Computer Networks: Design Proposal

Team: Eklavya

Abhishek Gambhir
2011CS1002
gmabhishek@iitrpr.ac.in

Prakhar Asthana
2011CS1027
prakharas@iitrpr.ac.in

Prateek Singh
2011CS1028
sprateek@iitrpr.ac.in

Abstract

Chatterbox is the device which can be used to chat using text messages using ad hoc medium. Interaction between chatterboxes is the example **M**obile **A**d-hoc **Net**work(MANET). In this paper, we discuss the routing protocols and application interface of Chatterbox.

Assumptions

- All devices in the network are in *promiscuous receive* mode i.e. hardware should deliver every received packet to network driving software without filtering based on link layer addressing.
- Each device in network is recognized by an unique single IP called home IP address. This IP is either provided by the manufacturer itself or it can be obtained from the central server on internet when device operates for the very first time(This puts a constraint that device should be connected to internet for the very first time). All the ad hoc communication happens through home IP address of device rather than 128-bit physical address.

Routing Protocol

We considered and discussed several routing protocols for our purpose. These include Destination Sequenced Distance Vector(DSDV), Temporarily Ordered Routing Algorithm(TORA), Dynamic Source Routing(DSR), Ad-hoc On Demand Distance Vector(AODV) and Zone Routing Protocol(ZRP). Out of these we think DSR is best suited for our purpose. Reasons for making this choice are discussed in section : Meeting Design Requirements(Networks related)

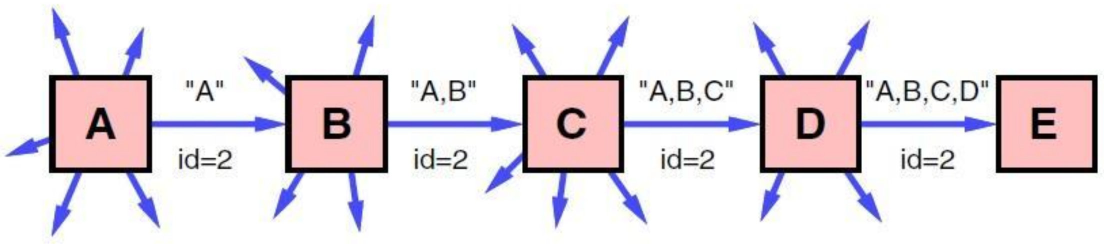
DSR is an on demand source routing protocol [1]. Its mechanism is called source routing because it is the source which controls the path a packet follows to the destination rather than intermediate nodes. It is called on-demand because routes are discovered only when source wants to send some information to the destination. There are no periodic updates involved. DSR consists of two basic subroutines called route discovery and route maintenance.

Route Discovery

Route discovery is fairly simple in DSR. When a source wants to send message to destination, it first searches for a path to destination in its cache. If there is no such path, it broadcasts ROUTE REQUEST message. Each ROUTE REQUEST message has address of both initiator(source) and target(destination) of route discovery subroutine and it also consists of a unique *request id.*
If ROUTE REQUEST reaches a node, which has already seen a REQUEST from the same initiator having the same request id, it discards the REQUEST.

If ROUTE REQUEST reaches the target (destination) of the route discovery, it sends a ROUTE REPLY message to the initiator of ROUTE REQUEST.

If ROUTE REQUEST reaches a node other than target node, it searches for a path to target in its cache. If such a path is found, it appends the path in ROUTE REQUEST, and send a ROUTE REPLY to the initiator of ROUTE REQUEST. Otherwise it appends its address in the ROUTE REQUEST and retransmits it.



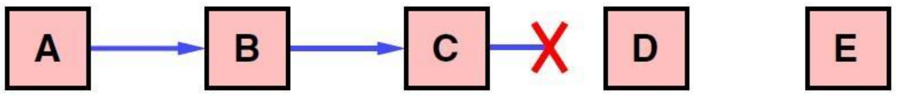
Route Discovery example: Node **A** is the initiator, and node **E** is the target.

If the link between nodes is bi-directional, initiator of ROUTE REPLY can use the reverse of the path discovered in ROUTE REQUEST to send ROUTE REPLY to the initiator of ROUTE REQUEST. Otherwise, it can broadcast a new ROUTE REQUEST hoping for discovery of some different path to the original initiator. To avoid infinite looping of ROUTE REQUEST message in this case,. ROUTE REPLY is piggybacked on ROUTE REQUEST.

Thus when ROUTE REPLY message is received by source, it knows what path a data packet will follow.

Route Maintenance

Route maintenance is called when a some node in path from source to destination moves out of the communication zone. When a node discovers that it is no longer receiving any acknowledgements(either active or passive), it sends a ROUTE ERROR message to the initiator of ROUTE REQUEST. Source on receiving ROUTE ERROR, starts again with route discovery.



