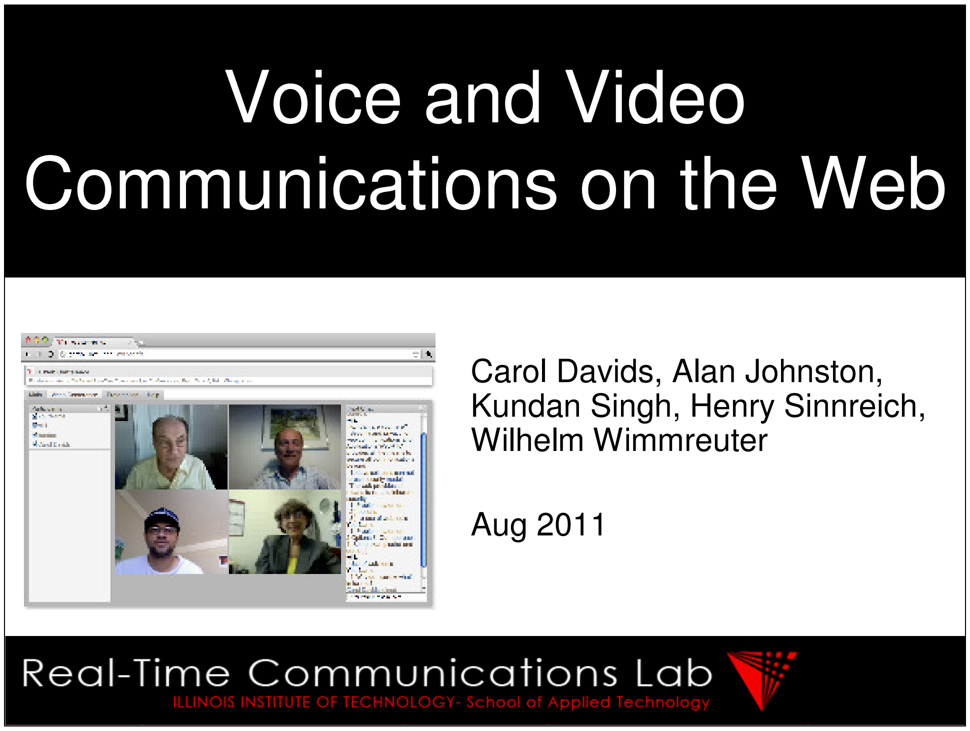
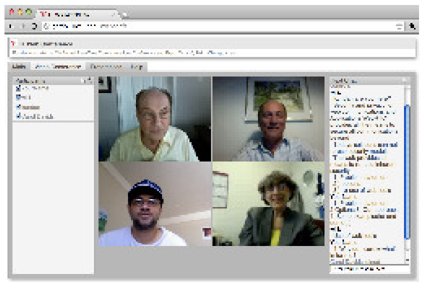


Voice and Video Communications on the Web



Carol Davids, Alan Johnston, Kundan Singh, Henry Sinnreich, Wilhelm Wimmreuter

Aug 2011

Real-Time Communications Lab
ILLINOIS INSTITUTE OF TECHNOLOGY- School of Applied Technology

