

Paper:

I3P: A Protocol for Increasing Reliability and Responsiveness in Massively Multiplayer Games

Jeremy Kackley*, Matthew Gambrell**, and Jean Gourd*

*University Of Southern Mississippi, School of Computing, 118 College Dr., Box 5106, Hattiesburg, MS 39406

**Amaze Entertainment, 816 Congress Ave SE, Suite 600, Austin, TX 78701

E-mail: {jeremy.kackley, mgambrell, jean.gourd}@gmail.com

[Received June 1, 2007; accepted September 12, 2007]

Developing broadband and internet technologies offers possibilities for new ways of minimizing the server bottleneck in online gaming as well as an increase in response and reliability. We look at a peer-to-peer (P2P) approach to circumnavigate some of the reliance on the central server and propose a protocol designed to increase responsiveness and reliability – which is also useful in meeting the unique requirements of a P2P approach.

Keywords: I3P, Massively Multiplayer Online Gaming (MMOG), peer-to-peer (P2P), protocol, transport

1. Introduction

Collectively, Massively Multiplayer Online Games (MMOG) control a significant portion of the gaming market [1]. Indeed, they are the latest and greatest thing as is evidenced by the recent popularity of games such as *World of Warcraft* [2]. Major gaming companies have decided to devote significant resources and development time in order to create, maintain, and support old and new MMOGs. There are, however, several issues that they all share to a greater or lesser degree. Generally, MMOGs feature lower reliability and responsiveness than other gaming mediums, which is partially intrinsic to their nature. This is not to say that it is intolerable; merely that they are not known for extremely high responsiveness. This situation is not one that is desirable to most players, although it is readily tolerated. Moreover, there is a limit of several thousand players per server, although this varies from design to design, and reliable numbers are difficult to attain.

One can draw several conclusions from these facts. We will revisit the first two issues later, but first we must address the issue of player limits. One can reasonably construe that the server is the bottleneck, as there is a limit to the amount of information a single machine can handle and transmit. Although this increases over time as machines get faster, there are ways to get around this – namely parallel and distributed models where many lesser machines take the place of one major machine. P2P models also achieve a similar goal as a group of mediocre ma-

chines collectively perform a task far beyond their individual capabilities [3].

To revisit the first issue, that of responsiveness and reliability, the more information that is transmitted, the more likely it is for some of that information to be dropped. Also, the server acts as a single point of failure in the entire system. Additionally, all the clients may not be able to attain a good connection to the server, if for no other reason than geographic location. Disregarding internet lag, the server itself may be slow to respond simply due to being overloaded with too many tasks.

There is a single solution that has the potential to address all of these issues: P2P networking. P2P based networks avoid a single point of failure and scale extremely well [4]; the more peers involved, the more likely it is that a good connection can be made. The less the server is asked to perform, the more users can be supported in a single “world.” In section 2, we discuss this in more detail.

This paper is structured as follows: we discuss how P2P networking will aid in reducing server needs in section 2 and present I3P, our proposed protocol, in section 3 including preliminary cost analysis. In section 4, we discuss the crucial process of signal selection. We briefly discuss other works in section 5, present results in section 6 and conclude and discuss future directions in section 7.

2. Method

The development of consumer broadband and Internet technologies will provide new opportunities for the utilization of P2P approaches in multi-player games. Effective NAT traversal and eventually IPv6 deployment will help ensure peer connectivity, and increasing bandwidth will more readily support more connections than just the one to a server. P2P approaches can result in less server load and lower latencies, so taking advantage of these developments will be a desirable challenge to undertake. Additionally, these techniques may ultimately lead to more players existing on a single server.

One method of applying a P2P approach to a MMOG is to observe that you can partition parts of the game into *chunks* that are managed by sub-servers. This partitioning can be performed in a variety of ways. One partitioning



scheme would be to partition the world into physical sections, while another would be to partition the player base into logical groups according to some criteria such as locality to each other or a measure of the likelihood of their interacting [3].

Once partitioned into tasks performable by sub-servers, the next step is to transfer as many of the tasks the sub-servers were performing onto the player base as possible. In order to do this, these tasks must be identified. First, the sub-servers must authenticate and validate each player. In addition, they must communicate perceptive data to the clients and validate any received messages. Lastly, the sub-servers must remember the state of each client and decide the effect of the interaction of clients in addition to any reaction the environment has to them.

One way to move some of these tasks onto clients would be to have a group of clients that continuously communicate their intentions to each other, and amongst themselves decide on a sequence of events. They may even decide the results of their interactions, particularly if they were all using the same seed. The major difficulty in this is how to have them effectively communicate with each other and decide on a sequence of events. It is very important that the sequence of events be the same for everyone. This is the question this paper attempts to address: how to reliably broadcast a player's intentions to a group of peers with whom s/he is communicating with.

