Implementation of web socket over p2p network as a review paper

Ms. Kalyani Dhodre[1], Mrs. Hemlata dakhore[2]
Department of Computer Science & Engineering, G.H.R.I.E.T.W.,
Rashtrasant Tukdoji Maharaj Nagpur University,
Nagpur, India
[1]kalyanidhodre@gmail.com, hemlata.dakhore@raisoni.net

Abstract:- P2p streaming has been poular and is expected to attract even more users.The proposed scheme can achieve high bandwidth utilization and optimal streaming rate possible in ap2p streaming system.
The prototype implementing the queue based scheduling is developed and used to evaluate the scheme in real network.between one or more number of clients running untrusted code into controlled environment to a remote host that has opted into communication from that of the code p2p network which is use in case of The distribution ofthe videos.where proposed design which enables flexiblecustomization of video streams to support heterogeneousof receivers, highly utilizes upload bandwidth of peers,and quickly adapts to network and peer dynamics.

Keywords—*peer to peer coding, scalable video coding, network coding, web socket*

*****

I. INTRODUCTION

From past of the time, making web applications that need bidirectional correspondence between a customer and a server (e.g., texting and gaming applications) has required a misuse of HTTP to survey the server for redesigns while sending upstream notices as unmistakable HTTP calls .This can be given by WebSocket Protocol The capacity to accomplish high spilling rate is attractive for P2P gushing. Higher spilling rate permits the framework to show video with better quality. It likewise gives more cusion to ingest the transfer speed varieties brought about by associate agitate and system blockages when consistent bit rate (CBR)video is shown. The way to accomplish high gushing rate is to better use companions' transferring data transmission. In this area, we propose a line based lump booking calculation that can accomplish near 100% companions' transferring transmission capacity usage in viable P2P organizing environment.In P2P framework, the asset use is determinedby the overlay topology and aggregate conduct of piece planning at individual associates. At framework level, line based versatile piece booking requires completely associated network among taking an interest peers. At companion level, information lumps are pulled/pushed from server to peers, stored at associates' line, what's more, handed-off from companions to its neighbors. The accessibility of transfer limit is deduced from the line status, for example, thequeue size or if the line is vacant. Signs are gone amongst associates and server to pass on the data if a companion's transfer limit is accessible.

II. ADAPTIVE QUEUE-BASED CHUNK SCHEDULING

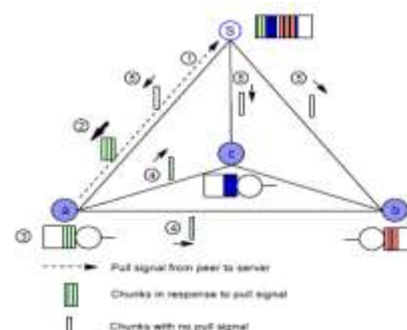

Fig. 1.Queue-based P2P system with four nodes.

1. peera sends pull signal to the content source server;
2. content source server send three chunks in response to the pull signal;
3. three chunks are cached in the forward queue;
4. cached chunks are forwarded to neighbor peers;
5. duplicate chunk is sent

Fig. 1 depicts a P2P streaming system using queue-based chunk scheduling with one source server and three peers. Each peer mainstains several queues including a forward queue. Using peer as an example, the signal and data flow is described next. Pull signals are sent from peers a to the server whenever the queues become empty (or have fallen below a threshold) (step 1 in Fig. 1). The server responds to the pull signal by sending three data chunks back to peer a (step 2).These chunks will be stored in the forward queue (step 3) and be relayed to peer b and peer c (step 4). When the server has responded to all 'pull' signals on its 'pull' signal queue, itserves one duplicated data chunks to all peers (step 5). Thes data chunks will not be stored in forward queue and will not be relayed further.

Next we first describe in detail the queue-based scheduling mechanism at the source server and peers.

