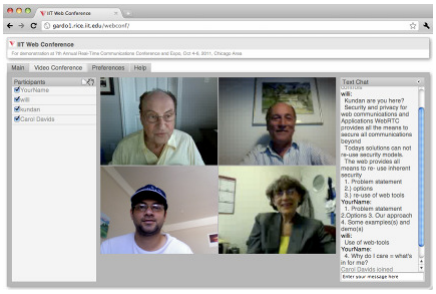
# Hello 1 2 3, can you ~~hear~~ see me now?

Internet video communication: past, present and future



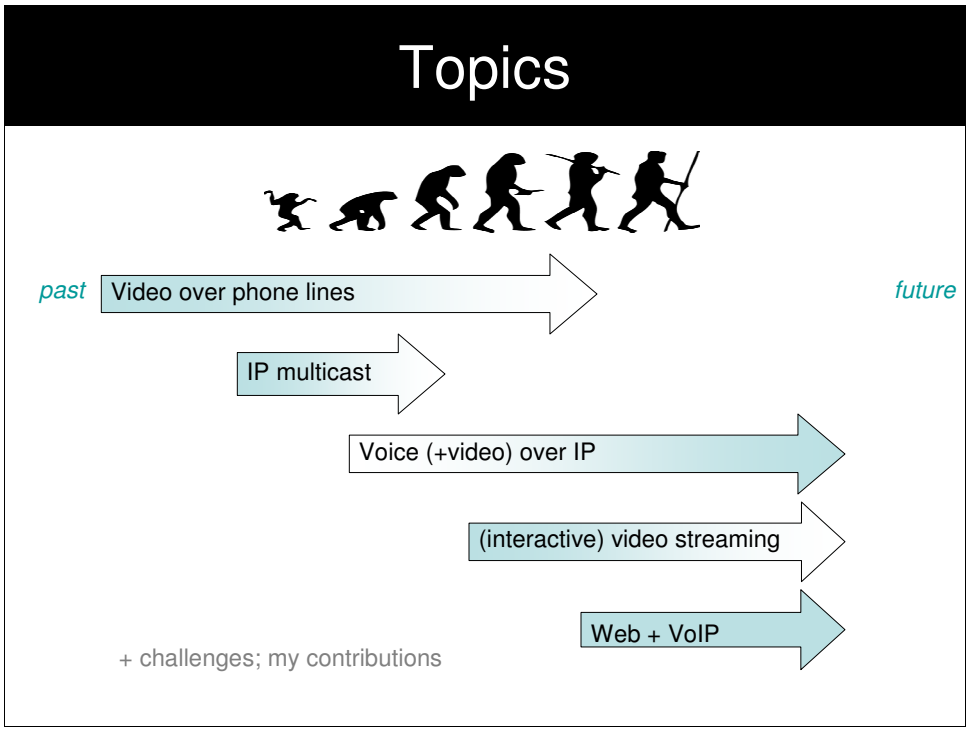
Kundan Singh, PhD
Nov 2011



Abstract:

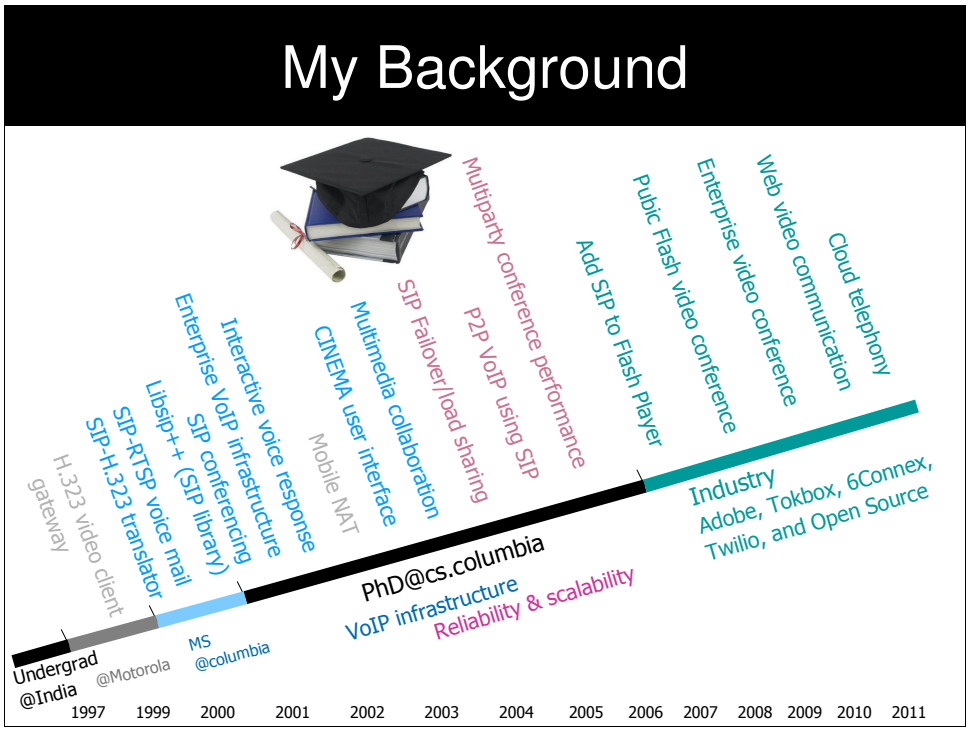
Modern video communication systems have roots in several technologies: transporting video over phone lines, using multicast on Internet2's Mbone, adding video to voice-over-IP (VoIP), and adding interactivity in existing streaming applications. Although the Internet telephony and multimedia communication protocols have matured over the last fifteen years, they are largely being used for interconnectivity among closed networks of telecom services. Recently, the world wide web has evolved as a popular platform for everything we do on the Internet including email, text chat, voice calls, discussions, enterprise applications and multi-party collaboration. Unfortunately, there is a disconnect between the web and traditional Internet telephony protocols as they have ignored the constraints and requirements of each other. Consequently, Adobe's Flash Player is being used as a web browser plugin by many developers for voice and video calls over the web.

Learning from the mistakes of the past and knowing where we stand at present will help us build the Internet video communication systems of the future. I present my point of view on the evolution, challenges and mistakes of the past, and, moving forward, describe the challenges in bridging the gap between web and VoIP. I highlight my contributions at various stages in the journey of Internet audio/video communication protocols.

