Design and Performance Evaluation of a Mechanism to Share Contents over Broadcast Channels

Dario Di Sorte, Mauro Femminella, and Gianluca Reali

Abstract — In this paper we present a novel mechanism (Wireless Data Sharing, WDS) to share popular contents among second-level proxies connected to a parent proxy through a bandwidth-limited, wireless, broadcast channel. The idea is to publish the transmission of a content requested by a second-level proxy so as to make all the other ones able to capture such a content during the reliable unicast transmission from the parent towards the requesting second-level proxy over the broadcast channel. This would allow users served by these proxies to decrease the time to access contents, without any additional bandwidth consumption. In addition, such an approach would increase channel efficiency and decrease central server load.

We provide a detailed description of the architecture and functions of the WDS system, with explicit reference to the prototype we have implemented. Such a prototype has enabled us to evaluate the performance of the data sharing mechanism from both user and network viewpoints. In more detail, we have carried out a measurement campaign in an IEEE 802.11 environment to get quantitative results about: (i) bandwidth saving, (ii) content access latency, and (iii) signaling overhead.

***Index Terms* — cache pre-fetching, popular contents, broadcast and bandwidth-limited channel, reliable transport.**

I. Introduction

Whenever a given set of contents is particularly popular among different clusters of users and there is a broadcast and bandwidth-limited channel, it could be recommended to deploy a data sharing architecture able to make contents quickly available to all the community.

The network scenario we consider is depicted in Fig. 1. There is a two-level proxy hierarchy, where the first level proxy (parent) is in charge of serving a number of second level proxies (children). Each of them, in turn, serves the requests made by a cluster of users. The parent proxy and the proxy caches can communicate through a wireless broadcast channel (e.g., satellite [11][12], WiMax, WiFi). For instance, we can consider (i) a campus which has different departments distributed over an area which may access the central proxy through either a WiFi or a WiMAX network; (ii) an ISP which has a number of web caches displaced in a very large area connected through a satellite link, (iii) a vehicular scenario (e.g., a train fleet) where the Internet access is offered to travelers through a satellite connection.

The Authors are with Dipartimento di Ingegneria Elettronica e dell'Informazione (DIEI), University of Perugia, via G. Duranti 93, I-06125 Perugia, Italy, e-mail: {disorte,femminella,reali}@diei.unipg.it, Fax: +39 075 5853654.

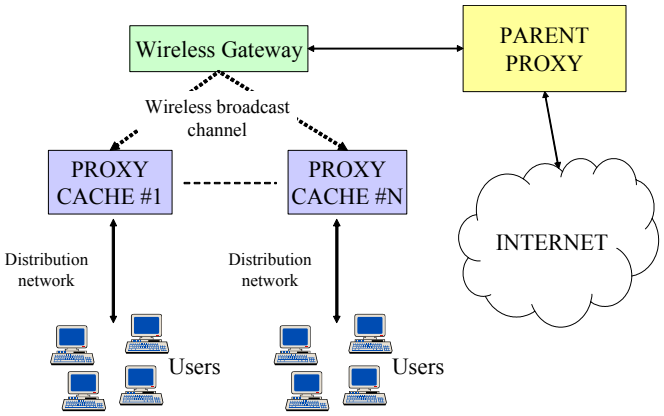

Fig. 1 – Network scenario.

Proxies aim to improve the performance, providing end users with faster access to contents. In fact, by absorbing a portion of the workload, proxy caches can improve client response times as well as increase the capacities of servers and networks, thereby enabling them to serve a potentially larger clientele [7]. In addition, a group of caches co-operating with each other results in a powerful paradigm to improve cache effectiveness. One approach to co-ordinate caches in the same system is to set up a caching hierarchy (e.g., see [8]). With hierarchical caching, caches are placed at multiple levels of the network. When the document is found, either at a cache or at the original server, it travels down the hierarchy, leaving a copy at each intermediate caches along its path. Further requests for the same document travel up the caching hierarchy until the document is hit at some level. As highlighted in [12], with reference to web traffic, there is a large percentage of requests that are not satisfied by caches (miss event), in particular when a small client population is connected to the cache. To reduce the number of miss events it is recommended to preload the cache with new documents, expecting that users are likely to request them. Prefilling caches, however, requires additional disk space and may waste network bandwidth. The disk space is not a problem, since large disks are becoming increasingly cheaper, whereas the bandwidth waste can be a serious problem when the channel is bandwidth-limited and expensive.

The goal of this paper is to present a novel mechanism (Wireless Data Sharing, WDS) to share popular contents among servers over broadcast channels. Our idea is to publish the transmission of a content requested by a WDS-Child

(WDS-C) so as to make all the other WDS-Cs able to capture such a content during the reliable unicast transmission from the WDS-Parent (WDS-P) towards the requesting WDS-C over the broadcast channel. This would allow users belonging to all clusters to decrease the time to access contents and consult hot contents off-line (e.g., during outage periods of the wireless channel). In addition, such an approach would increase channel efficiency, decrease central server load, and emulate reliable multicast procedures.

WDS can be put in operation in a number of application scenarios, and in general whenever there is a central repository of information and a set of user clusters which have to access such an information via proxies connected to the central server through a broadcast channel.

