FFI-rapport 2013/02926

# On caching in military networks

Frank T. Johnsen, Trude H. Bloebaum and Ketil Lund

FFI Forsvarets forskningsinstitutt

# On caching in military networks

Frank T. Johnsen, Trude H. Bloebaum and Ketil Lund

Norwegian Defence Research Establishment (FFI)

27 January 2014

FFI-rapport 2013/02926

1277

P: ISBN 978-82-464-2336-4
E: ISBN 978-82-464-2337-1

## Keywords

Nettverksbasert Forsvar

Mellomlagring

Kooperativ caching

DSProxy

SOA

## Approved by

| | |
|---|---|
| Bjørn Jervell Hansen (for Rolf Rasmussen) | Project Manager |
| Anders Eggen | Director |

## English summary

This report presents different approaches to caching, covering existing and suggested solutions for both civil systems and for use within NATO. The idea is to provide an overview of different approaches to caching in the context of Network Based Defence, with a particular focus on possible related long-term research activities to be pursued in FFI project 1277 “Information and Integration Services in INI”.

## Sammendrag

Denne rapporten presenterer ulike former for mellomlagring (engelsk: caching) sett i kontekst av løsninger som finnes og er foreslått både i sivile systemer og spesifikt for NATO. Målet er å gi en oversikt over ulike tilnærminger til mellomlagring med fokus på Nettverksbasert Forsvar, og da spesifikt i forbindelse med mulige langsiktige forskningsaktiviteter i FFI-prosjekt 1277 "Informasjons- og Integrasjonstjenester i INI".

# Contents


| | | |
|---|---|---|
| **1** | **Introduction** | **7** |
| 1.1 | Terminology | 7 |
| 1.2 | Caching explained | 8 |
| 1.3 | NATO standardization and caching | 10 |
| **2** | **Caching** | **11** |
| 2.1 | Overlay networks | 12 |
| 2.2 | Cache replacement algorithms | 12 |
| 2.2.1 | Statistical replacement policies | 13 |
| 2.2.2 | Key-based replacement policies | 13 |
| 2.2.3 | Function-based replacement policies | 13 |
| 2.3 | Proxy caching | 13 |
| 2.3.1 | Web caching | 14 |
| 2.3.2 | Streaming media caching | 15 |
| 2.3.3 | Service caching | 16 |
| **3** | **Cooperative caching** | **18** |
| 3.1 | Cooperation models | 19 |
| 3.2 | Discussion | 20 |
| **4** | **Summary** | **20** |
| | **Bibliography** | **27** |
| **Appendix A** | **Overlay networks** | **28** |
| **Appendix B** | **Introduction to Bloom filters** | **30** |

# 1 Introduction

One of the main goals of NATO Network Enabled Capability (NNEC) is to increase mission effectiveness by interconnecting military entities. Sharing information between decision-makers can help guide them towards making the right decisions at the right time, and a common information infrastructure is needed to facilitate sharing of relevant information across system and national boundaries. This leads to a requirement for an agile communication infrastructure which can support the communication needs of national forces, and at the same time provide interoperability with coalition forces. The information infrastructure will have to support a number of different usage scenarios, from fairly static environments where services are stable, to dynamic environments where both services and service users come and go in a non-deterministic fashion.

In the NNEC vision, the stovepipe systems of the past, offering services and applications to a limited, geographically co-located group of users will shift to a dynamic federation of systems, which will allow services previously only available within the strategic domain to be made available in the tactical domain. Likewise, situational awareness information needs to be fed from the tactical domain into the strategic domain to better inform planning and operational decision making. This necessarily involves increased transfer of data across tactical links with constrained communications pathways in both directions. Measures must be taken to ensure the efficient utilization of such links (e.g., data compression and other optimizations, see [33] for an overview of techniques). However, these previously investigated measures can be combined with *caching* (i.e., creating a local copy of a resource) in an attempt to further reduce the load on the network.

There are several potential benefits to using caching in dynamic network environments such as military tactical networks. The overall load on the network can be reduced, as storing information closer to the users means that information has to traverse fewer links in order to reach its destination. This is particularly useful in cases where the same information is requested by multiple users, and these users are connected to the network on the same side of a network bottleneck such as a reach-back link. Caching a copy of the commonly needed data on the user side of this bottleneck link will then significantly reduce the amount of information that needs to traverse this link.

In addition to lowering the overall load on the network, caching can also give a higher availability of information to the user. Network disruptions along the information path between the content provider and the user can make information unavailable. If this information is cached closer to the user the impact of such disruptions can be reduced.

## 1.1 Terminology

We will now present how we interpret some central terms that are used throughout the report. This allows us to avoid misunderstandings, since different sources sometimes have quite different interpretations of these terms.

- *Streaming* is playback of a continuous multimedia object over a network [36]. Streaming

objects are not downloaded first and then played back, but rather playback is started before they are completely downloaded [35, 36].

- *Metadata* is data describing the multimedia objects, for example a video can have metadata describing its format, bandwidth requirements and content [36].
- The concept of *caching* is storing a copy of data close to the client in order to allow faster access. Another benefit is that communication with the place from which the data originated is reduced [48]. This can also be achieved with *replication*. The difference between the two is subtle but important, and is defined by [64]: Both caching and replication result in making a copy of a resource. Caching is a decision made by the client of a resource (such as a proxy), and not the owner of the resource. If the owner of the resource decides when and how to make a copy, then it is considered to be replication.
- A *proxy* is a node in the network between a client and a server which the traffic passes through. A proxy can be used for several purposes, such as caching, firewalling and content adaption. A proxy which performs caching will often be called a caching proxy [48]. This is a generic term that we will use for all intermediaries in this report. Thus, we do not make the same distinction as the Consultation, Command and Control (C3) Classification Taxonomy, which introduces both proxies and brokers (the latter can be seen as a special-purpose proxy for use with publish/subscribe).
- An *overlay network* is a virtual network built on top of an existing network structure.

## 1.2 Caching explained

Caching, as defined above, involves making copies of resources closer to a client. This copy can be made at the client side, and thus only serve one client, or it can be made in an intermediate node, a proxy, which can serve many clients. The latter is what we focus on here in this report. We will now look at an example to show how a caching proxy operates. For simplicity, we assume one server and one proxy in the network, and this proxy can cater to one or more clients. The proxy contains a cache, a finite storage space it can leverage in its attempt to serve clients. Without a proxy the client would just connect directly to the server and retrieve the desired resource from there. When a proxy is present, the client will use the proxy as if it were the actual server. In that case, the proxy will attempt to serve the client from its local cache, and only if that fails will it forward the request to the server.

Let us explore an example: A client wants to retrieve a resource called *item 2* from the server. It connects to the proxy and requests this resource. However, the resource is not present in the proxy's local cache. This constitutes a *cache miss*, and the proxy forwards the request to the server that satisfies the request. The response can be cached by the proxy (whether or not the proxy chooses to store a resource is determined by a so-called *cache admission policy*). Figure 1.1 shows a cache miss where the proxy adds the item returned by the server to its cache.

