ISP-aided Biased Query Search for P2P Systems in a Testlab

Vinay Aggarwal, Anja Feldmann

2007
ISP-aided Biased Query Search for P2P systems in a Testlab

Vinay Aggarwal and Anja Feldmann

Deutsche Telekom Laboratories / TU Berlin
{vinay.aggarwal, anja.feldmann}@telekom.de

Summary. More than half of Internet traffic today is contributed by peer-to-peer (P2P) systems. P2P systems build their overlay topology largely agnostic of the Internet underlay, which often leads to traffic management challenges for Internet Service Providers (ISP) and potentially inefficient neighborhood selection for P2P nodes. To overcome this, we propose to use an oracle hosted by the ISPs, so that ISPs and P2P users can cooperate for improved performance. The oracle can be queried by P2P nodes while choosing neighbors for content search, and it will rank the possible neighbors of the querying node according to a locality indication, like the AS-hop distance. The ISP gains by keeping traffic within its Autonomous System (AS, and the P2P node can experience improved performance like lesser delay and better bandwidth.

In this paper, we evaluate the benefits of our scheme by performing experiments in a real Testlab consisting of routers, switches and computers running actual instances of P2P applications. We show how we configure representative AS topologies for P2P networks using VLANs and trunking ports, and present experimental results with content search phase of a P2P network using different file sharing and search query distributions.

1 Motivation

A significant portion of the Internet traffic today is contributed by P2P systems, which are realized as overlays on top of the underlying Internet routing architecture. Some of the most popular applications that rely on P2P systems [2, 5] include file sharing systems like BitTorrent, Gnutella and eDonkey, VoIP systems such as GoogleTalk and Skype, and also real-time streaming systems such as MySpace and YouTube. Recent measurement studies have reported that P2P systems account for more than 50% of total Internet traffic [3, 4, 5].

Indeed, P2P system applications are one of the major reasons cited by users for upgrading their Internet access to broadband [6]. Yet, P2P traffic also poses a problem for the ISPs as it is very dominant and the traffic flows
are almost impossible to engineer for them [3, 4]. This is primarily because ISPs have no control over the overlay structure of P2P systems, which is formed largely independent of the Internet routing underlay.

As P2P systems often choose their neighbors without any respect for network locality, the traffic may have to cross network boundaries multiple times [4, 7], even though the content is often served from within the ISP’s network. Studies have shown that due to content language and geographical regions of interest, the desired content is often available “in the proximity” of interested users [8].

Let us consider how unstructured P2P file-sharing networks tend to maintain their topologies. New P2P nodes usually retrieve a list of members of the P2P network either via a well known Web page, a configuration file, or some history mechanism [2, 5]. They then pick some subset of these as possible neighbors either randomly [10] or based on some degree of performance measurement [11].

We, in [1], propose that each ISP offers a service, called the oracle, to its P2P users. When the P2P user wishes to connect to the overlay network, it supplies the oracle with a list of possible P2P neighbors. The oracle ranks them according to their proximity to the querying node. This ranking can then be used by the P2P node to select a nearby neighbor which improves the performance of the P2P system. The ISP can use this mechanism to steer P2P traffic, for example, to express preference for P2P neighbors that are within its network. The simplest version of such an oracle is a server that ranks IP addresses according to whether they are hosted by its ISP or not. The next better version may use AS-hop distance, which is defined as the number of AS-hops on the chosen BGP [9] route for the IP address, or take BGP policies into account. In addition, one may use intra-domain routing information to keep P2P traffic even more localized.

The benefits to P2P nodes of using the oracle are: (1) they do not have to measure the path performance themselves, as in [11]; (2) they can take advantage of the knowledge of the ISP; (3) they can expect improved performance in the sense of low latency and high throughput as bottlenecks at inter-ISP transit/peering links [3] can be avoided.

The ISPs benefit as they can influence the neighborhood selection process of the P2P network to, e.g., ensure locality of traffic flows and therefore again have the ability to manage the flow of their traffic. They also gain cost advantages, by reducing costs for traffic that leaves their network [3, 4].

While there are some proposals that aim to localize P2P traffic, e.g., [11, 12, 13], our solution is much simpler and more general. Besides being applicable to P2P nodes of all overlays, it promotes the idea of ISPs and P2P users actively co-operating such that both benefit, as discussed in detail in [1].

