International Workshop for Peer-to-Peer Systems (IPTPS) 2006
Scribe Notes

Scribe: Robert Gilbert

An Experimental Study of the Skype Peer-to-Peer VoIP System

Due to its proprietary design and encrypted protocols, not very much is known about the inter-workings of the Skype system. This work involves an experimental study which gathers a variety of measurement information specific to Skype clients and servers. The contribution of the work is to aid in understanding of the system, as well as to guide the development of related architectures.

By capturing five months worth of experimental data (some 13 gigabytes of encrypted Skype traffic) on their supernode, the authors make some interesting observations.

- About 75% of the time was spent on protocol control, leaving 10% dedicated to relay traffic, and the rest of the time spent idling.
- About 50% of the Skype clients were NATed. Thus, these clients never could be promoted to supernodes.
- Of all supernodes, close to 50% came from within a university setting.
- Skype clients follow diurnal, work-week patterns. This is quite different from supernodes which exhibit very low churn (about 95% uptime)

The most interesting result of this analysis is that the Skype network, built upon both file-sharing and telephony principles, is markedly different than the two technologies from which it inherits.

Questions:

- The cost per bit of a telephone call is more than the cost per bit of a Skype transfer. Does Skype traffic really affect universities greatly?

- Does Skype consider uptime when promoting supernodes?
- This was not tested.

- What kind of confidence do you have in the user data collected?
- Due to proprietary and encrypted nature, no confidence metric was used.

- How can you say that Skype is ISP unfriendly?
- Supernodes provide a transit service which potentially bogs down the ISP.

- While universities are hotbeds for supernodes, someone is _still_ paying the ISP for the traffic use. Doesn't this conflict with your argument for ISP unfriendliness?

- ISPs are certainly incurring more of a bandwidth cost than if Skype traffic were not present.

- But the amount of bandwidth used is trivial - so is this really a problem? The university seems to have no problem allowing me to read the NY Times on-line while having no incentive to do so, why would it be different in the case of Skype, a system that uses arguably less bandwidth than my web browser? It seems as though your study is a testament to the quality and potential of p2p systems.

- Are nodes chosen for transit link quality?
- The authors feel that Skype closely models the well-studied KaZaA system - it seems as though nodes are chosen close in the IP space (perhaps utilizing RTT measurements).

Understanding Pollution Dynamics in P2P File Sharing

Pollution has recently been introduced as a defense mechanism to discourage illegal p2p downloads. The idea is to tamper with the content or meta-data of files and then inject these corrupted versions into the network to dilute the available files. A key question to this work is how does user behavior affect pollution spread? The authors have set up an experimental study to answer this.

The authors divide the content download cycle into three phases: the Preparation stage, the Download stage, and the Post-download stage. Using two metrics, pollution awareness probability (the fraction of users that recognize a polluted file) and slackness distribution (the interval between when a download completes and a user checks the quality of the result), the authors carry out their study on 30 human subjects.

The results show that users perform the quality check of downloaded content either directly after the download completes or after more than 12 hours. Furthermore, overall, users exhibit a very low awareness of most types of pollution (most commonly overlooked pollution types were incomplete files, meta-data modifications, and noise insertion). Furthermore, a qualitative result found that faster and more upbeat music tended to be harder to classify as pollution or not.

Questions:

- Does Bittorrent or eMule rank as high as KaZaA in injected pollution?
- This study only took into account KaZaA, as it has been demonstrated to have as high as 70% of content polluted.

- Isn't it true that polluters can modify supernodes to more effectively increase pollution?
- Yes.

- Do you model the evolution of interest among files?
- No, in the author's experience, pollution dynamics depend, in part, on when the attacker launches the pollution attack.

- All of your examples were for music files. Did you test any other file types?
- No, since music is easy to tamper with and much more likely to be re-downloaded in the face of pollution awareness, these file types were solely considered.

Fallacies in Evaluating Decentralized Systems

Much research is being done today in the realm of decentralized systems such as peer-to-peer overlays and ad hoc networks, but, due to the very small number of deployed applications (mostly file-sharing), possible avenues for future work are still rather poorly understood. This work seeks to make explicit some of the key problems stifling research growth.

The authors present three primary problems concerning the current evaluation methodology of decentralized systems.

