BruteSaaS: A centrally managed pull-based peer-to-peer system for software distribution in enterprise environments.

Niels Joubert
BruteSoft Inc.
Sunnyvale, CA
7 January 2011
Email: niels@brutesoft.com

***Abstract*—BruteSoft has created a pull-based software distribution system that allows central management of peer-to-peer data distribution. This solution uses the net client bandwidth to distribute large amounts of data without the need for dedicated servers. In this paper we show our load-balancing algorithm that allows our distribution system to achieve endpoint performance without taxing wide area network bandwidth. We show how this approach leads to a set of modifications to the BitTorrent protocol that allows BitTorrent to function well in an enterprise environment, respecting WAN and LAN traffic capabilities to deliver software packages 84 fold faster than a standard client-server approach, without the strain BitTorrent normally places on the backbone of an enterprise network. Finally, we propose an extension to our current software distribution system to support smartphone devices, seamlessly using set-top boxes, local machines and central data stores to improve the performance and lessen the server requirements of mobile software delivery.**

***Keywords*-Peer to peer computing; Content distribution networks; Information technology;**

I. Introduction

Provisioning computers with software — whether it is an update, a patch, updated virus definitions or a new install — means moving data to a machine and executing a set of instructions to make this software available to the end-user. End-users do this for their personal computers by downloading or using physical media to acquire the software package, and manually click through instructions to install it. Enterprises face the challenge of keeping their computers updated with the latest software and licenses required by their employees and keeping their computers secure by applying patches in a timely fashion. Expecting employees to be responsible for manually following policies is unreasonable, especially in the face of time-sensitive security updates and virus definitions, and requiring an IT team to follow the same manual practices as end-users to maintain computers does not scale beyond very small businesses.

The software industry has responded by creating management tools for enforcing software policies and performing remote, unattended software installs. These tools allow system administrators to configure the software ecosystem of computers, and aid package movement and installation.

Traditional software delivery inside the enterprise uses a central repository on a dedicated server that publishes software packages to the network. Clients, upon receiving instructions to install a certain package, connect to this central server to download the necessary package. The central repository has to support serving software packages to all the client machines attempting to do an install, thus the central repository is often load-balanced across multiple physical servers.

Enterprises often exist in multiple physical locations, with the enterprise network topology necessarily reflecting this physical layout, with WAN backbone links between offices, and local LANs connecting machines within an office. WAN connections are necessarily lower-bandwidth than the total bandwidth available inside an office, so to avoid having all clients connect to a central repository for software downloads, distribution servers are placed in branch offices to mirror the central software repository. The infrastructure requirements for software provisioning thus demands maintaining both a pool of central servers and possible distribution servers in branch offices. As the amount of clients in an enterprise increase, so does the requirements of the software distribution infrastructure and the associated costs of purchasing, maintaining, housing and powering these servers.[1]

Software management tools have approached this problem with several attempts to provide scaling without expensive infrastructure requirements. IP Multicast technology allow a single stream of packets to reach many client machines, with scaling supplied by the Network Layer of the infrastructure rather than the end-point servers[2]. Multicast, when available[2], requires that all receiving machines coordinate perfectly, correctly receive each packet with no interruptions or failures. As the number of clients increase, failures inevitably occur — both as network load causes packet loss

[1]Just powering the average server requires 275 Watts[1] for operation and an equivalent amount for cooling, thus the infrastructure requirements has a very real effect on the operating budget of an organization.

[2]Multicast has not enjoyed the penetration of point-to-point IP connections, and is often not implemented or enabled on routers, severely limiting its usability.

and users interact with computers while software distribution occurs. This prevents system administrators from doing on-demand software distribution or operating during business hours. This approach also does not scale to devices that experience large fluctuations in network quality and availability such as laptops and mobile devices.