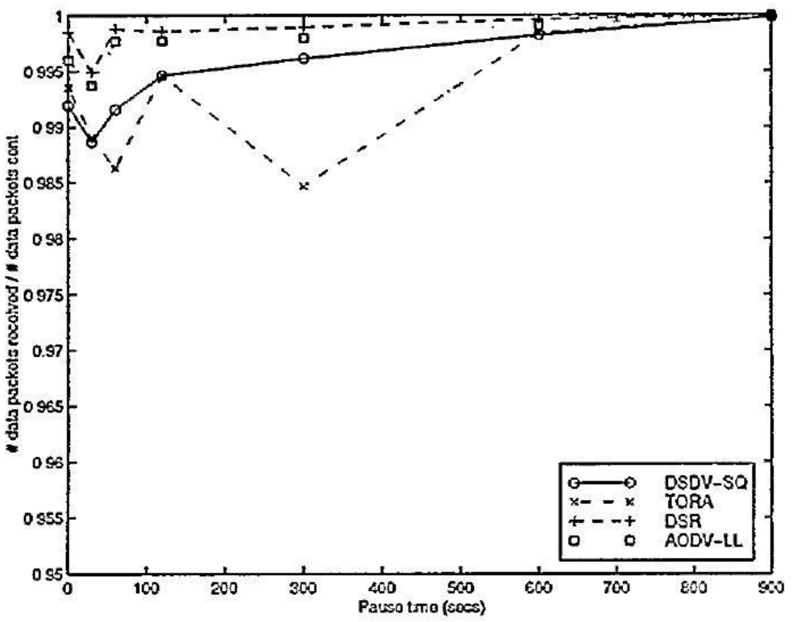
Route Maintenance example: Node **C** is unable to forward a packet from **A** to **E** over its link to next hop **D**.

Meeting Design Requirements(Network Related)

Users should be able to see message seconds after it is sent.

A high packet delivery ratio, less average end-to-end delay and higher probability choosing optimal path is required. DSR is found to be better in all these parameters as compared to other protocols as under:

- DSR has a better packet delivery ratio (>0.95) as compared to DSDV, TORA and AODV for upto 30 mobile devices with pause time ranging from 0s (constantly moving) to 900s and mobility rate of 20 m/s. Packet delivery ratio of DSR is ~1 for pause times >900s [2].



Comparison of the fraction of application data packets successfully delivered as a function of pause time. Speed is 1 m/s.

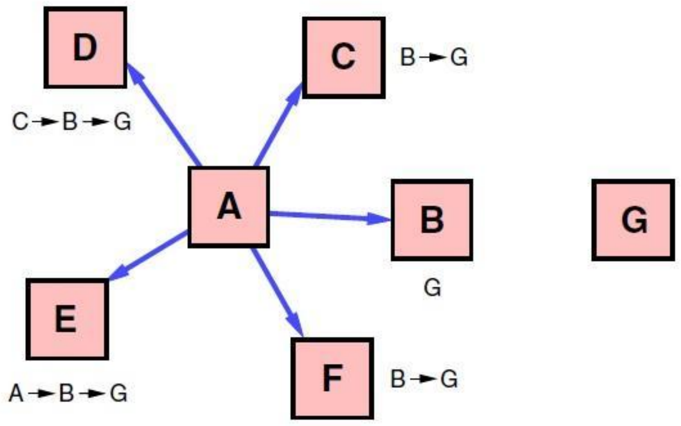
- DSR’s packet delivery ratio is also much better than ZRP’s (0.4) for 50 mobile nodes with pause time upto 150s and mobility rate of 20 m/s.[3]
- For pause times greater than 2 minutes, DSR’s end-to- end delay is close to AODV’s (0.05s) but much better than ZRP’s[3]
- Probability of DSR finding optimal path is higher than that of DSDV,TORA,AODV and ZRP[2].

Chatterbox should work to the extreme of scales : In a stadium with tens of thousands of users, as well as in much sparser environments.

DSR has very high routing overhead as compared to other protocols. Overhead of No- cache version of DSR (devices do not store any routing information whatsoever) has been calculated to increase linearly[4].

For >500 sources with 30% active sources, DSR performs poorly and its packet delivery ratio drops below 0.1 (AODV performs just a little better)and end to end delay increases upto 50s. [5]. But several optimizations in DSR has been suggested. One such optimization is including only source and destination address in header instead of whole path. This can result in up to 47% less control packets in stable networks(devices are stationary) and up to 4% in case of continuously moving packets[6]. Also various caching strategies have been suggested to improve DSR’s packet delivery ratio by 15%.[7].

- **Preventing ROUTE REPLY storms [1]:** ROUTE REPLY storms in DSR happen when many neighbours of some node have path to the target in their cache. Then upon receiving ROUTE REQUEST, each one of them will try to send a ROUTE REPLY when only one of them is best of all. This can result to local congestion and collision between packets.



A ROUTE REPLY storm could result if many nodes all reply to the same ROUTE REQUEST from their own Route Caches. The route listed next to each node shows the route to destination **G** currently listed in that node’s Route Cache.

To prevent this, each node waits for a random time $t= H * (h-1+r)$, before sending the ROUTE REPLY. Here

h=No. of hops in the route to be returned by this node's ROUTE REPLY
H=A small constant which is at least twice the maximum wireless propagation delay
r=A random number generated between 0 to 1

This delay randomizes the time at which each node will send ROUTE REPLY with nodes having shorter paths sending ROUTE REPLY first. During the waiting time, each node promiscuously listen for ROUTE REPLY from any other node with the same initiator id. If it sees that some other node has already transmitted ROUTE REPLY, which also has the same initiator id, it infers that the initiator of ROUTE REQUEST has already received a shorter path to the target from some other node and thus it cancels its ROUTE REPLY transmission.

We believe that in the stadium scenario, DSR can work fairly well if we apply the above mentioned optimization techniques along with the regular protocol.