- The limelight effect: there are certain characteristics of file-sharing systems (e.g., high churn due to a lack of incentive to stay after a download, resulting in low availability, low bandwidth due to use at home and not at work, etc.) that are not directly applicable to other sorts of decentralized approaches. Yet it becomes tempting to generalize these traces to all p2p applications, resulting in a widely pessimistic view that may not be warranted.
- Gravitation: Great research results for specific systems tend to yield further interest in a small number of different environments. Thus, a biased attention is given to a small number of application environments.
- De facto standards: Common evaluation methods are very helpful, a methodology can reach a critical mass. For example, ns2 is the de facto wireless network simulator. It is powerful, but it has simplistic assumptions, such as the random-waypoint mobility model. Thus, there is an incentive in the research community to use this de facto standard even if it would be appropriate to do a different type of evaluation. The results of such evaluations may suffer as a consequence.

The proposed avenue to resolve these issues is to develop and utilize a common library for all simulators, models, and testbeds. This technical solution combined with the non-technical solution of requiring a deep commitment to keep this common library alive would result in a real and potentially more adequate evaluation solution.

Questions:

- Computer architecture has also struggled with these problems (SuperScalar as a de facto standard). Do you see this drastic change a little too optimistic?
- No.

- What about the potential problem that by bringing about more options, critical researchers will not have as much expertise in any of the evaluation models, leading to misunderstanding of perceived results?

- You spoke about the dangers of simulators, but it was a surprise to see where you ended up. It was expected that you'd say "we need to build an uber simulator" or "we need to build more real systems". How is your proposal not just more of the same?
- The authors are not arguing that simulation is but a tool to analyze a specific part of the system, but they feel that evaluation could/should be made easier.

- What you're saying is what ns2 was designed to solve - you're proposing to build another simulator, right?
- ns2 is but one simulator - it's not the end-all-be-all.
- Particular things about ns2 cannot be fixed (e.g., scaling ns2 beyond about 200 nodes).

- How will communities take to new models?

International Workshop for Peer-to-Peer Systems (IPTPS) 2006 Scribe Notes

Scribe: Shaomei Wu

Robust Incentives via Multi-level Tit-for-tat

To deal with the free-riders in peer-to-peer (P2P) file-sharing network, a lot of research efforts have focused on the use of incentive systems to encourage sharing among users. However, these systems are generally vulnerable to variants of the Sybil Attack, in which users take advantage of advantage of the zero-cost nature of online identities to crate multiple identities to actively collude and cheat the incentive systems.

The authors discuss the collusion cases found in Maze, a large and deployed P2P file sharing network and examine two typical incentive designs: Tit-for-tat and EigenTrust. Tit-for-tat algorithm does not scale because the reputation is purely ranked by private history of each peer's personal perspective; whereas, based on shared history and global reputation, EigenTrust suffers from both false negatives and false positives. To achieve the best balance between these two strategies, the authors propose multi-trust algorithm, which is a hybrid of them. Multi-trust incentive derive reputation metric for service differentiation, considering-ideally-all the opinions of the peers in the same network, however, the reputations from first-tier friends are given the highest priority.

The evaluation focuses on two aspects of the multi-trust algorithm: first, the two-level rank metrics is capable enough of boosting the coverage significantly beyond what a private-history solution can offer; second, multi-trust mechanism deals with the colluders as effectively as EigenTrust does while reducing the side effects.

Q & A:

1. How to determine a node is cheating? If a node is trusted, all his friends are trusted, that means, an undistinguished colluder would induce to a big range of unreliability.
   Answer: Multi-trust scheme does not actually define a node is cheating or not; instead, we calculate a queue point for each peer and serve the nodes with lower queue positions first. Multi-trust eliminates the effect of colluding but not prevents it happening.

2. Does Maze have long-tail? If yes, how is it like?
   Answer: Yes. The architecture of Maze basically guarantees that both small files and large files have the same possibility to be long-tail.

3. What is the difference between Tit-for-tat and link spamming? Are there any same problems like link spamming? If yes, how to handle these problems?

Answer: The link spamming case is like the star-shape colluding. We can not actually remove it; only reduce the affect of it.

4. To solute the false positive in global reputation based incentive system, can we rank the pages to get a rough idea of reliability.
Answer: We have tried page ranking, but it doesn't show obvious improvement.

Exploring the Feasibility of Proactive Reputations

Reputation mechanisms can help peer-to-peer (P2P) systems avoid unreliable or malicious users. While reputations assess a peer's trustworthiness using historical feedback of its past interactions, short peer life-times will make the reputations less reliable and more vulnerable to bad-mouthing or collusion attacks, because they are generated from only a small number of past transactions.

