2023-07-26

Reducing power and bandwidth use in apps to keep users happy

25 minutes allocated. 20 minutes for talk, 5 minutes for questions.

Hello, today I'm going to talk about two things you might want to pay attention to in your apps, how easy it is to do so, and some caveats to be aware of when implementing them. I'm going to talk about bandwidth and metered data, and then about power and power saving. For each, I'll give the APIs you need to use, and then some examples of how other apps make use of them.

Networks

- Networks can be metered
- Metered status is broadcast by the network, or set by the user
- Metered networks are generally slow or cost money per unit data

Firstly, let's talk about metered networks and saving bandwidth. A metered network is one where there's some cost per unit data, or some cap on the amount of data you can use. Generally the metered status is advertised by the network (using some special DHCP features), but sometimes it isn't and has to be set manually by the user. An example of a metered network would be tethered internet from a phone, or a 4G modem.

Reducing power and bandwidth use in apps to keep users happy

└Networks

Networks

- Networks can be metered
- Metered status is broadcast by the network, or set by the user
- Metered networks are generally slow or cost money per unit data

How can you save data? By downloading less stuff, downloading it less frequently, or not doing things altogether. For example, your app could check for updated content less often; not automatically download images or generate link previews; only download summaries of content and require the user to explicitly request the full version of something if they want it; etc.

`Gio.NetworkMonitor.network-metered`

GLib provides an API for indicating whether the network connection is metered, as a property, and as a `notify` signal for when that changes. You will always want to listen to the signal, as users may change networks several times during the lifetime of your app's process.

So, determining whether the network is metered is as simple as checking a property and connecting to a signal. Great! So you do that and disable or throttle some functionality in your app. But what if the user was expecting that functionality to work? It would be good to give them some feedback.

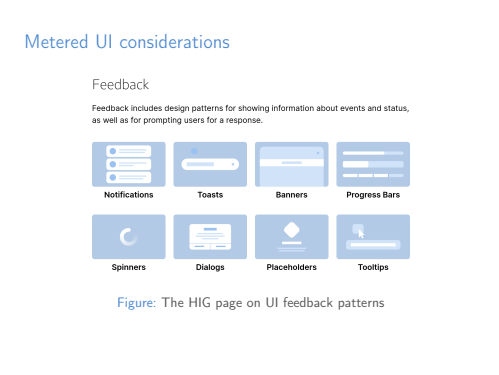
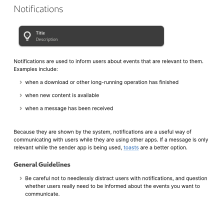


Figure: The HIG page on UI feedback patterns

The GNOME human interface guidelines (HIG) provide guidance on patterns for feedback to the user. You can read the HIG online. Being on a metered network is a temporary state, but it can be that way for a while. It's also a notable state transition when going from non-metered to metered, or vice-versa. That gives us several options for how to notify the user.

Reducing power and bandwidth use in apps to keep users happy

└─Metered UI considerations



Notifications

Title
Description

Notifications are used to inform users about events that are relevant to them. Examples include:

- when a download or other long-running operation has finished
- when new content is available
- when a message has been received

Because they are shown by the system, notifications are a useful way of communicating with users while they are using other apps. If a message is only relevant while the sender app is being used, toasts are a better option.

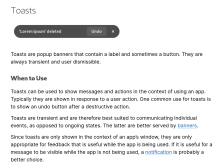
General Guidelines

- Be careful not to needlessly distract users with notifications, and question whether users really need to be informed about the events you want to communicate.

Figure: The HIG page on notifications

Here's what the HIG has to say about notifications. They're useful for when users are looking at another app, and you should be careful to not needlessly distract the user with them.

Metered UI considerations


Toasts

Toasts are popup banners that contain a label and sometimes a button. They are always transient and user dismissible.

When to Use

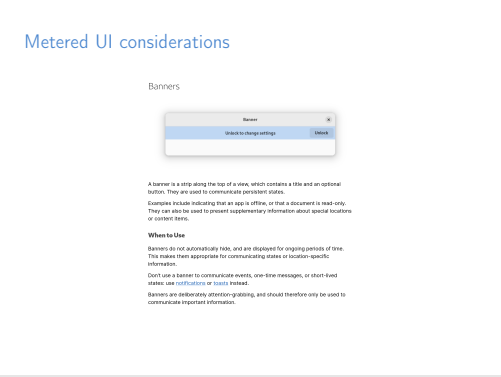
Toasts can be used to show messages and actions in the context of using an app. Typically they are shown in response to a user action. One common use for toasts is to show an undo button after a destructive action.