Device should be able to talk to servers on internet if in range of some wifi access point

To incorporate provision of devices talking through internet also, each device generates *locally unique* two *interface indices* , one each for ad hoc and wifi.We here assume that chatterbox has two network interfaces. But even if its not, there will be no problem. Interface indices in that case will still provide a logical separation, if not physical, for protocol to work. We also assume that device connects to wifi router (if in range) using standard protocol.

We modify ROUTE REQUEST message a little. When each device append its address in the ROUTE REQUEST message, it also appends the interface indices of the corresponding interface on which ROUTE REQUEST packet was received and is now being transmitted. Also whenever a node transmits ROUTE REQUEST packet, it does so on both ad hoc and wifi(if possible).

When this ROUTE REQUEST message reached the intended receiver, it reverses the path in ROUTE REQUEST and sends ROUTE REPLY over that path, assuming links are bidirectional. If not, then it piggybacks the ROUTE REPLY over new ROUTE REQUEST message to the source, as stated earlier.

Thus when the initiator of ROUTE REQUEST receives the ROUTE REPLY, it knows the exact path and medium(adhoc/wifi) a packet will follow from it to destination and can thereby decide which path to choose. We think source should choose the path which is redirected to internet for minimum number of times because ad hoc networks are generally faster in close proximity as compared to internet.

During these transmissions, a device receives all the packets addressed to its home IP address even if it is currently connected to the internet and its present IP is not equal to home IP. This includes packets of ROUTE REQUEST, ROUTE REPLY and chat messages. Thus network card in device will recognize two IPs: current and home(They may be same).

- **Controlling Internet Radius:** While in ad hoc mode, the device is able to send to its nearby devices only. Ideally, we want this thing also, when device is sending any packet via internet too. Even when we are sending a packet through internet, we want to send it to a person who is within approximately 10 km^2 radius. So, when we are broadcasting ROUTE REQUEST packet via internet, we don’t want it to be a full broadcast. We don’t want a ROUTE REQUEST message from India to be broadcasted to USA. So we want it to be a broadcast limited by region. For that we use GPS receiver in device. When a new ROUTE REQUEST message is initiated, the initiator appends its GPS coordinates in it. Whenever some node relays this message to internet, it sends this message to the central server on the internet. Central server reads the GPS coordinates of the initiator of ROUTE REQUEST and then transmits ROUTE REQUEST to only those devices (connected to internet), which fall in the 10 km^2 radius of the initiator. This however, requires devices connected to internet to inform central server about their GPS coordinates periodically.

Chatterbox should work with changes in connectivity due to mobility

Changes in ad hoc connectivity are handled by route maintenance subroutine while changes in wifi connectivity due to movement of users can be handled by one of indirect or direct routing approach.

Protocol for Group Chatting

Group chatting starts by some user creating the group and adding other people in it as its members. In group chatting any user can add any other user he/she knows **without asking for permission.** This design decision is made to avoid the overhead and complexity of confirmation. But if a user does not like a group, he/she always has an option to leave the group.

For group chat there is no buffer maintained on any device for chatting messages because waiting for chatting messages while chat is still going on will pause the chat for that user. So, the message which comes is directly transferred to application layer.

For a message which comes later due to some delay in the network, it will be shown in the chat window when it arrives but it will also have its time stamp on it of the time when the sender sent it indicating its order to the receiver, we can not show message in sorted order if it is too much delayed because for that we will have to sort the messages which is costly application to do for every out of order message.

Need for state of a device

Let us consider a case when a device A in the group is unable to get a message from a device B in the group due some network delay , now another device C sends a message to the group which is supposed to reach A also , but C wants to know if A also have same messages as C or it has something missing , if there is something missing then there is inconsistency between chat window A with the chat window of C, then C will provide some missing messages to A from its own message list . For this purpose we define ***state*** of A which contains the fingerprints of last K messages of A, here K may be a small integer like between 5-10. Now fingerprints of both C and A will be matched in order to check for consistency between both the devices.

For fingerprinting we need an efficient and good hash function such that even for a small change in the message will reflect a huge change in the fingerprint, for this purpose we took idea from MOSS (software used to detect copies among documents), and used its winnowing [8] technique for fingerprinting as it is researched and known to be working good .

1. Every device in the group will have a ***state*** which will increase as the group chat progress. Whenever a device wants to send a message, first it sends a STATE_REQUEST message to all the group members. STATE_REQUEST message contains state of the group **according to sending device.**
2. On receiving a STATE_REQUEST message, devices reply with the STATE_DECLARE message. STATE_DECLARE message contains the state of the group **according to devices which received STATE_REQUEST**.
3. When STATE_DECLATE reaches initiator of STATE_REQUEST, it checks which group members have their state different from its.
4. In case of unequal states, two situations may arise based on the initiator of STATE_REQUEST having state less or more than that of any group member. Situation #1 is handled first. Afterwards it is followed by handling of situation #2.
5. **Handling Situation #1:** In this case, device will send HUNGRY message to the device which has the highest state among the received STATE_DECLARE messages . On receiving HUNGRY message, a device will send FOOD message to the the initiator of HUNGRY message. FOOD message contains all those messages in the group, **which will make states of both devices equal.**
6. **Handling Situation #2:** In this case device sends the appropriate FOOD messages to the group members which have their states less than this device.
7. At this point, all group members have same state. Therefore the device which started STATE_REQUEST can now deliver its message to the group members.