To solve this problem and provide reliable reputation ratings for unknown peers or newcomers, the authors propose the idea of proactive reputations, which allow peers to proactively initiate transactions with one or more peers for the express purpose of generating reputation ratings. In the design of proactive reputation, all the nodes anonymize a portion of the messages they originate so that the proactive requests are indistinguishable to malicious nodes from normal traffic. Three different models are proposed to shape normal application traffic: 1. peers anonymize outgoing traffic at a predefined constant rate; 2. peers vary their rate of anonymous transactions at some predefined time interval; 3. peers dynamically define a random number of messages N and a random anonymization rate R for these messages, and reset both values after every N incoming messages.

Given these three anonymization models, the authors deploy a set of simulations to examine the effect of varying the length of proactive bursts injected into the network. The effectiveness of these mechanisms is measured with the metrics of Absolute Difference (AD). For low size of message window, experiment results demonstrate that bursts of proactive requests blend in well and are nearly undetectable under traffic analysis by target peers.

Q & A:
1. Have you considered non-uniform file storage?
Answer: Not yet, our system currently considers only uniform sized file blocks.

2. Can you use proactive reputations along with the regular reputation systems?
Answer: Yes.

3. Which one would a peer choose to believe, a global reputation or a proactive reputation?
Answer: Since the proactive reputations are firsthand, they are more trustworthy.

4. Will you test each newcomer in the system using this scheme?
Answer: no, proactive reputations will not be used each and every time. Peers will risk interacting with newcomers. However, a few peers will opt to proactively transact and verify.

Byzantine Fault Isolation in the Farsite Distributed File System

In a peer-to-peer (P2P) system of interacting Byzantine-fault-tolerant replicated-state-machine groups, as system scale increases, so does the probability that a group will manifest a fault, because the probability of group containing too many faulty machines to suppress the fault will grow with the increase of BFT groups. The analysis of operational fault rate shows that when the scale reaches 10^5 groups, the operational fault rate is 0.45, which means, almost half of all operations exhibit faults.

Byzantine Fault Isolation (BFI) is a method to prevent a faulty BFT group from corrupting the entire system. Technique of BFI is based on using formal specification to design a distributed system with well-defined semantics. BFI semantically specifies the faulty behavior that can manifest when faults occur in the distributed system. In BFI spec, the authors modify formal distributed system specifications by augmenting the state of each machine with a new flag indicating whether the machine is corrupt, and adding a new action to set a machine's corrupt flag.

Compared to increasing the size of BFT groups, an alternate way to reduce the large-scale operational fault rate, BFI doesn't need redundant computation or intra-group message traffic overhead. With a tree-structured system of 10^5 BFT groups, the authors also quantified the benefit of BFI to scalable systems: BFI can achieve the same operational fault rate as 10-member BFT groups without the corresponding 60% drop in throughput.

The authors also show an example of BFI in Farsite, and prove that file corruption cannot spread to unrelated regions of the file-system namespace when BFI is employed.

Q & A:
1. What is the Farsite Semantic Spec for?
Answer: Formally specify the system.
2. What is the biggest problem in BFT groups?
Answer: BFT groups have negative throughput scaling; scalable systems can be built with BFI.
3. Is there any redundancy existing in your system?
Answer: Yes, but we know the children and parents of all the nodes, so we know where the redundancy is.
4. How do you form groups?
Answer: We design and select some physical machines to become a group.
5. How do you prevent nodes behaving badly?

Answer: There is no way to prevent nodes behaving badly, what we do is to isolate the bad nodes from the entire system.

6. What is the intuition behind Farsite?

Answer: The Farsite spec includes a semantic specification and a distributed system specification. They combine to define a large distributed P2P file system with high scalability and strong consistency.

7. What is the optimal size of groups?

Answer: We don't know. From our experience, 4 members in a group is good enough.

8. Is there a subnet problem in BFI?

Answer: No. Because we have assumption that every machine is equally close to each other.

9. Is there any commercial application of Microsoft on this project?

Answer: No.

10. Is this solution realistic? There will still be some machine with faults and we cannot access them anymore.

Answer: Yes, it is a question. Ideally, don't trust any other machines to store your sensitive data.

International Workshop for Peer-to-Peer Systems (IPTPS) 2006 Scribe Notes

Scribe: Stacy Patterson

Proactive Replication for Data Durability

In data intensive DHT applications, replication is often used to ensure data durability in the face of node failures. Previous approaches to replication, such as that used in DHash, are reactive. A new replica is created when an existing copy is lost, due to failure or departure, in order to maintain a constant number of replicas. The bandwidth footprint associated with this method is bursty which is undesirable in certain scenarios, for example when a user has to pay for peak usage.

