Planet Scale Software Updates

Christos Gkantsidis[1], Thomas Karagiannis[2], Pablo Rodriguez[1], and Milan Vojnović[1]
[1] Microsoft Research
Cambridge, UK
{chrisgk,pablo,milanv}@microsoft.com
[2] UC Riverside
Riverside, CA, USA
tkarag@cs.ucr.edu

ABSTRACT

Fast and effective distribution of software updates (a.k.a. patches) to millions of Internet users has evolved into a critical task over the last years. In this paper, we characterize "Windows Update", one of the largest update services in the world, with the aim to draw general guidelines on how to best design and architect a fast and effective planet-scale patch dissemination system. To this end, we analyze an extensive set of data traces collected over the period of a year, consisting of billions of queries from over 300 million computers. Based on empirical observations and analytical results, we identify interesting properties of today's update traffic and user behavior.

Building on this analysis, we consider alternative patch delivery strategies such as caching and peer-to-peer and evaluate their performance. We identify key factors that determine the effectiveness of these schemes in reducing the server workload and the network traffic, and in speeding-up the patch delivery. Most of our findings are invariant properties induced by either user behavior or architectural characteristics of today's Internet, and thus apply to the general problem of Internet-wide dissemination of software updates.

Categories and Subject Descriptors: C.2.2 [Computer - Communication Networks]: Network Protocols-Applications D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

General Terms: Management, Measurement, Performance, Design.

Keywords: software updates, peer-to-peer, caching.

1. INTRODUCTION

Large scale and fast dissemination of software updates to millions of Internet users is becoming crucial to maintain high levels of protection and offer updated services and applications. As users become more proactive in keeping their software updated, the amount of traffic generated by software updates and security patches is rapidly increasing. In fact, during certain periods of time, patch distribution can account for a large fraction of the traffic in corporations and across the Internet (see Table 1).

SIGCOMM'06, September 11–15, 2006, Pisa, Italy.

URLRank	Site	GBytes	%
1	download.microsoft.com	535.5	9.80%
2	www.download.windowsupdate.com	344.8	6.30%
4	au.download.windowsupdate.com	246.4	4.50%
8	download.windowsupdate.com	96.0	1.80%

Table 1: Patch traffic generated by downloads of Service Pack 2 (SP2) inside a large corporation (1-30 Sep 04). Windows Update dominates the top 10 sites rated by bandwidth consumption.

Patches upgrade existing software with the intent to fix security vulnerabilities, update drivers, distribute new virus definitions, or release new functionality. The number of operating systems and applications, such as web browsers, games, etc., that provide online patching services is rapidly increasing. Despite the growing popularity of software updates, little is known about the process of creating and releasing patches, the traffic characteristics of patch distribution, and the potential of alternative distribution strategies.

In this paper, we analyze one of the largest update services in the world, the *Windows Update* system. Windows Update provides an automated update service for the Windows operating system, Office applications, and Exchange and SQL servers. Our goal is to find general principles and properties that can be used as guidelines to design and better architect fast and cost effective planet-scale patch dissemination. Based on a combination of empirical observations and analytical results, we identify interesting properties of today's update traffic and user behavior, such as the frequency of updates, the possibility of grouping multiple patches, the spatial and temporal characteristics of user requests, and the percentage of computers that are always online and, hence, can be instantaneously patched.

Furthermore, we study different patch delivery strategies (e.g. caching and P2P) and evaluate their performance and their potential to improve speed of patch delivery. We identify key factors that determine the effectiveness of these schemes both in terms of workload reduction on the central server and the overall Internet. For the case of P2P, we find analytical and empirical evidence showing that P2P patching is highly effective in reducing the load on the central servers. Nonetheless, P2P can generate significant load into ISPs. Hence, we analyze locality algorithms to reduce inter-ISP traffic.

We have analyzed a vast number of data traces collected over the period of a year at different points of the Windows Update service infrastructure. In total, we have parsed several billions of queries from almost 300 million computers. We believe that we have observed a significant fraction of all the computers in the Internet. To the best of our knowledge, this is the first study of a very popular update service, and we deem it can be used to draw important conclusions regarding software update distribution.

We deem that most of our findings are induced by user behavior, architectural characteristics of today's Internet, or standard soft-

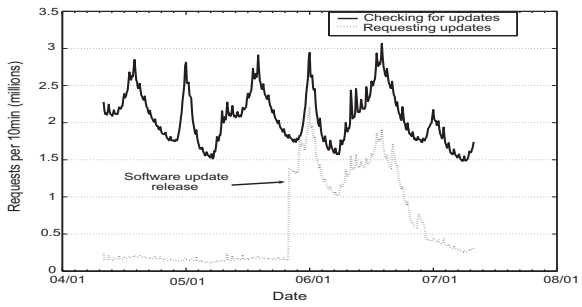

Figure 1: Number of update queries and corresponding downloads over three days. Note that a security patch is released during the second day.

ware engineering processes and development cycles, and thus apply to the general problem of Internet-wide dissemination of software updates. Some of the findings of our paper are as follows:

a) Software patches can be efficiently clustered into a small number of groups, decreasing the complexity and improving the effectiveness of any dissemination solution.

b) Approximately 80% of IPs appear during the first day of a patch release, while the number of unique IPs drops abruptly with the number of days. We also characterize the temporal correlation of users and find that for more than 90% of the IP population arrival rates are quite bursty.

c) We find that the percentage of machines that are always online and thus could benefit from an idealized instantaneous patching system is approximately 20% of the population.

d) Computers that use the update service seem to be highly updated, with more than 90% of all observed users updated with all security patches. This is expected and shows the importance of automated update systems.

e) We have estimated that the potential workload reduction provided by existing caches varies from 25% to 35%, while a full cache deployment by ISPs would result in almost all requests for an update covered by caches.

f) Despite the small size of patches and the diversity of requests, P2P distribution can considerably reduce the load on the server if users stay online for a short time after completing the download.

g) We quantify analytically the impact of P2P locality on ISP traffic. Based on analytical and experimental results, we show that locality can reduce inter-ISP traffic by more than an order of magnitude. (Similar observations have been made in [11] for file sharing applications.)

The rest of the paper is organized as follows: Section 2 describes our data sets and the Windows Update system, Section 3 characterizes patches and studies their clustering, Section 4 characterizes user's behaviors, Section 5 compares different dissemination strategies, Sections 6 and 7 present related work and conclude the paper. Proofs are deferred to the appendix of the technical report [6].

2. SYSTEM AND DATA DESCRIPTION

For this study we used a variety of datasets that span a large number of significant updates for over a year period. The discussion of the various datasets presupposes knowledge of the Windows Update (WU) system architecture and thus we will briefly present here its basic characteristics.

2.1 System description

The windows update architecture consists of a set of *update servers* where users query for new updates and a large number of *distribution servers* from which users download updates (see Fig. 2). Each user querying for updates will first initiate a request to an update server, and if available updates exist, will be redirected to one of the distribution servers. Overall, there are two types of requests: a) Requests through "Automatic Updates" that occur in pre-specified time intervals or 5-10 minutes after boot time if the pre-specified time expired in between reboots. b) Requests originating at the Windows Update website after a manual query.

The automatic update system periodically queries for updates with inter-polling times independent and identically distributed, uniform in $[18, 22]$ hours (in fact, the left-end of the interval is slightly smaller than 18). In most of our traces, only a very small percentage of the queries happen through manual updates (we will indicate it otherwise). In total, it is estimated that approximately **300 million** users are updated for every patch released.

Updates are distributed through the binary delta compression scheme [16]. Delta compression ensures that each user will receive only a "diff-file", which will correspond to the difference (*delta*) between the latest version released and the current version of the file being updated at the user machine. Thus, for every file to be patched, there exists a collection of different deltas that specify all the possible *diffs* between older releases of the specific file and the latest release. Each delta is OS-specific as well as country-specific (i.e., deltas differ per country or OS).

Patches[1] are regularly released every month. The set of patches that fix a given vulnerability is known as "Knowledge Bases". At larger time intervals (e.g., years), large collections of Knowledge Bases are released at the same time, which are called *Service Packs* (SPs). SPs consist of all the updates previously released up to that point in time (i.e., all Knowledge Bases previously released) as well as new files that introduce new functionality and/or major product improvements. As such, *Service Packs* are significantly larger in size compared to monthly patches. For example, *Service Pack 2 (SP2)* consisted of 800 Knowledge Bases compared to 8-10 Knowledge Bases that are usually included in a monthly release. SP2 introduced automatic updates by default. Prior to SP2, updates were made mostly manually.