If any device adds a new device to the group it will have to send its contact details to all other devices in the group in order to make them aware of the new device and will also be responsible for showing recent group chat history to the new device.

Even if the device gets disconnected the other devices will not be able to detect it and will try sending messages to it (whether they get ACK's or not) . This will remove the overhead of detecting the disconnected node ,but introduces an unnecessary congestion by sending trying to send messages to disconnected device , but we considered this congestion as okay because if the device would not have been disconnected then the congestion would have been the same .

When a user leaves a group his/her device's entry gets removed from every other device's table and none of them will send him the messages . Note that disconnected is different from leaving the group.

Q: How to define state?
A: We propose that state be defined as a function which is monotonically increasing wrt to message count in group and also depends upon the latest k messages in group.

Potential problems to be considered

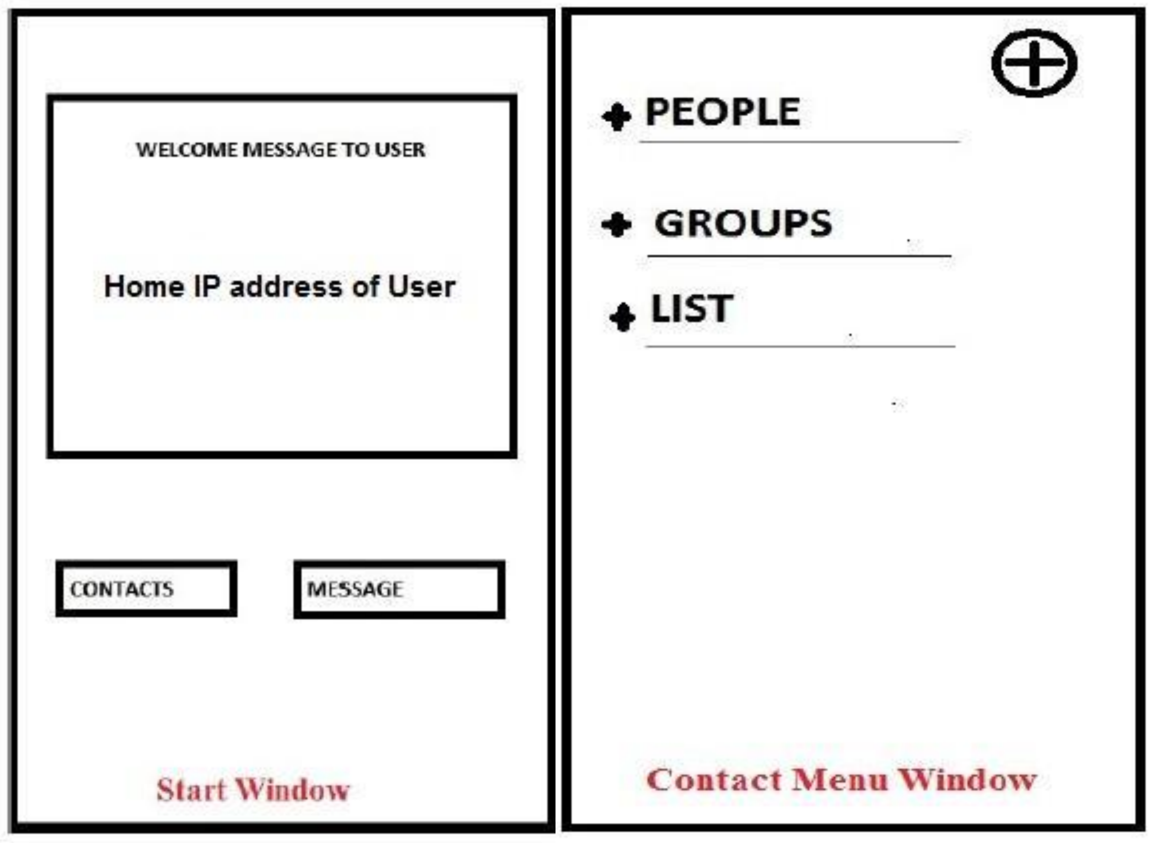
If assume half of the group gets disconnected from the other half but none of them left the group, now both halves will be able to communicate within their half group but not with other and hence their chat windows will contain different messages.
How to deal with the halves when they again come in the range but have different view of the window and may have added different new members also?

Application Interface

Maintaining Contact Number

Before sending a message, user should know the **home address IP** of the recipient just like in case of telephone communication.

