fundamentally tools for personal data portability. They transmit sensitive, user-specific information, including personal account balances, portfolio positions, and trade execution confirmations (successes or failures). Therefore, the performance of these APIs is critical for managing personal financial data and falls squarely within the scope of this analysis.*

In brokerage trading, data feeds can run at ultra-low latency, on the order of microseconds in co-located systems. These speeds are well below human perception thresholds, so we

[69] https://plaid.com/docs/transactions/webhooks/
[70] https://www.wearepay.uk/what-we-do/payment-systems/faster-payment-system/
[71] https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2024-11/EPC004-16%202025%20SCT%20Instant%20Rulebook%20v1.0.pdf
[72] https://docs.truelayer.com/docs/payout-webhooks

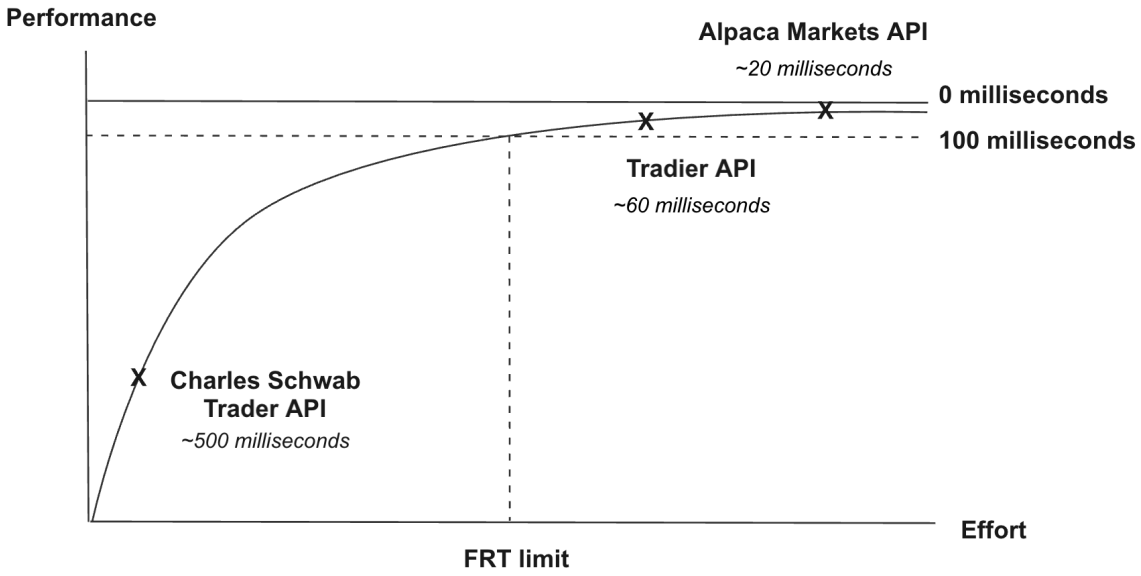
again use ~100ms as the functional real-time benchmark.

Schwab streaming API: Described as delivering updates in “up-to-the-second” time.[73] In practice, documentation implies a best-case latency of about 0.5s from market update to client[74] under optimal conditions (stable connection, highest priority streaming). Schwab calls this feed real-time[75] but offers no strict latency SLA. So, ~500ms is an inferred best case (with occasional network or rate-limit delays causing slower updates).

Alpaca API: Streams most market updates in as little as ~20ms[76] (aside from rare outliers, eg, a delayed bar arriving ~30s late due to late trade reports).

Tradier API: Streams quotes essentially as they happen[77]; unofficial reports suggest ~60 ms typical latency[78]. Neither Alpaca nor Tradier publishes a strict SLA, but in both cases we assume sub-second latency in the best case under normal conditions.

Figure 5. Functional real time across brokerage implementations



3.5 Use case: messaging & social media

In this use case, modern messaging and social media platforms are capable of delivering events in tens of milliseconds over optimised streaming protocols. Since this is below the perceptual “instantaneous” threshold, we apply Nielsen’s 100ms limit as the FRT reference point.

