User Reachability in Multi-Apps Environment

Kundan Singh
Avaya Labs Research
Santa Clara, CA, USA
singh173@avaya.com

***Abstract*— Recent progress in web real-time communication (WebRTC) promotes multi-apps environment by creating islands of communication apps where users of one website or service cannot easily communicate with those of another. We describe the architecture and implementation of a multi-platform system to do user reachability in multiple communication services where users decide how they want to be reached on multiple apps, e.g., in an organization that has voice-over-IP, web conferencing and messaging from different vendors. Our architecture separates the user contacts from reachability apps, supports user and endpoint driven reachability policies, and has several independent and non-interoperable WebRTC-based apps for two-way and multi-party multimedia communication. Our flexible implementation can be used for enterprise or personal communications, or as a white-labeled app for consumers of a business.**

Keywords—system design; mobile app; user reachability; multi-services; VoIP; WebRTC; caller policy

I. Introduction

In today's multi-apps environment, user reachability is often done manually via user presence and iteration, e.g., check if the user is online on Gtalk or Yahoo before sending a message there, or try Hangout video with fallback to phone. WebRTC (web real-time communication)[4] encourages this behavior further, although such web services often have similar features of web conferencing or click-to-call. However, a user likes to reach and be reached from her people irrespective of the service or device, and be able to select the best available mode, device or app, e.g., use text message in noisy environment.

Our approach to automate and simplify user reachability is to *decouple the contacts from communication apps*. Contacts are managed by the user, or dynamically injected by her context, e.g., current browsing or calendar. We have developed such an app, Strata Top9, which is a front-end to launch and interact with other apps to reach a user on voice, video or text. The user independently installs the communication apps from various services. Strata determines the right app with automatic fallback, e.g., use the video app, and if fails, try a phone call.

We have also developed cross-platform communication apps using WebRTC to initiate a video call, join a conference or an upcoming meeting from calendar, discover and connect with other local users, or translate between speech and text modes. Our apps use existing services based on Avaya's IP office, Conferencing or Media Server, or for endpoint driven apps, a Resource Server. They focus on mobile usability, but can also be installed as native desktop apps or accessed in a browser.

We describe the architecture and implementation of this multi-apps user reachability using dynamic contacts and user driven policies. The paper contains motivational use cases (Section II), differences from related work (Section III), pieces of the system architecture (Section IV), implementation of communication apps (Section V) and conclusions (Section VI).

II. Motivational Use Cases and Requirements

People use multiple communication apps due to device constraint, personal preference, enterprise policy, etc., e.g., Facetime on iOS vs. Hangout on Android, Facebook for friends vs. messaging or VoIP in office. In both personal and business communications, many users can be reached in multiple ways, on different apps, devices or communication modes, e.g., with multiple unified communication (UC) and messaging systems from different vendors seen in hospitals and banks today.

Consider the scenario in Fig.1 with three communication services and ten users connected to some of these. People are on multiple services, e.g., Gail on the hospital phone number and the public video call app. Richard (on left) uses a social app to reach his friends, some of whom work at First Hospital. The dotted arrows show the caller or receiver's preferred mode, e.g., Bobby likes to receive email; Richard prefers video to reach Kathy. Contacts in Richard's app show their preferred modes, or undefined "?" for pending contact requests.

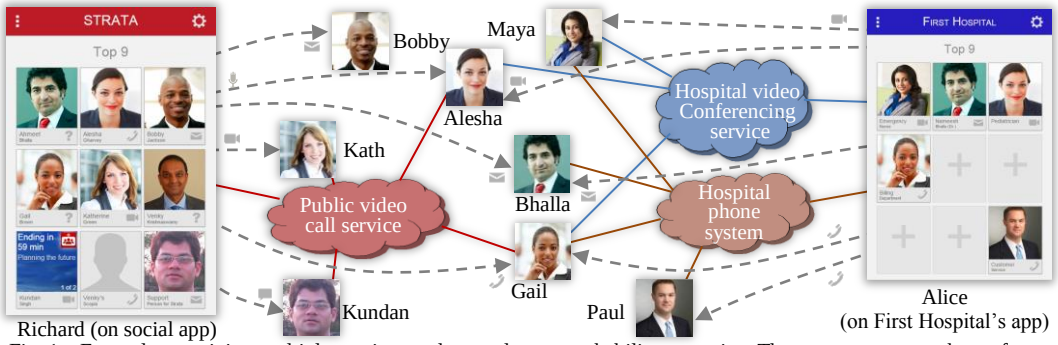


Fig. 1. Example containing multiple services and several user reachability scenarios. There are two screenshots of our app: the social app on left and a white-labelled app customized for one fictitious business (First Hospital) on right.