2.2 Data characteristics

In order to profile the distribution process of such a vast system with myriads of interactions, we have collected an extensive number of traces, diverse in type and duration, at various points within the aforementioned architecture. Our datasets are described in Table 2. Overall our collected traces amount to approximately two Terabytes worth of analyzed data.

3. CHARACTERISTICS OF PATCHES

Designing efficient mechanisms for distributing software updates, requires understanding their characteristics, e.g. the number and size of files affected, the frequency of update releases, and the relations between the individual patches. The problem arises from the fact that machines can have a large set of configuration states. Recall that a single software update may change many files but individual users may be interested in only a subset of them. The primary reason is that users can patch at different times, ending up with a different set of files or versions. Such differences in the configuration and the interests of the users significantly complicate the

[1]We use the terms patches, updates, deltas, and diffs interchangeably.

Table 2: Characteristics of the collected data sets.

Set	Period	Characteristics	Collection point	Type	Description
I	4-6th Jan '06	300M polls/day 150M distinct IPs	All update servers	Polls for updates. IIS Logs	Mostly Automatic polls for new updates. See Figure 1 for query rate.
II	10-12 Aug '04 14-16 Jan '05	70K distinct IPs	One download server	Service Pack 2 (SP2). IIS Logs and packet-level traces	Manual downloads. Close to SP2 release (6 Aug'04). SP2 patched a large number of files (>4K) and included 12K+ deltas. Aggregate size for all deltas equals 266MB.
III	10-26 Oct '05 14 June '05 10 Aug '05	300K distinct IPs	Two download servers	Monthly patches. IIS Logs and packet-level traces	Mostly Automatic downloads. Regular monthly patches for small set of files (mostly critical updates).
IV	2001-2006	-	XP build release tree	Update history for all Windows XP files	Facilitates the study of delta production and publishing. Includes delta creation time, size, hash key, version, etc.

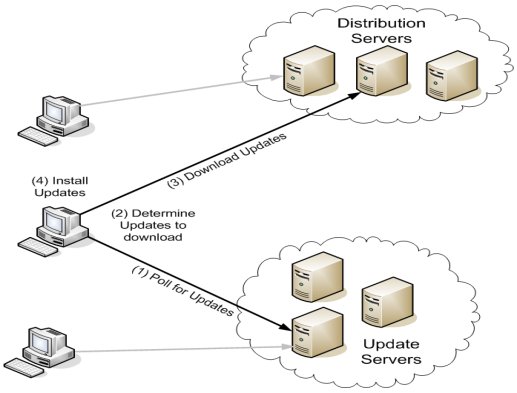

Figure 2: Overview of the update process. A client checks first with the update servers for new patches, and then, if it needs to get updated, downloads the patches from the distribution servers.

design of a patch delivery system. In Sec. 3.1 and 3.2 we examine the relationships between the updates for individual files with the objective to identify clusters of files that are updated together. If efficient clustering is possible, then, the number of possible states that need to be covered by a patching system can be significantly reduced. This can result in simpler, more efficient and scalable delivery system since fewer distribution channels are required (e.g. multicast channels, etc) and user requests are spread across fewer files which translates into higher cache efficiencies.

A file patch describes the differences between the version that exists in the user computer and the newest version. When a software update requires changes in files that already exist in the user's computer, it is more efficient to send file patches instead of the files themselves. Patches may be significantly smaller than the corresponding files and, hence, transmitting patches results in significant bandwidth savings at the server and improved update delivery times for the user, especially for modem users. Using Set-II and Set-IV we have determined that the mean size of a file distributed during XP SP2 is 73.2 KBytes, while the mean patch size is 32.9 KBytes. We have also measured that for 30% of the files, patches provide savings of at least 5x. On the other hand, the use of patches increases the diversity of the user population since the users differ not only in the components they wish to update, but also in the specific patch, which depends on the version of their files, they need to obtain.

In Fig. 3 we plot the cumulative number of released patches and new files as a function of their release time. Observe the large number of new files and deltas in three time instances that correspond to the release of the XP operating system and the releases of two major service packs (SP1 and SP2). Observe also the release of many updates in between the SPs with an average rate of 21 new patches per month.

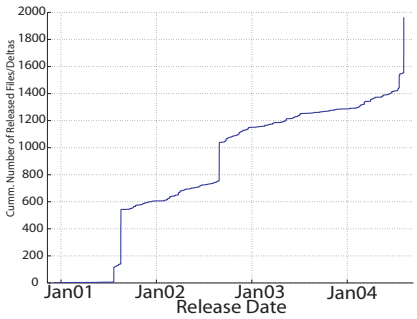

Figure 3: Set of patches included in SP2 and their releases times. Major jumps correspond to the release of XP-RTM (Aug 01), SP1 (Sep 02), and SP2 (Aug 04).

3.1 Clustering of files

We will now try to identify clusters of files that are requested together (in the next section we will repeat the same exercise for patches). To this extent, we have used traces of requests for the Windows XP SP2 distribution and the full history of updates in the XP source tree (data Set-II and Set-IV). In order to accurately determine the clustering relationship across files, we first filter out those clients for which we do not see a complete patching session. Filtered entries corresponded mostly to slow (e.g. modem) users, which required several sessions to fully patch. Filtering out those users did not bias our clustering results as our goal is indeed to sample users for which we see a complete patching session over the observation interval.

To quantify the set of files that are clustered together, we do the following analysis. Let x_i be the binary vector of user requests for file i; i.e., $x_i(u) = 1$ iff user u requested file i. We compute the cosine correlation $\rho_{i,j}$ between any pair of files i and j as:

$$\rho_{i,j} = \frac{x_i^T \cdot x_j}{\sqrt{x_i^T \cdot x_i} \cdot \sqrt{x_j^T \cdot x_j}}$$

Files that are always requested together have a correlation $\rho_{i,j} = 1$. We assume that files i and j are correlated if $\rho_{i,j} > 0.9$. We then construct a graph of the files as follows. Each node represents a file and there is an edge between two files if the correlation between the files is greater than $\rho_{i,j} > 0.9$. We then identify the connected components of that graph. Observe that, even though our method does not guarantee that every two files assigned to the same component are related to each other, we have observed that in every component the minimum correlation between each pair of files is large (more than 0.8).

In total, our data set had requests for 2029 files. From our analysis we were able to identify 26 non-overlapping groups accounting for 2003 files; only 26 files could not be assigned to a group (see Fig. 4). This represents an important clustering effect. The groups

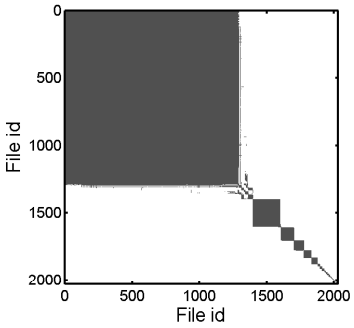

Figure 4: Groups of files identified in the SP2 download. Dark dots appear whenever two files are highly correlated. Group boundaries appear as squares.

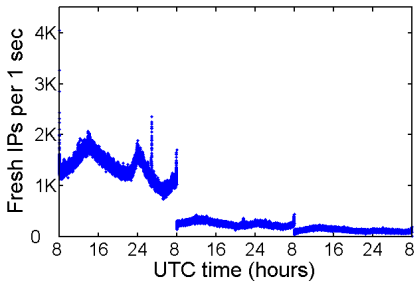

Figure 6: The rate of distinct IPs observed over three days, starting from Jan 4, 2006, 8:00 UTC. The majority of distinct IPs are observed during day 1. Peaks are observed at around 14:00 UTC and 24:00 UTC. The breakdown of IP counts over continents reveals that the first peak is due to North American users and the second due to Asian ones.

correlate with functional components of the operating system; a typical example of a group is the set of files related to the networking functionality.

In Fig. 5(a) we plot in more detail the sizes of the groups. The largest group, which includes 1460 files, was requested by all users. This group corresponds to updates of core components of the OS, as well as components that were introduced in SP2 for the first time. The other groups included 200 files or less. Some of these groups correspond to rare configurations, such as language specific files, and received few requests. Observe that the clustering of files can significantly reduce the complexity of the system; indeed, by publishing groups of files instead of individual files, the number of publishing elements reduces by two orders of magnitude (from 2029 to 26).

We now quantify the benefits of publishing only a few major groups (e.g. over a few multicast trees) and distribute the remaining files individually. Fig. 5(c) shows the percentage of requests that can be satisfied by publishing the largest k groups, for $k = 1, \ldots, 26$. Note that the 5 largest groups account for 1877 of the files and are responsible for more than 97% of the requests, which indicates that the distribution a few groups can satisfy a very large number of requests. It is outside of the scope of this paper to describe ways to automate cluster discovery. However, our analysis demonstrates that clustering of patches has a large potential grouping in reducing the complexity of a patch distribution system.