---

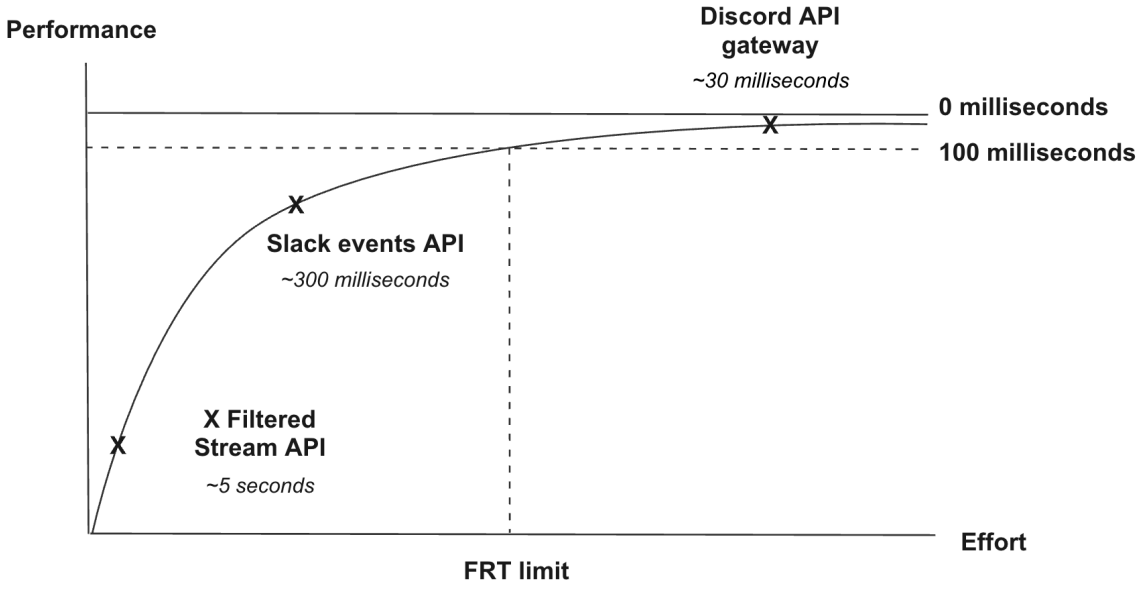
[73] https://www.schwab.com/execution-quality
[74] https://allensarkisyan.github.io/schwab-td-ameritrade-streamer/td-streamer/
[75] https://www.schwab.com/trading/web-trading
[76] https://forum.alpaca.markets/t/why-is-a-small-subset-of-1m-ohlcv-bars-delayed-by-30s-from-sip-websockets-data-connection/6207/2
[77] https://documentation.tradier.com/brokerage-api/overview/streaming
[78] https://www.elitetrader.com/et/threads/tradier-api.284313/page-2

Slack Events API: No official latency SLA (Slack only requires that apps ack events within 3 s).[79] In practice, events are pushed immediately when they occur, anecdotal evidence suggests around ~300 ms end-to-end.[80] We use ~0.3 s as the inferred best-case latency for Slack's event delivery.

Discord Gateway: Events on a persistent WebSocket typically arrive in the 30-300 ms range on a good connection.[81] There's no official latency promise, but ~30 ms can be considered an ideal-case scenario.

X (Twitter) Filtered Stream: This is currently the fastest feed offered by X, real-world reports show roughly ~5 s from a tweet's creation to its delivery to the client[82] under good conditions. (X provides no guaranteed latency; ~5 s is an inferred best-case, with actual speed varying based on filters, load, etc.)

Figure 6. Functional real time across social media implementations



3.6 Use case: mail

In this use case, calendar synchronisation can propagate changes within tens of milliseconds on a push-capable protocol. Given this is faster than the perceptual 100ms limit, we use Nielsen's definition of "instantaneous" as the relevant FRT level.

Fastmail (JMAP): With an active push connection, Fastmail delivers mailbox updates almost instantaneously. Fastmail's own reports and user anecdotes indicate new-mail

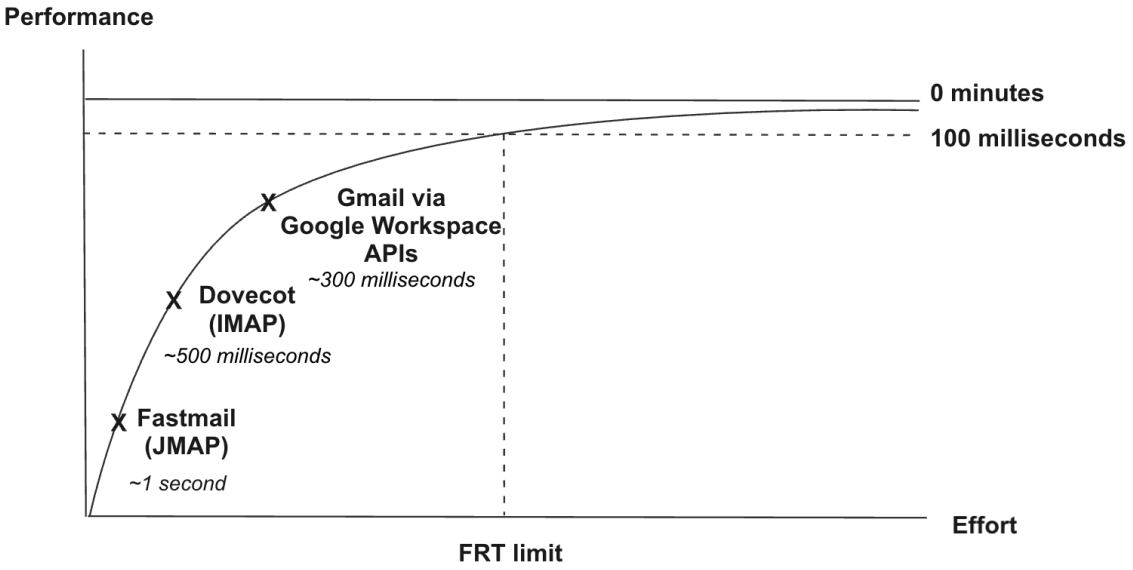
[79] https://docs.slack.dev/apis/events-api/
[80] https://community.boomi.com/s/article/Slack-Events-Integration-Framework
[81] https://javadoc.io/doc/org.javacord/javacord-api/3.1.0/org/javacord/api/DiscordApi.html#getLatestGatewayLatency–
[82] https://devcommunity.x.com/t/filtered-stream-delay/245941

notifications often arrive within ~1s in best cases.[83] [84] Any longer delays usually stem from extra processing (spam filters, mailbox locks) rather than the push protocol itself.