A. Peer side scheduling and its queuing model

Fig. 2 depicts the queuing model for peers in the queuebased scheduling method. A peer maintains a playback buffer that stores all received streaming content from the source server and other peers. The received content from different

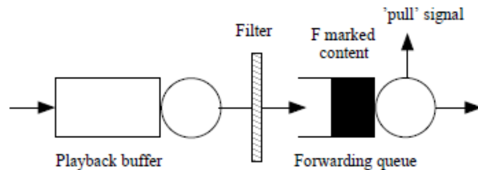

Fig. 2. Queue Model of Peers

nodes is assembled in the playback buffer in playback order.The peer's media player renders/displays the content from thisbuffer. Meanwhile, the peer maintains a forwarding queue which is used to forward content to all other peers. The received content is partitioned into two classes: F-marked content and NF-marked content. *F (forwarding)* represents content that should be relayed/forwarded to other peers.

NF(nonforwarding)

indicates that content is intended for this peer only and no forwarding is required. The content forwarded byneighbor peers is always marked as *NF*

B. Server side scheduling algorithm and its queuing model

Fig. 3 illustrates the server-side queuing model of the decentralized method. The source server has two queues: a content queue and a signal queue. The content queue is a multi-server queue with two dispatchers: an F-marked content dispatcher and a forward dispatcher. The dispatcher that isinvoked depends on the control/status of the 'pull' signal queue. Specifically, if there is 'pull' signal in the signal queue, a small chunk of content is taken from the content buffer.This chunk of content is marked as *F* and dispatched by the F-marked content dispatcher to the peer that issued the 'pull'signal. The 'pull' signal is then removed from the 'pull' signal queue. If the signal queue is empty, the server takes a small chunk of content from the content buffer and puts that chunk of content into the forwarding queue to be dispatched. The forwarding dispatcher marks the chunk as *NF* and sends it to all peers in the system.

C. Proof of optimality for queue-based chunk scheduling

It demonstrate that the line based planning strategy for both the companion side and the server-side accomplishes the greatest P2P live spilling rate of the framework. Given a substance source server and an arrangement of companions with known transfer limits, the greatest gushing rate, rmax, is administered by the accompanying recipe The main case is termed as server asset poor situation where the server's transfer limit is the bottleneck. The second case is termed as server asset rich situation where the companions' normal transfer limit is the bottleneck.Assume that the sign proliferation delay between an associate and the server is immaterial and the information substance can be transmitted at a self-assertive little sum, then the line based decentralized booking calculation as depicted above accomplishes the most extreme gushing rate conceivable in the framework. Evidence: Suppose the video substance is partitioned into little lumps. The server conveys one lump every time it serves a "force" signal. A companion issues a draw sign to the server at whatever point the sending line gets to be unfilled. _ indicates the piece size.For peer i, i = 1, 2, n, it requires some serious energy of (n − 1)_/uito forward one information lump to all associates. Let ribe the greatest rate at which the "force" sign is issued from associate i. Consequently ri= ui/(n − 1)The greatest amassed rate of "draw" sign got atServer, It takes server _/us toserve a force signal. Henceforth the most extreme "force" signal rate a server can oblige is us/_. Presently consider the accompanying two situations/cases:

In this situation, the server can't deal with the "draw" signal at most extreme rate. The sign line at the server side is thus never unfilled and the whole server transfer speed is utilized to transmit F-checked substance to peers. Interestingly, a companion's forward queuebecomes unmoving while sitting tight for the new information content from the source server. Since every companion has adequate transfer data transmission to hand-off the F-checked substance (got from the server) to all different associates, the associates get content conveyed by the server at the most extreme rate. Henceforth the gushing rate is steady with the Equation (1) and the most extreme spilling rate is reached.In this situation, the server has the transfer ability to benefit the "force" signals at the greatest rate. Amid the time frame when the "force" signal line is vacant, the server transmits copy NF`-checked substance to all companions. The server's transfer transmission capacity used to serve NF-stamped substance is along these lines For every individual associates, the situation in which the server is asset rich portrayed previously. Once more, the gushing rate achieves the upper bound as showed in Equation (1). This finishes up the verification. Note that on the off chance that 2 where the total "force" signal landing rate is littler than the server's administration rate, it is accepted that the companions get F-stamped content promptly subsequent to issuing the "draw" signal. The above suspicion is genuine just if the "draw" signal does not experience any lining defer and can be adjusted promptly by the substance source server. This implies (i) no two "force" signals touch base at precisely the same and (ii) a "draw" sign can be overhauled before the landing of next approaching "draw" signal. Suspicion (i) is regularly utilized as a part of lining hypothesis and is sensible since a P2P framework is a disseminated framework as for associates creating "pull" signals. The likelihood that two "force" signals touch base at the very same time is low.