3.2 Clustering of patches

We have repeated the same analysis as above using as input the patch requests. In total, our data set had 3379 patches. In Fig.5(b) we plot the sizes of the groups of patches (the groups are ordered by their size). We have identified 125 groups that account for 3188 patches; 191 patches could not be assigned to a group. The number of groups is again significantly smaller than the total number of patches. However, compared to file grouping, grouping of patches provides a smaller aggregation factor. This is due to the larger number of possible user configurations that arise when considering individual patches; users differ not only in the component they need to update, but also in the patch they need to download to update their local version. From Fig. 5(c), we also observe that to satisfy the same number of requests, the required number of groups of patches is larger than the required number of groups of files. Thus, the user population covered by each group of patches is smaller.

This analysis indicates that publishing individual deltas instead of files decreases the clustering efficiency, consequently increasing the complexity of the system. Ideally, software update systems should publish individual files rather than deltas, and generate patches on-the-fly using chunck-based hash techniques (e.g. similar to LBFS [17]). The main challenge to produce deltas on-the-fly is, however, the required computation capacity at the servers. To scale such computation, one can use distributed systems such as P2P networks that support on-the-fly generation of updates; deltas can be generated automatically from updated peers, rather than from the server. We will study the potential of such P2P patching systems in Section 5.

4. USERS CHARACTERIZATION

In this section, we examine the intrinsic properties of update traffic and user behavior with respect to software updates.

4.1 Traffic Properties

We now characterize the arrival patterns of update queries. Queries arrive from two types of machines: **always-online-machines (AOM)** and **non-AOM**. We define AOM as those machines that have an active automatic update service that periodically queries for updates. Non-AOM machines are those that go On and Off and stay offline for a period greater than the pre-specified query interval. The automatic update service in those machines will query soon after rebooting since the query time expired. Queries can also occur from machines where the users manually check for updates, however, these are rare events.

4.1.1 Distincts IPs over time

We first examine the aggregate volume of user queries with respect to the distinct IPs using Set-I. We try to identify both AOM and non-AOM users. Note that in the case of AOM, the aggregates of queries will not exhibit spatial or temporal correlations due to the randomization process of query polling. Correlations on the other hand can be a consequence of queries that are initiated by user actions, either by deliberate manual querying or because of the queries initiated shortly after a computer boot time, for those computers that missed a scheduled query time. Ideally, in a system where instantaneous patching were feasible, it would be desirable that the majority of users poll for updates as close in time to the release of the patch as possible, so that the vulnerable population is minimized.

Approximately 80% of the observed IPs appear within the first day; the number of fresh IPs drops by an order of magnitude during the second day, and is further reduced by factor of 2 in day three. Fig. 6 shows the number of distinct IPs per second for the three day trace, where the large drops across the three days are evident through the change in the mean of the time-series. Within each of the three days, we respectively observe approximately 117, 22, and 11 millions of distinct IPs. Overall, *the number of fresh IPs within a*

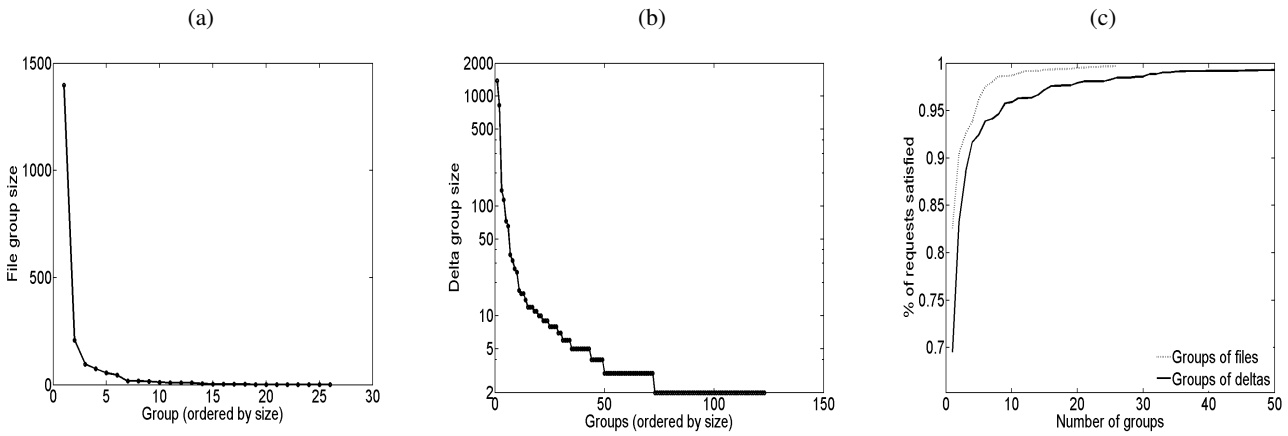

Figure 5: (a) Group size distribution for File grouping. (b) Group size distribution for Delta grouping. (c) Number of requests satisfied by publishing an increasing number of groups for Delta and File grouping.

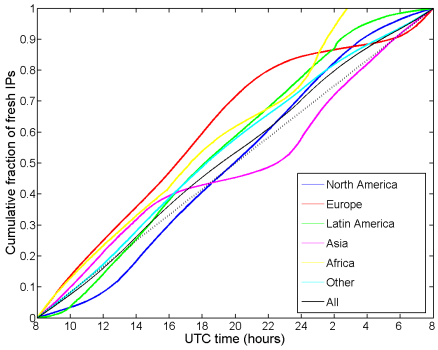

Figure 7: The fraction of distinct IPs per continent observed within the first day versus time. The slope of each curve corresponds to the query arrival rate per continent. The differences in the rates result from time-of-day effects

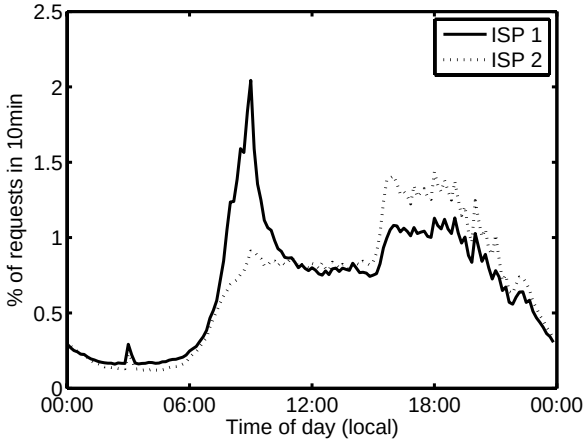

Figure 8: Number of queries for two European ISPs in the same country over time (as % over total number of queries). While the time-of-day effect pattern is similar, the different profile of ISP customers further affects the query arrival rate (ISP1 focuses on business customers, while ISP2 focuses on residential customers).

day decreases abruptly with the number of the days since the initial observation time.

4.1.2 *Time-of-day effects*

Time-of-day effects are present in user queries. This may be surprising given that the system was designed to smooth the arrival of update requests. However, it can be explained by the fact that a large number of machines initiate a query shortly after a machine boot time. Indeed, queries from non-AOM users will lead to time of day dependencies on the rate of query arrivals. Such dependencies will be pronounced when examining aggregate rates per continent; by mapping IPs to their continent of origin.

Fig. 7 shows the cumulative fraction of distinct IPs observed within the first day over time, where the slopes of individual curves correspond to the arrival rates over time per continent. Our initial observation time is 8:00 UTC and thus Europe exhibits the largest query arrival rates at the beginning and at the end of the day. The peak rate for North America happens within the interval 13:00 to 16:00 hours, which is consistent with the 5 hours time difference between UTC and US East coast and additional three hours difference from US East to West coast. The time difference of UTC to Beijing is 8 hours and as expected the query rates from Asia peak at around 24:00 UTC. Similar curves are obtained for the second and third day of our traces.

4.1.3 *Uniformity and burstiness of queries*

We further characterize the temporal correlation of user requests by studying the statistical properties of the arrival process. While time-of-day dependences are evident in the continent aggregate rates, such correlations appear even more emphasized when we examine the per-AS query arrival rate. However, in addition to time-of-day effects, ASes may exhibit different query rates depending on the profile of their customers. Fig. 8 presents a typical example of two geographically collocated ISPs where the profile of their subscribers (residential vs. corporate) results in dissimilar query arrival patterns within the day.

In order to statistically characterize our entire sample of approximately 19K observed ASes, we analyze the distribution of queries in time and try to understand whether they are uniformly distributed and, if not, quantify their burstiness.