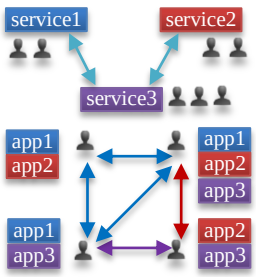


Fig. 2. Reachability: pair-wise service federations vs. user driven apps in the endpoint.

Alice's app (right) downloaded from the First Hospital's website is pre-populated with important contacts of on-call nurse and billing department. They automatically update as the staff changes shift, indicating who will receive the call, and in what mode. In automatic fallback, if Alice's video call fails to the on-call nurse, Maya, the app tries to reach her phone. The patient can fill the empty slots with her pediatrician or primary care provider, or put other dynamic contacts, e.g., "Billing/Ace HMO" to directly reach the right person, unlike navigating voice prompts; or keywords "pregnancy, natural" to reach a nurse with matching skills. The contact picture can instead show dynamic content, e.g., next calendar meeting, live video of the doctor, or periodic snapshots from her webcam to show if she can receive a video call. The contact may be non-person, e.g., meeting bridge; and may not be call or text reachable, e.g., click to open/edit a shared document or personalized webpage. Important system requirements to support such use cases are:

1. Multiple communication apps and services, independent of each other and of the contact list.
2. User driven reachability decisions by caller and receiver, besides any service enforced policies.
3. Diverse multi-platform apps; selection per call attempt.
4. Automatic fallback of apps, devices or modes; either caller or receive can set the preferred or required mode.
5. Minimum reachability via phone and email; e.g., when a doctor accepts the contact request from his patient, he gives a personal guarantee to respond in a timely manner.
6. Asymmetric contacts; e.g., a doctor does not have to add his patients in his contact list, to keep the list small.

III. Background and Related Work

Voice communication and email have historically provided universal reachability via phone numbers and email addresses. Today's communication tools of VoIP, IM, on web, or over-the-top apps are often based on open protocols, albeit in a service provider's "walled garden", which hinders reachability on another service, or locks the ecosystem [1][2][3]. WebRTC [4] for plugin-free browser-to-browser communication further makes it easy to create such silos [8]. Past user reachability efforts roughly fall under three overlapping categories: pair-wise federation, global location service or multi-protocol apps.

Pair-wise federation works for a few popular services, but does not scale with the growing number of WebRTC websites [5][6][7][8]. Lack of incentive to providers or less flexibility in server-side translation further hinders this approach. Projects like hookflash, &yet and matrix.org are emerging to provide global WebRTC signaling and location services. Convincing websites to use them or change apps to follow their APIs is hard; so they tend to form more isolated ecosystems. SigOfly [9] dynamically downloads the JavaScript code from the target's app provider for cross-service authentication and reachability, but requires the websites to use its APIs. Moreover, this approach does not work for installed mobile apps.

Pidgin and Trillian are multi-protocol apps. Due to lack of a signaling protocol specification in WebRTC – every site can implement its own call setup – such efforts are impractical with growth [7]. Both SIP and XMPP allow external protocol reachability [10][11], e.g., if lookup resolves to a mailto or http URL, the caller is redirected to open an email or web client. These are not popular in today's proxy-focused services. User specified reachability with time-of-day, calendar or presence [10][12], or fine-grained user preferences to select mobility or mode [13][14][15] are known. These existing systems based on multi-protocols reachability do not work when, say, a SIP provider allows only its own app or device to connect to its service. In practice, existing multi-protocols reachability is not the same as the desired multi-apps.

We conclude that we are in a multi-services and multi-apps environment which is very hard to interoperate or federate globally. Thus, solving user reachability with user driven apps in the endpoint is a viable option. Unlike pair-wise service federation, we let the user select her reachability apps (Fig.2); this freedom promotes innovation. Toutain et al [8] realize that users are overwhelmed by the number of communication apps and need a simple way to reach their contacts. They conclude that the user's contacts must be independent of the services. Unlike ours, there is no implementation, and it proposes to interoperate identity management to tie the user presence to the contact list. Our app does not include presence, and hence, with no global identity service, is easier to deploy or scale.

