SELLS: A Space Efficient Local Look-up Search Peer-to-Peer Protocol for Trustworthy Key Distribution

Graciela Perera, Robert Kramer, Anthony Weaver
Youngstown State University
gcperera@cis.ysu.edu, kramer@cis.ysu.edu, aweaver01@student.ysu.edu

Abstract

Unstructured peer-to-peer (P2P) networks for content distribution are decentralized and robust. Searching for content in the network is based on the Gnutella protocol. The broadcast search paradigm used by Gnutella is inefficient and generates much traffic overhead from query messages. The broadcast updates with local look-up search (BULLS) protocol, based on Gnutella, can reduce traffic overhead at the expense of a space inefficient global data structure. Improving BULLS can be achieved by the use of a Bloom filter (BF). A BF is a space efficient probabilistic data structure for membership queries that is widely used in protocols and applications. That is, they can be used to efficiently determine if a file is stored at a host. The cost is a small, bounded probability of error called false positive. The false positive rate can be tuned according to application requirements.

In this paper, we introduce the use of Bloom filters in the design and evaluation of a P2P protocol that enables a space efficient local look-up search (SELLS). SELLS is enabled by including Bloom filters in the BULLS protocol. SELLS stores all the shared files in the network in a data structure called an inverse Bloom filter (IBF). IBF uses a global Bloom filter (GBF) and a set of local Bloom filters (LBF). The GBF stores the files that are improbable to be found or downloaded from the network. Additionally, there is one LBF per host in the network storing the shared files of a given host. Because the Bloom filter summarizes the files shared at a particular host, SELLS maintains the confidentiality of the files shared, is space efficient, and reduces traffic overhead by locally searching for files. We also explore the application of SELLS for trustworthy key distribution.

Introduction

Popular unstructured peer-to-peer (P2P) networks, such as Gnutella, distribute content (files) in a decentralized manner, are self-organized, and are robust [1]. Additionally, the advent of wireless and mobile communication has enabled more people to exchange or share files anywhere, anytime, and on any device. Evidently, for P2P to be widely adopted and deployed on any device, it is of great importance to design space efficient protocols with reduced traffic overhead.

The broadcast search paradigm used by Gnutella is inefficient and generates much traffic overhead from query messages [3, 4]. Users in a Gnutella file sharing P2P network search for files by broadcasting queries or flooding the network with queries [9]. Much of the traffic is overhead from the flooding of query messages and the associated queryhit response messages from searches for popular files [3, 5]. Many P2P protocols focus on limiting query flooding and do not allow hosts to know what files are shared by other hosts [2].

Recent P2P file sharing networks, such as FastTrack (i.e., Kazaa), use the concept of supernodes or ultrapeers to proxy search requests from other hosts called leaves and limit flooding [2, 4]. Limiting flooding excludes the leaves with low probability of responding to the queries from file searches. Ultrapeers store the directory of the files shared by each of its assigned leaves [9]. Although ultrapeers know the files shared by its leaves, they do not know what files are shared by other ultrapeers. The BULLS protocol, unlike FastTrack, can allow each ultrapeer to determine the entire set of files shared in the network at the cost of a space inefficient global data structure [1]. BULLS' global data structure's space efficiency can be improved by representing the files shared by each host with a Bloom filter (BF). A BF is a space efficient probabilistic data structure for membership queries widely used in protocols and applications. That is, a BF can be used to efficiently determine if a file is stored at a host. The cost is a small probability of error called a false positive. The upper bound of the false positive rate can be tuned according to application requirements [7, 8].

If traffic overhead is reduced and each ultrapeer can efficiently store all the files shared in the network, then flooding would become suitable for a wide range of applications that have not been explored by existing P2P networks. In particular, it is of interest to study a novel P2P application that allows the distribution of secret keys instead of files. A P2P application for distributing secret keys will allow communication between users to be secure and independent of a central authority [11].

In this paper, a new design and evaluation of the BULLS protocol is investigated. The new protocol is called space efficient local look-up search (SELLS). SELLS has all the desirable properties of BULLS and a space efficient data structure that stores the shared files. SELLS stores all the shared files in the network in a data structure called inverse Bloom filters (IBF). IBF uses a global Bloom filter (GBF) and a set of local Bloom filters (LBF). The GBF stores the files that are improbable to be found or downloaded from the network. Additionally, there is one LBF per host in the network. Each LBF stores the shared files of a given host. Because the bloom filter summarizes the files shared of a particular host, SELLS maintains the confidentiality of the files shared, is space efficient, and reduces traffic overhead by locally searching for files. Also, a novel application of SELLS for trustworthy key distribution is discussed.