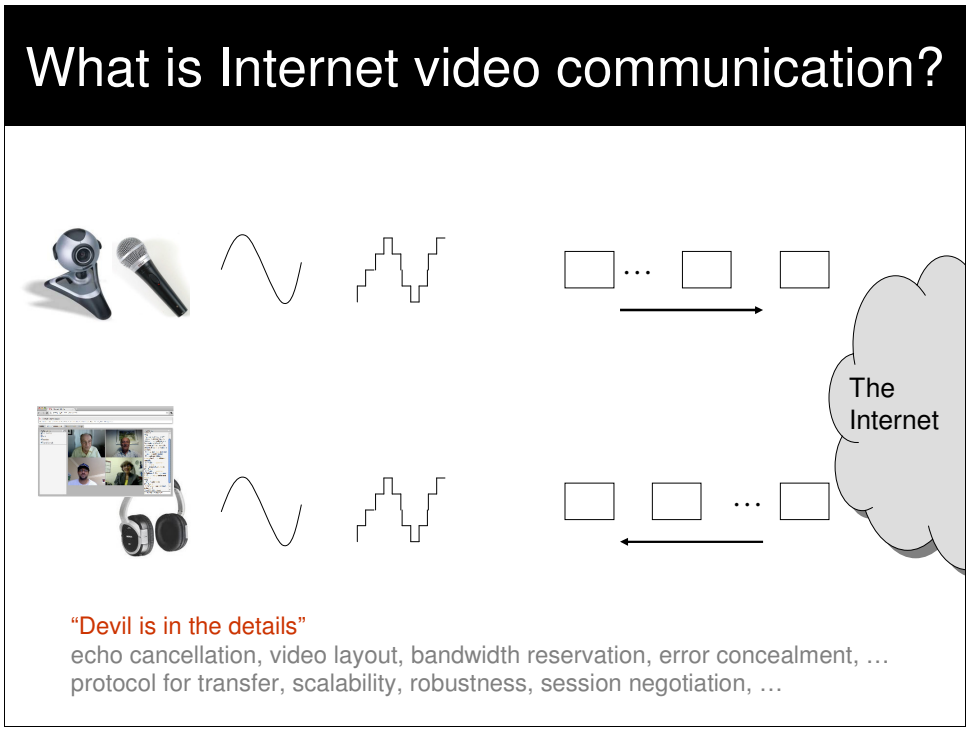



In this presentation, I will talk about five main topics on what influenced the evolution of Internet audio and video communication protocols. It started with transporting pictures or video over phone lines. The then existing telephony protocols were modified to carry video bits. With the innovations in the Internet Protocol (IP) such as multicast, researchers started building audio and video conferencing tools for IP multicast. To facilitate user lookup and discovery in an Internet oriented way, voice-over-IP (VoIP) protocol names the Session Initiation Protocol (SIP) evolved. Video was yet another media to carry without having to modify the core of SIP. At the same time innovations in Internet related to video streaming and web exploded the video consumption on the Internet via web sites like YouTube and web browser plugins such as Adobe Flash Player. Adding interactivity to Flash Player allowed building video conferencing on the web. However, closed nature of the plugin forced the standards bodies to look in to bringing standards for real-time communication to web.

Different origins of the evolution solved different problems and contributed to different challenges. I have had the opportunity to work in many of these areas and during my presentation I will highlight my contributions on these topics. I will focus more on my recent work on the last two topics related to web-based video communication.

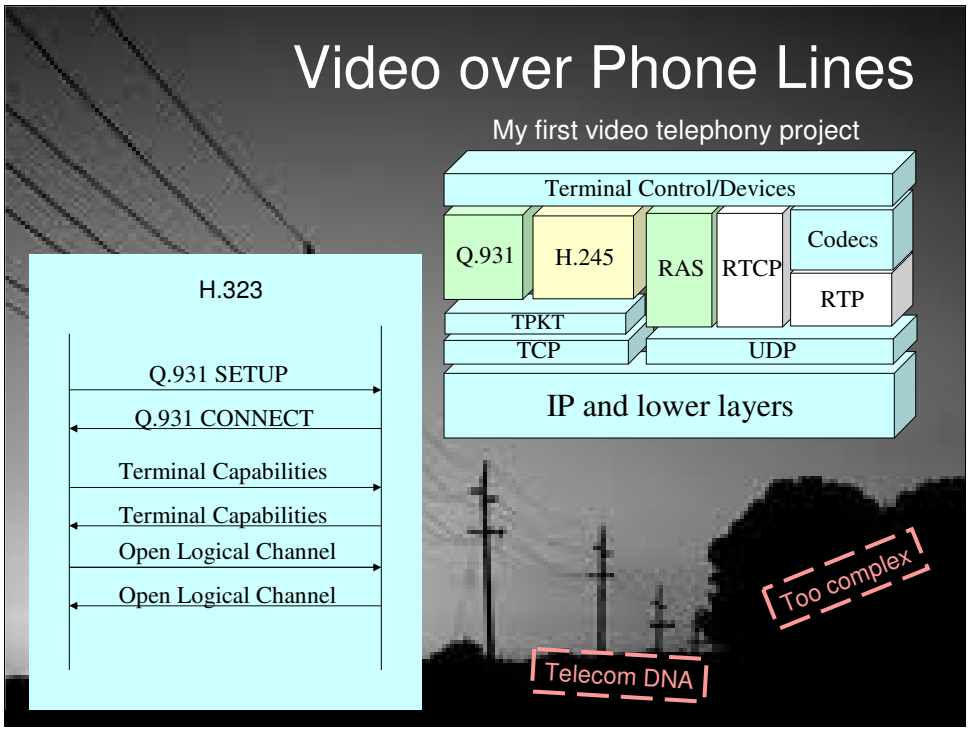



One of the first project I did was building an H.323 video phone for LAN. During my time at Columbia University with Prof. Henning Schulzrinne, I contributed to several systems projects on SIP-based applications such as translating between SIP and H.323, doing multiparty conferencing and collaboration, enabling server-less SIP network using peer-to-peer algorithms. In the last five years I have focused mostly on web-based audio and video communication systems in various industry projects as well as in my open source projects.




Before jumping on to the topics let us look at what Internet video communication really is? It is essentially capturing audio and video from one host, digitizing them, transporting the real-time data from one IP address to another over the Internet, and playing the media on the other host. However there are a number of details that go behind the scene to make this simple flow really work well in practice. For instance various audio quality processing that improve the clarity of speech, reduce the bit rate needed on the network, reduce the effect of noise and echo from surroundings, efficiently handle packet losses and variable delay on the Internet while maintaining real time nature of the flow, multiplexing or synchronization of various media, robustness against failures of intermediate network elements, negotiation of audio and video configuration parameters so that both ends can understand each other and determine that they can use the best possible configuration under the given practical scenario, laying out or mixing media from multiple participants in a conference, and so on.

My past focus has been in the signaling and network side of the system, but there are a number of other media related things that are equally important.



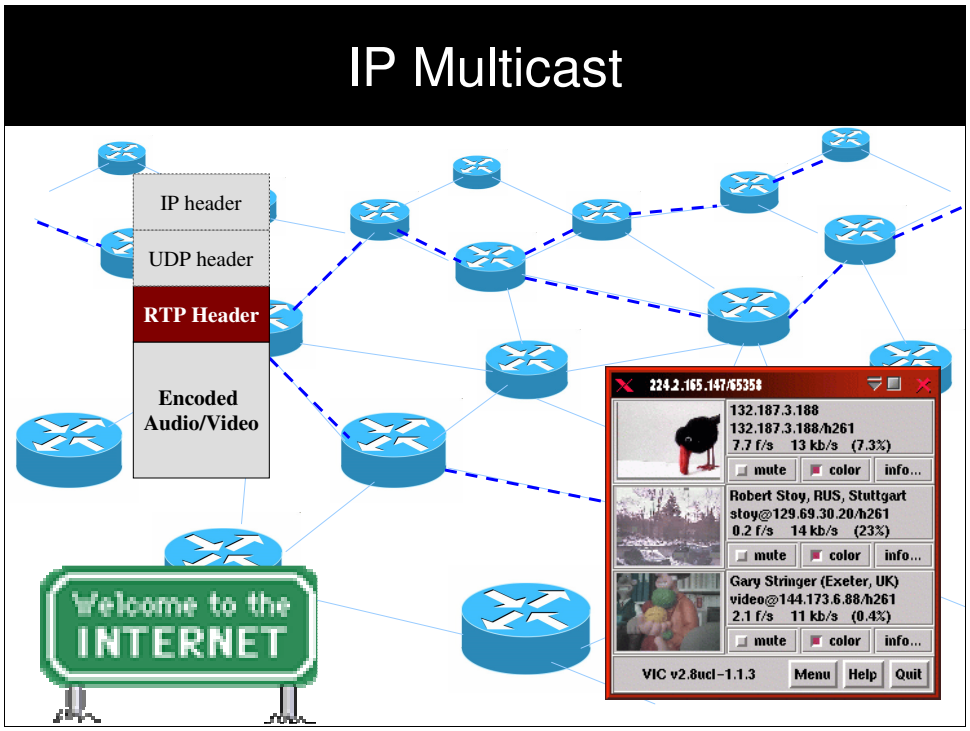

Audio video communication is not new. As early as 1960's we had picture phones as they were called. And today we have dozens of brands of video phone equipments to choose from. Many of these early video phone systems worked on the philosophy of carrying encoded video bits over phone lines, or some connected circuit. ITU-T created recommendations for video terminals over ISDN and modem connections. The H.320 multimedia systems are still popular for room based video conferencing. The same of set of signaling and session negotiation protocol is applied to packet switched network as well. For example, ITU-T H.323 started out as the video communication protocol for local area network but many of the crucial pieces such as Q.931 and H.264 were borrowed from the previous circuit switched network based protocols.