We admit as null hypothesis that queries from an AS are uniformly distributed over time, which would be true if the users were querying at random times, uniformly within 0-24 hours. To examine this hypothesis, we perform Kolmogorov-Smirnov test (KS-

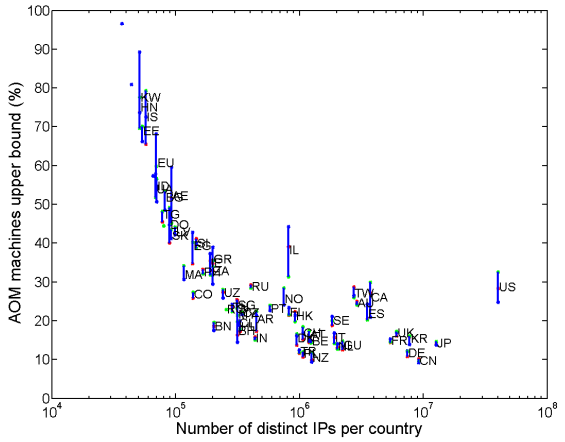

Figure 9: Estimated percentage of distinct IPs classified as AOM per country versus the total number of distinct IPs per country. The estimates are upper bounds with 95% confidence. The larger the number of distinct IPs per country, the tighter the estimate would be to the uknown parameter.

test) for each AS. [2] Overall, for the first day of our trace the null hypothesis cannot be rejected for 52% of ASes. However, these ASes only amount to less than 0.1% of the distinct IP's observed over the day! We obtained the same percentages by running the KS test for each of the other two days. Thus, *for 50% of the ASes that acount for more than 90% of our IP population, the uniformity of query arrivals is rejected.*

We also examined the magnitude of the query rate burstiness by evaluating the maximum workload of a hypothetical server that serves AS-aggregate of queries and quantified that, indeed, the burstiness is in many cases larger than if the AS-aggregate query rates were uniform in time.

4.2 Estimated Always on-line Machines

Always Online Machines (AOM) are of specific interest since they can be instantaneously patched using an ideal push patching system (e.g. for instance a ubiquitous multicast channel joining all users). The polling instants of an AOM user can be modeled as a renewal stochastic process with inter-polling times independent and identically distributed, uniformly in $[18, 22]$ hours.[3]

We use an estimation technique based on the aggregate counts of queries to estimate an upper bound on the number of AOM users with a fixed confidence, which is a good estimate for the unknown number of AOM users in aggregates of many users, provided that over some time intervals the number AOM users dominates, i.e. the number of other users can be neglected (as it would be over night periods for aggregates of users with geographical proximity). The estimation technique uses the fact that the number of fresh IPs observed from AOM users over a time interval is a binomial random variable and uses known large deviation bound that holds for binomial random variables. For space reasons, the details of the estimation method are provided in [6]. Alternatively, we could have collected per host query instants and classified the hosts by using a hypothesis test to check whether the observed samples are drawn from the known inter-query time distribution, but note that the duration of our Set-I is 3 days and thus with mean per-host inter-query time of 20 hours, we would have only 3.6 per host queries.

Using Set-I, our estimates suggest that *approximately 20% of the population is "always" online* and thus could be patched inmediately. This estimate is based on the results in Fig. 9, which shows the estimated percentage of AOM users for each country versus the number of distinct IPs observed from that country. The estimates are upper bounds and would be tight to the unknown percentage for countries with sufficiently large number of the observed IP's. Fig. 9 suggests this to hold for countries with $> 300K$ distinct IP's and it is for those estimates that we assert the 20% figure.

The trace we used to determine Always On Machines (Set-I) also includes requests from users that manually visit the Windows Update site to search for updates using a browser (e.g. as opposed to automatic updates). Such events could bias our estimation of AOM users and add noise to our AOM estimation. However, such users are a small percentage of the total population and account for very few requests during the low activity periods (e.g. a user opening their laptop in the middle of the night), thus, they add a very small error to our AOM estimation.

4.3 Frequency of computer updates

We now study how up-to-date computers are kept around the world. To this extent, we examine a Windows XP patch from June 2005 (Set-II).[4] The minimum size of a requested delta for this patch is 22 KBytes, while the maximum was 800KBytes. Smaller deltas correspond to updates for recent versions of the file, while larger deltas update older versions. In Fig. 10, we plot the CDF of requested delta sizes in different parts of the world. Jumps in the graph correspond to requested deltas. For instance, we see a first jump around 20KB, which corresponds to all computers that are regularly updated and need few changes. Similarly, jumps at larger deltas relate to less updated computers.

We can see that there is a large difference on how updated computers are kept around the world. For instance, US and Japan users keep their machines highly updated, with 90% of users updating from the inmediate previous version, while this happens only for $50 - 70\%$ of users in China or France. The fact that users are more or less updated, is likely a combination of download speeds and the amount of time they spend on-line.

From this plot, we can also determine how many deltas per file should be made available for download to satisfy a large number of users. This is an interesting question that impacts the design of an on-line patch system. Providing a large number of deltas per file increases the chances that nodes do not have to download the complete file. However, it increases system's complexity and reduces the access reference locality of any given file, since requests are spread across many deltas. From our data, we can observe that 90% or more of all users can be satisfied by very few deltas of recent versions (e.g. 2-3). Thus, a very small set of deltas is sufficient to patch most users, thereby, decreasing the need for publishing and archiving a large number of older deltas per file.

Always Patched Users: We have also calculated the percentage of users that are patched with the latest critical updates. To this end, we have analyzed the behavior of 300 Million users that periodically query for updates (Set-I), and monitored the percentage

[2]The KS statistic is defined as the maximum absolute deviation of an empirical CDF and a candidate CDF assumed under null hypothesis. If, instead of using the absolute deviation, we use either the most positive or the most negative deviation, the KS statistic is called one-sided.

[3]This polling rule may be regarded a good design choice as it is a random walk on a 24 hour clock, with inter-polling times having a density so that a host polling instant converges to a uniform distribution over a day.

[4]This is a security patch addressing HTML help vulnerability; see http://support.microsoft.com/kb/896358.

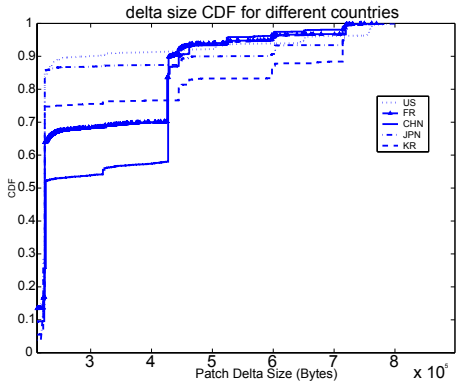

Figure 10: Distribution of the number of requests for different delta sizes across different countries (US, FR, CHN, JPN, KR). Smaller deltas indicate more updated computers. All deltas correspond to the same file.

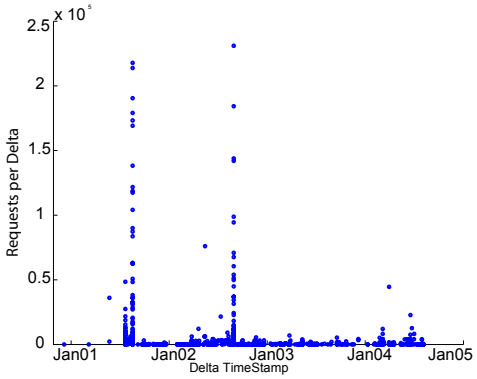

Figure 11: Requests per delta included in SP2. Notice three major spikes, corresponding to people requesting SP2 installation on top of XP-RTM, XP-SP1, or XP-SP1 plus all security patches released to date.

of them that need to patch at any point in time (see Figure 1). We have observed that for the most part, the number of users that query and are not fully updated with all previous patches (e.g. require an update) is less than 10%. We have also tracked what happens immediately after the release of a security patch during the second day of the trace. We have observed that the number of users requiring the security patch rapidly decreases to less than 20% of the total users by the end of the second day.

In summary, the results indicate that a large percentage of the population is highly updated. Such encouraging results are in sharp contrast with the results obtained by analyzing the state of computers prior to the release of SP2, when the automate update service was not turned On by default. To highlight this, Fig. 11 shows the timestamp of those deltas requested during the SP2 distribution; older timestamps indicate users updating from older versions. It shows that the number of users that were updated with the most recent patches was less than 5%, with 22% of users updating from SP1 versions and 60% from XP RTM versions. This emphazises the importance of automatic patching schemes, which do not rely on manual intervention.

