Sending Data with WebRTC

Sending Data with WebRTC

- Up to this point, we have focused solely on the audio and video capabilities of WebRTC.
- However, there is an entire subject that we have not even started to talk about— arbitrary data.
- It turns out that WebRTC is good at transferring data; not just audio and video streams, but any data we might have.

Sending Data with WebRTC

Over the course of this session, we will cover the following topics:

- How the data channel fits into the WebRTC puzzle
- How to create a data channel object from a peer connection
- What are the encryption and security concerns
- What are the potential use cases for the data channel

Stream Control Transmission Protocol and data transportation

- As it turns out, sending data over a peer connection is somewhat of a hard task.
- Typically, sending data between users today is done through the use of a strict TCP connection.
- This means using technologies such as AJAX and WebSockets to send data to a server and then receive it at the other end.
- This can often be a slow and cumbersome issue for high-performance applications.

Stream Control Transmission Protocol and data transportation

- In our WebRTC model, we have already covered the need for a higher speed, lower latency connection between users.

- With this connection, we have the ability to send audio and video data quickly between our peers.

- The protocol used today for audio and video data, however, is specifically designed for frames of a video and audio stream.

Stream Control Transmission Protocol and data transportation

- The SCTP is yet another technology that sits on the independent WebRTC stack.

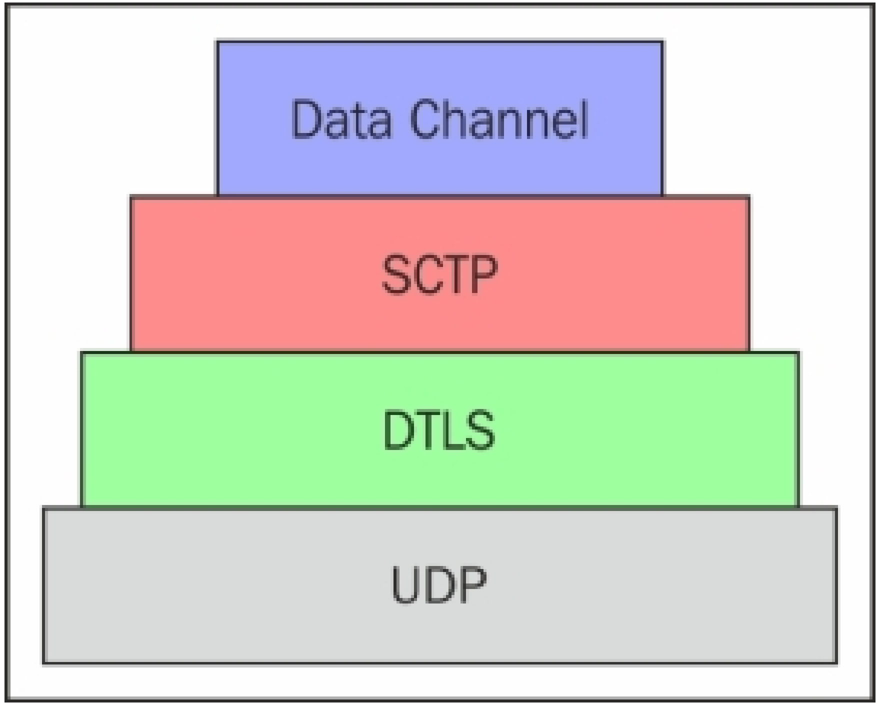
- This is another reason why WebRTC is such an interesting technology.

- It has brought many new methods of data transportation to every JavaScript developers' fingertips.

- SCTP sits on top of the Datagram Transport Layer Security (DTLS) protocol that is implemented for every WebRTC connection and provides an outlet for the data channel to bind to for data transportation.

Stream Control Transmission Protocol and data transportation

- So far, our stack currently looks something like this:



Stream Control Transmission Protocol and data transportation

- This may look like a lot of complexity but it is all for the benefit of the power of SCTP.
- The designers of WebRTC realized that every application would be unique in how it wants to harness the power of the data channel.
- Some might want the reliable delivery of TCP, while others might need the high performance of UDP.
- It was this issue that caused them to choose SCTP, which gives the best of both the worlds.