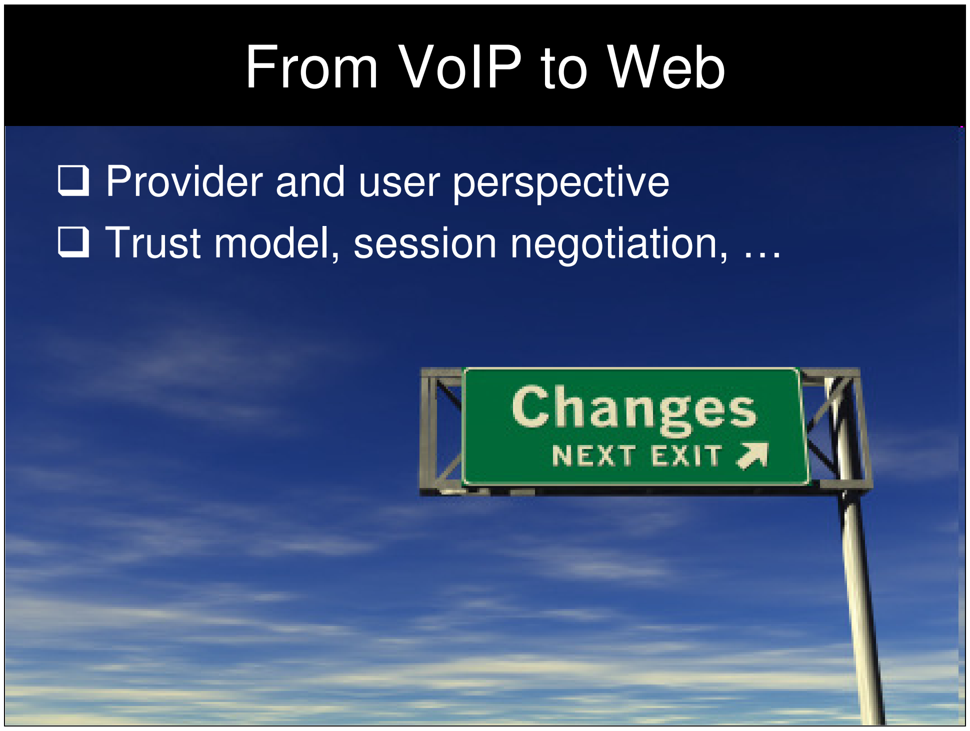
I will talk about our short paper on SIP APIs for voice and video communications on the web. This is a joint work with Carol, Alan, Henry and Wili with support from students at real-time communications lab. In particular Isioma and Harinath helped in implementing part of this project.

I will talk about the motivation and challenges of web communications in comparison with traditional VoIP. Then I will describe the available platform options such as modifying the browser, using a plugin or a separate host application. Finally, I will describe the components of our project along with a brief demonstration.

Web and VoIP have evolved into separate islands of innovations. There is a very strong motivation to bridge the gap to enable the millions of web developers to include communications on web pages. Seamless integration of web and communications will result in many more innovative applications. Due to separate islands of innovations, there are different protocols, programming language APIs, developer tools, and developer communities.

Web companies are fast paced, with quick turnaround, and build easy to use application whereas VoIP is traditionally linked to complex protocol machinery which is difficult to get right across different equipments.

Critical voice and video programming primitives are also missing in web: UDP transport, listening socket, native device access.



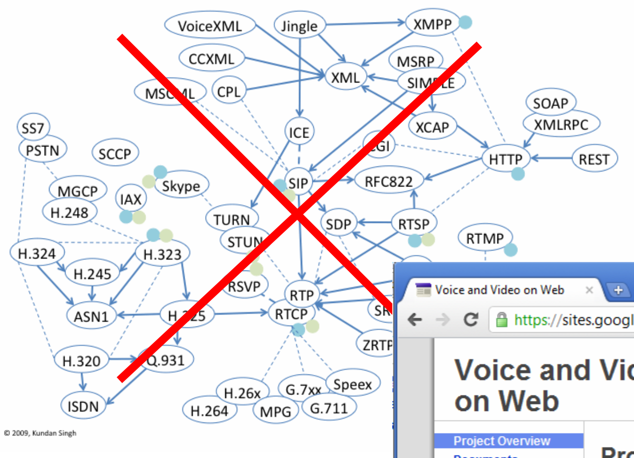
From VoIP to Web

- ❑ Provider and user perspective
- ❑ Trust model, session negotiation, …

Changes
NEXT EXIT

Moving from traditional VoIP to Web communications requires lot of changes. Firstly the provider perspective changes. Typically a web site owner wants to own the content and customer interactions, unlike a VoIP provider who should provide connectivity to other VoIP networks as well. From user's perspective, unlike a software rendition of a phone or a phone book, web allows communication to be part of existing web browsing experience. For example, embedded click to call, auto-conference among current visitors on a page, etc. Trust model is different in web. The way session is negotiated is different because unlike offer/answer model of VoIP here the publisher may advertise the session and anyone visiting a web page automatically joins instead of explicit answer.