This work advocates a different approach replication, namely proactive replication. In proactive replication, new copies of data are constantly being made in the background, with the ultimate goal of achieving a constant replication rate. To this end, the authors introduce Tempo, a DHT that uses proactive replication. In Tempo, each node has a maintenance budget that defines the amount of bandwidth a node can use to proactively create more replicas.

Experiments with Tempo show than it has lower and more uniform bandwidth usage than required for a reactive approach. Tempo also provides the while the same level of data durability as the reactive scheme. Future work will address the question of how to best determine which objects to replicate and when to replicate them and how to determine an optimal maintenance bandwidth budget.

Q/A
Q: Have you looked at expressing the tradeoffs between reliability and bandwidth as an optimization problem (minimizing bandwidth while maximizing durability)?
A: We have not looked at that, but it could used to determine a maintenance policy, meaning the policy of what objects to replicate and when to create new replicas.

Q: Does data eventually become obsolete?
A: We are only interested in data durability so we assume we want to keep files around for a long time. The system designer could come up with an expiration policy for data, but that is not part of this work.

Q: If multiple copies of an object disappear simultaneously, is it possible that there may not be enough bandwidth in the system to create new copies before all copies are lost?
A: This is possible, but it seems to be more of a risk for reactive systems. The hope is that in a proactive replication system, enough copies will have been created to survive these failures. It is also possible to base the maintenance policy to give higher priority to

replicating objects that have the smallest number of copies.

Q: Do your experiments consider the performance of the system as a function of the amount of data it is storing?
A: We do not have experiments on this.

Comment: It was suggested that a hybrid approach, using both proactive and reactive replication, might make sense in the problem setting.

F2F: Reliable Storage in Open Networks

An ideal P2P storage system should be automatically scalable and reliable. However, in many P2P systems, nodes have little incentive to remain the system, for long periods of time. This high degree of churn results in a high cost of

Rather than presenting a scheme to deal with churn, the authors advocate creating a P2P system that will, by its nature, have a low churn rate. The authors propose an open system design that has an intrinsic incentive for peers to remain in the system for long periods of time. The system, called f2f, is based on a social network, where nodes only share network and resources with friends. In f2f, nodes presumably remain in the system because they want to "help their friends". Additionally, if a node goes offline, it is possible to use out of band information to determine the nature of the failure. A friend can call up the friend with the failed machine and tell him to turn on his computer, or in the case that the friend's hard drive has failed, remove the friend from the network. Drawbacks to the f2f design are that it may not achieve storage efficiency since each node can only make use of its immediate neighbors. f2f also does not provide any efficient data location service.

The authors also present a case study of using f2f as a distributed backup, called Block Party. In the Block Party system, each node donates some space to be used to backup other computers. Each node's data is divided into chunks and the chunks are backed up at friends' computers. In this application, there is no need to search for data since it is located at the node's immediate neighbors. Block Party also requires less maintenance bandwidth than other P2P-based distributed backup systems.

Q: Is only one copy made of each file block? In other words, is the replication factor 1?
A: Our experiments use only a replication factor of 1.

Q: If you only make one copy of the data, at which friends do you chose to store this copy?
A: The blocks are distributed across multiple friends.

Q: How do you balance the load of the system, assuming that the number of friends follows a power law distribution?

A: There is no problem with load balancing for nodes with high degrees. Nodes with low degrees (1 or 2 friends) are problematic. If you can enforce a minimum degree for each node by requiring each user to have a minimum number of friends, then load balancing is not an issue. The incentive for a user to have lots of friends is that he can back up all of his data.

Q: In existing systems, some sort of measurement is used to decide when a machine has left the system permanently, and the system then automatically replicates to another machine. In f2f, what happens when a machine fails?
A; A user only cares if a friend has permanently failed. For example, permanent failure could occur if the friends hard drive fails. So, when the friend comes back online, all data that was previously stored at that friend's machine is lost. We anticipate that friends will notice if other friends have been offline for a long time and then contact those friends out of band.

Q: Can you use f2f in a system that is not well-provisioned? Would it work in a DSL-based system or would the bandwidth requirements be too great?
A: We believe it is possible to run this system on DSL machines. We would need to use something like Temp to smooth out the burstiness of maintenance traffic.

