Redundancy and Distributed Caching in Mobile DTNs

Mikko Juhani Pitkänen
Helsinki Institute of Physics
Technology Programme
pitkanen@cern.ch

Jörg Ott
Helsinki University of Technology
Networking Laboratory
jo@netlab.tkk.fi

ABSTRACT

Delay tolerant networking (DTN) allows endpoints to exchange information in networks where end-to-end path may not exist at any given time. In this opportunistic model, routing and forwarding functionality in intermediate nodes enables data transfer following the store, carry, and forward paradigm. Thereby, even in sparsely populated settings, node-to-node contacts can be exploited for communications where network infrastructure does not exist or is not viable to use. Beyond plain message forwarding, the increasing amount of storage in modern devices enables nodes to hold messages for an extended period of time. This feature can then be utilized to create an opportunistic cooperative storage. Having established how to leverage DTN routing nodes for distributed content retrieval, caching, and storage in previous work, in this paper, we investigate applying redundancy schemes to improve message delivery and content retrieval.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks; C.2.4 [**Distributed System**]: Distributed applications; C.2.6 [**Internetworking**]: Routers

General Terms

Algorithms, Performance, Design, Reliability

Keywords

Delay Tolerant, DTN, Caching, Storage, Bundle, Application, Mobile Ad-hoc Networking

1. INTRODUCTION

An increasing number of handheld devices are shipped today with short range wireless technologies which allow opportunistic communication with other portable personal devices, in addition to talking to infrastructure networks. At the same time, the amount of memory in these devices is constantly increasing to the extent where it is not too scarce a resource anymore. Together with the improvements in the user interface design, these personal communication devices have become an interesting platform to create, store, use, access, and share an increasing amount of content.

Infrastructure networks are *the* primary means for accessing and exchanging information, particularly as they enable access to resources in the Internet. But they have their drawbacks, too: the network services may be too costly to use, may be congested, or may not cover peripheral areas. Moreover, today's most prominent mobile applications (mail, web, file and database access) rely on infrastructure-based servers as "hubs" for communication. This dependency prevents mobile nodes from exploiting proximity for communication even though their basic capabilities would allow it.

Protocols have been designed to enable communication in mobile ad-hoc networks (MANETs), e.g., between nodes in sensor networks, but also to extend the reach of infrastructure networks by using other mobile nodes for packet forwarding. It has been recognized, however, that the density of *cooperating* mobile nodes will often be insufficient to establish end-to-end paths between a mobile node and its fixed or mobile peers. To overcome the node density constraints, the concept of delay-tolerant networking (DTN) [7] has been applied to MANETs: DTN nodes interact by means of exchanging messages in a store, carry, and forward manner, not requiring the existence of an instant end-to-end path [15, 25]. However, only enabling mobile nodes to communicate with one another and with the Internet does not yet address the application needs—and thus ultimately the user demands—of completing their respective operations within reasonable time. Studies on routing protocols for opportunistic communication between mobile and/or fixed (infrastructure) nodes have shown that message delivery time may be significant, depending on the contact and inter-contact times experienced by the involved nodes [9, 8, 30].

This motivates exploiting the storage space available on mobile devices by empowering intermediate nodes for caching or even for distributed storage, thus trading storage for communication resources. Assuming suitably designed or modified application protocols [16, 17], intermediate nodes may keep messages containing information resources in addition to forwarding them. Such intermediate nodes can then reply to passing requests if they hold a local copy of the sought resource—assuming that that the resources are identifiable (so that requests and stored resources can be matched) and that sufficient information about the application protocols is available to allow replicating or constructing responses [18].

The nature of DTNs allows transmitting large messages which may thus contain complete Application Data Units (ADUs), e.g., a web resource, as opposed to only ADU fractions in small IP packets. Working with such messages facilitates application-supportive operations as mentioned above because all the application data is

MobiArch'07, August 27–31, 2007, Kyoto, Japan.

available at once and no reassembly from individual packets is needed.

We have shown that opportunistic caching by minimally enhanced (and optionally application-aware) intermediate DTN nodes may improve the accessibility of information to mobile nodes in network scenarios where sparsely distributed mobile nodes cooperate to provide access to resources in the Internet [18]. Likewise, distributed storage may be realized amongst cooperating mobile nodes. However, while DTN messages originated by the application may contain integer ADUs, the forwarding process may lead to message fragmentation [26]. This may be due to short-lived contacts or limited storage capacity allowing only parts of a message to be transmitted or received. In addition fragmentation may stem from routing protocols, as seen with erasure- or network-coding based routing [30, 8, 31].

