REfactor-ing Content Overhearing to Improve Wireless Performance

Shan-Hsiang Shen, Aaron Gember, Ashok Anand, and Aditya Akella
University of Wisconsin, Madison
{shan-hsi,agember,ashok,akella}@cs.wisc.edu

ABSTRACT

Many systems have leveraged the broadcast nature of wireless radios to improve wireless capacity and performance. While conventional approaches have focused on overhearing entire packets, recent designs have argued that focusing on *overheard content* may be more effective. Unfortunately, key design choices in these approaches limit them from fully leveraging the benefits of overhearing content. We propose a cleaner refactoring of functionality wherein overhearing is realized at the sub-packet payload level through the use of IP-layer redundancy elimination. We show that this dramatically improves the effectiveness of prior overhearing based approaches and enables new designs, e.g., enhanced network coding, where content overhearing can be more effectively integrated to improve performance. Realizing the benefits of IP-layer content overhearing requires us to overcome challenges arising from the probabilistic nature of wireless reception (which could lead to inconsistent state) and the limited resources on wireless devices. We overcome these challenges through careful data structure and wireless redundancy elimination designs. We evaluate the effectiveness of our system using experimentation on real traces. We find that our design is highly effective: e.g., it can improve goodput by nearly 25% and air time utilization by nearly 20%.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design, Performance

Keywords

Wireless Networks, Redundancy Elimination, Throughput

1. INTRODUCTION

A common issue in wireless networks is severely constrained throughput performance, especially when link quality is poor or node density is high. An important class of techniques that aim to improve wireless performance are those based on wireless overhearing. These techniques use the fact that wireless radios can opportunistically overhear packet transmissions, which can be leveraged in a variety of ways: e.g., nodes can suppress unnecessary transmissions (e.g., RTS-id [3]), make better forwarding decisions (e.g., ExOR [10]) or perform network coding (e.g., COPE [20]).

Conventional overhearing based approaches all rely on overhearing packets (both headers and payloads) in full. In contrast, more recent overhearing approaches argue that a *content-centric* design can lead to even better capacity improvements. Focusing on the content being overheard can facilitate, for example, suppression of duplicate data across different transfers, which is not possible in the packet-centric approaches. Ditto, the first such content overhearing system [13], leverages overhearing at the granularity of data chunks roughly 8 to 32KB long and offers nearly an order of magnitude better throughput in multi-hop mesh networks compared to traditional forwarding.

Unfortunately, existing approaches do not go far enough in leveraging the full benefits of content overhearing. The main reasons for this are: (*i*) the granularity at which content overhearing operates and (*ii*) how content overhearing is implemented. For instance, in Ditto, content overhearing is implemented at the granularity of 8-32KB chunks, using an unconventional pull-based transport. This prevents Ditto from being applicable to short flows and flows with dynamic content, which make up a significant fraction of Web and enterprise flows [8, 15] and are typical of request-response applications. Additionally, Ditto caches content only at mesh nodes, not wireless clients, providing no benefits over the last hop wireless link. We discuss these and other drawbacks in greater detail in §2.

We argue that a re-factoring of content overhearing is necessary to realize the full benefits. In this paper, we present a system called REfactor that pushes content overhearing lower down the stack to enable fine-grained overhearing, specifically through intelligent use of recent *IP-layer redundancy elimination* (RE) [7] technology. In traditional RE, a wired transmitter removes duplicate strings of data (as small as 32-64B) from individual packets by comparing them against prior packets; the receiver reconstructs full packets from a local cache of prior packets. REfactor shows how this idea can be generalized to a wide range of wireless settings, including infrastructure-based, mesh, and ad-hoc networks.

IP-layer RE has recently been leveraged in the context of cellular wireless networks, providing up to 60% bandwidth savings on last hop cellular links [22]. However, REfactor takes RE a step further by adding the benefits of content overhearing, a feature not currently available in commercial cellular networks. REfactor is able to exploit the inherent fine-grained redundancy known to exist at the sub-packet (or "packet chunk) level, not only *within* a single

MobiCom'11, September 19–23, 2011, Las Vegas, Nevada, USA.

client, but also *across* multiple clients [8]. Additionally, we address many of the challenges introduced by the wireless domain, discussed below, in a manner that provides additional benefits, rather than seeking solely to minimize overhead [22].

Enabling RE-based content overhearing in a range of wireless settings introduces many key challenges. Whereas in conventional wired RE approaches the sender and receiver caches are tightly synchronized, the probabilistic nature of wireless overhearing and the possibility of a receiver overhearing from multiple transmitters mean that sender and receiver caches are almost guaranteed to be out of sync, which significantly impacts the correctness and performance of RE. Furthermore, wireless nodes are often memory and CPU constrained; hence we need new light-weight RE designs.

We develop novel data structures to overcome these challenges. We present the notion of *self-addressing packet chunks*, which allows us to track cache residence in a consistent and low overhead way across an entire wireless deployment and vastly simplifies redundancy encoding and decoding. We also design simple approaches for estimating reception probabilities, which we then employ in a *model-driven* fashion to decide whether it is worth removing duplicate bytes. We find that the model-driven approach, coupled with the fact that our design leverages many possible opportunities for overhearing content, make REfactor reasonably robust to errors in reception probability estimation, which is a notoriously hard problem. This makes REfactor easy to use in practice.

We find that our refactoring has the obvious effect of significantly improving and broadening the effectiveness and applicability of content overhearing and redundancy elimination in wireless settings. Emulation experiments using our prototype written in Click [21], show that REfactor can improve goodput in infrastructure wireless networks by nearly 25% and utilization by 20%. REfactor's goodput improvements are not just the result of simply removing repeated chunks from packets: REfactor's focus on packet chunks provides more opportunities for overhearing, and the smaller packets REfactor creates have much lower packet error probability—imposing 7-27% fewer packet losses. We find that model-driven RE is quite beneficial, whereas blindly applying RE on all packets can result in a drop in goodput. REfactor can tolerate up to 20% error in overhearing probability estimation. Additionally, our self-addressing chunks approach offers high speed operation (our prototype, e.g., offers up to 0.8Gbps in software) while requiring modest sized caches (64-256MB) and ensuring effective duplicate removal (75% of optimal). Detailed results are in §5.

REfactor has important architectural implications as well. In particular, it substantially enriches various existing overhearing-based approaches and enables new ones. For instance, REfactor can be easily combined with COPE [20] and mesh routing techniques [12] to improve network capacity (see §2 for detailed examples). Combining our REfactor prototype with COPE [20] improves utilization by 3-14% compared to using just COPE.

2. BACKGROUND AND MOTIVATION

Traditional overhearing-based approaches to improving wireless network capacity and throughput have relied on packets being overheard in full. For example, RTS-id [3] adds a special ID field to an RTS packet to allow receivers to determine if they recently overheard a packet, thereby avoiding transmission of the packet. In contrast, some recent approaches have argued that shifting the focus from packets to *content* can result in substantial throughput and capacity improvements. Ditto [13] was the first system based on this notion of *content overhearing* (as opposed to the conventional packet overhearing ideas). Ditto functions on named data chunks that are independent of packets. Wireless mesh routers in Ditto cache directly received chunks and chunks reconstructed from overheard packets. When a client requests a particular chunk, Ditto attempts to serve the request from a upstream wireless mesh router, avoiding the need to transfer a chunk all the way from the mesh gateway to the client.

2.1 Limitations of the State-of-the-art

In what follows, we argue that Ditto's approach does not leverage all redundancy opportunities, and its narrow focus limits its applicability to a variety of practical scenarios.

Limitations due to large chunks. Ditto names data chunks of size 8KB or larger. This leads to two problems: First, Ditto fails to identify finer-granularity content overlap across network flows. In fact, recent studies have shown that a major portion of redundancy in Internet traffic arises from overlapping chunks as small as 64B in size [8]. Second, many nodes may not overhear a large chunk in full and may fail to reconstruct it. Indeed, experiments using Ditto show that, on average, 75% of the potential locations for overhearing in a campus testbed could not completely overhear, and failed to reconstruct, almost 50% of chunks [13]. A recent RE framework for cellular networks operates on chunks as small as 8B, enabling redundancy removal within a client's traffic at fine granularities [22]. However, applying this system to other wireless scenarios (such as those in §2.2.2) misses out on overhearing opportunities to remove redundancy between clients.