Minimum Protocol Requirements

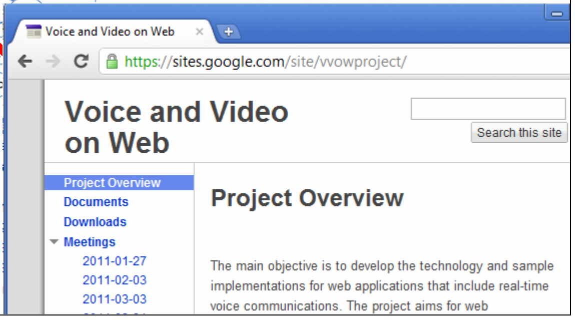


HTTP

signaling and control

UDP

media transport


Voice and Video on Web

https://sites.google.com/site/vvowproject/

Voice and Video on Web

Search this site

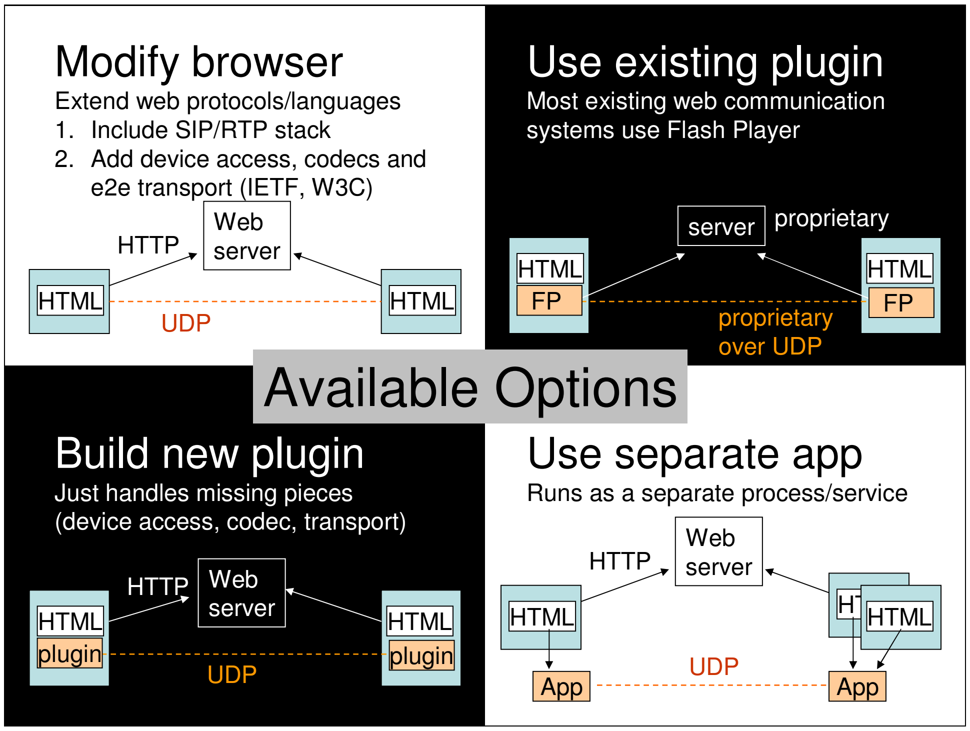
Project Overview
Documents
Downloads
Meetings
2011-01-27
2011-02-03
2011-03-03

Project Overview

The main objective is to develop the technology and sample implementations for web applications that include real-time voice communications. The project aims for web

While there are many different protocols for multimedia communications, you just need two protocols on the web: HTTP for signaling and control such as discovering other people on the web page and sending invitation to connect, and UDP to do low latency real-time media transport. All other services, protocols and mechanisms are outside in the application space. This was the main motivation to start our project on voice and video on web.

Understanding the minimum requirements is useful in designing the solution. It helps in keeping unwanted stuff outside the scope.



Available Options

Modify browser

Extend web protocols/languages

1. Include SIP/RTP stack
2. Add device access, codecs and e2e transport (IETF, W3C)

Use existing plugin

Most existing web communication systems use Flash Player

Build new plugin

Just handles missing pieces (device access, codec, transport)