Protocol Description

Current popular P2P protocols based on Gnutella use the concept of ultrapeers and leaves to reduce the query/queryhit messages exchanged or overhead traffic [2]. Although, the BULLS protocol reduces overhead traffic in most cases when compared to Gnutella, it can be improved by including in its design the new concept of ultrapeers described in the Gnutella protocol version 0.6 [1, 9]. The new protocol description of BULLS, which SELLS is based on, is represented using a finite state machine (FSM).The notation for the FSM diagram used in this paper shows the states as vertical lines and transitions as horizontal arrows, indicating the direction of the transition. The transitions are triggered when the input or condition specified above the arrow is met. The output or actions are specified below the arrow and occur simultaneously while making the transition. The dotted arrows are the initial and final transitions in the FSM. The initial transition does not have an originating state, and final transitions do not have a destination state.

The FSM that describes the new protocol behavior of BULLS is only related to the exchange of messages (i.e., query, queryhits, and updates) as this is considered to be the overhead traffic. Low data rate message exchanges from connectivity maintenance (i.e., ping and pong messages) are not considered in the FSM. File downloads (one per queried file found) are not considered because not only have these been already studied and improved upon [2], these are non-overhead traffic that is useful.

BULLS

BULLS is a P2P Gnutella-based protocol that reverses the broadcast search paradigm and explores broadcasting file updates instead of queries. Figure 1 is the BULLS FSM based on Gnutella protocol version 0.6. The FSM shown in Figure 1 only describes the behavior of an ultrapeer host. Ultrapeer hosts, not leaf hosts, exchange overhead messages (i.e., query and queryhit messages). The behavior of a leaf host in BULLS is the same as in Gnutella; that is, only generating query message overhead traffic. The queryhit message response from an ultrapeer to a query message from one of its leaves is omitted from the FSM because it does not impact the overall overhead queryhit traffic. In addition, the data structure used by BULLS is only stored by ultrapeer hosts. Each ultrapeer host stores in the data structure its own share file listing (set of shared files) and the shared file listing of the leaf hosts connected to it. The four states for the improved version of BULLS with ultrapeers are defined as in reference [1]. They are Initialize, Idle, Search, and Select. A detailed description for each of the states and transitions is given below:

- **INITIALIZE**: An ultrapeer host entering the network can be in this state by requesting to receive neighbor addresses of ultrapeers (neighbors) or leaves and downloading the data structure from a specialized bootstrapping host. Upon the reception of a response with the requested neighbor addresses, the ultrapeer host connects to its neighbors, forwards its own shared file list (one update message per file shared) and the share file listing of the leaves (one update message per file shared) to ultrapeer neighbors (neighbor host that are ultrapeers) only, and transitions to Idle.