To evaluate the impact of using the oracle one should ideally study P2P systems with many nodes over the Internet, a network with many ASes and complex intra-AS topologies. Yet as the oracle is not yet offered by ISPs, it is not trivial to perform experiments on an Internet-wide scale. Hence, we have
to make use of Testlab facilities and simulators for experiments. Simulations on the graph properties of oracle-influenced P2P overlays in [1] reveal that biased overlay graphs maintain a small diameter, small mean path length and node degree, while the densely connected subgraphs are now local to the ISPs. This implies that when P2P nodes consult an oracle while choosing their neighboring nodes, they are able to keep a large number of their peerings within the AS, thus resulting in improved performance and lesser bottlenecks, while at the same time, do not sacrifice any of the nice graph properties of typical random graphs.

While simulations on P2P networks and overlay-underlay graphs in a simulation framework (as in [1]) allow us to experiment with large-sized graphs, containing thousands of P2P nodes, we still have to model the P2P networks and the routing protocols. Hence, in this paper, we use a Testlab facility to perform the P2P experiments, so that we can run the actual P2P system code, and validate and refine our network models. The advantage of using a Testlab is that we can experiment with real traffic instead of simulated flows, and can configure the network devices like routers, switches and links to generate variegated network scenarios and traffic environments. We get control over the network entities, which enables us to perform a wide range of experiments using real applications, network stacks and operating systems. Also, we have better control and visibility over the test environment as compared to running the experiments on the Internet, and can additionally eliminate the risk of inadvertently affecting the proper functioning of the Internet due to traffic generated by our experiments. Debugging and developing new applications hence becomes more feasible.

The paper is organized as follows. In Section 2, we introduce the hardware setup of our Testlab. We then explain how we configure various network topologies in the Testlab using routers, VLANs and other resources in Section 3. The experiments are performed with the Gnutella P2P file sharing protocol, which is briefly reviewed in Section 4. The actual experiments and results are detailed in Section 5, followed by a conclusion and outlook on future work in Section 6.

2 Introduction to the Testlab

We have built a Testlab, which is a collection of real devices - routers, switches and computing machines, that provides us the facility to perform experiments with real traffic in a realistic environment.

2.1 Hardware setup

The hardware setup of the Testlab consists of the following devices:

- three Cisco 2691XM routers, named c1, c2, c3
• three Juniper M7i routers, named j1, j2, j3
• a Cisco 3750G24-TS switch, named c4
• two Cisco 2950SX-24 switches, named c5, c6
• a Cisco 3500XL switch, named j4
• a Cisco 3550-12G switch, named j5
• a Cisco 2950 switch, named j6
• nine Opteron-based load generator PCs with 4GB RAM, named loadgen101 to loadgen109
• 13 Athlon-based load generator PCs, named loadgen201 to loadgen213

Each router is equipped with two GigaBit Ethernet interfaces, one connected to an inbound or backbone switch, the other one to an outer switch. The Cisco and Juniper routers are organized in two groups or clouds which are interconnected over the two inbound switches c4 and j4. All the PCs run Linux. A graphical representation of the setup is shown in Figure 1.

Fig. 1. Hardware layout of Testlab

2.2 Configuration Management

All routers, switches and computers are connected via a management network and administrative access to them is provided through a separate channel via ssh. Different topologies can be configured with the help of a centralized control server. A program stores the configuration status of the hardware in a backend database, which can be retrieved and restored to the routers using a command line interface. In other words, we can design and setup different topologies on the Testlab hardware, in order to perform realistic experiments of multiple different scenarios.
2.3 Physical Network Layout

As shown in Figure 1, the network is divided into four clouds: one cloud containing all Cisco routers, another cloud containing all Juniper routers, and two clouds of load-generators. The Cisco cloud and the Juniper cloud are connected to each other by the switches c4 and j4 and to the two load generator clouds by the switches c5, j5, c6 and j6. Host-to-host connections can be set up either directly, using just a switch, or by using routed network links using one or more routers.

