Security Issues in BitTorrent like P2P Streaming Systems

Ratan Guha Darshan Purandare
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando 32816 USA
{guha, pdarshan}@cs.ucf.edu

Keywords: BitTorrent, DDoS, P2P, QoS, Share Ratio

Abstract

We present a streaming protocol that addresses the security issues pertaining to BitTorrent like P2P systems. Any efficient streaming system would not be effective if it cannot withstand security threats. We focus on free rider's, whitewasher's and malicious payload insertion attacks which implicitly affect Quality of Service like important metrics for the streaming system. We propose a novel idea of alliance formation, wherein the nodes collaborate with their preferred participating peers for a symbiotic association. The protocol enforces a strict policy that dissuades free riders and whitewashers from their selfish behavior. We propose a lightweight integrity checking algorithm for malicious garbled payload insertion to guard against security threats to the swarm. We present an evaluative comparison of the system performance in a cheating node free environment versus system with prevalent threats. Our simulation results show that cheating and malicious node problem can be significantly reduced by our proposed protocol. We explore the security vulnerabilities in the contemporary p2p file sharing systems and make security recommendations for the same.

1. INTRODUCTION

Media streaming of live events and Video on Demand (VoD) are an integral part of today’s entertainment world. Streaming of TV channels and live events such as sports are more common now. This places a high demand for an efficient media streaming mechanism to meet the desired Quality of Service (QoS) by an end user. The most important QoS metrics are minimal jitters, minimal latency from the streaming server and a robust and scalable streaming system.

Other factors which are equally important are system scalability, and fairness towards the end user. However, security has significant impact on these metrics. Media streaming is inherently more prone to attacks as it is very difficult to monitor the participating nodes in the swarm. A steady state may involve thousands of nodes, not all can be trusted. Security forms one of the most critical issues in any kind of system. All the above mentioned metrics are undermined if the system is vulnerable to security threats.

We focus on present day security threats in the p2p streaming systems. We enumerate some very important threats.

1. Malicious Payload insertion: Malicious data payload insertion is a security threat in p2p streaming environment. This is particularly a concern because of the large number of users in the swarm. Every node in the system is not trusted and it is quite likely that a bad node might garble or send malicious code and propagate further in the swarm. This problem gets aggravated exponentially as number of nodes in the swarm increase and leads to poor QoS.

2. Selfish Nodes/Free Riders: One of the very important factors of media streaming is a continuous and an uninterrupted service of streaming content. Selfish or cheating nodes may intentionally not forward the streaming content ahead in the swarm. This can cause slow decay of overall system performance if there are large numbers of such cheating nodes.

3. Whitewashers Problem: In a reputation based p2p system where a node is given credit for sharing uplink bandwidth, some users tend to change their identities. They enter in the swarm, they cheat i.e. they download the streaming content without contributing to the swarm. After a while their reputation becomes bad, so they leave the swarm and come again assuming a new identity. These cheating nodes resort to IP spoofing to hide from system administrators who consistently log the IP addresses.

4. DDoS attack: A DDOS attack is a common online assault that aims to overwhelm the network bandwidth and prevents access to the servers. Current streaming technology does little to prevent these common attacks. For example, on Bit torrent central servers, users were prevented from downloading upto 10 hours. This can bring down the entire network infrastructure.

In this paper, we address the above mentioned security issues by proposing a new p2p streaming protocol based on BitTorrent’s swarming technology. Section 2 deals with the related work. In section 3, we present the details of our protocol. Section 4 discusses the details of our simulation setup. We also perform an in depth mathematical analysis and overhead incurred in implementing security countermeasures. In section 5 we discuss the results of our experiments. Subsequently, we have conclusion and references.

2. RELATED WORK

Although there has been significant work towards improving efficiency in p2p streaming environment, there has been very little done towards addressing the security threats in such systems. To the best of our knowledge, very little literature has addressed the malicious and garbled payload insertion problem, though there have been attempts to address selfish nodes and whitewashers problem in the related literature. We briefly mention peer work done in the area of p2p streaming systems.