On Object Maintenance in Peer-to-Peer Systems

The goal of this work is to explore how churn effects the maintenance overhead for both proactive and reactive replication schemes. In proactive replication, an estimate is made on node availability, and replication is done in the background, using this availability estimate as a guide, to maintain a constant number of replicas. In reactive replication, a new copy of data is made "on demand" when a copy is lost due to node departure.

The authors distinguish between two types of churn, temporary and permanent, and explore the replication strategies under each type of churn independently. Temporary churn describes nodes that leave the system temporarily and return, with all data in tact. In temporary churn, data is not lost by failures, and so it is possible to mask temporary churn by creating enough copies of the object to survive the minimum level of node availability. In both the proactive and reactive replication approaches, once the necessary level of redundancy is met, maintenance overhead is low.

Permanent churn describes the permanent departure of nodes, and thus the loss of data on these nodes. When a node leaves the system, "repairs" are needed to create new copies of the objects stored at it. In this case, the maintenance overhead, or repair frequency, depends on the rate of churn and the amount of redundancy. Since the amount of redundancy is determined by the rate of temporary churn, the maintenance required for permanent churn is actually dependent on the rate of temporary churn.

Additionally, the authors investigate the effects of storage capacity on maintenance costs. Since highly available nodes are online more often, they are chosen more often to back up data. Therefore less redundancy is required to mask temporary churn. As storage utilization increases on these highly available nodes, lower available nodes must be used to store replicas. Therefore, the repair cost increases as nodes fill to capacity.

Q: Do you store the file or just index?
A: We store the file.

Q: Rather than deciding some arbitrary cutoff for differentiating between temporary and permanent churn, would it make sense to ask the question "Is this object available when I need it?" and evaluate the replication scheme from this viewpoint? If a file is lost and no one needs it, then this is not a problem. It might make sense to consider query load rather than object availability.
A: The goal of this work is to provide 100% availability of the object. We did not consider the query load aspect.

Comment: The characteristics that different temporary and permanent churn may be system dependent.

Panel

Q: Do we have a hope of solving the churn problem?
A: Since we can't determine permanent failure rates ahead of time, we need to replace data as soon as it is lost.

Q: Current trend is that storage rates double quickly, but bandwidth increases slowly. Will there be a time when online back is not possible? And then what will we do?
A: Online backup is impossible if the disk is so unreliable that the amount of time required to transfer the data exceeds the lifetime of the disk.

Q: Even if data is fully backed up online, it might take a month to recover all of the data.
A: You don't need to wait for all of the data to do useful work.

Q: You don't need to back up your entire hard drive. You only want to back up the files that it took you time to create. So you need the backup rate to be faster than the creation rate. Does this mean that the backup rate can be a constant?
A: No. As technology advances, rate of creation will increase. The size of digital photos will grow. The size of movie files will increase, and users will be able to acquire (download) them more quickly. Local backup is always an option if online backup in infeasible.

Q: In order to distinguish between temporary and permanent churn, would you need permanent failure detector?
A: You a might not need to be able to distinguish between the two types of churn, but if work is done in response to a temporary failure, and the node rejoins the system, then that is work that does not need to be done in the future when there is a permanent failure.

Q: If I wanted to create a high availability data center with multiple sites and hundreds of machines, what techniques from this work will be transferrable to solve this problem?
A: The fundamental idea is that one must copy data faster than it is lost. Any technique used to address this end can be used in the above situation.

Q: Is bandwidth the limiting factor of these systems? If so, can we exploit locality to help with the bandwidth problem? We can use local bandwidth for replication and long distance bandwidth for durability.
A: We want to minimize the object maintenance cost and local operations might be less expensive.

Q: Have you considered the notion that not all data needs wide-area replication?
A: This is not part of the systems now, be we can consider it in the future.

Q: You can't necessarily rely on social connections as an incentive to provide durable storage. Once friends are required to do work to store data, then may disappear, or they may even hold data hostage.
A: This should encourage you to make quality friends.

Q: Have you considered using coding schemes?
A: We have thought about it. A coding approach would require less storage space to back up the data and would provide security, allowing backups to be stored at friends' friends. But, we have not yet done any work on this.

Q: What if wanted to delete the data from the system?
A: You would need a delete operation. In order to delete the object, it must be possible to locate every single copy. In a system like Tempo, this may not be possible. Also, temporary churn is problematic because a machine may be offline when the delete is performed.

International Workshop for Peer-to-Peer Systems (IPTPS) 2006 Scribe Notes

Scribe: Lakshmi Ganesh

Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast

Peer-to-Peer multicast solutions typically use one of two strategies - tree-based, and tree-less. Tree-based algorithms have the advantage of no latency overhead, but tree construction and repairs are complex and time consuming. On the other hand, tree-less swarming algorithms are robust and fair but can result in high latency overhead.

