Cost-efficient content-distribution techniques for mobile devices

Davy De Winter

Supervisor(s): Filip De Turck, Joris Moreau, Bart Dhoedt, Piet Demeester

Abstract— **When users want to download content (e.g. music-files, large webpages) on their mobile devices (e.g. PDAs) they are confronted with large delays when a lot of users are simultaneously downloading. We first focused on the performance-evaluation of a number of classic content distribution architectures (including load-balancing and encryption-offload). In many scenarios, not the access network bandwidth is the main bottleneck, but rather the overloaded server infrastructure. From our results can be concluded that these architectures are not capable to cope with the file-distribution problem in a cost-efficient way. Next we have proposed and are evaluating peer to peer and multicast solutions to solve this problem. For peer to peer we first developed a centralized model and compared this to distributed models. For multicast we are evaluating and comparing the possibilities to add reliability to multicast to make it suitable for file-transfer. Moreover an emulation testbed has been set up.**

Keywords— **performance-evaluation, content-distribution, reliability, cost-efficiëncy, peer to peer, multicast**

I. Introduction

THERE are specific scenarios where people want to download in a very small time-window the same content in a reliable way from e.g. a webserver. Realistic examples are electronic magazines: every week on a certain day a new version of the e-magazine is published and a lot of abonnees want to read it almost instantly. Servers may not be able to cope with this peak-load. We've first evaluated and determined the optimal configuration of some classic content distribution systems: the three most-popular webservers (Apache 1.3, Apache 2.0 and Microsoft IIS 6.0), a classic high-performance load-balancing system (Linux Virtual Server with Direct Routing, LVS-DR) and encryption-offloading hardware-systems. In the second part of this paper we propose some solutions to handle the specific problems that exist with these classic architectures: peer to peer networks and reliable multicast. We've augmented existing peer to peer solutions with fairness-techniques and adapted them so they're ideally suited for distributing a limited number of files under a large number of users. We've also extended existing multicast-solutions with reliability-mechanisms. The implementation and evaluation of these proposals is an ongoing work and results will be published in a follow-up to this paper.

II. Performance evaluation classic architectures

A. Web servers

We'll focus on Apache 2.0 (the results of IIS 6.0 and Apache 1.3 are highly comparable). We've evaluated the webserver performance using Avalanche-and Optiview-equipment to emulate a certain number of (HTTP-and HTTPS-)requests per second (or SimUsers/sec) and measure the delay, the 2 most important performance-parameters. Both of these are mostly influenced by 3 factors: the ciphersuite used for SSL-encryption (if HTTPS is used), the filesize of the content and the access network bandwidth. The influence of the last 2 factors on the maximum number of SimUsers/sec is illustrated in figure 1. We performed our tests on a machine with Linux kernel 2.6.11, 3 Gbit-ethernet nics with TCP-offload support, 2 Gb of RAM and 2 AMD Opteron CPUs 1.6 Ghz. The filesize was 2 Mb. As every request is handled by the webserver by a separate thread or process, with slow access networks, packets cannot be sent fast enough and (socket-) buffers get full. Because buffers are full, processes and/or threads have to wait before they can send new network packets to the receivers and go to sleep. As new requests keep coming new threads and/or processes are created. Eventually, the maximum number of threads and processes is reached faster as is the case with fast access networks and this becomes the limiting factor (and not the CPU for example) with slow access networks concerning the maximum number of SimUsers/sec. On the contrary for fast access network bandwidths, the CPU becomes the first limiting factor when encryption is used. It also depends heavily on the ciphersuite as is clear from figure 1 (e.g. DES-CBC3-SHA is more CPU-intensive because previous packets are used to encrypt current packets). In figure 2 one can see a comparison between the theoretical delay for each access network bandwidth and the measured (average) delay for the maximum number of SimUsers/sec as determined in figure 1. Delay is sometimes up to 7 times higher then the theoretical forecast.

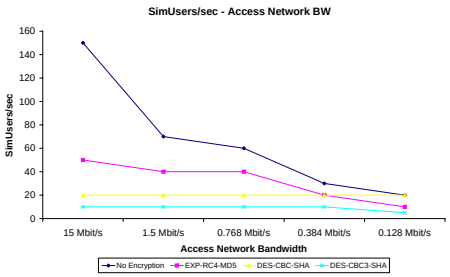

Fig. 1. The maximum number of SimUsers/sec in function of the access network bandwidth and encryption for Apache 2.0

B. Load balancing and encryption offloading

In order to test load-balancing we've chosen to evaluate LVS-DR [1]. We used 2 backend-webservers and one load-balancer. Requests coming from one of the Avalanche emulators go to the load-balancer and are spread across the back-endservers (both

D. De Winter is with the department Industriële Wetenschappen (INWE) Hogeschool Gent (Hogent), Gent, Belgium. E-mail: Davy.DeWinter@Hogent.be .