Although the H.323 protocol has evolved over the years, the first version defined pretty complete but complex set of primitives for establishing the call, negotiating the terminal capabilities, and creating the media channels for audio, video and text. First the terminals would discover each other via a registration, admission and status to an entity called the gatekeeper. Then they send some telephone call signaling messages over TCP to establish a call. This is followed by negotiating terminal capabilities and establishing individual logical channels for audio, video and data as needed.

The main problem it tried to solve was to enable existing video terminals over yet another network while keeping backward compatibility with existing networks like ISDN, so reusing the existing primitives made sense. However, since Internet engineers had already felt the success of Internet application protocols such as HTTP for web, SMTP for email and FTP for file transfer, this new protocol was too complex to digest. It inherited many of the telecom stuff without adding value over the Internet, and the Internet part was just a small dot in the system, e.g., use of RTP and RTCP for transport of media.

*Once you have the hammer, everything looks like a nail*

H.323 suffered from the hammer-and-nail syndrome. The ITU-T already had the complex video communication specification, and they saw the Internet's packet switched network as yet another nail to be nailed. They could not see the real Internet problems and philosophy such as openness and simplicity of the Internet application protocols. It was more like using a sledge hammer, too bulky and too complex, to fix a screw.




With the popularity of Internet Protocol and innovations such as IP multicast, researchers started experimenting with desktop video conferences over multicast test bed, Mbone, on Internet2, a network of research and academic institutes. Tools such as robust audio tool (rat) and video conferencing tool (vic) allowed very simple multiparty audio and video conferences. Application level transport protocol, RTP, was invented to carry sequence and timing information that are needed to correctly playback real-time media, for occasional feedback of the quality of service, and time synchronization of multiple media.


Although these multicast-based applications embraced very simple text based session description protocol for describing the multicast sessions, they could not be used on general Internet where multicast was not available. Presence of middle boxes such as NATs and firewalls, and lack of global multicast broke the premises on which these multicast video conferencing tools were built. I wouldn't call these attempts as failed attempts for video communication because they inspired subsequent Internet engineers to create better protocols while keeping the core benefits of the simple real-time transport protocol and simple text based signaling protocol.

It was clear that text based simple session protocols were going to stay… and would work hand-in-hand with other Internet application protocols like HTTP and email.


One of the problems in simple RTP based multicast conferencing was how to discover the other participants to ask them to join. While H.323 was available, it required huge telecom investment to work on H.323. If you were not already a telecom player with an existing telecom stack, it was very difficult for you to nail that screw I talked about earlier. So SIP evolved as a very simple replacement to H.323 to solve two problems: (1) how to discover other parties and invite them to a session, and (2) how to negotiate any session parameters. It re-used existing session description protocol and existing Internet technologies such as DNS, stateless core, robustness against failures, etc.

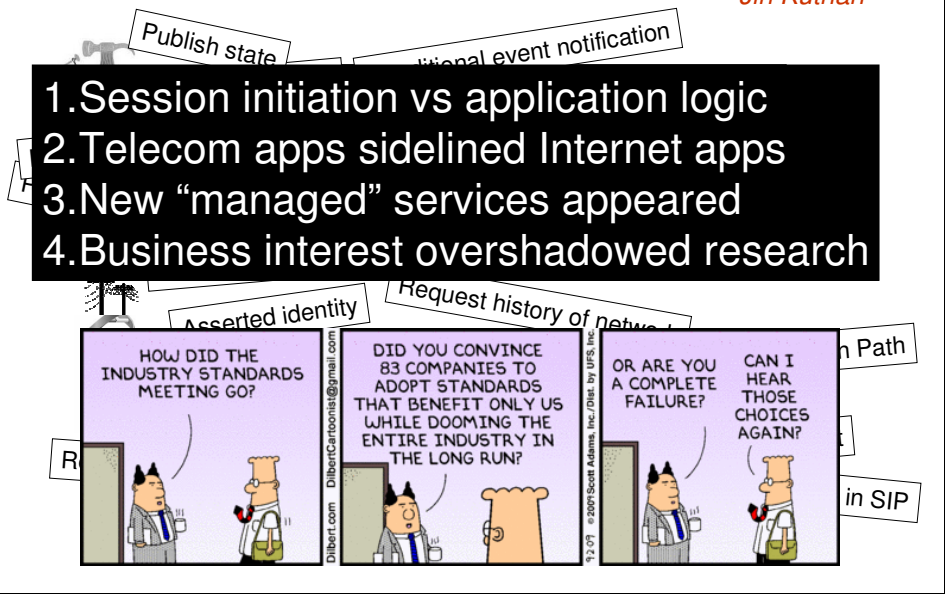
When I first started working on SIP, I was very thrilled because compared to H.323 where any minor feature needed planning for several weeks or months, the complete SIP/RTP stack could be prototyped in a class assignment. It was that simple.

Being second, SIP had to face many hurdles from the telecom mindset but eventually it survived. While video was always there, it was sidelined in real implementations. Video was considered as yet another extension in the end-point that can be easily added without modifying the protocol.

The main advantages of SIP are (1) its simplicity and (2) how quickly one could build many different types of applications. I also got the opportunity to work on several SIP based systems.

*"SIP has become the SS8 of PSTN trunking"*

*-- Jiri Kuthan*



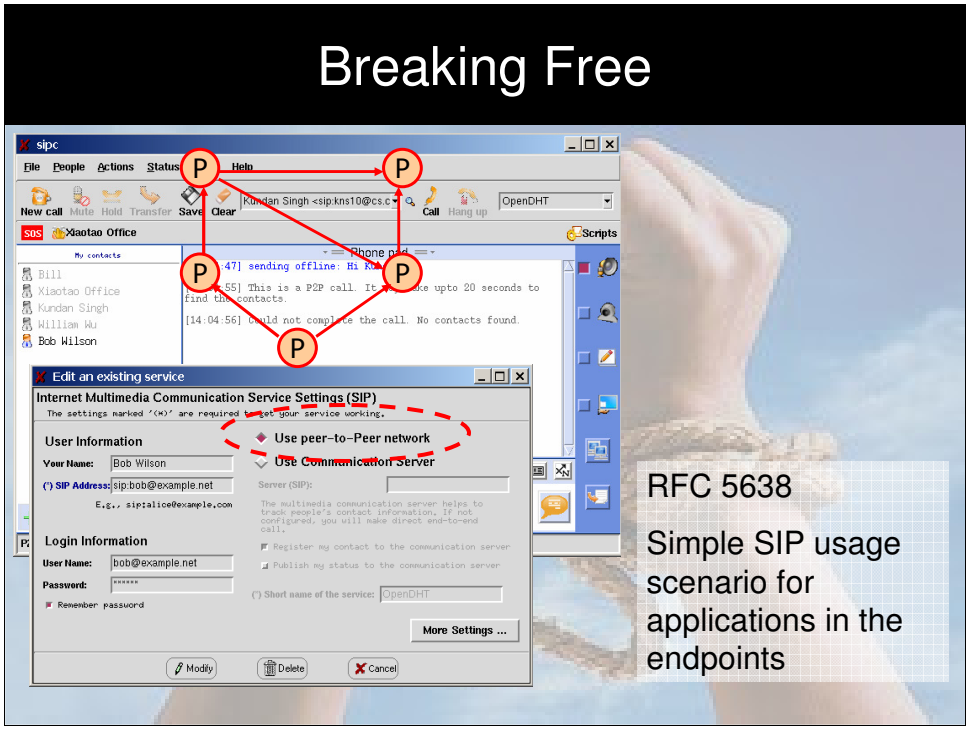

SIP was designed to solve two problems – discover other users or devices and negotiate parameters to connect them in a session. The focus should have been real Internet problems such as security, end-to-end services, traversal across middle boxes.