The optimal solution, therefore, attempts to combine the advantages of tree-based and tree-less algorithms while avoiding their setbacks. Fairness and robustness are achieved by using the multi-tree approach of so-called tree-less algorithms. To avoid the attendant latency issues, a fast tree-building mechanism is proposed. Nodes first build a neighbor graph using an algorithm called 'Swaplinks'. Trees are then constructed over this neighbor graph by having nodes communicate with their neighbors to negotiate optimal parent-child relationships, while satisfying maximum out-degree constraints. Two issues arise with this approach: loop avoidance, and load-balancing. Bloom filters are used to detect loops; subsequently, those parent-child relationships are avoided which result in loops. To address load-balancing, local negotiation of parent-child relationships is instituted: children tell overloaded parents about their other (under loaded) neighbors; the overloaded parent then chooses the best (based on load, latency etc.) such neighbor to take over as the new parent.

The performance of this system was measured by a flash-crowd simulation: nodes join, leave in bursts. The system was seen to perform well. In conclusion, Chunkyspread is simple and achieves fine-grained control over load with good latencies, and is best suited to non-interactive live streaming applications.

Questions:

Q - This algorithm is still tree-based - what of repair overheads?
A - As future work, we are thinking of using the swarming approach specifically during unstable phases, thus avoiding repair overheads.

Q - Won't parent-child switches cause jitter?
A - Care is taken that these switches will not result in loss of slices; however, they could cause redundant slices.

Q - What happens when lots of nodes join together? Bottleneck?

A - We tested our simulation for 50% nodes joining at really high speeds and it seemed to work. ☺

ChunkCast: An Anycast Service for Large Content Distribution

Content Distribution Networks (CDNs) typically slice and distribute data since it is known that chunking improves download time. Object downloads then require the location of a set of peers that together store all of the chunks of that object. Existing applications address this problem in one of two ways: chunk-level queries are issued, resulting in one query per chunk of an object - this causes high overheads; alternatively, object-level queries are issued which seek only to find a subset (*any* subset) of peers that together contain all the chunks for that object - no attempt is made to optimize peer selection, resulting in potentially sub-optimal download times. ChunkCast provides the optimal solution by maintaining a distributed implicit indexing tree per object. As a result, object-level queries are supported, thus minimizing query overheads; additionally, optimal peer selection is supported, so that download time is optimized. Finally, optimal chunk selection is also supported - i.e. the order in which chunks should be downloaded is determined so as to optimize network bandwidth utilization.

ChunkCast is based on a publish-subscribe model. Objects are identified by Object IDs (OIDs), and chunks are represented by bit-vectors. A publish message is routed towards the root of the OID, and each node in the forwarding path stores state (i.e.. which chunks of which object are stored at which node). Lookups can then be satisfied incrementally by nodes on the path to the root of the object. The root of the object stores all state about the object. Peer selection is governed by some ranking function R (random/locality-based).

A ChunkCast prototype was implemented on Bamboo using Vivaldi as its network coordinate system. As future work, it is planned to implement BitTorrent clients that use ChunkCast for content distribution and lookup.

Questions:

Q - Every query must travel all the way up the tree - this presents a bottleneck, does it not?
A - Not really. Lookups are cheap; besides, peer information is periodically refreshed.

Q - In actual BitTorrent systems, the filtration (for peer selection) would return very few candidate peers, would it not?
A - This needs to be evaluated; as future work, we plan to run BitTorrent using ChunkCast to see what the impact is.

Q - BitTorrent peers advertise new pieces to their neighbors so that this information is known immediately. Here, however, you have to do a lookup in order to get this information. So BitTorrent would be more responsive, right?

A - We can use the BitTorrent advertising schemes in conjunction with ChunkCast's algorithm; you will thus have responsiveness, as well as good peer selection.

Tribler: A social-based Peer-to-Peer system

The goal of this project is to create a near-cast infrastructure for content creation, distribution and lookup. This is unlike existing content distribution solutions, which typically only provide a distribution framework. Tribler is a next-generation P2P application that will understand your friends and your tastes, in addition to keeping track of network congestion etc. Being social-based, Tribler eliminates the incentives problem that P2P systems typically face. Additionally, since transactions occur between nodes sharing long-term relations (such as 'friends', or 'friends of friends'), the issue of churn is circumvented.