Toasts are transient and are therefore best suited to communicating individual events, as opposed to ongoing states. The latter are better served by banners.

Since toasts are only shown in the context of an app's window, they are only appropriate for feedback that is useful while the app is being used. If it is useful for a message to be visible while the app is not being used, a notification is probably a better choice.

Figure: The HIG page on toasts

This is the HIG guidance on toasts. They are useful in the context of an app, often in response to a user action. They are transient and are therefore best suited to communicating events rather than states.



The HIG guidance on banners. They are persistent and are used to communicate persistent state. They are deliberately attention-grabbing, so should only be used to communicate important states.

Metered example: Fragments

- https://gitlab.gnome.org/World/Fragments/-/merge_requests/156
- Pause torrents when moving onto a metered network
- Resume them when moving off the metered network (unless the user has modified them since)
- Display an infobar while metered

Let's look at an example app which supports changing its behaviour when on a metered network. Fragments is a torrent client, which might be running with torrents downloading or seeding in the background. The user probably doesn't want to spend their precious metered bandwidth on those torrents, unless they make an explicit choice to.

Reducing power and bandwidth use in apps to keep users happy

Metered example: Fragments

Metered example: Fragments

- https://gitlab.gnome.org/World/Fragments/-/merge_requests/156
- Pause torrents when moving onto a metered network
- Resume them when moving off the metered network (unless the user has modified them since)
- Display an infobar while metered

So, Fragments will detect the state transition for moving from a non-metered to a metered network, and pause all torrents at that point. Since the user probably expected those torrents to complete at some point, it will resume the torrents again when moving from a metered to a non-metered network, unless the user has explicitly modified the torrents in the meantime.

Metered example: Fragments

- https://gitlab.gnome.org/World/Fragments/-/merge_requests/156
- Pause torrents when moving onto a metered network
- Resume them when moving off the metered network (unless the user has modified them since)
- Display an infobar while metered

Throughout all of this, Fragments will show an infobar telling the user that torrents were disabled because of being on a metered network. The HIG recommends using infobars to communicate persistent state. To be really up to date with recommended widgets, this should be a libadwaita banner rather than a GTK infobar, but infobars are what's currently in use in the rest of Fragments. If a banner were used, the explanatory text should be shorter.

Reducing power and bandwidth use in apps to keep users happy

Metered example: Déjà Dup

Metered example: Déjà Dup

- https://gitlab.gnome.org/World/deja-dup/-/blob/main/libdeja/Network.vala#L59
- Doesn't begin a backup while on a metered network
- . . . unless a preference is set to override this

Another example app is Déjà Dup, a backup program. It can be configured to back up to a network location, which could use significant amounts of data. So, if running on a metered network it will delay backups unless explicitly told to run one by the user. The scheduling is overrideable with a hidden preference.

Reducing power and bandwidth use in apps to keep users happy

Metered example: Déjà Dup

Metered example: Déjà Dup

- https://gitlab.gnome.org/World/deja-dup/-/blob/main/libdeja/Network.vala#L59
- Doesn't begin a backup while on a metered network
- . . . unless a preference is set to override this

Since Déjà Dup runs in the background and does its backups on a schedule, the user will almost never see its UI. So the only way to notify the user that an expected backup has not run due to being on a metered network is a desktop notification. Déjà Dup tracks when it has most recently notified the user about a scheduled backup, so it doesn't emit notifications too often.

Reducing power and bandwidth use in apps to keep users happy

Metered example: Déjà Dup

Metered example: Déjà Dup

- https://gitlab.gnome.org/World/deja-dup/-/blob/main/libdeja/Network.vala#L59
- Doesn't begin a backup while on a metered network
- . . . unless a preference is set to override this

Most apps should not use notifications to tell the user about being on a metered network. Imagine what would happen if you were running 10 apps which changed their behaviour on a metered network, and they all emitted a notification when you moved from a non-metered to a metered network. Notifications need to be well justified (and I think Déjà Dup's use is justified).