Limitations due to pull-based transport. Ditto's reliance on named data chunks, each of which spans several packets, forces it to use an alternate transport protocol instead of using TCP end-to-end. Specifically, Ditto uses a pull-based transport protocol, DOT [27], where remote servers send chunk IDs to clients, who then request them one after the other; requests may be opportunistically served by a local cache. This leads to several problems: First, it requires chunk identifiers to be known beforehand. This works for static content but not for dynamically generated content; clients are forced to use default transport designs for dynamic content, removing the opportunity for performance improvements. Second, applications with with short messages—e.g., gaming flows, twitter feeds, several request-response applications, short HTTP flows, etc.—may actually observe a degradation in performance in the average case (because the pull-based approach invariably adds additional RTTs). Lastly, no performance benefits are offered on last hop wireless links. Chunks must be transferred in full across the last link from mesh router to client.

Another set of popular approaches for improving wireless capacity are those based on network coding [11, 20]. As the authors of Ditto mention, it may be possible to use opportunistic content-overhearing to augment coding and improve its overall effectiveness. However, given the mismatch in the granularities and transport models used in Ditto and prior coding approaches, it is unclear if the synergy between overhearing and coding can be exploited.

2.2 REfactor

Our paper shows that a careful re-factoring of content overhearing can address the problems above optimally and dramatically improve wireless capacity and performance. We argue for pushing content-awareness "lower down the stack" through the use of *IP-layer packet caches that perform redundancy elimination* (RE) [7]. Packet caches can be used to suppress byte strings that have appeared in earlier overheard packets both within and between clients. We refer to our approach as REfactor. The cleaner re-factoring in REfactor offers many benefits:

- IP layer RE can remove duplicates as small as 64B in an application-agnostic fashion, even from dynamically generated

content. REfactor benefits applications with short flows—even those lasting a single packet—which are common in enterprise settings [15]. Thus REfactor leads to more effective overhearing-based designs.

- REfactor requires small IP-layer modifications and retains the conventional push based model of content dissemination that is prevalent today.
- Because REfactor leverages all possible opportunities for overhearing, it's performance is reasonably robust to errors in some aspects of the design (in particular, reception probability estimation, which is a notoriously hard problem). Thus, it is easy to use in practice.
- REfactor leads to smaller packets which consume less bandwidth and suffer lower loss rates. Operating on packets also allows REfactor to run at very high speeds: As we show in §5, our prototype offers 0.6-0.9Gbps.
- REfactor can be applied transparently in a variety of scenarios, including wireless infrastructure-based and peer-to-peer communications, and opportunistic routing in multi-hop mesh networks. In particular, REfactor is more directly aligned with packet-based coding approaches. Hence it provides practically viable opportunities to enhance network coding to further improve network capacity.

2.2.1 REfactor Overview

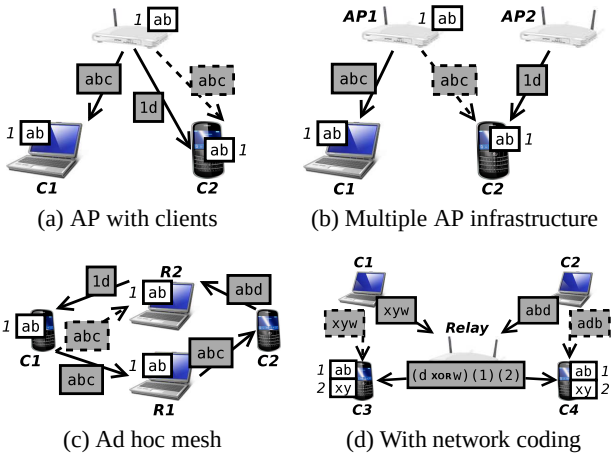
We start with a common scenario where REfactor can be used: an AP operating in infrastructure mode with some number of associated clients. The AP and clients can overhear and cache packets. When the AP receives a packet from the wired network, it scans the content for duplicate strings of bytes that appeared in earlier packets. The AP then calculates the expected benefit for the receiving client from performing RE on the packet, which depends on the AP's estimate of whether the receiving client is likely to have cached the relevant earlier packets, either from transmissions to the client or from overheard transmissions to some other client. If the likely benefit is high, the AP "encodes" this packet, i.e., removes the duplicate bytes and inserts a shim instead. The shim contains a pointer to a memory location in the client and allows the client to reconstruct the original packet using its local cache. Thus, if the receiving client has the content pointed to by the shim, then it can decode the packet. However, if the content is not cached in the client, the client needs to request the missing content, incurring additional transmissions. This penalty is imposed when the AP's estimate of whether the client has the content is incorrect.

In Figure 1a, we illustrate the benefit REfactor offers in this scenario. The transmission of the packet payload `abc` to *C1* is overheard by *C2*, and the chunk `ab` is added to all caches. The data `abd` is sent to *C2* via a packet with a shim header "`1`" plus the non-redundant data `d`. Because *C2* overheard and cached the chunk `ab`, it as able to reconstruct the full packet using the cache entry specific in the shim header. The reduction in the size of the second packet transmission improves overall network throughput.

2.2.2 REfactor in Other Scenarios

REfactor also helps in other diverse scenarios.

Multiple AP infrastructure. We start with multiple APs operating in infrastructure mode. As shown in Figure 1b, a client may be able to overhear transmissions from both it's associated AP and other nearby APs. An AP can remove redundancy based on any chunks a client may have overheard, regardless of which AP they were overheard from. *C2*, which is associated with *AP2*, overhears *AP1*'s transmission of `abc` to *C1*. *AP2* can therefore remove redundancy from its transmission of `abd` to *C2*.



(a) AP with clients (b) Multiple AP infrastructure

(c) Ad hoc mesh (d) With network coding

Figure 1: REfactor applied to diverse scenarios

Ad hoc meshes. REfactor can also be applied to transmissions between clients via a mesh or ad hoc network. Figure 1c shows the use of REfactor to achieve transmission reduction and, correspondingly, capacity improvement, in a small mesh network; this can be easily extrapolated to a larger mesh. Using normal forwarding, based on metrics such as ETX [12], 4 transmissions are required for two clients to send a packet to each other via a relay. By applying REfactor to the situation, we can reduce the size of the fourth transmission, resulting in $4 - \delta$ transmissions, where δ is proportional to the amount of redundancy removed: *C1*'s transmission of `abc` to *R1* is overheard by *R2*, which caches `ab`. *C2* transmits `abd` to *R2*, followed by *R1* transmitting `abc` to *C2*. Lastly, *R2* removes the redundancy from `abd`, sending `1d` to *C1*, since it knows *C1*'s cache contains `ab`.

Opportunistic routing in multi-hop meshes. In a similar fashion, REfactor can also be applied to opportunistic routing schemes in mesh networks (not shown in Figure). In approaches such as ExOR [10], the transmitter orders relays on the basis of their packet overhearing probability, before sending a batch of packets. Using REfactor, ExOR can be modified in two ways: First, the effective batch size can be reduced by removing strings that are duplicated either within the batch, or across prior batches sent by the transmitter. Second, the ordering of relays could take into account whether or not a relay has portions of content in the batch already cached; a relay with high overhearing probability could be given a high priority for forwarding if it has a significant fraction of bytes in the batch cached as it could prove invaluable in speeding completion time of the batch.

Networking coding. Network coding systems, such as COPE [20] have traditionally relied on coding full packets without paying attention to packet contents. REfactor can be combined with network coding to leverage duplication in packet payloads to help coding improve network capacity even further. We present the combination REfactor + COPE in Figure 1d. In this scenario, *C1* has a packet destined for *C4* and *C2* has a packet destined for *C3*, both of which must be sent via the relay. COPE imposes only 3 packet transmissions compared to 4 in the regular case, as *C3* can overhear *C1*'s transmission and *C4* can overhear *C2*'s transmission, providing a coding opportunity. REfactor + COPE leverages this overhearing even further by removing chunks known to exist in the destination client's caches: Assuming *C3* overheard an earlier transmission `abc` and *C4* overheard `xyz`, the relay can remove the redundancy (`ab` and `xy`) from the current packets and code the re-

mainder of the current packets, d $\oplus$ w. The coded packet, plus small shims to "encode" the removed redundancy, is broadcast to *C3* and *C4* simultaneously. COPE, in contrast would broadcast the much larger abd $\oplus$ xyw. *C3* and *C4* are able to obtain their packets by reversing the network coding and filling in removed chunks from their caches. Thus, REfactor + COPE reduces the number of transmissions to $3 - \delta$, where δ is the relative difference in the size of a full un-encoded packet (e.g., abd) and the above coded packet along with the shims. Assuming chunks are all the same size, $\delta \lesssim \frac{2}{3}$ in the example above, resulting in nearly $\frac{2}{9}$ better capacity than COPE.