In this paper, we generalize our notion of cooperative caching and storage from dealing only with complete resources—i.e. unfragmented bundles—to operating also on fragments: to work with the aforementioned routing protocols, to take advantage of short contact times, to allow exploiting even limited (remaining) storage capacity, and to be able to deal with very large resources. After reviewing related work in section 2, we introduce the research problem and our protocol design in section 3, and present our validation setup and simulation results in section 4. We conclude this paper with a summary and hints at future work in section 5.

2. RELATED WORK

In this paper, we assume delay-tolerant networking [7] as basis for information exchange, using the DTNRG architecture [3]. We focus on hybrid networking environments, where users in mobile ad-hoc networks may indirectly obtain access to infrastructure networks via other mobile nodes or dedicated gateways. Routing in such opportunistic networks has been studied extensively (for surveys see [33, 19]). We borrow routing and forwarding algorithms from the vast literature and focus on adding content storage and retrieval.

2.1 DTN and Erasure Coding

Jain *et al.* have discussed how redundancy can be used to cope with communication failures in DTN [8]. They formalize the problem of sending coded blocks over different paths so that the overall message delivery probability is maximized. And they also propose strategies on deciding on how large fraction of blocks should be sent over which path. Erasure coding-based flooding has been shown to provide good performance in opportunistic networks [30], reducing the message delivery delay. Other proposals [2, 23] to use erasure coding discuss the benefits of removing the need for interactive communications, especially sending acknowledgments, during reliable data transfer. Moreover, modern coding theory has proposed several codes (Raptor, LT, Tornado) that achieve fast decoding by using only small amount of extra information. While these codes may provide interesting approaches for DTN-based data transfer, they have disadvantages when used to provide redundancy for storage: modern low-density parity-check codes (LDPC) suffer from property that not all combinations of fragments can be used to reconstruct a resource (e.g., file) [4]. Furthermore, Reed-Solomon (RS) codes have been shown to be also more efficient with coding parameters m and n in the range we envision to be used with our cooperative storage design [4].

Kamra *et al.* [10] present Growth Codes which can be used to increase the amount of data that reaches sink in zero-configuration sensor networks. Growth codes also allow recovering partially transferred data the utility of which is, however, highly application-dependent. How to generate new coded fragments in a bandwidth-efficient manner to replace the failed ones is discussed in [5]. They propose regenerating codes which require less bandwidth for storage maintenance than previous approaches.

2.2 Storage in the Network

Erasure coding has been used to provide reliability of the archival layer for collaborative network storage in the OceanStore project [12]. Scalable peer-to-peer overlays have also been studied in the content addressable network (CAN) [22] and Chord [29] projects. These projects have provided algorithms for addressing content in scalable manner. However, in highly dynamic networks that we focus on in our study, the maintenance cost of look-up structure easily becomes unacceptably high. The PAST project has continued to study the caching in storage overlays. Study has shown how locally controlled caching may be used to increase the fetch performance in the storage overlays [24]. The study also discussed how even during very high storage utilization the small files may be still cached to where some unused storage exists. This has in common to our motivation to look into cooperative caching since we may opportunistically cache even smaller bundle fragments without inducing overhead of registering the stored fragment to look-up service.

Our earlier work has shown how large files can be efficiently striped and erasure coded for storage both in local [20] and in Grid environments [21]. The chosen striping scheme easily works with data entities at the quite arbitrary sizes, similar to our ADUs. The Internet Backplane Protocol [1] creates a global storage service by sharing an end system's disk resources via an infrastructure network and offering access and management functions. While the above approaches provide rather predictive resource access, our focus in more on probabilistic operation.

Recent research has discussed optimizing the file availability in peer-to-peer content distribution communities, defined by intermittently connected (mobile) nodes that contribute storage, contents, and bandwidth [11]. The primary goal is to satisfy file requests from within the community, which also increases the speed of downloads from the global network since less peers compete for the outgoing link. Only when the community cannot serve the file the request should be sent further to the global network. Similarly, cooperative caching may be applied in ad-hoc networks to serve requests from mobile users and reduce the distance requests and responses have to travel (and thus the communication overhead) [32]. This issue is further discussed in the context of cooperative caching for multimedia contents in [13].

Finally, the bundle protocol [26] defines the *custody transfer* mechanism which combines hop-to-hop reliability and persistent storage to enable reliable end-to-end communications. As node resource constraints may lead to congestion, Seligman *et al.* propose mechanisms to temporarily shift storage load by moving custody bundles from one node to another and retrieving them when the congestion has cleared [28]. Moreover, Seligman has has shown how the storage usage of the custody transfer mechanism can lead to efficient resource utilization in networks with intermittent connectivity when targeting reliable transfer [27].