III. RELATED WORK

A. Protocol Overview

The protocol has two parts: a handshake and the data transfer. The handshake from the client looks as follows:

GET /chat HTTP/1.1

Host: server.example.com

ISSN: 2321-8169
749 - 754

Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
The handshake from the server looks as follows:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket

Connection: Upgrade. Combined with the WebSocket API , it provides an alternative to HTTP polling for two-way communication from a web page to a remote server. Then same technique can be The Web Socket Protocol is designed to supersede the existing bidirectional communication technologies that use HTTP as a transport layer to benefit from existing infrastructure (proxies, filtering, authentication).Such technologies were implemented as trade-offs between efficiency and reliability because HTTP was not initially meant to be used for bidirectional communication[2]. The Web Socket Protocol attempts to The address the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructure dedicated port without reinventing the entire protocol. This last point is important because of the traffic patterns of interactive messaging do not closely match standard HTTP traffic and can induce unusual loads on some components. it will significantly improve their performance. it present the design of a P2P streaming system that employs both scalable video coding and network coding where design is modular and can be used as an improvement plug- in other P2P streaming systems. The p2p mechanism can potentially achieve a very high efficiency of data exchange between end devices,its very useful in particular network infrastructure[1].In addition, we quantitatively show the expected performance gain from the proposed design using actual scalable video traces in realistic P2P streaming environments with high churn rates, heterogeneous peers, and flash crowd scenarios.In particular, our results show that the proposed system can achieve (i) significant improve-ment in the visual quality perceived by peers (several dBs are observed), (ii) smoother and more sustained streaming rates (up to 100% increase in the average streaming rate is obtained), (iii) higher streaming capacity by serving more requests from peers (iv) more robustness against high churn rates and
flash crowd arrivals of peers[1].

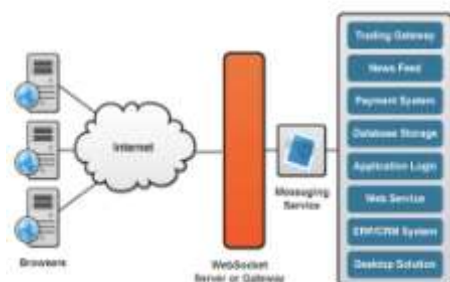

Fig1.Architecure of web socket protocol server.

IV. RELATED WORK

A. Protocol Overview

The protocol has two parts: a handshake and the data transfer. The handshake from the client looks as follows:
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
The handshake from the server looks as follows:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat

The leading line from the client follows the Request-Line format..TheRequest-Line and Status-Line productions are defined intoAn unordered set of header fields comes after the leading line in the both cases. The meaning of these header fields is specified in the Section 4 of this document. Additional header fields may also be present, such as cookies. The format and parsing of headers is as defined in the client and server have both sent their handshakes, and if the handshake was successful, then the data transfer part starts. This is a two-way communication channel where each side can, independently from the other, send data at will.[1]After a successful handshake, clients and servers transfer data back and forth in conceptual units referred to in thatof specification as "messages". On the wire, a message is composed of one or more frames. The WebSocket messages does not necessarily correspond to a particular network layer framing, as a fragmented message may be coalesced or split by an intermediary. A frame has an associated types. Each frame belonging to the same message contains the same type of data. Broadly speaking, there are types for textual data , binary data (whose interpretation is left up to the application), and control frames (which are not intended to carry data for the application but instead for protocol-level signaling, such as to signal that the connection should be closed). This version of the protocol defines six frame types and leavesit in tenreserved for future use.