	Theoretical Delay (ms)	No encryption	EXP-RC4-MD5	DES-CBC3-SHA	DES-CBC-SHA
1,5 Mbit/s	11180	37366	14090	11532	20081
768 Kbit/s	21845	47182	101870	22412	28378
384 Kbit/s	43690	218210	196811	55238	78221
128 Kbit/s	131070	897536	715216	445535	261563

Fig. 2. Theoretical delay vs. measured delay for Apache 2.0

running Apache 1.3). With LVS-DR the load-balancer only has to process incoming traffic, never outgoing traffic, which reduces the load on the balancer itself heavily (the main reason for chosing LVS-DR). The load-balancing is done using the number of established TCP-connections per server (possibly weighted). With standard-PC's (Athlon 3500+ / 0.5 Gb memory / PCI-bus) one can handle up to 10 back-end servers with similar specs. However, a big router is necessary between the backend-servers and load-balancer, and the clients. A load-balancing architecture can solve the problem of handling a very large number of requests but cannot reduce costs. We also performed some tests on an SSL-offloading proxy hardware-system; this can save the backend-server cpus from the burden of realtime-encryption but costs are still very high. Previous results proved the classic architectures are not scalable in a cost-efficient way. Therefore other solutions which are proposed further are necessary.

III. Cost-efficient reliable content-distribution with peer to peer and Multicast

A. Peer to peer solutions

We're currently developing 2 peer to peer solutions (a central Napster-like model and a more distributed Chord-based model). We'll focus on the second one (figure 3) The original Chord [2]

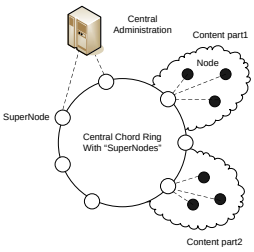

Fig. 3. Chord based P2P File distribution

is a peer to peer architecture using consistent hashing. The hosts are arranged in a Chord-ring and due to consistent hashing all files are evenly distributed under the participating nodes. Chord was originally designed when the number of files was much higher then the number of hosts. In our file-distribution scenario,the adverse is true. Therefore we'll treat hosts like content and vice versa. We've cut the original file (e.g. the e-magazine) that has to be distributed in different pieces. Every node in a central Chord-ring is responsible for one piece and for a number of nodes that got the same hashvalue. Once a new node joins the network, it gets all the hashvalues for every piece from the central administration and the corresponding supernodes decide which node will have to upload the piece of content. Once a node has uploaded as much content as it has downloaded, it can leave the network (fairness).

B. Reliable Multicast solutions

The main problem with the classic architectures is the unicast-nature: there exists a one-to-one relationship between sender and receiver and if there are 50 000 users, the same content has to be sent 50 000 times. Therefore we're also looking for the possibilities to use multicast to spread the file(s) (see figure 4). With multicast the content only has to be injected once in the network and it is spread to all receivers that are interested (receivers who joined a multicast group). So the netwerk is much less loaded. Today, IP Multicast is mostly used to spread streaming video content and it is not problematic when there are small packet losses (over the unreliable IP-network - e.g. UDP over IP is often used). In our scenario however, reliability is needed otherwise the downloaded file will be unusable. We'll augment the Castgate [3] multicast architecture (which connects multicast-aware network-islands over the internet (layer 3)) with open-source flute [4] implementations (layer 4 to 7) which add reliability to the streams using error correction codes to handle small packet losses. To handle late arrivals and large packet losses, the file can be continuously sent by the sender in a carrousel-mode. Making sure slow receivers are not a bottleneck for fast receivers, the same data can be sent on different "logical tunnels". A fast receiver can join multiple channels in order to reassemble the file faster. Congestion control information can then be sent over a control tunnel. These possibilities are currently integrated and evaluated.

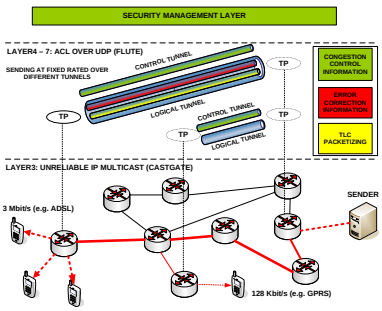

Fig. 4. Architecture for reliable multicast file distribution

IV. Conclusions

We showed that despite hardware-advances, scalability or cost problems to distribute large files under a very large number of users in a small time window (an application that will be emerging in the near feature when e-magazines are coming up) still exist. Therefore we proposed two possible cost-efficient solutions based on existing technology to solve this specific problem.

References

[1] LVS: The Linux Virtual Server project, http://www.linuxvirtualserver.org/
[2] Chord: a scalable peer to peer lookup service for the internet, http://pdos.csail.mit.edu/chord/
[3] Castgate: enabling internet multicast, http://www.castgate.net
[4] MCLv3: an Open Source Implementation of ALC and NORM, http://www.inrialpes.fr/planete/people/roca/mcl/mcl.html