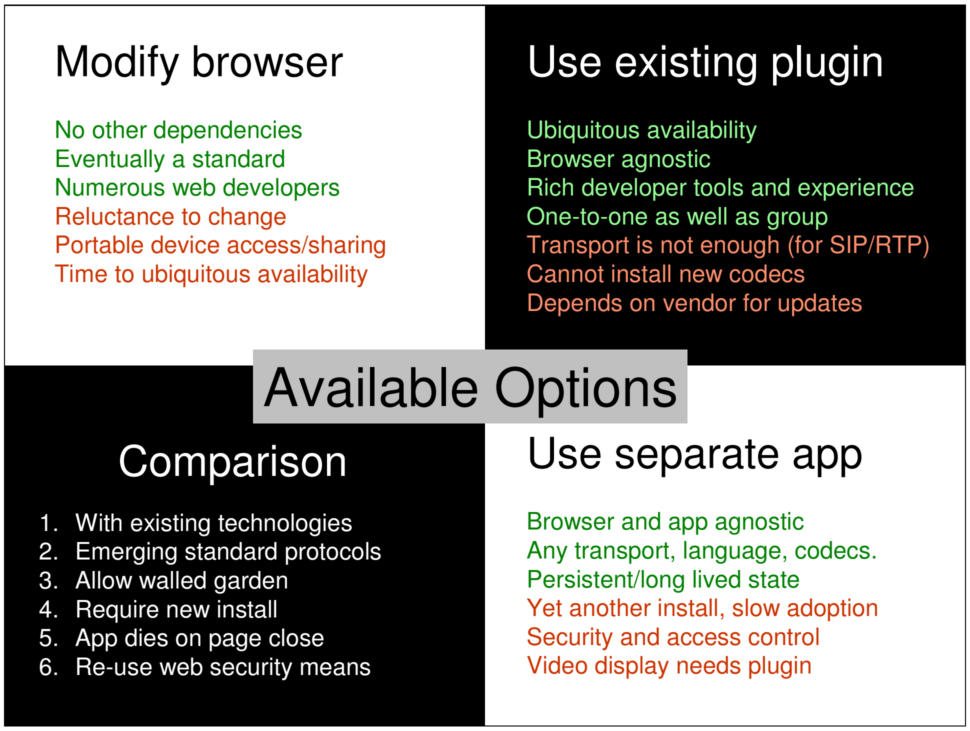
Use separate app

Runs as a separate process/service

At the high level there are four alternatives to enable voice and video communications in the browser: (1) modify the browser to use new protocols and programming API, (2) use an existing plugin that provides end-to-end media path and device access, (3) build a new plugin to provide the missing pieces in the browser such as device access, codec, transport, and finally (4) use a separate application that runs on user's computer, and allows the web browser as well as other applications to enable end-to-end media path and device access.

In the first option, you can either include a complete SIP/RTP stack in a browser or add only the missing pieces. The new working groups at IETF and W3C are focused on this effort. Including a full SIP/RTP stack is possible but presents difficult challenges in terms of agreeing on common API and interoperability among multiple implementations. The signaling is better left to HTTP and web protocols instead of trying to use SIP in this context.

Existing web-based communication systems have largely built upon Flash Player (or Silverlight) browser plugin. Flash Player allows end-to-end media path using a proprietary protocol (RTMFP) over UDP.



Available Options

Modify browser

No other dependencies
Eventually a standard
Numerous web developers
Reluctance to change
Portable device access/sharing
Time to ubiquitous availability

Use existing plugin

Ubiquitous availability
Browser agnostic
Rich developer tools and experience
One-to-one as well as group
Transport is not enough (for SIP/RTP)
Cannot install new codecs
Depends on vendor for updates

Comparison

1. With existing technologies
2. Emerging standard protocols
3. Allow walled garden
4. Require new install
5. App dies on page close
6. Re-use web security means