3 Configuration of Topologies in Testlab

To decide what kind of topologies we wish to configure in the Testlab, we need to consider how routing works in the Internet and in P2P systems. In the Internet, which is a collection of Autonomous Systems (AS), packets are forwarded along a path on a per-prefix basis. This choice of path via the routing system is limited by the contractual agreements between ASes [9] and the routing policy within the AS (usually shortest path routing based on a fixed per link cost). P2P systems, on the other hand, setup an overlay topology and implement their own routing in the overlay topology which is no longer done on a per-prefix basis but rather on a query or key basis [5]. In unstructured P2P networks queries are disseminated, e.g., via flooding or random walks while structured P2P networks often use DHT-based routing systems to locate data. Answers can either be sent directly using the underlay routing or through the overlay network by retracing the query path.

Accordingly, we devise topologies with multiple ASes, where each AS hosts multiple machines running the P2P applications. As a router can be taken as an abstraction of an AS boundary, and we have 5 routers available, we decide to form 5-AS topologies. Each router connects to 3 load-generators, and we were able to run 3 instances of P2P applications on each machine concurrently. This gives us an upper bound of 5-AS topologies, with 15 machines, running 45 P2P clients concurrently. We connect the 5 routers in different fashions as shown in Figure 2, to arrive at 4 different AS topologies, which we call realistic, ring, star, and tree topologies. We now explain in detail, how we configured the hardware to achieve the desired topologies. We hope that this discussion will help other researchers who face similar challenges while configuring Testlab devices.

3.1 Router Configuration

As a first step, we define a subnet layout structure using the following scheme for assigning IP addresses to interfaces:

- Router-to-router connection (inter-AS traffic): subnets 10.0.n.0/24, where n has a numerical value from 1 to 3.
• Router-to-loadgenerator connections (intra-AS traffic): subnets 10.1.n.0/24, where n has a numerical value as 14, 24, 34, 44 and 54.

Since each router has only two interfaces, one for router-to-router connections and the other for outbound connections, we have to assign multiple IP addresses to each router interface to create more than one router-to-router connections on a router. At first, the idea of defining multiple IP addresses on the specific interfaces came up, but this could not be easily configured on our devices, so we decided to use IEEE 802.1Q VLANs for creating our subnet structure. Both Cisco and Juniper routers provide the opportunity to use VLAN tagging and virtual interfaces connected to VLANs.

IEEE 802.1Q VLANs

Virtual LAN, commonly known as VLAN [14], is a group of devices on one or more LANs that are configured so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical instead of physical connections, they are very flexible for user/host management, bandwidth allocation and resource optimization. By using VLAN capable hardware devices, it is possible to define more than one Ethernet segments on a port-by-port basis without changing the hardware setup. In the Testlab we use the widely used IEEE 802.1Q [15] standard.

The IEEE 802.1Q specification establishes a standard method for tagging Ethernet frames with VLAN membership information [16]. This standard defines the operation of VLAN Bridges that permit the definition, operation and administration of Virtual LAN topologies within a Bridged LAN infrastructure. The 802.1Q standard is intended to address the problem of how to break large networks into smaller parts so that broadcast and multicast traffic do not use more bandwidth than necessary. The standard also provides a higher level of security between segments of internal networks.

The key for 802.1Q to perform the above functions is in its tags [20]. 802.1Q-compliant switch ports can be configured to transmit tagged or untagged frames. A tag field containing VLAN (and/or 802.1p priority) information can be inserted into an Ethernet frame. If a port has an 802.1Q-compliant device attached (such as another switch), these tagged frames can carry VLAN
membership information between switches, thus letting a VLAN span multiple switches. However, it is important to ensure that ports attached to non-802.1Q-compliant devices are configured to only transmit untagged frames.

Configuration of VLANs

Both Cisco and Juniper vendors give us the opportunity to use IEEE 802.1Q VLAN tagging and more importantly, virtual interface connectivity to specified VLANs. Based on the previously introduced subnet design, a VLAN ID for every router-to-router subnet was created, using a number from 100 to 105. Since the switches c4 and j4 do not have the information which connection belongs to which VLAN, this correlation is defined port-by-port on the switches. This means that each port transporting one or more VLANs from virtual interfaces defined on the physical interfaces has to be enabled to pass this VLAN-related traffic on to other ports. These specifically configured ports transporting VLAN tagged traffic are referred to as trunking ports [21].