We use web-style code for call policy, unlike endpoint behavior in XML [12]. Our use of resource-based software architecture continues from [16][17][18]. The ability to launch external apps is inspired by the now discontinued webintents [19]; albeit extended beyond a single device using shared data. In summary, ours is a pragmatic way to deal with emerging WebRTC-based systems and covers multiple modes, devices and non-interoperable apps even if on the same protocol.

IV. System Architecture

A. Important definitions

Communication mode is one of video, phone or message. We use voice and phone interchangeably; phone does not mean a phone device, but may be a voice call on a softphone. Video or phone indicates real-time interaction. Message covers real-time as well as asynchronous apps, e.g., text chat vs. email, SMS, and voice/video messages. An app may have multiple modes, some limited by platform, e.g., no WebRTC video on Safari/iOS.

Communication app is typically a standalone application on desktop and/or mobile, or even in a browser. It may be limited by device or network, e.g., business IM only on VPN. An app is often tied to a service: a VoIP provider or hosted conference system. We use *service* and *app* interchangeably.

User reachability is defined as the ability to reach a user on one or more communication apps or devices. We also refer to email clients and phones as apps, although not controlled by our architecture. A *reachability item* is a triplet of *mode*, *app* and target *value*, e.g., VoIP address, phone number, or click-to-call or conference URL. A reachability list can have items on the same app or mode as shown below. The value is interpreted by the app, e.g., (1) could become tel:+18002223333,,13001234# in that conference service, and (3) could be sms:+14151234567.

```
(1) {"mode":"phone", "app":"Scopia", "value":"13001234"}
(2) {"mode":"phone", "app":"Phone", "value":"+14151234567"}
(3) {"mode":"message", "app":"Phone", "value":"+14151234567"}
```

B. *Separating contact list from communication apps*

This separation is crucial to support multiple diverse apps. Fig.3 shows that the user's contacts can be populated in many ways: by provider, managed by user, imported from mobile phone, or by user's context, e.g., discover other app users in a hotel guest room or emergency situations using local multicast (serverless), or discover other viewers of a website using a browser extension. A contact item may be static or dynamic. The latter changes its reachability by time or other factors. It may be a shared group contact, where reachability is for any, all or some of the group members, e.g., for customer support or group meetings. It may be auto-populated by data mining, e.g., of email/IM to find my frequent/recent contacts, or by phrase "I will get back"; or from an email thread or invite group.

We consider two types of apps: *dialers* and *communicators* (Fig.3). A dialer only does outbound interaction request. Once launched it does not return to the contact list, except on failure. A communicator can return to the contacts app for intermediate decisions or to apply policy on received requests.

C. *Cross-app interaction and handoff*

HTML5 registerProtocolHandler, Android web-intents or iOS app extension is used to open an external dialer, e.g, a mailto or tel URL open the native email or phone client. A communicator is either separate or integrated with the contacts to simplify inter-app messages. A provider's dialer app is often used as is. A communicator requires changes to separate the call setup and conversation; e.g., a softphone that informs the contacts app on incoming call, and proceeds on approval. The user may change the mode to message (Fig.8a), or move it to another app or device, informing the caller about the change. Consequently, three types of handoffs in Fig.4 are (a) device, (b) app, or (c) call component, e.g., move a call to the desktop phone, a video call from desktop to phone, or share desktop screen in a mobile call or add mobile touch-input white-board to a desktop call.

D. *Loosely coupled resource-based architecture*

Our architecture has loosely coupled independent apps with data-level mash-up. We use the resource service [17][18] hosted on Amazon EC2 for contacts and communicators. It is a simple web server supporting secure and authenticated data access and event notification (Fig.5). It stores pieces of data or resources in a file-like hierarchy without knowing its semantics. The app logic in the client defines the semantics, e.g., one app subscribes to a user's contacts at /users/{user}/contacts and another app updates, triggering an event to the first. The server transparently forwards end-to-end messages, e.g., to send call event or WebRTC signaling data. More details on mash-up, use of WebRTC, and user driven access control are in [17][18].

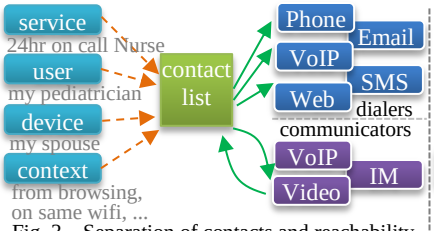


Fig. 3. Separation of contacts and reachability apps including dialers and communicators.