The service architecture deploying the WDS consists of a module on the WDS-P and a module on each WDS-C. The parent is able to generate unicast UDP advertisements upon data requests from a WDS-C. In this way, the system is able to distribute the information needed to enable the data acquisition process by all WDS-Cs through packet sniffing from the active TCP connection between the WDS-P and the requesting child. It is worth noting that, since the wireless channel is a shared resource, this way of prefilling contents does not imply any additional bandwidth consumption. In fact, when a content is being transmitted over the shared channel to the requesting proxy, the channel can not be accessed in any way by other proxies.

Thus, with reference to the scenario shown in Fig. 1, our objective is to prefill all second-level proxies with contents which are considered popular. The popularity is an index associated with an object which identifies its level of importance among a given set of users (i.e., it is a sort of appreciation index). There is a lot of work in the literature on this issue (e.g., see [9]), and this aspect is definitely beyond the scope of this work. The WDS system can foresee a two-level decision. The former may happen at the WDS-P, which, upon a request, may decide to advertise or not the content transmission over the broadcast channel. The latter may be executed at the WDS-Cs which have not requested the content. They may or not execute the capture process according to the set of information included in the UDP advertisement; in principle, this mechanism allows all proxies under the same wireless access network to share the same contents.

It is worth noting that this prefetching approach is quite different from the classic way to intend prefetching (see, e.g., [10]), which consists of downloading documents that will very likely be requested in the near future. More specifically, whenever a block of information is requested, the system can estimate what additional information will be needed in the next few user accesses, and transmit some of them to the local disk beforehand, according to certain criteria. In this way, reduced documents download times are traded for increased bandwidth requirements, which is not recommended in bandwidth-limited scenarios.

To sum up, the contribution of this work is twofold:

- a detailed description of the architecture and functions of the WDS system, with explicit reference to the prototype we implemented. The feasibility "proof" is obtained by designing and implementing a true prototype of the WDS system based on the well-known SQUID proxy cache [3]. The reference scenario is an IEEE 802.11 access network [4];
- the WDS prototype has enabled us to evaluate the performance of the data sharing mechanism from both user and network viewpoints. In this regard, the Byte Hit Rate (BHR) represents the percentage of the traffic volume directly served by the proxy without contacting the remote, origin server. This performance figure is strictly related to both bandwidth saving and content access latency. Also, the amount of signaling data associated with UDP advertisement transmission can be considered a cost of the WDS procedure in terms of bandwidth consumption. Thus, we have carried out a measurement campaign in an IEEE 802.11 environment to get quantitative results about: (i) bandwidth saving, (ii) content access latency, and (iii) signaling overhead.

The paper is organized as follows. Section II describes the WDS systems in both architectural and functional aspects. Section III first presents the test-bed used to evaluate the performance of the WDS approach in an IEEE 802.11 environments and then the relevant quantitative results. Finally, in Section IV, we draw our conclusions.

II. Wireless Data Sharing Mechanism

Our purpose is to enable the prefetching of contents into WDS-Cs. In other words, the objects sent on the wireless broadcast channel upon a specific request by a WDS-C may be caught also by those proxies which have not made such a request. It is worth noting that we consider an object any cacheable files, i.e., static documents such as html pages, images, documents, videos and so on.

When a given WDS-C, say X, requires a document, it has to completely receive the object, thus the dialogue between X and the WDS-P has to be reliable and TCP-based, whereas the other WDS-Cs may also catch partially the object. Thus, the aim of the transport mechanism is to have a reliable channel only towards the WDS-C that issued the request, whereas the other "receivers" may decide to either complete separately the download in background or discard the fragments of the received object. In more detail, when a client of X issues a request, if X does not currently store such a content, in turn, X has to issue the request towards the WDS-P, which is in charge of retrieving the document from the remote, origin server, if the object is not locally stored in the WDS-P. At this time, assuming to have the object stored, WDS-P has to transfer it in a reliable way towards X. In addition, the WDS-P has to allow all the WDS-Cs but X to capture over the broadcast channel this new object, in order to potentially increase their cache BHR and decrease the latency towards their clients. However, the copy of the object at the other proxy WDS-Cs may be incomplete due to transmission errors on the wireless channel. This implies that another mechanism has to be designed, to

allow the completion of partially stored documents.

With respect to reliable multicast transport techniques (e.g., see [13]), the WDS data sharing solution has two main advantages. The former is that it is more flexible in the sense that there is the possibility to decide locally at each WDS-C if to acquire or not a content which is going to be transmitted over the broadcast channel. In fact, as mentioned above, in principle each WDS-C may or not execute the capture process. The decision can be taken on the basis of the set of information included in the UDP advertisement and of the current, local state of the proxy. The latter advantage is that in case of wireless technology with multi-rate capability such as 802.11, the multicast traffic has to be transmitted at the lowest rate among the set of the supported rates.

As for security issue, authentication and confidentiality can be ensured by a security protocol (at link, IP, or application layer) among the proxies involved in the architecture.

From now on, we especially refer to our implemented prototype, which uses HTTP/1.1 [2] and TCP to transfer data between proxies, and SQUID as proxy cache [3]. SQUID is a high performance proxy server with a large number of advanced services, with a high degree of configurability. In addition, SQUID provides exhaustive statistics for monitoring the performance of the proxy. These aspects have lead to the choice of SQUID for the prototype.

In order to explain the design of the system prototype, in subsection II.A we provide an overview of SQUID and a short description of internal procedures. Then, in subsection II.B, we describe in details the WDS system.