Later, another client requests this same resource, *item 2*, from the proxy. In this case the request can indeed be fulfilled by leveraging the proxy's cache. Figure 1.2 shows this – a so-called *cache*

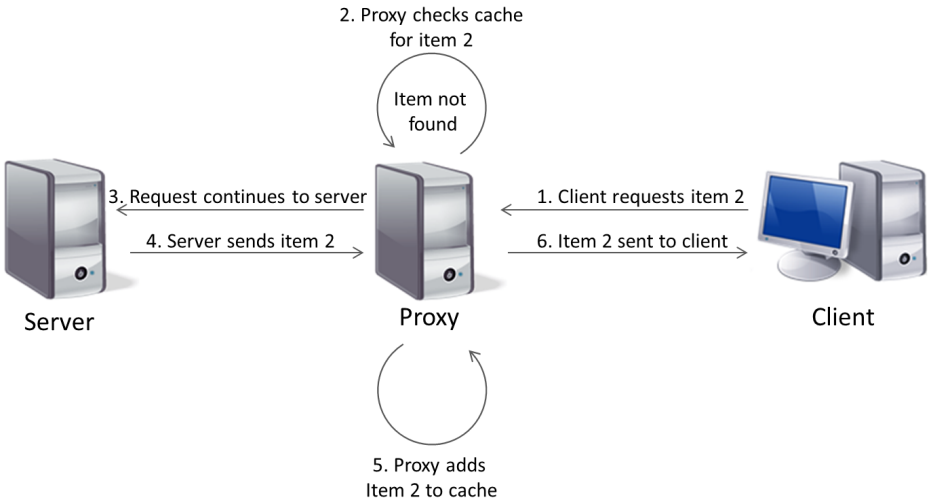


*Figure 1.1 The client issues a request that cannot be fulfilled by the proxy cache. This constitutes a* cache miss.

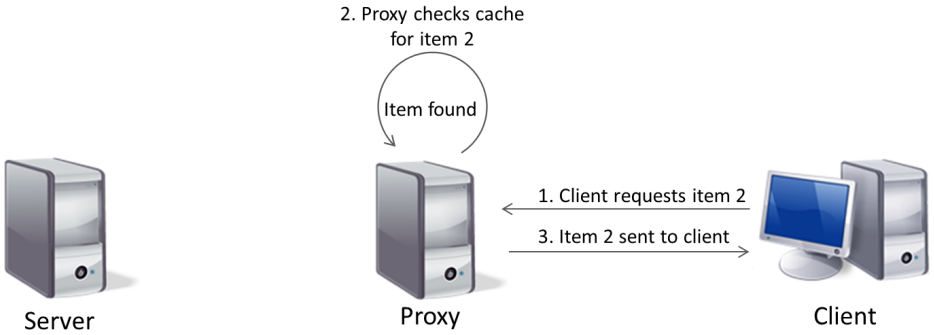


*Figure 1.2 The client issues a request that can be fulfilled by the proxy cache, i.e., a so-called* cache hit.

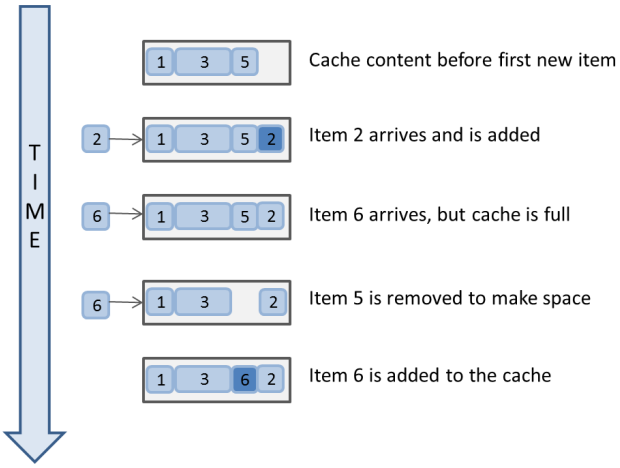


*Figure 1.3 Replacing a resource in the cache with another, i.e., so-called* cache replacement.

*hit*. In this case there is is no connection initiated between the server from which the resource originated and the proxy, as all communication is limited between proxy and client.

A cache is a finite resource. Over time it will fill up with cached resources. When it is full, a resource must be evicted from the cache according to a *cache replacement algorithm* (see Section 2.2) to make room for a new element. Figure 1.3 shows a cached resource being replaced.

## 1.3 NATO standardization and caching

The NATO Interoperability Standards and Profiles (NISP) mention several core concepts that should be addressed in the near term, including caching [13]:

> The NISP will contribute to the core technical model for systems designers to develop new platforms capable of the intensive compilation, cataloging, caching, distribution, and retrieval of data necessary to provide the life cycle information management and necessary information sharing across NATO members.

However, the current NISP series of documents do not provide any technical details on caching. Rather, caching is merely pointed to as one of several important mechanisms in the discussions on interoperability profiles and guidance [14]: "In the mobile environment (radio), mechanisms (e.g. Caching) should however be provided so as to compensate for any brief network fluctuations."

The C3 Classification Taxonomy (*C3 Taxonomy* for short) is a categorization of the functionality that is expected to be found in a networking information infrastructure (NII). It points to capabilities and concepts relevant to creating systems for C3 in NATO. The C3 Taxonomy also acknowledges caching as an important aspect in a NII. The taxonomy identifies *CIS Capabilities*, that are further divided into *User-Facing Capabilities* and *Technical Services*. The technical services encompass all anticipated technical services in NII, including *Core Enterprise Services* which are a major focus at NATO venues like *TIDE* [34], and also in FFI project 1277 "Information- and Integration Services in INI". Core Enterprise Services can, according to the C3 Taxonomy, be further divided into *Enterprise Support Services*, *Infrastructure Services*, and *SOA Platform Services*. SOA Platform Services cover all services expected to be employed in developing a SOA platform, where the *Message-Oriented Middleware Services* play an important role. The C3 Taxonomy introduces these services as follows:

> The Message-Oriented Middleware Services provide functionality to support the exchange of messages (data structures) between data producer and consumer services, independent of the message format (XML, binary, etc.) and content.
>
> Message-Oriented Middleware Services support different models of message exchange (direct, brokered, queues), exchange patterns (request/response, publish/subscribe, for solicit response (polling for response), and for fire and forget), topologies (one-to-one, one-to-many) and modes of delivery (synchronous, asyn-

chronous, long running). They also provide the support for routing, addressing, and caching.