We present a different approach from the client-server and multicast model. By making clients responsible for gathering the necessary data, rather than receiving data pushed from server, we separate the policy requirements of what to install from the source of the software package and the exact time of data acquisition. This allows us to exploit the redundancy of software installs moving the same data to many endpoints by sharing data between endpoints. The BitTorrent protocol, a well-known protocol that implements this approach, has been shown to work well for distributing large amounts of data to many machines, since transfers increase in speed and decrease in server load as more computers download the same file.[3] We show how BitTorrent can be used on enterprise networks, where it has not gained much penetration due to its invasive effects, by modifying the protocol to limit WAN connections and respect other traffic on the network. We then show how this modified protocol combined with an intelligent client can perform remote software management that outperforms currently available solutions, removes the need for distribution servers or multicast, enables on-demand software installs for all devices, and works without prohibitive network configuration requirements.

II. Paper Outline

First we present the patent pending software distribution model developed for the BruteSaaS products. We consider how peer-to-peer and BitTorrent technologies plays into this software distribution model, and we present a set of modifications to BitTorrent to reduce its footprint in the network topology of enterprises. We then discuss applying the same distribution model in mobile environments using transport mechanisms other than BitTorrent.

III. BruteSaaS Model

A. Data vs Metadata

Software provisioning requires the actual package to install, as well as significant infrastructure to supply identification, authentication, organization and monitoring of client machines, describe software installation instructions and dependencies, and associate software packages with client machines. BruteSaaS separates the software package — the data — from the rest of the system's information — the metadata. This separation allow management to be performed from a single central server while software packages can arrive at a client from the most suitable source.[3]

[3]The algorithm described in this section is currently has a provisional patent filed.

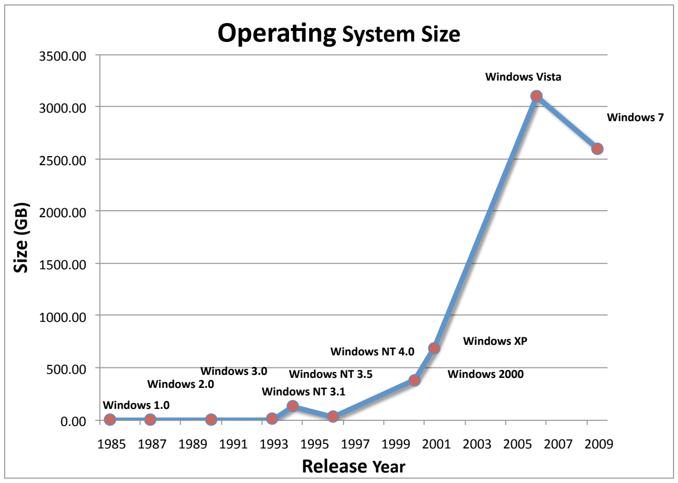

Figure 1. Growing of OS sizes over time

Software packages have increased in size over the last decade, with software packages easily measuring several gigabytes (see figure 1). This increase in package size along with the steady increase in computer deployment has caused tremendous demand for data - if a 2GB software package needs to be installed across 1000 machines, 2TB of data will have to me moved through the network.

The metadata required for remote software installs are of a much smaller size - authentication, identification and instructions for installing software packages are of kilobyte size, and can often be precomputed and cached for fast responses. A single commodity server can adequately support hundreds of thousands of client machines from a central datacenter[4].

B. Pull technology: the client as an intelligent endpoint

BruteSaaS builds most of its operational logic into a client that resides on each device managed by the system. The client is responsible for maintaining the software ecosystem on the machine in adherence with a centrally configured policy, requesting policy details and making the necessary changes on demand. This makes for a set-and-forget approach for the system administrator. To enable fine-grained control, a client can be remotely forced to evaluate its policy adherence and update itself, allowing software installations to be happen on-demand or on-instruction.

Whenever a client finds that its own configuration is out of sync with its set policy, it builds a list of software packages required to update its configuration, and requests metadata about these packages from the central server. This metadata includes a description of the software package in terms of a list of fixed-size data chunks that make up the data of the package. The download of a package then occurs on a chunk-by-chunk basis. The chunks themselves does not have to be requested from the central server, as described in section III-A, thus it attempts to download each chunk from the optimal client it can legally connect to. The protocol for finding peers - legal clients - and which peers a client can legally connect to will be described in section V. Only if a chunk is not available from any of its peers, will a client attempt to connect to the central server to request that chunk.