II.A Basic SQUID operations

The SQUID proxy is mainly composed of three modules: the Client Side, the Storage Manager, and the Server Side[1]:

- Client Side: this module communicates with its own clients, both at socket and at HTTP level (by processing HTTP requests and providing the replies). At the socket layer, it accepts and analyzes the new incoming TCP connections, and then sends the requested data over them. The information relevant to each connection is stored in a data structure named `ConnStateData`, whereas the information relevant to each request (at socket layer) is maintained in the structure `clientSocketContext`: in fact, with HTTP/1.1, it is possible to have more HTTP requests over a single TCP connection. The incoming requests from sockets are processed, in order to decide if, e.g., to forward or not them to other proxies. The relevant temporary data and these processing data are stored in the structure `clientRequestContext`. In this way, it is possible to decide if generating an event of HIT (object in cache), REFRESH (object update), MISS (object not in cache), etc. This process implies to inquire the Storage Manager module and to store the information about the status of each request in the `clientReplyContext`.
- Storage Manager: it is the element connecting the Client Side and the Server Side. For each object stored in the proxy, and for each request forwarded outside, a structure `StoreEntry` is created (containing the object and a set of further information, including those for indexing purposes). During the access to the `StoreEntry`, the proxy generates another structure, `MemObject`, in order to take into account all the clients concurrently requiring the object. The Client Side requests are associated with the corresponding `StoreEntries` through the `MemObjects`, in order to allow the notification of the responses arrivals. This module performs also an indexing operation, thus allowing high speed data access also in case of large amount of data.
- Server Side: this module forwards requests to other servers, on the basis of the proxy configuration: requests for missing objects may be forwarded to either the origin server or other proxies.

When a client requests an object from the proxy (see Fig. 2), it controls if it is stored in its cache. To this end, it searches the structure `StoreEntry` relevant to such an object. In case of a cache MISS, the proxy forwards the request outside, and an empty `StoreEntry` is created. A `MemObject` relevant to the requesting client is created as well. All the clients asking for such an object during the object download are added in the same `MemObject`. When TCP segments arrives to the proxy, the proxy performs an append of these data in the relevant `StoreEntry`. When the download from the external server is completed, the proxy satisfies the requests of the clients registered in the relevant `MemObject`, then the proxy discards this structure; now the object is in the `StoreEntry` and thus in the cache.

The SQUID proxy may support discovery and retrieval of documents from neighboring caches through the Internet Cache Protocol (ICP) [6].

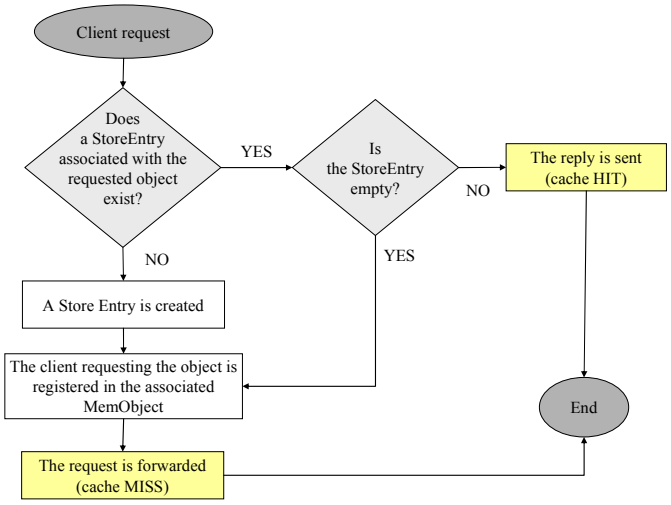

Fig. 2 – Flow diagram for SQUID operation.

[1] There are also other modules in charge of performing functionalities such as users authentication and I/O with memory units. However, a more complete description of all the proxy functionalities is beyond the target of this work.

II.A WDS procedure

The overall procedure to share a content requested by a user to a WDS-C is described in the following steps (see Fig. 3):

- a WDS-C sends a HTTP request to get a specific object from the WDS-P (Fig. 3.a);
- the WDS-P verifies if such an object is locally available (if not, it is in charge to retrieve it from an external server). Before the transmission of the object to the requesting WDS-C through the broadcast channel, it sends an advertisement UDP message to all WDS-Cs using the broadcast address (Fig. 3.b), containing the characteristics of the object that it is going to send: URL, dimension, field "last modified", details regarding the TCP connection over which the object will be transmitted, source and destination IP addresses. To this end, a specific module at application layer has to be implemented in the WDS-P;
- upon the reception of UDP advertisement, each WDS-C controls if the object is locally present and, in the negative case, if there is enough memory space to store it. Consequently, each WDS-C decides to enable or not the capturing process. In case of a MISS, it creates an empty `StoreEntry` (it has to be created, in order to catch eventual other request from other clients) and set up a filter to sniff the specific TCP connection. Clearly, the WDS-C that has explicitly requested the object will ignore the UDP message. To process the advertisement, a specific module at application layer has to be implemented in WDS-C;
- the data exchange between the WDS-P and the requesting WDS-C will proceed with classic HTTP over TCP, whereas, for what concerns the other WDS-Cs, they have to be able to sniff TCP segments relevant to the TCP connection between the requesting WDS-C and the WDS-P (Fig. 3.c). For this purpose, a suitable capture module must be envisioned and exploited in WDS-C. This way of operation allows prefetching, in the best case, the whole object with no additional cost in terms of shared wireless resource. In other words, no additional TCP traffic on the broadcast channel is required. Obviously, since the wireless channel could be characterized by a high bit error rate, non-requesting WDS-Cs could correctly receive only a part of the web object. If errors are detected in the received data, then the relevant segments will be discarded and the stored object will be incomplete (partial download). In order to be able to recover the object without starting the download from scratch, it is necessary to maintain a record for the object in which the missing parts of the document are indicated. It is worth noting that this does not imply an important increase of complexity because, in any case, each standard cache module must be able to maintain a record for each stored object for the purpose of managing it;
- at the end of the object transfer, each non-requesting WDS-C will have in the cache a (possibly) incomplete object. A strategy to manage incomplete objects is outlined below and based on the use of a threshold *Th*:
  - if the fraction of the object present in the cache is lower than *Th*, then the fragments of the object correctly received are discarded, because the advantages of maintaining them in the cache may be considered negligible;
  - if the fraction of the object present in the cache is higher than *Th*, then the missing part of the object is immediately requested by the relevant WDS-C on a TCP connection (Fig. 3.d). For this purpose, it is necessary to develop a module on the WDS-C able to drive HTTP/1.1 to request part of a object. Clearly, in this case, the WDS-P transmits data on the TCP connection without any advertisement;
  - if, in the meanwhile, a user has requested the incomplete object, then, independently of the amount of data correctly captured, the missing part of such an object is requested to the WDS-P (Fig. 3.d).