B. Opening Handshake

The opening handshake is intended to be compatible with HTTP-based server-side software and intermediaries, so that a single port can be used by both HTTP clients talking to that of Theserver and WebSocketclients talking to that of The server. To this end, the WebSocket client's handshake is an HTTP Upgrade request:
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com

 ISSN: 2321-8169
749 - 754

Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

In consistence with, header fields in the handshake might be sent by the customer in any request, so the request in which diverse header fields are gotten is not huge. The "Solicitation URI" of the GET strategy is utilized to recognize the endpoint of the WebSocket association, both to permit various areas to be served from one IP deliver and to permit different WebSocketendpoints to be served by a solitary server. The customer incorporates the hostname in the |Host| header field of its handshake according to , so that both the customer and the servercan check that they concede to which host is in use.TheWebSocket Protocol in December 2011 TheAdditional header fields are utilized to choose choices into the WebSocketProtocol. Run of the mill alternatives accessible in this rendition are the subprotocol selector (|Sec-WebSocket-Protocol|), rundown of expansions backing by the customer (|Sec-WebSocket-Extensions|), |Origin| header field, and so forth. The |Sec-WebSocket-Protocol| ask for header field it canbeused to demonstrate what a subprotocols (application-level conventions layered over the WebSocket Protocol) are worthy to the client.The server chooses one or none of the satisfactory conventions and echoes that esteem in its handshake to show that it has chosen that convention. Sec-WebSocket-Protocol: visit The |Origin| header field is utilized to ensure againstunauthorized cross-root utilization of a WebSocket server by scripts utilizing the WebSocket API as a part of a web program. The server is educated of the script birthplace producing the WebSocket association demand. On the off chance that theserver does not wish to acknowledge associations from this birthplace, it can dismiss the association by sending a suitable HTTP mistake code. This header field is sent by program customers; for non-browserclients, this header field might be sent in the event that it bodes well with regards to those clients.Finally, the server needs to demonstrate to the customer that it got the customer's WebSocket handshake, so that the server doesn't acknowledge associations that are not WebSocket associations. This keeps an aggressor from deceiving a WebSocket server by sending it carefullycrafted parcels utilizing XMLHttpRequestor a structure submission.Toprove that the handshake was gotten, the server needs to take two bits of data and consolidate them to frame a reaction. The primary bit of data originates from the |Sec-WebSocket-Key| header field in the customer handshake:

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ= For this header field, the server needs to take the worth (as present in the header field, e.g., the base64- encoded adaptation short any driving and trailing whitespace) and link this with theGlobally Unique Identifier "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" in string structure, which is unrealistic to be utilized bynetwork endpoints that don't comprehend the WebSocket Protocol. A SHA-1 hash (160 bits), base64-encoded, of this link is then returned in the server's handshake.

C. Closing Handshake

The closing handshake is far simpler than the opening handshake.Either peer can send a control frame with data containing a specified control sequence to begin the closing handshake (detailed in Section 5.5.1). Upon receiving such a frame, the other peer sends a Close frame in response, if it hasn't already sent one. Upon receiving that control frame, the first peer then closes the connection, safe in the knowledge that no further data is forthcoming. After sending a control frame indicating the connection should be closed, a peer does not send any further data; after receiving a control frame indicating the connection should be closed, a peer discards any further data received.It is safe for both peers to initiate this handshake simultaneously. The closing handshake is intended to complement the TCP closing handshake (FIN/ACK), on the basis of TCP closing handshake is not always reliable end-to-end, especially in the presence of intercepting proxies and other intermediaries.By sending a no of Close frames and waiting for a Close frames in response, certain cases are avoided where data may be unnecessarily lost. For instance, on some platforms, if a socket is closed with The data in the receive queue, a RST packet is sent, which will then cause recv() to fail for the party that received the RST, even if there were data waiting tobe read.