2.3 Design Challenges

Although REfactor can offer substantial advantages as the above examples show, a careful design is needed to realize the benefits in practice. First, since overhearing is probabilistic in nature and caches are fixed size (hence, old content is evicted over time), a sender may not have an accurate view of whether the intended receiver has a certain content chunk already cached. In turn, this could lead to incorrect encodings and the resulting retransmissions negate the overall benefit of duplicate suppression in REfactor. Enforcing explicit synchronization of caches—which is a candidate solution for this problem—can add excessive overhead. Second, wireless nodes may be processing and memory constrained (for example, the clients in the above scenario could be smartphones), so REfactor mechanisms should require minimal resources from them. Designing REfactor to maximally leverage IP-Layer content overhearing while accounting for the above issues is challenging.

3. REfactor DESIGN

In this section, we describe the design of REfactor. For simplicity, we focus on the setting outlined in §2.2.1, namely, optimizing the downlink traffic performance of a wireless AP with a collection of clients. However, our basic building blocks, along with a few extensions described at the end of this section, apply to other scenarios as well.

REfactor applied to the single AP scenario involves the following steps: (i) When the AP receives a packet from the wired backend, we "chunk" it and compute a "fingerprint" per chunk. (ii) For each chunk, the corresponding fingerprint is used to refer to a content cache data structure that helps determine the probability of the intended receiver having cached the chunk. (iii) The AP computes the expected throughput benefit from removing chunks in the packet. If this exceeds a certain threshold, the AP removes the chunks and replaces them with the fingerprints instead. (iv) If the AP observes a hash collision for a chunk, it does not encode the packet, and it invalidates the chunk stored in its content cache for the collided hash. (v) If a client is unable to decode a packet using a fingerprint supplied by the AP, it requests a chunk retransmission from the AP. (vi) The AP updates the cache-residence probabilities associated with each chunk of the packet. We describe each of these steps and the underlying design issues next.

3.1 Chunking

Prior works have considered several different approaches for removing packet-level redundancy, which trade-off memory usage, processing time, and redundancy opportunities.The earliest, by Santos and Wetherall [25], supports redundancy elimination (RE) at the full packet level. While simple, this approach severely limits RE opportunities. Support for partially redundant payloads is provided by two different classes of approaches: Max-Match and Chunk-Match [5].

In Max-Match, the encoder computes a rolling Rabin-Karp hash [23] for each 32B region of a packet and selects a subset of these, based on hash values, to serve as packet fingerprints. The fingerprints are stored in a hash table, with each fingerprint pointing to the corresponding packet, which is stored in a packet cache in FIFO fashion. Fingerprints computed for an incoming packet are checked against the fingerprint table; a matching packet, if found, is retrieved and compared byte-by-byte around the 32B match region to identify the region of maximum overlap. The overlap region is removed and replaced with a shim, which carries the memory offset of the packet in the downstream decoder's FIFO-ordered packet cache from which the missing bytes can be constructed. The downstream cache is maintained in a similar fashion.

Chunk-Match computes and selects Rabin hashes in a similar fashion, but the chosen 32B regions form the boundaries of the chunks into which the packet is divided. A SHA-1 hash, which forms a fingerprint, is computed for each chunk and inserted into a chunk hash table. Each unique chunk is cached in FIFO order. When an incoming packet has a chunk matching against the chunk hash table, the matched region is replaced with the chunk hash. In both cases, the MAXP algorithm[1] has been found to be effective at selecting hashes offering a uniform distribution across a wide variety of packet payloads [7, 8]. We employ this in our design.

Chunk-Match's focus on chunks means it is less effective at identifying redundancy than Max-Match. As a result, packet-level RE systems have traditionally preferred Max-Match [5, 7, 26]. But we choose Chunk-Match in designing REfactor due to the following benefits:

1. *Effective memory usage:* A specific chunk only needs to be stored once, while Max-Match's packet-based approach requires storing full packet payloads, even if part of the payload already exists in another payload.[2]
2. *Better at accommodating overhearing:* As we argue below, Chunk-Match can be used to design simple techniques to handle wireless overhearing, without requiring complex operations at clients or APs or imposing too much overhead. In contrast, Max-Match requires clients to employ additional data structures and meta data to track overheard content, which imposes additional memory and computing overhead.
3. *More overhearing opportunities* Overheard packets that have duplicate bytes suppressed can be more effectively leveraged in Chunk-Match, because Chunk-Match can cache whatever chunks remain in those packets. In contrast, Max-Match would discard such packets because it needs full payloads.
4. *Ability to leverage partial packets:* Although not discussed in this paper, Chunk-Match can be effectively combined with partial packet recovery schemes [17] to further leverage partially overheard payloads. It is difficult to do so with Max-Match.

Chunk-Match still has key limitations due to which it cannot be applied directly in REfactor. First, the large size of the SHA-1 hash means the effectiveness of redundancy removal is limited. Also, the decoder (i.e., the client) has to compute and store SHA-1 hashes for cached chunks, which is expensive from both an energy and memory view-point. As an improvement, the encoder can only maintain the chunk hashes in a hash table, while the decoder maintains only the chunks in a FIFO cache: the encoder looks up chunk hashes for a match and replaces each match with a memory address in the

[1]MAXP selects hashes that are the maximum over all hashes computed over a p-byte region.

[2]The optimizations to Max-Match suggested by EndRE [5] to address high resource costs are not feasible for REfactor.

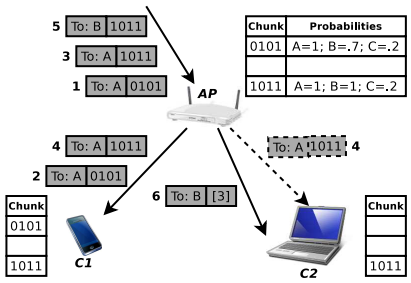


Figure 2: REfactor in practice. Solid lines indicate normal packet delivery and dashed lines indicate overheard packets. Transmissions are numbered in order from 1 to 6. AP and client cache contents after all transmissions are shown.

decoder's FIFO cache [5]. However, this approach is only suitable for point-to-point deterministic RE, e.g., across a wired link. In wireless, probabilistic overhearing causes the encoder to lose track of the decoder's chunk cache. One alternative, of having the AP compute chunk hashes and transmit them along with packets and maintaining a chunk hash table at the client impose high compute, network and memory overhead, as we show in §5.1.

3.1.1 Our Modifications

We modify and extend prior Chunk-Match designs in important ways to to address these drawbacks and to better tailor Chunk-Match to wireless overhearing.

Self-addressing chunks. In REfactor, we need to carefully manage the location of chunks within AP and client caches. Chunks must be cached such that the AP can provide a fingerprint in place of a redundant chunk that allows a client to locate a chunk within its cache, or identify a cache miss. As shown in Figure 2, *C2* only overhears the second packet transmitted to *C1*. A FIFO cache would be insufficient because the AP and *C1* would store the redundant chunk in the second cache slot, while *C2* would store the chunk in the first slot because it is the first chunk *C2* overheard. A proposed RE system for cellular networks suffers from a similar issue in the presence of packet loss [22].

The key innovation in our approach is that we select a slot in the cache (encoding or decoding) based on the content of a particular chunk (Figure 2). Thus, a chunk is *self-addressing*, i.e., the chunk itself identifies its location in the cache, and a removed chunk can be identified by the cache location. In particular, we use a compact n-bit hash (we use $n = 20$) of a chunk as the memory address where it is stored. The encoder simply sends the n-bit hash instead of the chunk to the downstream decoder.

This approach avoids the pitfalls of employing FIFO-based caches without relying on tightly synchronized caches; namely, an offset into the cache refers to the same chunk regardless of what other chunks a client may have overheard. It avoids the high computational cost of SHA-1 hashes and the overhead of maintaining the corresponding metadata. Furthermore, it is easy to identify cache misses: a lookup at a fixed-offset given by a chunk-hash can be used to determine whether or not a chunk is located in the cache.