Tribler is essentially a set of extensions to BitTorrent. It addresses several of BitTorrent's deficiencies by supporting inter-swarm communication, adding strong authentication (using challenge-response with public keys) and providing better content identifiers. Additionally, Tribler supports semantic clustering through gossip about previous download behavior. BitTorrent's tit-for-tat scheme limits nodes' download speeds to their maximum upload speeds - this can prove an annoyance while using ADSL connections which give you small upload speeds; Tribler overcomes this limitation by providing cooperative downloading support.

Apart from the design ideas outlined thus far, a significant contribution of this project is in the amount of data that has been gathered and analyzed about various real-world P2P systems.

Questions:

Q - How do peers bootstrap? How do peers get to learn of peers in other swarms?
A - Through gossip. Nodes can also ask their neighbors to tell them about other nodes they know.

Q - How do you exchange information? One for one?
A - No, there is no inter-swarm tit-for-tat.

Q - How do you deal with transitive trust relationships? I trust my friend, who trusts his friend, but I don't trust his friend.
A - I'd like to see that this is a pressing issue before we attempt to address it.

Request addressed to all three speakers: These are all great applications which will hopefully see wide usership before long; Could you add a logging function so that we can obtain nice datasets?!

International Workshop for Peer-to-Peer Systems (IPTPS) 2006 Scribe Notes

Scribe: Stacy Patterson

Fair File Swarming with FOX

In a BitTorrent-like content distribution system, files are divided into blocks and random peers contact each other and trade blocks. This approach is efficient, but cheating is common. Peers are often greedy and do not wish to upload blocks to other peers. To address this problem, the authors propose FOX, the Fair, Optimal eXchange protocol. The goals of FOX are to minimize system-wide download time while guaranteeing the every peer contributes as much in terms of uploads as it receives in terms of downloads.

Analysis is first given in terms of an idealized structure, and then the assumptions are relaxed to yield a practical solution. The ideal structure is a k-ary tree with the server at its root. It is similar to a multicast tree, except the server sends a different block to each of its children. Blocks are exchanged at the leaves as well as from children nodes to parent nodes. This structure allows for pipelined data flow and optimally efficient download time. It also allows for punishment of nodes that don not upload to others. Since the download structure is cyclic, a node can punish its upstream neighbors using a "grim trigger", thus cutting off downloads.

While this scheme guarantees fairness, it is vulnerable to "meltdowns" if nodes do not cooperate. To increase the robustness, the authors virtualize the tree structure, associating a community of nodes with each node in the tree. Community membership is restricted; IDs are generated using consistent hashing. Within communities, fairness in ensured through a tit-for-tat mechanism. In the case where an entire community is unfair, the grim trigger is invoked. Communities make the system resilient to cheating peers while still providing the provable fairness.

Q: Do you create a community based on locality?
A: In order to ensure fairness, communities must be restricted, so this does not allow for locality-based community selection. This could be the subject of future work.

Q: Does the pipeline model assume that there is uniform bandwidth on all links?
A: Yes. We have also considered an extension where each server sets up multiple FOX instances each using a small amount of bandwidth.
A server joins a number of FOX instances based on the amount of bandwidth it has.

Q: Do you assume that identities can be verified.
A: In the ideal case, verification is not needed. In the community system, ID verification is necessary.

Q: The punishment mechanism is vulnerable to Denial of Service attacks. Is the

assumption that if one node goes down, the entire system fails?
A: In the idealized case, the failure of one node brings the entire system down. In the community-based approach, the system can tolerate single node failures.

Q: Why does a child need to make use of a long cycle to cut off flow to its parent? Is it possible for the child to apply direct punishment? Can you just change the graph if a node is not cooperating?
A: We cannot apply direct punishment because it can be difficult to figure out exactly which node is cheating

Q: Can we organize nodes to form communities that optimize the system performance?
A: Ideally, yes. However, in order to ensure fairness, we need to use a restricted community join model.

Exploiting BitTorrent for Fun (But Not Profit)

The robustness P2P content distribution system like BitTorrent depends on the willingness of peers to contribute to the system by uploading as much as they download. Since it is not in a peer's best interest to upload files, the system must provide incentives for peer participation. To ensure fairness, BitTorrent uses a tit-for-tat mechanism to prevent peers from downloading without uploading. However, this mechanism is vulnerable to exploits. This work investigates three BitTorrent exploits and their effects on the robustness of the system.