The fact that SIP needed to prove itself as working in all the previous telecom use cases rather than the emerging Internet applications, and the fact that it was so simple to implement and extend, it started an infestation of SIP related extensions, Internet drafts and RFCs. Many of these extensions were just to please a closed walled garden network of a carrier, or to carry a unique parameter between two telecom equipments that decided to use SIP and IP in the middle. Looking back over the last 5-10 years, these bullets summarize the reason for the undue complexity that resulted due to the specification explosion in my opinion.

The core protocol didn't have a clear separation between the session initiation aspect and the application logic. This meant that any changes in the application logic resulted in yet another extension even if the core remained the same. One of my friend recently commented that SIP has become the SS8 of PSTN trunking – not necessarily something we are proud of but that is the reality. With more telecom involvement and the need to build closed walled garden, the complexity became too overwhelming for many. The original philosophy and original principles behind the design were thrown out of the window. It became so complex that the original problem it was meant to solve was no longer solved easily, e.g., security and traversal. With this SBC (session border controller) became the new buzzword which could potentially solve every problem without caring about end-to-end philosophy behind the origin of SIP.

In summary, SIP got misused to solve problems that it was not supposed to solve, or the problems that were not really the core problems on the Internet but more of the business problems of the telecom world. Back to back user agents and SBCs are just the opposite of what SIP stands for in principle.

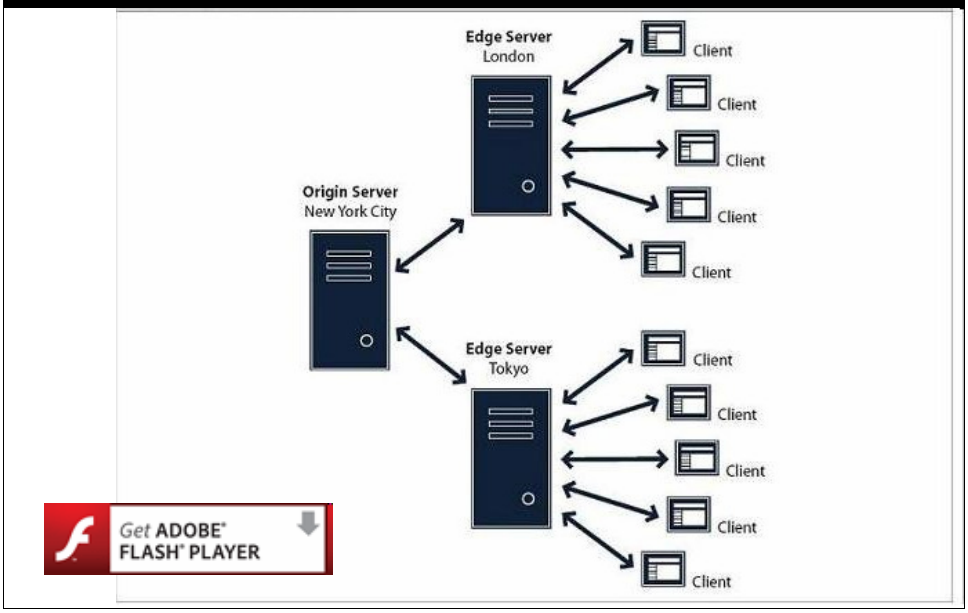
Today many vendors and carriers use SIP but in a way that prevents you from directly talking to them over SIP. For example, comcast has SIP based voice platform, Apple's Facetime uses tweaked SIP, but they don't allow a direct SIP connectivity from your SIP phone to the service. It has yet to come to the Internet people…




By 2004, I had done a lot of work on SIP based systems, I started seeing the prevalence of “managed” SIP services and closed walled garden. And I wanted to break free of such closed systems. In line with the original motivation of end-to-end services, we started a peer-to-peer initiative using SIP. I did a few prototype implementations to demonstrate the technology, wrote a few papers to show the point, and eventually a P2PSIP working group was formed to further the standardization of the effort. The idea was to get rid of the servers and be able to discover each other on a peer-to-peer network. If there are no servers then there will be no “managed” service and there will be no walled garden of services.

With the growing number of SIP specifications, and more than 100 RFCs in the family, the original end-to-end goal started getting lost in the wild. I co-authored an RFC to itemize the core set of a dozen or so RFCs that are needed for SIP services in the end-points that do not worry about telecom extensions, features or interoperability. This was an attempt to tell the Internet folks that it is not as complex as it seems for your applications.

## Interactive Video Streaming




With the emergence of IP multicast, the idea of Internet radio and TV emerged. Even though Internet multicast never became available, the idea persisted. Eventually, video viewing and streaming via web became so popular that it gave rise to an entirely new industry. In the process, the Adobe's Flash Player plugin became the most popular mechanism to deliver video to the end user.

Since Flash Player was already capable of doing video streaming, it was just a minor step to add interactive communication where Flash Player could also publish the video to others.

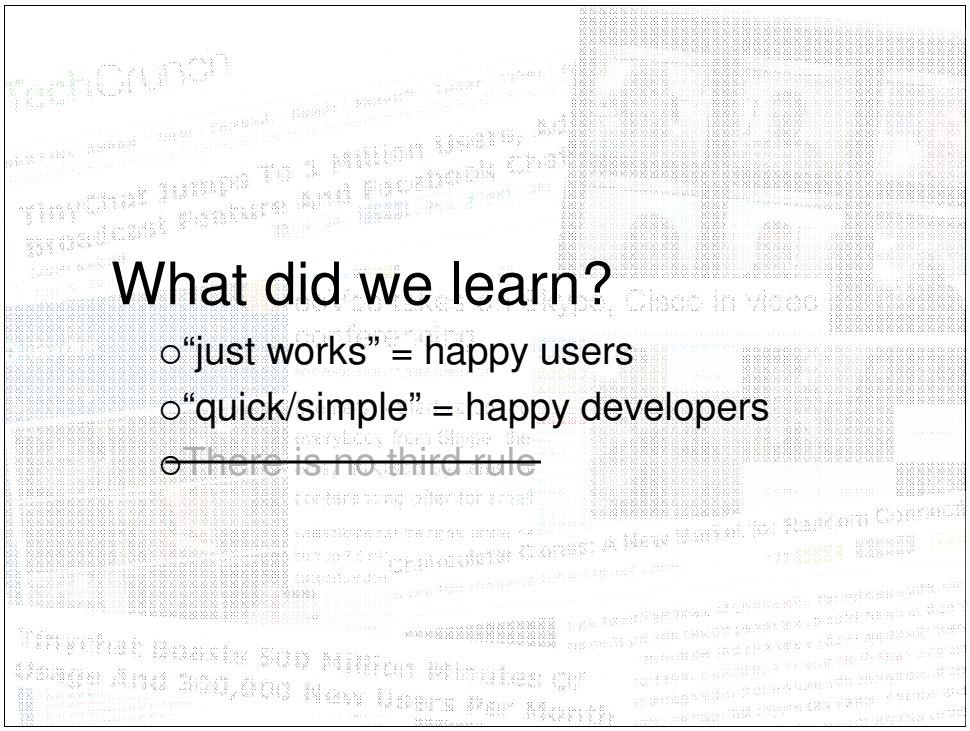

Let us review what makes Flash Player a approach promising, and makes it different from other approaches.