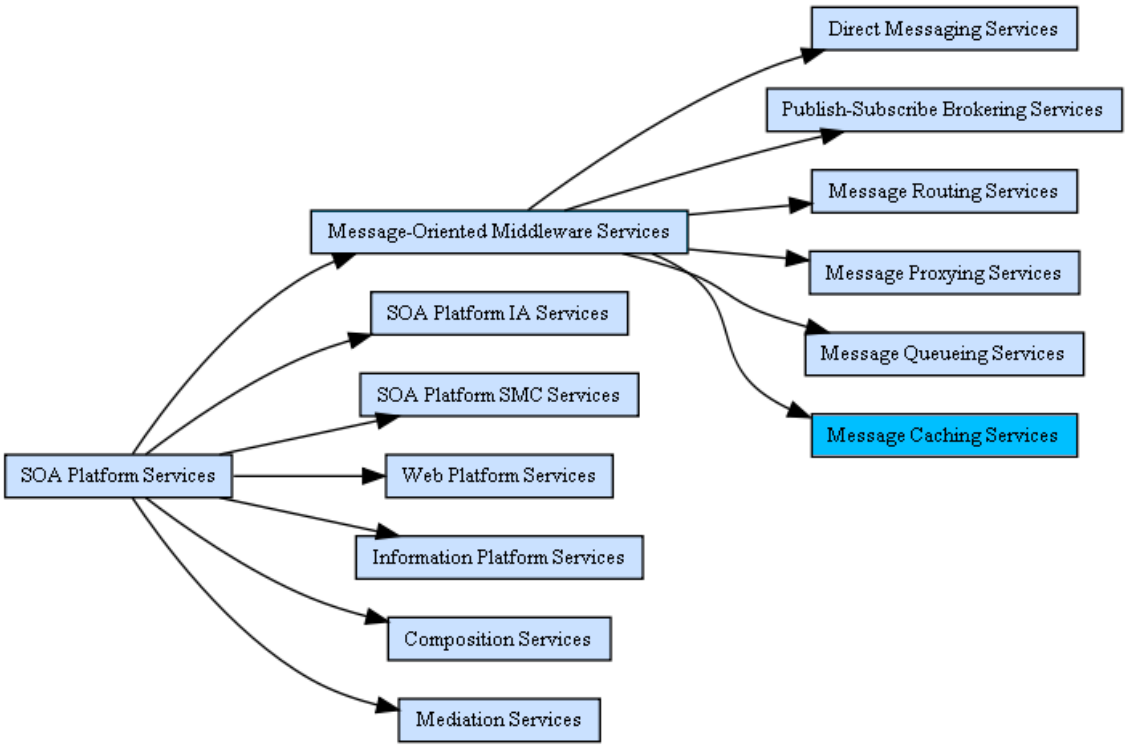


*Figure 1.4 C3 Taxonomy SOA Platform Services*

As illustrated by Figure 1.4, the C3 Taxonomy further subdivides Message-Oriented Middleware Services, introducing concepts such as Message Routing, Queuing, Proxying, and Caching that support efficient message exchange within the NII. In this report we focus on the aspects related to *Message Caching Services*. For further discussion about the C3 Taxonomy and how it relates to the Norwegian defence information infrastructure, see [10].

# 2 Caching

Caching is widely used both in hardware and software to enhance performance. In hardware both instruction and data caches are used to speed up CPU and bus operations [46]. In software systems caching can be of both data and computational results, as well as of connections in networking contexts [67]. In databases the term is used about the storing of intermediate results [15]. In an operating systems context caching is used about block- or file caches in file systems, and about page replacement algorithms in virtual memory management systems [62]. Various techniques related to networking also use concepts of caching, like file storage in distributed file systems [62], web caching [48] and caching of streaming media [27]. In this report we are concerned with software caching, with the purpose of reducing network load. We also briefly introduce overlay networks, as they can play a role in cooperative caching.

## 2.1 Overlay networks

Both proxy caching schemes and overlay network techniques have developed since they were originally introduced. Early solutions were highly static in nature; both proxy caches and overlay networks were manually configured. In the beginning proxy servers were stand-alone machines, but later evolved into shared caches by utilizing each others' storage space by cooperation. Originally cooperation was done in a static manner by using techniques such as URL hashing, organizing the proxies into hierarchies, or by grouping proxies into pools with shared directories. Later, schemes evolved which are more dynamic in nature, by allowing the proxies to form groups based on certain network properties.

Overlay networks developed from manual configuration to automatic systems for integrating new nodes into the network. The first challenge that was addressed was taking the placement of the nodes in the Internet into account when building an overlay network, so called topology awareness. However, Internet network dynamics were not taken into account, leading to reduced overlay network performance when *traffic shifts*[1] occurred. This problem was resolved by introducing measurement-based overlay networks that automatically adapt themselves to changing network conditions. In addition, focus has shifted from application specific solutions to overlay networks providing general support for a wide array of purposes.

One example is the ongoing standardization efforts by the IEEE P1903 Working Group (see `http://grouper.ieee.org/groups/ngson/`) for creating the so-called Next Generation Service Overlay Networks (NGSON). For military networks overlays have not been adopted to the same extent as in the civil domain yet. However, there are some experimental initiatives suggesting using P2P networks (which can be seen as a form of overlay) for publish/subscribe [25], and service discovery [57]. Overlays are discussed further in Appendix A.

## 2.2 Cache replacement algorithms

The heart of any caching scheme is the cache replacement algorithm; the policy which is applied when evicting elements from the cache. A software cache is kept in memory, on a hard drive, or a combination of these. Either way the cache is a finite resource that needs to be administrated appropriately. One of the key complications of cache replacement policies is that the elements to be cached not necessarily are of homogeneous size. Also, when replacing an element in the cache not only the relative frequency, but also other parameters like transfer time savings and expiration times might be factors worth taking into account. This report categorizes cache replacement algorithms according to the criteria in [6], which divides the replacement schemes into three categories: 1) Statistical replacement policies, 2) Key-based replacement policies, and 3) Function-based replacement policies.

[1] Network load (i.e., network traffic) is often bursty and there can be unpredictable changes and shifts in traffic demand, for instance due to hotspots and flash crowds, or because a link goes down, there are changes in the inter-domain routing, or because traffic in an overlay is redirected [3].

### 2.2.1 Statistical replacement policies

Algorithms which do not take the element size into account when deciding which element to throw out are called *statistical replacement policies*. These policies may be directly extended in order to accommodate for varying size elements by applying the replacement policy to the cache a sufficient number of times to make room for a new element. The difficulty with such policies in general is that they fail to pay sufficient attention to element sizes [6]. These algorithms are fair in the respect that they treat all elements the same. Examples of algorithms falling into this category are FIFO, Second Chance, NRU, LRU, NFU, and aging, all of which are described in [63]. Such algorithms are frequently used in proxy products, for example LRU is the default cache replacement algorithm in the Squid web proxy [60].

### 2.2.2 Key-based replacement policies

The *key-based policies* sort elements based on a primary key, and break ties using a secondary key or even a tertiary key. The idea in using key-based policies is to prioritize some factors over others [6], for example by preferring to keep either large or small elements in the cache. Elements which do not fit the criteria are penalized, making these kinds of algorithms a poor choice in a mixed environment. Examples of such algorithms are LRU-THOLD and LRU-MIN, described in [4].

