Video streaming over p2p using AQCS algorithm

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

Abstract:-P2p streaming has been popular 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 un trusted 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 of the videos. where proposed design which enables flexible customization of video streams to support heterogeneous of 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, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls[1].This can be provided by WebSocket Protocol The capability to achieve high streaming rate is desirable for P2P streaming. Higher streaming rate allows the system to broadcast video with better quality. It also provides more cushions to absorb the bandwidth variations caused by peer churn and network congestions when constant-bit-rate (CBR)video is broadcasted[1]. The key to achieve high streaming rate is to better utilize peers' uploading bandwidth. In this section, we propose a queue-based chunk scheduling algorithm that can achieve close to 100% peers' uploading bandwidth utilization in practical P2P networking environment[3].In P2P system, the resource utilization is determinedby the overlay topology and collective behavior of chunk scheduling at individual peers. At system level, queue-based adaptive chunk scheduling requires fully connected mesh among participating peers. At peer level, data chunks are pulled/pushed from server to peers, cached at peers' queue,and relayed from peers to its neighbors[4]. The availability of upload capacity is inferred from the queue status such as thequeue size or if the queue is empty. Signals are passed between peers and server to convey the information if a peer's upload capacity is available.

.

II. ADAPTIVE QUEUE-BASED CHUNK SCHEDULING

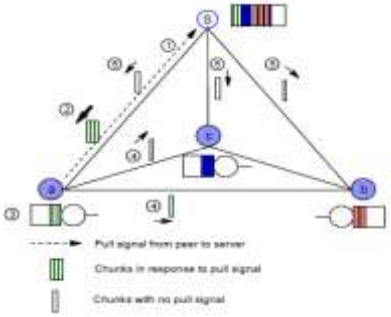

Fig. 1.Queue-based P2P system with four nodes.

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). These 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

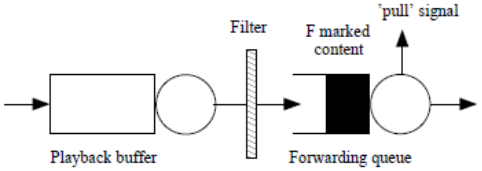

Fig. 2. Queue Model of Peers

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[2]*.

B. Server side scheduling algorithm and its queuing model

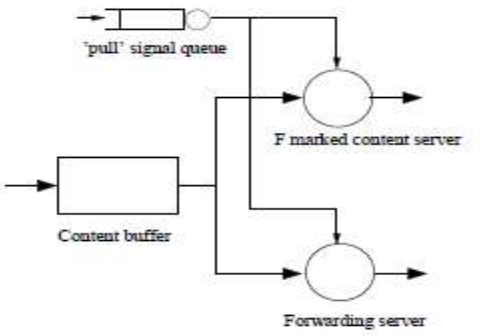

Fig. 3. Queue Model of Source Server

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' signalqueue. 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[1].

C. Proof of optimality for queue-based chunk scheduling
It show that the queue-based scheduling method for both the peer-side and the server-side achieves the maximum P2P live streaming rate of the system. Given a content source server and a set of peers with known upload capacities, the maximum streaming rate, rmax, is governed by the following formula The first case is termed as *server resource poor scenario* where the server's upload capacity is the bottleneck. The second case is termed as *server resource rich scenario* where the peers' average upload capacity is the bottleneck.Assume that the signal propagation delay between a peer and the server is negligible and the data content can be transmitted at an arbitrary small amount, then the queue-based decentralized scheduling algorithm as described above achieves the maximum streaming rate possible in the system. *Proof*: Suppose the video content is divided into small chunks. The server sends out *one* chunk each time it serves a 'pull' signal. A peer issues a pull signal to the server whenever the forwarding queue becomes empty. _ denotes the chunk size.For peer i, i = 1, 2, . . . , n, it takes time of (n − 1)_/uito forward one data chunk to all peers. Let ribe the maximum rate at which the 'pull' signal is issued from peer i. Hence ri= ui/(n − 1)The maximum aggregated rate of 'pull' signal received atServer, It takes server _/us toserve a pull signal. Hence the maximum 'pull' signal rate a server can accommodate is us/_. Now consider the following two scenarios/cases:

In this scenario, the server cannot handle the 'pull' signal at maximum rate. The signal queue at the server side is hence never empty and the entire server bandwidth is used to transmit F-marked content to peers. In contrast, a peer's forward queuebecomes idle while waiting for the new data content from the source server. Since each peer has sufficient upload bandwidth to relay the F-marked content (received from the server) to all other peers, the peers receive content sent out by the server at the maximum rate. Hence the streaming rate is consistent with the Equation (1) and the maximum streaming rate is reached.In this scenario, the server has the upload capacity to service the 'pull' signals at the maximum rate. During the time period when the 'pull' signal queue is empty, the server transmits duplicate NF`-marked content to all peers. The server's upload bandwidth used to serve NF-marked content is therefore For each individual peers, the scenario in which the server is resource rich described above. Again, the streaming rate reaches the upper bound as indicated in Equation (1). This concludes the proof. Note that in case 2 where the aggregate 'pull' signal arrival rate is smaller than the server's service rate, it is assumed that the peers receive F-marked content immediately after issuing the 'pull' signal. The above assumption is true only if the 'pull' signal does not encounter any queuing delay and can be serviced immediately by the content source server. This means that (i) no two 'pull' signals arrive at the exact same time and (ii) a 'pull' signal can be serviced before the arrival of next incoming 'pull' signal. Assumption (i) is commonly used in queuing theory and is reasonable since a P2P system is a distributed system with respect to peers generating 'pull' signals. The probability that two 'pull' signals arrive at exactly the same time is low. [3]

II. 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. 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 WebSocket 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 WebSocket Protocol attempts to Theaddress the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructurededicated 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].

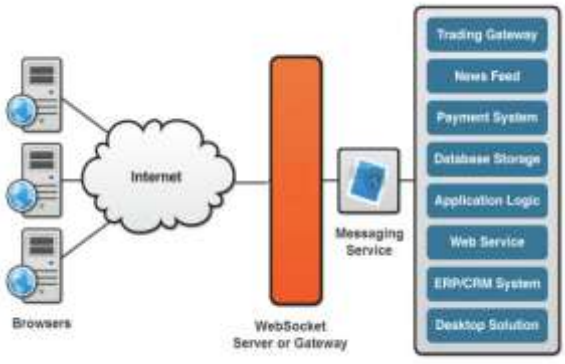

Fig1.Architecure of websocket protocol server.

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

In compliance with, header fields in the handshake may be sent by the client in any order, so the order in which different header fields are received is not significant. The "Request-URI" of the GET method is used to identify the endpoint of the WebSocketconnection, both to allow multiple domains to be served from one IP address and to allow multiple WebSocketendpoints to be served by a single server. The client includes the hostname in the |Host| header field of its handshake as per , so that both the client and the servercan verify that they agree on which host is in use.TheWebSocket Protocol in December 2011 TheAdditional header fields are used to select options into the WebSocketProtocol. Typical options available in this version are the subprotocol selector (|Sec-WebSocket-Protocol|), list of extensions support by the client (|Sec-WebSocket-Extensions|), |Origin| header field, etc. The |Sec-WebSocket-Protocol| request-header field it canbeused to indicate what a subprotocols (application-level protocols

layered over the WebSocket Protocol) are acceptable to the client.The server selects one or none of the acceptable protocols and echoes that value in its handshake to indicate that it has selected that protocol. Sec-WebSocket-Protocol: chat The |Origin| header field is used to protect againstunauthorized cross-origin use of a WebSocket server by scripts using the WebSocket API in a web browser. The server is informed of the script origin generating the WebSocket connection request. If theserver does not wish to accept connections from this origin, it can choose to reject the connection by sending an appropriate HTTP error code. This header field is sent by browser clients; for non-browserclients, this header field may be sent if it makes sense in the context of those clients.Finally, the server has to prove to the client that it received the client's WebSocket handshake, so that the server doesn't accept connections that are not WebSocket connections. This prevents an attacker from tricking a WebSocket server by sending it carefullycrafted packets using XMLHttpRequestor a form submission.Toprove that the handshake was received, the server has to take two pieces of information and combine them to form a response. The first piece of information comes from the |Sec-WebSocket-Key| header field in the client handshake:

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==For this header field, the server has to take the value (as present in the header field, e.g., the base64-encoded version minus any leading and trailing whitespace) and concatenate this with theGlobally Unique Identifier "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" in string form, which is unlikely to be used bynetwork endpoints that do not understand the WebSocket Protocol. A SHA-1 hash (160 bits), base64-encoded, of this concatenation 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 to be 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 TrackThe WebSocket 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 WebSocket Protocol uses the origin model used by web browsers to restrict which web pages can contact a WebSocket server when the WebSocket Protocol is used from a web page. Naturally, when theWebSocket Protocol is used by a dedicated client directly (i.e., not from a web page 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 WebSocket Protocol December 2011desired. This is achieved by having a strict and elaborate handshakeand 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 WebSocket server,for example, as might happen if an HTML "form" were submitted to a WebSocket server. This is primarily achieved by requiring that the server prove that it read the handshake, which it can only do if thehandshake contains the appropriate parts, which can only be sent by a WebSocket 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 suchasXMLHttpRequest [XMLHttpRequest].

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.

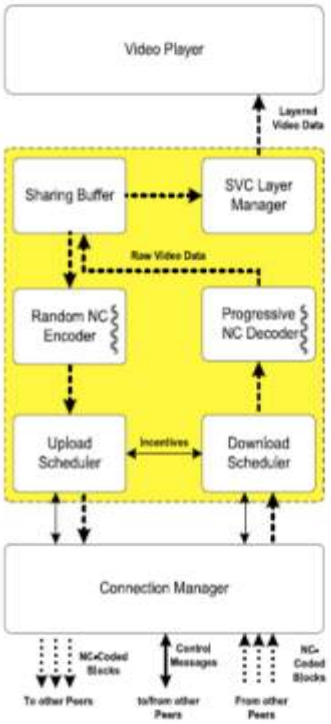

Fig 2: Peer Software Architecture. Dashed arrows denote video data, and solid arrows denote control messages

III. MEASUREMENT AND ANALYSIS OF PROJECT

working within an intranet boundary, since you likely have cont It's easier to communicate via TCP sockets when you're rol 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 tcp the 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 messageA 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. The over-all cost of building and maintaining this type of network is comparatively very less. In AQCS, the chunks size is set to be 1KByte, andthe server replies each pull signal with only one chunk. Weexperiment with other parameters and the current setting givesus the best performance. The server increases the streamingrate by increasing the number of chunks generated per second. If download window is set to be 30 secondsand moves forward every 10 seconds. The server producesfour chunks per second and increases the streaming rateby increasing the chunk size (this way the buffer map size remains the same) . A fullmesh is formed among nodes. The content source server'suploading capacity is set at 1 Mbps. We gradually increasethe streaming rate from 480 kbps to 960 kbps. For each streaming rate, we conduct one set of experimentsfor each scheduling algorithm. Each experiment lasts for 300seconds. Based on our off-line data analysis, the experimentduration of 300 seconds is appropriate since the systemsgoes into the steady state within tens of seconds.This indicates that when the system has high resource index,the chunk miss ratio is insensitive to scheduling algorithms.Even random scheduling can have very good performance.Indeed, most current commercial P2P streaming systems onthe Internet operate at streaming rates of 400kbps or lower.3) the previous experiment, the server uploading bandwidth is1Mbps. Since the average peer uploading capacity in thesystem is slightly above 1Mbps, it corresponds to the serverresource poor scenario. In the following experiments, weincrease the server bandwidth to 3:2Mbps, leading to the server resource rich scenario. when streaming rate goes beyond 960kbps while AQCS still maintains zero miss ratio up to 1*;* 100kbps.