The most important reason is that the Flash Player plugin is available to almost everyone with a PC and Internet. It is ubiquitous, more than a specific brand of browser. Anyone with an internet connection and a browser can generally use Flash-based applications. Recently, web has evolved as a popular platform for everything we do on Internet including email, text chat, voice calls, discussions, enterprise applications and collaboration. And Flash Player fixes your web browser to do things that it is not already capable of, e.g., device capture and media transport.

The second reason is that developers find it very easy to work on Flash platform. Its cross browser platform support is amazing. Unlike a few hundred VoIP companies, there are millions of web developers. The programming language (ActionScript/MXML) used by Flash Player are similar to the web application languages such as JavaScript and HTML. The learning curve is quick, the development tools are awesome, and the community support is excellent!

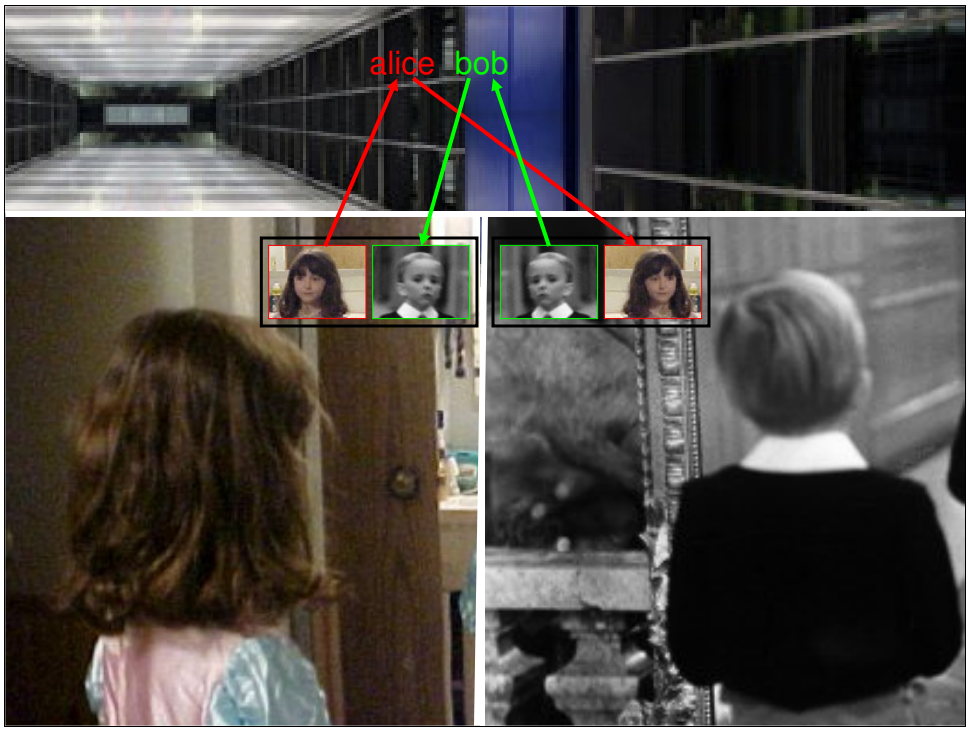
Thirdly, the end user sees the whole rich internet application experience as embedded and immersive in to what he is doing. For example, when browsing Facebook, you can chat with other friends within the Facebook web page.

Finally, there is no additional download needed to install and configure, but it is just there. This is huge plus for some use cases, e.g., in cyber-café or secure machines, where you do not have the luxury to install and configure Skype, Adium, Yahoo, MSN, etc, you can still do Flash-based video conference.

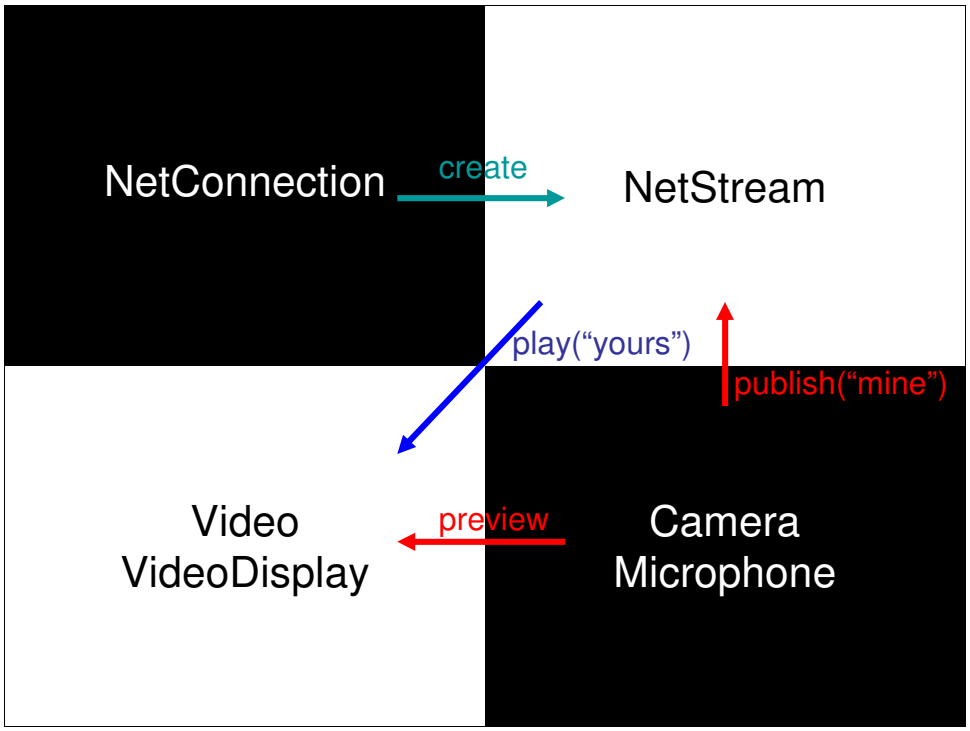



These are the reasons for the popularity of Flash based video communication web sites in recent years. It is too easy and quick for you to get started -- both from developers and users point of view. In technology, it is not always the big that wins, but the fast one does. And Flash Player gives the necessary tools to quickly add web video communication to your software product, service or platform.

What we learn from this are two important lessons: if the system “just works” without much installation, configuration, how-tos, then users are happy, which makes them use your system again and again, and helps your business. If the system is quick and simple to build, then developers are happy and motivated to add more functions, learn quickly, and innovate! This moves your business quickly from idea to production, instead of having to make big investments. There is no third rule to success in your video communication business!




Let us look at how a simple two party video call works with Flash Player. Suppose Alice and Bob want to do a video call using the Flash media service. The media service provides abstract named streams which Alice can publish and Bob can play, and vice-versa. From the software point of view each side has two Video components, one configured to publish and other to play, to build such as video call application.




From the implementation point of view, Flash Player provides four components of classes or abstractions related to video communication application.

The network connection, NetConnection, represents the association between the client Flash application and the media service, which is necessary to create a media stream.

The camera and microphone abstractions provide a platform independent device input. The Flash application does not deal with raw or encoded media stream, but just connects the camera and microphone to either video rendering object or the media stream.

Typically your application will attach the camera to the local video display, and also to the published stream name. And play the stream name of the remote side to the video display.