The **Start window** below appears when user turns on the devices

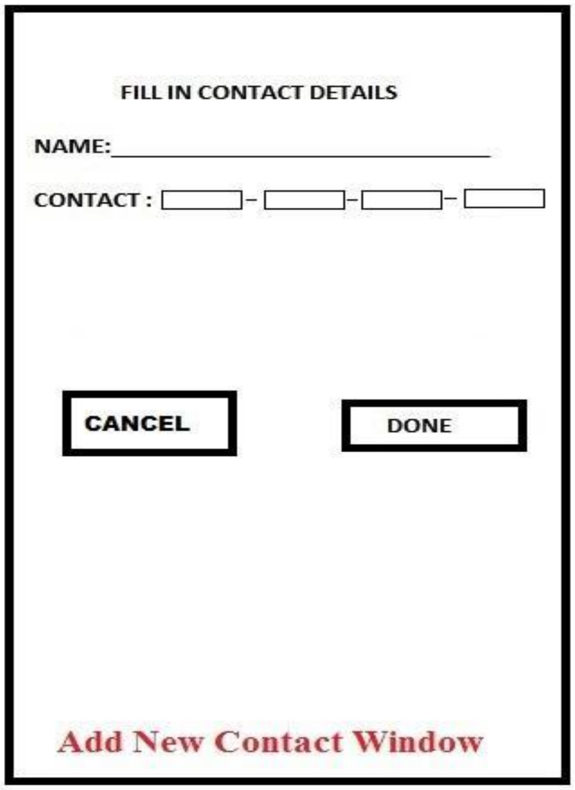


As explained earlier home IP address of user is displayed in Start Window. Here we have 2 buttons: Contacts and Message. Tapping on Contacts will result in appearance of **Contacts Menu Window.**

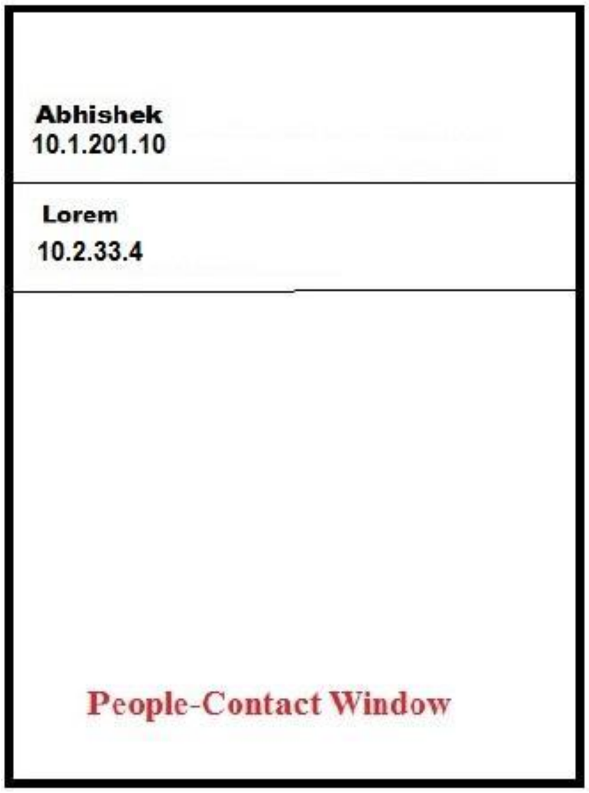
Here contacts are displayed in 3 categories. People, Groups and List.

“PEOPLE” is analogous to normal contacts, where details of a contact is shown. Tapping on “GROUPS” will show all groups in which the user is involved. “List” is just a collected of contacts which user made in order to facilitate adding many people at once in senders list of a message. e.g. On occasion on new year, user may want to send message to many of his friends at one go.

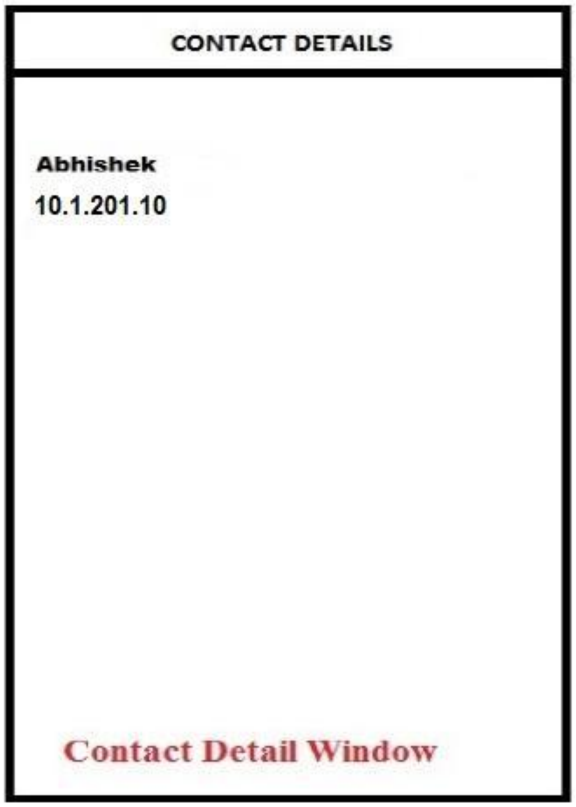
Tapping on the **+** sign in Contact Menu Window will open **Add New Contact Window**. Tapping on “PEOPLE” will open **People-Contact Window.** Tapping on any contact in People-Contact Window will open **Contact Detail Window** while holding on any contact will result in “pop up” of a **Option Mini Window.** If contact detail is tapped in Contact Detail Window, it will open in Edit Mode.