3. DTN-BASED REDUNDANT STORAGE

We consider a (sparse) mobile community consisting of set of nodes, as depicted in figure 1, willing to contribute storage and communication resources to other nodes in the community, while other nodes (those not containing a storage icon) only provide forwarding functions. The community nodes cooperate to create distributed storage from these resources, used for caching information

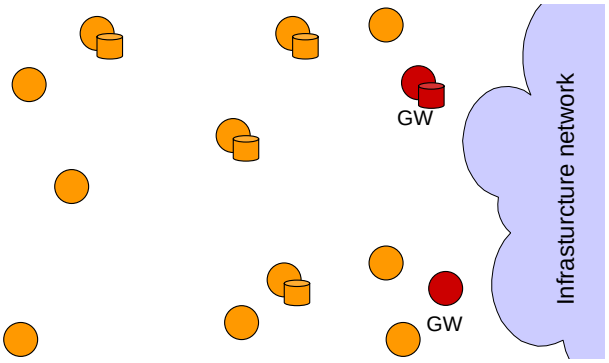

Figure 1: Cooperative DTN storage with gateway nodes.

resources previously retrieved from the infrastructure network via gateway nodes. Fulfilling requests from within such a community network increases the performance of fetching data from outside the community as the gateway nodes become less congested [11] and reduces the response time, particularly in disconnected environments [18].

3.1 DTN Nodes for Distributed Storage

To enable cooperation between nodes, we have defined an architecture for supporting message-based content storage in DTN nodes [18]. In this environment, the ADUs exchanged between DTN nodes are semantically self-contained protocol messages, e.g., comprising all objects of a web page. The contained resources have unique identifiers (e.g., derived from the respective URIs). The identifiers and the application layer protocol operation (e.g., POST vs. GET) are made visible to the DTN layer via a dedicated bundle protocol extension header. Forwarding in DTN nodes operates on these complete ADUs. We have defined extensions to the per-node bundle processing: routing, forwarding, and queuing behavior may be adapted upon the identified protocol operations and the enclosed resources (if any). In particular, this allows us to leverage the DTN store-carry-and-forward paradigm and make DTN nodes keep a copy of a message for a longer period of time than required by the forwarding algorithm, i.e., cache the ADU.

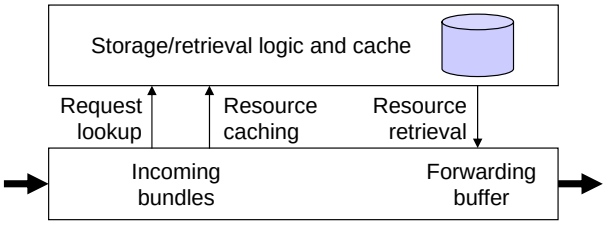

Figure 2: Architecture for distributed DTN storage module.

We propose a design for a community DTN node that includes a cache-enabling module and contributes storage to the forwarding buffer and the caching functionality, as illustrated in figure 2. The storage/retrieval logic module provides interfaces to perform cache lookups for resources stored as bundles in the queue or the cache for retrieving them to respond to passing requests. It also provides a caching interface for storing resources based on probabilistic selection (or other selection policies to be defined in the future).

3.2 Modeling Erasure-Code-based Operation

We consider a sparse network of mobile nodes N_i which send, forward, and receive requests for resources and responses containing one out of many resources U as self-contained and identifiable ADUs. We call these operations $req(U)$ and $rsp(U)$, respectively. Some of these nodes act as Internet gateways and thus are the ultimate target for requests as they represent all resources in the Internet and hence the origin of responses. Responses may be cached by intermediate nodes forwarding them. This leads to a model in which requests may also be served from the cooperative storage formed by the community nodes. Our research problem is then defined as introducing redundancy to bundle communication and storage in order to increase the response probability and the cache hit rate and, at the same time, to reduce the response latency.

To address these issues we do not define a new DTN forwarding scheme, but (1) allow bundles to be fragmented for transmission and (2) additionally apply application level erasure coding on top of existing protocols. We propose to use Reed Solomon codes, allowing us to divide single bundle carrying a resource U of size $S(U)$ into n source blocks and to encode m additional blocks of the same size. Any combination of n blocks out of $n+m$ of equal size $S(U)/n$ can be used to recover the bundle.

Generally, a request $req(U)$ for a complete resource U is sent by a node N_s. The request is forwarded until it hits a node N_r that either has a cached copy of U or is a gateway and can thus retrieve U from the Internet. For our starting point, which we refer to as *one bundle*, bundles are never fragmented [18].