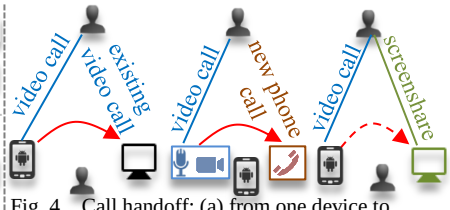


Fig. 4. Call handoff: (a) from one device to another, (b) from one app to another, and (c) call component handoff, e.g., for screenshare.

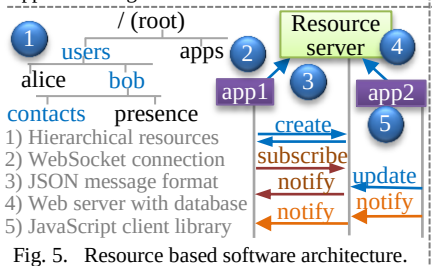


Fig. 5. Resource based software architecture.

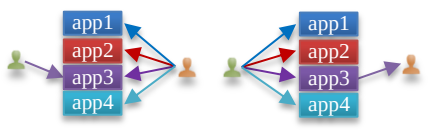


Fig. 6. Reachability: (left) in proactive presence, receiver publishes presence in apps, and the caller knows which to use before a call attempt, vs. (right) caller tries the apps on demand during call-setup until picked by the receiver.

Our client apps implement many scenarios: voice and video call, conferencing, text chat, contact list and user reachability. The contacts and communicator apps mash-up at the data level, e.g., for received call event, or caller policy access. Data namespaces enable multi-tenancy and app customization.

E. *Proactive presence vs. on-demand reachability*

We prefer on-demand reachability to active user presence, i.e., the caller side tries to reach the receiver's reachability items with fallbacks (Fig.6). There are many reasons for this decision as follows. (1) Relying on presence fails in a multi-apps environment because email, phone or conference bridge codes are always present. The question is not whether the user is available, but where. (2) Softphones supporting presence often use different protocols. (3) Presence systems scale poorly due to rich presence traffic, or periodic refresh of presence soft state on battery constrained devices. (4) An online status does not guarantee a call answer, and may require fallback.

Our desktop app uses persistent WebSocket, on which we may enable presence if needed. Our mobile app uses WebSocket only when the device is awake, but uses platform specific low power event channel when asleep, e.g., Google Cloud Messaging (GCM) on Android. Our contacts app does not use presence, but a launched app such as a third-party instant messenger can still use it internally to determine if the call will succeed. The on-demand and active presence are combined in practice.

F. *User reachability and fallback*

Fig.7 shows an example user reachability process when Richard tries to reach Kathy. The algorithm runs on the caller's Strata Top 9, but can instead be at the server. The first step is to resolve any dynamic contact, e.g., to get the next meeting for a calendar contact, or to map "customer service" to the currently reachable agent. The contact type defines the tool to resolve, e.g., to extract a bridge number data from calendar. This step is skipped as it is not a dynamic contact in this example.

Next, all target reachability items are fetched. This does not apply if a specific item is found in the previous step. Based on Kathy's email and phone number entered during signup, three default items are pre-populated: (1) the default communicator app for video, voice and text, (2) the phone app for voice call, and optionally mobile SMS, and (3) the email app for message. She may not be available now on Strata, or may be on many devices, or on her employer's apps based on Avaya IP office (ipoffice) or Messaging (amm). She has already configured all her items before. The items are ordered in decreasing preference: the default communicator (Vclick) first; phone and email

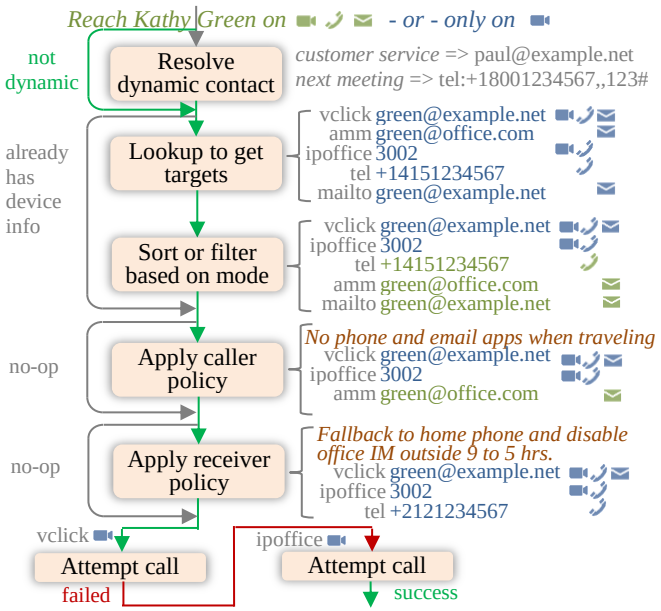


Fig. 7. Example user reachability process in Strata Top9

last; and other items (amm, ipoffice) in between. Receiver can modify the default values or their order if needed (Fig.8g).