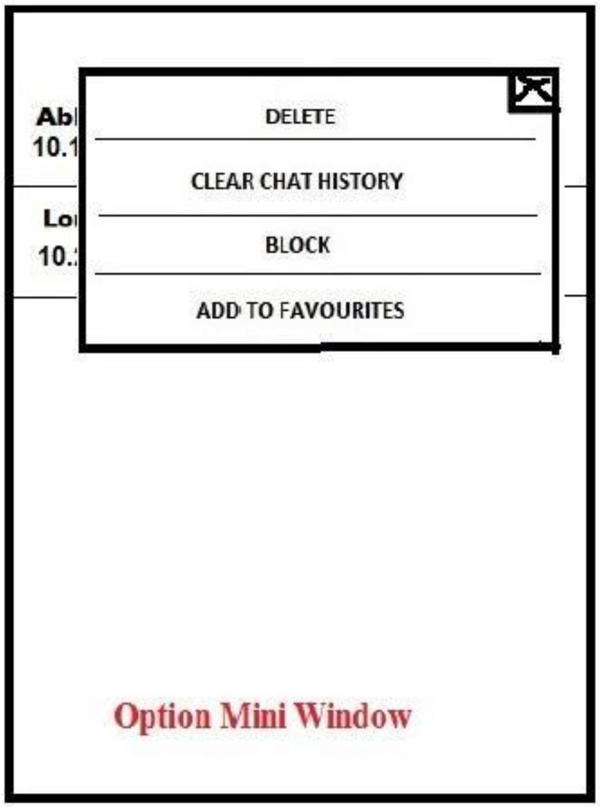
Add New Contact Window



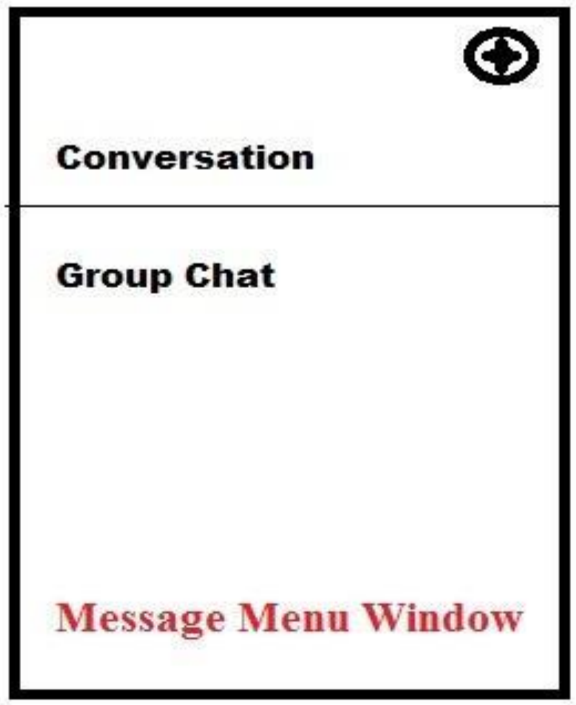
People-Contact Window

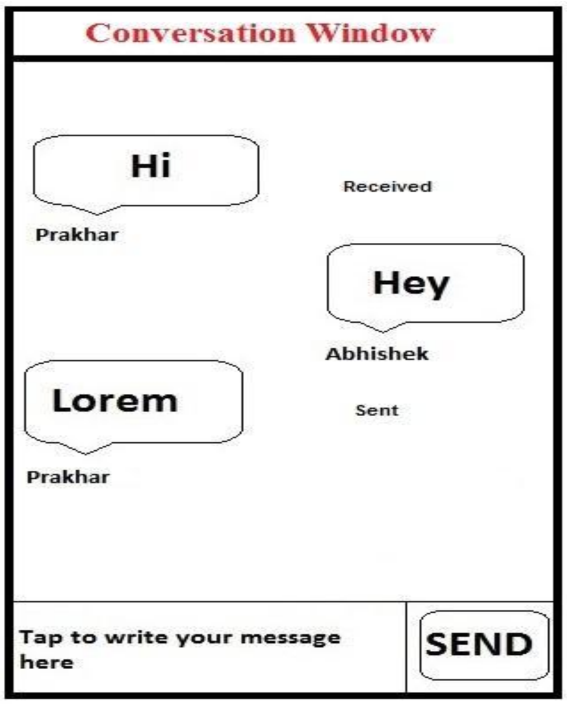


Contact Detail Window

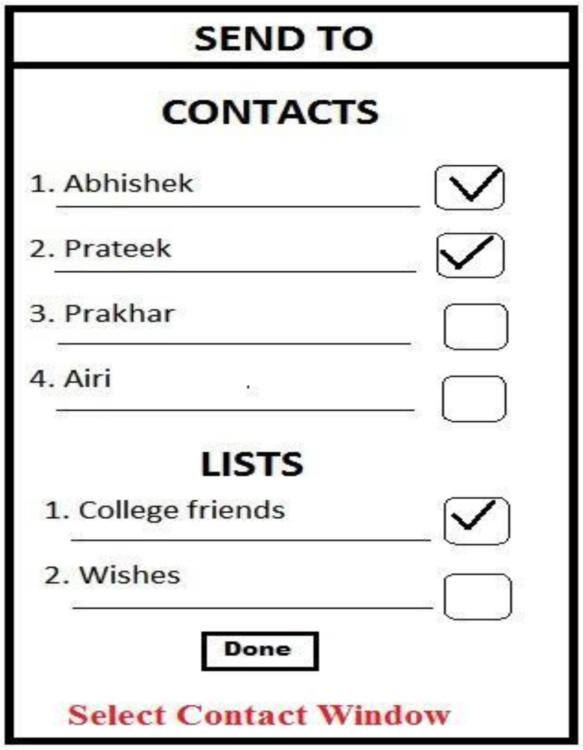


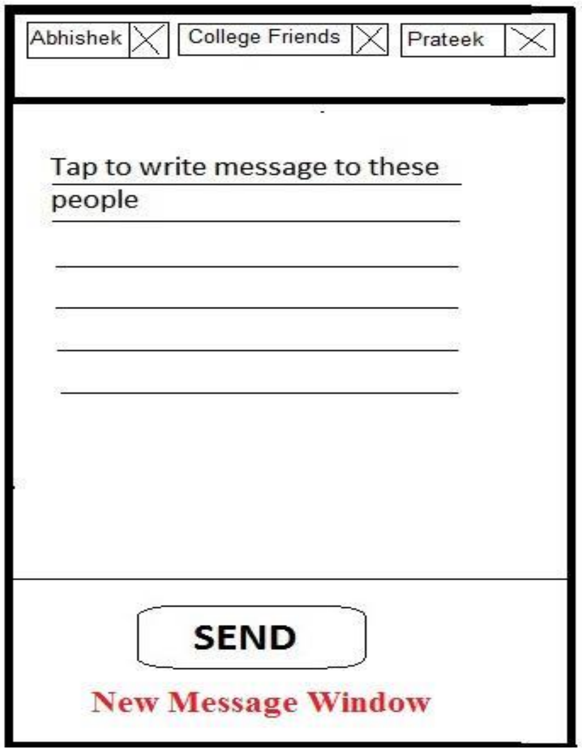
Option Mini Window



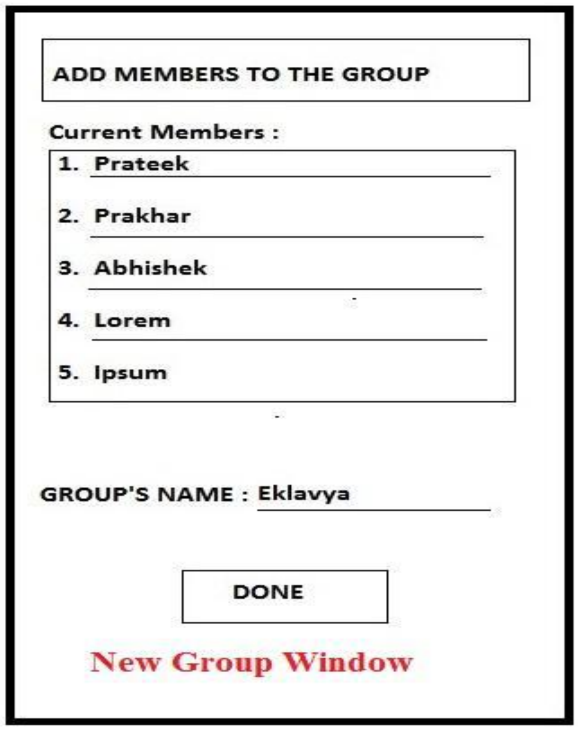


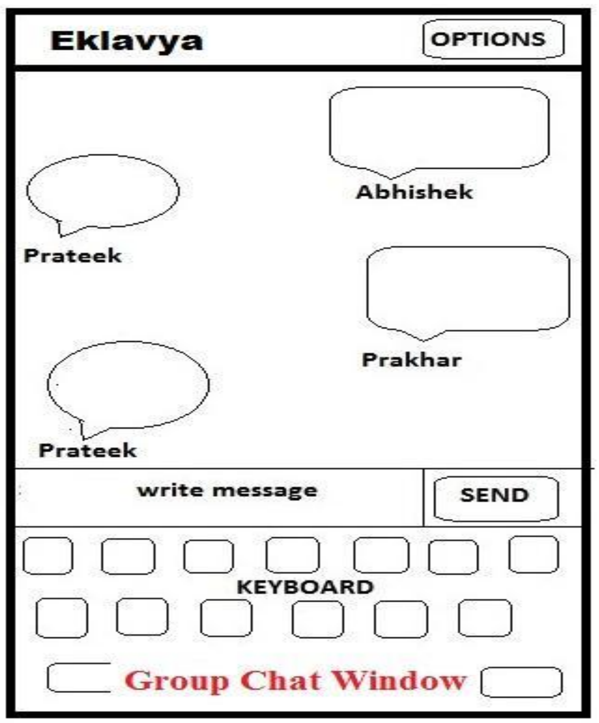
Tapping on “Message” in Start Window will open **Message Menu Window.** In message menu window we have two options “Conversation” and “Group Chat”. Tapping “Conversation” will list all the *one-on-one* conversation that user had in his device’s history. Tapping a particular conversation will open **Conversation Window.** There’s also a **+** sign on top of message menu window, tapping which will allow composing new message in **New Message Window.** To send from New Message Window, contacts can be selected from **Select Contact Window**