Dovecot IMAP (IDLE): By design, Dovecot throttles new-mail alerts with a ~0.5 s debounce (delay) before notifying clients.[85] This sets a best-case latency floor of roughly half a second (plus minimal network transit). Some servers have eliminated or reduced this builtin delay[86], achieving a few hundred milliseconds in practice. With no official latency SLA from Dovecot, ~0.5s is an inferred best-case for near-instant IMAP notification.

Gmail push (Pub/Sub): Google provides no strict latency guarantee, describing its Pub/Sub notification service only as “near-real-time.” Actual performance varies with server locations and load. One measurement by a developer found ~300ms to be an achievable bestcase latency when the Pub/Sub system is fully warmed up and publisher/subscriber are in the same region.[87]

Figure 7. Functional real time across email implementations



[83] https://www.fastmail.com/blog/what-we-talk-about-when-we-talk-about-push/
[84] https://news.ycombinator.com/item?id=15854226
[85] https://dovecot.org/mailman3/archives/list/dovecot%40dovecot.org/thread/J2L67F75QW5MJBIRKMBGA2AKNJHRC33X/
[86] https://github.com/chatmail/relay/issues/72
[87] https://www.davidxia.com/2021/08/benchmarking-kafka-and-google-cloud-pub-slash-sub-latencies/#:~:text=With%20my%20specific%20test%20parameters%2C.According%20to

4. Discussion: factors affecting real-time

4.1 Intermediary processes

4.1.1 Managing complexity and ensuring security

Many implementations that fall short of the FRT speed do so by design, intentionally trading speed for safety and managing scale. For example, aggregators like Yodlee (~5 min latency) and Plaid (~10s) use an intermediary-hub and hybrid model that slows data transfers down but vastly eases integration. These intermediaries[88] handle thousands of different bank connections, secures sensitive credentials, and perform fraud checks. Directly connecting to every bank's API would be impractical. Services like Plaid or Yodlee act as universal adapters, offering one unified interface at the cost of some latency.

Similarly, instead of pushing updates directly to every client, Gmail offloads that burden to a cloud intermediary (Google Cloud Pub/Sub). This design adds a slight delay but allows Gmail to reliably fan out huge volumes of notifications globally without maintaining countless individual WebSocket streams. Smaller platforms like Slack or Discord handle real-time events with direct server-to-client WebSockets, but at Gmail's scale an intermediary service is necessary to achieve stability and reach.

4.1.2 Adding analytical value

Sometimes a deliberate processing delay is built in to add value. For instance, the X (Twitter) Filtered Stream still delivers data via a direct API connection, but under the hood it runs incoming tweets through moderation and enrichment pipelines. This internal 'intermediary' step adds latency by design, yielding a safer and richer feed than a raw firehose.

Similarly, Dexcom G7 acts as its own intermediary for third parties. Namely, the device's cloud service delays data by >1 hour specifically to provide a retrospective/trend data service, rather than a live feed. In contrast, other CGMs like Libre 3 or Guardian Connect do not introduce such delays, instead relaying data to partners much closer to real-time. In each case, added latency is an intentional trade-off to filter or enrich the data before it reaches external developers.

Thus, the introduction of an intermediary pathway is a deliberate architectural choice which can be deployed to i) manage complexity, ii) enhance security, or iii) add analytical value to the data. While such systems may not be functionally real-time, their latency is often a legitimate and necessary trade-off for these other essential functions.

---

88 https://theodi.org/insights/explainers/what-are-data-intermediaries/#:~:text=What%20is%20a%20data%20intermediary%3F,'

4.2 Intrinsic cadence

4.2.1 Physical cadence

As noted earlier, every data source has an intrinsic cadence that sets a hard limit on update frequency. In health monitoring, for example, CGMs (Libre 3, Guardian Connect) cannot reflect blood-sugar changes faster than the body's own chemistry (~5 minutes for glucose to diffuse into interstitial fluid). Likewise, wearable heart-rate monitors (eg, Polar H10, Garmin Vivosmart 5) deliberately average readings over ~8 seconds (a full breath cycle) to yield a stable BPM instead of noisy beat-by-beat data. In such cases, trying to force faster updates would either deliver no new information or produce meaningless noise. This cadence is a deliberate design choice to produce a stable and clinically meaningful reading.

Attempting to sample more frequently than this natural cadence would be counterproductive. For instance, a CGM polling for updates every second would not yield new information, it would simply return the same reading for five minutes while needlessly draining the sensor's battery. For a heart rate monitor, it would deliver a stream of noisy data that is less useful for tracking fitness or health trends. Therefore, the "near-live" taxonomy classification can be evidence of a well-designed system that has optimised its architecture to match the physical reality of the data source.