In the ideal environment, which we consider here, bandwidth is plentiful and players crave high responsiveness and few network glitches. A protocol which ekes as much responsiveness and reliability out of the network as possible will prove helpful.

In keeping with the trend in this paper, it should be noted that in the future we will see increasing utilization of wireless network technologies and increasing demands placed upon them by users. Therefore, there will be an attendant stress on reliability through additional packet loss and latency glitches. Reliability will be crucial.

3. Protocol

Our solution is a protocol which strives for the aforementioned goals by recruiting each peer to forward messages to all the others. The key notion is for each peer to keep track of which messages all the others have received. This will enable the peer to preemptively update the others with whichever messages it believes them to be missing. Under some circumstances, the need for back-and-forth protocol negotiation will therefore be reduced, improving latency. If unneeded messages are received from a forwarding peer, they are simply discarded. A peer will take any opportunity to inform other peers of its understanding of the current situation.

Suppose there are two signals which peer *Z* can transmit:

1. Send message *M* from queue *X* to peer *Y*
2. Send "Z has received *X* up to *N*" to peer *Y*

The first signal merely conveys a sequence-numbered user message, which may either have originated on peer *Z* or on another peer (and is being forwarded by peer *Z*).

The second signal is a means of indicating which messages peer *Z* has received. After receiving this signal, peer *Y* would no longer need to preserve messages *N* or older from peer *X* on peer *Z*'s behalf. Once peer *Y* no longer needs to preserve a message from *X* for anyone else, it is safe to discard it.

These two signals are all the tools we need for the protocol. Crafting the optimal signal to send at any given point in time will be the challenge. Optimization is also necessary to meet our design goals, as the most unintelligent choices, such as sending the same signal repeatedly, will result in a broken protocol. A reasonable introductory algorithm for signal selection should at least prove that these signals are sufficient for building a functional protocol, however poorly it performs. We present such an algorithm in section 4.

We must stress that each of the signals defined here is tiny. Once the transport layer [5], assumed in this case to be UDP since much of the work of TCP will be reconstructed by our protocol, adds its overhead, the signal payload will be a distressingly small portion of the datagram. It is a relatively straightforward matter to ameliorate this by concatenating several signals into one datagram, thereby accepting the new constraint on the signal selection algorithm that the destination peer for several concatenated signals must match. The algorithm may decide to send a smaller datagram than it could due to a fervent desire to rush out a signal to another peer. This concatenation is expected to be fairly necessary in practice and is implicit in our algorithm.

Our algorithm combines several *update* signals into one larger concept: a matrix of sequence numbers which will be naively blasted over the network repeatedly. This has the benefit of avoiding complex logic for determining which *update* signals to send; we just always send all of them. We will also roll into the datagram an assortment of old *message* signals from various queues, which we believe that the current recipient peer needs, until we run out of room at the practical UDP datagram size limit of approximately 1,280 bytes [6, 7]. An implementation of this algorithm would not necessarily be coded in terms of the underlying two-signal protocol; rather, the underlying protocol is used as a logical backbone.

Our protocol creates an environment where broadcast messages can be rushed to each peer with all available resources. It would be the responsibility of a client of our protocol to temporally order messages. A simple approach would be for a peer to require approval from every other peer in the group before considering its proposed action to be confirmed. If one of the other peers had previously requested something contradictory for which confirmation was pending, it would not approve the action. Of course, in this situation neither peer will get the confirmation unless a priority is built into the actions or peers. If a peer receives a contradictory but higher-priority confirmation request, then it would first send a cancellation

of its lower-priority request, followed by confirmation of the higher priority request. Since our protocol guarantees in-order delivery within the context of a peer-pair channel, peers observing this chaos would find that the lower-priority request was withdrawn before the higher-priority request was confirmed.

As another example, during combat each peer could take complete responsibility for one of the opponents. The temporal ordering would be defined as the order in which messages arrive at the peer controlling the pertinent characters. This peer would broadcast acknowledgements or corrections to other peers.

In general, it is hoped that application-tuned distributed decision-making would minimize the need to temporally order messages, since the temporal ordering is only one approach to the arbitration of conflicting messages; merely the one that comes most readily to mind from a client/server point of view.

In getting started, as an example, suppose peer *A* enqueues the first message for broadcast to peers *B* and *C*. Peer *A* is maintaining knowledge of the peer group in the form of sequence numbers as follows:

$$A[B][A] = 0$$

which indicates to peer *A* that peer *B* believes that *A*'s latest sequence number is 0.