Stream Control Transmission Protocol and data transportation

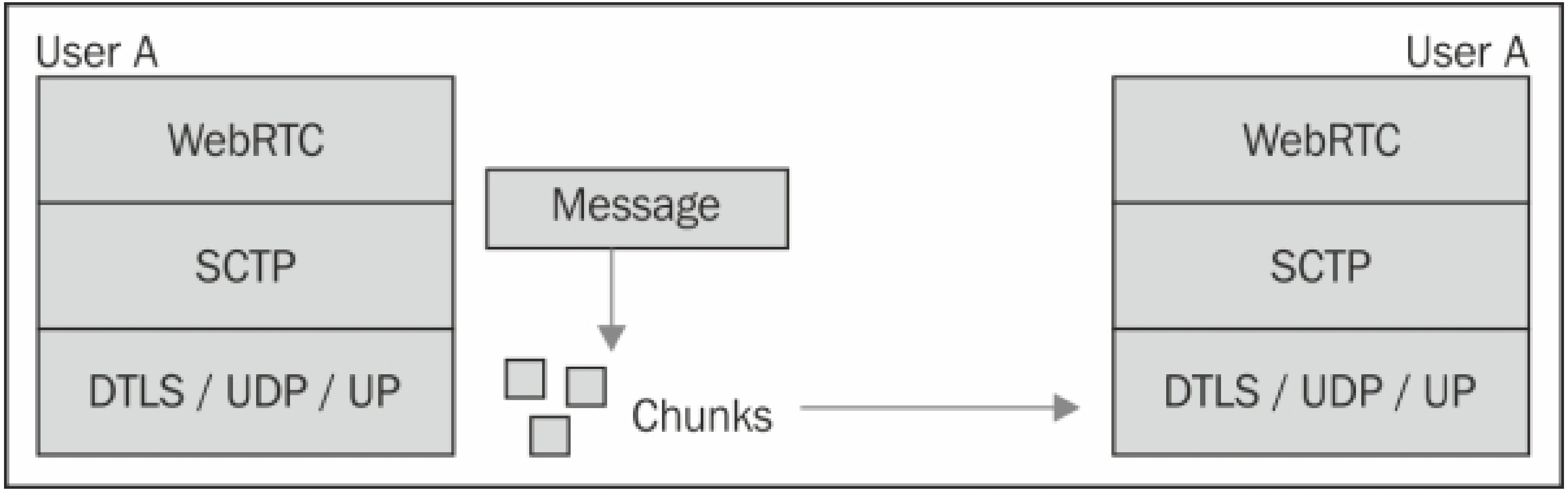
- The SCTP specification is defined by using multiple endpoints, which send messages broken down through chunks. Here is a list of what these are:

- Endpoints: These are defined as any number of connections between two IP locations

- Messages: These are any portions of data sent to the SCTP layer from the application

Stream Control Transmission Protocol and data transportation

- Chunks: These are packets of data ready to be sent across the wire and can represent a part of any message.



The RTCDataChannel object

- Now that we understand the underlying technologies at play, we can start learning how the actual RTCDataChannel object API works.
- The great thing is that this API is much less complex than the underlying workings of the SCTP.
- The main function to create a channel comes from an already established

- RTCPeerConnection object:

REFER TO THE FILE 6_1.txt

The RTCDataChannel object

- Now that we understand the underlying technologies at play, we can There are a number of states that a data channel can be in:

- connecting: This is the default state, where a data channel waits for a connection

- open: This is the state when a connection is established and communication is possible

- closing: This is the state when a channel is currently being deconstructed

The RTCDataChannel object

- The way that the browser notifies the application about the data channel states is through a series of events.
- When the other peer creates a channel, the ondatachannel event is fired on the RTCPeerConnection object, notifying that a channel has been created.
- Then the RTCDataChannel object itself has a few events to notify when it is opened, closed, has an error, or gets a message:

REFER TO THE FILE 6_2.txt

The RTCDataChannel object

Data channel options

- You may have noticed a dataChannelOptions object passed in our example.
- Since SCTP gives many different configurable ways to send data to the other peer, these options have been made available to the end developer.
- The options passed is optional and should be a regular JavaScript object:

REFER TO THE FILE 6_3.txt

The RTCDataChannel object

Sending data

- The send method of the data channel is overloaded, much like the one on WebSockets.
- This allows many different JavaScript types to be sent over the transport layer.
- Using a different data type can help speed up the performance of an application.
- This is due to the heaviness of most string-based encodings, which require more chunks to send data

The RTCDataChannel object

Currently, data channel supports the following types:

- String: A basic JavaScript string
- Blob: A data type introduced to represent a file-like format of raw data
- ArrayBuffer: A typed-array of fixed length that cannot be modified
- ArrayBufferView: A view frame of an underlying ArrayBuffer

The RTCDataChannel object

- Simply drop the variable in the send function and the browser will figure out the rest.

- You can then identify the type on the other side by testing its type:

REFER TO THE FILE 6_4.txt

Encryption and security

- Having messages transmitted in a secure way is one of high importance for the designers of the WebRTC protocol.

- The reasoning behind this is that many large companies would not consider using a WebRTC application without the right level of security implementation.

- To get the widest adoption rate possible meant that security had to be implemented right into the design of the API itself.

Encryption and security

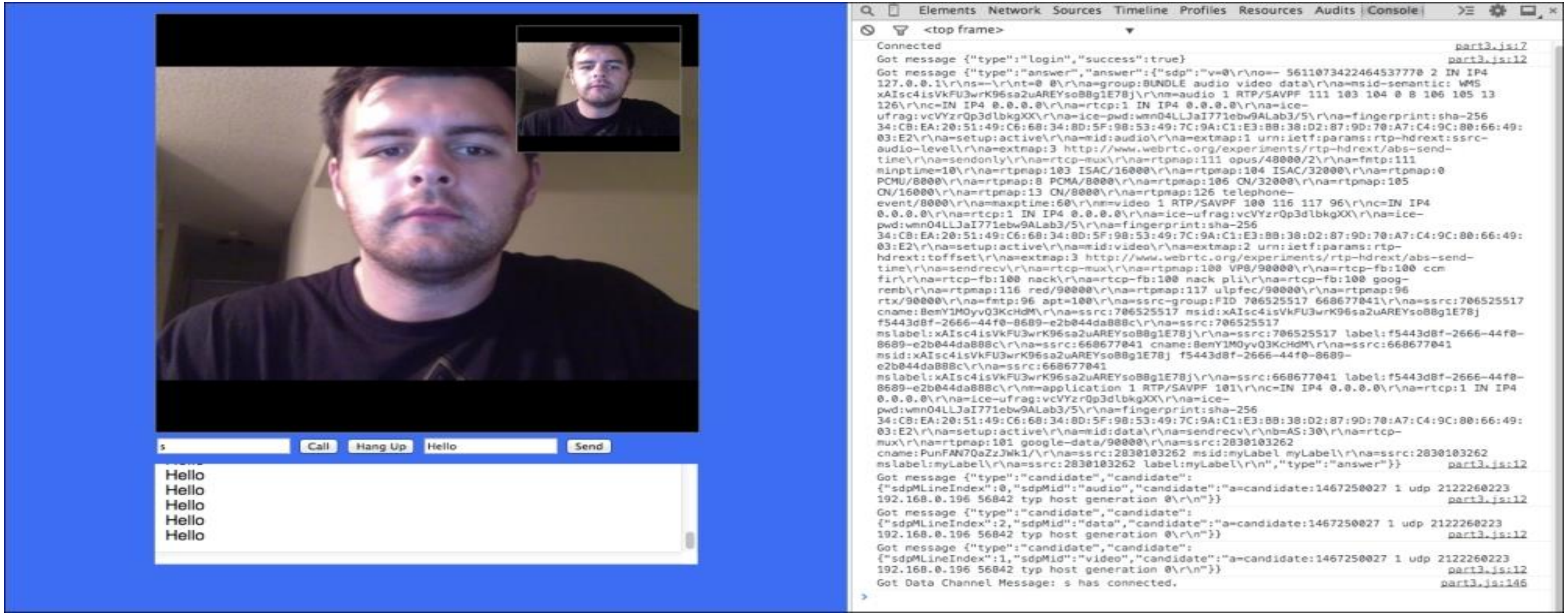
- The encryption technology used had to satisfy several requirements for being used in peer applications:
- Messages should not be readable if they are stolen while in transit between peers
- A third party should not be able to forge messages to look as if they were sent by the connected peer
- Messages should not be editing-enabled while in transit to the other peer