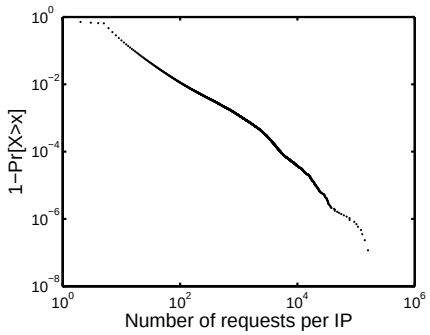

Figure 12: CCDF of the number of requests per IP address.

5. PATCH DISSEMINATION STRATEGIES

We now investigate alternative update-delivery strategies and compare them to standard central server solutions. Using analytical and empirical results we evaluate the potential of (i) caching, (ii) peer-to-peer, and (iii) peer-to-peer with locality. Our purpose is not to evaluate in detail the various strategies, but to study some 'what if' scenarios and to provide qualitative results.

To evaluate the alternative strategies, we assume that hosts are partitioned into groups, which we call subnets. For example, all hosts that belong to the same ISP, or the same Autonomous System (AS) belong to the same subnet; in the rest of this section, subnets may be thought of ASes or ISP networks. We shall make the assumption that inter-subnet traffic (e.g. cross ISP) is expensive and intra-subnet is preferential. We shall study the effect of the alternative policies in reducing the server load and the inter-subnet traffic.

5.1 Caching

In this section we first estimate the benefits of a patching distribution system that uses the currently deployed Web caches. We then estimate the potential benefits of an ideal full cache deployment with caches at each AS.

Assume that caches are deployed in a subset $\mathcal{C}$ of subnets. The reduction in the workload of the download servers can be computed using the assignment of hosts to subnets, and the number of patches per host. Denote with $N_{i,j}$ the number of hosts from subnet j that need update i. The number of requests directed to the central server from a subnet $j \in \mathcal{C}$ is at least 1, if there exists a host in j that needs i. Indeed, a lower bound of the number of downloads D from the central server is:

$$D \geq \sum_{j \in \mathcal{C}} \sum_{i} 1_{N_{i,j}>0} + \sum_{j \notin \mathcal{C}} \sum_{i} N_{i,j}$$

where the equality holds for infinite-capacity caches. Without caching, the number of downloads from the central server is $\sum_j \sum_i N_{i,j}$, and, hence, caching reduces the load at the server by a factor α:

$$\alpha = \mu \cdot \left(1 - \frac{1}{\overline{S}}\right)$$

with

$$\mu = \frac{\sum_{j \in \mathcal{C}} \sum_i N_{i,j}}{\sum_j \sum_i N_{i,j}} \text{ and } \overline{S} = \frac{\sum_{j \in \mathcal{C}} \sum_i N_{i,j}}{\sum_{j \in \mathcal{C}} \sum_i 1_{N_{i,j}>0}}.$$

In other words, μ is the fraction of updates needed in subnets covered by caches and $\overline{S}$ is simply the mean number of updates per subnet over subnets that deploy caches. A similar approach can be used to derive the benefit in terms of bytes.

Estimated caching deployment: To calculate the benefit of using the currently deployed web caches for update dissemination, we

need to estimate the parameters μ and $\overline{S}$. Using Set-I, we identify IP addresses from which we have received multiple requests in a period of 17hr. Since each (AOM) machine polls the update servers at most once during a period of 17hr, those IP addresses are shared by multiple machines and likely represent caches, NAT devices, firewalls, modem pools, etc. It is impossible to identify the type of the device using our data; instead, we assume that an IP address belongs to a cache if the number of requests received from that IP is above a threshold. Our estimate is an upper bound on the number of currently deployed large caches, since we falsely classify large NATed points as caches.

We have used Set-I and counted the number of polls and the number of distinct IP addresses observed. Fig. 12 plots the complementary CDF of the number of machines per IP address. We observe that it follows a heavy-tailed distribution, with a few IPs having a very large number of machines (as large as 240K machines).

We have experimented with various thresholds for identifying caches and our estimates for μ, $\overline{S}$, and α are given in Table 3. For thresholds over 10 users/IP (which should exclude most home NATed environments) the estimate of the percentage of requests from users behind caches, μ is 20-29%. Note that larger thresholds (e.g. 50 users/IP), do not change the estimate significantly. Although the percentage of queries that can be satisfied from caches is non-negligible, the central servers still need to handle 210-240M users, hence, the benefit of using existing caches is rather limited.

Threshold	# Caches	Users Covered	μ	$\overline{S}$	α
> 2	25,351,342	186,766,792	62%	7.37	53.58%
> 5	5,504,963	115,942,495	38%	21.06	36.20%
> 10	1,718,094	88,419,310	29%	51.46	28.44%
> 25	656,903	69,701,692	23%	106.11	22.78%
> 50	209,765	59,933,202	20%	285.72	19.92%

Table 3: Estimate of the load reduction α at the server by using existing caches. We assume that an IP address belongs to a cache if we have received more than *threshold* requests from that IP. We also assume the distribution of a single file, hence $\overline{S} = \frac{\sum_{j \in \mathcal{C}} N_j}{\sum_{j \in \mathcal{C}} 1_{N_j > 0}}$. Data from Set-I.

Full cache deployment: We now study the impact of an ideal caching deployment with a cache at each subnet ($\mu = 1$). We have assumed that each subnet represents an AS. Even though this scenario is not representative of the current Internet, it allows us to estimate the potential benefit of deploying more caches. Observe that if every subnet has a cache, then the server needs to serve at most as many copies of an update as the number of subnets, therefore, for very popular files, the cache hit ratio will be close to one. For less popular updates or those with many deltas, the hit ratio may be significantly lower.

To evaluate the cache performance in a realistic setting with a mixture of files, we have used the data from Set-II. Observe that Set-II contains observations for a significantly smaller population than Set-I (due to observing a single server), but, on the other hand, allows us to identify the files and deltas requested by each user. The diversity in the set of files and deltas requested by the users may reduce the benefit of caching.

Using Set-II and assuming requests for files (and not deltas), we have estimated $\overline{S} = 4.18$, which translates into $\alpha = 76\%$. We have also studied the impact of delta distribution on caching performance (as opposed to publishing only the latest update), and obtained values of $\overline{S} = 3.01$, and $\alpha = 67\%$. The cache efficiency would have been greater if we were able to monitor more servers or collect data for larger periods of time. However, even for user populations of tens of thousands (as opposed to hundreds of millions) that request many different sets of files and deltas, the cache efficiency is still quite high.

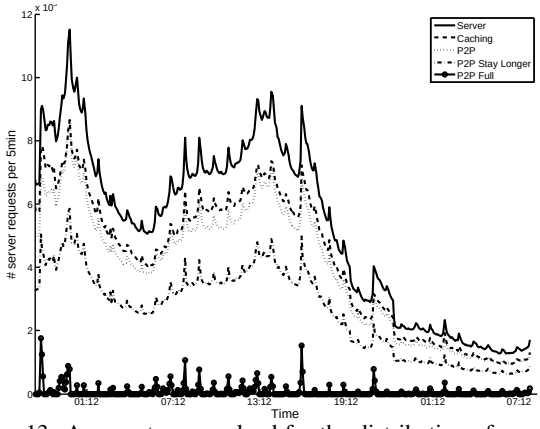

Figure 13: Aggregate server load for the distribution of one patch with (i) client server; (ii) caching; (iii) p2p with upload time equal to download time; (iv) p2p with upload time twice as download time; (v) p2p with uploaded data as much as downloaded (full).

5.2 Peer-to-Peer

We now consider P2P as a delivery mechanism for software updates. In P2P , end-systems collaborate in the patch distribution, e.g. a patch can be downloaded from any randomly chosen peer that is online and has the file. P2P systems are already being used to download software and they are becoming popular among content providers [2]. A P2P scheme is self-scalable, since the system capacity increases with the number of users, copes well with flash-crowds, and, hence, it is attractive for patch dissemination.

P2P systems are very effective when a large number of users demand a large file. However, P2P patch distribution systems face a number of challenges: a) average patch size is small, which limits the periods that a peer stays connected, b) potentially large set of patches, which increases the diversity of peer requests, c) multiple versions per patch, which reduces even further the opportunities for sharing. Only if a large number of peers target the same version of the same patch at the same time, P2P provides significant savings for the content provider and the end-users. Similarly, a P2P system that favors intra-subnet connections is effective in reducing the backbone traffic (i.e. inter-subnet traffic) only if there is a large concentration of peers in the same subnet. For instance, for a given arrival pattern it is not clear how long peers need to stay online to satisfy a target fraction of requests within a subnet. This will be addressed later in the paper (see Corollary 1 in Sec. 5.3.2).