### 2.2.3 Function-based replacement policies

*Function-based policies* use general functions for the different factors such as latency, connection cost, expiration time, entry time of element in the cache, transfer time cost and last access time since entry [6, 7]. These algorithms are usually more computationally expensive than the statistical or key-based ones. Just like the key-based algorithms they will favor some elements while penalizing others. Examples of algorithms of this type are Lowest Latency First (often also referred to as “Wooster and Abrams” after the names of the creators) [70], LNC-R [53], GD-Size [42], S-LRU [65], and PSS [65].

## 2.3 Proxy caching

The trend in proxy caching has been to evolve from stand-alone caches to schemes where caches cooperate. The cooperation methods have been refined, and there is a shift from static cooperating clusters to proxies forming dynamic cooperation groups reflecting on network changes and cost issues. This trend is evident both in classic web caching schemes and also in schemes for caching of streaming media. With the recent introduction of *Service-Oriented Computing* [31], it is also important to consider caching related to services. Below we present a brief overview of the developments within these three areas:

### 2.3.1 Web caching

Caching is important to reduce network traffic, server load, and improve end-to-end latency. The issue has been studied in detail for web traffic, starting with CERN httpd [9]. Later, the Harvest project [12, 17] improved caching by introducing cache hierarchies, a concept that was further utilized in the Squid project [2, 69] where cooperative caching was introduced.

A simple way to allow inter-proxy cooperation is to divide the URLs among the proxies. This technique is called URL hashing and makes each proxy responsible for processing requests for the URLs that belongs to their partition. The Cache Array Routing Protocol [1] implements URL hashing and is supported by many proxies.

Another approach to cooperation is explicit tracking of cached objects in a directory service. Proxies will then query the directory when it is unable to obtain an object from its local cache (a so-called *local miss*) to find a cached copy, rather than broadcasting queries or performing URL hashing. The CRISP proxy [19, 23, 24] implements such a directory-based cooperation model. By fully replicating the directory service on every proxy, one eliminates the need for synchronous directory queries, an approach which has been used in the Relais cache [38]. Full replication of the directory leads to space overhead in the proxies. To deal with this issue one can try to limit the directory replicas by using a Replicated Partial Directory (RPD) [22]. The RPD approach maintains a separate directory service that keeps full location information. It is also possible to reduce space overhead by storing the directory in a compressed form. Summary cache [21] by Fan et al. is one such approach which uses *Bloom filters* (see Appendix B) to represent a compact directory. An alternative implementation by Rousskov and Wessels is called cache digests [52]. One can also reduce the space overhead for the replicated directory by using a coarser granularity. This is done in the CacheMesh project [68], where location tracking is done at the granularity of entire web sites.

The approaches considered so far are not fully suitable for the scale of the Internet. In the global Internet, origin servers may often be closer, better connected, or more powerful than remote proxies. Thus, deciding between origin servers and remote proxies to serve a local miss is important for cooperative caching. Cache routing is one approach that addresses this issue by always forwarding requests to a neighbor proxy in the direction of the origin server. To perform cache routing one needs network topology information, which can be routing tables from network routers as suggested by Grimm et al. [26] or to mimic a distance vector protocol as suggested by Michel et al. [39]. Cache routing avoids bad decisions in choosing between a remote proxy and the origin server in terms of network proximity, but does not necessarily make optimal decisions. For example, the network path towards the origin server through the proxies may be longer and slower than going directly to the origin server. Vicinity caching [47] makes explicit choices between remote proxies and the origin server based on observed costs of fetching objects from various sources. In this approach each proxy defines a *vicinity* of other proxies relative to an origin server. This vicinity contains only those proxies it is preferable to contact and fetch objects from in addition to the origin server. Each proxy tracks object locations only within the vicinity, and will not forward a re-

quest to a remote proxy unless its directory indicates that the object is cached there. This approach suggested by Rabinovich et al. can be used for static, preconfigured vicinities, but they outline how adaption is possible by allowing the scheme to measure distance or other cost metrics and change the vicinities based on this. Such *dynamic vicinities* would have the benefit of letting the cooperation scheme adapt to changes in the network characteristics as they occur. Since overlay networks can perform the necessary monitoring and route maintenance to support such a scheme, it would be beneficial to apply the ideas of the vicinity cache on top of an overlay network.

### 2.3.2 Streaming media caching

As argued in [36], data types can be divided into time-independent, i.e., *discrete data*, and data like audio and video that have a temporal aspect, i.e., *continuous data*. Sensors can also create other continuous streams of data besides audio/video. Thus, streaming media requires other caching solutions than those used for classic discrete web content. To increase the efficiency of the distribution system and overcome the varying network quality of the Internet, partial caching approaches have been developed. There are two general approaches; *temporal partial caching*, i.e., caching a part of the entire movie's length, and *quality partial caching*, i.e., caching only part of the quality (for example by storing only the basic layer in a layer encoded video).

The partial caching approach called proxy prefix caching [55] stores the first part of the movie in a proxy called a prefix cache, and delivers the rest of the movie from a root server for that movie. This original version by Rexford et al. does not utilize multicast or optimize the selection of the prefix length. Several variations have been introduced to reduce costs predictably as presented in [66]. Similar approaches have also been introduced independently as Mcache [49].

Zhang et al. introduce video staging [71], a partial caching approach aimed at smoothing and reduction of resource requirements. Layered caching [28, 45, 51] tries to overcome vulnerability to changes in network quality by caching the entire length of the movie but reducing its quality if the movie's popularity does not warrant its complete storage. These schemes assume the availability of a layer encoded video. Another approach to quality partial caching which does not require layer encoding is employed by the QBIX proxy [54], where in-proxy rescaling of videos is performed. In such a proxy an entire movie is cached, and when its popularity diminishes it is not evicted from the cache but converted to lower quality, thus requiring less storage space.

A look at a distribution system with several layers of caches is taken by Chan and Tobagi, who investigate several related approaches for the optimized delivery of movies in a hierarchical distribution system, where each server may hold part of the length of a movie [16].

Acharya et al. proposed the MiddleMan [5] cooperative caching techniques, which utilize the aggregate storage of client machines. Videos are split into equal size segments, which can then be striped between proxies to form a simple kind of load balancing. Cooperation is coordinated by a centralized coordinator, which is responsible for a fixed set of caches which they call a *proxy cluster*. By allowing coordinators to exchange information, it is possible to extend the MiddleMan

scheme to feature cooperation between proxy clusters as well.