Adding text-based chat

- Now we will use what we have learned in this session to add data channel support to our communication application.
- Since the data channel can be used for any arbitrary data, we are going to add text-based chat as another feature of our application.
- The users will have a text box to enter a message into and a display of all the messages in the current call.

Adding text-based chat

When we are finished, it will end up looking something similar to this:



```
Hello
Hello
Hello
Hello
Hello
```

```
Connected                                                                        part3.js:7
Got message {"type":"login","success":true}                                      part3.js:12
Got message {"type":"answer","answer":{"sdp":"v=0\r\no=- 5611073422464537770 2 IN IP4
127.0.0.1\r\ns=-\r\nt=0 0\r\na=group:BUNDLE audio video data\r\na=msid-semantic: WMS
xAIsc4isVkFU3wrK96sa2uAREYsoB8g1E78j\r\nm=audio 1 RTP/SAVPF 111 103 104 0 8 106 105 13
126\r\nc=IN IP4 0.0.0.0\r\na=rtcp:1 IN IP4 0.0.0.0\r\na=ice-
ufrag:vcVYzrQp3dlbkgXX\r\na=ice-pwd:wmnO4LLJaI771ebw9ALab3/5\r\na=fingerprint:sha-256
34:CB:EA:20:51:49:C6:68:34:8D:5F:98:53:49:7C:9A:C1:E3:B8:38:D2:87:9D:70:A7:C4:9C:80:66:49:
03:E2\r\na=setup:active\r\na=mid:audio\r\na=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-
audio-level\r\na=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-
time\r\na=sendonly\r\na=rtcp-mux\r\na=rtpmap:111 opus/48000/2\r\na=fmtp:111
minptime=10\r\na=rtpmap:103 ISAC/16000\r\na=rtpmap:104 ISAC/32000\r\na=rtpmap:0
PCMU/8000\r\na=rtpmap:8 PCMA/8000\r\na=rtpmap:106 CN/32000\r\na=rtpmap:105
CN/16000\r\na=rtpmap:13 CN/8000\r\na=rtpmap:126 telephone-
event/8000\r\na=maxptime:60\r\nm=video 1 RTP/SAVPF 100 116 117 96\r\nc=IN IP4
0.0.0.0\r\na=rtcp:1 IN IP4 0.0.0.0\r\na=ice-ufrag:vcVYzrQp3dlbkgXX\r\na=ice-
pwd:wmnO4LLJaI771ebw9ALab3/5\r\na=fingerprint:sha-256
34:CB:EA:20:51:49:C6:68:34:8D:5F:98:53:49:7C:9A:C1:E3:B8:38:D2:87:9D:70:A7:C4:9C:80:66:49:
03:E2\r\na=setup:active\r\na=mid:video\r\na=extmap:2 urn:ietf:params:rtp-
hdrext:toffset\r\na=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-
time\r\na=sendrecv\r\na=rtcp-mux\r\na=rtpmap:100 VP8/90000\r\na=rtcp-fb:100 ccm
fir\r\na=rtcp-fb:100 nack\r\na=rtcp-fb:100 nack pli\r\na=rtcp-fb:100 goog-
remb\r\na=rtpmap:116 red/90000\r\na=rtpmap:117 ulpfec/90000\r\na=rtpmap:96
rtx/90000\r\na=fmtp:96 apt=100\r\na=ssrc-group:FID 706525517 668677041\r\na=ssrc:706525517
cname:BemY1MOyvQ3KcHdM\r\na=ssrc:706525517 msid:xAIsc4isVkFU3wrK96sa2uAREYsoB8g1E78j
f5443d8f-2666-44f0-8689-e2b044da888c\r\na=ssrc:706525517
mslabel:xAIsc4isVkFU3wrK96sa2uAREYsoB8g1E78j\r\na=ssrc:706525517 label:f5443d8f-2666-44f0-
8689-e2b044da888c\r\na=ssrc:668677041 cname:BemY1MOyvQ3KcHdM\r\na=ssrc:668677041