Several works have studied the p2p media streaming over IP multicast in the last decade. There are two main categories in such overlay multicast systems, which are *Tree based Protocols and Gossip based protocols.*

1. Many overlay streaming systems employ a tree structure, stemmed from IP multicast. Constructing and maintaining an efficient distribution tree among the overlay nodes is a key issue to these systems. Distributed algorithms, such as SpreadIt [1], NICE [2], and ZIGZAG [3] perform the constructing and routing functions across a series of nodes. For a large-scale network, these algorithms adopt hierarchical clustering to achieve minimized transmission delay (in terms of tree height) as well as bounded node workload (in terms of fanout degree). Still, an internal node in a tree has a higher load and its leave or crash often causes buffer underflow in a large population of descendants. Several tree repairing algorithms have been devised to accommodate node dynamics; yet the tree structure may still experience frequent breaks in the highly dynamic Internet environment. There are many other solutions addressing the unbalanced load or vulnerability of the tree structure. Examples include building mesh-based tree (Narada [4] and its extensions, and Bullet), maintaining multiple distribution trees (SplitStream [5] and leveraging layered coding (PALS) or multiple description coding (CoopNet [6]).

2. Gossip (or epidemic) algorithms have recently become popular solutions to multicast message dissemination in peer-to-peer systems. In a typical gossip algorithm, a node sends a newly generated message to a set of randomly selected nodes; these nodes do similarly in the next round, and so do other nodes until the message is spread to all. The random choice of gossip targets achieves resilience to random failures and enables decentralized operations. The use of gossip for streaming is not straightforward because its random push may cause significant redundancy, which is particularly severe for high-bandwidth streaming applications. In such a scenario, the video data provided by some seeding nodes are spread among nodes of asynchronous demands, and one or more nodes can collectively supply buffered data to a new demand, thus amplifying the system capacity with increasing suppliers over time.

In addition other important p2p streaming scheme called PROMISE [7], which relies on an application level P2P service called CollectCast [8] to select peers and dynamically adapt to network fluctuations and peer failures.

2.1 Contributions

The specific contributions of this paper are:

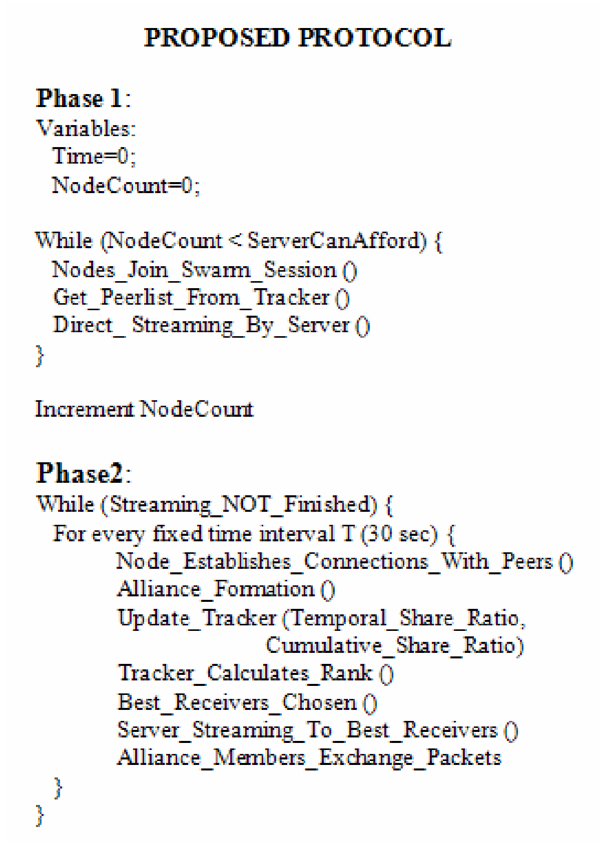
A novel p2p media streaming framework: We introduce a novel media streaming framework. It is a loosely coupled system, and does not enforce strict policy for choosing the peers. Peers collaborate together for a common cause that builds an incentive mechanism for the peers to contribute.
Theory of alliance formation: We introduce the notion of an alliance, wherein nodes preferentially group with peers who are of best utility to them.
Integrity check algorithm: We propose a lightweight integrity checking algorithm for malicious garbled payload insertion to guard against security threats to the swarm.

3. OUR PROPOSED SOLUTION

We propose a media streaming framework to create an efficient streaming system as well address the security issues prevalent in such environments. Our approach exploits end users uplink speed to load balance the bandwidth contention at the server. The media server relays stream to the preferred peers based on their utility to the swarm. These preferred nodes may vary over the streaming session.

The main entities involved in such p2p streaming environments are (i) Nodes (ii) Media relaying server and (iii) Tracker. Some users in the swarm receive streaming packets from media server. They, in turn, forward it to other peer nodes in the system. We enumerate the details of the protocol as follows.
As a new user comes into the swarm, it contacts the tracker. The tracker issues him a unique node ID, gathers its IP address and other details like time of arrival in the swarm. The tracker issues the node a peer list including the server address.