The first exploit deals with peers that download without uploading to other peers. In this case, honest peers can suffer when selfish peers use up too much bandwidth. Experiments show that the robustness of the system can suffer from this attack. In the second exploit, peers locate the fastest peers to download from. In a closed setting, selfish peers achieve a faster download rate, but the public BitTorrent system is robust to this type of attack. In the third exploit, a peer falsely advertises the rarest pieces of a file. These pieces carry the highest value allowing the peer to download more from other peers. This exploit is a moderate fairness violation, but in practice the robustness of the system does not suffer. The attacker is able to download files more quickly but then is also able to provide the file to other peers.

From examining the effects of these exploits, the authors have identified guiding design principles that can be used in P2P content distribution systems to minimize the effects of the exploits. In future work, the authors plan to investigate the effects of combined exploits as well as the possibility of devising a methodology to create new exploits. Additionally, they want to investigate the effects ofselfish behavior in multi-torrent systems.

Q: One exploit involves connecting only to the "fastest" peers. What is considered fast

to one peer may not be considered fast to other peers. So, is it possible this exploit does not have much effect?
A: Connecting to the fastest peers has an effect because it deprives other peers of bandwidth.

Q: How do you determine which peers are the fastest peers?
A: We observe advertisements made by other peers. Frequent advertisements indicate fast peers. In the future, we could also check latency to peers.

Q: Would the effects of connecting to the fastest peers be tempered because if every peer used this exploit, no one would benefit?
A: Our assumption is that there are very few selfish peers, but this is an interesting point.

Q: Is the exploit in which peers do not upload really an exploit? Isn't it somewhat common for peers not to upload because they are behind firewalls?
A: If every peer could use this exploit, then eventually the system would become a client/server system. Eventually, the server would crash.

Anatomy of a P2P Content Distribution System with Network Coding

New P2P systems are emerging as scalable and cost effective solutions for content distribution. In these systems, when a server makes a file available, the file is divided into blocks. Blocks are uploaded to different peers and peers exchange blocks until every peer has every block. One of the major drawbacks of this approach is that it is difficult to find an optimal scheduling of blocks in a distributed fashion. This work proposes Network Coding as an alternative scheme for P2P content distribution that does not require scheduling.

In Network Coding, the file is divided into linearly independent pieces. Any number of these pieces can be combined at a peer to generate a new linearly independent piece. One a peer receives enough pieces, it can reconstruct the entire file. The peer can verify the integrity of the file by using a Secure Random Checksum (SRC) received from the server via a secure connection.

Network Coding allows for a simpler system design, as peers do not need to locate rare blocks or make any other download decisions. The authors demonstrate the feasibility and practicality of Network Coding through a prototype implementation. Tests show that a P2P content distribution system using Network Coding can result in a ten fold savings for the content provider over a client/server approach. Network coding also overcomes the slow rate at the start and end of downloads observed in BitTorrent-like systems, achieving a near constant download rate. Additionally, the prototype allows users behind firewalls and NATs to participate in uploads by initiating connections in both directions. Finally, the results show that the overhead of encoding and decoding the blocks is low, making Network Coding a promising alternative for P2P content distribution.

Q: This scheme avoids the need to find specific blocks. Instead peers can download from

every other peer. How does this help performance?
A: We can give preference to users with more bandwidth. They can create more connections.

Q: Do the random numbers used for checksums need to be revealed publicly?
A: They are exchanged between each client and the server securely. We have done some work on creating a publishing SRCs but it is not included in this paper.

Q: What is the generation size?
A: Coding of a file is not related to the number of nodes in the network. Performance depends on the number of blocks and the number of generations.

Q: If you assume the server is online for the duration of the download, can a conventional hash function such as SHA1 be used to ensure the integrity of the file?
A: We cannot use that approach in our system. Homomorphic hash functions do not work in galois fields.

Panel

Q. For Avalanche, how do you consider fairness?
A. Christos: We have controlled user population so we do not care about fairness. I believe that people may not upload files to others because they don't want to get caught. This is not the case in Avalanche so we do not need an incentive mechanism. Dave: Fairness applies in other cases than illegal downloads. Limited upload bandwidth is one example. A rational peer would like to use as much download bandwidth as it can. Nikitas: I agree with Dave. People freeride for both reasons.

Q. Is Bit Torrent so bad that it needs so much research?
A: Christos: Bit Torrent does not give you assurance that download will work in ALL cases. Network coding does. We want system to work well "no matter what". Nikitas: Bit Torrent is a good idea. It is popular for a reason - it works. It was not developed by a researcher, but researchers should explore why it works.