As a consequence, when this message A_1 , with sequence number 1, is enqueued, peer *A* knows that peer *B* is unaware of it. It is suitable, then to transmit this message to peer *B*. Along with that message will be the entire matrix of sequence numbers including:

$$A[A] = 1, 0, 0$$

indicating that peer *A* is aware that it knows that it has received message 1 (obviously), and:

$$A[B] = 0, 0, 0$$

indicating that peer *A* is aware that peer *B* knows that nobody has received any messages. Since *A* has not yet received any transmission from *B*, this is truth as *A* knows it. Peer *A* cannot assume that *B* will know about A_1 , since the packet may get lost along the way.

When peer *B* receives the transmission, it will receive with it the A_1 message and will therefore be able to update its own personal set of sequence numbers:

$$B[B] = 1, 0, 0$$

indicating that *B* is now aware of A_1 . Along with the transmission is the entire matrix of *A*'s sequence numbers, from which can be assimilated an obvious fact:

$$B[A] = 1, 0, 0$$

indicating that peer *B* is now aware that peer *A* is aware of A_1 .

Now, *B*, being a dutiful member of the peer group should think himself aware of something important:

$$B[C] = 0, 0, 0$$

which is interesting because *B* has A_1 laying around and

would like to upgrade *C*'s knowledge. In this way, *C* may receive A_1 even if the transmission directly from *A* to *C* failed; and it will happen in more or less two network hop intervals: $A \rightarrow B \rightarrow C$ instead of the three it would take, at best case, for *A* to recognize the failure and retransmit, and *B* to acknowledge. If *C* happened to have a transmission en route to *A* when the $A \rightarrow C$ transmission was lost, then *A* would receive:

$$C[A] = 0, 0, 0$$

and know that either *C* has not yet received its transmission of A_1 , or that it has been lost. Depending on the sense it may develop, in an advanced implementation, of the timing of communications between the two peers, it may choose to go ahead and try another transmission. Or, by the time that *B* receives the A_1 transmission, *A* may decide that it is time to transmit another message, A_2 . In that case, since *A* has:

$$A[A] = 2, 0, 0$$

and

$$A[C] = 0, 0, 0$$

it then possesses approximately two messages (A_1 and A_2) more than *C* has. Both messages will go out with the transmission, and so the lost message will be corrected without any error recovery protocol.

With this practice – in the most paranoid case – given a restricted view of the situation involving only the two peers, a peer could transmit the same message over and over as fast and often as possible until it receives an acknowledgment from the receiving peer; all this in an effort to ensure the speediest delivery to that peer. But since a peer knows that it needs to update all other peers anyway given the broadcast assumption in this protocol, it may as well transmit to them all and let them take care of forwarding the message to anyone who may be missing it.

In implementation, we may have many messages to choose from that we feel need transmission. An incredibly naive technique can be used to select messages, such as round robin selection from the oldest messages in each queue until the datagram is full, for example.

3.1. Efficiency

In analyzing bandwidth efficiency, we define an overhead percentage equal to:

$$\frac{(\text{TotalBytesSent} - \text{LogicalBytesBroadcast})}{\text{TotalBytesSent}}$$

An idealized broadcast which transmits each message byte would have an overhead percentage of 0%. A straightforward, idealized simulation of broadcasting by transmission of identical messages to each peer would yield:

$$\frac{M(N-1) - M}{M(N-1)} = 1 - \frac{1}{N-1}$$

where M is the size of the message and N is the number of peers in the network.

These efficiency values asymptotically approach 100% as overhead increases. Initial investigation into the bandwidth efficiency of this protocol reveals overhead percentages as low as 85% for typical cases involving 5-10 peers and message sizes from 40-200 bytes. Overhead can be higher for smaller messages but can never be lower than the corresponding levels for the straightforward broadcast simulation.

4. Signal Selection

Since the promise of this protocol relies on a better algorithm for signal selection, it behooves us to discuss briefly what may be involved in this.

Suppose, in addition to the queues, we track information such as when a message was added to a queue, when it was last sent in a *data* signal to each peer, how rapidly each kind of signal is arriving from each peer, etc. Such information we could consider our state, and each state can be scored according to the desirability of what is existing in that state. For example, it is perhaps far more desirable to work on emptying a very stale queue by sending a *data* signal for it than it is to send an *update* signal for a fresh and relatively unoccupied queue. Our best decision for the next transmission would be to undertake the transition to the most valuable state. A more sophisticated algorithm, then, would define a scoring function with tunable parameters. The following subsections discuss some principles to take into consideration.

4.1. Recovery

We may want to try very hard to recover peers that have fallen far behind. This parameter will be determined by the cost to the application of having its decision-making logic ruined by latency.

Due to this factor, older messages have a higher priority; we may decide to compensate by giving brand new messages a priority boost since we may not want to penalize faster peers while the network attempts to recover from the lagging peer.