The size of the chunk-hash represents a trade-off between the amount of memory required and the potential for redundancy. An n-bit hash allows a cache to store 2^n chunks but requires $2^n * m$ bytes of memory, where m is the maximum chunk size (based on parameters of the MAXP algorithm). A larger hash allows more unique chunks to be stored but requires more memory in clients which may already be resource poor, e.g., smartphones. The size of the chunk-hash and the method used to compute it also impacts the likelihood of collisions. We evaluate the trade-offs in hash-size in §5 and discuss how to deal with collisions in §3.3.

Overhearing estimation. Our second modification accounts for the facts that different clients overhear packets with different probabilities and that overhearing probabilities change over time. In our example (Figure 2), it is possible that *C3* (not shown) may not overhear either transmission to *C1* because *C3* is farther away and can only receive transmissions at a lower rate. However, *C2* is able to hear half of the transmissions to *C1*. To maximize the removal opportunities we want to be able to estimate how likely it is a particular client has a particular chunk. We add a *reception probability vector* to each chunk entry at the AP to aid the decision of whether or not to remove redundancy. §3.2 discusses how this vector is computed/updated and how it is used to guide the decision of whether or not to remove redundancy.

Handling cache misses. Lastly, we need to account for the fact that the AP's estimation of the contents of a client's cache may not be completely correct. In our example, we remove the redundancy from the transmission to *C2*, but it is possible *C2*'s cache may not have contained the removed chunk. We address this *cache miss* by providing a content chunk request mechanism. Since the shim header contains all the necessary information to identify a specific block of redundant content, the client uses the shim header from the original packet in its request for missing data. The AP replies to the missing content request with the shim header and the content of the chunk, allowing the client to properly reconstruct the packet and pass the packet to the network stack for normal processing. As a result of this recovery mechanism, our Chunk-Match approach requires the AP to store the contents of chunks in its cache, contrary to what recent optimizations suggest [5].

3.2 Removing Redundancy

In prior RE systems, a redundant chunk is either always removed [26] or a removal decision is based on network-wide optimization [9]. Furthermore, these systems assume packet caches at the sender and receiver are tightly synchronized, so a chunk present in the sender's cache is guaranteed to be in the receiver's cache. This synchrony assumption does not hold in REfactor because of opportunistic overhearing. Potential differences in the contents of sender and receiver caches requires REfactor to make a removal decision based on estimates of the receiver's cache contents.

A naive approach that always removes redundancy imposes high cost: Every chunk missing from a client's cache requires two additional packet transmissions to receive the missing data from the AP. Figure 3 shows the expected transmission time savings from removing a single 64B chunk from a packet (details in §3.2.2). There is no expected benefit when the probability of the receiver's cache containing the chunk is $< 90\%$, due to the high cost of cache misses. Therefore, we want to minimize cache misses by making a wise decision on whether or not to remove a redundant chunk.

3.2.1 Reception Probability Vectors

Our insight is to include a *reception probability vector* with each chunk stored in the AP's cache. A vector, V, contains an entry for each client currently connected to the AP, indicating the likelihood of a client having the chunk in its cache.

We know a client's cache will be guaranteed to contain a chunk if it existed within packets successfully sent to that client in the past. Hence, for the destination client d, we set $V_d = 1$ after the packet containing the chunk has been ACKed. If the chunk was removed from a packet sent to the client, we set $V_d = 1$ either: (*a*) after a request for the chunk has been received (§3.1.1) and the reply to

Rate	Fraction of nodes who overhear
6-24Mbps	0.15
36Mbps	0.12
48Mbps	0.08
54Mbps	0.06

Table 1: Median fraction of nodes in the Jigsaw testbed who overhear transmissions at various 802.11g rates [4]

the client has been ACKed, indicating the client has now cached the chunk it was missing; or (*b*) a few seconds after the original packet was ACKed, indicating the client already had the chunk because no request for a missing chunk was received.

All other clients could only have received a chunk via overhearing. We show in §5.3 that a highly accurate reception probability estimate is not necessary to realize the benefits of REfactor, so we take a low-overhead approach to estimating overhearing likelihood: reception probability is based on the rate r_d used to communicate with the destination client d and the rate r_i the AP uses to communicate with the overhearing client i. Measurements by Afanasyev et al. on an indoor 802.11g testbed [4] show that the chance of overhearing is relatively consistent for all 802.11b rates (1-11Mbps) and the five lowest 802.11g rates (6-24Mbps); noticeable differences in overhearing probability only exist for the three highest 802.11g rates (36, 48, and 54Mbps). The median fraction of nodes expected to overhear a transmission at a given rate is shown in Table 1.

Based on these findings, reception probabilities for a given wireless deployment can be calculated using a simple heuristic formula: If $r_i \geq r_d$, $V_i = 0.99$. A client which normally receives transmissions at a higher rate is very likely to receive transmissions at a lower rate, but we still want to be able to discern between clients which are guaranteed to have the chunk ($V_d = 1$) and clients which are highly likely to overhear the chunk ($V_i = 0.99$). If $r_i < r_d$, $V_i = \frac{e_d}{e_i}$, where e_j is the recipient fraction when sending at rate r_j, such as those shown above in Table 1.

More complex mechanisms, e.g., CHARM [18], may be able to provide better estimates of reception probability. In general, accurate estimation is hard, in part because overhearing probabilities can change at fine timescales [6]. However, as shown in §5, highly accurate predictions are unnecessary. In particular, we find that *directly using* the measurements in Table 1 may be good enough and estimating reception fractions for each deployment may not be needed. This is a highly desirable property of REfactor.

Reception probability vectors for chunks are updated every time the chunk is transmitted. For each client we store the maximum of an existing probability and the probability for the current transmission. When new clients join the network, reception probabilities are recorded for the clients for any chunks transmitted *after* they connect. When clients leave the network, reception probabilities for the clients are not stored for any newly transmitted packet chunks, and probabilities for the clients are invalidated in existing vectors.

3.2.2 Deciding to Remove: Model-Driven RE

REfactor decides whether or not to remove a redundant chunk based on the reception probability and a simple model of expected benefits. Benefit is measured as the reduction in transmission time resulting from the removal of a redundant chunk. We refer to this approach as *model-driven* RE.

The transmission time for a packet is a combination of wireless header transmission time t_h and per-byte payload transmission time t_b, which depends on the data rate to the client in question. Our experiments show a typical value of $t_h = 290\mu s$ for an indoor

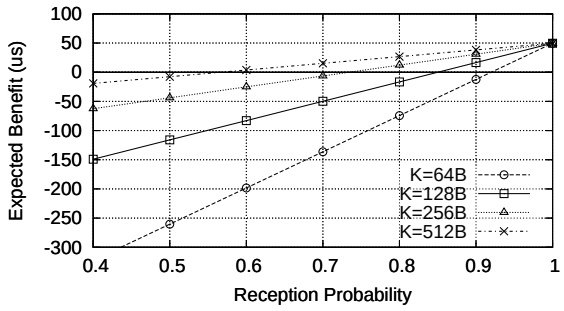


Figure 3: Expected benefit from removing a single 64B chunk from a packet with K total redundancy

setting with a client and AP separated by 2m, and $t_b = 0.885\mu s$ for a transmission rate of 11Mbps. For simplicity, we assume all packets are MTU (1500B) in size, making the total transmission time for a normal packet $t_h + 1500t_b$. Removing a k byte chunk of redundant content from a packet and replacing it with a h byte header makes the transmission time $t_h + (1500 - k + h)t_b$, a savings in air time of $(k - h)t_b$. If the load due to other nearby APs is ρ, then only $(1 - \rho)(k - h)t_b$ of the savings can be used toward improving the throughput of the current AP's own transmissions.

A removed chunk which does not exist in a client's cache requires two extra packet transmissions to obtain the missing chunk. The additional transmission time is $2t_h + (2h + k)t_b$, reducing the savings by this amount. Recall that V_d is the probability client d's cache contains the chunk. The expected benefit of removing a chunk in terms of free airtime (in μs) that could be used toward additional transmissions of the AP is:

$$Exp[B] = V_d(1 - \rho)(k - h)t_b - (1 - V_d)(2t_h + (2h + k)t_b)$$

This equation is a worst case estimate of expected benefit. In practice, the fixed header transmission time $2t_h$ associated with obtaining missing chunks only needs to be incurred once for each packet with ≥ 1 missing chunks. Applying the equation to multiple chunks in a packet will take into account the fixed header transmission time for missing chunks multiple times. We adjust $Exp[B]$ by taking into account the total number of redundant bytes, $K = \sum k$, in a packet, setting the fixed header transmission time for obtaining missing chunks to $\frac{2k}{K}t_h$ for each k byte chunk. This change allows REfactor to be more optimistic in removing redundancy.