D. Design Philosophy

The WebSocket Protocol is to be designed on the principle that there should be minimal framing (the only framing that exists is to make the protocol frame-based instead of stream-based and to support a distinctions between Unicode text and binary frames). It is expected that metadata would be layered on top of the WebSocket by the application Fette&Melnikov Standards TrackTheWebSocket Protocol December 2011 layer, in the same way this metadata is layered on top of TCP by the application layer (e.g., HTTP). Conceptually, WebSocket is really just a layer on top of TCP that protocol frame-based instead of stream-based and to support a distinction between Unicode text and binary frames. It is expected that the metadata would be layered on top of WebSocket.

Itdesigned in such a way that its servers can be share a port with HTTP servers, by having its handshake be a valid HTTP Upgrade request. One could conceptually use the other protocols to establish client-server messaging, but the intent of WebSockets is to be provide a relatively simple protocol that can coexist with HTTP and deployed HTTP infrastructure (such as proxies) and that is as close to TCP as is safe foruse with such infrastructure given security considerations, with targeted additions to be simplify usage and keep simple things.The protocol is intended to be extensible; future versions will belikely introduce additional concepts such as multiplexing

E. Security Model

The Web Socket Protocol uses the origin model used by web browsers to restrict which web pages can contact a Web Socket server when the Web Socket Protocol is used from a web page. Naturally, when the Web Socket Protocol is used by a dedicated client directly (i.e., not from a web page

 ISSN: 2321-8169
749 - 754

through a web browser), the origin model is not useful, as the client can provide any arbitrary origin string. This protocol is intended to fail to establish a connection with servers of the pre-existing protocols like SMTP and HTTP, while allowing HTTP servers to opt-in to supporting this protocol if Fette&Melnikov Standards Track. The Web Socket Protocol December 2011desired. This is achieved by having a strict and elaborate handshake and by limiting the data that can be inserted into the connection before the handshake is finished (thus limiting how much the server can be influenced).

It is similarly intended to fail to establish a connection when data from other protocols, especially HTTP, is sent to a Web Socket server,

for example, as might happen if an HTML "form" were submitted to a Web Socket server. This is primarily achieved by requiring that the server prove that it read the handshake, which it can only do if the

handshake contains the appropriate parts, which can only be sent by a Web Socket client. In particular, at the time of writing of this specification, fields starting with |Sec-| cannot be set by an attacker from a web browser using only HTML and JavaScript APIs such

as XML Http Request [XML Http Request].

F. Relationship to TCP and HTTP

Relationship to TCP and HTTP The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. By default, the WebSocket Protocol uses port 80 for regular WebSocketconnections and port 443 for WebSocket connections tunneled over Transport

Layer Security (TLS).

G. Establishing a Connection

When a connection is to be made to a port that is shared by an HTTP server (a situation that isquite likely to occur with traffic to ports 80 and 443), the connection will appear to the HTTP server to be a regular GET request with an Upgrade offer. In relatively simplesetups with just one IP address and a single server for all traffic to a single hostname, this

might allow a practical way for systems based on the WebSocket Protocol to be deployed. In more elaborate setups (e.g., with load balancers and multiple servers), a dedicated set of hosts for WebSocket connections separate from the HTTP servers is probably easier to manage. At the time of writing of this specification, it should be noted that connections on ports 80 and 443 have significantly different success rates, with connections on port 443 being significantly more likely to succeed, though this may change with time.