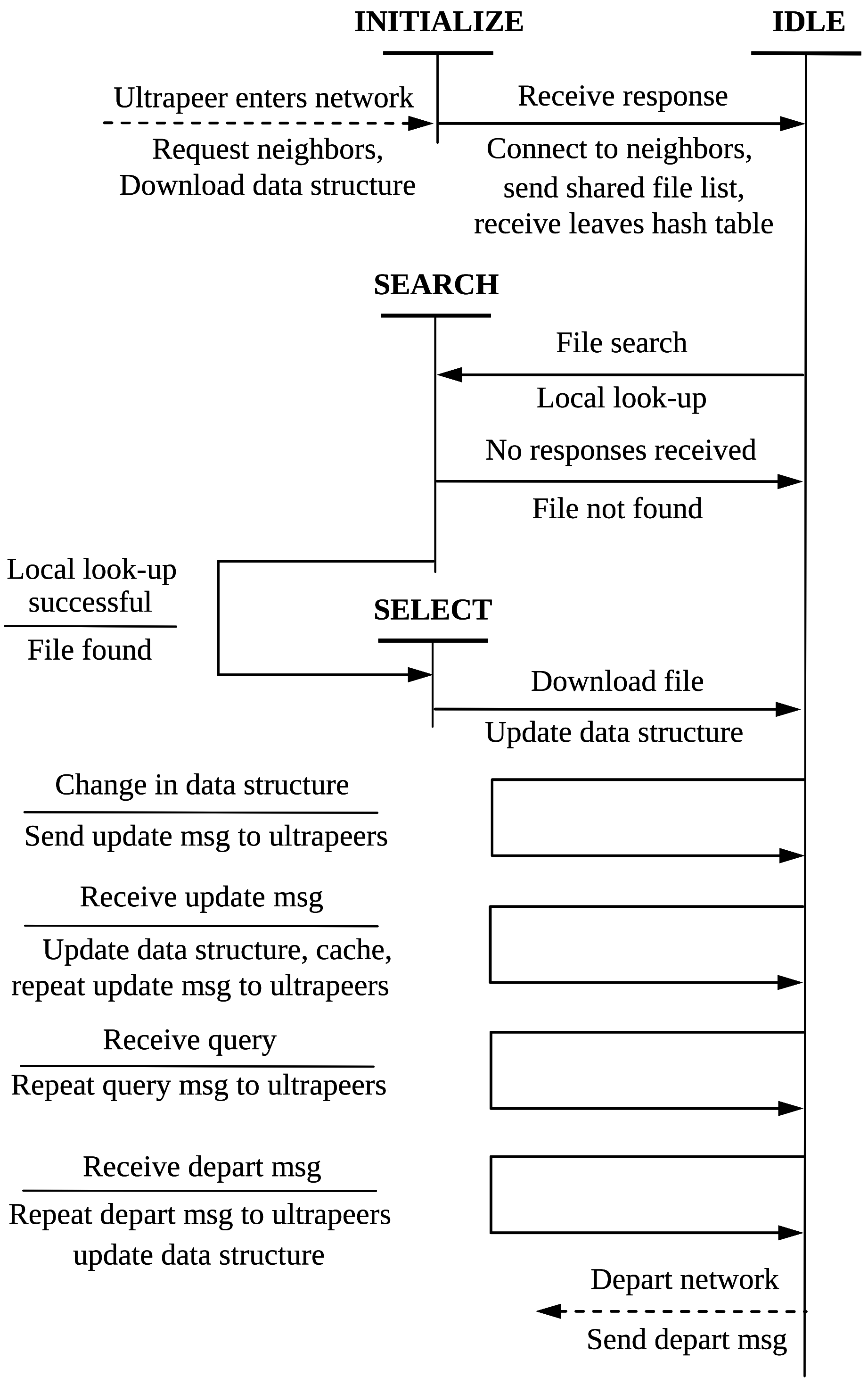


Figure 1. BULLS FSM based on Gnutella version 0.6

- **IDLE**: In this state, an ultrapeer host can: 1) make a file search by a local look-up in the data structure and transition to Search; 2) detect a change in the data structure, repeat to ultrapeer neighbors via an update message the changes in the data structure (one update per change), and remain in Idle; 3) receive an update message, modify the data structure with the update received, store it in the cache, repeat it (send an update message to all ultrapeers neighbors except the one from which the message was received), and remain in Idle; 4) receive a query message from a leaf host, repeat the query to all of its ultrapeer neighbors, and remain in Idle; 5) receive a depart message, update the data structure by modifying the departing host's row entry, and repeat the depart message to ultrapeer neighbors; or, 6) disconnect from the network by sending a depart message.

- **SEARCH**: In this state, the ultrapeer host waits for results from a local look-up and it can: 1) transition to Select if the local look-up is successful, or 2) transition to Idle if the local look-up does not return results.

- **SELECT**: In this state, a host is selected from which to download a file (the host can be an ultrapeer or a leaf). The set of possible hosts to select from is returned by the successful local look-up executed in the Search state. The ultrapeer host downloads the file, updates its shared files, updates its data structure, and transitions to Idle.

The transitions that impact the amount of overhead traffic generated are the same five transitions that impact the overhead traffic in reference [1]. The transitions that impact the amount of overhead traffic cause the broadcast of the shared file list and the broadcast of updates when the shared file list is modified; that is, when a file is added, deleted, or downloaded. Also, the data structure used by the improved BULLS protocol remains the same as described in reference [1].

SELLS

The behavior of the SELLS protocol is similar to the BULLS protocol behavior described in Figure 1. SELLS differs from BULLS in the data structure used to store the share file listing (the files shared by each host). SELLS uses a Bloom filter to represent the list of shared files by each ultrapeer host and can be used to performed file look-ups in the Bloom filter to obtain the list of IP addresses or hosts from which the desired file can be downloaded.