An approach which combines several of the above mentioned techniques is the Self-Organizing Cooperative Caching Architecture (SOCCER) [30]. Its self-organizing distributed nature contrasts with the centralized approach of MiddleMan. The SOCCER architecture uses both caching and multicast techniques, and allows the proxies to form dynamic meshes for forwarding streaming data. State information is distributed periodically as expanding rings. This ensures that the frequency of state messages exponentially decreases with increasing scope, while it still ensures global visibility over a larger time scale. The scope restrictions are TTL-based[2], and will thus lead to changes in the dynamic meshes when routes change. SOCCER employs a cost function to decide which proxies to interact with, making it a suitable scheme for use in a wide area network. MiddleMan, on the other hand, makes no distinction between proxies in a cluster, thus making it most suitable for use within a local area network. Tests have shown that SOCCER yields the best overall performance when compared to hierarchical schemes, thereby indicating that self-organizing cooperation is a beneficial approach to streaming media caching on the Internet [30].

When interconnecting different networks, multicast may not always be supported across different types of technologies even if they are IP networks. In the Internet, service providers typically do not enable routing of multicast packets across network domains. This is also usually the case in military networks. Technically, multicast across network boundaries can be achieved, though, as e.g., discussed in [29]. A natural approach to circumvent dependence on IP multicast would be to employ some sort of overlay network, thus enabling the proxies to use application level multicast where IP multicast is unavailable [48]. See Appendix A for a brief overview of overlay networks.

### 2.3.3 Service caching

Broadly speaking, service caching can be divided into two categories; caching of actual services and caching of messages sent between services and service consumers. Duplication of actual services is often used for load balancing and/or redundancy. However, the services copies are typically located within the same physical location. This is particularly so for services that need to keep some kind of state, for instance accessing a database. Due to uncertainties related to the actual implementation of a service, it is our strong opinion that actual services should not be subject to caching, only replication.

Caching of messages, on the other hand, is primarily used for offloading the network between the service and the service consumer, and for increased delivery reliability. All service requests and responses are intercepted by proxies, and if a proxy has recently relayed a similar request, it simply returns the corresponding response from that request, instead of forwarding the new request all the way to the service.

There are also examples of caching mechanisms that lie somewhere between these two categories.

[2] TTL, short for ”time to live”, refers to a field in the IP packet header. By setting this value to a number *X*, you ensure that the packet is forwarded no more than *X* times. This can be used to limit the scope of multicast packets, and is the fundamental technique behind the ”rings” in e.g. the SOCCER architecture.

Web map service caching (WMS-C) as implemented by the MapProxy [44] is one such example. Normally, a Web map service generates map tiles on the fly for each request. Using WMS-C, a number of map tiles of given scales are generated in advance, and when a map request arrives, it is served from the tile cache instead of the actual Web map service. This way, the request can be served much faster, and the load on the server is significantly reduced.

In this report, we will focus on the message caching category, as this is where most of the remaining research challenges are located. This is also the only category of service caching that is referred to in the C3 Taxonomy.

The C3 Taxonomy argues that the Message Caching Services should provide functionality to conditionally store messages sent between producers and consumers. The core idea being that the messages can be served to consumers later if they need to resynchronize their state, or were unavailable and lost some messages. The Message Caching Services are intended to support synchronous (request/response) and asynchronous (publish/subscribe) communication. Further, the C3 Taxonomy identifies a set of functional requirements for message caching (see Table 2.1).

| *Name* | *Requirements* |
|---|---|
| Store Messages | The Message Caching Services shall store messages. |
| Retrieve Messages | The Message Caching Services shall retrieve messages. |
| Expire Messages | The Message Caching Services shall automatically expire messages stored in the cache. |
| Clear Cache | The Message Caching Services shall allow an administrator to clear the cache. |
| Provide Security | The Message Caching Services shall apply appropriate security to the contents of the cache, and only deliver content to authorized users. |

*Table 2.1 Message Caching Services: Functional Requirements*

These requirements apply to caching of messages related to services, regardless of communication paradigm: Request/response or publish/subscribe.

### Request/response caching

So far NATO has not focused on request/response caching. That is not because it is not beneficial or important, but because the basic infrastructure needs to be in place before it makes sense to start optimizing it. The infrastructure in this instance being the NNEC NII with the complete set of services as described in the C3 Taxonomy, is still far from being a fully implemented reality. That being said, FFI has in a previous project performed some preliminary experiments with a stand-alone proxy cache for caching service data as a part of the Delay and disruption tolerant SOAP proxy (DSProxy) [37, 59]. We think further experiments in this area can be beneficial and aid in the evolution of a viable, national Network Based Defence (NBD). Naturally, any emerging NATO initiatives should be acted upon and harmonized with the national efforts. We think a solution

involving cooperating, caching proxies for request/response caching could be worth pursuing as discussed further in Section 3.2.

### Publish/subscribe caching

In the Core Enterprise Services SOA baseline [18], a set of OASIS specifications called Web Services Notification (WS-Notification) [43] have been selected as the standard for publish/subscribe.

However, initial use of WS-Notification within NATO identified a few shortcomings that were necessary to address. The main problem is that notification consumers are unable to receive notifications that were published prior to the consumer subscribing. Similarly, notifications published during connection errors between publisher and consumer are also lost.

The NCIA addressed this problem by proposing a cache-based solution called *Notification Cache* [41]. This cache categorizes and stores all published notifications, and these can later be retrieved through standardized Web services interfaces, both request/response and publish/subscribe. The Notification Cache is an add-on to the WS-Notification specification, and does not break the standard. Thus, an existing WS-Notification consumer can use a Notification Cache-enabled service without modification.

In addition to an implementation of the Notification Cache specification itself, NCIA has also developed a test suite (using SoapUI), to test the compliance of implementations towards their specification. At CWIX 2013, NCIA used this test suite to validate the Norwegian implementation of the Notification Cache. The implementation was provided by FFI, and used FFI's in-house developed prototype implementation of WS-Notification as its basis.

The FFI implementation only focused on the central parts of the Notification Cache specification, and non-essential parts, such as error handling, were left out. The goal of the implementation was to verify whether the specification of the Notification cache was sufficiently precise and unambiguous to be used as basis for an implementation.

The CWIX experiments showed that although some bugs were identified, the Notification cache specification was to a large extent sufficiently precise to be used as a basis for implementation. However, it is a challenge to achieve full interoperability, even with a well profiled standard. Such profiling will be the subject of future *TIDE Sprints* (see [34] for further information on TIDE and TIDE Sprints). Finally, the CWIX experiments also revealed the need for some minor modifications of the Notification cache specification itself.