NetConnection

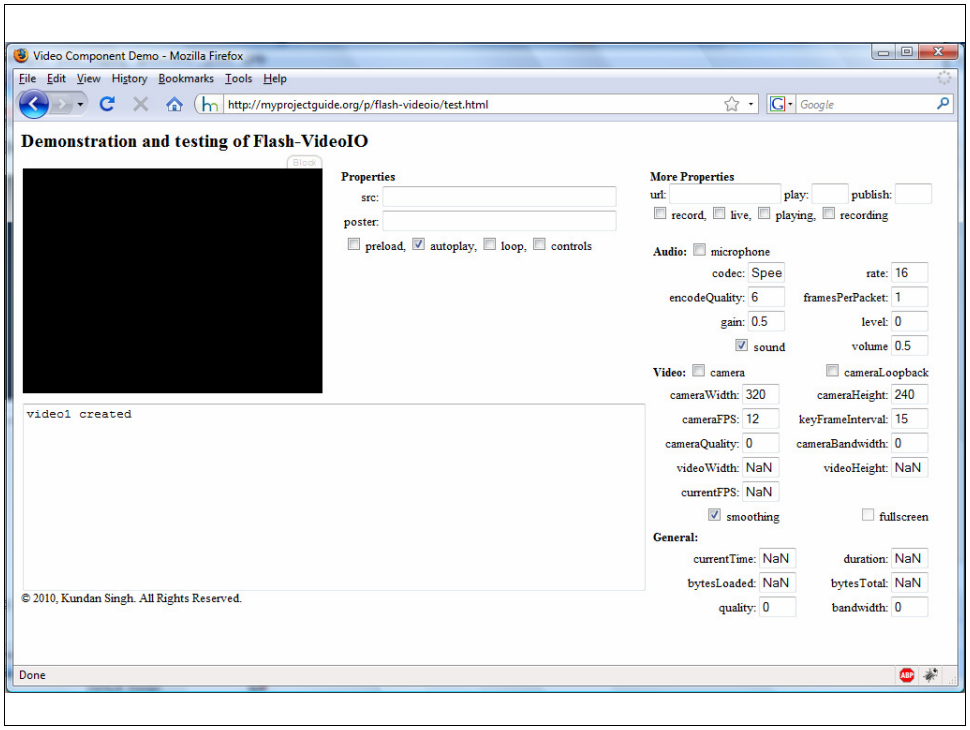
NetStream

Video
VideoDisplay

Camera
Microphone


My VideoIO project combines all these abstractions into a single easy to use Flash application with extensive JavaScript API. This example shows how the "src" property can set the VideoIO component to a video publish mode. As mentioned before with two such video components and some form of negotiation for media stream names, you can build a video call application.

In the last few years I have had the opportunity to explore this area in detail and to contribute in both industry and various open source projects. The Flash VideoIO is one of my interesting projects that brings the power of Flash Player plugin to regular web developers for video communication and messaging related application. There are several examples of how to use this in various use cases. You can visit the project web site at http://code.google.com/p/flash-videoio/ for more information.

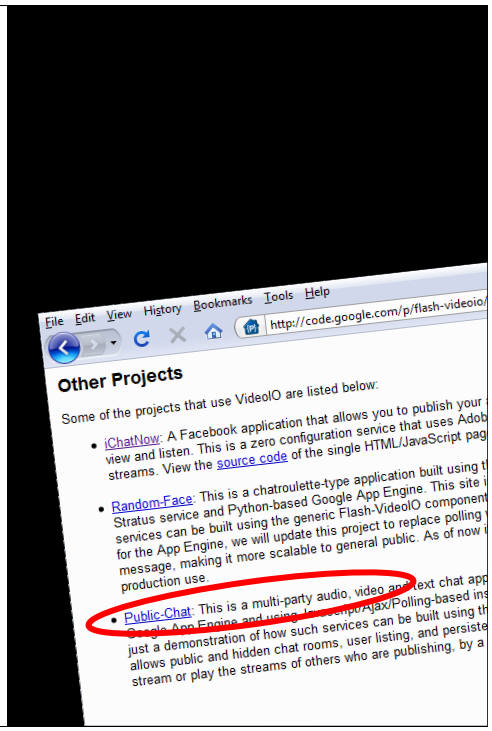



It defines a simple but extensive web API to enable various use cases related to web video communication and messaging. You can explore the various API properties and calls from its test page.

Multi-party audio/video conference

http://public-chat.appspot.com/

< 1000 lines of code




I have built several sample applications using VideoIO such as chat roulette type random chat system, multi-party video conferencing. This is built on Google App Engine, using standard web application techniques such as AJAX, and asynchronous channel API-based IM and presence.

# siprtmp.py in action


It is possible to interoperate between Flash Player and standard SIP terminals. For example, siprtmp is one such project that I have been involved with. The gateway allows a Flash application embedded in a web page to register to SIP server and make or receive SIP calls. The translation mechanism and source code of the gateway is at http://code.google.com/p/rtmplite/source/browse/trunk/siprtmp.py and I have co-authored a technical report describing the translation as well.

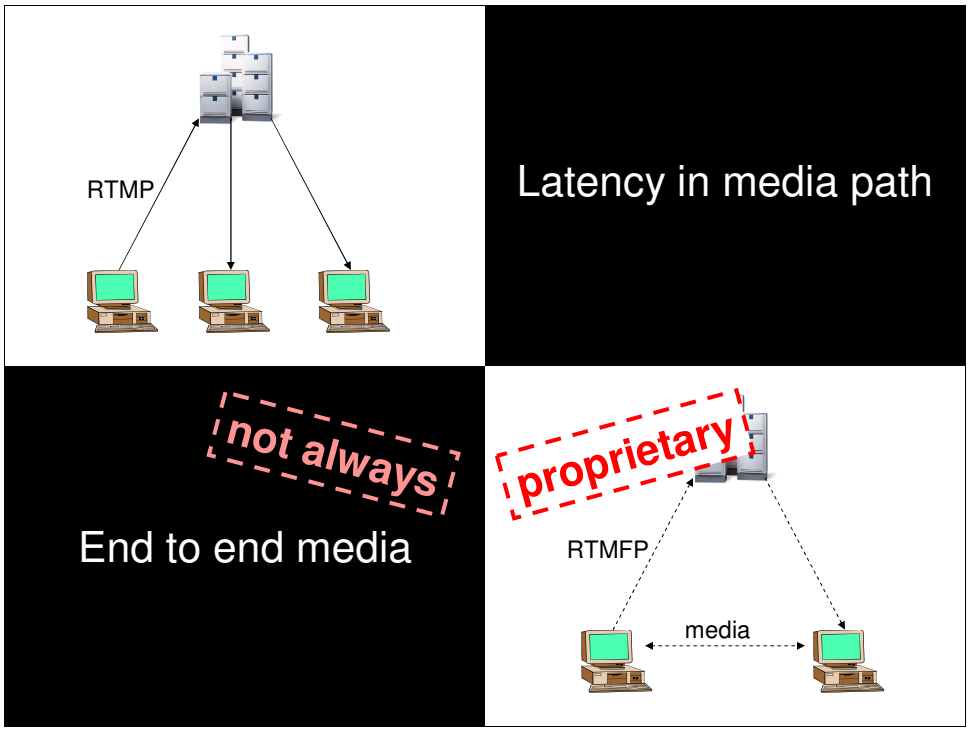
While the gateway can allow you to translate signaling and voice, the video stream is more involved because the proprietary/patented Sorenson codec used by Flash Player. Recent version of Flash Player allows H.264 capture from the browser and enables interoperability with other H.264 supporting SIP devices.

You can use the gateway to implement several web telephony use cases such as PC-to-phone dialing, allowing a phone user to dial into a web conference, or integrating your web application with existing VoIP infrastructure.

Moreover, I am working on a project to integrate siprtmp with P2P-SIP adaptor to enable downloadable external application that converts your web applications to SIP phones.

“All that glitters is not gold”

Now that we have seen the power and ease of Flash based audio video communication, let us look at some limitations.




The origin of Flash Player's interactivity comes from video streaming use case which is typically client server in nature. This diagram shows the high level client-server architecture for Flash video communication. RTMP (real-time messaging protocol) runs over TCP and allows you to create named streams at the media server, and publish or play media over it. This works well for one-to-many streaming application where provider can install multiple servers and do load balancing. It also works well for NAT and firewall traversal because of the client-server nature of the TCP connection, and both the media and signaling paths are same.