The falloff for the newness factor would be modulated by our opinion of the quality of the network. If we can assume that datagrams generally make it on the first try, then their bonus for being new will wear off the first time a datagram is sent.

4.2. Popularity

This dampens a score by selecting messages that are likely to be sent frequently. Since each peer has some idea of what each other peer has, then it will have some information about how another peer is doing its scoring. If another peer is likely to score a message high, then we should score it lower.

4.3. Responsiveness

1. Each peer should prioritize sending a message to another that has just sent it something, so that it knows not to send it again.
2. Each peer should extra-prioritize sending a packet to another that has just sent it something that it did not need – since the other peer thinks it is old and that we need it badly, it is likely to keep scoring it higher.

4.4. Time Effects

We can determine whether it is optimal to concatenate signals by considering whether it is possible to achieve a higher score by saving the time cost of transmission overhead. This is not overly difficult. Let us define time in terms of bytes (knowing that it is readily convertible to actual time by factoring in a bandwidth estimation or requirement):

$$C(S(t)) = \text{value of a state at time } t.$$

So, we pick one:

$$C(S_1(t)) + C(S_2(t + O_H + N))$$

where the constant O_H represents the overhead of sending a new message. This is the value of the state after action 1 plus the value of the optimal state after action 2, $O_H + N$ time units later where the signal associated with S_1 takes N bytes to transmit. Now, consider the following cost:

$$C(S_1(t)) + C(S_3(t + n)).$$

Here, S_3 is constrained to a subset of states defined as those reachable by sending a signal to the same peer as involved in S_1

Of course, since the optimal value for our next state is now dependent on the optimal value for the state after that, we descend into a recursive definition. This has the benefit of setting up something for spare CPU cores to do while waiting for a datagram to transmit, even if the value ends up being slight.

Deciding the ordering of messages to send is indeed a problem. The determination of messages to concatenate can be subsumed into the question of ordering by setting up our ordering methodology carefully.

If one were to concatenate, one would be able to send a new message without incurring the overhead. But to concatenate, a second message must be sent to the first peer. If a third message, destined for another peer, was more crucial than the candidate for concatenation, then one would accept the overhead cost and simply go ahead and send the message to the other peer.

We can solve this problem by finding the maximum value among all possible states. The states are generated by taking all permutations of pending messages. Each permutation has a corresponding vector of times corresponding to the delay before each message within that permutation would be sent. The times vector can be generated as follows:

```
function generate_times(vector V of messages)
  time T = 0
  vector R of result times

  for each message M in V
    T.add( R )
    if M is first or
      M.prev.recipient != M.recipient
      T += OH
    T += M.size
  return R
```

Now here is an example of a scoring function:

```
function score_state(vector T of times,
                    vector V of messages)
  if vector V is too long, return zero
  time TIME = 0
  var C = 0
  for each index i in the state vectors
    C += (V.priority + V.age) / TIME
    TIME += Ti
  return C
```

This function will prefer states with high priority and aged messages scheduled earlier.

Once the highest-scored state has been determined, the next datagram to be sent is the concatenation of the first message in the state and any immediately subsequent messages destined for the same peer. Of course due to the embracing of all permutations this has a terrible time complexity. Depending on the `score_state` function, however, this problem may be amenable to dynamic programming.

5. Other Works

Chen et al. in [8] conducted a study with the help of a commercial MMOG in which they concluded that TCP was poorly suited for the needs of MMOGs. In brief, the reasons they report are small packet sizes leading to wasted bandwidth due to large headers, unnecessary in-order delivery of packets leading to avoidable lag, an ineffectual congestion-control mechanism since game traffic is network limited – not application limited, and ineffectiveness of TCP's fast-retransmit algorithm leading to increased latency.

Our algorithm suffers from the large-header size, but our broadcast scheme avoids retransmit delays and we do not ensure in-order delivery. In-order delivery is only an issue because it is often not important in MMOGs, and if a client is waiting for a packet unnecessarily, then this leads to completely unnecessary lag. I3P proposes a solution to the large-header size vs. payload size issue by concatenating several smaller messages into one larger message. Interestingly enough, in spite of the poor-suitedness of TCP for MMOGs, the most successful MMOG to date, *World of Warcraft* [2], is reported to be using TCP as its protocol [8].

Mauve et al. propose Real-Time Application Level Support for Distributed Interactive Media (RPT/I) [9], which has been put forth as a potential standard protocol for interactive distributed applications. While this certainly would seem applicable to MMO gaming situations, their focus seems to be on shared media; their cited target examples are shared whiteboards and distributed Java

animations.