B. Impact of Server Scheduling Rule and Capacity

The server gives strict priority to push out fresh chunks so that new content can be quickly distributed among peers. The experiment results suggest that (i) the freshchunkfirst scheduling at source server plays an important role in improving the system performance; and (ii) most deprived scheduling, although has been theoretically shown to be optimal does not seem to bring performance improvement in our experiments. Next we investigate the impact of server bandwidth. In the previous experiment, the server uploading bandwidth is 1Mbps. Since the average peer uploading capacity in the system is slightly above 1Mbps, it corresponds to the server resource poor scenario.when streaming rate goes beyond 960kbps while AQCS still maintains zero miss ratio up to 1*;* 100kbps. The miss ratio for AQCS increases linearly at the end, since the system resource index drops below one when the streaming rate is larger than 1*;* 100kbps. More interestingly, increasing server capacity from 1Mbps to 3:2Mbps extends the scheduling insensitive region dramatically from 480kbps (½ = 2:2) in Figure 5(a) to 960kbps (½ = 1:155) in Figure 6(a). Obviously, increasing server capacity can increase the resource index of the system. Andsystem performance will improve as resource index increases. However, according to equation 2)in that increasingserver uploading capacity brings in more dramatic performance improvement than barely increasing system resource index. illustrates the average server bandwidth utilizationunder different streaming rates. This suggests that the server bandwidth plays an important role in reducing the chunk miss ratio for these two scheduling algorithms. On the contrary, the server utilization in AQCS is low until the streaming rateexceeds 1Mbps and the resource index falls below 1. bring down the server load and improve the system scalability.conducted additional experiments by continuously varying server upload capacity. We fix the streaming rate at 640kbps and increase the server bandwidth from 600kbps to 1:2Mbps. When the server bandwidth is low, there are performance gapsamong them. Again, algorithms with fresh-chunk-first rule have better performance. As the server bandwidth approaches 1:2Mbps (twice the streaming rate), the performance becomes insensitive to scheduling and all of them have nearly zerochunk miss ratios. This again shows the unique impact of the server bandwidth on the whole system. One explanation is that a server with high bandwidth can simultaneously upload a chunk to many peers.r. The same amount of bandwidth increase on a regular peer does not have such significant impact. Our results here suggest that *investing onserver bandwidth can dramatically bootstrap theperformance*

D. Impact of Buffering Delay

In client-server based video streaming, client side video buffering is necessary for continuous playback in face of network bandwidth variations. In P2P streaming, each peer maintains a moving window that specifies the range of video chunks to be downloaded. The window normally advances

at the video playback rate. The window size determines the length of playback delay. The larger window size gives peers more time to download chunks. However, the larger window size imposes longer playback delay. The dimensioning of buffering delay is indeed a trade-off between the streaming delay performance and playback continuity In AQCS,the download window is 15 seconds and moves forward every 1 second. downloading of all missing chunks entering into the moving window. The peer stays idle to wait for the next window advance. The fast downloading at low streaming rate enables us to reduce the download window size to achieve shorter playback delays on all peers. AQCS at the streaming rate of 1120kbps.Where AQCS not only achieves zero chunk miss ratio, Next we examine the root cause of the delay difference in different scheduling algorithms. In P2P streaming, a peer downloads chunks either from the server or from other peers. Therefore, the variability in the uploading rates and downloading rates on all peers collectively determine peer delay performance. To verify this, we conduct two sets of experiments at different streaming rates. For each set of experiments, we keep track of peers' average uploading and downloading rates every 10 seconds.

rates. The digit `1*0* and `2*0* in the legends represent results for the streaming rate 640kbps and 960kbps, respectively. we see that when the streaming rate is low, for scheduling algorithms, the peer's download AQCS has stable uploading at low and high streaming rates.To reduce chunk miss ratio under uploading and downloading rate oscillations, one can introduce a large buffering delay on peers. The streaming rate and the server bandwidth are fixed to be 640kbps and 1Mbps. We then vary the download window size of all peers from 10 seconds to 50 seconds.Therefore, the average number of hops that a chunkneeds to traverse to reach all peers decreases. Consequently, peers can download chunks faster and the chunk miss ratio becomes smaller. To verify the path length of each chunk, we append a hop counter to each chunk, which records how many hops a chunk has traversed. Upon receiving a chunk, each peer increments the chunk's hop counter and forward it to its descendants in the chunk delivery tree. the distribution of the average hop count of randomly sampled chunks received by all peers. When all peers connect to 6 neighbors, each chunk needs to traverse in average 6 hops from server. As peer degree increases to 14, the average path length drops down to below 4. Next we examine the impact of peering degree in heterogeneous random topology.