Depending on the mode of the call attempt, the reachability items are sorted and filtered. Richard may initiate conversation in default mode, say video, by clicking on Kathy's contact, or select a specific mode, say video, by click-and-hold on her contact. The two cases behave differently in our app. The former falls back to other modes if Kathy is not reachable on video, but the latter fails if the selected mode is not in the list. In the former, the list is sorted for video, phone and message, in that order. In the latter, items without video are removed.

Next, the optional caller and receiver policies are applied. In this example, Richard's caller policy disallows using any phone or email apps when he is traveling, which filters out those from the list; and Kathy's receiver policy disallows employer's messenger and adds a last resort as her home number to reach her outside office hours.

Finally, the items are attempted with sequential fallback, e.g., if Kathy is offline on Strata (unreachable via Vclick), try video on ipoffice, and then a phone call to her home number (a mode fallback). Mode fallback is not done within the same app, e.g., if the Vclick video call fails, then do not reattempt voice or message on Vclick. However, an individual app may support mode handoff or transfer, e.g., an incoming video call in Vclick can be answered as a voice or message session (Fig.8a, right).

Typically, fallback can happen only if a call attempt on an app can return an error. This works for communicators and dialers that can return the result to the contacts app. It does not work for some dialers, e.g., native phone or email clients opened using a tel, sms or mailto URL. We have developed a modified phone dialer (Engagement Dialer) using our VoIP system that can return a result. Multiple line presence is the responsibility of the app, e.g., Vclick supports it and enables the user to run Strata on multiple devices, where the first one to answer is connected; whereas the ipoffice logs out the previous device when the user logs in from a new one.

G. User driven policies.

User customized reachability order (Fig.8g) works for most people. We also support programmable policy for finer control. The caller policy is applied to outbound request, and the receiver one to inbound. They are written in JavaScript-like code with only a few supported constructs as shown below.

(a) To reach my colleagues, prefer video and avoid my personal instant messenger.	`if (receiver.email =~ "*@office.com") {`
	`  prefer("video");`
	`  exclude("message", "AIM");`
	`}`

(b) Always call my cell after office hours irrespective of caller's preferred mode, and stop further policy lines.	`if (now.hh >= 17) {`
	`  choose("phone","Phone","+1212123456");`
	`  break;`
	`}`

(c) When I am traveling, only receive message mode; and fallback to my personal messenger service.	`if (location.address.country != "India") {`
	`  include("message");`
	`  deprecate("message", "AIM", "alice");`
	`}`

The script supports simple as well as nested if-else controls, and a break to stop further script processing. JSON (JavaScript Object Notation) objects representing the caller, receiver, current time and location, and comparators and regular expression are used in the policy decision. The caller and receiver objects contain the user attributes, e.g., user identity, name, preferred mode and contact's approval state. The now object has the current time in various formats, including time zone and UTC data. The location object has the current device location fetched using HTML5 and Google's geocoding APIs. The app does not retrieve the device's location unless the script uses location. Some examples are shown below.

caller	{"email": "bob@example.net", "name": "Bob Wilson", "type": "video", "state": "approved"}
receiver	{"email": "alice@office.com", "name": "Alice Smith", "type": "phone", "state": "pending"}
now	{"YYYY": 2015, …, "hh": 19, "mm": 38, …, "tz": "+07:00", "string": "2015-08-03 19:38:42", …, "utc": { "time": … }}
location	{"address": {"street_number": …, "locality": …, "state": "California", "country": "United States",…, "short": {"country": "US", …}}}

The script uses some functions to alter the behavior: include, exclude, prefer, deprecate and choose. Each function takes three parameters: mode, app and target value. Only choose requires all three, but others treat app and value as optional. These functions manipulate the reachability list shown in Fig.7. The choose function deletes the list, and adds only a single reachability item supplied in the function. The include and exclude functions filter the list to include only desired items or exclude undesired ones. If an optional parameter is missing, it acts as a wildcard, matching any item. The prefer and deprecate functions re-order the list to move certain items to the beginning (most preferred) or the end (least preferred). If all three parameters are supplied, then these two functions also act as a way to inject one reachability item at the beginning or end of the list.