Our protocol is marked by different phases. During the phase I, when there are not enough number of peer nodes, the server streams directly to the nodes. As number of nodes in the swarm increase, the server cannot cater to everybody's request. This marks the beginning of the phase II. It eventually reaches a steady state that stays till the end of the streaming session. In this phase, a new node establishes connections with peer nodes for the stream content. For mutual benefit, peer nodes that interact more form a group called alliance. For the initial run, the alliance is formed based on peer list issued by the tracker. As the nodes continue to stay in the swarm they tend to use the best possible connection to form an alliance i.e. they use the nodes with maximal interaction to form an alliance. An alliance size can vary anywhere between 4 and 8. A node can be a member of more than one alliance. Having more than one alliance facilitates multiple paths for receiving the streaming content. Everybody in their alliance look for an in-order stream. They also buffer the stream contents for future use. Nodes cache some old streams so that if a new user comes into the swarm they can serve him with these streams.



```
PROPOSED PROTOCOL

Phase 1:
Variables:
  Time=0;
  NodeCount=0;

While (NodeCount < ServerCanAfford) {
  Nodes_Join_Swarm_Session ()
  Get_Peerlist_From_Tracker ()
  Direct_ Streaming_By_Server ()
}

Increment NodeCount

Phase2:
While (Streaming_NOT_Finished) {
  For every fixed time interval T (30 sec) {
        Node_Establishes_Connections_With_Peers ()
        Alliance_Formation ()
        Update_Tracker (Temporal_Share_Ratio,
                        Cumulative_Share_Ratio)
        Tracker_Calculates_Rank ()
        Best_Receivers_Chosen ()
        Server_Streaming_To_Best_Receivers ()
        Alliance_Members_Exchange_Packets
  }
}
```

Figure 1. Proposed Protocol

As a node receives a new streaming packet, it uses ANNOUNCE protocol to inform the peer nodes in its alliances. The peer nodes in turn propagate the stream packet to their respective alliances. This policy gives every node chance of building its own reputation among its alliance.

Every node has two variable parameters viz. temporal share ratio (TSR) and cumulative share ratio (CSR). Share ratio is defined as ratio of uploaded stream content to the downloaded stream content. Temporal share ratio is the share ratio of a node for a small time interval *t*, where t is much less than than total time elapsed. Cumulative share ratio is the share ratio of a node since its arrival in the swarm. The nodes, after every specific time interval, send these parameters to the tracker. The tracker calculates *Utility Factor*, a function of cumulative and temporal share ratios. UtilityFactor = UF = *f(TSR,CSR)*. For example, TSR is 75% and CSR 25%. These can be varied to have utility factors in several ways. If the emphasis is on recent contribution, then TSR is a better measure, else if total contribution is main factor while choosing candidates for direct streaming, CSR is a better measure. Further, it calculates rank based on node's utility factor. The nodes with higher rank are the candidates for receiving direct streaming from the server. Tracker informs the server about these ranked nodes after specific intervals of time.

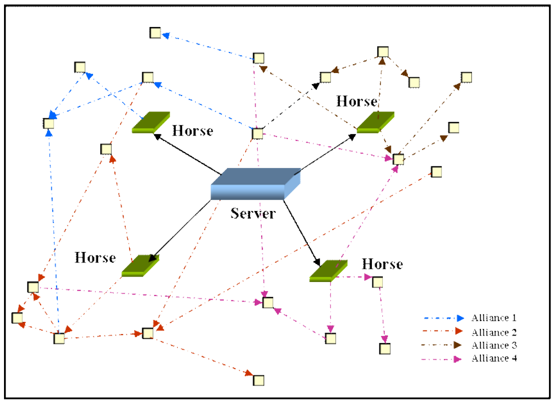


Figure 2. High Level Diagram depicting our Protocol. Server releases the stream packets to the horse nodes, they in turn forward to other peer nodes in the swarm. High ranked nodes become horses and are placed up in the hierarchy.

These ranked nodes by virtue of their utility to the swarm percolate up in the hierarchy and are given higher priority to get the streaming data directly from the server. These nodes have to consistently perform well in terms of forwarding stream content to others to remain "priority nodes" else they could be replaced by better performing nodes. This serves as a natural incentive for the nodes to contribute and be up in the hierarchy. This reduces its end to end latency and is guaranteed to receive direct streaming from the server.

3.1 Alliance

We formulate our alliance theory based on the best interaction (in terms of bytes exchanged) between the peers.

