Paper title: Vclick - endpoint driven enterprise WebRTC.
Speaker: Kundan Singh. Track: Systems and Architectures.

"Smart end points and
a dumb network"

Hello everyone. My name is Kundan Singh.

This paper presentation is really about how to apply this fundamental principle of Internet to multimedia collaboration. Internet's growth is largely attributed to the end-to-end principle. However, today's multimedia conferencing systems do the opposite, with bulky and expensive servers and dumb terminals.

Vclick is a web-based multimedia collaboration system that includes audio/video conferencing, text chat, screen sharing, shared whiteboard, notepad and others.

It runs all the application logic in the endpoint browser, and is independent of any legacy VoIP systems such as SIP or XMPP.

This paper shows why endpoint driven system is useful, what is Vclick and how various features in Vclick are implemented?

John is my co-author.

Let me start with a quick video demonstration of Vclick.

This is Vclick's configuration. User registered as this email address.

One of the things Vclick does is to inject click to icon on third-party websites such as our web based corporate directory. The blue icon in the browser indicates that Vclick has registered and can receive call. Right click on this icon allows selecting how you want to communicate. Or just click to initiate a default video call.

When the other machine answers the call, it becomes a two party video call. I can move or resize the picture in picture box.

The text chat allows text conversation. It also allows drag and drop file sharing.

Additionally, it has screen sharing, shared white-board, and shared notepad. White-board uses SVG for drawings. Each of these widgets - text chat, white board, video boxes, etc., -- has its own URL, runs in an iframe, and can be launched in a separate tab if needed.

For example, I can right click on a remote participant video, and open that video in a new tab. This opens that video app or widget in that tab, and supplied the right parameters to receive the video stream. I could copy this URL and send in an IM to another person, who can then see and hear only this participant of the conference.

There are other client side features here as well. For example, ability to zoom part of a video. Or to see the signaling messages for testing or debugging.

I am going to use a third device, a mobile device, to call this user. An incoming call notification can be clicked to answer or closed to decline. When answered, I am now in two different 2-party calls.

I can drag a participant from one call to another, to merge the two calls, to make it a 3-party call here.

Double click a video to make it larger. Change the window size to see how the videos change positions and sizes. Double click the larger video to make it same size again. The flexible layout is pretty useful for wide range of use cases, including portrait and landscape orientations of mobile devices.

Multi-party call also allows private chat and private audio if needed.

This page allows creating a meeting URL, which can be clicked to join a pre-configured meeting. This is possible because the call initiation app is independent of the video conversation app.

The contact list is just a web page that accesses the presence data of users. I can add a new contact by email. The click to call icon is same, with right click option.

Let me show a text only conversation to this target. This app also has the client side logic to enable text to speech and speech recognition. This allows hands-free mode of text chat.

These are only some of the features of Vclick. Note that all the features and apps in Vclick are implemented using endpoint driven app logic.

So why is endpoint driven system needed? Existing web video conferencing systems tend to have bulky servers and thin clients. The server implements all the features such as video layout, mixing, text chat and such.

The client user interface is often rigid with fixed boxes. For example, you cannot easily move a participant video to a new tab on a different monitor. Or you cannot easily share multiple screens in a conference.

Applications are tightly coupled, e.g., you must use the same client to invite a new participant, or must chat from within that video conference UI, even though call invitation and chat may not be related to the application logic of active video conference.

Many times, these web clients are implemented as single-page applications, that have their own problems. For example, memory leak build up over a long running process.

On the other hand, Vclick is an endpoint driven system, with a very thin server. In fact the server only provides shared data and event notifications.

All the application logics are in the individual applications, that are independent of each other.

These applications mash up at the data level, by manipulating shared data, instead of asking permission from another app.

An individual app runs in an iframe or a tab, and a complete app consists of several iframes or tabs, unlike a single page app.

Fig.5 shows another way to look at it. The data server is called a resource server in our architecture. The call initiation app uses the presence resource to deliver the "invite" event to the target. Once target joins, the individual conversation apps such as for video conference or text chat do not need to deal with presence, but only with call membership resource.

Another developer can create a new app, e.g., to use the presence resource to create a user's contact list, without affecting call initiation or text chat apps.

Vclick differs from other existing web conferencing systems in three ways: (1) it has loosely coupled independent apps, instead of a single rigid app, (2) it separates the user data from the app logic, so that apps can mash up at the data level, and (3) it runs all the app logic in the endpoint, so that the server is light weight, robust, and scalable.

This is the gist of what Vclick is all about.

Vclick uses WebRTC or Web Real Time Communication for audio video media flows.

If you are not familiar with WebRTC, Fig.2. shows how the two JavaScript apps in the two browsers establish a peer connection, and stream audio and video from one to another. The web server is used as a signaling channel to exchange the negotiation parameters between the two apps to establish the media path.

In our, the light weight resource server also serves as the signaling server between the browsers. The server stores shared heirarchical data, and allows access and events on these data resources. For example, one client can subscribe to a presence resource /users/bob/presence, and can receive notification when it is updated by another client, to update its presence user interface.

This basic concept is used in various app logic, e.g., for call initiation, presence, call membership, shared whiteboard, and so on. Fig.6 shows two browsers of Bob subscribed to the presence data, and both receive call "invite" event. The caller app cancels the other branch when the first one accepts the invite.

With regard to the system design, the paper also presents some guidelines. For example, keeping call initiation separate from conversation apps allows adding new way to initiate call or add new conversation app without affecting the other.

Paper also shows the details of the video layout algorithm in Fig.8. In summary, the algorithm tries to minimize the empty space as shown here.

On the left all videos are of same size, and on the right the speaker video is larger than others.

There are many other details in the paper. I will highlight the problems here, and leave it to the interested readers to read the paper for more details.

What's more in the paper?

- How to transfer or merge using video drag-n-drop?
- Why do we use uni-directional peer-connections?
- What are the limitations of full mesh media path?
- And how do we address that?
- What are the challenges of including a telephony endpoint?
- What are differences in cloud vs on-premise hosting?
- What are the problems in running Vclick on mobile?
- And how do we address those?
- Why does WebSocket fail to work sometimes?
- How do we do endpoint driven reliability?

Again, this is the gist of what Vclick's endpoint driven system architecture is about. With Vclick we show that many complex multimedia collaboration scenarios can be implemented in an endpoint driven architecture.

That concludes my presentation. We have been running Vclick internally as well as on the cloud for several years now. It has became a test bed for some of us to try out new features in WebRTC or to quickly integrate WebRTC in existing or new projects.