To sum-up, the main system modules to implement are:

- WDS-P: a module able to generate multicast UDP advertisements to be sent to proxy WDS-Cs;
- WDS-C: a module able to:
  - catch and understand advertisements from the WDS-P;
  - capture TCP packets;
- WDS-C: a module able to
  - request to the WDS-P, through a standard TCP connection, only specific fragments of an object;
  - to forward the entire object to the Storage Manager, thus emulating a classic, complete object download. In fact, in current implementation, the SQUID Storage Manager does not support the partial memorization of objects, and, due to the complexity of the functional upgrade, our choice is to introduce a so-called adaptation module to avoid implementing this feature directly into SQUID. It is worth noting that such an operation is performed only when the proxy completes the object download.

The complete set of operations at WDS-P and WDS-Cs are described by the flow diagrams shown in Fig. 4 and Fig. 5, respectively. Note that we have assumed that the WDS-P publishes all contents, and WDS-Cs capture all contents transmitted in the channel. From the protocol viewpoint, these modules act as an adaptation layer in the proxy protocol stack. In this regard, Fig. 6 and Fig. 7 show the protocol stack of WDS-P and WDS-C, respectively.

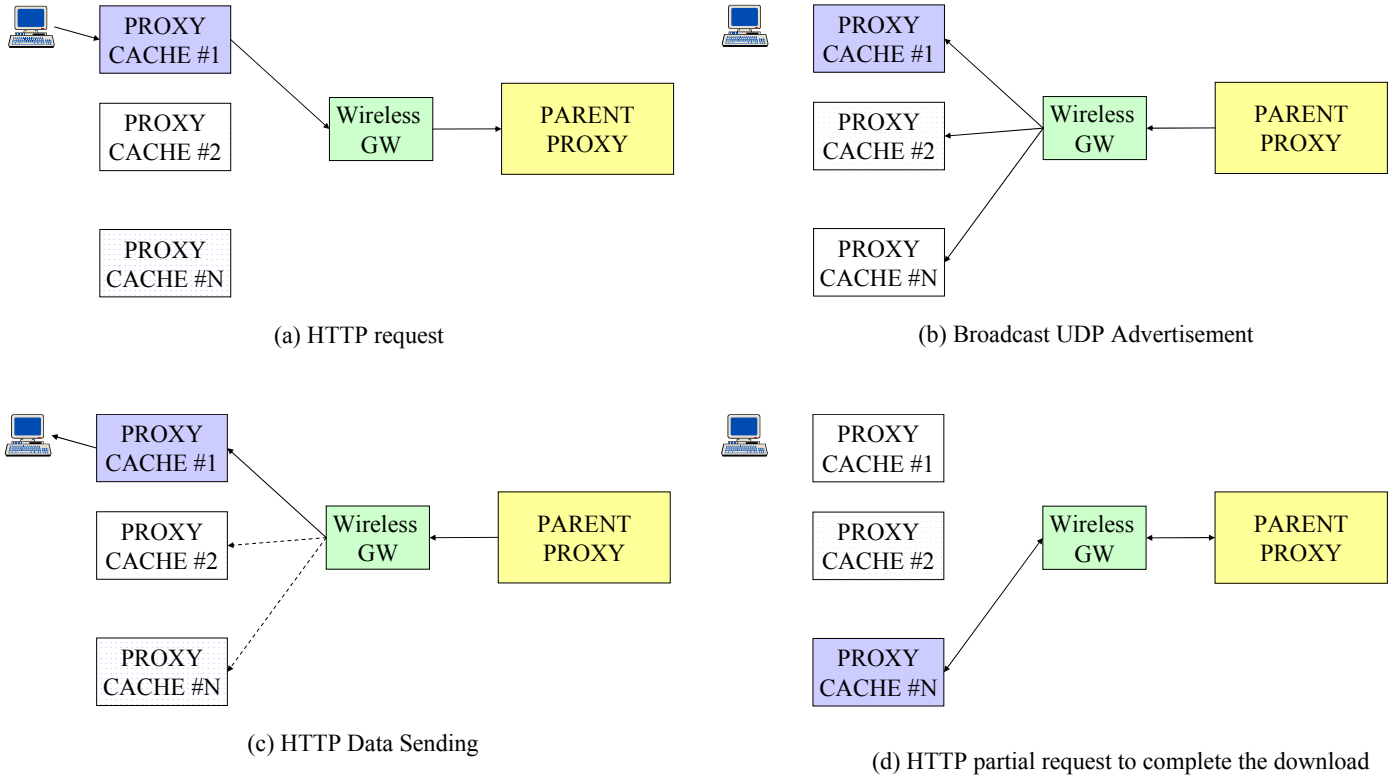

Fig. 3 – WDS procedure.