Figure 3 shows the expected benefit from removing a single 64B chunk from a packet with K total redundancy for varying reception probabilities, assuming $\rho = 0$. As the graph shows, expected benefits increase with reception probability. Furthermore, for a given reception probability, higher amounts of total redundancy K increase the expected benefit from removing a single k-byte (in this case 64B) chunk. Similar graphs can be plotted for other rates.

The AP uses the expected benefit model to encode redundant chunks if $Exp[B]$ exceeds some threshold.

3.3 Collisions

We say that a hash collision happens when an n-bit hash of a chunk for a new packet indexes to an already occupied cache slot. Collisions should be handled carefully as they impact transmission correctness. In REfactor, the AP checks the new and already cached chunks for collisions by performing a byte-by-byte comparison of their contents. If they do not match exactly, the cache entry is marked as a collision. No chunks which hash to a collided entry are ever removed from a packet by the AP. All clients will also be

able to detect the collision because they will never receive a packet with a collided chunk removed, so they will recognize the collision in their byte-by-byte comparison. This approach reduces the potential redundancy removal opportunities, but it ensures no client application will receive an incorrectly reconstructed packet.

To avoid the entire cache filling with collision entries, the wireless AP will periodically initiate a *cache flush*. A cache flush clears all entries from the AP's and client's caches using a three phase process: (1) the AP broadcasts a cache flush request to all clients, (2) the clients clear all the entries in their cache and send an ACK, (3) when the AP has received ACKs from most of the clients, it clears its cache. The AP does not cache chunks from new packets while a cache flush is in progress. In the event a client does not acknowledge the flush request, due to lost packets or client disconnect, the AP will not encode any packets sent to the client until a retransmitted flush request has been acknowledge.

Whenever a client associates with the network, the client empties its local cache. An AP uses the association as a signal to clear old reception probability entries for the client.

3.4 Other Scenarios and Issues

REfactor uses the same caching and model-driven RE mechanisms to improve throughput in the scenarios presented in §2.2.2 but requires a few design extensions to fully function in these scenarios. Namely, the ability to estimate reception probabilities and communicate cache contents for *unassociated clients*, i.e. clients not directly communicating with an AP or via a specific relay.

Unassociated client reception. Clients may be able to overhear transmissions from other APs (as in Figure 1b) or nearby mesh nodes, or relays (as in Figure 1c). However, the relay has no way of knowing the client can overhear without explicit knowledge of the client's presence. Furthermore, the relay cannot estimate the reception probabilities for the client without knowing the rate the relay would use to communicate with the client.

We extend clients to notify a relay when they can overhear traffic from that relay. In infrastructure mode, a client can determine the AP it can overhear from (*AP1*) based on its beacons and send a message via its associated AP (*AP2*) to notify *AP1* its transmissions can be overheard by the client. *AP2* includes the rate it uses to communicate with the client, which provides an upper bound on the rate *AP1* would be able to use to communicate with the client. In a mesh network, a client can send a list of all relays it can overhear from to each of the relays in the list.

Overhearing notifications only need to be sent periodically. A relay will maintain reception probability vector entries for unassociated clients for all chunks sent after an overhearing notification is received.

Unassociated client caches. Knowing a client can overhear transmissions from another AP (*AP1*) is insufficient for the client's associated AP (*AP2*) to be able to leverage overhearing opportunities. Periodically, *AP2* must request cache information for the client from *AP1*. *AP1* sends a bit vector to indicate which cache slots the client likely overheard. *AP2* can update its reception probability vectors for these slots to account for chunks it may not have known the client overheard. As shown in Figure 1b, *C2* overheard the chunk `ab`, allowing *AP2* to potentially remove the chunk from a future transmission to *C2*. *AP2* uses the bit vector from *AP1* to update its cache to reflect this.

In a mesh network, cache contents can be updated using the same mechanism, or a relay can update its cache based on its knowledge of the path taken by a packet. Consider the example scenario shown in Figure 1c: if `abc` was received by *R2* from *R1* and *R2* knows *C1* can overhear *R1*, *R2* can add the chunk `ab` to its cache and indicate with high likelihood that *C1* overheard the chunk. In this manner, cache contents are communicated implicitly. A similar idea applies to the network coding approach shown in Figure 1d.

Min chunk size	REfactor	SHA hash based scheme
32	640 Mbps	64 Mbps
64	910 Mbps	118 Mbps
128	1203 Mbps	152 Mbps

Table 2: Comparison of encoding throughput for REfactor and a SHA hash scheme for different minimum chunk sizes.

4. IMPLEMENTATION

Our REfactor prototype is implemented as a pair of Click [21] modules. The *encoder* module is used at the AP to cache chunks, identify and remove redundancy and respond to requests for missing chunks. The *decoder* module is used at clients to cache chunks, reconstruct packets and request missing chunks. Each module is about 400 lines of code. Both use kernel-level Click to enable REfactor to work at the max 802.11g transmission rate of 54Mbps.

We chose to implement REfactor in Click because of the ease of deployment and flexibility this approach provides. Clients can easily run our Click decoder module to obtain the benefits of REfactor without operating system or application modifications. Furthermore, the encoder module can easily be deployed on an upstream network middlebox to serve multiple wireless APs. This avoids the need to modify AP firmware, which is often proprietary, and does not constrain REfactor due to the limited memory and processing power in many APs [1, 2]

5. EVALUATION

We conduct an evaluation of the various benefits of REfactor. Our default settings is an AP operating in infrastructure mode with two associated clients. We also show the benefit of using REfactor in some of the other scenarios discussed in §2.2.2. Our evaluation utilizes traffic from real-world traces containing realistic packet chunk redundancy patterns. We focus on the following sets of issues: *(i)* How does our scheme, which uses self-addressing chunks, compare against SHA hash based alternatives (§3.1) in terms of speed and effective RE? *(ii)* What is the trade-off between cache-size and collision likelihood imposed by the self addressing chunks approach? *(iii)* What is the overall benefit of REfactor under various realistic redundancy patterns and varying levels of overhearing? What aspects of REfactor's design contribute most to its benefits? Can and should REfactor's operation be adapted to observed traffic patterns? *(iv)* How does REfactor perform in an actual infrastructure-based wireless setup? *(v)* How does REfactor help in the other scenarios in §2.2.2?

5.1 Speed and Redundancy Removal

We evaluate the effectiveness of self-addressing chunks compared to a scheme where the encoder (AP) transmits SHA hashes to the client in encoded packet shims (§3.1).

Speed. We benchmark the encoding speed on a desktop with a 2.4 GHz CPU and 8GB DRAM, mimicking a middlebox (co-located with the AP) which can perform encoding on behalf of the AP. Table 2 compares the encoding speeds for REfactor and a SHA hash based scheme. With a 1GB chunk cache and a minimum chunk length of 64B, our unoptimized Click module can encode at the rate of 910Mbps. This rate is sufficient for an AP to serve 30 clients

Min chunk size	REfactor		SHA hash based scheme		
	Redundancy detected	Effective RE	Redundancy detected	Effective RE	Effect. RE w/ hash shipping
32	0.31	0.27	0.41	0.22	0.03
64	0.28	0.26	0.38	0.29	0.19
128	0.23	0.22	0.31	0.27	0.22

Table 3: Comparison of effective redundancy removal for REfactor and a SHA hash scheme

each at the rate of 18Mbps. In contrast, SHA hash based encoding is 8× slower (118Mbps); SHA1 hash computation is a major performance bottleneck in this scheme. Our lightweight decoding operations impose low overheads on clients, as well. We used a low-end laptop with 1.66 GHz CPU and 2GB DRAM to measure the decoding throughput. The measured decoding throughput is 160 Mbps (for chunk size ≥32B). In contrast, the decoder throughput for a SHA hash based scheme is only 50 Mbps, even with 128B chunks, because clients have to compute SHA1 hashes for each cached chunk.