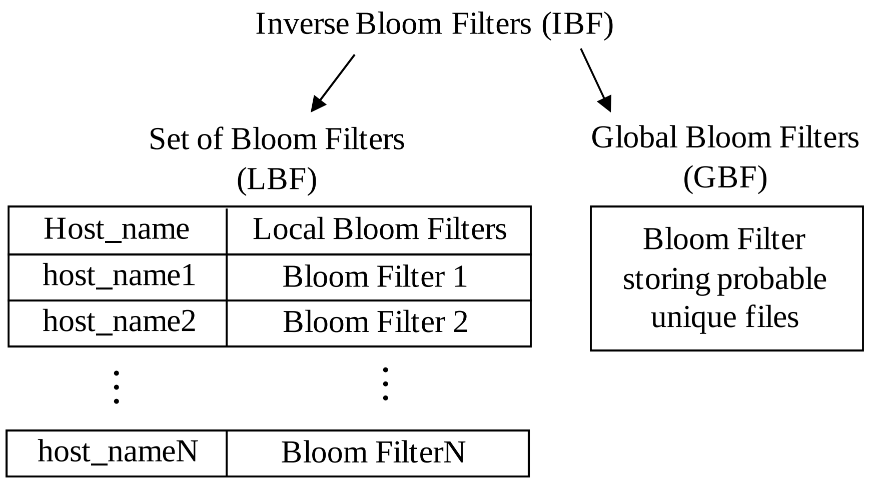
A Bloom filter is compact, randomized data structured for representing a set, and it supports membership queries (i.e., whether or not an element belongs to set S). A Bloom filter has the drawback of allowing a rate of false positives in membership queries [7, 8]; that is, a membership query incorrectly recognizes a non-element as a member of a set. A Bloom filter is a set $S = \{s_1, s_2, ..., s_n\}$ of n elements and an m-bit array that uses k independent hash functions $h_1, h_2, ..., h_k$, each with range between $\{1, ..., m\}$ where $m > n$ [7, 8]. Initially, the m-bit array has all bits set to zero. The i^{th} bit in the m-bit array is set to one if, and only if, there exists an element e in S and some j^{th} hash function such that $h_j(e) = i$. The m-bit array used for the Bloom filters may yield a false positive when determining membership of elements and can be used to determine when an element does not belong to a set. If an element v is not in set S, then there is at least one bit in the position in $\{h_1(v),...,h_k(v)\}$ of the m-bit array that is equal to zero. The probability of a false positive fp for an m-bit Bloom filter for a set of n elements and k independent hash functions is

$$fp = (1-p)^k \text{ where } p \approx e^{-kn/m} \text{ [7,8].} \qquad (1)$$

The rate of false positives can be controlled to a given tolerance by setting the parameters k, n, and m properly. There is a clear tradeoff between the size of the Bloom filter and the rate of false positives. A good practical estimate for k that can yield a wise and low false positive rate is

$$k = \ln 2(m/n) \text{ [7,8].} \qquad (2)$$

Thus, to design a Bloom filter, we only need to know the number of elements to be represented and the size of the Bloom filter.



Inverse Bloom Filters (IBF)
Set of Bloom Filters (LBF)
Global Bloom Filters (GBF)
Host_name
Local Bloom Filters
host_name1
Bloom Filter 1
host_name2
Bloom Filter 2
host_nameN
Bloom FilterN
Bloom Filter storing probable unique files

Figure 2. SELLS data structure

Bloom filters allow the data structure used by SELLS to be more space efficient than BULLS. SELLS uses a compact probabilistic representation or Bloom filter for all the files shared by each host, instead of representing each file as a separate item. Also, a global Bloom filter is used to filter searches with low probability of success. The SELLS' data structure is called an inverse Bloom filter (IBF). The two components that make the IBF are a set of local Bloom filters (LBF) and a global Bloom filter (GBF). The IBF used by SELLS is described in Figure 2 and explained in detail below:

- **LBF**: The first component shown in Figure 2 is a table with each row representing the data stored for a host in the network. The columns of the table in Figure 2 are the two basic types of data stored. The first column is the hostname, and it is used to identify a host in the network (IP address or host identification number). The second column is the Bloom filter representing the file share listing (set of filenames shared) of a host.

• **GBF**: The second component shown in Figure 2 is a Bloom filter constructed by applying a generalization of the logical operations (i.e., and, or, and xor). That is, these operations are applied to the Bloom filters stored in LBF. The operations will create a Bloom filter that will store the improbable files to be found in the network or the files that are not stored by any host in the network.