Power

- Traditionally restricted to 'is the computer on battery?'
- Now a wider definition: power saver mode
- Can be explicitly enabled by the user in anticipation of being away from power for a long time

In the second part of the talk, I'd like to talk about power saving. Traditionally, apps have changed their behaviour based on whether the computer is on battery power. But now we have a wider definition: is it in 'power-saver' mode. This can be set by being on battery, or by an explicit request from the user, or by helper software.

Reducing power and bandwidth use in apps to keep users happy

Power

- Traditionally restricted to 'is the computer on battery?'
- Now a wider definition: power saver mode
- Can be explicitly enabled by the user in anticipation of being away from power for a long time

Apps use power by using computer resources. In general, this means using the CPU, using large amounts of memory, accessing disk or the network a lot (particularly if the network is wireless), or just waking up frequently so the processor can never get into a deeper sleep state. Saving power means doing less of that, doing it less often, or grouping together things so that they're all done at once rather than waking up several times to do several things.

`Gio.PowerProfileMonitor.power-saver-enabled`

GLib provides an API for indicating whether the computer is in power-saver mode, and it's quite similar to the metered data API. It's a property plus a `notify` signal, so all you need to do is read the property and connect to the signal. You will want to listen to the signal, as the power mode may change over time.

Gio.PowerProfileMonitor.power-saver-enabled

So if you use this API to disable, reduce or change some functionality in your app – as with metered data – you need to be able to notify the user somehow.

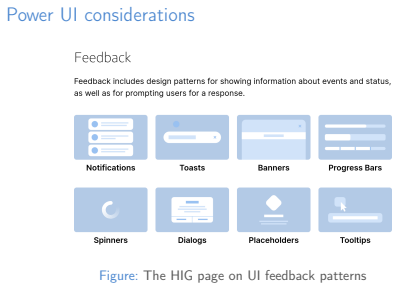


Figure: The HIG page on UI feedback patterns

Just as with metered data, the HIG provides guidance on patterns for feedback to the user. Being in power-saver mode is a temporary state, but it can be that way for a while. It's a less notable transition when going from another power mode to power-saver; it's more about the state here. This gives us several options for how to notify the user.

Reducing power and bandwidth use in apps to keep users happy

Power UI considerations

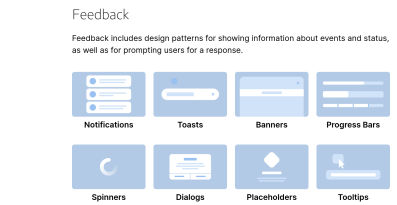


Figure: The HIG page on UI feedback patterns

As with any of the UI work here, follow the guidance in the HIG as it applies to your app. If anything seems odd, or isn't covered by the HIG, or you want a second opinion, ask on the GNOME design channel on Matrix.

Reducing power and bandwidth use in apps to keep users happy

Power example: Blanket

Power example: Blanket

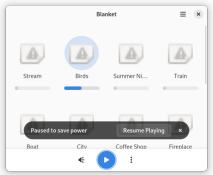


Figure: Blanket showing a toast when pausing sounds in power-saver mode

(The missing images here are an issue with how I've built it, and not an issue with Blanket.)

Reducing power and bandwidth use in apps to keep users happy

Power example: Blanket

Power example: Blanket

- https://github.com/rafaelmardojai/blanket/pull/312
- Pauses background playback when entering power saver mode
- Shows a toast explaining why playback has been paused
- In case Blanket's UI is not visible, this toast doesn't have a timeout

Here's an example of an app which changes its behaviour in power-saver mode. Blanket is an app to play soothing background sounds, such as rain or birdsong. When entering power-saver mode, it will pause the background sounds, as continually doing audio decoding drains power. It will display a toast in its window to explain why the sounds have stopped, and the toast contains a button to start playback again.

Reducing power and bandwidth use in apps to keep users happy

Power example: Blanket

Power example: Blanket

- https://github.com/rafaelmardojai/blanket/pull/312
- Pauses background playback when entering power saver mode
- Shows a toast explaining why playback has been paused
- In case Blanket's UI is not visible, this toast doesn't have a timeout

Blanket's UI may not be visible (you can close the window and it will continue playing), so the toast doesn't have a timeout. If the user wonders why the sounds have stopped, they will open Blanket's window, and see the toast. Another way to implement this would have been as a banner with a close button, but that's more of a way to represent an ongoing state, whereas pausing the sound playback is a transitional change. Blanket doesn't re-start playback when the system changes away from power-saver mode, as suddenly playing sound without user interaction is quite jarring.