If we allow for fragmentation of responses, N_r may only have one or more fragments of the resource (except if N_r is now a gateway node), a fragment of U being denoted as $U[a,b]$ which indicates that the fragment starts at offset a, ends at offset b (both inclusive), and is of size $b-a+1$; with $U = U[0, S(U)-1]$. In this case, N_r returns $U[a,b]$ but forwards the request without change, indicating which fragments have already been returned, or splitting it up into two requests $req(U[0, a-1])$ and $req(U[b+1, S(U)-1])$. We refer to this scenario as *frag x/x*, denoting that resource bundles are split into x fragments.

To simplify the model and the subsequent simulation, we assume that nodes fragment and store resources only at a minimal granularity of F bytes, equal for all nodes, and that each resource consists of an integer number of such fragments:[1] $a - b + 1 = kF$ and $S(U) = lF$, with $k, l = 1, 2, ...$ We model a request for a complete resource $req(U)$ as a sequence of requests for its fragments—$req_i(U[(i-1)F, iF-1])$ with $i = 1, ..., n$ and $n = S(U)/F$—sent out in rapid succession. All requests are responded to individually, even if a responding node N_r holds multiple fragments.[2]

Extending the model further, we allow for erasure-coded transmission of response bundles, with each erasure-coded block treated as a fragment. In a simple case, a gateway node N_g splits the retrieved resource into n fragments—the source blocks—and then adds m coded blocks of equal size, all of which are then sent independently.[3] We make two simplifying assumptions: 1) N_s

[1]This can be achieved by padding which is necessary for erasure coding anyway.

[2]Note that, in practice, the client would issue just a single request (as it would not know the resource size). The first node holding one fragment of the resource—which thus knows the total size and the fragmentation and erasure-coding parameters (if any)—could create a partial response and easily generate the requests for the remaining fragments.

[3]Note that, depending on the erasure coding scheme used, a coded block (symbol) does not necessarily reflect a consecutive portion of the resource (i.e., a source symbol) and source symbols may not be

knows n and m and issues $n + m$ individual requests, one for each fragment—this splitting could easily be done at the first node that has one fragment and thus knows the total size and the encoding parameters. 2) The generation of the erasure-coded fragments takes place at the gateway—which could also be done in any other node in the network [31]. This scheme is referred to as *code n/(n+m)*.

Requests for resource fragments are identified (see next subsection) and treated similarly to requests for the entire resource and so are the corresponding response bundles. This common treatment allows intermediate nodes who do not understand erasure coding or fragmentation to reply to requests if they hold a copy of the resource (fragment) in question. In addition, the nodes which do not have functionality to perform the look-ups in the forwarding buffer or cache may still participate with plain forwarding functionality. If a match for one fragment is found, the reply is generated and the request for this fragment is dropped while the requests for other fragments are forwarded further.

3.3 Protocol Aspects

We extend the bundle protocol specification by an *Application Hints* header block that carries sufficient information for identifying the resource contained in a bundle (e.g., its URI), the application protocol and operation, the application-layer lifetime [18]. We add information about the total bundle size, a bundle fragment offset and size or the erasure coding scheme (if any) and the symbol number—which are included in every bundle. Finally, we use the bundle layer identity for loop detection and to prevent bundles from oscillating between nodes.

The per bundle overhead incurred is 4 bytes for the extension block, expected 4+ bytes for the application-layer lifetime, variable length fields for the fragment offsets and the total bundle size, 1 bytes for the operation type, 8 bytes for the erasure coding information, and a variable length field to carry the resource identification. Assuming, e.g., 100 bytes for the URI length, the per bundle overhead will be some 120 bytes. This appears acceptable compared to the typical size of, e.g., a web requests (some 500–700 bytes) and does not make the request grow beyond the typical MTU size and become negligible compared to the typical size of web page contents.

Above the bundle protocol and our extensions, we assume an application protocol in which two requests from different nodes can be satisfied by the same response bundle (except for the modified destination address), given an appropriate bundle encapsulation.[4] This holds, e.g., for many types of HTTP responses, netnews articles, RSS feeds, and mails (to mailing lists) and thus addresses quite a few applications that can benefit from sharable and/or cachable contents.

Furthermore, we assume for now that, if security mechanisms (e.g., for origin authentication and integrity protection) are applied, they do not affect re-targeting a response. This means that content protection is only applied to those parts of the message that remain unchanged, e.g., by including an S/MIME signed message in the bundle payload block body rather than signing the entire bundle. But this kind of content protection independent of the transport is desirable for cached contents anyway.

Finally, if not all response fragments can be recovered it may be beneficial if the application protocol allows retrieving only parts of the resource from the Internet if it has not changed (as achievable by the HTTP headers *Range:* and *If-Modified-Since:*). Achieving selective retrieval may also be replicated at the bundle layer using our fragment selection mechanisms described above.

transmitted at all. Hence, we label the coded fragments by numbers rather than byte ranges.

[4]Alternatively, the application protocol must be known to the node so that a matching response can be created from a stored bundle.