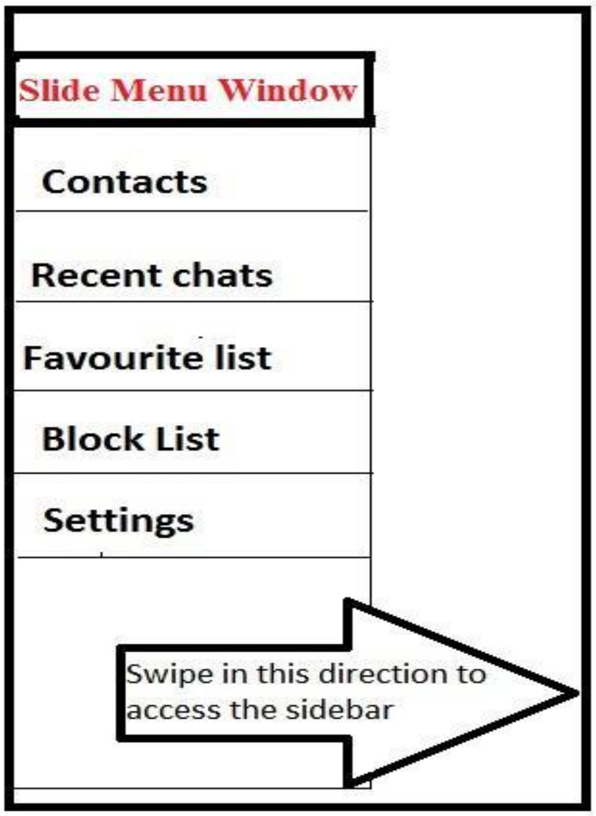
Selecting groups in Contact Menu Window will list all the groups. User can also create a new group in **New Group Window.** Group chats become visible by tapping “Group Chat” in message menu window . A typical **Group Chat Window** is shown below

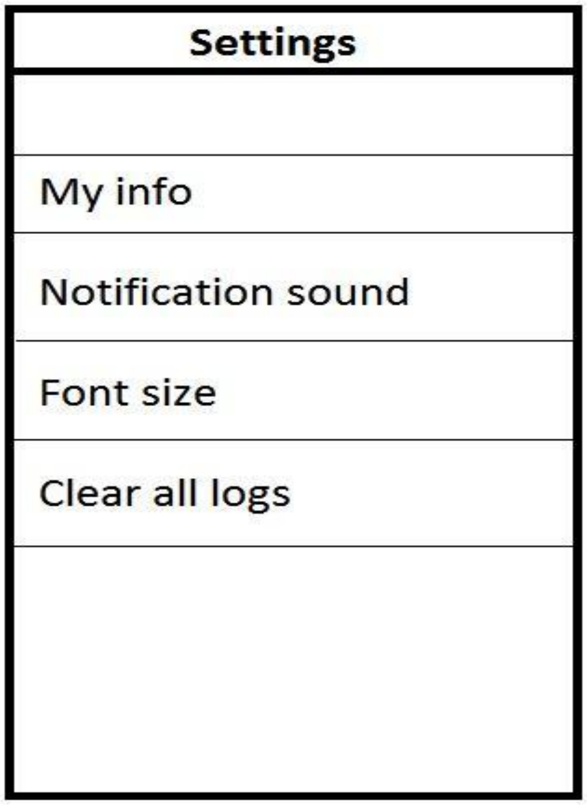




Here “Options” in Group Chat Window include option of leaving group and seeing which people are members of group.

To see blocked list, favourite conversations and other settings, user can always slide right so that a **Slide Menu Window** appears. From here user can also block/unblock contacts or tune settings of chatterbox





Meeting Design Requirements(Application Related)

Users should be able to restrict message from particular individuals.

Chatterbox will drop any packet sent by user currently in the blocked list of receiver **unless the packet belongs in a group chat.**

Users should be able to send message to single person or many persons

This is easily achieved. Default message sending is to a single person. But messages to many persons can be sent at once by creating a list of those persons or from **New Message Window.**

Messages should be presented to users in sensible order

This can be achieved by appending timestamp of sender in the message. But in a group chat, to maintain coherency, we need to make sure all devices are in perfect sync wrt time. This can be achieved by making sure that any new device is perfectly synced with old ones and device shuts down if the battery level falls below a particular level and that particular level is such that device's internal clock keeps running for a long time.

Conclusion

We chose DSR for chatterbox design project because of its high packet delivery ratio but it fails to scale much because of high overhead packets. Another reason for choosing DSR was that it can easily be extended to scenarios where devices can also connect to the internet. We also discuss functionality of group chats in our proposal but problem of group getting divided into two exclusive sets is still there which we believe can only be solved when some central server resides in the vicinity of devices so that it can store the messages being exchanged in the group and provide the whole chat history whenever it is required.

References

1. Johnson, David B., David A. Maltz, and Josh Broch. "DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks." *Ad hoc networking* 5 (2001): 139-172.
2. Broch, Josh, et al. "A performance comparison of multi-hop wireless ad hoc network routing protocols." *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1998.
3. Mittal, Shaily, and Prabhjot Kaur. "Performance Comparison of AODV, DSR and ZRP Routing Protocols in MANET's." *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on*. IEEE, 2009.
4. Santivanez, Cesar A., et al. "On the scalability of ad hoc routing protocols."*INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. IEEE, 2002.
5. Naumov, Valery, and Thomas Gross. "Scalability of routing methods in ad hoc networks." *Performance Evaluation* 62.1 (2005): 193-209.
6. Tamilarasi, M., et al. "Scalability Improved DSR Protocol for MANETs."*Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on*. Vol. 4. IEEE, 2007.
7. Marina, Mahesh K., and Samir Ranjan Das. "Performance of route caching strategies in dynamic source routing." *Distributed Computing Systems Workshop, 2001 International Conference on*. IEEE, 2001.
8. Schleimer, Saul, Daniel S. Wilkerson, and Alex Aiken. "Winnowing: local algorithms for document fingerprinting." *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003.