Technical Disclosure Commons

Defensive Publications Series

March 20, 2019

API for connecting to captive portals

Ezra Gorman

Timothy Hartley

Jonathon Blumenthal

API for connecting to captive portals

ABSTRACT

This disclosure describes an API that enables clients to connect to access-controlled WiFi networks. The WiFi service provider describes requirements to connect to the network via the API. The API enables clients to satisfy the requirements in an automatable fashion and to inspect the status of their connection. The API can be integrated into an app of a WiFi service provider, and provides a robust, standardized WiFi onboarding process and user experience. The API can also be integrated into an app of a cellular service provider for the purposes of offloading to a WiFi network. Alternately the API can function in a standalone manner.

KEYWORDS

- Captive portal
- HTTP redirect
- HTTPS redirect
- WiFi hotspot
- WiFi offloading

BACKGROUND

A captive portal is a web page that the user of a public-access network is obliged to view and interact with before access to a network is granted. Captive portals are typically used at airports, railway stations, hotels, coffee shops, business centers, etc. that offer WiFi hotspots (wireless access points). A captive portal is used by the WiFi service provider to obtain user acceptance of terms of service, payments for the service, verification of identity, etc.

Navigating captive portals in a programmatic and extensible manner is currently limited, and where available, non-standard. Consequently, clients cannot automatically detect the presence of a captive portal and connect in a seamless manner.

DESCRIPTION

This disclosure describes a machine-interactive API that enables clients, e.g., web-clients, smartphone apps, JavaScript applications, or other code that interacts with the API, etc., to connect to access-controlled WiFi networks. In particular, the API has the following characteristics.

- The API enables a WiFi service provider to delineate the steps or requirements to connect to the network, e.g., secure the user's agreement to the terms of service; enable the user to make service-related payments; enable the user to buy a subscription plan; offer to the user an option for viewing ads; request and obtain permission from the user to share relevant information, e.g., location information; with the user's permission, verify the user's identity; etc.
- The API enables the WiFi service provider to provide multiple paths, e.g., payment options including direct carrier billing (DCB), vouchers, credit cards, etc., for connecting, each with corresponding requirements.
- The API can provide different outcomes, e.g., different bandwidths, based on the user's selections.
- The API can include display, venue, or partner attribution details to clients.
- The API can be used obtain additional information from the client, including identity verification, as necessary. Identity verification is done with user permission, and can be executed via identity service providers, via know-your-customer (KYC) routines, etc.

- The API works across multiple client types, e.g., web-clients, native applications, etc.
- The API enables display of custom offers, ads (rendered on the server side), sponsorships, etc.
- The API provides discovery support, e.g., provides a list of access points or hotspots near the user such that clients can automatically connect to or offload from cellular services. The API thereby enables seamless merging of different (cellular vs. WiFi) network types. A unified user experience is maintained across a multiplicity of networks, network types, hotspots, and platforms.
- The API displays information with full internationalization (i18n) support.
- The API enables a user of WiFi services to navigate the flow of the captive portal, e.g., by meeting service provider requirements and by presenting different screens based on the step of the flow that the user currently is in.
- The API enables a client to access multiple WiFi service providers simultaneously. The same client code supports multiple different hotspots with different (and possibly changing) connection flow requirements. This enables faster experimentation over the selection of networks.
- The API supports auxiliary integration points with other flow or content providers.

The described API can be implemented, e.g., as a RESTful API. If implemented as a RESTful API, e.g., a series of JSON calls going back-and-forth between client and portal, a device can have an app that can detect the presence of the captive portal and connect to it by navigating through the flow of the captive portal.

Fig. 1 illustrates an example sequence of communications between a user (102) of a client device (104), the captive portal (106), the API (108), a WiFi service provider (110), and

the network packet data protocol (PDP) (112), per techniques of this disclosure. A user's traffic is directed to the captive portal using an HTTP(s) redirect protocol (114). The client interacts with the API to get configuration information (116). Terms of service (ToS) are provided to the user and accepted by the user (118).

A request is made of the user to enter the user's telephone number, which is accepted (120). The ToS is updated as necessary (122). The user's identity is verified (124). Other requirements can also be satisfied in a similar fashion. The client requests and receives the status of the connection (126). The API requests and receives approval to access websites outside the captive portal, e.g., content outside the walled garden, on the Internet. Once the user is finished with the access granted by the WiFi service provider, the user requests and receives approval to re-enter the walled garden (128). The API provides the client the status of the connection, which is displayed to the user as a success page (130).