4. SIMULATIONS

We have extended the functionality of the publicly available Java-based *dtnsim* [9] to include the functionality described in sections 3.2 and 3.3. For investigating protocol behavior in our scenarios, we run simulations using publicly available mobility trace [6]. The trace provides behavior of 100 users at MIT recorded during the academic year 2004-2005. For our simulation reported below, we have used the "devicespan" information which provides the inter-personal Bluetooth connectivity. The connectivity periods are recorded by using commodity mobile phones, which fits our envisioned usage scenario.

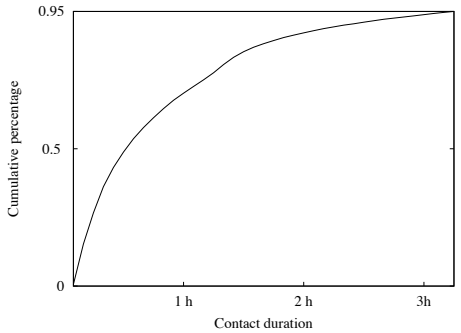

Figure 3: Contact durations in the mobility trace.

Figure 3 shows the cumulative distribution function of contact durations between the nodes in our simulations. Granularity of 5 minutes was used in plotting the distribution and we can observe how the majority of the connections are relatively short-lived. We use a trace with over 62,000 bidirectional connections among 100 nodes over 288 days and make the following assumptions: The nodes communicate through bidirectional wireless links with 100 ms delay and a capacity of 2.1 Mbit/s (Bluetooth EDR). Bundles (and fragments) are transmitted in an all-or-nothing fashion: if the link breaks before the transfer is complete, the bundle is dropped on the receiving and retried on the sending side. Reactive fragmentation [26] is not used.

Each node contributes 512 MB of buffer space for (short-term) DTN forwarding using a FIFO queue. If cooperative caching is enabled, 256 MB out of these are reserved for longer term caching. Requests for resources or resource fragments may be answered from the regular forwarding queue and its part "protected" for caching. Regularly queued bundles have a TTL of one hour, whereas those in the cache part are kept for 12 hours. Messages are discarded after eight hops (as we have observed that responses generally come from close by) or when their TTL expires. Bundles are also removed from the queue after successful forwarding. If the queue—regular or cache part–is full when a new bundle arrives, the arriving bundle is dropped. Our simulation results have shown that the queue capacity of 256 MB is never reached, i.e., the observed communication performance is influenced by the user contacts rather than by congestion.

For our simulations, we have added client and gateway functionality to a subset of nodes. From those nodes having large numbers of Bluetooth contacts, four nodes (101, 102, 14, 30) were chosen to act as clients and four nodes (79, 86, 91, 88) to act as storage gateways. The clients send requests for files that have sizes picked

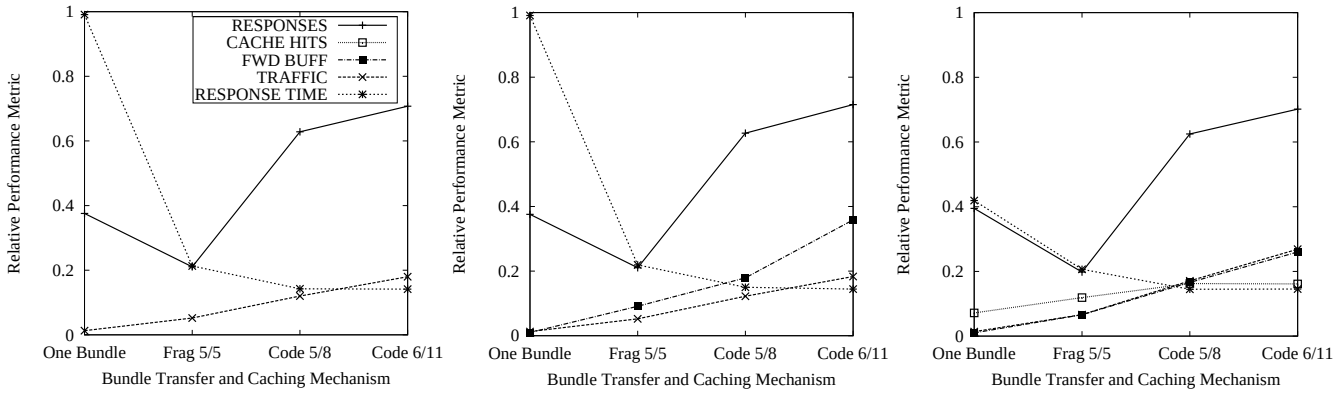

Figure 4: FC, responses from a) only gateways (left), b) also forwarding buffers (center), c) enhanced caching (right).

evenly distributed from {600, 1200, 1800} KB. To emphasize the community aspect, all clients share a common core interest of six resources out of which 70% of the requests are chosen; the other 30% of are selected from 400 other possible URIs. The requests are sent out from each client, at application level, at 100 minute (small variations added to avoid synchronization) intervals.