How Alliance works: Initially, when the nodes arrive in the swarm they make requests to the nodes in the peer list. After some time, nodes tend to interact more with certain peers. They keep an account of most recent profitable interactions with the peer nodes. The nodes then create an alliance by sending an alliance request packet to such nodes. The other node can accept the alliance or reject it (based on its own state which is based on how many alliances he is currently member of). If it accepts it send a packet back to the requesting node with a success message. These two members of the alliance can further expand the alliance as follows. For example, node 1 sends an alliance request packet to node 6. Node 6 accepts alliance invitation and return success message to node 1. Further, node 6 wants to form an alliance with say node 12. It will send a request packet to node 12. It will include the IDs of 1and 6 (himself). If 12 agree to join the alliance, 12 will send success packet to both 1 and 6. Now all three nodes 1,6 and 12 know that they are all part of an alliance. This is illustrated in figure 3. Similarly, any node can expand the alliance till it reaches the upper limit for the alliance (K in our case). If some new node requests for an alliance out of these nodes it would not include in the same alliance and would instead form a new alliance.

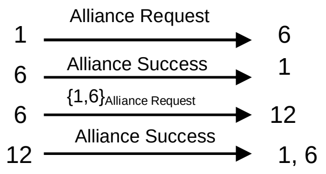


Figure 3. Alliance formation and interaction between nodes 1, 6 and 12. The nodes either accept or reject the alliance invitation. As more nodes join the alliance grows in size, it varies from 4 to 8.

Why Form Alliances*:* We show that alliance formation is a more organized node management methodology which facilitates real time stream transfer where several nodes contend for the same stream packet. Alliance formation has significant impact on the following metrics.

1. QoS: We define QoS as occurrence of minimal jitter during streaming and minimal end to end latency between server and the node itself. A node can be a member of different alliances. Nodes use an ANNOUNCE mechanism to publish their content in their respective alliance. The members of this alliance procure the new published content and further propagate in their respective alliances. This way the stream content is propagated in the swarm in minimal time. It ensures that a node will get the stream content in the minimal number of hops. In case of failure the node is guaranteed the stream content from the second best path. Using this mechanism we ensure that the end to end latency is kept to minimum. It maintains a good QoS in the system

2. Robustness and Reliability: A node is associated with more than one alliance. In case of node failure, the nodes in the alliance do not suffer as they have alternate channels of obtaining stream contents. The probability of all the nodes going down in an alliance is very less. It is assumed that the source server is up and active and there is no failure at the server end, in which case there are bound to be jitters and bad quality streaming at the user end. Nodes randomly join and leave the streaming session. Hence, even in case of node failure or departure our protocol is robust and reliable.

3. Scalable: There is no upper bound on the number of end users joining the swarm. If an efficient alliance mechanism is in place, it forms a strong mesh of nodes in the swarm. This increases everybody's chances of obtaining streaming contents in the best possible manner. As it can support infinitely many users, that share a part of their uplink bandwidth towards the swarm, it forms a scalable system.

4. Incentive Mechanism: A new node initially requests other nodes in the peer list for the streaming content. If a node is not engaged in alliance he has to constantly query the nodes for stream packets. By forming an alliance he becomes a regular member to receive the stream content. It also distributes the stream content to other alliances. This creates a trust with the members of his different alliances and this increases its chances of obtaining streaming contents. In nutshell, it is in every node's interest to be involved in the alliance, get served as well as serve others. Our protocol enforces co-operation among end users as well as rewards altruistic nodes. These high quality altruistic nodes receive the streaming content directly from the server. Other well performing nodes get in minimal hops. This serves as natural incentive for nodes to maintain high share ratio.

5. Fairness: Our protocol gives every node a chance to get served by the server directly. This is by virtue of their utility to the swarm (calculated by tracker). Irrespective of the arrival times, every node who serves the system with its uplink bandwidth gets an even chance to get the best QoS. The non-contributing nodes are warned before they are kicked out. The warned users who cooperate with the protocol are allowed to stay in the swarm in accordance with our self healing policy. Thus, our model is a self healing and self punishing one and turns the node behavior from selfish to altruistic enhancing the overall system performance. This way our proposed approach addresses the fairness front.

6. Security: If a malicious node intentionally forwards garbled data payload instead of a legitimate stream content, peer members of its alliance can immediately detect it using light weight integrity check algorithm described in the next sub section. Further, it informs the peers as well as the tracker about the presence of such malicious nodes in the swarm. This foils any intent to waste the peer bandwidth in the swarm; thereby preventing degradation of the overall system performance. Contributing nodes are guaranteed the streaming content with minimal latency and jitters, while the freeriders (selfish nodes) are automatically punished for non-cooperation. This scheme ensures fairness and dissuades whitewashers from assuming new identities multiple times.