4.2.2 Digital cadence

In purely digital contexts, platforms often impose cadence artificially for stability. For example, high-frequency trading APIs (Alpaca with ~20ms updates, Tradier ~60ms) enforce strict rate limits on requests. In these cases, using a push-based direct streaming model is essential for real-time updates. Without this, any attempt at millisecond polling would be throttled by the platform's rules.

4.2.2.1 Rate limits and sampling intervals

Rate limits control how many requests a client can make in a specific period.[89] These are necessary mechanisms for ensuring system stability and fair access, as they prevent any single user from overwhelming the platform's servers with too many requests in a short period.[90] For example, poorly configured or malicious application could send millions of requests without these limits being in place. This would consume a disproportionate amount of server resources like network bandwidth.

Rate limits are particularly visible on many of the implementations we have examined like X (Twitter), Slack, and Charles Schwab. The Charles Schwab Trader API limits market data requests to 120 per minute, while the X API limits timeline requests to 900 per 15-minute window. Similarly, Slack's API has multiple tiers, with most methods allowing between 20-50+ requests per minute. In particular, these limits guide developers away from high-frequency polling.

This architecture contrasts sharply with the "push" models, which are designed to bypass

[89] https://www.cloudflare.com/en-gb/learning/bots/what-is-rate-limiting/
[90] https://raghavan.usc.edu/papers/drl-sigcomm07.pdf

this problem entirely. Instead of the client repeatedly asking for data and hitting rate limits, the server automatically pushes updates as they happen, better enabling a truly live data stream. However, a "pull" model is still chosen for important trade-offs. It gives the client full control over when to request data and is significantly more efficient for conserving resources like device battery life. This often makes it a deliberate choice when achieving real-time speed is not the primary goal (eg, Dexcom G7).

4.2.2.1 Mutability

Finally, the mutability of data can introduce complexity.[91] [92] For an “immutable” stream like stock ticks, new data is simply appended. This is true for the brokerage APIs from Alpaca and Tradier, where each new stock price is a new, unchangeable fact. It's also true for health sensors like the FreeStyle Libre 3 or a Polar heart rate monitor; a glucose or heart rate reading from a minute ago is a historical measurement that cannot be altered. In these cases, the challenge is to deliver new data as fast as possible. The architecture is optimised for a one-way flow of information.

However, on platforms like Slack, messages can be edited or deleted (ie, “mutable”). This creates a complex cadence of “CREATE”, “UPDATE”, and “DELETE” events, shifting the architectural challenge from simply delivering new data to synchronising all changes in real-time. Interestingly, many of the systems with higher latency in our study (Slack, X, Plaid, Yodlee, Gmail, etc.) allow records to be edited or deleted after creation, unlike simpler feeds that only ever append new data. A message can be edited, a transaction status can be updated from "pending" to "cleared," or an email can be deleted. This creates a complex cadence of events beyond just "new data." To illustrate, being "real-time" here can mean:

- Delivering new messages quickly.
- Delivering edits to old messages just as quickly to all users.
- Delivering deletions of old messages to ensure they are removed from everyone's view instantly.

This means the system has to keep every user’s view fully in sync, not just deliver new items. Any lag or failure in synchronising changes could, for example, cause someone to see and reply to a message that was already deleted, or act on a transaction status that’s outdated. Handling new events in addition to edits and deletions in real time adds complexity (and with it, some latency) compared to a simple append-only feed.

To conclude, designing continuous and real-time data transfers around a data source's intrinsic cadence can help to i) ensure data quality by respecting physical limits, ii) maintain system stability via rate limits, or iii) preserve the integrity of mutable data. In other words, the fastest possible architecture is not always the most effective, as optimising for a data source's 'natural rhythm' ensures the information is meaningful, stable, and reliable.

4.3 Interoperable standards

The relationship between interoperability, open standards, and latency is also complex. Certain open standards (IMAP IDLE for email, JMAP, WebSockets, etc.) were created so

---

[91] https://www.barroso.org/publications/TheTailAtScale.pdf
[92] https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43864.pdf

that any third-party app can get updates without constant polling.[93] For instance, IMAP IDLE lets an email server instantly notify a client when a new message arrives, and Slack/Discord use standard WebSocket connections to push events to external apps in real time. Open standards eliminate the need for each service to poll continuously, allowing low-latency updates across a diverse ecosystem of apps through a common protocol.

The Open Data Institute (ODI) argues that key parts of our data infrastructure (including identifiers, standards, and registers) should be treated as public goods.[94] From this perspective, standards like IMAP or open protocols for financial data are foundational components of a shared infrastructure that enables competition and innovation.[95] Closed, proprietary ecosystems may achieve marginal performance gains by avoiding the overhead of interoperability. Yet, this can come at the cost of a less dynamic and more concentrated market.

4.3.1 The counterfactual of closed ecosystems

Finally, we note that closed, vertically integrated ecosystems can sometimes achieve the absolute lowest latency. For example, the fastest solutions we examined (such as Medtronic and Garmin's sensor systems) tightly control every component (from hardware through cloud to app) and avoid any third-party handoffs. This end-to-end integration can help to eliminate the overhead of interoperability, squeezing out extra milliseconds of delay.[96] Conversely, when the goal is to work across a diverse open ecosystem of apps and devices, using open standards is crucial for compatibility, even if it introduces a bit more latency.