Clients are thus also responsible for accepting requests from other clients, possibly replying with chunks from a software package, and publishing which chunks it makes available to its peers. Clients will report progress and current state throughout the updating process to a central server for monitoring purposes.

C. *Role of central servers*

A central server is responsible for endpoint identification, authentication and configuration. The system administrator organizes the software ecosystem on machines by grouping clients into sets of machines, henceforth referred to as a computer-set, and mapping it to a shared software configuration. The administrator defines a software policy for a computer-set by mapping a computer-set to a list of software. The server then exposes an interface through which clients can report their current configuration to receive a list of software to install and remove in order to become compliant with its software policy. The server will also supply a client with a list of peers for each software package, and maintain a list of peers for each software package.

The central server also serves as the root package repository from which all software packages originate, but like a content distribution network, not necessarily the server a client will download the package from. The central server will act as a "superseeder" - it will respond to download requests from peers, but it will limit the amount of connections it will accept to avoid flooding WAN connections with superfluous downloads if a package is already available from the requesting client's peers.

Identification and authentication of endpoint machines in enterprise environments has been addressed by several well-established technology stacks, including Microsoft's ActiveDirectory, the LDAP protocol with extensions, and specific solutions built into products like SpiceWorks and BigFix. We plug into whichever of these solutions are available and require clients to identify and authenticate themselves through these avenues before the server will respond to any requests from a client.

The central server will also provide a monitoring infrastructure to which clients can report their current status, so that system administrators can see an overview of the software availability and status of their enterprise

IV. Peer to Peer in Enterprise Settings

Enterprise topologies tend to be derivations of the traditional hub-and-spoke topology. A central datacenter contains company-wide services, with WAN connections forming a backbone to branch offices. This is unlike the internet, where the topology forms a mesh with much more intricate connections.

Peer-to-peer technologies tend to be shunned in the enterprise, since it does not provide the centralized control and authentication, or respect for the described network topology, of client-server models. Enterprises are often forced to install second-tier servers in branch offices to control the bandwidth and latency between endpoints and requested data by mirroring data locally. Peer to peer technology has not been able to flourish in such an environment without flooding WAN links with connections to peers in other branch offices, since peer-to-peer protocols typically assume the cost of connecting to a peer is independent of their location[5].

A many-to-many, peer-to-peer protocol for the enterprise has to limit the majority of peer traffic to the local LAN within an office. Peers separated by WAN links should not connect to each other, but rather download from a common central server, thus organizing into many local swarms with each swarm connected to the central repository. Peers should prefer to download files from their local swarm rather than connect to the central server, thus the server should seed a file to each swarm as few times as possible. The peer-to-peer protocol should also respect other traffic sources and limit the amount of resource usage on endpoints to avoid interrupting end-users. We designed a derivative of the BitTorrent protocol that incorporates these requirements, which we now present.

V. BitTorrent Protocol Modifications for Enterprise Environments

BitTorrent has been shown to work extremely well in situations where a single file has to be delivered to many end-points. Unfortunately, its randomized selection of peers and aggressive multi-connection downloading and uploading schemes often throttle other network traffic and cause bottlenecks in the network backbone. We introduce a client selection algorithm that limits WAN traffic, provide reliability through a central fallback system, and minimize system footprint by limiting resource and network bandwidth usage.

A. *Automatic Topology Discovery and Local Peer Discovery*

A BitTorrent client implementing the BitTorrent protocol maintains a list of connected peers and its relationship with each peer. This relationship captures its interest in a peer (if it has data the client wants) and whether it's choking a peer (responding to requests from the peer). Clients inform their peers of all the pieces of a file they have as they acquire pieces, so each peer knows the distribution of file pieces through its connected peers. A peer selects the next piece to request according to a rarest-first algorithm by choosing from pieces made available by its peers. Peers in a swarm will inform each other as they acquire chunks, enabling each client to maintain the list of available chunks per