Note that these policies apply only during initiation, not in an active call. The policy engine is currently in the Strata Top9 app, and hence, only used if the call is initiated or received by this app. We have web-based policy script editing, and in future will have graphical interface with drag-and-drop editing.

V. Implementation of Cross Platform Apps

Strata Top9 is an enterprise app for desktop and mobile to quickly connect with any of the user's top 9 contacts. Both in personal as well as business communications, people often initiate conversation with only a handful of contacts on a regular basis or attend online meetings based on a few workflows, e.g., from their calendar events. Since it is easy to change a contact slot, the limit of 9 is not an issue for many people. It allows an aesthetically sound 3x3 layout on a phone (Fig.1), but can be changed to 4x4 or, in landscape mode, 4x2. Moreover, additional contacts can appear in subsequent pages beyond the first Top 9 page. The user adds a new contact in an empty slot, or changes an existing one. If the target is not a Strata user, it allows inviting via email. The Top 9 page includes person and non-person contacts. The received page only shows people, i.e., those who sent me a contact request. The receiver accepts or declines the contact request, or changes the approval state at any time on the received page. The caller and receiver can set a preferred mode independently. A non-person contact can use any app and does not need approval, e.g., next meeting in calendar or specific dial-in bridge (Fig.1). The same contact can be in multiple slots, e.g., for different preferred modes, or for calendar contacts, to show multiple upcoming meetings.

The user manages her reachability (Fig.8g) to enable others to reach her when offline on Strata. Strata supports many apps (Fig.8f), and more can easily be added. The reachability data is stored in the resource server. The app data, e.g., IP office login credentials, are in device's local storage, so that separate app instances on different devices can be customized, e.g., to use IP office only on the work PC but not the personal tablet.

We developed Strata and other WebRTC apps in HTML, JavaScript and CSS, and using ChromeApp and Apache Cordova tools and frameworks [20], ported to native apps on desktop as well as mobile. The desktop native app runs using the Chrome browser's native client plugin. The Cordova framework converts the web app to a native mobile app, to be uploaded to Google Play Store (Android) or Apple App Store (iOS). WebRTC is included by Cordova on Android. We use cordova-plugin-iosrtc for WebRTC on iOS. Our implemented apps are listed below, and some are shown in Fig.8.

Default communicator using Vclick: Vclick [18] is a collection of loosely-coupled apps that mash up using the resource server and are independent of legacy VoIP systems. The realization in Strata includes only a subset of apps – for text chat with optional attachments and speech/text translation, and full mesh WebRTC-based voice/video calls and conferences (Fig.8a).

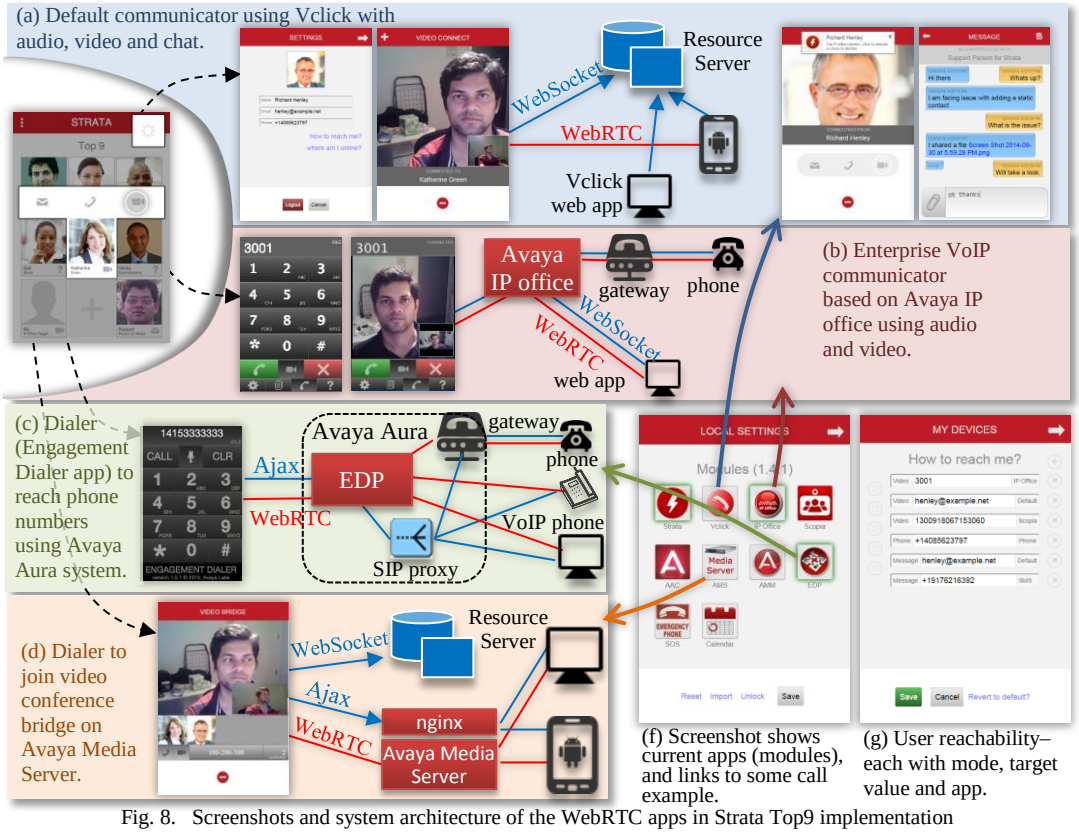
IP office phone: Avaya IP office is a VoIP system for small and midsize businesses. We built IP office phone, a communicator app, to connect to this VoIP system to make or receive voice or video calls. Besides the separate app (Fig.8b), an integrated-to-Strata version is implemented. Unlike the peer-to-peer media path of Vclick, it anchors the media path at the server.