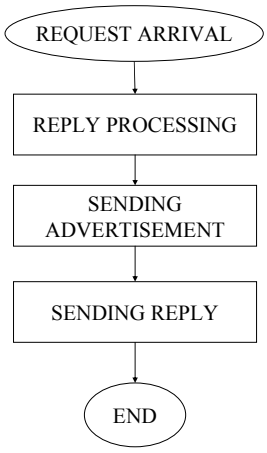

Fig. 4 – WDS-P operation.

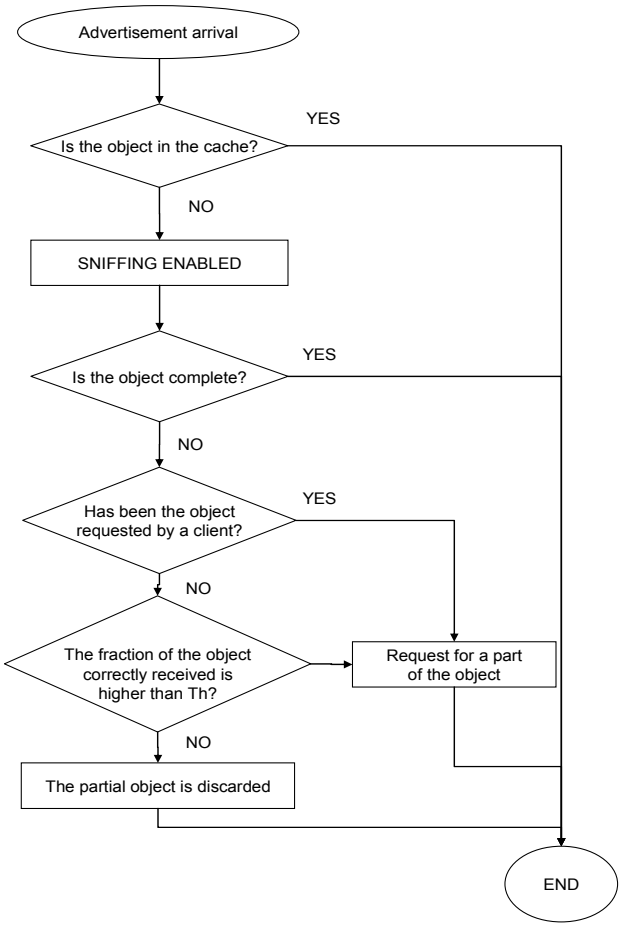

Fig. 5 – WDS-C operation.

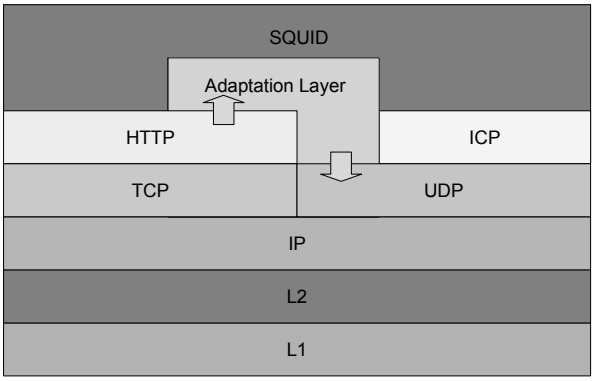

Fig. 6 – Protocol stack in the WDS-P.

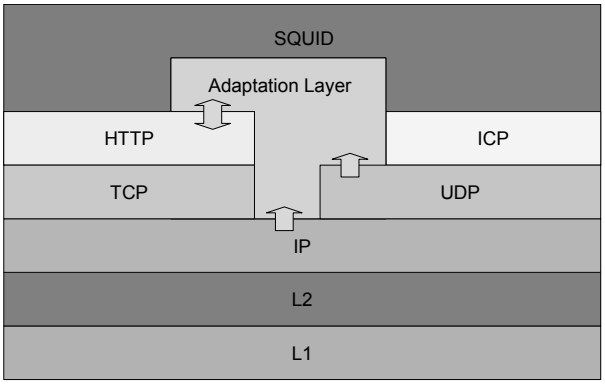

Fig. 7 – Protocol stack in the WDS-C.

III. Experimental Performance Evaluation

In this Section, we first describe the test-bed in an IEEE 802.11b environment, and then we report the results from a measurement campaign, the goal of which is to assess the WDS performance and compare them with a legacy scenario. In more detail, we investigate the following performance figures:

- download time reduction with respect to the legacy scenario;
- Byte Hit Rate (BHR) increase, which is strictly related to wireless bandwidth saving, with respect to the legacy scenario
- signaling overhead of the overall process with respect to the legacy scenario.