Redundancy removal. We compare the effectiveness of redundancy removal for REfactor and the SHA hash scheme in Table 3. We use a real trace with high overall redundancy (45%) and a 1GB chunk cache for both schemes. With small chunks (32-64B), REfactor (0.31) detects 75% of the redundancy detected by the SHA hashing scheme (0.41), the gap being due to collisions. However, the shim overhead of the SHA hashing scheme is quite high as a shim must carry a 20B hash. As a result, REfactor's redundancy removal (0.27) is 25% better than the SHA hash scheme (0.22). Using larger chunks (64-128B), the effectiveness of the SHA hash scheme improves, despite a decrease in detected redundancy, since the relative shim cost drops. However, decoding overhead on clients is still high.

One way to overcome SHA hash computation overhead at the decoder is for the encoder to ship SHA hashes for *every* chunk contained in a packet, as opposed to just sending SHA hashes for encoded regions. Unfortunately, the additional overhead of shipping SHA hashes reduces the effectiveness of RE by 25% (Table 3, last column). Using larger chunks (256B), the shipping cost and encoding overhead would go down, but the detected redundancy itself significantly drops to 0.2 (not shown).

To summarize, REfactor's design, in particular, the use of self-addressing chunks, gives the right trade-off in terms of speed, overhead and effectiveness of RE.

5.2 Caching

We now study the effectiveness of our self-addressing chunk storage and provide guidelines on how to configure caches. In particular, we vary the hash-size n from 14-bits to 22-bits in size, resulting in 1MB to 256MB sized caches, and we compute how much redundancy we are able to remove from network traffic relative to an ideal infinite cache, which identifies 50% of bytes as redundant for the specific trace we study. In all cases the average chunk size is 64B. We show our results in Figure 4.

As expected, larger caches identify greater amounts of redundancy overall: e.g., a 512MB cache can identify nearly 60% of the overall ideal redundancy, whereas a 64MB cache can only identify up to 25%. In practice, caches can be provisioned on the basis of the average client's constraints: in an environment with laptops, using 256-512MB for caches is reasonable. When handhelds are employed, 128MB caches may be used.

We find there appears to be a "sweet spot" for flushing rates for

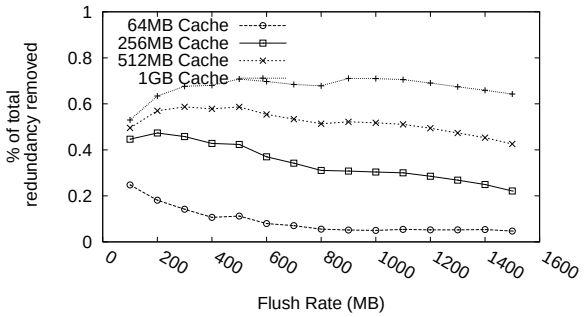


Figure 4: Impact of cache flush rate and cache size

most large cache sizes. For example, flushing a 512MB cache after every 500MB of traffic allows 10% more redundancy to be removed than flushing every 100 MB, and 16% more compared to flushing every 1.5GB. This is because a rapid rate of flushing (e.g., every 100MB) controls collisions better but reduces opportunities for removing redundancy; on the other hand, too slow a rate of flushing (e.g., every 1.5GB) does not eliminate collided entries fast enough. In general, flushing after every 200-300MB of traffic works well.

5.3 Goodput

We now evaluate the improvements REfactor provides. We use a simple emulated two-client setup with a single AP operating in infrastructure mode, as described in §2.2.1. In our setup, *C1* is located close to the AP, with perfect overhearing and a fixed high transmission rate (54Mbps), while we vary *C2*'s location relative to the AP. Depending on *C2*'s location, its ability to overhear transmissions to *C1* varies.

We experiment with five different overhearing scenarios, using overhearing probabilities at *C2* of 90%, 70%, 50%, 30% and 10%. An overhearing probability of 90%, for example, means the bit error rate is such that a *full length packet* (1400B) is overheard with 90% chance; a smaller packet that REfactor creates could, obviously, be overheard at a higher probability. We assume that *C1* overhears all transmissions to *C2*.

We use a simple Click [21] configuration to emulate overhearing, where we super-impose packet reception probabilities at various packet sizes and transmission rates. These are derived from real-world measurements we collected in a relatively noise-free environment. For the five overhearing scenarios above, we fix the transmission rate to *C2* at 54Mbps, 36Mbps, 24Mbps, 11Mbps and 1Mbps, respectively. In all five cases, we assume the bit error rate in transmissions between the AP and *C2* is 8.8×10^{-6}, resulting in an 8.5% packet loss rate for full length packets.

The traffic transmitted to each client in our experiments is based on a real-world trace we gathered on an outbound link from a university web server. We use traffic to destinations in the real trace to construct a traffic mix for the two clients in our simulation. We pick three sets of traces that offer high (49%), medium (23%), and low (4%) inter-client redundancy, i.e., bytes shared across clients, where redundancy is measured using the Max-Match approach with a large cache; the intra-client redundancy is 1%, 24% and 46%, respectively. Note that the overall redundancy is roughly similar (≈50%) across the three traces.

We do not claim the traces we study reflect the actual redundancy we expect to see in traffic sent to, and shared between, wireless clients; quantifying the redundancy is not a goal of our work and this issue has been explored in prior studies [5, 8, 14]. Rather, our goal is to use the traces to recreate a variety of realistic redundancy

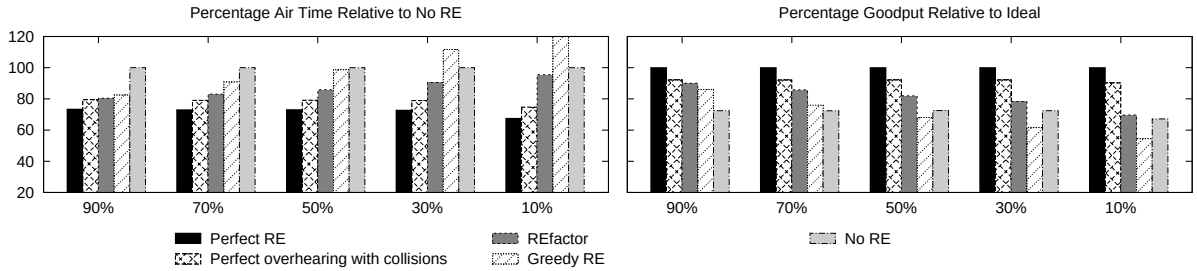


Figure 5: Total air time across both clients (a; left) and *C2*'s goodput (b; right) for a trace with high overlap.

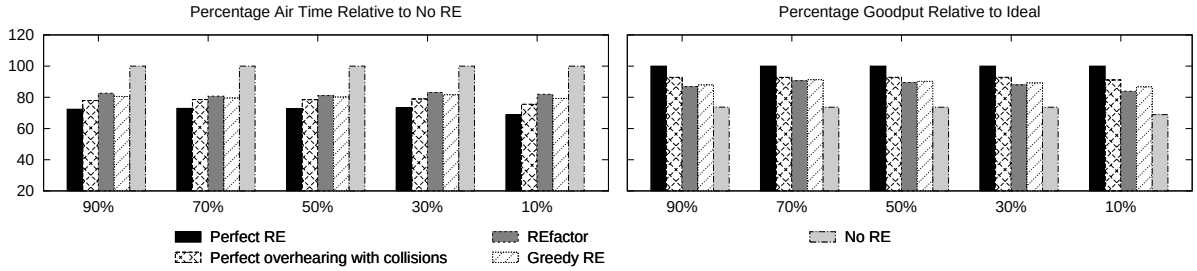


Figure 6: Total air time across both clients (a; left) and *C2*'s goodput (b; right) for a trace with low overlap.

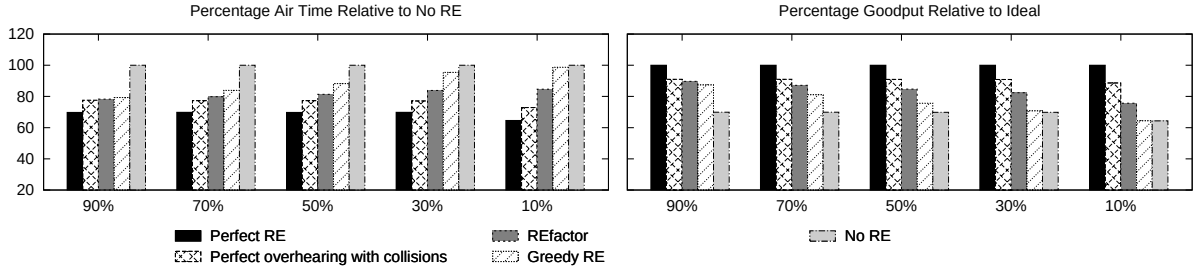


Figure 7: Total air time across both clients (a; left) and *C2*'s goodput (b; right) for a trace with medium overlap.

patterns, i.e., granularities of redundancy, and spacing of redundant bytes in time and across hosts, to study how they impact the benefits of REfactor given its design choices, and also to understand the conditions in which REfactor offers the most benefits.