## 3 Cooperative caching

Cooperative proxy caching is individual proxies sharing their cached objects with each other's clients [48]. In the remainder of this report cooperative proxy caching will be called cooperative caching. This section gives an overview of some known cooperative caching schemes.

| Cooperation Model | Load Scalability | Geographical Scalability | Location Overhead | Miss Penalty | Cache Sharing | Suitability for NBD |
|---|---|---|---|---|---|---|
| Broadcast queries | low | high | highest | highest | medium | poor |
| Cache hierarchy | low | high | lowest | highest | medium | poor |
| URL hashing | high | lowest | lowest | high | medium | poor |
| Pure directory | high | medium(P) | medium | low | high | OK |
| Replicated directory | medium | high(P) | high | lowest | high | OK |
| Summary cache | high | high(P) | low/med | lowest | high | good |
| Coarse-grained directory | high | low | low/med | high | medium | poor |
| Cache routing | high | high | low/med | ? | medium | OK |
| Vicinity cache | high | high | ? | lowest | high | good |

*Table 3.1 An overview of various cooperation schemes and their properties.*

## 3.1 Cooperation models

There are several cooperation scheme characteristics to take into account when choosing a cooperation scheme [48]:

- Load scalability.
- Geographical scalability.
- Location overhead.
- Agreement level.
- Miss penalty.
- Cache sharing.
- Pruning.

Table 3.1 shows several existing cache cooperation schemes. This table is adapted from [48] to include the last column which rates each scheme's theoretical suitability for NBD. The agreement level has been omitted. In our scenario we want a scheme fitting military networks, so we ignore the agreement level issue since all proxies will be required to have the appropriate trust chain and other security issues sorted before they can cooperate closely. This is beyond the scope of this report, as it is in the domain of FFI project 1294 "ICT security in the cyber domain" and not project 1277. Obviously we want a scheme with good scalability and cache sharing, as well as having a low miss penalty and low overhead. The pruning problem – deciding between remote proxies and origin servers for processing a local miss – is of the utmost importance. It affects the geographical scalability of a scheme. Schemes that do not take the pruning problem into account are marked with a (P) behind their rating in the geographical scalability column. That indicates

that the scheme can only get that particular rating if the scheme is modified to solve the pruning problem.

We see in Table 3.1 that summary cache [21] and vicinity cache [47] look particularly promising. The vicinity cache is the better of the two for actual use since it takes the pruning problem into account. Summary cache can only achieve a high geographical scalability if it is modified to solve the pruning problem. In the following we discuss vicinity cache further in our suggested approach to a cooperative caching design.

### 3.2 Discussion

By using the ideas from the vicinity cache, combined with recent developments in Bloom filters [58] – the compact way of representing directories that summary cache uses – one should get a good cooperative caching scheme with not too high an impact on the network compared to the benefits of such a scheme. Further, we want to make the location overhead of the algorithm fall into the low/med category, which should be possible by combining the caching scheme with link cost knowledge that the overlay possesses. The vicinity cache proposed in [47] requires such link information to perform proxy pruning. [48] argues that the costs for obtaining this is unclear, hence leading to an unknown rating for location management in Table 3.1. We suggest pursuing a network of proxies with an overlay that can perform periodic network monitoring and collect link statistics. Identifying and/or creating such an overlay suitable for deployment in NBD is one area for potential future research. Assuming such an overlay exists, networking information is readily available and can presumably be obtained and used by the caching scheme at low cost.

## 4 Summary

As we have shown in this report, caching can consist of a number of different mechanisms. Which mechanisms to use in a given case will be dependent on several different factors, such as type of data (volatility), type and dynamicity of networks, type of applications, user needs, etc. In addition, the maturity of the different caching mechanisms varies considerably, meaning that the time horizon of the planned use also will be of importance.

Not all data types are equally suited to be cached, depending on the volatility of the information contained within the data object. As a general guideline, the benefits from employing caching techniques increases as data becomes more static, and the number of requests for the same information goes up. Data that rarely changes, but is of common interest to many users, such as maps, will benefit more from caching than information with frequent changes, such as positional information about mobile units. In addition, single usage information, such as VTC data should not be cached.

Furthermore, the main observed benefits of caching can differ from one network setting to the next. In static networks with high capacity data links, the main benefits that caching will yield is a reduction of load on the original content server, and a lowering of the latency observed by

clients. In more dynamic networks the list of observable benefits also includes better network usage efficiency and increased content availability.

In addition, which caching technique is most suitable will vary between network types. In dynamic networks it will for instance be beneficial to use a cooperative caching scheme that does not rely on heavy communication between caches, but rather utilizes already existing information about networking conditions to make its caching decisions.

Some of the potential benefits caching can yield require full fledged cooperative caching schemes, but one can gain significant benefits from just implementing stand-alone caching as well. Because of this we recommend to first look into stand-alone caches, in particular for request/response Web services, as this is not currently covered by NATO standardization. Next, it would be interesting to investigate further optimizations once a stand-alone request/response caching proxy is in place. In that case it would make sense to leverage the ideas from overlay networks and the vicinity cache to create an overlay network with cooperatively caching proxies.

## References


[1] Cache array routing protocol and microsoft proxy server version 2.0. http://www.microsoft.com/technet/archive/proxy/prxcarp.mspx.

[2] Squid web proxy cache. http://www.squid-cache.org/.

[3] Henrik Abrahamsson. Internet traffic management. Mälardalen University Licentiate Thesis No.95, http://soda.swedish-ict.se/3507/1/lic_thesis_henrikabrahamsson.pdf, November 2008.

[4] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. Technical report, Techical Report TR 95-12, Department of Comupter Science, Virginia Polytechnic Institute and State University, July 1995.

[5] Soam Acharya and Brian Smith. Middleman: A video caching proxy server. In *Proceedings of NOSSDAV*, June 2000.

[6] Charu C. Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the world wide web. *Knowledge and Data Engineering*, 11(1):95–107, 1999.

[7] Sarmed AL-Najim. Web caching: Architectures, models and importance to the internet.

[8] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145. ACM Press, 2001.

[9] T. Berners-Lee, A. Lutonen, and H.F. Nielsen. Cern httpd. http://www.w3.org/Daemon/Status.html, 1996.

[10] Trude Hafsøe Bloebaum, Jo Erskine Hannay, Ole-Erik Hedenstad, Svein Haavik, and Frode Lillevold. Architecture for the Norwegian defence information infrastructure (INI) — remarks on the C3 Classification Taxonomy. FFI Report 2013/01729, 2013.

[11] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[12] C.M. Bowman, P.B. Danzig, D.R. Hardy, U. Manber, M.F Schwartz, and D.P. Wessels. Harvest: A scalable, customizable discovery and access system. Technical report, University of Colorado, Boulder, USA, 1994. Tech. Rep. CU-CS-732-94.

[13] C3B Interoperability Profiles Capability Team. NATO Interoperability Standards and Profiles Volume 2: Near Term. Allied Data Publication 34 (ADatP-34(G)) (http://nhqc3s.nato.int/architecture/_docs/NISP/pdf/NISP-Vol2-v7-release.pdf), 8 March 2013.

[14] C3B Interoperability Profiles Capability Team. NATO Interoperability Standards and Profiles Volume 4: Interoperability Profiles and Guidance. Allied Data Publication 34 (ADatP-34(G)) (http://nhqc3s.nato.int/architecture/_docs/NISP/pdf/NISP-Vol4-v7-release.pdf), 8 March 2013.