Use separate app

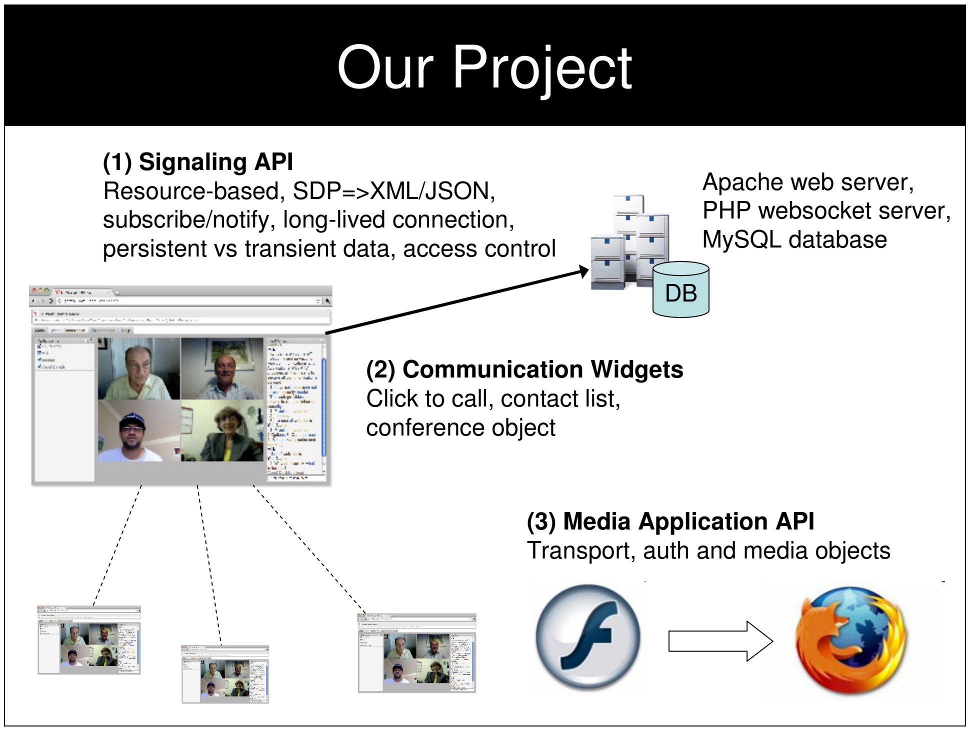
Browser and app agnostic
Any transport, language, codecs.
Persistent/long lived state
Yet another install, slow adoption
Security and access control
Video display needs plugin

This slide compares the various options. The green lines are advantages and red ones disadvantages. Modifying the browser to adopt the emerging standards is the ideal solution in the long run. It doesn't have additional dependencies on plugins or other applications. However, typically changing the browser for new standards takes time, and much more time before the feature is ubiquitously available to many common browsers.

Traditionally, plugins such as Flash Player and silverlight have filled the lack of real-time support in the browser. The main advantage of Flash Player is that it is already available on most PCs and thus do not require additional installation. Moreover availability of rich developer tools and user interface experience makes it a good choice. Same application code works on all browsers, instead of having to write a lot of browser dependent hacks. The main problem is that developers and users are dependent on plugin vendor for updates such as for security or new features. Secondly the existing programming primitives in Flash do not allow implementing a full SIP/RTP stack or installing new codecs. Building a separate plugin solves some of these problems but the challenges of portability across all platforms and all browsers makes it a tough answer to the problem.

Using a separate application is not only browser agnostic but can also be used by other host applications. Unlike plugins or web page's DOM states, a separate application can keep persistent and long lived states. For example, existing solutions such as Host Identity Protocol for NAT traversal, mobility and multihoming can be easily incorporated. NAT ports can be pre-detected to speed up connection setup. Going from one web page to another within the same domain can easily preserve sessions. The main problem is that a new installation slows the adoption among end users. Allowing access some multiple competing web pages or browsers require careful security and access control mechanism. Finally video display needs some plugin presence in the browser for immersive experience.

These options can also be compared with criteria such as it can built using existing technologies (no, yes, yes), whether it can use emerging standards protocols (yes, no, yes), whether building a walled garden is easy (no, yes, no), whether a new installation is required (no, yes, yes), whether session dies on page close (yes, yes, no) and whether existing web security can be reused (yes, yes, no).



Our Project

(1) Signaling API
Resource-based, SDP=>XML/JSON, subscribe/notify, long-lived connection, persistent vs transient data, access control