But the latency in media path can become huge, and is not desirable for interactive video calls. Luckily Adobe has added another protocol, RTMFP (Real-time media flow protocol), which enables end-to-end media over UDP and significantly reduces the latency in the media path.

It works, but has two problems. First it is proprietary, and second, it cannot always do end-to-end media because of certain firewalls and NAT restrictions. Because it is proprietary, third-party cannot build scalable and distributed media relays, similar to super-node architecture of Skype. Thus, for a robust service, the provider needs to invest in service infrastructure or fall-back to client-server RTMP. There has been some effort in reverse engineering RTMFP, e.g., the openRTMFP's cumulus project.


Until last year Flash Player did not have good echo cancellation and until recently it did not have high quality video encoder. This was the reason for many problems and customer dissatisfaction when I built my first global Flash based video communication services.

Unfortunately what this means is that developers are always tied to what the plugin vendor provides and dependent on the vendor to provide new features and security fixes. For example, the recent addition of H.264 in Flash Player 11.0 has a bug that prevents playing live H.264 streams where a single frame is split into multiple slices and sent as one NALU per slice. This prevents many existing SIP systems to interoperate with Flash via my siprtmp gateway.

Other missing features are full support for general purpose UDP socket and access to encoded media data to the application so that the application can use alternative transport mechanism.


Flash Player is just one of the several ways to do web-based video communication. The core problem is something else – web has had tremendous growth recently and existing VoIP protocols fail to deliver the promise for web users. Web and VoIP have evolved as 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.

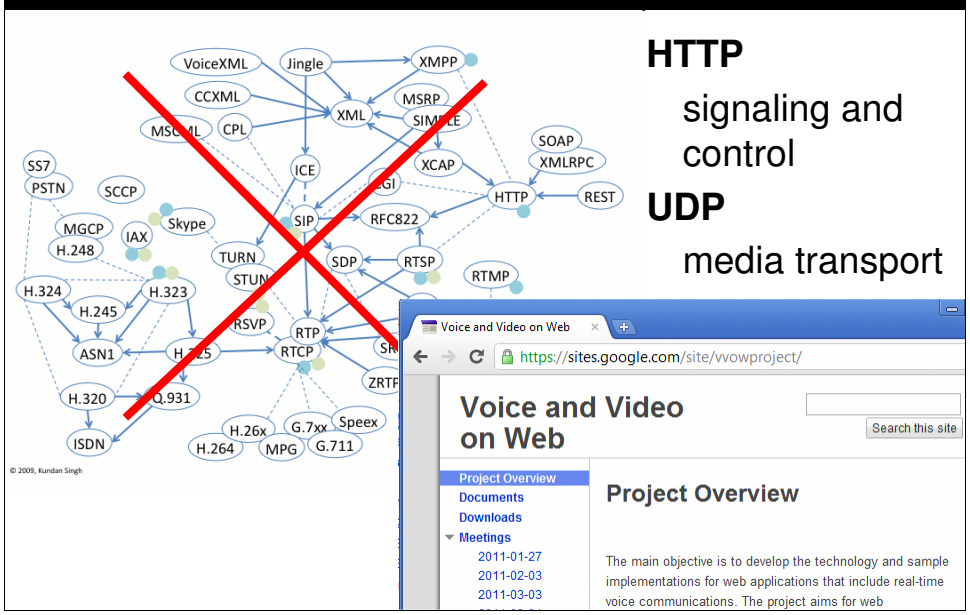
Web companies are fast paced, with quick turnaround, and build easy to use applications whereas VoIP is traditionally linked to complex protocol machinery which is difficult to get right across different equipments. Critical programming primitives are also missing in a web browser: UDP transport, listening socket, native device access. While Flash Player solves some of these problems, its closed implementation does not give a generic application level UDP socket or access to encoded media packets to the application, restricting the hands of the developers.

# From VoIP to Web

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


Moving from traditional VoIP to Web communications requires a lot of changes. Firstly the provider's 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 the existing web browsing experience. For example, embedded click to call, auto-conference among current visitors on a page, etc. Trust model is different on the 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.

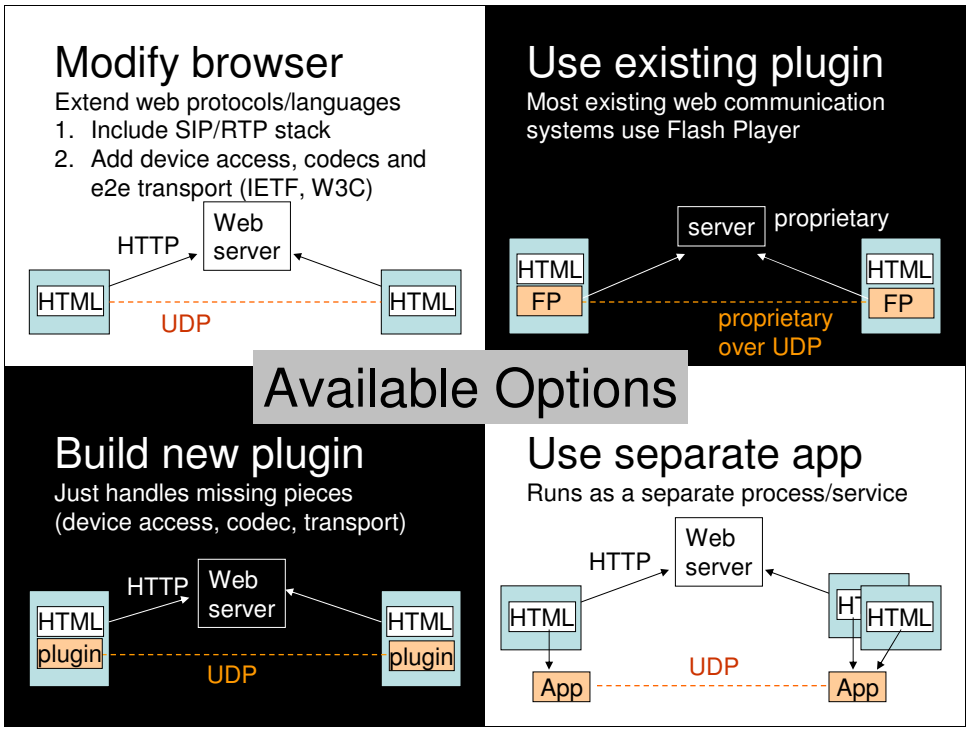
# Minimum Protocol Requirements




Although there are many different protocols for multimedia communications, you just need two protocols on the web: HTTP for signaling/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 my project in collaboration with other researchers on “voice and video communications on web”.

I have an initial prototype of a web video conferencing and slide sharing system along with a published paper in a recent conference.