[15] R. G. Casey and I. M. Osman. Replacement algorithms for storage management in relational data bases. *The Computer Journal*, 19(4):306–314, november 1976.

[16] S.-H. Gary Chan and Fourad A. Tobagi. Distributed Server Architectures for Networked Video Services. *IEEE/ACM Transactions on Networking*, 9(2):125–136, April 2001.

[17] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A hierarchial internet object cache. In *Proceedings of the 1996 Usenix Technical Conference*, 1996.

[18] Consultation, Command and Control Board (C3B). CORE ENTERPRISE SERVICES STANDARDS RECOMMENDATIONS: THE SOA BASELINE PROFILE VERSION 1.7. Enclosure 1 to AC/322-N(2011)0205, NATO Unclassified releasable to EAPC/PFP, 11 November 2011.

[19] Syam Gadde et al. The CRISP web cache. http://www.cs.duke.edu/ari/cisi/crisp/, 1999.

[20] Trude Hafsøe et al. Towards Automatic Route Maintenance in QoS-Aware Overlay Networks. Technical Report 329, University of Oslo, November 2005. ISBN 82-7368-284-6.

[21] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[22] S. Gadde, J. Chase, and M. Rabinovich. Directory structures for scalable Internet caches. Technical report, Department of Computer Science, Duke University, 1997. CD-1997-18.

[23] S. Gadde, J. Chase, and M. Rabinovich. A taste of Crispy Squid. In *Proceedings of the Workshop on Internet Server Performance*, 1998.

[24] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large Iinternet caches. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 93–98, 1997.

[25] T. Ginzler. A robust and scalable peer-to-peer publish/subscribe mechanism. In *Communications and Information Systems Conference (MCC), 2012 Military*, pages 1–6, 2012.

[26] C. Grimm, M. Neitzner, H. Pralle, and J.-S. Vöckler. Request routing in cache meshes. *Computer Networks and ISDN Systems*, 30:2269–2278, 1998.

[27] Carsten Griwodz. *Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure*. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, Apr 2000.

[28] Carsten Griwodz, Frank T. Johnsen, Simen Rekkedal, and Pål Halvorsen. Caching of Interactive Multiple Choice MPEG-4 Presentations. in proceedings of the International Workshop on Multimedia Systems and Networking (WMSN 2006), 2006.

[29] M. Hauge, M. A. Brose, , and O. I. Bentstuen. Group communication in tactical networks – a discussion. in proceedings of the Military Communications and Information Systems Conference (MCC), October 7-9, 2013, Saint-Malo, France., 2013.

[30] M. Hofmann, E. Ng, K. Guo, S. Paul, and H. Zhang. Caching techniques for streaming multimedia over the internet. Technical report, Bell Laboratories, April 1999. BL011345-990409-04TM.

[31] M. N. Huhns and M. P. Singh. Service-oriented computing: key concepts and principles. *Internet Computing, IEEE*, 9(1):75–81, Jan-Feb 2005.

[32] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. pages 197–212.

[33] Frank T. Johnsen. Pervasive web services discovery and invocation in military networks. FFI-rapport 2011/00257, 2011.

[34] Frank T. Johnsen and Trude H. Bloebaum. Travel report: TIDE Sprint autumn 2013. FFI-reiserapport 2013/02613, 2013.

[35] Don Larson. Does Multimedia Have a Dark Side? `http://webdeveloper.com/multimedia/multimedia_dark_side.html`, 2000.

[36] Ketil Lund. *Adaptive Disk Scheduling in a Multimedia DBMS*. PhD thesis, University of Oslo, July 2003.

[37] Ketil Lund, Espen Skjervold, Frank T. Johnsen, Trude Hafsøe, and Anders Eggen. Robust Web services in heterogeneous military networks. IEEE Communications Magazine, special issue on military communications, October 2010.

[38] M. Makpangou, G. Pierre, C. Khoury, and N. Dorta. Replicated directory service for weakly consistent replicated caches. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 92–100, 1999.

[39] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching: Towards a new global caching architecture. *Computer Networks and ISDN Systems*, 30:2169–2177, 1998.

[40] Akihiro Nakao, Larry Peterson, and Andy Bavier. A Routing Underlay for Overlay Networks. In *Proceedings of the ACM SIGCOMM Conference*, August 2003.

[41] NCI Agency. TTB Notification Cache V1.1.0. `http://tide.act.nato.int/tidepedia/index.php?title=TTB_Notification_Cache_V1.1.0` (Access required a Tidepedia account), 26 March 2013.

[42] Nicolas Niclausse, Zhen Liu, and Philippe Nain. A new efficient caching policy for the world wide web. http://www-sop.inria.fr/mistral/personnel/Nicolas.Niclausse/articles/wisp98/.

[43] OASIS. OASIS Web Services Notification (WSN) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn, October 11th 2006.

[44] Omniscale. Mapproxy. http://mapproxy.org/, accessed 2013-12-05.

[45] Shantanu Paknikar, Mohan Kankanhalli, K. R. Ramakrishnan, S. H. Srinivasan, and Lek Heng Ngoh. A caching and streaming framework for multimedia. In *Proceedings of the ACM International Multimedia Conference (ACM MM)*, pages 13–20, Marina del Rey, CA, USA, October 2000.

[46] David A. Patterson and John L. Hennessy. *Computer Organization & Design*. Morgan Kaufmann, 2 edition, 1998.

[47] Michael Rabinovich, Jeff Chase, and Syam Gadde. Not all hits are created equal: cooperative proxy caching over a wide-area network. *Computer Networks and ISDN Systems*, 30(22–23):2253–2259, 1998.

[48] Michael Rabinovich and Oliver Spatscheck. *Web caching and replication*. Addison Wesley, 2002.

[49] Sridhar Ramesh, Injong Rhee, and Katherine Guo. Multicast with cache (mcache): An adaptive zero-delay video-on-demand service. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Anchorage, AK, USA, April 2001.

[50] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM'02*, 6 2002.

[51] Reza Rejaie, Haobo Yu, Mark Handley, and Deborah Estrin. Multimedia proxy caching for quality adaptive streaming applications in the Internet. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 980–989, Tel-Aviv, Israel, March 2000.

[52] Alex Rousskov and Duane Wessels. Cache digests. *Computer Networks and ISDN Systems*, 30(22–23):2155–2168, 1998.

[53] Peter Scheuermann, Junho Shim, and Radek Vingralek. A case for delay-conscious caching of Web documents. *Computer Networks and ISDN Systems*, 29(8–13):997–1005, 1997.

[54] Peter Schojer, Laszlo Bözörmenyi, Hermann Hellwagner, Bernhard Penz, and Stefan Podlipnig. Architecture of a quality based intelligent proxy (QBIX) for MPEG-4 videos. In *The Twelfth International World Wide Web Conference.*, pages 394–402. ACM, 2003.