Reducing power and bandwidth use in apps to keep users happy

Power example: NewsFlash

Power example: NewsFlash

- https://gitlab.com/news-flash/news_flash_gtk/-/merge_requests/148
- Affects whether to enable background syncing of news feeds
- Adds to existing logic which disables them on metered networks
- No UI

Another example is NewsFlash. This is a feed reader application. When in power-saver mode, it will disable background refreshing of feeds, saving on wakeups and network activity. Feeds will still be refreshed if the user explicitly requests it.

Reducing power and bandwidth use in apps to keep users happy

Power example: NewsFlash

Power example: NewsFlash

- https://gitlab.com/news-flash/news_flash_gtk/-/merge_requests/148
- Affects whether to enable background syncing of news feeds
- Adds to existing logic which disables them on metered networks
- No UI

The logic to implement this also disables refreshing of feeds when on a metered network. Colocating both of these checks is quite common — typically measures taken to reduce network data use will also reduce power consumption.

Reducing power and bandwidth use in apps to keep users happy

Power example: NewsFlash

Power example: NewsFlash

- https://gitlab.com/news-flash/news_flash_gtk/-/merge_requests/148
- Affects whether to enable background syncing of news feeds
- Adds to existing logic which disables them on metered networks
- No UI

Since it's only background refreshes which change behaviour, NewsFlash doesn't need to notify the user at all. Seeing updates for their feeds now vs in a few hours' time will not really affect them.

What haven't I covered?

- Profile your apps for power usage: reduce wakeups, reduce spinning
- `powertop` is amazing
- Profile your apps for network usage too: `nethogs`
- `Gio.MemoryMonitor.low-memory-warning`
- Don't use web views or write your app using web technologies if you can help it, they're terrible for power use
- Occasionally look at your project's CI resource and GitLab disk use

There's a lot of other approaches to improving power and network use which I haven't covered. These things can be optimised in general, even when not on a metered network or in power-saver mode. `powertop` and `nethogs` are useful for seeing when your app is waking up or how much data it's using. There are many different techniques for optimising an application, but they all come down to looking at the app's behaviour first and identifying problems.

What haven't I covered?

- Profile your apps for power usage: reduce wakeups, reduce spinning
- `powertop` is amazing
- Profile your apps for network usage too: `nethogs`
- `Gio.MemoryMonitor.low-memory-warning`
- Don't use web views or write your app using web technologies if you can help it, they're terrible for power use
- Occasionally look at your project's CI resource and GitLab disk use

Also, take a look at `Gio.MemoryMonitor`. It's a similar API to the two I've covered today, but for situations where the machine is running low on memory. In these situations, your app should free up resources which it doesn't immediately need (making a CPU/memory/disk tradeoff). I've run out of time to cover this though! Talk to me after for more information about it.

What haven't I covered?

- Profile your apps for power usage: reduce wakeups, reduce spinning
- `powertop` is amazing
- Profile your apps for network usage too: `nethogs`
- `Gio.MemoryMonitor.low-memory-warning`
- Don't use web views or write your app using web technologies if you can help it, they're terrible for power use
- Occasionally look at your project's CI resource and GitLab disk use

Another completely different area of resource use is development: we've only covered resource use by users of your app, but sometimes development resources can be more significant than we think. How often does your CI run? Does it need to run that often? How many tens of gigabytes of project data are you storing on GitLab, and is it all necessary? Or has some of it been forgotten about?

What haven't I covered?

- Profile your apps for power usage: reduce wakeups, reduce spinning
- `powertop` is amazing
- Profile your apps for network usage too: `nethogs`
- `Gio.MemoryMonitor.low-memory-warning`
- Don't use web views or write your app using web technologies if you can help it, they're terrible for power use
- Occasionally look at your project's CI resource and GitLab disk use

So, if your app uses the internet, please add support for metered data detection. If it could be changed to do less work when in power-saver mode, please do that. And in any case, it would be great if you could take 10 minutes to do some quick profiling with `powertop` and give your CI a checkup!