When a SELLS host inserts a file in the file shared list, the corresponding Bloom filter will be updated by setting the bits obtained from hashing the name of the file inserted. Contrary to a file being inserted in the file shared list, the deletion of a file causes the Bloom filter to be reconstructed. That is, each element in the file shared list is inserted in a new Bloom filter that will substitute the first one.

Additionally, because SELLS stores the shared file list as a single item instead of separate items, when an ultrapeer host enters the network, it behaves different from BULLS. That is, it forwards its own shared file list (one update message or one Bloom filter per shared file list) and the share file listing of the leaves (one update message or one Bloom filter shared file list) to ultrapeer neighbors (neighbor host that are ultrapeers) only and then transitions, as BULLS does, to the Idle state.

Flow Model

The flow models developed in this section result in expressions that are statistics for the storage requirement of the data structure of BULLS (S_{bulls}) and SELLS (S_{sells}) in bytes. Also, expressions for the overhead traffic per host in messages per second for Gnutella version 0.6 ($X_{gnutella}$), improved BULLS (X_{bulls}), and SELLS (X_{sells}) are developed. The flow model for Gnutella version 0.6 (Gnutella) is included as the baseline comparison for the overhead traffic.

BULLS and SELLS are both based on the same FSM shown in Figure 1. The difference in the overhead traffic between BULLS and SELLS is that BULLS stores a file shared list per host, while SELLS stores one Bloom filter per host and a single global Bloom filter. All flow models are developed as a function of the independent variables described in reference [1] and the independent variables (N_{lbf}) and (N_{gbf}) representing the size (bytes) of the Bloom filters in the data structure used by SELLS. All assumptions for the flow models not explained in this paper are assumed to be as in reference [1]. The five assumptions for this work are explained below:

1. The flow model for Gnutella and BULLS described in reference [1] defines the total number of files shared in the network to be $M_{files} N_{hosts}$, given that each host shares M_{files} files [1]. Thus, for the improved BULLS protocol and SELLS, it is assumed that at least M_{files} files are shared by each ultrapeer host. Also, it is assumed that N_{hosts} is the total

number of ultrapeer hosts in the network since ultrapeers are the hosts that generate the overhead traffic.

2. For simplicity, it is assumed that there are an equal number of leaves connected to each ultrapeer and that the total number of files shared by all leaves connected to an ultrapeer is M_{files}. The total number of files shared by all the leaves in the network is $M_{files} N_{hosts}$, as each group of leaf hosts connected to an ultrapeer shares the same number of files as the ultrapeer itself. This is a reasonable assumption given that the ultapeer capacity must at least be equal to the aggregated capacity of its leaves. The total number of files in the network is $2M_{files} N_{hosts}$.

3. It is also assumed that ultrapeers have a degree D (total number of ultrapeer hosts connected to an ultrapeer) and that the number of leaves connected to any ultrapeer is approximately $5D$ [2, 3, 6, 8].

4. The rate of file searches per host (ultrapeer or leaf), R_{search}, corresponds to the total file query search activity initiated by the user at a host (successful and unsuccessful searches). It is assumed that search activity is the same for a leaf and ultrapeer host. Search activity depends on the user and not on the host capability.

5. In the improved version of BULLS and SELLS, $2M_{files}$ messages are required to broadcast the entire shared file list of an ultrapeer (M_{files} files) and the entire shared file list of all the leaves (M_{files} files) of the ultrapeer. Additionally, the ultrapeers have the rate of updates as $R_{updates}$, but a leaf update rate is less than that of an ultrapeer. It is a requirement that ultrapeers have more bandwidth available than leaves [9]. Thus, it is reasonable to assume that leaves have half or less the rate of updates of an ultrapeer; that is $0.5R_{updates}$.

Storage Requirements of BULLS and SELLS

In the improved BULLS version described above, each host must store the data structure that contains all of the names of all files stored in the network by all hosts (ultrapeers and leaves). The total number of files shared by all the leaves in the network is $M_{files} N_{hosts}$, and the total number of files shared by all ultrapeers in the network is $M_{files} N_{hosts}$. Thus, the total number of files shared in the network is $2M_{files} N_{hosts}$. The size of this data structure (in bytes) for BULLS with ultrapeers or the improved BULLS is

$$S_{bulls} = N_{hosts} \left(N_{hostname} + 2M_{files} N_{filename} \right). \tag{3}$$

The first term is the number of bytes required to store all the *hostnames*. The second term is the total number of bytes necessary to store the *filenames* of all the files shared by each host.