3.2 Tracker and Server Functionality

3.2.1 Tracker

The tracker performs the following functions.

1. ***Assigning Peers***: As a new user contacts the tracker for streaming service, tracker obtains node's IP address and arrival time in the swarm. The tracker assigns the node a unique ID, a peer list, and the server address.
2. ***Rank Computation***: After every constant time interval, all nodes update the tracker with their temporal and cumulative share ratio. First, tracker calculates Utility Factor using the share ratios. It then ranks the nodes in decreasing order of their utility to the swarm. Finally, it informs the server about the highest ranked nodes.
3. ***Stream counter***: The tracker monitors the current stream counter that the server is relaying and updates new incoming nodes with this information.

3.2.2 Server

Media Relaying server coordinates with tracker, and they communicate with each other. Media server informs the tracker about the current streaming counter number. The server helps the new incoming nodes to bootstrap themselves and relays the content to the most promising nodes.

3.3 Security issues: Our Solution

We explain how alliance theory helps system deal with the security threats as mentioned above. We enumerate possible security attacks in the p2p streaming environments and how our proposed protocol addresses them. We propose the following measures against the above mentioned attacks.

1. Selfish Nodes/Free Riders: One of the very important factors of media streaming is continuous and uninterrupted service of streaming content. Selfish or cheating nodes can intentionally not forward the streaming content ahead in the swarm. This can cause slow decay of overall system performance if there are large numbers of such cheating nodes. The utility factor for such nodes goes down by their behavior and they become automatic choice for being kicked out of the alliance. It's not in their self interest to exhibit such behavior. Thus, our media streaming protocol takes care of free rider nodes.

2. Whitewashers Problem: Once kicked out of the swarm, every node has a steep curve to attain alliance and the minimum necessary steps. Therefore, whitewasher nodes do not gain any advantage by assuming new identity. If they perform well with new identity they will be part of the self healing model, if they don't they are victims of the self punishing model.

3. Malicious/Garbled Payload insertion: Malicious data payload insertion is a security threat in p2p streaming environment. This is particularly a concern because of the large number of users in the swarm. Every node in the system is not trusted and it is quite likely that a bad node might garble or send malicious code and propagate further in the swarm. This problem gets aggravated exponentially as number of nodes in the swarm increase.

Light Weight Integrity Checking Algorithm

A malicious node can propagate garbled content to peers, and the flooding effect of packets can spread such bad content very fast in the swarm. As a consequence, peer bandwidth is wasted, and QoS is affected. If such cheating continues to exist in the swarm, they can disrupt the entire streaming session. To detect such malicious nodes and eliminate them from the swarm, we propose a lightweight integrity checking algorithm that exploits the use of public key cryptography.

Algorithm 1. Lightweight Integrity Check Algorithm

```
1: Tracker issues Server's Public Key(KU) to Nodes
2: Encryption:
        Server encrypts CRC with Private Key (KR),
        appends to stream and relays
3: Decryption:
        Nodes upon packet arrival, decrypt CRC with KU
4: Nodes compute CRC of the arrived packet
5: if $CRC_{Decrypted} = CRC_{DataPayload}$ then
6:      Media playback stream
7:      Forward to peers
8: else
9:      Drop the packet
10:     Notify Peers and Tracker about bad packet
11: end if
```

Tracker issues server's public key to the newly arriving nodes in the swarm. Server, while relaying the stream content, encrypts the Cyclic Redundancy Check (CRC) with its private key, and appends it to the stream packet. Every node upon arrival of packet, decrypts the CRC and compares with the CRC calculated on the data payload. In case it doesn't match, the packet is dropped and the node signals the tracker and informs the peers in its alliances about the malicious node. Tracker blacklists the node and monitors its respective alliances. On receipt of future complaints from other alliances, i.e. if the node behavior persists, the tracker throws it out of the swarm; else the node can continue to participate in the swarm. Thus, the model exhibits self healing and self punishing policy for the non cooperating nodes.

Overhead Analysis
To encrypt the CRC-32 i.e. 32 bits with a RSA public key (128 bit strength), it requires (4/15.4) = 0.26 micro seconds (assuming standard RSA encryption rate using standard wincrypt [13] tool is 15.4 Mbps). To break a RSA key of 128 bits requires time in the order of several days using parallel computers. Thus, an encrypted CRC is safe in its meaningful time. For example, the time needed to perform the above computation is 0.26 micro seconds. Adding all other overheads while encrypting, we can safely say that in more 1000 hops, it will produce a latency of 1 second. Network topology of nodes is very critical in understanding the latency effect. Therefore, we can for all practical purposes, ignore this latency. The overhead of performing the above calculation is well worth saving the swarm from malicious nodes.