To summarize, the techniques of this disclosure feature the following.

A well-structured API

The techniques provide an API to enable clients to gain knowledge about networks at certain locations, get their network connection state, to receive instructions about the requirements to connect to the network (scoped to the current connection), and RESTful methods to complete the requirements. The system presents to clients a graph of connection flow requirements and steps. The API represents an application-level control of connectivity to a captive portal which does not require network-layer interaction on the client side. A client can thereby focus on the user-experience, rather than on integration with complex network protocols. The API is also a single end-point for clients, and is agnostic to platform or client-type. If a client

is restarted with the same session id, it can use connection session state to resume uncompleted operations.

A connection requirements graph

The techniques construct and expose a graph that represents the possible flows a user can follow to connect to the network. The graph may have multiple paths with multiple different final end states. This provides the full set of configuration nodes that the client executes prior to the user gaining full access to the WiFi network. The set of returned configuration nodes may define dependencies, thus creating an ordered execution. Once all nodes are successfully executed, the client can transition the user out of the walled garden. The graph is presented to clients, and can be altered or adapted for many offer types or flow paths. Dynamic or runtime alterations of the connection flow graph enable flexibility and responsiveness to dynamic conditions. The techniques enable a client to decide when to (partially or fully) tear down the walled garden, e.g., after the user submits KYC information, after the user completes a survey, etc.

Configuration node

A configuration node comprises a name, a set of dependent nodes, an indicator of whether or not the node is required to execute, and a single piece of configuration data.

Configuration data

As mentioned before, there are four or more types of configuration data, as follows.

- Terms of service configuration data (TOS): Defines a single TOS that is required for the user to accept. It is possible to look up the status of the TOS acceptance such that the client can skip the requirement to re-accept TOS.

- Identity verification configuration data, known as know your customer (KYC): Defines a single KYC requirement for network access. KYC is started by the user submitting a configurable set of identifying information and responding to optional challenges (such as receiving an SMS OTP).
- User provided data configuration data (UPD): Defines a set of user input that is collected by the client and sent back to the API.
- Composite configuration data: A collection of nodes that defines a "subgraph." The nodes in the composite are either all required or any required. This allows, for example, for multiple TOS configurations to be sent while only requiring the user accept one.

The configuration data for a given session does not change after the first request to get the configuration data. This, along with the connection session state, enables clients to resume some or all operations.

User interface interactions

The techniques provide granular detail such that clients can generate needed user interfaces to guide a user through the process. The system supports additional user-experience customization options, e.g., display configuration details for custom branding and display. The system natively supports internationalization and localization. The system provides clients with user input validation information and enforces such validation on the server side.

Discovery

The techniques support the ability for applications to find hotspots, learn what is offered by a particular hotspot, and determine the best hotspot to connect to.

Abstraction of dependencies

The techniques decouple user interactions from network enforcement and management interaction. The techniques also provide translation of flow constraints to network specific implementations, e.g., allow/disallow internet access, limit bandwidth rates, limit duration of access, etc.

Centralized management

The techniques store configurations for flows and graphs in a central repository, enabling configuration of flows, graphs, and offers to be customized by location, user, time of day, etc. The central storage of configurations enables configurations management across large and/or mixed/federated networks. Generalized client libraries can be used in multiple integrations against the same central service, reducing cost/complexity for client implementers.

Alternately, a captive portal API can be implemented using a non-RESTful interface. Further, the connection configuration can be pre-delivered to client, or configuration can be delivered out-of-band. Still further, the configuration graph may not be configurable; rather, clients can synchronize to well-known graph features. As another alternative, connecting to a captive portal can be implemented via a network-level connection mechanism acting as the transport protocol for the connection system. Rather than sending full graph, only the next required step can be presented to the client. Additional details such as post-connection disposition, e.g., landing page, follow-on content for the user, etc., can be included.