For SELLS, the size of this data structure (in bytes) is:

$$S_{sells} = N_{hosts}\left(N_{hostname} + N_{lbf}\right) + N_{gbf}\ . \tag{4}$$

The first term is the number of bytes required to store the *hostnames*. The second term is the total number of bytes used to store the set of *filenames* shared by each host. Because each set of *filenames* shared by a host is represented by a Bloom filter of size N_{lbf} bytes, the total number of bytes necessary to store all the *filenames* shared by all hosts in the network is $N_{hosts}N_{lbf}$. The third term is the total number of bytes (N_{gbf}) of the Bloom filter (GBF), representing the improbable files to be found in the network.

Overhead Traffic of Gnutella, BULLS, and SELLS

The overhead message rate for Gnutella, BULLS, and SELLS is based on the overhead traffic equations described in reference [1]. The overhead message rate per ultrapeer host for Gnutella is:

$$X_{gnutella} = R_{search}D\left(N_{hosts} - 1\right) + R_{search}DN_{hosts} + R_{search}N_{hops}P\left(N_{hosts} - 1\right). \tag{5}$$

The first term is the rate of query messages seen by each ultrapeer host and initiated by an ultrapeer host. Each ultrapeer host receives D copies of each query message sent by every other ultrapeer host. The second term is the rate of query messages seen by each ultrapeer host and initiated by a leaf host. Each ultrapeer receives D copies of each query message. The third term is an approximation of the rate of queryhit response messages seen by each ultrapeer host. Queryhit messages are returned via the backward path a query message was received; thus, each queryhit message travels on average N_{hops} and is received by N_{hosts} ultrapeer hosts.

The overhead message rate per ultrapeer host for BULLS is

$$X_{bulls} = 0.5R_{update}D(1.5N_{hosts} - 1) + D\left(N_{hosts} / T_{stay}\right)\left(2M_{files} + 1\right). \tag{6}$$

The first term is the rate of flooded directory update messages seen by each ultrapeer host as a result of ultrapeer hosts adding or deleting a shared file. The second term is the rate of flooded directory update messages seen by each ultrapeer host as a result of leaf hosts adding or deleting a shared file. When all searches are successful (i.e., a file is found) and files are not otherwise added or deleted to a host (ultrapeer or leaf), $R_{updates}$ will be the same as R_{search} . The third term is the rate of flooded update messages seen by each ultrapeer host as a result of hosts entering the network (flooding their entire listing of shared files and share files list of its leaves to all ultrapeer hosts) and from depart messages from departing ultrapeer hosts. It is assumed, as in reference [1], that the rate in which hosts enter and depart the network is the same. For BULLS, the total number of update messages is the same as the total number of files shared in the network ($2M_{files}$) plus the depart message. Thus, the overhead message rate per ultrapeer host for SELLS is

$$X_{sells} = 0.5R_{update}D(1.5N_{hosts} - 1) + D(N_{hosts} / T_{stay})(5D + 3). \qquad (7)$$

The first term and second term, as in BULLS, corresponds to the update messages from an ultrapeer host or leaf adding or deleting a shared file. The third term in SELLS, as in BULLS, is the rate of flooded updated messages resulting from a host entering or departing the network. SELLS, in contrast to BULLS, does not flood the network with the entire listing of shared files and share files list of its leaves. SELLS floods the network with one update message representing the complete listing of shared files of a host. Thus, the total number of update messages is the total number of shared file lists by an ultrapeer ($5D + 1$) and one update message for the GBF. Also, as in BULLS, we must add one additional update message for the depart message.

Performance Evaluation

The performance evaluation is based on the flow models previously described and the representative case explained in reference [1]. The numerical values selected for the overhead traffic variables of Gnutella, BULLS, and SELLS are the same values used reference [1]. The overhead traffic rate is studied as a function of the rate of hosts entering (and leaving) the network for the flow models. The variables M_{files}, P, D, N_{hosts}, R_{search}, and R_{update} are fixed, and T_{stay} (amount of time a host stays connected to the network) is varied from a couple of hours to days.