H. Subprotocols Using the WebSocket Protocol

The client can request that the server use a specific subprotocol by including the |Sec-WebSocket-Protocol| field in its handshake. If it is specified, the server needs to include the same field and one of the selected subprotocol values in its response for the connection to be established. toavoid potential collisions, it is use to be recommended names that contain the ASCII version of the domain name of thissubprotocol'soriginator. for example carporationwere to create a Chat subprotocol to be implemented by many servers around a Web, they could name it "chat.example.com". If theExample Organization called thecompeting subprotocol "chat.example.org", then the two subprotocols could be implemented by servers simultaneously, with thatserver dynamically selecting which subprotocol to be use based on the value sent by the client. These subprotocols would be considered completely separate by WebSocketclients. Backward-compatible versioning can be implemented by reusing the same subprotocol string but carefully designing the actual subprotocol to support this kind of extensibilitythe data received from others and process this data to be create proper scalable video streams and to ensure smooth videos quality. As senders, peers encodes the videos of data using network coding positions of with parameters based on their own upload capacity as well as the characteristics of the receiving peers. A simplified model for the software architecture of a peer in our system is shown in the Fig. 1. A similar model is used for source nodes, but with some of differences as elaborated later. We do not address the design or optimization of trackers; the function of the tracker is orthogonal to the work to be presented in this paper. We also do not to be addresses other problems in mesh-based P2P streaming systems, including neighbor selection, gossip protocols (for exchanging data availability), incentive schemes, and overlay optimization which all have been heavily researched in the literature. All of the above issues are abstracted in the Connection Manager component in a Fig. 1, while our work is focused on the components in the shaded box in that of figure. The separation and abstraction of functions enable us to the support different P2P streaming systems with minimal changes in our design and code. Therefore, our work is fairly general.

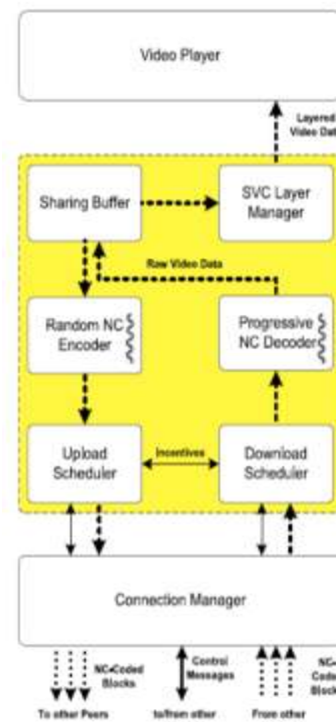

Fig 2: Peer Software Architecture. Dashed arrows denote video data, and solid arrows denote control messages

ISSN: 2321-8169
749 - 754

V. MEASUREMENT AND ANALYSIS OF PROJECT

It's easier to communicate via TCP socketswhen you're working within an intranet boundary, since you likely have control over the machines on that network and can open ports suitable for making the TCP connections.Over the internet, you're communicating with someone else's server on the other end. They are*extremely* unlikely to have any old socket open for connections. Usually they will have only a few standard ones such as port 80 for HTTP or 443 for HTTPS. So, to communicate with the server you are obliged to connect using one of those ports.

Given that these are standard ports for web servers that generally speak HTTP, you're therefore obliged to conform to the HTTP protocol, otherwise the server won't talk to you. The purpose of web sockets is to allow you to initiate a connection via HTTP, but then negotiate to use the web sockets protocol (assuming the server is capable of doing so) to allow a more "TCP socket"-like communication stream.When you send bytes from buffer with a normal tcpthe send function returns the number of bytes of the buffer that were sent. If it is a non-blocking socket or a non-blocking send then the number of bytes sent may be less than the size of the buffer. If it is a blocking socket or blocking send, then the number returned will match the size of the buffer but the call may block. With WebSockets, the data that is passed to the send method is always either sent as a whole "message" or not at all. Also, browser WebSocket implementations do not block on the send call.But there are more important differences are on the receive side of things. When the receiver does a recv (or read) on a TCP socket, there is no guarantee that the number of bytes return correspond to a single send (or write) on the sender side. It might be the same, it may be less (or zero) and it might even be more (in which case bytes from multiple send/writes are received). With WebSockets, the receipt of a message is event driven (you generally register a message handler routine), and the data in the event is always the entire message that the other side sent.Note that you can do message based communication using TCP sockets, but you need some extra layer/encapsulation that is adding framing/message boundary data to the messages so that the original messages can be re-assembled from the pieces. In fact, WebSockets is built on normal TCP sockets and uses frame headers that contains the size of each frame and indicate which frames are part of a message. The WebSocketAPIreassembles the TCP chunks of data into frames which are assembled into messages before invoking the message event handler once per message A It is easy to install so the configuration of computers on of that network, All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources..P2P is more reliable as central dependency is eliminated. The Failurity of one peer doesn't affect on functioning of other peers. In case of Client –Server network, if server goes down whole network gets affected. here is no need for full-time System Administrator. Every user is the administrator of his machine. User can control their shared resources.