we set the number of neighbors of a peer proportional to its uploading bandwidth. the average chunk miss ratio is less than 5%, while the average miss ratio in the homogeneous case is 20%. This suggests that peers with high bandwidth should be assigned with moreneighbors in order to improve the whole system performance. Since the existence of super peers can dramatically improve performance, more considerations should be given to them during the system design. Other than achieving good streaming performance, such as small chunk missing ratio and low chunk delay, how to make P2P streaming systems robust against peer churn is another major design consideration. In the rest of that section, the impact of peering degree on the resilience of P2P streaming systems. In that work, all 100 peers join the system at the beginning and each peer has homogenous number of neighbors. After the system enters its steady state, if we create peer churn events by removing a certain percentage of peers from the system .we samplethe performance of the system after each batch peer removal without implementing any churn recovery mechanisms. In practice, a peer losing neighbors can obtain more neighbors through directory service such as tracker. After each batch peer removal, a random set of peers in the system are sampled to evaluate their playback performance residual peering degree of the sampled peers after different percentages of peers leave the system. When 10% peers leave, the average chunk miss ratio on the sampled peers is around 12% and the average number of connections of each peer drops to around.

While 40% peers leave, the chunk miss ratio increases to20% and the average residual peer number drops below .As larger percentage of peers leave, the remaining peers have less neighbors and the performance becomes worse. Large peering degree helps in improving system robustnessin the face of peer churn. The system with initial peer degree 18 has better resilience than others. Even when 50% peers leave, it can still has around 15%chunk loss, while the ratio of that with initial peer degree 10 is nearly twice as much.e.g. prefetching and longer tolerable delays, and challenges, e.g. less concurrent peers. We will extend our current prototype system to support P2P VoD service. Experimental study will be conducted to test the applicability of live streaming results to VoD, and obtain new results unique to P2P VoD designs.

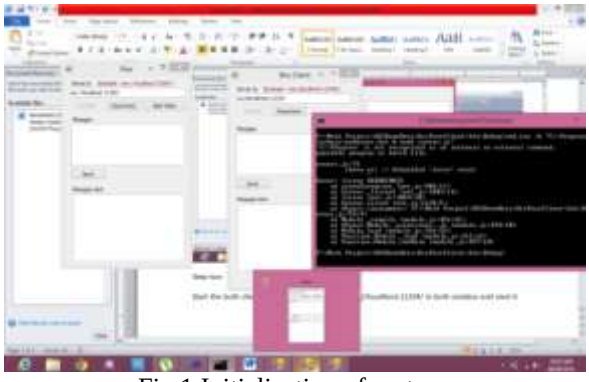

Fig 1:Initialization of system

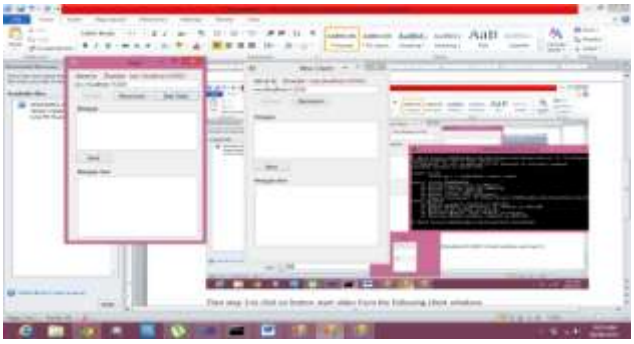

Fig 2:Connection of client towards peers

Fig 3:Connection of peer toward client

Fig :streaming video over p2p network

IV. 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.

V. REFERENCES

[1] Mu, Johnathan Ishmael, William Knowles, MarkRouncefield, 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.