We use three metrics: (1) *Total air time* defined as the total amount of time spent in transmission to either client (including missing chunk requests and responses). This is an indication of the medium's utilization. (2) *Goodput of a client* defined as the total number of bytes transferred to the client divided by the total time spent by the client in receiving them, which includes time spent in retransmissions due to packet losses or missing chunk transmissions due to cache misses. (3) *Packet loss rate* at the client.

5.3.1 High Inter-client Redundancy

We first study the performance where a lot of content is shared *across* clients and little overlap exists *within* a client's traffic. The results in Figure 5 show the overall air time (a) and *C2*'s goodput (b).

The bar "perfect RE" corresponds to the case where we assume no collisions occur and all clients can overhear all traffic. In "perfect overhearing with collisions", we assume cache sizes are limited to 512MB, resulting in collisions, but nodes can overhear all traffic. The bar "greedy RE" reflects a RE approach which always encodes packets as opposed to REfactor's use of model-driven RE. Finally, "no RE" represents a situation where no content overhearing or redundancy elimination is applied.

The overall airtime is plotted relative to the total airtime under "no RE", showing the lowering in utilization due to various schemes. The goodput is plotted relative to *C2*'s goodput under "perfect RE" showing how close to ideal the improvement in goodput gets. We make these observations:

1. REfactor offers substantial improvements relative to "no RE", with airtime (Figure 5a) being 20% lower in the highest quality link case (90% overhearing) and 7% lower in the lowest quality link case (10% overhearing). REfactor's airtime is between 6% and 30% worse than "perfect RE," the difference arising due to the need to account for collisions, and the need to account for and recover from cache misses. *C2*'s goodput (Figure 5b) is 24% and 4% better than "no RE" in the highest quality and lowest quality link cases, respectively.
2. In the poor quality link case, e.g., 10% overhearing, REfactor may overhear as few as 10% of the packets compared to "perfect overhearing with collisions"; thus, one may expect that it should only be roughly 10% as effective in improving goodput (Figure 5b) as "perfect overhearing with collision". Instead, we see that REfactor is 4% more effective

Overhearing %age	Loss % (% better than "no RE")
90	6.2 (27)
70	6.7 (21)
50	7.1 (16)
30	7.6 (11)
10	7.9 (7)

Table 4: Loss % with REfactor. The loss rates due to "no RE" and "perfect RE" are 8.5% and 5.3%, respectively. We show % lowering of loss rate relative to "no RE" in brackets.

than "no RE", whereas "perfect overhearing with collision" is 34% more effective. Thus, REfactor does not seem to lose as much performance as we might expect under low quality links. The reason for this is that encoded REfactor packets are smaller, and hence they experience lower packet loss rates compared to "no RE"; see Table 4 which shows that REfactor imposes 7-27% fewer drops than "no RE". Fewer losses helps improve goodput. More importantly, the packets that are not lost also carry valuable unique bytes that contribute to removing redundancy from future packets. This effect is likely to be much more pronounced in situations where there is a much greater amount of content shared across clients, e.g., in flash crowds.

3. Comparing REfactor against "greedy RE", the gap is small at high quality links, but increases significantly when link quality falls below 50%. At 50% overhearing, our approach improves goodput by over 13%, whereas "greedy RE" results in a 6% *drop* (Figure 5b). Thus, model-driven RE in REfactor plays a crucial role in ensuring robust performance, especially under poor overhearing and high inter-client redundancy.
4. Consider the performance for the link with 70% overhearing probability, where "greedy RE" still offers non-trivial goodput benefits (5%; Figure 5b) compared to "no RE". Comparing this with the results for the 90% link, we can conclude that if REfactor used 90% as the reception probability estimate for the link in its model driven RE, but the actual probability was 70%, then REfactor's overall performance would still be better than "no RE". This shows REfactor's overhearing probability estimation is robust to a certain degree of error especially when link quality is reasonable. However, at poorer link qualities (50% or below) mistakes can prove costly. Thus, it helps to be conservative with encoding, i.e., use a high threshold for expected benefit in model driven RE, when link quality is poor and when inter-client redundancy is high.

5.3.2 Low Inter-client Redundancy

Next, we look at the traffic mix with very high redundancy within client traffic (i.e. high intra-client redundancy) and low redundancy between clients (i.e. low inter-client redundancy). We plot total airtime and relative goodput achieved by *C2*, both in relative terms as before, in Figure 6. We note the following:

1. Compared to the high inter-client redundancy case above (Figure 5a), REfactor offers better airtime goodput improvements relative to "no RE": e.g., at 30% overhearing, REfactor is 22% better than "no RE" in the low inter-client redundancy case (Figure 6a), whereas in the high inter-client redundancy case (Figure 5a), it is 8% better than "no RE". Because REfactor does not have to deal with overhearing, it is able to derive substantial benefits most of which are due to IP-layer RE itself.
2. The variation in overhearing rate has a slight effect on the benefits of REfactor, with benefit dropping as overhearing becomes poor. While most of REfactor's benefits with this trace are from intra-user redundancy, the trace does have a small amount of inter-client redundancy (4%); at low overhearing probability, model-driven RE in REfactor would conservatively decide against encoding most, if not all, inter-client redundant packets, resulting in a drop in goodput.
3. The performance of "greedy RE", which encodes all packets, is slightly better than REfactor. Whatever bytes are saved in this fashion contribute to high goodput and the small number of cache misses that result (for the inter-client traffic) can be easily recovered through retransmissions. Thus, when intra-client redundancy is predominant—the redundancy pattern can be determined by profiling traffic on the fly—it is best to turn off model-driven RE and encode all data.

C2's Distance from AP	No RE Goodput	REfactor Goodput	Percentage Improvement
3m	4.0Mbps	3.4Mbps	20%
6m	3.0Mbps	2.6Mbps	14%
10m	1.3Mbps	1.2Mbps	6%

Table 5: Performance improvement provided by REfactor in a real infrastructure-based wireless setup.

5.3.3 Medium Inter-client Redundancy

In Figure 7 we show a situation where redundancy is roughly equally inter- and intra-client. Comparing with Figures 5 and 6, the performance offered by REfactor is intermediate compared to the prior two cases, as expected. We also note that the performance of "greedy RE" is almost comparable to that of "no RE" at 30% and 10% overhearing. Thus, with less redundancy, the impact of incorrect estimation of link overhearing is even less pronounced: more specifically, if REfactor used 90% or 70% as the overhearing probability estimate for a link that current has 50% overhearing (or even lower), the performance of REfactor may still be noticeably better than not using RE or overhearing. This also means that model-driven RE can be somewhat more aggressive, i.e., use a lax threshold for expected benefit, under this kind of traffic pattern.

5.4 Overall Benefits: Testbed Results

While the results we have discussed so far are derived from an emulated infrastructure-based scenario, we also measure REfactor's performance using an actual wireless AP and two clients. We use the high inter-client redundancy trace, and we vary *C2*'s distance from the AP from 3 to 10 meters to explore a range of overhearing probabilities. Table 5 compares the goodput without RE and using REfactor.

Similar to the results in Figure 5b (where REfactor's goodput improvement over "no RE" ranges from 24% to 4%), the benefits from REfactor in a real wireless setup range from 20% to 6%. This confirms that our emulated setup provides a reasonable representation of REfactor's performance improvements in practice.