5)The over-all cost of building and maintaining this type of network is comparatively very less.

VI. CONCLUSIONS AND FUTUREWORK

In this paper, we showed an implementation of websocket protocol which is a application work as client as well as server.I will also utilizing the algorithm of Adaptive queue based chunk scheduling where it provide full band width utilization in p2p network. Then designing of P2P streaming systems with scalable video coding and network coding can solve both of the above problems. The evaluation study confirms the significant potential performance gain, in terms of visual quality perceived by peers, average streaming rates, streaming capacity, and adaptation to higher peer dynamics. I will explore queue control design space to further improve its performance.

VI. REFERENCES

[1] Mu Mu, Johnathan Ishmael, William Knowles, Mark Rouncefield, Nicholas Race, Mark Stuart, and George Wright:" P2P-Based IPTV Services: Design, Deployment, and QoE Measurement" IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 14, NO. 6, DECEMBER 2012
[2] K. Mokhtarian and M. Hefeeda.Efficient allocation of seed servers in peer-to-peer streaming systems with scalable videos. In Proc. of IEEE International Workshop on Quality of Service (IWQoS'09), pages 1–9, Charleston, SC, July 2009
[3] Z. Liu, Y. Shen, K. Ross, J. Panwar, , and Y. Wang. Substream trading: Towards an open P2P live streaming system. In Proc. of IEEE Conference on Network Protocols (ICNP'08), pages 94–103, Orlando,FL, October 2008.
[4] Z. Wang, H. R. Sheikh, and A. C. Bovik, "No-reference perceptual quality assessment of JPEG compressed images," in Proc. IEEE Int. Conf. Image Processing, 2002
[5] ShabnamMirshokraie, Mohamed Hefeeda School "Live P2P Streaming with scalable video coding and network coding", February 22–23, 2010
[6] L. D'Acunto, M. Meulpolder, R. Rahman, J. A. Pouwelse, and H. J. "Modeling and analyzing the effects of firewalls and NATs in P2P swarming systems," in Proc. IEEE Int. Parallel&DistributedProcessing,Workshops and Phd Forum (IPDPSW) Symp.pp.1-8, 2010, .
[7] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "QoE in pull based P2P-TV systems: Overlay topology design tradeoffs," in Proc. IEEE 10th Int. Conf. Peer-to-Peer Computing, pp 1-10 2010, .
[8] J. Ishmael, S. Bury, D. Pezaros, and N. Race, "Deploying rural community wireless mesh networks," IEEE Internet Comput., pp. 22–29, 2008.
[9] M. Wang and B. Li. Lava: A reality check of network coding in peer-to-peer live streaming. In ProcOf IEEE INFOCOM'07, pages 1082–1090, Anchorage, AK, May 2007.
[10] X. Chenguang, X. Yinlong, Z. Cheng, W. Ruizhe, and W. Qingshan. On network coding based multirate video streaming in directed networks. In Of IEEE International Conference April 2007.
[11] J. Zhao, F. Yang, Q. Zhang, Z. Zhang, and F. Zhang. Lion: Layered overlay multicast with network coding. IEEE Transactions on Multimedia, October 2006.
[12] K. Nguyen, T. Nguyen, and S. Cheung.Peer-to-peer streaming with hierarchical network coding.In Proc. of IEEE International Conference on Multimedia and Expo (ICME'07), pages 396–399, Beijing, China, July 2007.