The clients are capable of sending bundle requests in plain, fragmented, or coded form. When a client requests a resource in coded form it sends separate request for each of the $n + m$ fragments and starts decoding when the first n fragments have arrived. With extra fragments are discarded by the client.

In figures 4 and 5, we show the impact of redundancy and caching for two extreme routing strategies—*First Contact* (section 4.1) and *flooding* (section 4.2)—together with several different transfer strategies as per section 3.2: non-fragmented *one bundle*, fragmented *Frag 5/5*, and coded (*Code 5/8* and *Code 6/11*). The values for m and n can be chosen freely but we limit our observations to two alternatives, one being significantly more redundant than the other. The three subfigures show performance metrics for a) end-to-end communication without caching support, b) caching support in the regular forwarding buffer (TTL=1 h, messages dropped after forwarding), and c) enhanced caching for half of the forwarding buffer (TTL=12 h, messages kept after forwarding). With c), bundles or fragments are cached at a probability of 50%.[5] The reference values to scale both figures are provided in table 4.

Table 1: Reference values for scaling figures 4 and 5.

Metric	Value
Traffic during simulation	450 GB
Average response time	2500 s
Number of results	2500
Number of cache hits	2500
Number of forwarding buffer responses	200

4.1 Non-Redundant Forwarding

First, we start by investigating a protocol that does not create multiple copies of a message in the intermediate nodes. We use the *First Contact* (FC) forwarding [9] with a minor extension to prevent message oscillation. This forwarding scheme simply chooses the first available link and forwards the first message(s) in the FIFO queue. If multiple links are available simultaneously one of them is chosen randomly.

The simulation results are shown in figure 4. Without caching (a), the one bundle strategy manages to deliver a larger amount of responses than the simple fragmentation. This does not come as a surprise as multiple fragments need to be delivered to lead to successful delivery of the entire bundle. But fragmentation significantly reduces the response delay which suggests that contacts are often too short to allow for transmitting entire large messages (so that the fragmentation capability appears crucial). Both coding approaches increase the success in the message delivery beyond the one bundle strategy and reduce message delivery time further—which is in line with other findings (e.g., [30]). As expected, the total traffic grows with fragmentation (smaller fragments propagate further) and increases further with erasure coding since overhead is added.

For FC forwarding, allowing responses from bundles or fragments stored in the forwarding buffer (b) has only a marginal effect on the number of responses or the response delay: both figures are largely identical. The line *FWD BUFF* indicates that less than 10% responses come from the forwarding buffer—which is not surprising for a protocol that only maintains a single copy of a message in the network.

We continue by enhancing the bundle lifetime and maintaining a copy in the dedicated cache part of the buffer even after forwarding (c), the number of responses increases marginally and the response time drops drastically for the one bundle case—while remaining roughly the same for fragmented and coded bundle transmission. Again, FC forwarding does not spread the messages widely, thereby limiting the likelihood for cache hits.

4.2 Redundant Forwarding

We now look at a forwarding algorithm that adds redundancy and thus potentially increases the spread of each message. The flooding approach also uses a FIFO queue but forwards the message to all available contacts at a time. This opportunistically introduces redundancy while messages are being transferred. Messages are deleted from the queue if at least one forwarding was successful. Hence, the algorithm is less aggressive than epidemic routing. Figure 5 illustrates the simulation results for the same three scenarios.

As expected (remember that we do not face congestion), the performance without any caching (a) is far better than for FC forward-

[5]We have run simulations allowing either all or at most one fragment per message to be cached at a single node, with little difference in the resulting client performance. We report only on the latter.