Engagement dialer: It allows dialing out a phone number using the enterprise or cloud VoIP service of Avaya's Engagement Development Platform (EDP) and Aura software suite (Fig.8c). It handles the tel URLs including optional pauses and DTMF digits, e.g., tel:+18001234567,,,123#. Thus, Strata can use this to reach phone numbers or conference bridges, e.g., from tablets or desktops. If the target value of the contact is empty, it opens a generic phone dialer, allowing the user to enter the target number. This avoids having to add a one-time phone number in contacts.

Media Server (AMS) app: AMS allows multi-party audio and video conference using RESTful APIs for control and WebRTC for media. It does audio mixing and video switching based on active speaker. We built an AMS dialer app that uses the resource server to manage conference membership and moderator information, and to join the video bridge, without any legacy VoIP signaling.

Multimedia Messaging (AMM) app: AMM also has RESTful APIs to enable multi-party messaging. We built an AMM communicator app to send and receive text messages from Strata.

Next meeting/Calendar: The calendar app uses a light-



(f) Screenshot shows current apps (modules), and links to some call example.

(g) User reachability–each with mode, target value and app.

Fig. 8. Screenshots and system architecture of the WebRTC apps in Strata Top9 implementation

weight proxy to periodically fetch the user's calendar from her enterprise mail exchange server, and displays one or more ongoing or upcoming meetings. The picture cycles through multiple overlapping meetings if needed, and allows click to join via video or phone, instead of a manual dial-in of bridge number and code. This dynamic contact maps to a reachability item on AAC, Scopia or phone depending on the meeting data. AAC and Scopia represent a series of Avaya UC products for audio/video conferencing and online collaboration.

Furthermore, Strata can launch existing apps, e.g., email, phone, third-party Jabber apps, or conference client apps of AAC and Scopia, or can join their voice bridges.

VI. Conclusions and Future Work

We have described the user reachability problem and how it is aggravated in WebRTC-based multi-apps environment. To solve the problem, we have presented the architecture and implementation of our multi-platform system consisting of separate contacts and communication apps. Our Strata Top9 app covers many WebRTC-based cross-platform apps for VoIP and multimedia conferences. The white-labeled Strata app can be customized for specific businesses. Many of our apps are focused on enterprise use cases, but the flexible architecture can include other social apps, e.g., we have created separate mobile apps for SIP-in-JavaScript and LAN video phone to connect to public VoIP service, and to discover and connect to others in the same local area network, respectively. We are also modifying the Vclick webapp to inject dynamic contacts from the browsing context to the Strata app, e.g., to show who else is viewing the department webpage.

Our work shows that many useful features such as user reachability and handoff across devices or apps are possible with user driven apps. We focus on user driven reachability and policy decisions, unlike a global location service or pair-wise federations to make our system useful in practice for emerging WebRTC apps. Endpoint driven apps are also useful when local context is needed, e.g., user's location in dialing out an emergency call. Separate resource servers can be used for different groups or organizations. Our resource oriented software architecture allows an app to dynamically pick the data server independent of where the app is loaded from.