In essence, there is a trade-off. Namely: i) maximum speed can be attained through a self-contained design in certain contexts, whereas ii) achieving real-time data transfers across a diversity of applications benefits from an easily-accessible, common language (ie, open and interoperable standards).

5. Concluding remarks

This report has aimed to bring clarity to the "continuous and real-time" data portability mandates emerging in digital competition policy by introducing a new taxonomy and the concept of Functional Real-Time. In this way, our analysis is intended to be descriptive (ie, a map of how real-time data transfers work in practice) rather than normative.

A key finding is that a one-size-fits-all standard for "real-time" is impractical. Context is paramount, and latency is often the result of deliberate and sometimes necessary trade-offs. We observed that many implementations fell short of their Functional Real-Time thresholds, not because of technical limitations, but because of architectural choices that prioritise other valuable goals.

---

[93] https://policyreview.info/pdf/policyreview-2024-2-1749.pdf
[94] https://theodi.org/insights/guides/principles-for-strengthening-our-data-infrastructure/
[95] https://theodi.org/news-and-events/blog/data-as-a-strategic-asset/
[96] https://assets.publishing.service.gov.uk/media/5a75c284e5274a545822e01a/The_economics_of_open_and_closed_systems.pdf

Furthermore, the taxonomy developed in this report provides a common language for these factors. Together with the FRT framework, this highlighted a range of factors likely affecting the degree of "real-time", such as: intermediary analytical layers to enrich data, intentional delays to ensure security and manage complexity at scale, and the inherent overhead required to support open, interoperable standards over closed ecosystems. Furthermore, the intrinsic physical or digital cadence of the data source itself often sets a hard limit on how "live" a data transfer can meaningfully be.

5.1 Implications for competition policy and enforcement

The findings and frameworks presented here have direct implications for the implementation and enforcement of digital competition laws like the DMA.

1. **For compliance assessments:** Regulators can use the taxonomy as a checklist to assess a gatekeeper's data portability implementation. For example, an architecture that relies solely on a "pull" model with "periodic" updates for dynamic data (like social media posts) might fail a functional real-time test, suggesting a possible compliance failure.
2. **For remedy design:** The FRT framework provides a defensible, context-aware method for setting latency requirements in remedial orders. Rather than mandating an impossible "absolute" real-time, orders can specify that data must be delivered within the FRT threshold for its use case (eg, 100ms for messaging, 5 minutes for glucose data).
3. **For distinguishing legitimate from anti-competitive delay:** The taxonomy helps identify when latency is a legitimate trade-off (eg, for security or data enrichment) versus when it is a deliberate tactic to undermine portability. A deliberate, non-technical delay like the Dexcom G7's 3-hour API lag, which is not applied to its own first-party app, could be scrutinised as a potential violation of the DMA's spirit and letter.
4. **For promoting interoperable standards:** The analysis suggests that while closed ecosystems can achieve the lowest latency, open standards are crucial for a dynamic and competitive market of third-party services. Policymakers should consider mandating support for open, push-based standards (eg, WebSockets, JMAP) to ensure a level playing field.
5. **For cross-jurisdictional learning:** The taxonomy and FRT framework provide a common language and evaluation metric for regulators in the EU, UK, US, Japan, and other jurisdictions considering DMA-style rules. This can help harmonise compliance expectations for global gatekeepers and create a shared evidence base for what constitutes *effective* real-time data portability.

Building on this work, policymakers and developers can move beyond an absolutist view of speed by using the combined toolkit of the taxonomy and the FRT framework. Instead, they can engage in a more nuanced evaluation of data portability solutions, ensuring that data flows are fast but ultimately fit for purpose.

Annex A. Methodology

Use case selection

The use case areas, use cases and implementations were selected to provide a broad and representative sample of user-initiated and user-consumed personal data transfers. This approach was chosen to distil generalisable principles about what "continuous and real-time" means across a diverse set of common applications.

Use case area screening criteria

a. **User-oriented data?** We restrict to data that end users generate through their own activity and can lawfully port to themselves or authorised third parties. This directly tracks the DMA's obligation on gatekeepers to provide "effective portability […] including continuous and real-time access" for end users and authorised third parties, via appropriate technical measures (eg, APIs). Selecting user-oriented data keeps the study aligned to the DMA's target of end-user data access and real-time portability.[97]

   Sensitive data? Priority is given to domains where misuse or delay has higher risk (eg, health and financial contexts). Health data is expressly a GDPR special category requiring enhanced protections; analysing real-time portability where safeguards are strictest stress-tests policy and implementation. Financial data, while not a GDPR special category, is widely treated by regulators as high-risk in breach contexts, warranting heightened controls; examining it surfaces real trade-offs between speed, safety and consent.[98]

   Diverse data? We attempted to study across domains and data types (eg, physiological signals, transactions, communications) to derive a taxonomy general enough for policy use beyond a single sector. We intentionally selected cases that are as different as possible to reveal cross-cutting patterns between different data types. In the portability context, diversity improves external validity and supports competition/empowerment aims highlighted in recent OECD work.