The numerical results in Figure 3 demonstrate the difference between Gnutella, BULLS, and SELLS overhead traffic as a function of T_{stay}. Figure 3 shows that SELLS has a lower overhead traffic rate than Gnutella and BULLS. From Figure 3, it can be determined that:

- SELLS reduces Gnutella's overhead traffic by a minimum of 6 percent and a maximum of 42 percent.
- SELLS reduces BULLS' overhead traffic by a minimum of 5 percent and a maximum of 66 percent.

The numerical values for the storage requirements of BULLS and SELLS are based on the values in reference [1]. We haves selected values for the size of the Bloom filters that yield an acceptable false positive rate (i.e., less than 0.1 percent), and we have selected to use four hash functions. The size in bytes of the bloom filters are:

- N_{lbf} = 512 bytes or 4096 bits (assuming that the average host stores about 100 files)

  and the false positive rate is 7.50 x 10^{-5}, which is much less than 0.1 percent.

- N_{gbf} = 1 Mb or 8,388,608 bits (assuming the total number of files share is less than 8 million files) and the false positive rate is 5.1 x 10^{-20}, which can be considered negligible.

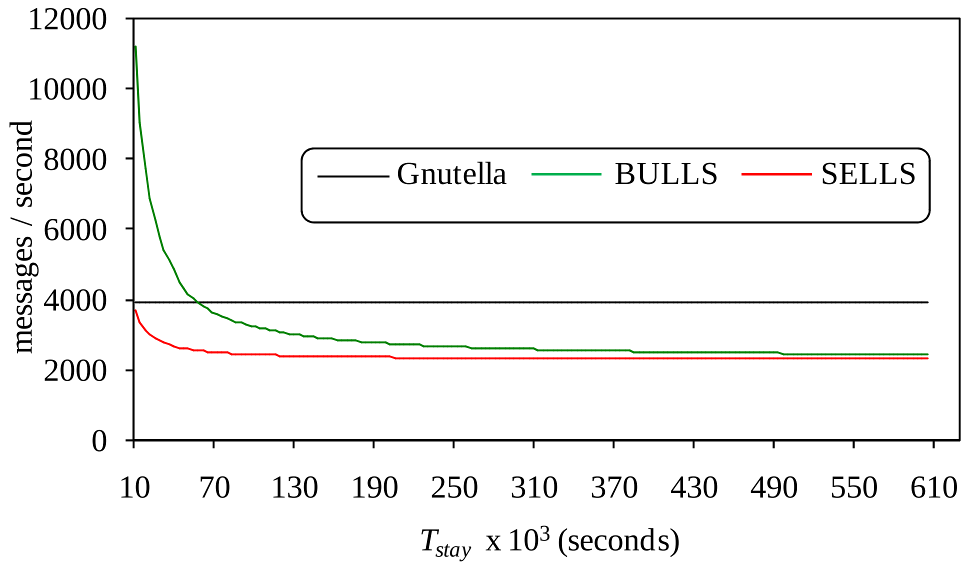


Figure 3. Impact of T_{stay} in overhead traffic

The numerical results for the representative values used in reference [1] with T_{stay} = 8 hours and equations 3, 4, 5, 6 , and 7 are:

- $X_{gnutella}$ = 3908 messages/second
- S_{bulls} = 7.83 x 10^8 bytes
- X_{bulls} = 5755 messages/second
- S_{sells} = 0.42 x 10^8 bytes
- X_{sells} = 2793 messages/second

The data structure size for BULLS and SELLS corresponds to 747 MB and 40 MB, respectively. Given that memory stick sizes are usually 1 GB or larger, the SELLS storage requirement can easily be satisfied at a cost of less than $40. The message rate corresponds to less than 200 Kb/sec, 350 Kb/sec, and 150 Kb/sec for Gnutella, BULLS, and SELLS, respectively, which is reasonable for broadband connections with a data rate of several

Mb/sec. If $T_{stay} = 8$ hours, then the SELLS overhead traffic rate is 34 percent less than Gnutella and 39 percent less than BULLS. In addition, SELLS reduces BULLS storage requirement by 94 percent at the expense of a false positive rate less than 0.1 percent.

Trustworthy Key Distribution Using P2P networks

P2P networks lack security and protocols to establish trust between peers. Establishing trustworthy key distribution with SELLS requires that each LBF store the trustworthy keys exchanged by a host in the network and that the GBF stores the improbable trustworthy keys. Thus, a host will accept a secure trusted connection if the key is found in at least one LBF and is not found in the GBF. SELLS assumes the existence of a host (Trusty) that has previously securely exchanged secret keys with other hosts (seed hosts). It is assumed that seed hosts have previously exchanged secret keys using a public key exchange protocol. Thus, connecting to the network for the first time has two steps: 1) connect to Trusty and download the GBF and LBF, and 2) establish a trusted communication with the host using the LBF from Trusty.