The configuration, as described above, gave rise to a problem with VLAN based connections from and to routers. It was related to the VLAN Trunking Protocol (VTP) default settings of the Cisco devices.

VLAN Trunking Protocol

VLAN Trunking Protocol, commonly known as VTP [17], is a Cisco Layer 2 messaging protocol that manages the addition, deletion and renaming of VLANs on a network-wide basis. VLAN Trunk Protocol (VTP) reduces administration in a switched network. When we configure a new VLAN on one VTP enabled router/switch, the VLAN configuration is propagated through all the switches in the domain. This reduces the need to configure the same VLAN everywhere. VTP is a Cisco-proprietary protocol, so it is used only by the Cisco routers.

The described network problems occurred because by default VTP seems to be enabled on all Cisco routers, so VTP messages from the switches and routers collided. To fix this problem, we disabled VTP on all devices except on c4 (which is supposed to work as VTP master). After defining a VTP domain, setting all missing Cisco devices to client mode and configuring the VLAN database [21], all the connections finally worked as desired.

3.2 Configuration of Computing Machines

At this stage the only requirement was to establish router-to-loadgenerator connections, which was achieved by using the commands:

- `ifconfig`: for defining IP addresses on a particular ethernet interface
- `route`: for setting up the gateway of routes.
This configuration is done manually by accessing all needed loadgenerators using an ssh console.

At this stage, we have configured the four different AS topologies shown in Figure 2 such that router-to-router connection is established by VLAN interfaces and each router is connected with 3 loadgenerators, which amounts to a total of 15 load-generators. The final configuration of the Testlab devices is shown in Figure 3. The configuration details of various topologies can be found in [20, 21].

Fig. 3. Configuration of Testlab devices to achieve the AS topologies: realistic, ring, star and tree
4 Introduction to Gnutella

Having successfully configured a multiple-AS topology in the Testlab, we needed to decide what P2P application to use for our experiments. Based on a number of considerations, we chose Gnutella [10]. Gnutella is one of the three most popular P2P file sharing networks in the Internet today, with about 2 million active users [2, 8]. Besides, it is an open-source protocol, which has been well researched in the networking community, e.g. [7, 8, 19]. Hence, its characteristics are well understood, and the existing literature allows us to compare and contrast our experimental results with established behaviour patterns of Gnutella, both in simulation frameworks as well as in the real Internet.

The Gnutella network is comprised of agents called servents, who can initiate as well as serve requests for resources. When launched, a servant searches for other peers to connect to by sending Hello-like Ping messages. The Pings are answered by Pong messages, which contain address and shared resource information. Search queries are flooded within the Gnutella network using Query messages, and answered by QueryHits. To limit flooding Gnutella uses TTL (time to live) and message IDs. Each answer message (QueryHit/Pong) traverses the reverse path of the corresponding trigger message. When a node receives multiple QueryHit messages for its search query, it selects one of the nodes randomly, and initiates a direct file download from this node using HTTP. Hence, while the negotiation traffic is carried within the set of connected Gnutella nodes, the actual data exchange of resources takes place outside the Gnutella network, using the HTTP protocol. Due to scalability problems, new versions of Gnutella take advantage of a hierarchical design in which some servents are elevated to ultrapeers, while others become leaf nodes. Each leaf node connects to a small number of ultrapeers, while each ultrapeer maintains a large number of neighbors, both ultrapeers and leaves. To further improve performance and to discourage abuse, the Ping/Pong protocol underwent semantic changes. Answers to Pings are cached (Pong caching) and too frequent Pings or repeated Queries may cause termination of connection.

5 Testlab Experiments

We began by installing Gnutella P2P software on each machine. To be able to install multiple Gnutella servents on each machine, we used the C-based, no-frills GTK-Gnutella [18] software. By installing three servents each on 15 machines, we had 45 Gnutella servents in our experiments. We assigned one servent on each machine to be an ultrapeer, while the other two were made leaf nodes. Thus, we had 15 ultrapeers, and 30 leaf nodes.

A central machine which was connected to all the other load-generators is used to run the oracle. When a Gnutella servent sends a list of IP addresses to the oracle, the oracle sorts this list in the order of, first, servents within the
querying servent AS, followed by servants in the AS which is 1 AS-hop away, followed by servants at increasing AS-hop distance.