Apache web server, PHP websocket server, MySQL database

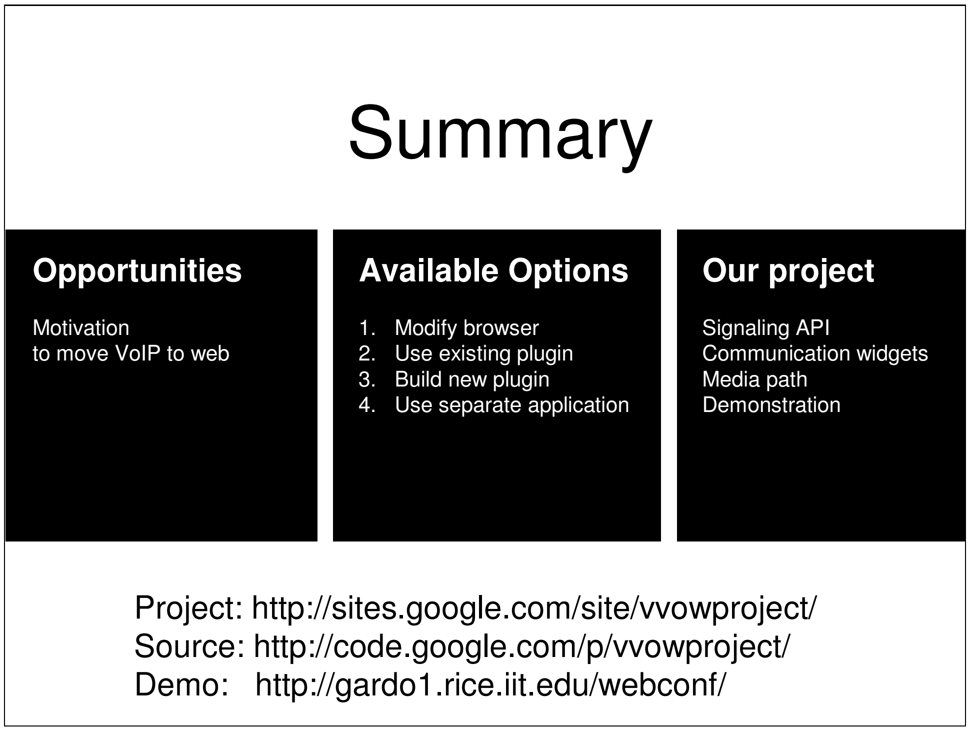
(2) Communication Widgets
Click to call, contact list, conference object

(3) Media Application API
Transport, auth and media objects

Our project has three parts: (1) signaling, (2) communication widgets, and (3) media application API. This applies to other web communication applications as well. The signaling API defines how people discover each other and how session is negotiated. Essentially this is a web version of SIP/SDP. As discussed earlier, the idea is to use web oriented protocol such as HTTP. All client server interaction happens over HTTP. The various data model inspired by SIP systems such as online users, and call state, are maintained as resources. The resource-oriented (unlike service oriented) API allows for more scalable system with complex logic inside the client. For example, list of logged in users are at /login, so doing a PUT or POST under that URL will allow an end-user to login, and be discovered by others. The session parameters are represented using web oriented formats such as XML and JSON. Finally, to receive events from other users or from web server, a subscribe/notify mechanism is implemented on top of long-lived websocket connection. The way it works is that a client can subscribe to changes in a particular resource, say /call/call123 and be notified whenever this resource or its immediate children change. (Show the message flow in Chrome during the demo). We have implemented a generic RESTful client-server API with these features so that the application can define its own resources for web communication.

The second part is the commonly used communication widgets. When the browser comes up it uses a communication widget (group of HTML, Javascript, Flash files) that implement a particular application. The widget connects to the signaling server to enable rendezvous as well as connects, detects, installs local separate application to handle media path.

The third part is the separate application with some media application API that receives commands from the web pages, and performs device access, media transport and codecs. In our preliminary implementation we have used Flash Player as an intermediate solution until we implement the separate application. It should be noted that the different parts can be used independent of each other, e.g., the media part can first detect if WebRTC extensions are available natively in the browser, and if not fall back to plugin or separate application approach.



Summary

Opportunities

Motivation
to move VoIP to web

Available Options

1. Modify browser
2. Use existing plugin
3. Build new plugin
4. Use separate application

Our project

Signaling API
Communication widgets
Media path
Demonstration

Project: http://sites.google.com/site/vvowproject/
Source: http://code.google.com/p/vvowproject/
Demo: http://gardo1.rice.iit.edu/webconf/

We have described the challenges of moving from VoIP to web based communication, compared available platform options, and described the architecture of our project. The project is open source and we are happy to accept your contributions. Further information are available on these web sites.