For more context, also read these related documents:

- Ilya's proposal for an updated NetInfo API
- Proposal for a chrome.system.network Chrome Apps API

Brief history of NetInfo to date

- Circa 2009, NetInfo was split from "System Information API" into a separate spec
- NetInfo v1 provided single **navigation.connection.type** attribute, which returned one of the following connection types:
    - `unknown, ethernet, wifi, 2g, 3g, 4g, none`

  Android browser implemented NetInfo v1 API.

- NetInfo v2 was re-drafted by Mozilla and added:
    - **onchange** event handler for when connection type changes.
        - *This enables the application to subscribe to change events instead of having to poll for them.*
    - **navigator.connection.type** was replaced with **"bandwidth"**
        - *This is a big lie.*
    - **navigation.connection.metered** (boolean)
        - *This is mostly a lie.*

  Bandwidth is a blatant lie. Estimating bandwidth is a fool's errand on wireless networks where the value fluctuates wildly based on location, time of day, number of active peers, and a myriad of other external factors affecting the local "network weather". Case in point, Mozilla's own implementation on Android ended up mapping static values based on connection type - see GeckoNetworkManager. So much for that…

  Metered is mostly a lie. In practice, knowing this would require some ability to talk to your upstream provider to figure out this parameter, or rely on user input. Unfortunately, none of the current implementations provide either. Instead, Android provides isActiveNetworkMetered(), but internally that simply maps "mobile" to metered. Not surprisingly, Firefox does the same thing. To be clear, this heuristic is consistent across iOS, Android, and Microsoft platforms, and it's not entirely unreasonable -- there are cases where it fails (e.g. WiFi tethering via 3G/4G, slow+metered hotel WiFi, etc.), but it is mostly right. That said, we can do better.

  Sidenote: Chrome had an implementation of NetInfo v2, but it was never enabled and was consequently nuked from the tree back in April.

NetInfo 3.0 proposal

Keep the good parts of v2 (onchange), revert “bandwidth” back to last-hop connection “type”, drop “metered” and replace with “cost”.

We should make no pretense about our ability to estimate bandwidth or latency. At best, what we can do is provide the **connection type of last hop** - in fact, perhaps we should even emphasize that what we’re surfacing is “last hop”.

- Yes, last hop can be misleading in some cases - e.g. Wi-Fi tethered via 3G/4G.
- Nonetheless, in majority of cases last hop connection type is still a signal that the application can successfully use to adjust its network access and update logic (see examples above). Further, if we take into account user settings for BW limits (aka, “cost”), then we have an override for default heuristics.

In summary, there are three parts to the v3 API:

- **navigator.connection.type** and **connectionClass** to retrieve last hop connection type.
- **connectionchange** event to indicate connection type change.
- **navigator.connection.cost** to retrieve some information about the metered/unmetered state of the connection.

```
interface Connection {
    // Last hop connection type, one of:
    // ‘ethernet’, ‘wifi’, ‘cellular’, ‘bluetooth’, etc
    readonly attribute DOMString type;

    // Class of the connection, values depending on the connection type.
    // ethernet: ‘10’, ‘100’, ‘1000’, ‘10000’, etc
    // wifi: ‘a’, ‘b’, ‘g’, ‘n’, ‘ac’, ‘ad’, etc
    // cellular: ‘2’, ‘2.5’, ‘2.75’, ‘3’, ‘3.5’, etc
    // bluetooth: ‘1’, ‘1.1’, ‘1.2’, ‘2.0’, etc
    readonly attribute DOMString connectionClass;

    // Returns true if the current connection class is considered to be at
    // least as good as otherClass. This makes it possible to compare
    // connection classes while allowing future extension with other
    // connection classes. ‘otherClass’ should be of the same type as the
    // current connection.
    boolean isClassAtLeast(DOMString otherClass);

    // Whether the user is near a data-plan limit. Possible values:
    //   unrestricted=0
    //       Default value for WiFi and ethernet connections
    //   metered=1
    //       Default value for mobile connections
    //   approachingDataLimit=2
    //       Returned if limit is set and BW usage exceeds 80%
```

```
    //   overDataLimit=3
    //       Returned if limit is set and BW usage exceeds 100%
    //
    // In cases where the plan limit is unknown, this field will be
    // set to unrestricted. [Author's note] The reason for not including a
    // separate "unknown" value is to protect against lazy coders who
    // forget to handle cases other than what their current machine is
    // reporting. If they correctly handle "unrestricted" but break when
    // the system detects some kind of metering, that's a better experience
    // for the user because the app will likely stop consuming data.
    readonly attribute DOMString cost;
    readonly attribute short costId;

    attribute EventHandler? onchange;
};
```

Examples of usage

```
// check current connection type, adjust logic
if(navigator.connection.type == "wifi") { ... }

// subscribe to connection type updates
window.addEvenListener("connectionchange", function(e) {
 var conn = navigator.connection;

 // Adapt based on current connection type (example) ...
 switch (conn.type) {
   case "wifi":
     runBackgroundUpdate();
   case "cellular":
     // if < 4G or approaching or exceeded BW limit
     if (!conn.isClassAtLeast("4") || conn.costId >= 2) {
       lowerPrefetchSize_orSomeSuch();
       break;
     }
    default:
     // ...
     break;
 }
});
```