Experiments with a simulation framework in [1] reveal that consulting the oracle for neighborhood selection, during bootstrapping stage as well as file-exchange stage, leads to significant increase in localization of P2P traffic. To explore the aspect of content search and exchange using an oracle in an actual Testlab with real P2P traffic, we devise the following scheme.

We first run an experiment with the unmodified Gnutella protocol running on each servent, which does not consult the oracle for neighborhood selection. We then run another experiment, where each servent (both ultrapeer and leaf node) consults the oracle. To concentrate on content search and exchange, we let each servent communicate with the oracle and send the Query search messages initially to only those neighbors which are within its AS. If the search is unsuccessful, i.e. the servent does not receive a QueryHit within a predefined time interval, the servent sends the Query message to neighbors which belong to ASes that are 1 AS-hop away. If the Query is still unsuccessful, it is sent to all the remaining neighbors in the network. Hence, a Gnutella servent consults the oracle actively during the content search phase, but not during the bootstrapping phase. In contrast, servants in unmodified Gnutella flood the Query messages to all their connected neighbors.

Since the file sharing pattern of P2P users can impact the content search experiment results, we employ two file sharing schemes:

- **Uniform**: every servent (both ultrapeer and leaf node) shares 6 unique files, leading to a total of 270 files in the Testlab.
- **Variable**: all ultrapeers share 12 files, half the leaf nodes share 6 files each, and the remaining leaf nodes share no files. The content of files within any

<table>
<thead>
<tr>
<th>Topology</th>
<th>Unmodified Gnutella</th>
<th>Biased Gnutella</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realistic</td>
<td>6604</td>
<td>2473</td>
</tr>
<tr>
<td>Ring</td>
<td>6623</td>
<td>2512</td>
</tr>
<tr>
<td>Star</td>
<td>6679</td>
<td>2533</td>
</tr>
<tr>
<td>Tree</td>
<td>6643</td>
<td>2468</td>
</tr>
</tbody>
</table>

Table 1. Total number of Query messages that are relayed in the network, using Uniform File Sharing

<table>
<thead>
<tr>
<th>Topology</th>
<th>Unmodified Gnutella</th>
<th>Biased Gnutella</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realistic</td>
<td>10194</td>
<td>4873</td>
</tr>
<tr>
<td>Ring</td>
<td>10939</td>
<td>4834</td>
</tr>
<tr>
<td>Star</td>
<td>10902</td>
<td>4863</td>
</tr>
<tr>
<td>Tree</td>
<td>10872</td>
<td>4847</td>
</tr>
</tbody>
</table>

Table 2. Total number of Query messages that are relayed in the network, using Variable File Sharing
Fig. 4. Plots showing the number of QueryHit messages (y-axis) for each Gnutella node (x-axis) in the four topologies, for the case of uniform file sharing. The solid lines denote unmodified Gnutella, while the dotted lines denote the oracle-influenced Gnutella.

AS is kept the same as in uniform scheme, i.e., only the files of one leaf node are moved to its ultrapeer.

The aim of the experiments is to compare the impact of the oracle on the content search process of P2P systems. More specifically, we wish to compare the number of QueryHit messages received by each servant with and without consulting the oracle, for uniform and variable file sharing schemes. We let each servant introduce a unique search query string in the network.

First, we measure the number of Query messages that are relayed in the entire Testlab network, and present the results in Tables 1 and 2. There are only 45 unique Query strings in both cases, but when a Query message is forwarded by a servant to its n neighbors, it is counted n times. This helps us to quantify the impact of biased neighbor selection on the scalability of the Gnutella network.

We see that consulting the oracle during content search reduces the number of Query messages that are relayed in the network, for both uniform and
variable file sharing. We see a higher number of messages in variable file sharing, which is due to the fact that a Query often arrives at a servent which is not sharing any content, and is hence further forwarded to this servent’s neighbors, thus generating more negotiation traffic. But even with variable file sharing, forwarding the Query messages with the help of ISP-hosted oracle to nearest neighbors reduces the negotiation traffic by at least 50%. As negotiation traffic for content search forms a significant portion of P2P traffic [19], we can conclude that consulting the oracle significantly improves the scalability of such P2P networks.