In the future, instead of exposing the reachability items to the caller app, we will create a server side policy engine that will hide any sensitive data. The policy engine could use other contextual data, e.g., input from GPS could indicate driving, and thus, disallow message or allow only hands-free call; underlying network with or without VPN, could affect the call security requirement and disable certain apps; received call could be transferred to a recordable bridge for accounting or if calendar shows a shared meeting; mobile data usage could be used to downgrade a video call to a low bandwidth voice; or background noise level could disallow a voice call, or trigger speech/text translation. Privacy of such detailed contextual input is paramount. Thus, a server side policy engine to aggregate and/or filter sensitive data is preferred.

Acknowledgments

The Strata Top9 project is a joint work with Steve Brock, Joyce Fong, Venkatesh Krishnaswmany and Laurent Philonenko. The following people have helped in integration or evaluation of some of our implemented apps: Biswajyoti Pal, Thiru Arjunan, Jaydeep Bhalerao, Gaurav Badge, Ramanuja Kashi, and Stephen Whynot.

References

[1] J. Grégoire, "On embedded real-time media communications", Proceedings of the 1st workshop on all-web real-time systems, Bordeaux, France, Apr 2015.

[2] N. Paterson, "Walled gardens: the new shape of the public Internet", Proceedings of the 2012 iConference, ACM, 2012.

[3] R. Tworeck, "The walled garden in reverse – open web", Online, Mar 2013, http://webrtcstrategies.com/, retrieved Aug 2015.

[4] A.B. Johnston and D.C. Burnett, WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, third edition, Digital Codex, 2014, ISBN 978-0985978860.

[5] L. Strand and W. Leister, "A survey of SIP peering", In proceedings of NATO architects of secure networks (ASIGE), May 2010, Genova, Italy.

[6] P. Saint-Andre, "XMPP protocol flows for inter-domain federation", XEP-0238, XMPP standards foundation, 2008.

[7] Sites that use or demo WebRTC, Online, http://www.webrtcworld.com/webrtc-list.aspx, retrieved Aug 2015.

[8] F. Toutain, E. Huérou and E. Beaufils, "On webco interoperability", Proceedings of the 1st workshop on all-web real-time systems, Bordeaux, France, Apr 2015.

[9] K. Hänsge, S. Drüsedow, P. Chainho, M. Maruschke, "Signalling-On-the-fly", in Innovations in Services, Networks and Clouds (ICIN 2015), Paris, France, Feb 2015.

[10] J. Rosenberg et al., "SIP: session initiation protocol", RFC 3261, IETF, Jun 2002.

[11] P. Saint-Andre, J. Hildebrand, "reachability addresses", XEP-0152, XMPP standards foundation, 2014.

[12] X. Wu, H.Schulzrinne, "Programming end system services using SIP", IEEE international conference on communications (ICC), Anchorage, Alaska, May 2003.

[13] J. Rosenberg, H. Schulzrinne, P. Kyzivat, "Caller preferences for SIP", RFC 3841, IETF, Aug 20014.

[14] M. Boussard et al., "Communication hyperlinks: call me my way", 13th international conference on intelligence in next generation networks, (ICIN), 2009, Bordeaux, France.

[15] S. Shanmugalingam, N. Crespi, P. Labrogere, "My own communication service provider", International congress on ultra modern telecommunications and control systems and workshop, 2010, Moscow.

[16] C.Davids et al., "SIP APIs for voice and video communications on the web", International conference on principles, systems and applications of IP telecommunications (IPTcomm), Wheaton, IL, Aug 2011.

[17] K. Singh and V. Krishnaswamy, "Building communicating web applications leveraging endpoints and cloud resource service", IEEE International Conference on Cloud Computing, Santa Clara, CA, Jun-Jul 2013.

[18] K. Singh and J. Yoakum, "Vclick: endpoint driven enterprise WebRTC", (to appear in) IEEE International Symposium on Multimedia (IEEE ISM), Miami, FL, Dec 2015.

[19] G. Billock, J. Hawkins, P. Kinlan, "Web Intents", W3C draft, 2013, http://www.w3.org/TR/web-intents/.

[20] Run Chrome Apps on mobile using Apache Cordova, https://developer.chrome.com/apps/chrome_apps_on_mobile, accessed Jul 2015.