4. DDoS attack: This is the most important and difficult attack to detect. Possible prevention techniques can be using Secure Overlay Services (SoS) [9]. Previous approaches for protecting networks from DoS attacks are reactive in that they wait for an attack to be launched before taking appropriate measures to protect the network. This leaves the door open for other attacks that use more sophisticated methods to mask their traffic. Secure Overlay Services (SOS) is an architecture that proactively prevents DoS attacks, geared toward supporting Emergency Services or similar types of communication. The architecture is constructed using a combination of secure overlay tunneling, routing via consistent hashing, and filtering. It reduce the probability of successful attacks by (i) performing intensive filtering near protected network edges, pushing the attack point perimeter into the core of the network, where high-speed routers can handle the volume of attack traffic, and (ii) introducing randomness and anonymity into the architecture, making it difficult for an attacker to target nodes along the path to a specific SOS-protected destination. Such kind of approach if used to secure the streaming server could be really useful in preventing such kinds of attack.

4. SIMULATION SETUP

We model server, tracker functionality and nodes in the swarm. Server is the only source of streaming packet in the system. We present the details of our simulation setup for our media streaming protocol as well as for our model and compare the results with the case in a swarm where there is random file exchange and no alliances are formed. Our focus is to quantify QoS in terms of jitter factor, and latency, fairness (in terms of share ratio) and percentage of uplink utilization. We also quantify the number of selfish nodes that turned altruistic (healed) after being warned and the number of nodes that were forced out of the swarm (punished). We evaluate the above effect in achieving the total system performance.

We created a discrete event custom simulator written in Python Programming Language [11]. As mentioned in [10] network propagation delay is relevant only in the case of small sized packets. For experimental purposes, we do not include the delay factor. BT like p2p systems involve bulk file transfer downloads and ignoring network propagation delay doesn't impact the results. We did not model the TCP congestion and delays within the network. The bottlenecks are assumed to be either the uplink or downlink bandwidth and not anywhere else in the network. We believe bulk file transfers lasts over a period of time and not modeling TCP dynamics for small intervals doesn't influence the results. Pouwelse [12] points out the real world torrent downloads do not necessarily follow any particular arrival pattern. The arrival pattern of nodes in the swarm is assumed to be under Poisson distribution which is closest to the real world compared to any other distribution.

In our simulation setup we have varied the following parameters: Number of users (N) from 128 to 8192, Stream size (S) from 256 MB to 2048 MB. Each stream packet is considered to be 128KB. Initial seed is considered to be powerful node capable of very good upload speed say 6 Mbps. The distribution of nodes among various bandwidth strata is uniform. The various bandwidth strata we have considered are (10000 Kbps, 5000Kbps), (8000 Kbps, 4000 Kbps), (3000 Kbps, 1000 Kbps), (1500 Kbps, 384 Kbps) and (784 Kbps, 128 Kbps), where the first member of the tuple is the max download speed and the second is the max upload speed.

The simulator has an upper bound for parameters like number of nodes in one alliance (defined as H in simulation), number of alliance a node should be member of (defined as K in simulation). We have varied these parameters for various set of experiments and attempted to find an optimal value for these parameters in terms of number of nodes in the swarm. The best candidates for direct streaming from server are calculated every 4 or 8 seconds based on their utility to the swarm.
The aim of the simulator is to test the scenario where there are malicious or cheating nodes in the system. The other very important part of the simulation was to test the scenario find average latency from the server, number of jitters occurred during the streaming session for every node. We also attempted to answer some questions like (i) How well the system performs in kicking such bad nodes? (ii) How much overall decay is caused due to these bad nodes to the overall swarm? (iii) To quantify how alliance theory has helped to reduce the bad node problem. (iv) How fair has the system been to all the nodes with respect to their share ratios to the swarm? (v) What is the upper bound on the number of nodes such streaming system can withstand

without degrading the steaming quality below a certain level (vi) For how many nodes the streaming session crashed? (vii) How many bad nodes after warning turned to altruism? (viii) How many nodes were actually thrown out the swarm after poor behavior?