Use case screening criterion

b. **Consumer application?** Within each area, we include only consumer-facing applications where a user (or a third party they authorise) actually consumes the data stream. This keeps the analysis focused on portability as a practical right and competition lever for **end users**, rather than back-end M2M/IIoT integrations. In turn, this operationalises the DMA's end-user portability mandate (real-time and continuous access "to data provided by the end user or generated through the activity of the end user") and ties findings directly to potential enforcement scenarios.[99]

---

[97] https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX%3A32022R1925
[98] https://gdpr-info.eu/art-9-gdpr/
[99] https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX%3A32022R1925

Implementation screening criteria

c. **Adoption:** The chosen implementations have significant user adoption or are considered industry standards, making the analysis relevant to a large segment of the market. Niche or obscure technologies were avoided.

   Availability of technical specifications: An important requirement was the public availability of technical specification documents, API documentation, or detailed technical descriptions. This was essential to enable an evidence-based classification of each implementation against the taxonomy dimensions.

Functional Real-Time (FRT) framework

The FRT framework is a conceptual adaptation of Maister's First Law of Service, which posits that customer satisfaction is a function of the gap between perception and expectation, often expressed as:

$$Satisfaction = Perception - Expectation$$[100]

This psychological principle is translated into a simple quantitative framework for evaluating data transfer latency. In this model, "Perception" is analogous to **Immediacy**, the technical threshold at which a system's response feels instantaneous to a human user. "Expectation" is analogous to **Timeliness**, the context-dependent requirement for how quickly data must be delivered to be useful for a specific task. This relationship is formalised in a metric called the Real-Time Surplus (RTS), calculated as:

$$RTS = Immediacy - Timeliness$$

The RTS score provides a clear, quantitative basis for determining whether a system's real-time performance is limited by human perception or by the inherent constraints of the process itself. The credibility of the FRT framework rests on the evidence-based definitions of its two core components:

1. **Immediacy (perception):** Immediacy is defined as the perceptual threshold for an "instant" response. This value is benchmarked at **100ms**, based on the foundational usability research of Jakob Nielsen.[101] This body of work establishes that 100ms is the cognitive limit for a user to feel that they are directly manipulating an on-screen object and that the system is reacting instantaneously. While some specialised research has identified lower perception thresholds for specific, highly sensitive tasks like direct stylus input, the 100ms standard remains the most widely accepted and empirically robust benchmark for general human-computer interaction. Anchoring Immediacy to this principle provides a stable, evidence-based reference point for all use cases where the user's perceptual experience is the primary consideration.
2. **Timeliness (expectation):** Timeliness is defined as "the shortest meaningful interval at which a phenomenon can be observed or reported without distortion." This value is inherently context-dependent and must be justified by the physical, physiological, or logical constraints of the specific use case. For each use case area in this study, a

[100] https://www.humanfactors.com/newsletters/are_we_there_yet_effects_of_delay.asp
[101] https://www.nngroup.com/books/usability-engineering/

Timeliness value was established through a review of relevant technical and scientific literature:

a. **Continuous Glucose Monitoring (CGM):** Timeliness is set at **5 minutes (300,000ms)**. This is determined by the hard physiological limit of glucose transport from blood capillaries to the interstitial fluid where sensors take their measurements. Sampling more frequently would yield no new information.[102] [103] [104] [105]
b. **Heart Rate Monitoring (HRM):** Timeliness is set at **8 seconds (8,000ms)**. This reflects the clinical requirement to average beat-to-beat intervals over at least one full respiratory cycle to produce a stable and meaningful beats-per-minute (BPM) reading, filtering out natural arrhythmia.[106] [107] [108] [109] [110]
c. **Digital communications (Finance, Messaging, Mail):** Timeliness is set at **15ms**. This value represents the physical limit of network technology, benchmarked against the lowest reported Verizon intra-Europe round-trip time (RTT). It represents the fastest possible data transfer, abstracting away from application-level or systemic delays. [111] [112] [113]

The dual-component nature of this framework resolves the central tension between the DMA's ambitious "real-time" mandate and the practical realities of data generation and transfer. It allows for a fair assessment by distinguishing between systems that are technically capable of being instantaneous (eg, financial data transfer) and those that are limited by immutable physical constraints (eg, physiological monitoring.)

Interpreting RTS and determining the FRT threshold

The final FRT threshold for a given use case is determined by the RTS score, providing a clear logic for setting a defensible performance benchmark.

- If **RTS is positive (Immediacy > Timeliness)**, the use case is **perception-limited**. The physical process is faster than human perception. Therefore, the FRT threshold is set by the higher standard of **Immediacy (100ms)**.
- If **RTS is negative (Timeliness > Immediacy)**, the use case is **process-limited**. An inherent delay in the underlying phenomenon is much longer than the threshold for human perception. Therefore, the FRT threshold is set by the more lenient standard of **Timeliness**.