There are many challenging problems in designing a patching distribution system based on P2P, such as guaranteeing secure and timely patch delivery, and protecting user privacy, which are outside of the scope of this paper. Instead, we focus on the potential benefits for the content provider and the end users. We also do not consider the impact of NATs, which has already been well studied [7, 9], but assume that efficient NAT traversal mechanism will be in place to permit full peer connectivity.

In Fig. 13 we show the number of requests received by the server after the release of a patch (on 5-Jan-06) over a period of 1.5 days. In the same figure, we show the server load when using a P2P and a caching system. The peer arrival pattern was derived from Set-I. We have used data from Set-II to assign download rates to the peers. We assume that peers have asymmetric bandwidth and can serve at a fraction, $1/4$ in our experiments, of their download capacity.

For the case of caches, we assume a scenario where caches have infinite resources and are placed at each IP address from which we have received more than 25 requests (see Table 3).

From Fig. 13, we can see that current caches provide a load reduction in the server roughly equal to 20%. The benefit of using P2P depends on how long peers stay in the system. If peers disconnect from the P2P network immediately after downloading the file, then they consume more download capacity into the system that the upload capacity that they offer. This is due to the asymmetric links that are common in today's Internet; the server needs to add extra capacity to deal with the asymmetry. However, even in that case, P2P provides similar benefits to caching. If peers stay in the system to serve as many bytes as they have received, then, the benefit of the P2P system increases dramatically and the load at the server becomes almost negligible (e.g. less than 10% of the load with a client-server architecture). This is an interesting result since it shows that for a given patch, there is enough overlap across peers to ensure that a P2P system performs well, even if peers only stay in the system for a small period of time.

Note that the assumption of infinite cache capacity may be unrealistic, and that our predictions of the cache performance may be optimistic. In fact many caches may not be able to handle the load generated by large flash crowds that follow the release of a patch. On the other hand, a P2P system is able to easily cope with this load as presented in Fig. 13 as long as peers stay on-line a little longer after they finish the download. Admittedly, the precise benefits of P2P will depend on the exact details of the P2P system, which we have not modeled in our analysis; however, our results suggest that P2P has great potential for patch dissemination.

5.3 Peer-to-Peer impact on the network

5.3.1 Peer-to-Peer with Random Matching

In P2P systems the burden of patch delivery shifts from the central server to the individual peers and, subsequently, to their subnet (e.g. AS, ISP). If the matching is random, then the vast majority of the peer exchanges will be between peers of different subnets. Assuming that each peer uploads as much as it downloads, then, the traffic that enters a subnet (total download) will equal the traffic that exits (total upload). As a result, the traffic at the inter-subnet links will double with random P2P matchings compared to the case of downloading from the server.

We use the user arrival pattern of Set-I to quantify the amount of inter-subnet traffic. We assume that a new update is made available at the beginning of the trace that is of interest to all users. We map IP addresses to autonomous systems[5] and use the ASes to assign users to subnets. We assume that users stay online for a short period of time, equal to 1min in our experiments. The 1min interval is roughly equal to the mean online time observed in our data; later we will consider with different online times. In Fig. 14 we plot the normalized number of downloads from and uploads to remote subnets as a function of the size of the subnet, i.e. the number of hosts in that subnet. From Fig. 14 we observe that, for large enough subnets, the upload and download traffic is linear to the size of the subnet. Moreover, the incoming traffic to a subnet equals the outgoing, and both of them increase with the size of the subnet.

5.3.2 Peer-to-Peer with Locality

To alleviate the adverse impact of inter-subnet traffic, we should augment the peer matching algorithm to give preference to "local" connections; in other words, peers should give priority to connections with other peers in the same subnet, instead of choosing peer

[5]Using data provided by the MS operations group.

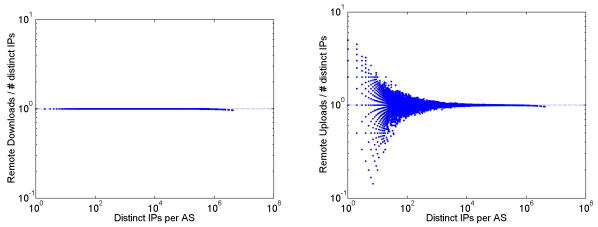

Figure 14: Peer-to-peer: (Left) per subnet downloads from remote subnets versus the number of distinct IPs per subnet; (Right) same but for per subnet uploads from remote subnets. One day worth of requests. Each host remains online for a fixed time of 1 min.

uniformly at random [11]. We assume that peers have an efficient mechanism for discovering peers in the same subnet; the discussion of such mechanisms is outside the scope of this paper. Next we analyze and quantify the impact of P2P with locality.

We wish to derive analytical estimates for (i) the amount of data downloaded from remote subnets, and (ii) the amount of data uploaded to other subnets. We perform trace-driven simulations to validate our analytical predictions and to estimate the workload reduction.

In order to simplify the exposition of the analysis, and with no loss of generality, we consider the dissemination of a single update. In the remainder of this section, we omit the details of the analysis details and present only the main results. The interested reader can find proofs in [6].

In summary, our findings are:

a) Locality decreases the amount of data uploaded per subnet by a factor that (approximately) decreases exponentially with the mean number of active users per subnet.

b) With locality the ratio of uploads to downloads per subnet increases as a function of the size of the subnet (recall that without locality the ratio was constant for large enough subnets).

c) Simple formulas that approximate the uploads and downloads per subnet.

Download traffic: We first analyze the impact of P2P with locality on the number of downloads from remote subnets. Recall that these downloads occur when a local host cannot find another (online) host in the same subnet to download from.

We assume that there are N_j hosts from subnet j that are interested in the update.[6] Each user query appears at a random time from the patch release, independent of other users, with cumulative distribution function (CDF) $A(\cdot)$ and density $a(\cdot)$.[7] After querying for an update, each user stays online for a random time drawn from a CDF $B(\cdot)$ with the complementary CDF denoted by $\overline{B}(\cdot)$. The number of downloads from remote subnets depends on the two distributions $A(\cdot)$ and $B(\cdot)$, and on the size of the subnet N_j as follows:

THEOREM 1. *The expected number of downloads, $D_j(t)$, from a remote subnet j in a time interval $[0,t]$ is*

$$\mathbb{E}(D_j(t)) \quad = \quad N_j \int_0^t \left(1 - \overline{B} \star a(s)\right)^{N_j - 1} a(s) ds \qquad (1)$$

where $\overline{B} \star a(t)$ is the convolution of $\overline{B}(\cdot)$ and $a(\cdot)$.[8]

[6]Since we focus on a single update, we drop the index i in $N_{i,j}$.

[7]For our polling users, the CDF $A(\cdot)$ is the residual-time distribution of the user inter-query time CDF (observed from a user query instant).

[8]That is, $\overline{B} \star a(t) = \int_0^t \overline{B}(t-s)a(s)ds$.

PROOF. See appendix of [6]. □

The ratio $\mathbb{E}(D_j(+\infty))/N_j$ can be interpreted as the expected fraction of queries requested from remote hosts. Observe that for given $A(\cdot)$ and $B(\cdot)$ the fraction of remote downloads decreases exponentially with the size of the local subnet.

We next present an estimate from the previous formula that requires to know only the mean host online time and a quantity that relates to the query arrival rate.

COROLLARY 1. *Suppose $a(t)$ is non-increasing*[9] *with t and that the following limit exists $a^* = \sup_{t>0} A(t)/t$. Denote with b the mean host online time and let $\rho = ba^*$. Then, if $\rho \leq 1$, we have:*

$$\mathbb{E}(D_j(+\infty)) \geq N_j(1-\rho)^{N_j-1}.$$

PROOF. See appendix of [6]. □

If the queries arrive from always on machines, or if the inter-poll times can be approximated by exponential distribution with mean $1/a^*$, then the parameter a^* is precisely the per-host query rate. Under the same assumptions, ρ can be interpreted as the mean number of host queries that fall in a time interval of length equal to the mean host online time. Note that for $\rho \ll 1$, $(1-\rho)^{N_j-1} \approx e^{-\rho N_j}$ and thus for large subnets the estimate of downloads that cannot be served locally can be approximated by $N_j e^{-\rho N_j}$.[10]

We assign users to subnets using the data in Set-I and simulate peer-to-peer with locality with a) query times randomly generated as if all users were AOM and b) with query times as observed in our data. Due to space limitations we only present the results of (a), but note that very similar results hold for (b). We first calculate the data downloaded from remote subnets and show the results in Fig. 15 (a) and (b). Observe that the analytical results are very close to the experimental. The number of requests that cannot be satisfied locally is less than about 500 for any subnet and drops significantly for larger subnets.