Pack et al. propose a Game Transport Protocol (GTP) in [10]. GTP possesses several features geared towards making it suitable for MMOGs such as a packet-based window scheme, an adaptive retransmission scheme, and multiple modes of support. The packet-based window scheme avoids the excessive bandwidth overhead associated with TCP's byte-based window scheme in the case of small game packets. Adaptive retransmit chooses not to retransmit some packets based on the fact that many packets in MMOGs become irrelevant very quickly. By multiple modes they allow the user to choose what the priority should be which can vary depending on the situation (e.g., on-time delivery being more important in a first-person shooter than in a chess game). Our protocol is primarily concerned with reliably and quickly transmitting data, to the exclusion of all concerns about bandwidth, and we achieve this through broadcast. The goal of GTP seems to be efficiency, while our protocol is focused on minimizing delay, and we plan to impose efficiency via somewhat higher level signal selection algorithms.

With regards to UDP, it is typically the protocol of choice for fast paced very active games [8]; while it is lightweight compared to TCP, it also lacks functionality permitting reliable transmission and session management [10]. I3P addresses UDP's shortcomings with respect to reliable transmission by ensuring that every peer involved gets its intended messages.

6. Results

We have developed a prototype implementation and simulation of the reference protocol which was useful in investigating its performance characteristics. Several parameters were included and proved useful in controlling the operation: A `message_count` defines the maximum number of messages which will be sent in each packet. Too many messages can result in some peers being neglected; it would probably be better to send fewer, older messages for all peers than to send a load of messages for a single peer. Without a constraint on the size of the packet, the protocol will collapse in inefficiency as the number of redundant messages increases. The packet size is also, as a practical matter, constrained by `udp_limit`, a limit on the size of a UDP datagram which is supposed to be safely capable of intact transmission. The prototype truncates packets at `udp_limit`. Truncated packets can have several messages recovered from them; truncated messages are discarded.

The simulation creates `peer_count` nodes, each of which broadcasts a message of size `message_size` to all other peers at each iteration of the simulation. A ratio `packet_loss` describes a proportion of packets which are lost by the network. Packet loss will generally hurt latency by necessitating more frequent retransmissions. With this protocol, less time is required to recover from lost packets since each peer is constantly retransmitting old information. Moreover, transient glitches

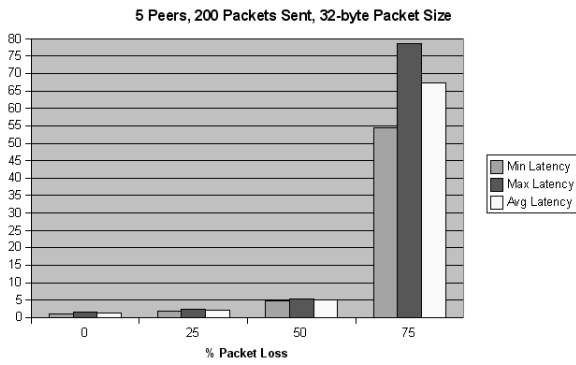


Fig. 1. Latency (ms) for 5 peers, 200 packets.

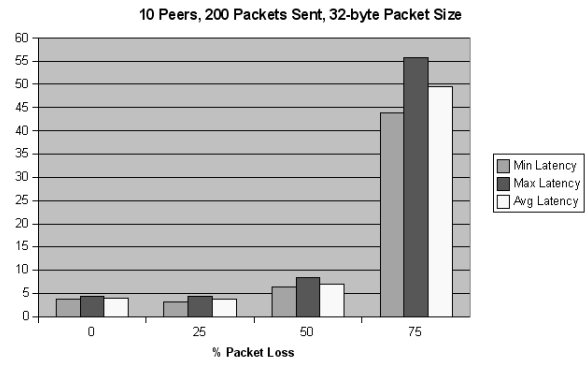


Fig. 4. Latency (ms) for 10 peers, 200 packets.

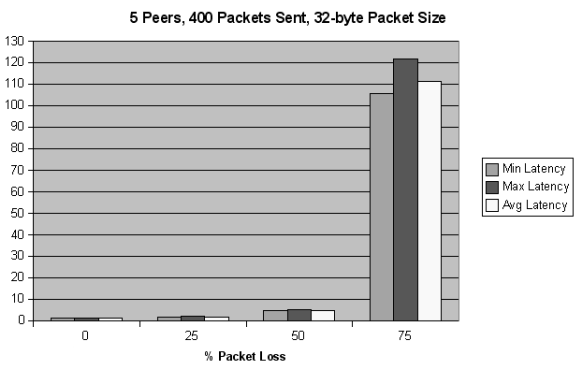


Fig. 2. Latency (ms) for 5 peers, 400 packets.