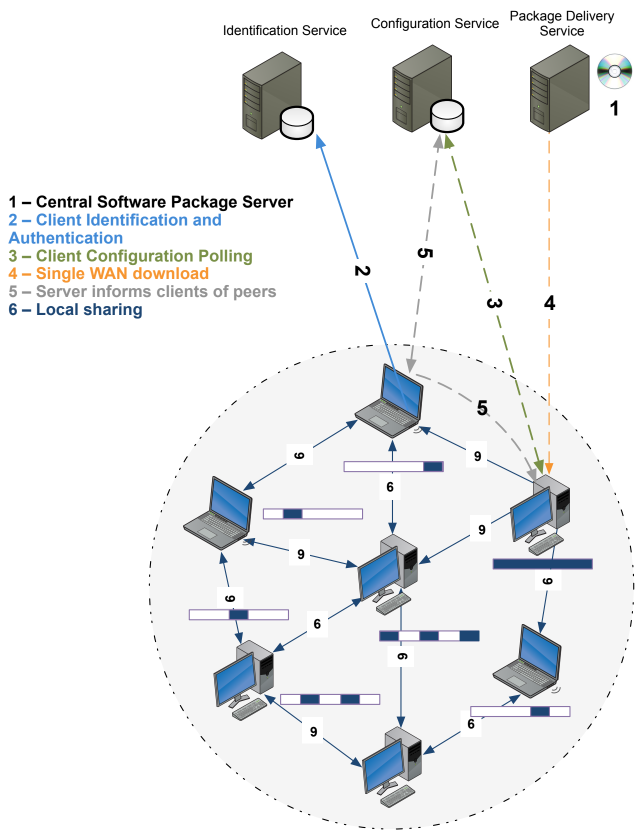
Identification Service
Configuration Service
Package Delivery Service
1
1 – Central Software Package Server
2 – Client Identification and Authentication
3 – Client Configuration Polling
4 – Single WAN download
5 – Server informs clients of peers
6 – Local sharing
2
5
3
4
5
6

Figure 2. BruteSoft protocol diagram

peer. Since each chunk is requested individually from the most appropriate peer, the download is extremely resilient to clients leaving and joining the swarm as chunks can easily be requested and re-requested from the available hosts.

The BitTorrent protocol defines a tracker responsible for supplying the list of all known peers to each client, allowing clients to decide who to connect to. Clients choose a random subset of this list to connect to. This approach is well-suited to the internet, but places undue strain on enterprise networks since many connections can travel over the WAN backbone between branch offices. The BruteSaaS client limits connections to peers grouped into its computer-set (machines that share a configuration) and clients that reside in the same subnet as itself. This works well since heavy BitTorrent traffic remains behind a single router.

To maintain the list of local peers, a client also implements the Local Peer Discovery algorithm first implemented for the μTorrent client. A client will listen for and respond to a broadcast for local clients, and will send a broadcast to find local clients on download start or resume. This maintains the local swarm even if the download is interrupted or the central server becomes unreachable.

B. *Central SuperSeeder Fallback*

Clients consider the central server to be a superseeder that provides a fallback when downloading cannot proceed from its peers. The central server always publishes the full data of any software package, and allows clients to request a chunk directly.

A client selects the next chunk to request based on a rarest-first policy of chunks currently available in its swarm. Only if no chunk is available from its local swarm will a client attempt to connect to the central server. On successful connect, chunks are requested from the central server until new chunks becomes available from its peers. Since a client requesting chunks from the central server will tend to remain in a state where it has more chunks than any other peer in its swarm, it becomes the ad hoc distribution server for the swarm, downloading chunks from the server and seeding these rare chunks to its local swarm. All the other clients in its swarm will refrain from connecting to the central server and attempt to get the chunks this peer is downloading into the swarm.

The server avoids many clients repeatedly attempting to download from itself in the case where all the clients in a swarm share a chunk faster than the client currently connected to the server can download more chunks. The server allows only one client per swarm to download from it, choking all other peers, forcing peers to wait for chunks from the single peer in a swarm downloading from the central server. If this peer disappears, the server unchokes other peers in the swarm as necessary, continuing the delivery of a software package to a swarm.