Recall that the number of requests that cannot be satisfied locally for a subnet j can be estimated by $N_j e^{-\rho N_j}$. This function achieves maximum at $N_j = 1/\rho$ and the maximum value is $(1/\rho)e^{-1}$. Given that the per-host query rate is 1 in 20 hours and the mean host online time equals 1 min, this yields an estimate for the maximum download from remote subnets of approximately 442, which agrees with the results of Fig. 15.

Upload traffic: We now present similar analysis for the number of remote uploads from a local subnet j with N_j hosts. This result will provide us the estimate of the upload traffic reduction for p2p with locality. Due to the underlying sampling of the peering hosts, the following distribution over "subnet sizes" plays a crucial role in determining the benefit of locality:

$$\nu(i) = \frac{\text{number of subnets with } i \text{ hosts} \cdot i}{\text{total number of hosts}}. \quad (2)$$

[9]This assumption on the query arrival process does not hold always in practice as we observed from our data of queries from distinct IPs, but it does hold for queries from AOM users and, for instance, it holds approximately for flash-crowds.

[10]Observe that the result of Corollary 1 holds exactly for systems where each poll counts as an update. Suppose each host polls according to a Poisson process with a rate $\lambda > 0$. Then, the result of Theorem 1 reads as $\lim_{t\to+\infty} \frac{\mathbb{E}(D_j(t))}{\lambda t} = N_j e^{-\rho N_j}$ and similarly for the result in Theorem 2, $\lim_{t\to+\infty} \frac{\mathbb{E}(U_j(t))}{\lambda t} \sim N_j \mathbb{E}(e^{-\rho S})$, where ρ is the ratio of the mean host online time to the mean host inter-polling time.

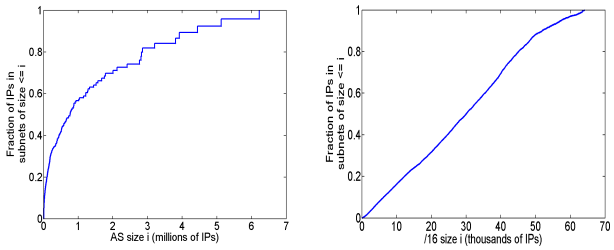

Figure 16: The CDF of ν (aggregated number of hosts in all subnets with i hosts). (Data: Set-I)

Before stating our result, we present estimates for the distribution ν of subnets that partition the users according to the autonomous system they belong to, and according to the two most important bytes of their IP address (/16 subnetting). Fig. 16 plots the empirical CDF's of ν using the data of Set-I.

The following result gives an estimate for the uploads from a local subnet to remote subnets:

THEOREM 2. *For subnet sizes N_j, bounded by an aribtrarily fixed constant, assume that the distribution ν is given by (2) for the total number of hosts $N = \sum_j N_j$ going to infinity. The expected uploads, $U_j(t)$, from a local subnet j to remote subnets in a time interval $[0,t]$ satisfies*

$$\frac{\mathbb{E}(U_j(t))}{N_j} \to R_t, \text{ as } N \text{ tends to infinity},$$

where R_t is:

$$R_t = \int_0^t \mathbb{E}\left(\left(1 - \overline{B} \star a(s)\right)^{S-1}\right) a(s)ds$$

with random variable S having distribution ν.

PROOF. See appendix of [6]. □

In the limit, the uploads to remote nodes is R_∞. R_∞ depends on (i) the rate of queries over time, (ii) the distribution of host online time, and (iii) the distribution of subnet sizes ν. A small value of R_∞ implies that the number of inter-subnet uploads is small, hence, the benefit of using P2P with locality is large.

Suppose now that the per-host query rate $a(t)$ is non-increasing with t (see also Corollary 1). A lower bound for R_t is given by:

$$R_t \geq \mathbb{E}\left((1-\rho)^{(S-1)}\right) A(t), \text{ for } \rho \leq 1, \quad (3)$$

where recall that ρ may be interpreted as number of polls in an interval of length equal to the average host online duration (see Corollary 1). The lower bound implies that R_∞ is larger than $\mathbb{E}\left((1-\rho)^{S-1}\right)$ (since $A(\infty) = 1$). This lower bound is tight in our data set, and, we believe, that is a good estimate of R_∞.

We now revisit the simulation results presented in Fig. 15, which were computed assuming that users stay online for 1min. We are interested in relating the user online times with R_∞ and, as a result, with the reduction in inter-subnet traffic by uploading preferential to nodes in the same subnet. Recall that the parameter ρ depends on the mean user online time b, since $\rho = ba^*$. We use Eq. 3 and the observed distribution of subset sizes (this is the distribution ν from which we pick the values S; the distribution is drawn in Fig. 16) to evaluate R_∞. Observe that R_∞ is decreasing with ρ, the mean number of polls during a typical online interval. Fig. 17 plots the value of R_∞ as a function of ρ. Observe that R_∞ decreases fast as we increase ρ. The average polling frequency in the Windows Update system is $a^* = 1/20$hr. For that polling frequency, we have

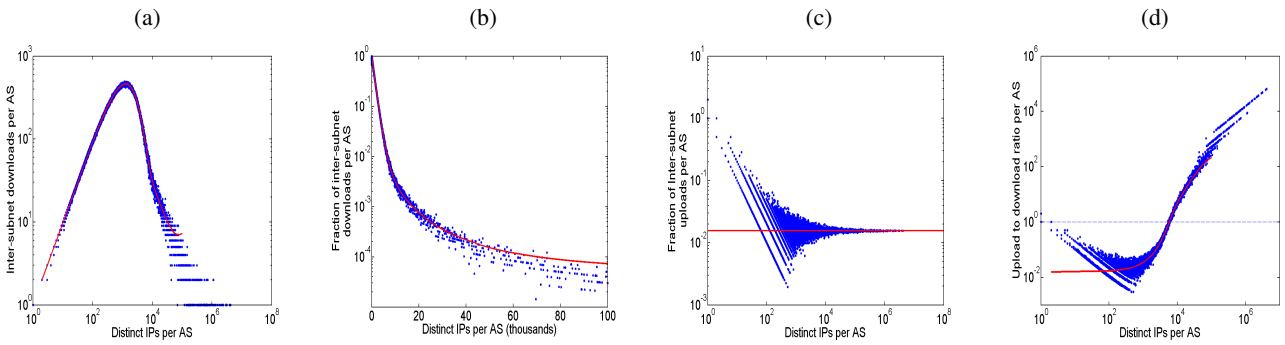

Figure 15: Impact of peer-to-peer with locality. Simulation results are shown with crosses and analytical results with solid lines. (a) number of downloads per subnet; (b) same but normalized with number of IPs in a subnet; (c) fraction of uploads per subnet normalized to subnet size; (d) upload to download ratio per subnet. The host online time is fixed to 1 min.

Table 4: Host online times to achieve a certain reduction in the number of remote uploads with P2P with locality. The hosts poll with a rate of 1 every 20 hours.

Subnet type vs R_∞	1/10	1/100	1/1,000	1/10,000
/16	14 sec	72 sec	10 min	72 min
AS	2.15 sec	72 sec	24 min	4 hours

computed the required mean host online time to achieve some specific numbers of remote uploads (small R_∞); the results are given in Table 4. Host online times as low as few minutes can reduce the number of remote uploads by more than an order of magnitude. In particular, if hosts stay online for 1min, then $\rho = 1/1200$. Experimentally we find that $R_\infty \approx 0.015$ (see Fig. 15(c)) for large enough subnets. This is an important result, since it implies that with locality the required upload traffic for large subnets decreases to 1.5% of the traffic without locality (e.g. random matching); an improvement of almost two orders of magnitude!

We have shown that locality reduces both the remote downloads (Theorems 1 and Corollary 1) and the remote uploads (Theorems 2 and Eq. 3). However, how does locality affect the balance of per-subnet uploads and downloads? Are there subnets that upload to other subnets significantly more than what they receive? To answer those questions we study the ratio of the remote uploads to the remote downloads; the result is shown in Fig. 15(d). We observe that large subnets, with more than 5K users in Fig. 15(d), upload to other subnets more than what they download; large subnets contribute resources. We use our results of Theorem 1 and Theorem 2 to numerically compute the expected per-subnet upload to download ratio. The results in Fig. 15(d) demonstrate conformance of the analytical result (solid line) with the experimental findings. Ideally, the ratio of uploads to downloads for all subnets should be close to 1. However, from Fig. 15(d) we observe that, even though the upload traffic reduces, the ratio of uploads to downloads is greater than 1 for subnets larger than $(1/\rho)\log(1/R_\infty) \approx 5040$ (as observed experimentally). This implies that with locality large subnets contribute more resources than what they receive. It remains an interesting issue to design P2P matching algorithms with low inter-subnet traffic and balanced uploads to downloads for all subsets, even for the very large ones.