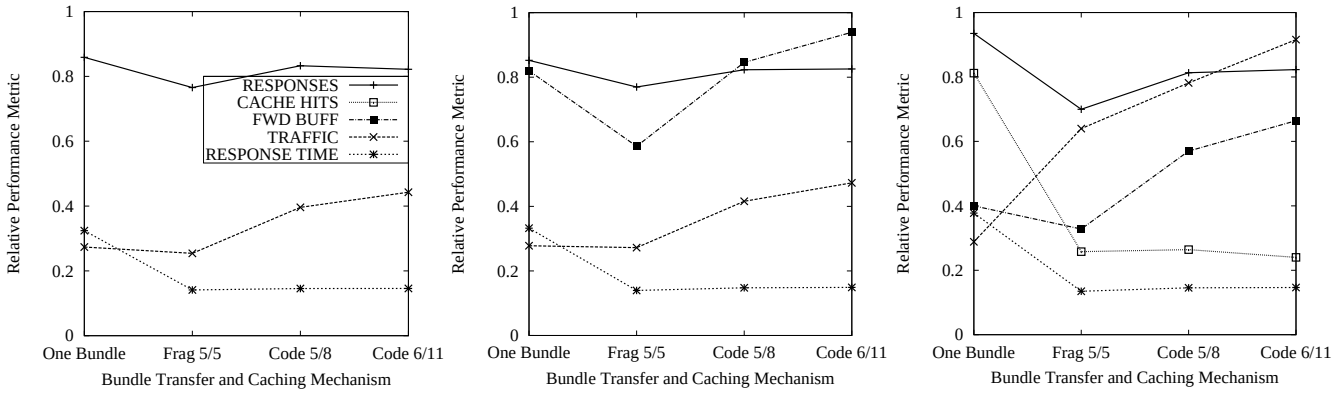

Figure 5: Flooding, responses from a) only gateways (left), b) also forwarding buffers (center), c) enhanced caching (right).

ing and the response time is much lower. Fragmentation reduces the response time further but coding does not yield additional reductions. The message delivery rate is lower with fragmentation, not reaching the one bundle case even with erasure coding—because the complete message already gets widely replicated so that retrieving it is easier than many fragments.

Allowing responses from the forwarding buffer (b) shows that some requests may be satisfied by this kind of caching but there is only a marginal overall improvement: the inter-contact times are too long for the 1 h caching period in the regular queue and bundle deletion after forwarding reduces the availability further. Using the enhanced caching (c), i.e., keeping messages for 12 h, improves the response rate for the one bundle case, reduces it for fragmentation, and leaves it unchanged for erasure coding. The response time increases slightly for the one bundle case (more responses arrive) and remains unchanged otherwise; total traffic increases for (b) and (c) as expected. With (c), fewer answers are generated from the forwarding buffer as the cache maintains the bundles longer.

4.3 Observation Summary

Combining the findings from both simulations, we observe that adding redundancy—either through erasure coding in the endpoints or by using flooding—improves the retrieval performance. Applying both, however, does not lead to further improvements in resource retrieval from within the community. Caching in intermediate nodes may provide additional performance gain, provided that the bundles are kept sufficiently long—which calls for considering application layer lifetimes for extended caching. This also motivates differentiating between bundles containing resources that can be re-used for satisfying further requests and those that simply need forwarding to their destination but have no value for third parties.

Looking closer at end-to-end delivery performance, we can observe how the response time (latency) always decreases with fragmentation and further with coding strategies. However, even if a small response time is a desirable property, at the same time it is important to maintain sufficient communication reliability. For reliability, the redundancy of transferred bundles needed to be added. Adding too much redundancy (both coding and flooding), however, leads to fewer responses being delivered. While this is particularly true if congestion occurs and bundles need to be dropped from the queues, we found this even in our simulations without congestion because the longer FIFO queues reduced the chances for a bundle to be forwarded during the relatively short contact periods.

We have confirmed our above observations on the protocol behavior with other traces which do not report upon in detail. We have also conducted simulations using the Bluetooth traces collected by the Haggle project [14] which captures community interactions between students of the university and some fixed infrastructure nodes. Moreover, we have used a challenging artificial network scenario with rapidly changing link states, similar to the simulation we used in [18].

5. CONCLUSION

In this paper, we have assumed mechanisms for cooperative caching in mobile DTN nodes and investigated how different redundancy mechanisms impact communication performance of mobile clients retrieving resources from an infrastructure network. We have considered two points in the spectrum of routing protocols, one allowing for just a single copy of each message in the system and one variant of flooding, and limited ourselves to simple drop-tail FIFO queuing, with static storage space allocations. We observe that applying erasure coding at the application layer may improve performance as may DTN-layer flooding. Caching in intermediaries may provide further gains provided that the storage period is sufficiently long and bundles are kept after forwarding. Given that flooding may easily lead to congestion, the more sensible application-controlled approach peered with caching is clearly preferable.

These findings give rise to numerous directions for future work: in particular, routing protocols in-between these two extremes need to be investigated: cooperative caching is expected to achieve the best gain if the information is spread widely but not everywhere, to allow exploiting the potential locality of references in communities. In addition, further queuing and replacement strategies peered with dynamically dividing storage space between the forwarding and caching buffer deserve attention—particularly in scenarios exhibiting congestion. Adapting such strategies based upon application parameters—such as bundle lifetime and size—may allow further optimizations. Finally, we have also considered distributed storage [18] to which we will apply the erasure coded opportunistic caching concepts, so that mobile user communities can share local resources as easily as retrieving others from the Internet.