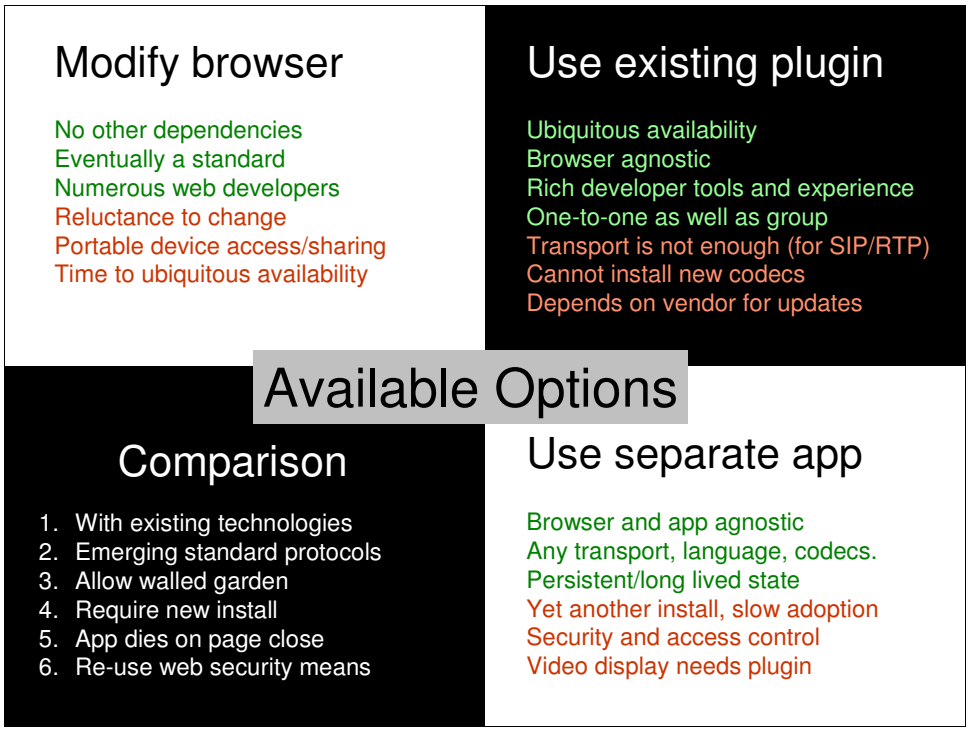
Understanding the minimum requirements is useful in designing the solution. It helps in keeping unwanted “fat” outside the scope.




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.

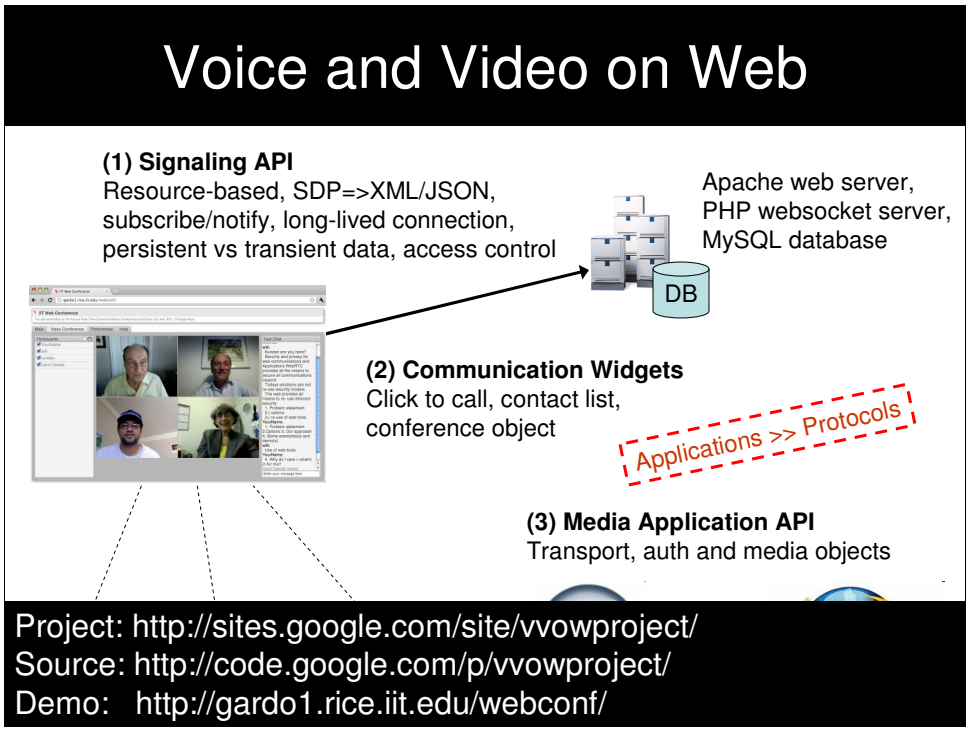



This slide compares the various options. The green lines are advantages and red ones disadvantages. Modifying the browser to adopt the emerging standards is the ideal solution in the long run. It doesn't have additional dependencies on plugins or other applications. However, typically changing the browser 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).



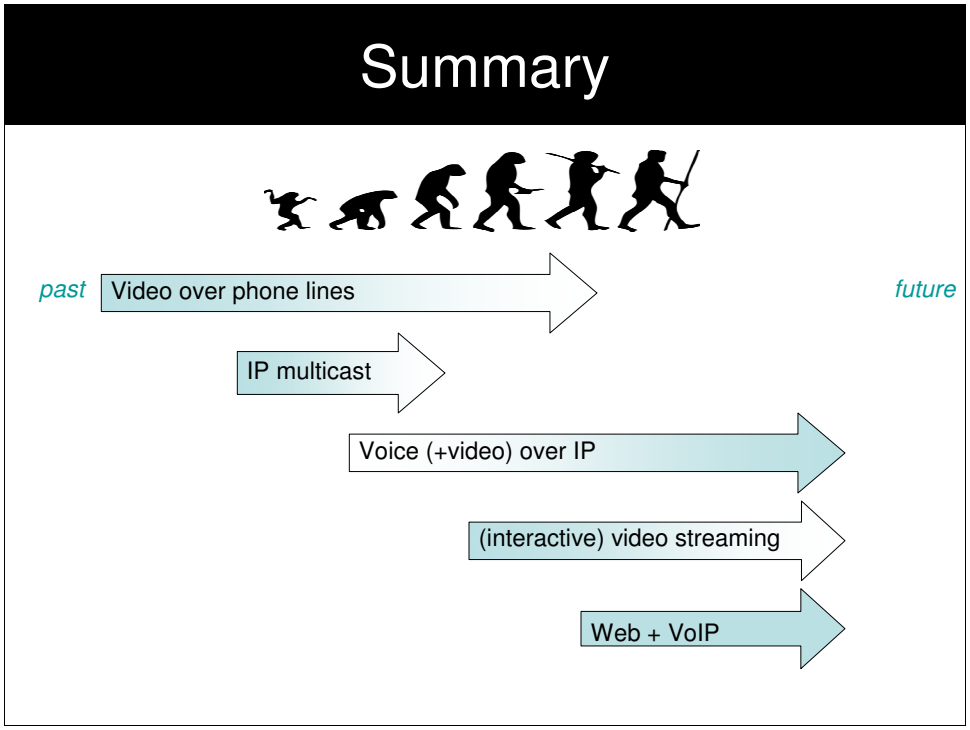

Our project has three parts: (1) signaling, (2) communication widgets, and (3) media application API. This applies to other web communication applications as well. The signaling API defines how people discover each other and how session is negotiated. Essentially this is a web version of SIP/SDP. As 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. We have implemented a generic resource-oriented client-server API with these features so that the application can define its own resources for web communication.

The second part contains 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.




In summary, we have talked about the various origins that have contributed to and shaped the Internet video communication technology we see today. We talked about the challenges various approaches tried to solve, and the drawbacks they suffered from.

We have seen some important lessons from the past about what works and what does not. And finally we have seen a number of intricacies of web based video communication platform with or without Flash Player.

## Moving Forward

- Stay technical; R&D
- Web RTC, but more on tech demos
- Global cloud-based SIP services
- Mobile and custom video devices

http://myprojectguide.org

One of the things that intrigues me about Internet video communication is that we are never satisfied with what we have. There is always a need to improve, and this I feel is a good thing for research. In near term future I would like to continue exploring the space from the technical angle.

I will continue to follow the web RTC work but more importantly I like to contribute to more technology demonstrations and system implementations that solve the problems rather than contribute in writing pages of specifications that create more problems.

One interesting project I wish to do is to create a global cloud-based SIP service that could be used by anyone on the Internet or web without being controlled by a single entity or “managed” service provider.

Finally, I feel that mobile and custom video devices will form the core of our future. While web provides convenience for many, a custom and affordable video conferencing device is what is needed. So I wish to work on some of these aspects of the area too.

Some of these projects as well as many other interesting projects are listed on my web site that I built for student projects.