This algorithm has the overall effect of limiting the amount of connections to the central server to a single connection per swarm, without any explicit specification of *which* peer is responsible for downloading chunks from the central server. This makes for an extremely robust downloading system where election of the peer downloading unavailable chunks from the server happens on an ad-hoc basis whenever no new chunks are arriving at a swarm. Since peers will not try to connect to the central server as long as a single client is downloading chunks, even if a swarm can share a chunk much faster than a chunk can be downloaded from a central server, this approach works even if there are gross imbalances between the bandwidth of WAN and LAN links.

VI. Architecture

BruteSoft has built the algorithm described in section V into its BruteSaaS product, a companion to the SpiceWorks desktop management suite. BruteSaaS allows system administrators to deploy software packages to computers on its network, where the identification and configuration of computers are managed through the SpiceWorks interface. Each machine managed through BruteSaaS has client software installed on it, responsible for implementing the algorithm from section V.

BruteSaaS associates each software package with a list of instructions to install or uninstall the package on a client. Upon client connection, the client's current state is used to build a list of changes the client should make to its configuration, where this list consists of the instructions for each software package along with the torrent file for the software package. The BruteSaaS client will query the central server for a list of its local peers, and pull the required software packages to itself, resuming the process if interrupted at any stage.

VII. Future Work

A. Evolving to support superscale distribution

Our current approach is well suited for distributions of tens to tens of thousands of machines. As enterprise topology grows beyond a hub and spoke model, the central server starts to experience heavy load as many swarms connect to it as the initial seed. The decision to consider the central server as the only superseeder no longer provides a good tradeoff. Work has been done to build world-scale peer-to-peer networks — one such instance in the Skype client — that relies on electing a subset of peers as ultrapeers, responsible for moving data between subswarms rather than relying on a central server. We plan to investigate a similar approach by exploiting the recursive tree-like structure of large enterprise topologies. Rather than have a single central server with all subswarms connecting directly to it, we suggest engineering an approach where tier 1 peers can connect directly with a central server, and tier 2 swarms connecting to these tier 1 peers, with tier 1 peers automatically elected in a swarm. Our initial approach, named FlashFlood, is illustrated in figure 3 (still only in the design phase). The actual hierarchy would self-organize, with tier-1 peers automatically selected, such that a zero-conf system remains possible.

B. Adaption to heterogeneous environments

The BruteSoft model proposes an intelligent endpoint that pulls metadata from a central server and software packages from the most appropriate source. There is dispute whether the BitTorrent protocol can function efficiently in mobile environments, and researchers have shown ways to extend the BitTorrent protocol for better performance in mobile environments[6], [7], [8]. Currently the main drawback of BitTorrent in the mobile world is the highly dynamic nature of peer availability and network quality, and the power and usage limitations of phones to serve as uploaders to other phones: thus we suggest an extension to our distribution system that uses BitTorrent between devices with long-running connection (desktops, laptops and set-top boxes) but abandons BitTorrent for a one-to-one peer-to-peer protocol for mobile phones.