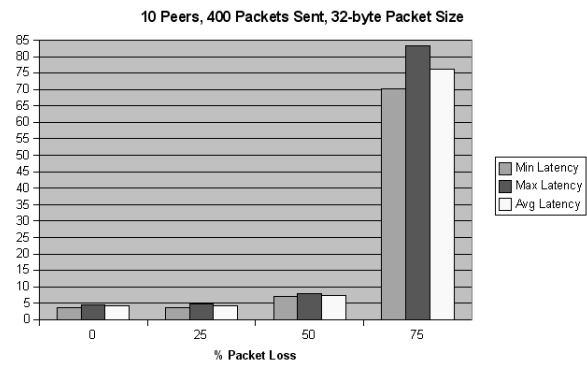


Fig. 5. Latency (ms) for 10 peers, 400 packets.

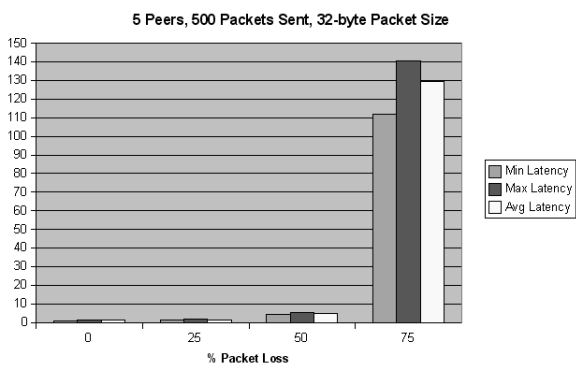


Fig. 3. Latency (ms) for 5 peers, 500 packets.

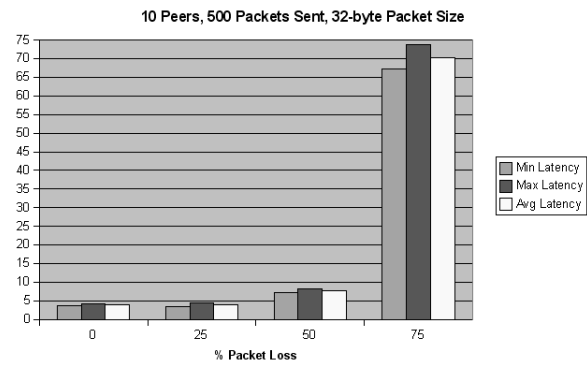


Fig. 6. Latency (ms) for 10 peers, 500 packets.

in one peer's connectivity do not prevent that peer from completing its broadcast, as other peers will be working on the *glitchy* peer's behalf even while the glitch occurs.

We analyzed the impact on the average latency of a message under a varying degrees of packet loss. The results of the test runs are included in Figs. 1 through 6. If you compare Fig. 3 and Fig. 6, for example, you will notice that the latency decreases for the greater number of peers.

Under an idealized scenario with very large bandwidth compared to the amount of logical data that needs broadcasting, the bandwidth consumed by the protocol overhead is large. This is in keeping with our goal of muster-

ing all available resources to minimize latency. When introducing a set of peers with significantly degraded bandwidth, the improvements to latency come from the increase of the amount of logical data broadcast relative to the number of physical bytes transmitted. For example, if one extremely bandwidth-starved peer wanted to send 10 bytes to 7 peers, it would send 10 bytes to the first peer which would rapidly forward the data to all other peers before the slow one had barely begun sending 10 bytes to the next peer. Thus it might accomplish the broadcast of 70 bytes of logical data while only physically consuming 10 or 20 bytes of physical bandwidth.

7. Conclusions and Future Directions

It is clear from our research that P2P networking holds great promise in improving MMOG performance. Indeed, MMOGs are already partitioned into tasks performable by sub-servers. We have discussed partitioning schemes that make more sense in a P2P environment and ways in which some tasks performed by the aforementioned sub-servers could be offloaded onto the clients. The I3P protocol is designed with efficiency in mind, allowing peers to communicate with each other in an economical manner and internally managing non-receipt of transmissions.

Our signal selection and concatenation algorithms are primitive in their current forms. An obvious next step is to develop solid, parameterizable algorithms and derive optimal values. The simplicity of the core signals will aid in modeling and simulation. A parallel effort in developing a more intricate state definition, with more record-keeping and performance variables, will provide a toolbox upon which the algorithms can be built. This could even facilitate the real-time selection of entirely different, simpler algorithms, each optimized for different situations, which would be activated as the situation demanded. These exercises may provide us with insights into improvements to the protocol.

Initial experimental results, however, look promising. Indeed it is clear that sharing the server load by delegating the work to a set of peers within the P2P gaming setting is cost effective for the game server side of things. The I3P protocol is intrinsically capable of handling redundancy and transmission errors.