IV.A Test-bed setup

Fig. 8 depicts the architecture of the test-bed set-up in our laboratory to perform measurement. It consists of:

- a WDS-P connected through a 100Mbps LAN to the Internet and to an IEEE 802.11b Access Point (AP);
- two WDS-Cs; one (WDS-C_{Req}) is used to request new contents, the other (WDS-C_{Capt}) is in charge to capture contents requested by the other one. It is worth noting that the cluster of users connected to a WDS-C is emulated by a single user which browses contents directly on the WDS-C machine (see the details about the configuration of browsers and proxies in Fig. 9). Both WDS-Cs are close to the AP and are connected with the maximum rate (11 Mbps);
- an IPerf wireless station acting as source of traffic, and an IPerf wired station acting as traffic sink. IPerf is a tool to generate traffic [5]. We have used it to vary the wireless network load.

It is worth noting that the processes implementing the WDS-P and the WDS-C systems demand a low amount of computing resources.

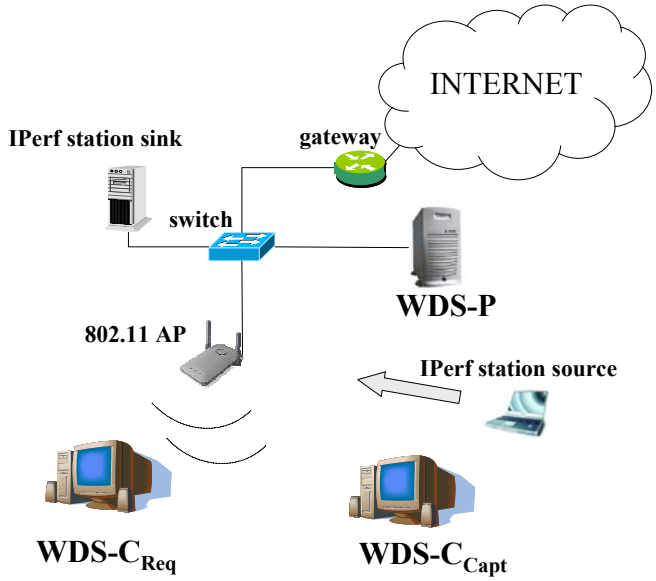

Fig. 8 – Demo scenario architecture.

Fig. 9 shows some details about the main protocol exchanges that occur upon a user request which cannot be satisfied directly by the queried WDS-C (see also Fig. 3).

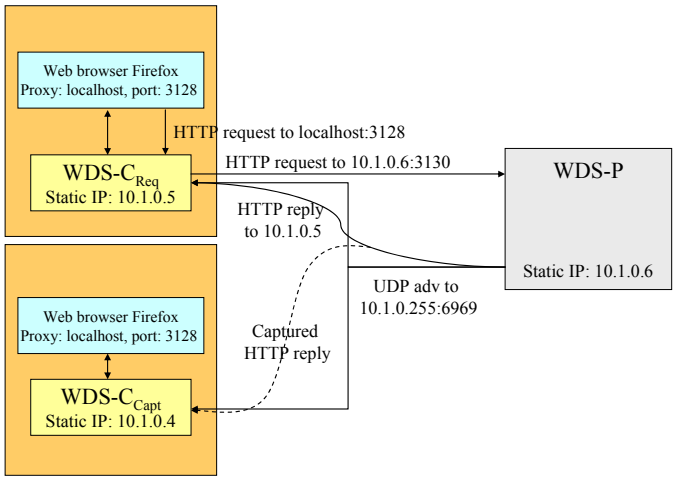

Fig. 9 – Demo configuration and protocol exchanges.

We have performed the following trial. The user on the WDS-C_{Req} requests a given set of contents. Then the other user on the WDS-C_{Capt} requests the same set of contents. In this way we are able to evaluate an upper bound to the performance of the WDS system in an IEEE 802.11 environment. In fact, we have assumed that the clusters of users served by the two

WDS-Cs request the same contents.

Thus, the reduction of the download time perceived by the user at the WDS-C_{Capt} can be evaluated by comparing his/her average download time with respect to the nominal latency; the nominal latency corresponds to the download time for the client issuing the request at the WDS-C_{Req}, because it implies the relevant proxy to issue the request to the parent. Thus, this first download is performed as if the WDS mechanism was not implemented.

As for the BHR, from the above considerations, it should be clear that the BHR of WDS-C_{Req} is equal to zero, whereas the BHR evaluated at the WDS-C_{Capt} should be definitely higher.

All these data can be inferred by a suitable processing of the SQUID log files.

We have considered different user profiles:

- classical web browsing (denoted as WEB profile);
- document search and download (denoted as MIX profile);
- direct document download from an a-priori known server (denoted as DOC profile);
- direct MP3 files download from an a-priori known server (denoted as MP3 profile).

Clearly, passing from the WEB profile to the MP3 profile, the average object size increases, and thus we expect better WDS performance in terms of latency reduction, BHR, and overhead.

IV.B Numerical results

In order to evaluate the performance of the WDS system, we have performed a number of trials with different sets of objects and all figures show the average results. The confidence intervals are very small and thus we have omitted them to improve the neatness of figures.

Now, let us start evaluating the benefit of the WDS system.

As first performance figure, we consider the BHR, which is strictly related to the amount of wireless bandwidth saving, since it represents the amount of bytes which can be served directly by the WDS-C proxy without issuing a request to the parent through the wireless channel.