We now measure the number of QueryHit messages received by each Gnutella servent, for the unique query string that it introduces in the network. We compare the number of responses received in unmodified Gnutella experiments with that of oracle-influenced Gnutella experiments. Figure 4

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.png}
\caption{Plots showing the number of QueryHit messages (y-axis) for each Gnutella node (x-axis) in the four topologies, for the case of variable file sharing. The solid lines denote unmodified Gnutella, while the dotted lines denote the oracle-influenced Gnutella protocol. The query strings used in these experiments are identical to those used in Figure 4.}
\end{figure}
Fig. 6. Plots showing the number of QueryHit messages (y-axis) for each Gnutella node (x-axis) in the four topologies, for the case of variable file sharing. The solid lines denote unmodified Gnutella, while the dotted lines denote the oracle-influenced Gnutella protocol. The query strings used are much shorter, and have a much lower chance of successful content search as compared to queries in Figure 4.

shows the results for uniform file sharing, while Figure 5 demonstrates the case for variable file sharing.

We see that while consulting the oracle during content search often reduces the number of QueryHit messages received by a servant, the difference is only nominal. Importantly, we did not find a single case where a Query yields a result in unmodified Gnutella, but fails to do so while consulting the oracle.

As the pattern and quality of query strings can also affect results [19], we make another set of experiments, this time by changing the set of query strings in variable file sharing. Here queries have a much lesser chance of finding file content, i.e. they are unlikely to yield a QueryHit. This helps to detect cases of servants which may get very few QueryHits with unmodified Gnutella, but may fail to yield any QueryHits at all when consulting the oracle. The results are shown in Figure 6.

We again see only a nominal reduction in the number of QueryHit messages for oracle-influenced Gnutella servants. Besides, we detect only 2 servants (both leaf nodes) in a total of 45, which did not receive any QueryHit when using the oracle, while they received 1 and 2 QueryHits respectively with the unmodified Gnutella.

Hence, we can conclude that consulting the oracle does not adversely affect the content search process of P2P networks. P2P nodes are easily able to search and share content, while the scalability of the P2P system improves
considerably. The overall load on the network is reduced by at least 50%, while the content search performance remains comparable.

Experiments performed in a simulation framework [1] using 1000 Gnutella nodes in 25 ASes have already shown that the benefits to the P2P system and the Internet underlay remain unchanged even in the face of heavy P2P churn, as well as larger overlay and underlay topologies. The P2P system continues to behave as per its protocol, with users able to locate and share content, and the bottleneck links at AS access points continuing to route traffic without packet loss and congestion.

6 Conclusion and Future Work

To evaluate the concept of P2P nodes consulting an ISP-hosted oracle for neighborhood selection, we perform experiments in a real Testlab. During the content search phase, each P2P node consults the oracle and sends queries only to the nearest neighbors. We find that consulting the oracle reduces the negotiation traffic in the network by at least 50%, and improves the scalability of the P2P network considerably. While there is a nominal decrease in the number of response messages to search queries, this does not adversely affect the search process, as all queries are still able to find content. Even with queries having a very low chance of success, cases of nodes unable to locate content due to consulting the oracle are very rare, if at all.

In the next step, we plan to add appropriate delays to network devices, and assign bandwidths to links, to be able to have even more representative network environments. Besides, we are experimenting with the oracle scheme in Planetlab to increase the scale of our experiments and to test the interaction of modified P2P clients with unmodified ones. We have realized the oracle as a Web server, and are in the process of installing Gnutella and BitTorrent clients on Planetlab nodes, which will consult the oracle while making neighborhood selection decisions.

Acknowledgements

This work is partially supported by EU FP6 DELIS project. Thanks are due to Ruediger Wolf-Sebottendorf and Shailesh Gupta, for helping us with setting up the Testlab and performing initial experiments. We are grateful to the technical support staff at TU Munich, for providing us access to the Testlab setup. Our thanks also go to the anonymous reviewers for their valuable feedback.

References

ISP-aided Biased Query Search for P2P systems in a Testlab

10. Gnutella v0.6 RFC. http://www.the-gdf.org