5.5 Extensions

5.5.1 Multi-AP Improvements

We extend our setup to two APs operating in infrastructure mode and three clients: *C1* and *C2* associated with *AP1* and *C3* with *AP2*. Additionally, *C2* is able to overhear transmissions from both

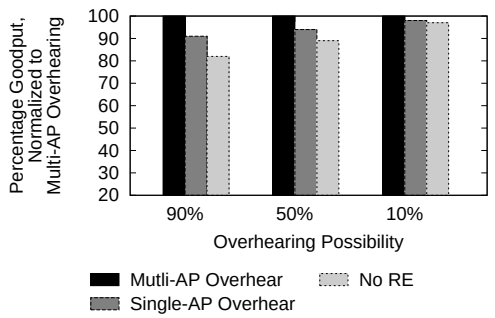


Figure 8: Goodput improvements in a multi-AP scenario

Overhearing %age	Overall Improvement (% better than COPE)	Relay to C3/C4 Improvement (% better than COPE)
90	14	38
70	11	26
50	10	21
30	6	13
10	3	7

Table 6: Air time savings %age with REfactor + COPE.

APs. Figure 8 compares the relative goodput for *C2* without RE, when overhearing only from its associated AP, and when overhearing from multiple APs. We observe that *C2* realizes up to 10% more benefit from REfactor when taking advantage of traffic overheard from other APs. As expected, returns diminish as overhearing probabilities decrease.

Our multi-AP simulation assumes both APs can overhear each other's transmissions, avoiding collisions. However, collisions may occur in the case of hidden terminals. For example, if *AP1* transmits to *C1* at the same time *AP2* transmits to *C3*, the two destinations will receive their respective packets, but the packets will collide at *C2*, who will be unable to overhear either packet. Such collisions prohibit *C2* from receiving maximum benefits from REfactor. However, REfactor may be able to reduce the likelihood of collisions due to decreases in the size of *C1* and *C3*'s packets.

5.5.2 REfactor + Network Coding

We implemented a simplified version of COPE [20] within our Click prototype and experimented with the scenario in Figure 1d. Our simplified version of COPE XORs packets, but we ignore confounding factors like pseudo-broadcast, retransmissions, and reception reports. When the *relay* is scheduled to send packets, it determines if it has packets for *C3* and *C4* that can be coded, removes redundancy from them and sends a coded packet along with a shim. For simplicity, we assume the relay has perfect knowledge of what *C3* and *C4* overheard for both COPE and our approach; in practice, the relay has to rely on feedback from clients [20].

The air time savings of REfactor + COPE, compared to just COPE, are shown in Table 6 for different overhearing percentages between *C1–C3* and *C2–C4* (assuming both links have the same overhearing probabilities). With 90% overhearing, REfactor provides 14% air time savings. This savings is purely from reduced transmission sizes from the *relay* to *C3*/*C4*: the savings for *relay–C3/C4* transmissions is 38% with 90% overhearing. Compared to air time savings with "perfect overhearing with collisions" in the single AP case, this is almost twice as much, a result of the transmission reductions realized via network coding. At lower overhearing (e.g., 10%), the overall relative benefit of using REfactor reduces (to 3%) because there is less content overheard.

6. RELATED WORK

A wide range of techniques have been suggested to improve wireless network throughput, including rate adaptation, partial data recovery, overhearing, and duplicate suppression. We discussed the latter two sets of approaches earlier in the paper. We discuss the remaining two next.

Rate Adaptation. Rate adaptation focuses on improving wireless throughput by reducing the number of unsuccessful transmissions [16, 24, 28]. The techniques proposed use estimates of channel quality to seek a packet transmission rate that reduces the number of required retransmissions. Currently rate adaptation schemes don't exploit overhearing and require sending all packets in full. But REfactor provides interesting opportunities for enhancement. REfactor reduces the size of some packets, lowering the chance of errors and the volume of retransmissions. This allows wireless communication to use higher rates.

Partial Data Recovery. Partial data recovery provides network throughput improvements by enabling nodes to extract portions of data which have been correctly received. PPR [17] uses SoftPHY hints to determine which bits of a packet are likely correct; only corrupted portions are retransmitted. MORE [11] forwards linear combinations of packets in wireless mesh networks, with the goal that a receiver has heard some of the packets in the combination and can deduce the rest. MIXIT [19] combines PPR and MORE. These approaches require low-level information from the PHY layer regarding which portions of the data is faulty. Our system only utilizes packets correctly received in full. However, we take partial data recovery to a higher level by removing portions of a payload a receiver already has and only sending data the receiver is missing. Thus, we reduce the number of low-level symbols that need to be forwarded and improve throughput without PHY or link-layer modifications. However, our technique can be combined with low-layer partial data recovery for even more throughput improvement.

7. CONCLUSION

In this paper, we described an IP-layer content overhearing technique called REfactor. In REfactor, wireless nodes maintain packet caches which they use to remove strings of bytes that appeared in packets they received or overheard earlier. We described novel data structures and mechanisms that allow REfactor to effectively support IP-layer overhearing based designs even on resource constrained hosts.

REfactor represents a refactorization of content overhearing ideas, moving overhearing below the transport layer. Through qualitative arguments and quantitative analysis based on extensive experiments, we showed that the refactoring provides significant performance (goodput) benefits, higher speed operation, and lower loss rates under a variety of situations for infrastructure wireless networks. It is also easy to adopt, requiring a simple software upgrade at the IP layer, and various aspects of it (e.g., cache sizes and overhearing mechanisms) are easy to configure. Finally, we showed examples of how REfactor can augment other overhearing based proposals in interesting ways to significantly enhance their effectiveness.

We believe that REfactor presents a promising start for a variety of follow-on studies on applying content overhearing to improve wireless performance. Some avenues for future work include combining REfactor with partial packet recovery schemes and applying REfactor to sensor networks.

8. ACKNOWLEDGEMENTS

This work was supported by NSF grants CNS-1040757, CNS-0905134 and CNS-1050170, and donations from Cisco and NetApp. Ashok Anand's research is supported by a Google PhD Fellowship.

9. REFERENCES

[1] Cisco enterprise wireless products. `http://cisco.com/en/US/prod/wireless/wireless_for_enterprise.html`.

[2] Linksys WRT54G series. `http://wikipedia.org/wiki/Linksys_WRT54G_series`.

[3] M. Afanasyev, D. G. Andersen, and A. C. Snoeren. Efficiency through eavesdropping: link-layer packet caching. In *NSDI*, 2008.

[4] M. Afanasyev and A. C. Snoeren. The importance of being overheard: throughput gains in wireless mesh networks. In *IMC*, 2009.

[5] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese. EndRE: an end-system redundancy elimination service for enterprises. In *NSDI*, 2010.

[6] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM*, 2004.

[7] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *SIGCOMM*, 2008.

[8] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: findings and implications. In *SIGMETRICS*, 2009.

[9] A. Anand, V. Sekar, and A. Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *SIGCOMM*, 2009.

[10] S. Biswas and R. Morris. ExOR: opportunistic multi-hop routing for wireless networks. In *SIGCOMM*, 2005.

[11] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. In *SIGCOMM*, 2007.

[12] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom*, 2003.

[13] F. R. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. G. Andersen. Ditto: a system for opportunistic caching in multi-hop wireless networks. In *MobiCom*, 2008.

[14] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus wifi networks. In *PAM*, 2011.

[15] S. Guha, J. Chandrashekar, N. Taft, and K. Papagiannaki. How healthy are today's enterprise networks? In *IMC*, 2008.

[16] G. Holland, N. Vaidya, and P. Bahl. A rate-adaptive MAC protocol for multi-hop wireless networks. In *MobiCom*, 2001.

[17] K. Jamieson and H. Balakrishnan. PPR: partial packet recovery for wireless networks. In *SIGCOMM*, 2007.

[18] G. Judd, X. Wang, and P. Steenkiste. Efficient channel-aware rate adaptation in dynamic environments. In *MobiSys*, 2008.

[19] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-level network coding for wireless mesh networks. In *SIGCOMM*, 2008.

[20] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. XORs in the air: practical wireless network coding. In *SIGCOMM*, 2006.

[21] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. on Computer Systems (TOCS)*, 18:263–297, August 2000.

[22] C. Lumezanu, K. Guo, N. Spring, and B. Bhattacharjee. The effect of packet loss on redundnacy elimination in cellular wireless networks. In *IMC*, 2010.

[23] M. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard University, 1981.

[24] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. OAR: an opportunistic auto-rate media access protocol for ad hoc networks. 11:39–53, 2005.

[25] J. Santos and D. Wetherall. Increasing effective link bandwidth by suppressing replicated data. In *USENIX ATEC*, 1998.

[26] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *SIGCOMM*, 2000.

[27] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for internet data transfer. In *NSDI*, 2006.

[28] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *MobiCom*, 2006.