Fig. 10 depicts the BHR as a function of the threshold parameter, *Th*, which ranges between 0.25 and 1, with the value of the network load as a parameter. The plot is relevant to the download of web pages (WEB profile) when the traffic varies from 0 to 4 Mbps. The behavior of the BHR depends on two variables: the *Th* and the network load. As for the former, it is worth noting that, if *Th*=1, only the objects which have been completely and correctly captured are stored into the cache of the WDS-C_{Capt}, whereas if some fragments are corrupted, the overall object is discarded from the cache. When the value of *Th* decreases below 1, additional objects can be stored into the cache. These objects are those for which the percentage of correctly captured bytes is higher than or equal to *Th*. Thus, the lower the value of *Th*, the higher the BHR. However, it is worth noting that if the number of correctly received bytes is higher than *Th,* and some fragments are corrupted, the object completion process requires the usage of additional wireless bandwidth (see Fig. 3.d). This operation could be executed, for instance, when WDS-C realizes that there is no intense activity on the shared channel. Thus, the choice of the value of *Th* is a trade-off between wireless bandwidth usage and download latency, since, if an object is already stored into the WDS-C, the service time towards the user will be definitely lower than the legacy case.

Finally, note that values ranging from 35% to 55% are justified by the presence of non-cacheable objects in web pages.

As for the network load, clearly, the higher the network load, the lower the amount of captured data for a given value of *Th*. In fact, additional load over the wireless channel increases the probability that a fragment received at the WDS-C_{Capt} is corrupted and thus discarded.

Clearly, if the fragment received by the WDS-C_{Req} is corrupted, this is requested again through the standard procedures of the TCP protocol, since the communication between the WDS-C_{Req} and the WDS-P is reliable.

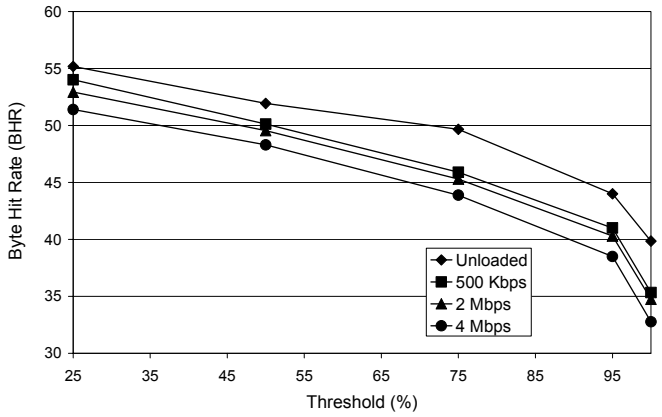

Fig. 10 – Byte Hit Rate for the WEB profile with traffic load as a parameter.

Fig. 11 shows the BHR as a function of the threshold value with the network load as a parameter for the DOC profile. Clearly, for what concerns the curve trends, the comments made for the WEB profile apply also in this case. However, the main difference is in the values of the BHR. In this case, the absence of non-cacheable objects allows to reach BHR values ranging from 80% up to 99.5%. In addition, in this case the effect of a threshold value lower than 1 is more evident than the WEB profile case. In fact, up to *Th*=0.95, the BHR decreases with *Th* almost linearly and for *Th*=1 there is a sudden decrease of the BHR. This is due to the fact that, even if a single fragment of a large document is not captured or is corrupted, the overall object is discarded. In a wireless channel, the probability of this event is not negligible and increases with the traffic load.

Fig. 12 depicts the values of the normalized latency as a function of the threshold parameter (*Th*) with the value of the network load as a parameter. The normalized latency is defined as the ratio between the average download time for the WDS system and the one for the legacy system. The plot is relevant to the DOC profile. As expected on the basis of the comments already done for the BHR, the lower the value of the threshold

and the higher the gain of the WDS system with respect to the legacy one. Clearly, higher gains mean lower values of normalized latency. The performance improvement obtained by the WDS system decreases with the network load. However, also in the worst case (*Th*=1 and network load equal to 4 Mbps), anyway there is a gain around 20% for the average download time.

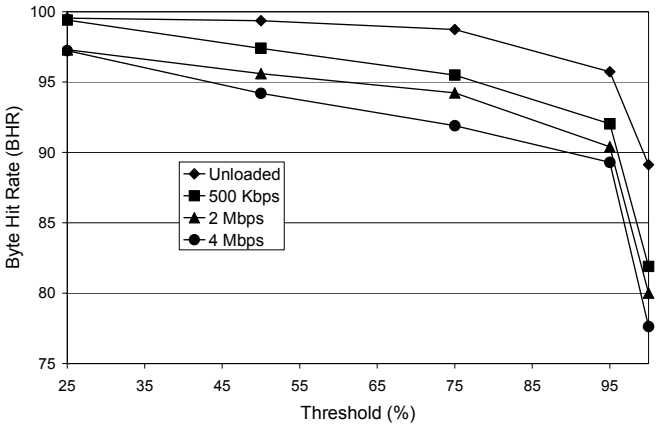

Fig. 11 – Byte Hit Rate for the DOC profile with traffic load as a parameter.