msid:xAIsc4isVkFU3wrK96sa2uAREYsoB8g1E78j f5443d8f-2666-44f0-8689-
e2b044da888c\r\na=ssrc:668677041
mslabel:xAIsc4isVkFU3wrK96sa2uAREYsoB8g1E78j\r\na=ssrc:668677041 label:f5443d8f-2666-44f0-
8689-e2b044da888c\r\nm=application 1 RTP/SAVPF 101\r\nc=IN IP4 0.0.0.0\r\na=rtcp:1 IN IP4
0.0.0.0\r\na=ice-ufrag:vcVYzrQp3dlbkgXX\r\na=ice-
pwd:wmnO4LLJaI771ebw9ALab3/5\r\na=fingerprint:sha-256
34:CB:EA:20:51:49:C6:68:34:8D:5F:98:53:49:7C:9A:C1:E3:B8:38:D2:87:9D:70:A7:C4:9C:80:66:49:
03:E2\r\na=setup:active\r\na=mid:data\r\na=sendrecv\r\nb=AS:30\r\na=rtcp-
mux\r\na=rtpmap:101 google-data/90000\r\na=ssrc:2830103262
cname:PunFAN7QaZzJWk1/\r\na=ssrc:2830103262 msid:myLabel myLabel\r\na=ssrc:2830103262
mslabel:myLabel\r\na=ssrc:2830103262 label:myLabel\r\n","type":"answer"}}       part3.js:12
Got message {"type":"candidate","candidate":
{"sdpMLineIndex":0,"sdpMid":"audio","candidate":"a=candidate:1467250027 1 udp 2122260223
192.168.0.196 56842 typ host generation 0\r\n"}}                                 part3.js:12
Got message {"type":"candidate","candidate":
{"sdpMLineIndex":2,"sdpMid":"data","candidate":"a=candidate:1467250027 1 udp 2122260223
192.168.0.196 56842 typ host generation 0\r\n"}}                                 part3.js:12
Got message {"type":"candidate","candidate":
{"sdpMLineIndex":1,"sdpMid":"video","candidate":"a=candidate:1467250027 1 udp 2122260223
192.168.0.196 56842 typ host generation 0\r\n"}}                                 part3.js:12
Got Data Channel Message: s has connected.                                       part3.js:146
>
```

Adding text-based chat

- To get started, we will change the call page of our application.

- We will add three new elements—an input text field, a button, and a div.

- The input area and button will allow the user to enter text while div will hold all the messages between each user.

REFER TO THE FILE 6_5.txt

Adding text-based chat

- Next, we will add some JavaScript to our page to add the data channel functionality:

REFER TO THE FILE 6_6.txt

Adding text-based chat

This code will setup a series of listeners for our data channel:

- onerror: This listener will catch any connection issues detected
- onmessage: This listener will receive messages from the other user
- onopen: This listener will tell us when the other user has connectec
- onclose: This listener will tell us when the other user disconnects

Adding text-based chat

- Now, we can add in an event listener when the user clicks on the Send button:

REFER TO THE FILE 6_7.txt

Adding text-based chat

- The message area also has a bit of style added to it to give it the scrolling message window look and feel, found in most communication applications:

REFER TO THE FILE 6_8.txt

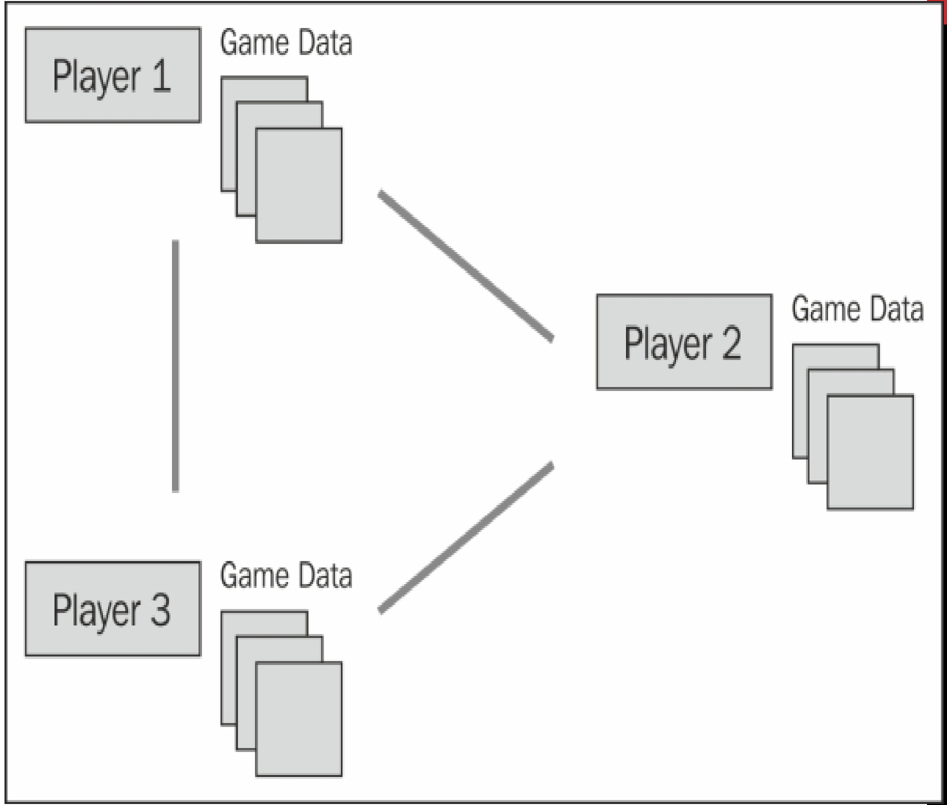
Use cases

- This is just the tip of the iceberg when using the WebRTC data channel. Adding text-based chat is the easiest extension of the protocol with what we have learned so far.
- Since the specification is just the arbitrary data between two users, the possibilities are almost limitless with what you can do.
- Gaming is one of the first logical extensions of the data channel protocol. In fact, peer-to-peer networking has been a common sight in many multiplayer games.

Use cases

- This is becoming an increasingly popular way to transfer large amounts of data among many users.

- The following diagram is an illustration of a common network layout for peer-to-peer games:



Summary

- Upon finishing this session, you should have a firm grasp of yet another piece of the WebRTC puzzle—data transfer.
- It is an often overlooked, yet extremely powerful part of the WebRTC specification that can bring new light to web-based applications.
- In this session, we have used the data channel to bring text-based chat to our currently running WebRTC demo.
- Now, you can officially say goodbye to our communication application, as this is the last time you will see it while reading this course.

Summary

- It is quite an amazing feat that we built over the course of just a few hundred pages.

- Not only have we built a multiuser application, but also one with many of the features found in popular communication applications built and used all over the world.

- This is really the power of not only web applications but also WebRTC.

Summary

- The first easy jump would be adding validation around the text input for each user. From there, it is easy to add the transfer of any data, even building a game that each player can control.

- In the upcoming sessions, we are going to start diving into more advanced applications and examples of WebRTC.

- This will cover topics such as actually sharing files between two users, developing WebRTC application for mobiles and also building networks with more than two users.