With respect to implementing I3P in a real-world gaming situation, we are currently working on a multiplayer game built on Microsoft's XNA Framework [11]. The software targets both the Xbox 360, where the game is enjoyed by four players on the same monitor, and a Windows PC, where the I3P protocol will take the place of the player-to-player communication. The simple communication needs of four closely-interacting players resemble the needs of a small group in a MMOG. Initial implementations of I3P in this setting will provide some rudimentary proof of concept; however, more work will be required in order to analyze and prove its efficiency in a MMOG setting.

We must also develop the setup and maintenance procedures necessary for making the I3P protocol useful in games. Particularly, newly-entering peers will confuse everyone else by their sudden appearance unless an additional mechanism is added to prepare existing peers for the discontinuity. Future analysis may also draw more direct analogies to TCP in an effort to glean clues from it for the details of an implementation.

References:

- [1] A.-G. Bosser, "Massively multi-player games: matching game design with technical design," In ACE '04: Proc. of the 2004 ACM SIGCHI Int. Conf. on Advances in computer entertainment technology, pp. 263-268, New York, NY, USA, 2004, ACM Press.
- [2] I. Blizzard Entertainment, "World of Warcraft," 2007. <http://www.worldofwarcraft.com>
- [3] A. Kementsietsidis, M. Arenas, and R. J. Miller, "Mapping data in peer-to-peer systems: semantics and algorithmic issues," In SIGMOD '03: Proc. of the 2003 ACM SIGMOD int. conf. on Management of data, pp. 325-336, New York, NY, USA, 2003, ACM Press.
- [4] X. Ren, K. Li, R. Li, and L. Yang, "An Improved Trust Model in P2P," In 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06), pp. 76-81, Los Alamitos, CA, USA, 2006, IEEE Computer Society.
- [5] S. Iren, P. D. Amer, and P. T. Conrad, "The transport layer: tutorial and survey," ACM Computing Surveys, Vol.31, No.4, pp. 360-404, 1999.
- [6] N. W. G. i. t. Defense Advanced Research Projects Agency, I. A. Board, and E. t. End Services Task Force, "Protocol standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications," 1987.
- [7] C. Shue, W. Haggerty, and K. Dobbins, "OSI connectionless transport services on top of UDP: Version 1," 1991.
- [8] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "An empirical evaluation of TCP performance in online games," In ACE '06: Proc. of the 2006 ACM SIGCHI int. conf. on Advances in computer entertainment technology, pp. 5, New York, NY, USA, June 2006, ACM.
- [9] M. Mauve, V. Hilt, C. Kuhmnnch, J. Vogel, W. Geyer, and W. Effelsberg, "RTP/I: An Application Level Real-Time Protocol for Distributed Interactive Media," 2000, IEF Draft.
- [10] S. Pack, E. Hong, Y. Choi, L. Park, J.-S. Kim, and D. Ko, "Game transport protocol: lightweight reliable transport protocol for massive interactive on-line game," In Proc. of the SPIE, Vol.4861, pp. 83-94, October 2002.
- [11] M. Corporation, "XNA Framework," 2007. <http://msdn2.microsoft.com>
- [12] S. Alda, "Component-Based Self-Adaptability in Peer-to-Peer Architectures," In ICSE '04: Proc. of the 26th Int. Conf. on Software Engineering, pp. 33-35, Washington, DC, USA, 2004, IEEE Computer Society.
- [13] S. Caltagirone, M. Keys, B. Schlieff, and M. J. Willshire, "Architecture for a massively multiplayer online role playing game engine," Journal of Computing Sciences in Colleges, Vol.18, No.2, pp. 105-116, 2002.
- [14] F. Cecin, R. Jannone, C. Geyer, M. Martins, and J. Barbosa, "FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games," In NetGames '04: Proc. of 3rd ACM SIGCOMM workshop on Network and system support for games, pp. 172-172, New York, NY, USA, 2004, ACM Press.
- [15] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza, "Locality aware dynamic load management for massively multiplayer games," In PPOPP '05: Proc. of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 289-300, New York, NY, USA, 2005, ACM Press.
- [16] M. DeLap, B. Knutsson, H. Lu, O. Sokolsky, U. Sammapun, I. Lee, and C. Tsarouchis, "Is runtime verification applicable to cheat detection?" In NetGames '04: Proc. of 3rd ACM SIGCOMM workshop on Network and system support for games, pp. 134-138, New York, NY, USA, 2004, ACM Press.
- [17] K. Endo, M. Kawahara, and Y. Takahashi, "A proposal of encoded computations for distributed massively multiplayer online services," In ACE '06: Proc. of the 2006 ACM SIGCHI int. conf. on Advances in computer entertainment technology, pp. 72, New York, NY, USA, 2006, ACM Press.
- [18] W.-C. Feng, F. Chang, W. Feng, and J. Walpole, "Provisioning online games: a traffic analysis of a busy counter-strike server," In IMW '02: Proc. of the 2nd ACM SIGCOMM Workshop on Internet measurement, pp. 151-156, New York, NY, USA, 2002, ACM Press.
- [19] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk, "Traffic characteristics of a massively multi-player online role playing game," In NetGames '05: Proc. of 4th ACM SIGCOMM workshop on Network and system support for games, pp. 1-8, New York, NY, USA, 2005, ACM Press.
- [20] L. S. Liu, R. Zimmermann, B. Xiao, and J. Christen, "PartyPeer: a P2P massively multiplayer online game," In MULTIMEDIA '06: Proc. of the 14th annual ACM int. conf. on Multimedia, pp. 507-508, New York, NY, USA, 2006, ACM Press.
- [21] C. D. Nguyen, F. Safaei, and P. Boustead, "A distributed proxy system for provisioning immersive audio communication to massively multi-player games," In NetGames '04: Proc. of 3rd ACM SIGCOMM workshop on Network and system support for games, pp. 166-166, New York, NY, USA, 2004, ACM Press.
- [22] A. Yu and S. T. Vuong, "MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," In NOSSDAV '05: Proc. of the int. workshop on Network and operating systems support for digital audio and video, pp. 99-104, New York, NY, USA, 2005, ACM Press.