CONCLUSION

This disclosure describes an API that enables clients to connect to access-controlled WiFi networks. The WiFi service provider describes requirements to connect to the network via the API. The API enables clients to satisfy the requirements in an automatable fashion and to inspect the status of their connection. The API can be integrated into an app of a WiFi service provider,

and provides a robust, standardized WiFi onboarding process and user experience. The API can also be integrated into an app of a cellular service provider for the purposes of offloading to a WiFi network. Alternately the API can function in a standalone manner.

FIGURES

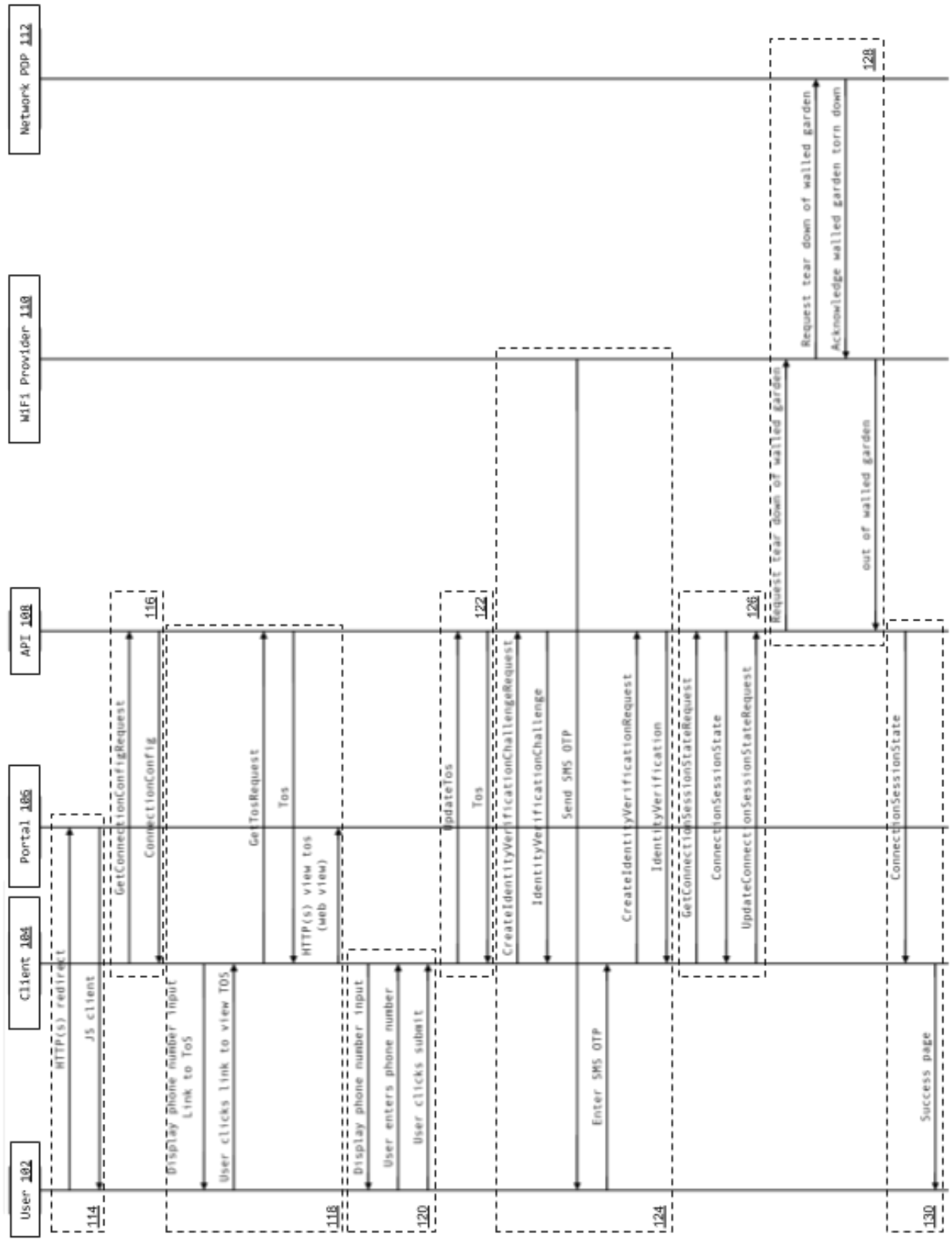

Fig. 1: Communications between the user, client, portal, API, service provider, and network PDP