5. RESULTS AND DISCUSSION

The complete results and discussion will be included in the full paper. We are carrying out exhaustive set of experiments to test our protocol under various scenarios. Initial results have shown us good results. Bad node can be efficiently eliminated from the swarm. Alliance theory is atleast 14.6% better than normal swarming approach.

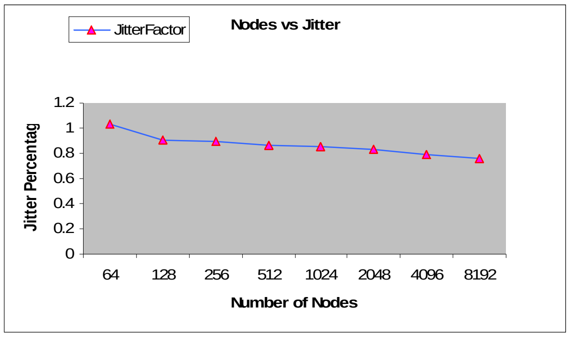


Figure 4: Nodes Vs Jitter

We tried to evaluate the jitter factor with respect to the total number of nodes in the system. It is evident that with the increase in the number of the nodes in the system the jitter factor decreases. The Alliance theory holds good as the number of the nodes increase. Thus the system is scalable.

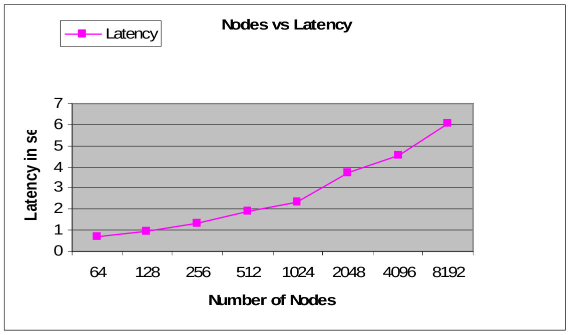


Figure 5: Nodes Vs Latency

Figure 5 evaluates the effect on Average System Latency when the number of the nodes in the system increases.

Figure 6 shows the node latencies when all the nodes are trusted and there are no malicious nodes in the system. The nodes with higher utility factors i.e.: The contributing nodes have low latency while the ones with lower utility to the swarm have higher latencies in accordance with the alliance theory.

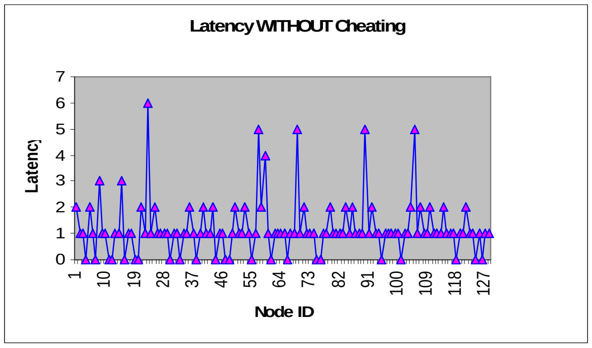


Figure 6: Latency without Cheating Nodes in the System

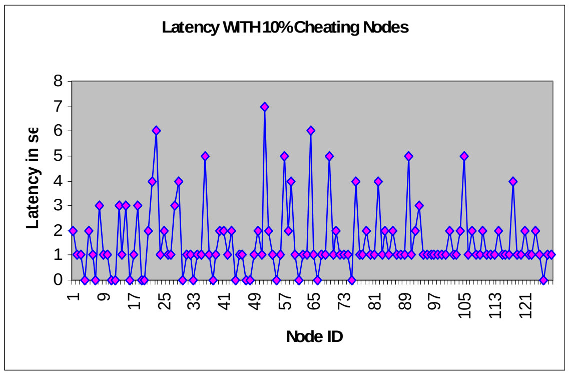


Figure 7: Latency with 10% Cheating Nodes in the System

For our experiment we injected 10% Cheating nodes in the system .The utility factor of the nodes is reduced .However the overall latency increases and the contributing nodes suffer on account of the cheating nodes. As every node is member of multiple alliances this effect is evident throughout the system and overall system performance gets affected.

Figure 8 shows an evaluative comparison of the system with and without cheating nodes. The effect of latency manifests in the contributing nodes as well degrading the system performance as latency is an important metric in P2P systems.