Name:
Jeremy Kackley

Affiliation:
Doctoral Student, the University of Southern Mississippi (USM)

Address:

118 College Dr., Box 5106, Hattiesburg, MS 39406

Brief Biographical History:

2005 Received B.S. in Computer Science, USM
2006- Teaching Assistant, USM
2007- Research Assistant, Center for Computing Consultancies, USM

Main Works:

- J. Kackley, M. Johnson, K. Yang, P. Wahjudi, J. Gour, and D. Ali, "Nationally organized distributed database system for evacuation planning management," In Proc. of the 2007 Int. Conf. on Industry, Engineering, and Management Systems, Cocoa Beach, FL.
- P. Wahjudi, J. Gour, J. Kackley, and D. Ali, "Applying fuzzy logic to the modeling and prediction of the effect of global pollution on the gulf coast," In Proc. of the 2007 Int. Conf. on Industry, Engineering, and Management Systems, Cocoa Beach, FL.
- J. Kackley, P. Wahjudi, and D. Ali, "A Peer-to-Peer approach to Virtual Private Networks," In Proc. of the 2006 Int. Conf. on Industry, Engineering, and Management Systems, Cocoa Beach, FL.

Membership in Academic Societies:

- Association for Computing Machinery (ACM)



Name:
Jean Gour

Affiliation:
Visiting Assistant Professor, School of Computing, the University of Southern Mississippi (USM)

Address:

118 College Dr., Box 5106, Hattiesburg, MS 39406

Brief Biographical History:

2004- Senior Research Assistant, Computational Science Research Facility for Defense Data Integration, USM
2004- Research Assistant, Database Research Lab for Intelligent Agents, USM
2004 Received B.S. in Computer Science, USM
2005- Research Assistant, DoD-related work on intelligent agents for data fusion
2007 Received Ph.D. in Computational Science, USM

Main Works:

- J. Gour, M. Cobb, P. Wahjudi, and D. Ali, "Smart server: Concepts and applications," International Journal of Intelligent Systems, Special issue on "Intelligent Approaches to Internet Usage and Applications," 2007.
- J. Gour, P. Wahjudi, and L. Guo, "Development, testing, and simulation of a smart phone card prototype," In Proc. of the 2006 Int. Conf. on Industry, Engineering, and Management Systems, Cocoa Beach, FL.
- J. Gour, J. Heath, P. Wahjudi, D. Ali, and M. Cobb, "Smart server: Integrating intelligence into a web server," In Proc. of the 2004 Int. Conf. on Industry, Engineering, and Management Systems, Cocoa Beach, FL.

Membership in Academic Societies:

- Institute of Electrical and Electronics Engineers (IEEE)
- Association for Computing Machinery (ACM)



Name:
Matthew Gambrell

Affiliation:
Programmer, Amaze Entertainment

Address:

816 Congress Ave SE, Suite 600, Austin, TX 78701

Brief Biographical History:

2004 Received B.S. in Computer Science, USM
2004- Research Assistant, Database Research Lab for Intelligent Agents, USM
2005 Received M.S. in Computer Science, University of Southern Mississippi (USM)

Main Works:

- Graphics techniques
- Embedded systems
- Artificial Intelligence in games