---

[102] https://pubmed.ncbi.nlm.nih.gov/24009261/
[103] https://pubmed.ncbi.nlm.nih.gov/26243773/
[104] https://pubmed.ncbi.nlm.nih.gov/31059282/
[105] https://pmc.ncbi.nlm.nih.gov/articles/PMC3005050/
[106] https://arxiv.org/abs/2306.07730
[107] https://www.ncbi.nlm.nih.gov/books/NBK549803/
[108] https://pubmed.ncbi.nlm.nih.gov/8598068/
[109] https://pmc.ncbi.nlm.nih.gov/articles/PMC6679242/
[110] https://pubmed.ncbi.nlm.nih.gov/25252274/
[111] https://www.verizon.com/business/terms/latency/
[112] https://www.netforecast.com/wp-content/uploads/2022-Internet-Latency-Report_NetForecast_NFR5149.pdf
[113] https://cdn.cabling-design.com/media/43612/64fd4fcf62bcd0.10037069.pdf

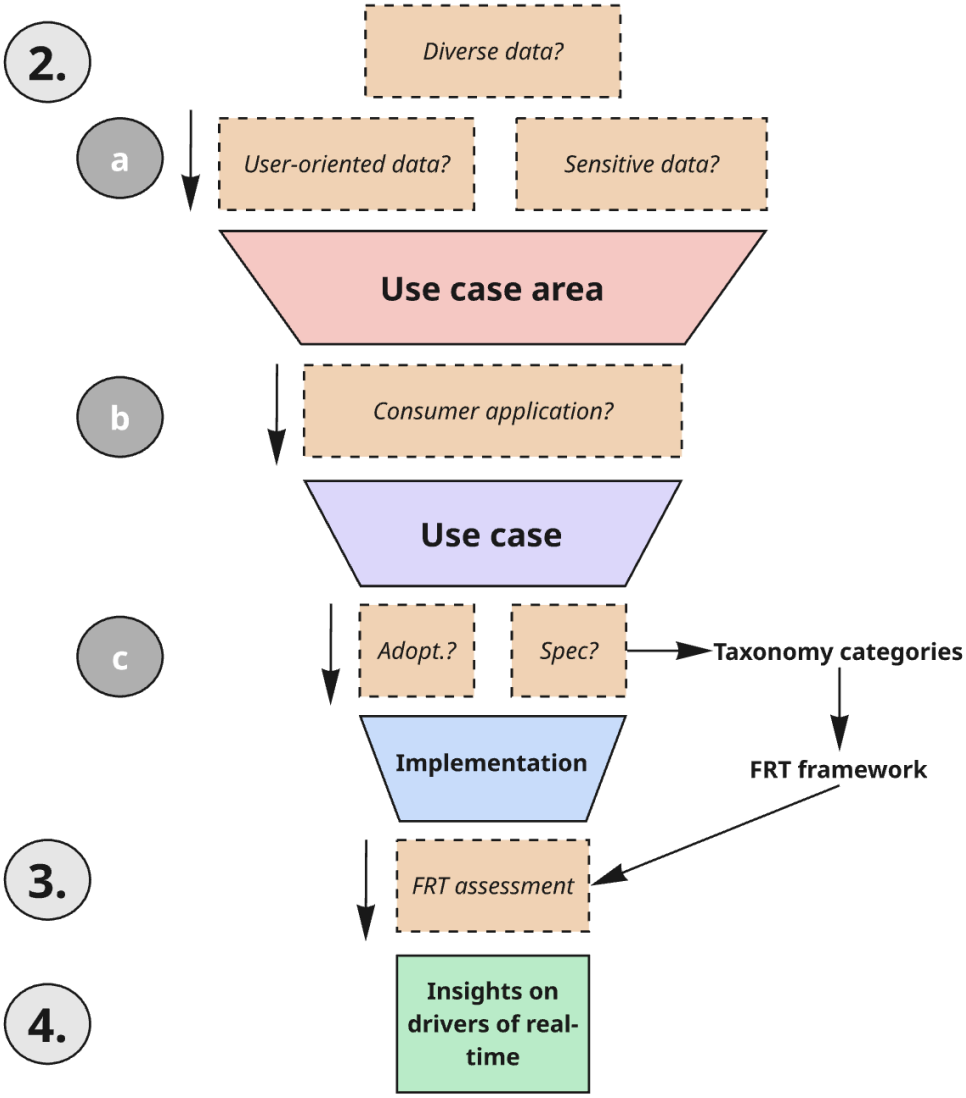
This logic allows for nuanced, contextually appropriate judgments. For example, a brokerage app with a 500ms latency would fail its FRT test, as this is well above the 100ms Immediacy threshold. In contrast, a continuous glucose monitor that delivers data every minute is well within its 5-minute Timeliness threshold and would therefore be considered functionally real-time for its purpose. Using this tool, we provide the FRT thresholds in **Table 2**.