A final remark is due concerning the nature of the results presented in Theorem 2 and Eq. 3. Both of them are asymptotic and should hold for large host populations. Indeed, our data set contains measurements for a large population (around 300M users), and, as a result, our empirical findings agree well with our analysis. (Note that the results in Theorem 1 and Corollary 1 are for arbitrarily given subnet sizes N_j.)

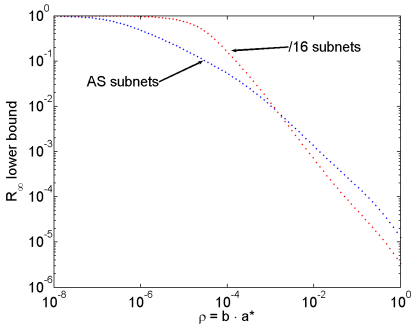

Figure 17: The estimate of upload reduction factor for peer-to-peer with local subnet preference for AS and /16 subnets, versus the product of the mean host online time and the characterization of host request arrival rate $a^* = \sup_{t>0} A(t)/t$. The result is an exact lower bound under assumption that host query arrival rate is non-increasing with time.

6. RELATED WORK

The problem of keeping a large number of machines updated with the latest software, fixes and software upgrades has been of great interest to many professionals especially in large IT departments. Previous work focused primarily on designing middleware and configuring systems to enable remote software installations and upgrades [5, 10, 18, 21–23]. Even though the networking issues involved in updating many machines in parallel have not been explicitly addressed in those papers and, to the best of our knowledge, in previous work, Shaddock et al. notices the problem that may arise by many machines simultaneously connecting to the software distribution center and attempting to download large files over a shared network [10]; they propose the use of a ticket system to distribute the load.

In terms of commercial interest there are many and diverse products that propose and/or use automatic updates [13, 15, 20, 25, 27]. We expect that in the future the number of products that use automatic updates will increase significantly.

From a networking point of view, the problem of distributing software updates is a content distribution problem. Content distribution, which has been studied for more than a decade, proposed various technologies, such as multicasting, caching, CDNs [1], peer-to-peer networks [4, 8], that can be used for the distribution of updates. Indeed the current Windows Update system uses an exten-

sive and geographically distributed content distribution network. However, software updates have distinct characteristics that distinguish them from traditional content distribution problems; exploiting those characteristics may lead to more efficient distribution. As an example, since the downloading of software updates is typically a background activity, user downloads can be scheduled appropriately with the goal of balancing the network load.

The design of an update distribution system that exploits similarities between users depends on understanding typical machine configurations and the software update process, in particular how and when the configuration of a machine changes. [12, 26] performed extensive studies for understanding the state of typical machines with the goal of understanding typical misconfiguration problems and increasing the reliability of end systems. [3] reports statistics on machine availability and load characteristics. The findings of [3, 12, 26] can be used to understand the state of a typical machine and are complementary to our work. Even though our analysis provides statistics at a coarser granularity, we study much larger and diverse population.

Previous work has presented extensive studies of very popular Internet applications, such as Web servers and video streaming services [3, 19, 24]. The problem of propagating updates to a large number of non-homogeneous users using broadcast channels has also been studied in [14]. To the best of our knowledge, our study is the first large scale study of a live patching system.

7. CONCLUSIONS

In this paper, we characterize a large commercial update service with the aim to draw general guidelines on how to best design and architect a fast and effective planet-scale patch dissemination system. Automatic software updating is one of the most prominent architectural issues in today's Internet since fast and effective update distribution to millions of machines is increasingly popular as a method for keeping machines up-to-date with the latest software features and bug fixes. In particular, rapid distribution of security patches is vital for protecting against security attacks and malware. Unlike traditional content distribution systems, such as the Web, patch distribution systems use a near-push functionality, have publication times that depend on development cycles or malware appearances, use differential update mechanisms, have distinct traffic patterns, and require minimum delivery times.

Based on a combination of empirical observations and analytical results, we identify interesting properties of today's update traffic and user behavior. We provide evidence that patches can be clustered into a small set of functional components, thus, reducing the complexity of any patch delivery system. We estimate the percentage of always on-line users and the characteristics of the user arrival pattern, and, using those estimates, study the performance of an (ideal) patching solution. We consider two standard content distribution architectures, caching and peer-to-peer, and evaluate their applicability to patch dissemination. We demonstrate that P2P has a great potential for providing fast and effective patch delivery. This is an interesting observation since current P2P systems work best when many users download few large files, rather than many small files as it is the case with patching systems.

Using extensive measurement data from multiple vantage points, we characterize the behavior of current patching systems and gain insights that may help design more efficient distribution mechanisms. We believe that the workloads used in this study represent typical behavior of live patching systems. Most of our findings capture general properties induced by either user behavior, architectural characteristics of today's Internet, or properties of current software engineering systems and development cycles, and thus apply to the general problem of Internet-wide dissemination of software updates.

Acknowledgments

We would like to thank the MS Software Distribution, Windows Update and MSCOMS teams for their tremendous support and help during the data collection process and for providing us with invaluable information. In particular, we would like to express our deepest gratitude to Ryan Auld, Jeff Davis, Josh Dunn, Taqi Jaffri, Deighton Maragh, Tom McGuire, Kurt Parent, Mark Roellich, Rob Satterwhite, Manoj Shende, and Mike Sligger.

8. REFERENCES

[1] Akamai home page. <http://www.akamai.com>.
[2] BBC iMP. http://www.bbc.co.uk/imp/.
[3] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *ACM SigMetrics*, 2000.
[4] B. Cohen. Bittorrent. <http://www.bittorrent.com>.
[5] J. Dunagan, R. Roussev, B. Daniels, A. Johnson, C. Verbowski, and Y.-M. Wang. Towards a self-managing software patching process using black-box persistent-state manifests. In *IEEE Intl. Conf. on Autonomic Computing*, 2004.
[6] C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnović. Planet scale software updates. Technical Report MSR-TR-2006-85, Microsoft Research, 2006.
[7] C. Gkantsidis, J. Miller, and P. Rodriguez. Anatomy of a p2p content distribution system with network coding. In *5th Int. Work. on P2P System (IPTPS)*, 2006.
[8] Gnutella. <http://p2pjournal.com/main/gnutella.htm>.
[9] S. Guha and P. Francis. Characterization and measurement of tcp traversal through nats and firewalls. In *ACM IMC*, 2005.
[10] C. Hemmerich. Automatic request-based software distribution. In *USENIX 14th System Administration Conf. (LISA)*, 2000.
[11] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *ACM/USENIX IMC*, 2005.
[12] E. Kiciman and C. Verbowski. Analyzing persistent state interactions to improve state management. 2005.
[13] Mac OS X: Updating your software. http://docs.info.apple.com/article.html?artnum=106704, 2005.
[14] S. Mahajan, M. Donahoo, S. Navathe, M. Ammar, and S. Malik. Grouping techniques for update propagation in intermittently-connected databases. In *IEEE Conf. on Data Engineering*, 1998.
[15] Microsoft update faq. http://update.microsoft.com/microsoftupdate/v6/default.aspx?ln=en-us.
[16] Using binary delta compression (bdc) technology to update windows operating systems. Microsoft online White Paper.
[17] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *SOSP*, 2001.
[18] P. Osel and W. Gnsheimer. OpenDist - incremental software distribution. In *USENIX 9th System Administration Conf. (LISA)*, 1995.
[19] V. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: Findings and implications. In *ACM SigComm)*, 2000.
[20] Red Hat Network. http://www.redhat.com/en_us/USA/rhn/, 2005.
[21] D. Ressman and J. Valdes. Use of Cfengine for automated, multiplatform software and patch distribution. In *USENIX 14th System Administration Conf. (LISA)*, 2000.
[22] M. Shaddock, M. Mitchell, and H. Harrison. How to upgrade 1500 workstations on saturday, and still have time to mow the yard on sunday. In *USENIX 9th System Administration Conf. (LISA)*, 1995.
[23] L. Sobr and P. Tuma. SOFAnet: Middleware for software distribution over Internet. In *IEEE Symp. on Applications and the Internet (SAINT'05)*, 2005.
[24] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *ACM SigComm*, 2004.
[25] Symantec corp. http://www.symantec.com/.
[26] H. Wang, J. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with peerpressure. In *USENIX OSDI*, 2004.
[27] ZDNet updates.com. http://updates.zdnet.com.