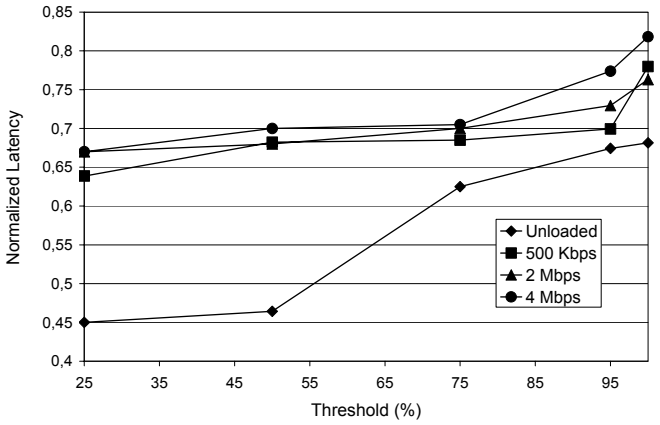

Fig. 12 – Normalized download time for the DOC profile with traffic load as a parameter.

Now, let us start evaluating the cost of the WDS procedure. Fig. 13 shows the histograms of the percentage overhead versus the user profile. The percentage overhead is defined as the amount of network resources used to advertise the transmission of a given content on the broadcast channel divided by the size of the content. Since the numerator of this ratio includes not only the advertisement body, but also all protocol layers headers (from the physical layer of IEEE 802.11 up to the UDP header), this quantity represents the percentage amount of signaling per byte of content. Thus, this is an *overestimation* of the ratio between the network resources needed to transmit the advertisement and those needed to transfer the object. Since this quantity does not take into account the amount of network resources to perform an (eventual) object completion process, the percentage overhead can be understood as the upper bound to the normalized signaling overhead when *Th*=1.

Fig. 13 shows that, when the average object size is small (WEB profile), the percentage overhead is around 1-2%, whereas it decreases very quickly for higher values of the object size (please note that the ordinate is plotted with the log scale). For the MP3 profile, where the average objects size is in the order of few Mbytes, the normalized overhead is around 10^{-5}, thus it is really negligible.

In any case, we can conclude that the cost of the WDS process in terms of bandwidth consumption, which is low, can be justified by the performance improvement in terms of BHR and download time.

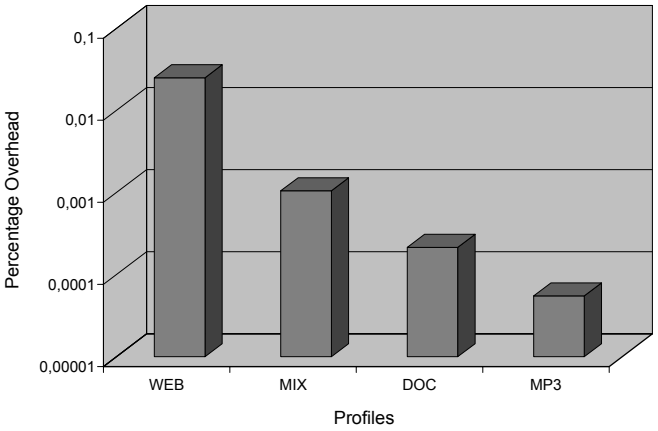

Fig. 13 – Signaling overhead.

IV. Conclusion

In this paper, we have presented a mechanism to share popular contents over a broadcast wireless channel. Its main feature is the capability to capture contents when they are transmitted over the shared, wireless medium, on the basis of the information previously advertised through a small UDP packet. In principle, this strategy allows a set of proxy children to be a replicated copy of the proxy master, without wasting precious wireless resources.

We have described both the main functional entities and the protocol exchanges among them. Then, we have presented a prototype implementation of the system, which is not only a "proof of concepts" of the proposed mechanism, but also the test-bed used to perform experiments to investigate its effectiveness. The obtained results show that the system allows a noticeable improvement in terms of BHR and download time reduction, in spite of an almost negligible signaling overhead.

References

[1] J. Wang, "A survey of web caching schemes for the Internet," ACM SIGCOMM Computer Communication Review, 29(5), October 1999.

[2] J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, R. Fielding, J. Gettys, "Hypertext Transfer Protocol-HTTP/1.1," IETF RFC 2616, June 1999.

[3] The Squid Web Proxy Cache, http://www.squid-cache.org/.

[4] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Standard 802.11, 1999.

[5] Iperf network performance tool, http://dast.nlanr.net/Projects/Iperf/.

[6] D. Wessels, K. Claffy, “Internet Cache Protocol (ICP), version 2,” IETF RFC 2186, September 1997.

[7] M. Raunak, P. Shenoy, P. Goyal, K. Ramamritham, P. Kulkarni, ”Implications of Proxy Caching for Provisioning Networks and Servers,” IEEE JSAC, 20(7), September 2002.

[8] H. Che, Y. Tung, Z. Wang, ”Hierarchical Web Caching Systems: Modeling, Design and Experimental Results,” IEEE JSAC, vol. 20, N. 7, September 2002.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web Caching and Zipf-like Distributions: Evidence and Implications,” Proceedings of IEEE Infocom '99, New York, USA.

[10] Z. Jiang and L. Kleinrock, “An Adaptive Network Prefetch Scheme,” IEEE JSAC, 16(3), April 1998.

[11] A. Armon, H. Levy, “Cache Satellite Distribution Systems: Modeling, Analysis, and Efficient Operation,” IEEE JSAC, 22(2), February 2004.

[12] P. Rodriguez, E.W. Biersack, “Bringing the Web to the Network Edge: Large Caches and Satellite Distribution,” Mobile Networks and Applications, 7, Kluwer, 2002.

[13] B. Adamson, C. Bormann, M. Handley, J. Macker, “Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol,” IETF RFC 3940, November 2004.