Table 2. RTS scores and FRT threshold for each use case

Use case area	Immediacy (ms)	Timeliness (ms)	RTS calculation (ms)	Limiting factor	Final FRT threshold
Blood Glucose Monitoring	100	300,000	-299,900	Process	**5 minutes**
Heart Rate Monitoring	100	8,000	-7,900	Process	**8 seconds**
Budgeting Apps	100	15	+85	Perception	**100 milliseconds**
Brokerage Apps	100	0.011	+99.989	Perception	**100 milliseconds**
Messaging & Social Media	100	15	+85	Perception	**100 milliseconds**
Mail & Calendar Sync	100	15	+85	Perception	**100 milliseconds**

The process for selection of the use case area, use case, implementations and the FRT assessment are summarised below in **Figure 8.**

Figure 8. Methodology workflow


2.
Diverse data?
a
User-oriented data?
Sensitive data?
Use case area
b
Consumer application?
Use case
c
Adopt.?
Spec?
Taxonomy categories
FRT framework
Implementation
3.
FRT assessment
4.
Insights on drivers of real-time

Annex B. Taxonomy placement

Table 3. Continuous and real-time data transfer taxonomy

Criteria	Subcriteria	Implementations
1. Where is data sharing initiated by the user?	*Initiated from the source platform*	Garmin The Vivosmart 5 API Fitbit (Charge 5/Sense) API
	Initiated from the third-party application	Plaid Webhooks Yodlee Webhooks TrueLayer Charles Schwab Trader API Alpaca Markets API Tradier API Discord API X Filtered Stream API Slack Events API
	Initiated from on-device pairing	Dexcom G7 FreeStyle Libre 3 Sensor Medtronic Guardian Connect Polar H10 (via BLE)
2. What is the data pathway?	*On-device hub*	FreeStyle Libre 3 Medtronic Guardian Connect Abbott Lingo Garmin Vivosmart 5
	Direct connection	Polar H10 Fitbit Charles Schwab Trader API Alpaca Markets API Tradier API Discord API X (Twitter) API v2 Slack Events API Fastmail (JMAP) Dovecot (IMAP)
	Intermediary hub	Dexcom G7 Plaid Webhooks Yodlee Webhooks TrueLayer Gmail & Google Calendar (Workspace APIs)
3. Whose credentials?	*User credentials only*	Dovecot (IMAP) Polar H10 (via BLE)
	User and vetted 3rd party credentials	Dexcom G7 (Dexcom API V3) Plaid Webhooks Yodlee Webhooks

		TrueLayer Charles Schwab Trader API Alpaca Markets API Tradier API Discord API X Filtered Stream API Slack Events API Gmail (Google Workspace API) Garmin The Vivosmart 5 API Fitbit (Charge 5/Sense) API
	On-device pairing - Proximity	FreeStyle Libre 3 Sensor Medtronic Guardian Connect
4. How is user consent obtained?	*Explicit consent via redirect*	Dexcom G7 (Dexcom API V3) Plaid Webhooks Yodlee Webhooks TrueLayer Charles Schwab Trader API Alpaca Markets API Tradier API Discord API X Filtered Stream API Slack Events API Gmail (Google Workspace API)
	Implicit consent via in-app invitation	FreeStyle Libre 3 Sensor Medtronic Guardian Connect Polar H10 (via BLE) Garmin The Vivosmart 5 API Fitbit (Charge 5/Sense) API Fastmail (JMAP) Dovecot (IMAP)
5. What is the duration of the permission?	*Time-bound*	Abbott Lingo Plaid Webhooks Yodlee Webhooks TrueLayer X (Twitter) API v2 Gmail & Google Calendar (Workspace APIs)
	Open-ended	Dexcom G7 FreeStyle Libre 3 Medtronic Guardian Connect Polar H10 Garmin Vivosmart 5 Fitbit Charles Schwab Trader API Alpaca Markets API Tradier API Discord API Slack Events API

		Fastmail (JMAP) Dovecot (IMAP)
6. How is new data delivered?	*Pull*	Dexcom G7 FreeStyle Libre 3
	Push	Medtronic Guardian Connect Abbott Lingo Garmin Vivosmart 5 Polar H10 X (Twitter) API v2 Alpaca Markets API Tradier API Charles Schwab Trader API
	Hybrid	Plaid Webhooks Yodlee Webhooks Slack Events API Discord API Fastmail (JMAP) Dovecot (IMAP) Gmail & Google Calendar (Workspace APIs) Fitbit TrueLayer
7. Can existing data be changed?	*Immutable (append-only)*	Dexcom G7 FreeStyle Libre 3 Medtronic Guardian Connect Abbott Lingo Polar H10 Garmin Vivosmart 5 Alpaca Markets API Tradier API
	Mutable (append with changes)	Fitbit Plaid Webhooks Yodlee Webhooks TrueLayer Charles Schwab Trader API Discord API X (Twitter) API v2 Slack Events API Fastmail (JMAP) Dovecot (IMAP) Gmail & Google Calendar (Workspace APIs)
8. How frequent is the data?	*Live*	Garmin Vivosmart 5 Charles Schwab Trader API Alpaca Markets API Tradier API Discord API Fastmail (JMAP)
	Near-live	Polar H10 Fitbit Plaid Webhooks

		Yodlee Webhooks TrueLayer X (Twitter) API v2 Slack Events API Dovecot (IMAP) Gmail & Google Calendar (Workspace APIs)
	Periodic	Dexcom G7