6. REFERENCES

[1] A. Bassi, M. Beck, G. Fagg, T. Moore, J. S. Plank, M. Swany, and R. Wolski. The internet backplane protocol: A study in resource sharing. In *International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May 2002.

[2] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *INFOCOM*, pages 275–283, New York, NY, Mar. 1999. IEEE.

[3] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H.Weiss. Delay-Tolerant Network Architecture. Internet Draft draft-irtf-dtnrg-arch-05, Work in progress, March 2006.

[4] R. L. Collins and J. S. Plank. Assessing the performance of erasure codes in the wide-area. In *DSN-05: International Conference on Dependable Systems and Networks*, Yokohama, Japan, 2005. IEEE.

[5] A. G. Dimakis, P.Brighten, M. J. Wainwright, and K. Ramchandran. Network Coding for Distributed Storage Systems. 2007.

[6] N. Eagle and A. S. Pentland. CRAWDAD data set mit/reality (v. 2005-07-01). Downloaded from http://crawdad.cs.dartmouth.edu/mit/reality, July 2005.

[7] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proceedings of ACM SIGCOMM 2003*, pages 27–36, August 2003.

[8] S. Jain, M. Demmer, R. Patra, and K. Fall. Using Redundancy to Cope with Failures in a Delay Tolerant Network. In *Proceedings of ACM SIGCOMM*, 2005.

[9] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *Proceedings of the ACM SIGCOMM*, 2004.

[10] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth codes: maximizing sensor network data persistence. *ACM SIGCOMM CCR*, 36(4):255–266, 2006.

[11] J. Kangasharju, K. W. Ross, and D. A. Turner. Optimizing File Availability in Peer-to-Peer Content Distribution. In *to appear in the Proceedings of Infocom*, 2007.

[12] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. November 2000.

[13] W. H. O. Lau, M. Kumar, and S. Venkatesh. A cooperative cache architecture in support of caching multimedia objects in manets. In *WoWMoM '02: Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, pages 56–63, New York, NY, USA, 2002.

[14] J. Leguay, A. Lindgren, J. Scott, T. Friedman, J. Crowcroft, and P. Hui. CRAWDAD trace set upmc/content/imote (v. 2006-11-17). Available at http://crawdad.cs.dartmouth.edu/upmc/content/imote, Nov. 2006.

[15] O. Mukhtar and J. Ott. Backup and Bypass: Introducing DTN-based Ad-hoc Networking to Mobile Phones. In *Proceedings of RealMAN 2006*, May 2006.

[16] J. Ott. Application Protocol Design Considerations for a Mobile Internet. In *Proceedings of the 1st ACM MobiArch Workshop*, December 2006.

[17] J. Ott and D. Kutscher. Bundling the Web: HTTP over DTN. In *Proceedings of WNEPT 2006*, August 2006.

[18] J. Ott and M. Pitkänen. DTN-based Content Storage and Retrieval. In *The First IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC)*, June 2007.

[19] L. Pelusi, A. Passarella, and M. Conti. Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks. *IEEE Comm. Magazine*, November 2006.

[20] M. Pitkanen, J. Karppinen, and M. Swany. GridBlocks DISK - Distributed Inexpensive Storage with K-availability. In *Proceedings of HPDC 2006*, 2006.

[21] M. Pitkanen, R. Moussa, M. Swany, and T. Niemi. Erasure codes for increasing the availability of grid data storage. In *Proceedings of ICIW06*, 2006.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. 2001.

[23] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *SIGCOMM Comput. Commun. Rev.*, 27(2):24–36, 1997.

[24] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM Press.

[25] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A Networking Architecture Designed Around Mobile Users. In *Proceedings of IFIP WONS*, Les Ménuires, France, January 2006.

[26] K. Scott and S. Burleigh. Bundle Protocol Specification. Internet Draft draft-irtf-dtnrg-bundle-spec-08, Work in progress, December 2006.

[27] M. Seligman. Storage usage of custody transfer in delay tolerant networks with intermittent connectivity. In *ICWN*, pages 386–392, 2006.

[28] M. Seligman, K. Fall, and P. Mundur. Alternative Custodians for Congestion Control in Delay-tolerant Networks. In *Proceedings of ACM SIGCOMM Workshop on Challenged Networks (CHANTS)*, September 2006.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. 2001.

[30] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure-Coding Based Routing for Opportunistic Networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[31] J. Widmer and J.-Y. L. Boudec. Network Coding for Efficient Communication in Extreme Networks. In *ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, 2005.

[32] L. Yin and G. Cao. Supporting Cooperative Caching in Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 5(1):77–89, January 2006.

[33] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys and Tutorials*, 8(4):24–37, January 2006.