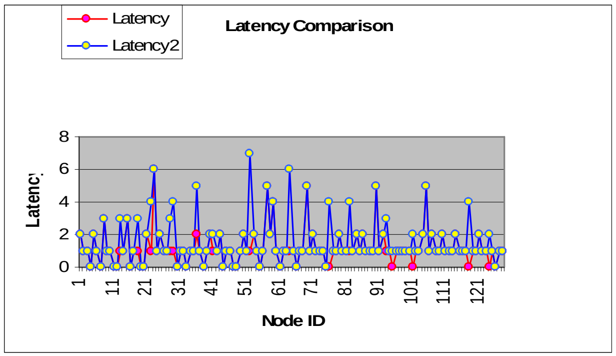


Figure 8: Comparison with and without cheating nodes

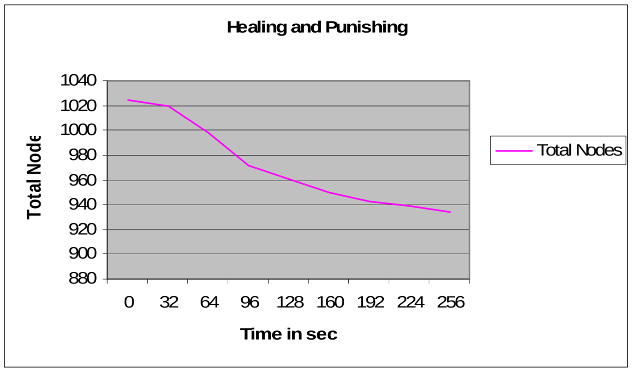


Figure 9: Self Healing and Self Punishing Model

According to the proposed protocol the cheating nodes are issued a warning when the peer nodes in its alliance report their behavior .The node behavior is monitored for the other alliances the particular node is a member of and in case if the behavior persists the node is thrown out of swarm. The figure shows reduced number of nodes in the system.11% nodes was found to be the victims of Self Punishing model whereas 9% of the nodes were healed in accordance with our Self Healing theory.

VI. CONCLUSION AND FUTURE WORK

The BitTorrent approach to p2p streaming is very efficient approach if used with alliance theory. It successfully addresses the free rider's problem and dissuades whitewasher's from assuming multiple identities. We propose a light weight check integrity algorithm to detect and eliminate malicious nodes from swarm. Our results successfully demonstrated the Self Healing and Self Punishing model behavior. We propose to use SoS kind of approach to save servers from unknown requests that can successfully prevent DDoS attack. Our custom simulator has shown how our proposed approach is an efficient proposition to solve the security related threats. The performance is promising and the results are inspiring. We plan to extend our current simulator to real world client by performing the experiments on Planet Lab.

REFERENCES

1. H. Deshpande, M. Bawa, and H. Garcia-Molina. "*Streaming live media over peer-to-peer network*". Technical report, Stanford University, 2001.
2. S. Banerjee, B. Bhattacharjee, C. Kommareddy, and G. Varghese. "*Scalable application layer multicast*". In Proc. of ACM SIGCOMM'02, pages 205–220, Pittsburgh, PA, USA, August 2002.
3. D. Tran, K. Hua, and T. Do. Zigzag: *An efficient peer-to-peer scheme for media streaming.* In Proc. of IEEE INFOCOM'03, San Francisco, CA, USA, April 2003.
4. Y. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. IEEE Journal on Selected Areas in Communications (JSAC), 20(8):1456–1471, October 2002.
5. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, Atul Singh, "SplitStream: HighBandwidth Multicast in Cooperative Environments ",In 19th ACM Symposium on Operating Systems Principles.
6. V. N. Padmanabhan and K. Sripanidkulchai. "*The Case for Cooperative Networking* ". Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA March 2002.
7. M. Hefeeda, A. Habib, B. Botev, D. Xu, D. B. Bhargava, PROMISE: Peer-to-Peer Media Streaming Using CollectCast, In Proc. of ACM Multimedia 2003, Berkeley, CA, November 2003.
8. Mohamed Hefeeda, Ahsan Habib, Dongyan Xu, Bharat Bhargava, Boyan Botev ", CollectCast: A Peer-to-Peer Service for Media Streaming", ACM/Springer Multimedia Systems Journal, October 2003.
9. Keromytis, A.D., Misra, V., Rubenstein, D.: SOS: Secure Overlay Services. In: Proceedings of ACM SIGCOMM (2002).
10. Ashwin Bharambe, Cormac Herley and Venkat Padmanabhan: "Analyzing and Improving a BitTorrent Network's Performance Mechanisms", Microsoft Research Technical Report Number MSR-TR-2005-03.
11. http://www.python.org/
12. Johan Pouwelse, "The BitTorrent P2P file-sharing system", http://www.theregister.co.uk/2004/12/18/bittorrent_measurements_analysis
13. http://www.wincrypt.com