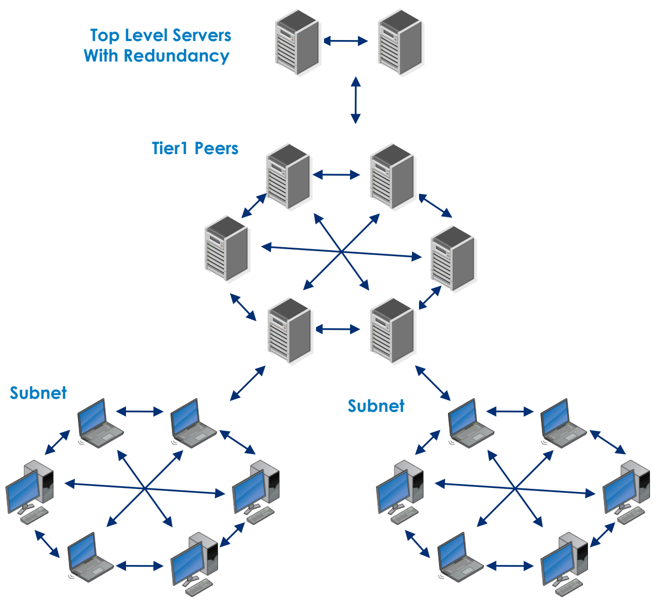

Figure 3. Flashflood superscale distribution

Clients that intelligently select a download location most appropriate for its current environment can provide significant improvement over standard client-server methods by offloading data transfers from a central distribution center to a cloud of midpoint devices topologically closer to the end-user's mobile phone. These phone-specific peers does not partake in uploading data to swarm members, but intelligently selects the most appropriate peer currently available to leech data from. Mobile phones tend to connect to the end-user's wireless network while the user is within range of their home or work hotspot. This allows for efficient software download from the always-on devices available on the end-user's local network, especially if the telecom company has established devices such as set-top boxes in the end-users's home. We propose investigating an infrastructure where midpoint devices provide a distributed cache for mobile devices, such that distributions servers and local machines can provide a wide variety of data sources for mobile phones to download from, removing load from the central distribution server architecture.

VIII. Related Work

Twitter uses BitTorrent as the distribution system for their Capistrano-based server deployment infrastructure. Twitter uses the traditional push-based approach without deploying a client, using BitTorrent as the download solution, and directly logging into each server to deploy the package downloaded through BitTorrent. Their distribution system does not need to use a different BitTorrent protocol, since all distribution stays within a single datacenter. Twitter has reported a decrease from 40 minutes to 12 seconds in distribution time.

BigFix addresses the problem of maintaining distribution servers by allowing system administrators to manually select client endpoints to serve as mirrors of a certain package, allowing other clients to connect to the promoted client rather than the central server. This does not attempt to use a peer-to-peer system, but rather eases the creation of second tier servers. This still requires manual management of the distribution network, and introduces brittle points of failure since non-dedicated client machines do not have the same guarantees of a dedicated server.

Microsoft's Branch-Cache system, specific to Windows 7, allow arbitrary clients to download from each other. This provides many of the same benefits of the BruteSaaS approach, but since chunks are not freely exchanged amongst all peers in local neighborhood, performance for any individual node depends purely on the performance of the single peer it downloads from.

IX. Conclusion

In this paper we demonstrated a new set of ideas for software distribution to allow peer-to-peer connections to be deployed in the enterprise. Modifications to BitTorrent that implements these ideas were made such that BitTorrent performs amicably in the enterprise environment. We've shown how BitTorrent can provide performance and reliability while respecting the topology of an enterprise. We presented the BruteSaaS product and how it uses these ideas to build a distribution tool that manages a BitTorrent-based distribution from a central server. Lastly, we've shown how our initial abstractions can be extended to the mobile world, and we've suggested a possible infrastructure to bring performance at low infrastructure cost to the mobile world.

Acklowledgements

The author would like to thank our private funding sources for their foresight in enabling the development of this technology.

References

[1] M. Kazandjieva, O. Gnawali, B. Heller, P. Levis, and C. Kozyrakis, "Identifying energy waste through dense power sensing and utilization monitoring," Computer Science Lab, Stanford University, Tech. Rep., 2010.

[2] S. Deering and D. Cheriton, "Host groups: A multicast extension to the internet protocol," Internet Engineering Task Force, RFC 966, Dec. 1985. [Online]. Available: http://www.rfc-editor.org/rfc/rfc966.txt

[3] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, 2003. [Online]. Available: http://bittorrent.org/bittorrentecon.pdf

[4] *BigFix Server System Requirements*. [Online]. Available: http://support.bigfix.com/bes/install/serverreq.html

[5] A. Krifa, M. Sbai, C. Barakat, and T. Turletti, "Bithoc: A content sharing application for wireless ad hoc networks," in *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, 2009, pp. 1 –3.

[6] J. Nurminen and J. Noyranen, "Energy-consumption in mobile peer-to-peer - quantitative results from file sharing," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008, pp. 729 –733.

[7] S. Rajagopalan and C.-C. Shen, "A cross-layer decentralized bittorrent for mobile ad hoc networks," in *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on*, 2006, pp. 1 –10.

[8] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla, "Code torrent: content distribution using network coding in vanet," in *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking*, ser. MobiShare '06. New York, NY, USA: ACM, 2006, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1161252.1161254