The feasibility of the key exchange between Trusty and other hosts can be achieved using the Diffie-Hellman protocol or any other public key exchange protocol [10, 11].

Conclusions

SELLS is a new space efficient protocol based on the BULLS protocol. SELLS, like BULLS, reverses the query broadcast paradigm by building a local data structure and enabling local look-up search in a host (i.e., without a broadcast query). The SELLS data structure is more space efficient than the BULLS structure due to the compact representation of the shared file listing using Bloom filters. Furthermore, today's information technology (IT) systems require protocols, like BULLS, that are efficient and can be used to establish trusted communications. In particular, SELLS can be used in IT systems that communicate over the Internet and need to exchange secret keys with geographically disperse/remote users. This allows users to establish a trustworthy communication that will decrease security risks.

The SELLS protocol was designed and evaluated using flow models. Numerical results based on a representative case show that SELLS needs 94 percent less storage than BULLS and that SELLS has less overhead traffic when compared to Gnutella and BULLS.

For a representative case, it was shown that SELLS reduces Gnutella's overhead traffic by a minimum of 6 percent and a maximum of 42 percent. Also, SELLS reduces BULLS' overhead traffic by a maximum of 66 percent and a minimum of 5 percent. The SELLS protocol was also designed with the goal of enabling a novel application for trustworthy key distribution.

References

[1] Perera, G., Christensen, K., and Roginsky, A. “Broadcast Updates with Local Look-up Search (BULLS): A New Peer-to-Peer Protocol,” *Proceedings of the ACM Southeast Conference*, (2006), 22(6):124–129.

[2] Androutsellis-Theotokis, S. and Spinellis, D. “A Survey of Peer-To-Peer Content Distribution Technologies,” *ACM Computing Surveys*, (December 2004), 36(4):335–371.

[3] Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., and Faloutsos, M. “Is P2P Dying or Just Hiding?” *Proceedings of GLOBECOM*, (December 2004), 1532–1538.

[4] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. “Search and Replication in Unstructured Peer-to-Peer Networks,” *Proceedings of the 16th International Conference on Supercomputing*, (June 2002), 84–95.

[5] Saroiu, S., Gummadi, P., and Gribble, S. “A Measurement Study of Peer-to-Peer File Sharing Systems,” *Proceedings of SPIE in Multimedia Computing and Networking*, (January 2002), 4673(1):156–170.

[6] Subhabrata, S. and Wang, J. “Analyzing Peer-To-Peer Traffic Across Large Networks,” *IEEE/ACM Transactions on Networking*, (April 2004), 12(2):137–150.

[7] Bloom, B. “Space/time Trade-offs in Hash Coding with Allowable Errors,” *Communications of the ACM*, (1970), 13(7):422–426.

[8] Mullin, J.K. “A Second Look at Bloom filters,” *Communications of the ACM*, (1983), 26(8):570–571.

[9] Klingberg, T. and Manfredi, R. “Gnutella Draft Specification v0.6,” (June 2002), http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html.

[10] Diffie, W. and Hellman, M. “New Directions in Cryptography, IEEE Transactions on Information Theory,” (1976), 644–654.

[11] Mao, W. *Modern Cryptography Theory and Practice*. Prentice Hall. (2004).

Biography

GRACIELA PERERA is currently an Assistant Professor in the Computer Science and Information Systems department at Youngstown State University. She received her Ph.D. in Computer Science and Engineering from the University of South Florida in August 2007. She has recently published a book on new search paradigms and power management for peer-to-peer file sharing, published by VDM Verlag. Her research interest is the design and performance evaluation of distributed search methods. She is currently focusing on the applications of peer-to-peer network protocols and security.

ROBERT KRAMER is currently an Associate Professor in the Computer Science and Information Systems department at Youngstown State University. He is interested in fast polynomial arithmetic and experimenting with Lego robotics. Dr. Kramer has more than 10 years of teaching experience and was the leader of the grant application from NSF’s STEM Scholars program that was successfully funded for $600,000.

ANTHONY WEAVER is currently a student at Youngstown State University, majoring in computer information systems. His research interests include network performance, security, and protocol design. Mr. Weaver is currently working on application design for network traffic analysis.