[55] Subhabrata Sen, Jennifer Rexford, and Don Towsley. Proxy Prefix Caching for Multimedia Streams. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1310–1319, New York, NY, USA, March 1999. IEEE Press.

[56] Kai Shen. Saxons: Structure management for scalable overlay service construction. In *USENIX NSDI '04*, pages 281—294, 2004.

[57] Magnus Skjegstad and Frank T. Johnsen. Search+: An efficient peer-to-peer service discovery mechanism. FFI Report 2009/01610, 2009.

[58] Magnus Skjegstad and Torleiv Maseng. Low-complexity set reconciliation using Bloom filters. in proceedings of the 7th ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing (FOMC '11), San Jose, CA, USA, June 9th 2011.

[59] Espen Skjervold, Trude Hafsøe, Frank T. Johnsen, and Ketil Lund. Delay and Disruption Tolerant Web Services for Heterogeneous Networks. IEEE MILCOM, Boston, MA, USA, October 2009.

[60] squid-cache.org. Squid configuration directive cache_replacement_policy. `http://www.squid-cache.org/Doc/config/cache_replacement_policy/`, Accessed 2013-12-04.

[61] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H. Katz. Overqos: Offering internet qos using overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1):11–16, 2003.

[62] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1 edition, 1992.

[63] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2 edition, 2001.

[64] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems*. Prentice Hall, 2002.

[65] Igor Tatarinov. Performance analysis of cache policies for web servers. Technical Report NDSU-CSOR-TR-97-06.

[66] Bing Wang, Subhabrata Sen, Micah Adler, and Don Towsley. Proxy-based Distribution of Streaming Video over Unicast/Multicast Connections. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, New York, NY, USA, June 2002. IEEE Press.

[67] Jia Wang. A survey of Web caching schemes for the Internet. *ACM Computer Communication Review*, 25(9):36–46, 1999.

[68] Zheng Wang. Cachemesh: A distributed cache system for world wide web. In *Proceedings of the 2nd Workshop on Web Caching*, 1997.

[69] Duane Wessels and K Claffy. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, 1998.

[70] Roland P. Wooster and Marc Abrams. Proxy caching that estimates page load delays. In *Selected papers from the sixth international conference on World Wide Web*, pages 977–986. Elsevier Science Publishers Ltd., 1997.

[71] Zhi-Li Zhang, Yuewei Wang, David H.C. Du, and Dongli Su. Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE/ACM Transactions on Networking*, 8(4):429–442, 2000.

# Appendix A Overlay networks

Overlay networks were originally used as a means to increase scalability of distribution systems, in particular for streaming services, through the implementation of application level multicast. However, the low cost and ease of deployment of overlay networks lead them to be adapted for other purposes as well. The general trend in overlay networks is moving away from statically configured solutions aimed at solving one particular networking issue or addressing the needs of one specific application. New overlay networks have automatic adaption as a central design principle, while at the same time aiming at supporting as large an amount of different applications as possible.

Early overlay networks were used as tools for building multicast trees, and were often statically configured by design. More dynamic solutions, like Overcast [32], still have limitations that make them unsuitable for other applications than multicast of streaming data. Overcast is intended for performing streaming based on bandwidth optimization, and is therefore not suitable for distribution infrastructures that handle more complex workloads.

An early optimization of overlay networks, both dynamic and statically configured, is the introduction of topology-awareness [50]. This means enabling the overlay network to take the physical placement of the node in the Internet into account when adding new nodes to the system or changing the system configuration.

The next step was measurement-based overlay networks. Common to most such overlay networks is the use of a network monitoring service which can detect changes in the underlying network. Based on the results of the monitoring, the overlay network will adapt itself to the changing network conditions to improve the service to the user. The first such networks were designed to serve one specific purpose, and combined overlay networks with other techniques, forming very specialized solutions. One such overlay network, OverQoS [61], aims to improve the QoS[3] of the network by bundling traffic together. This enables the application to introduce priorities between the different streams it is sending, giving one or a few streams a better service than others. Another important overlay network is Resilient Overlay Networks [8], a system for improving fault recovery. This overlay network is designed to respond to network problems faster than normal IP routing techniques, allowing traffic to be sent around problem areas. In the past few years trends have turned towards making more general purpose overlay networks which does not support one specific application, but rather aim to function as a middleware layer forming a general purpose overlay. These generic overlay networks offer overlay routing and network optimization for a number of different network properties. One such service forms an overlay routing layer [40] which more specialized overlay networks can be built on top of. An even

[3] Quality of Service (QoS) is an important part of any distributed system. QoS parameters can be found in communication protocols, operating systems, multimedia databases, file servers, and so on. QoS can also refer to those aspects directly affecting the human user. In the context of networks, QoS parameters typically refer to such items as error rate, delay, jitter, and so on. See FFI-Rapport 2006/03859, FFI-Notat 2006/02580, and FFI-Rapport 2012/02494 for discussions of QoS related to Network Based Defence.

more generic solution is given in the Saxons [56] overlay network, which is a common overlay structure management layer designed to assist in the construction of large scale wide area Internet services. It dynamically maintains a high quality structure with low overlay latency, low hop count distance, and high overlay bandwidth. However, overlay services which support both discrete and continuous media data may need a service which optimizes for several different metrics at the same time. The core idea in [20] is a self-organizing overlay network. This overlay network optimizes for several different QoS parameters at the same time by combining already established techniques like topology-awareness and network monitoring with an automated route maintenance mechanism that ensures efficient usage of Internet resources. This idea is intriguing, but so far it has not been implemented (the reason probably being that QoS is less of an issue in today's Internet, where overprovisioning of resources can alleviate most problems). It would be worth pursuing a solution along those lines for NBD, where it could form the basis for a QoS-enabled application-independent overlay network.

As a final remark it is worth mentioning P2P networks, which are typically not general overlay networks (e.g., they are seldom used when QoS is needed, or for streaming or service deployment) but built for the special purpose of efficient decentralized file distribution. For further discussions on P2P networks, their applicability to certain tasks beyond file transfer (e.g., service discovery), and a discussion regarding their suitability for use in NBD, please see [57].

# Appendix B Introduction to Bloom filters

This appendix is a revised version of Appendix B in [57]. It is reused here for the convenience of the reader since it provides a brief introduction to the concept of Bloom filters, which are central to many applications – caching being the focus of this report.

A Bloom filter is a hash-based data structure that provides a membership function with a certain probability of false positives, never false negatives. Bloom filters were first described by Bloom in [11].

More specifically, a Bloom filter comprises an array of bits and a number of independent hash functions. When a data element is inserted into the filter, the hash functions are used to calculate a set of hash values representing the data element. For each hash value, the corresponding bit is set in the array.

To check whether a Bloom filter contains a specific data element, the process is the same, except that the generated hash values are compared to the existing array instead of being stored. If all the bits corresponding to the different hash values are true, then the data element can be determined to have been stored in the Bloom filter with a given probability.

A small Bloom filter example is shown in Figure B.1.

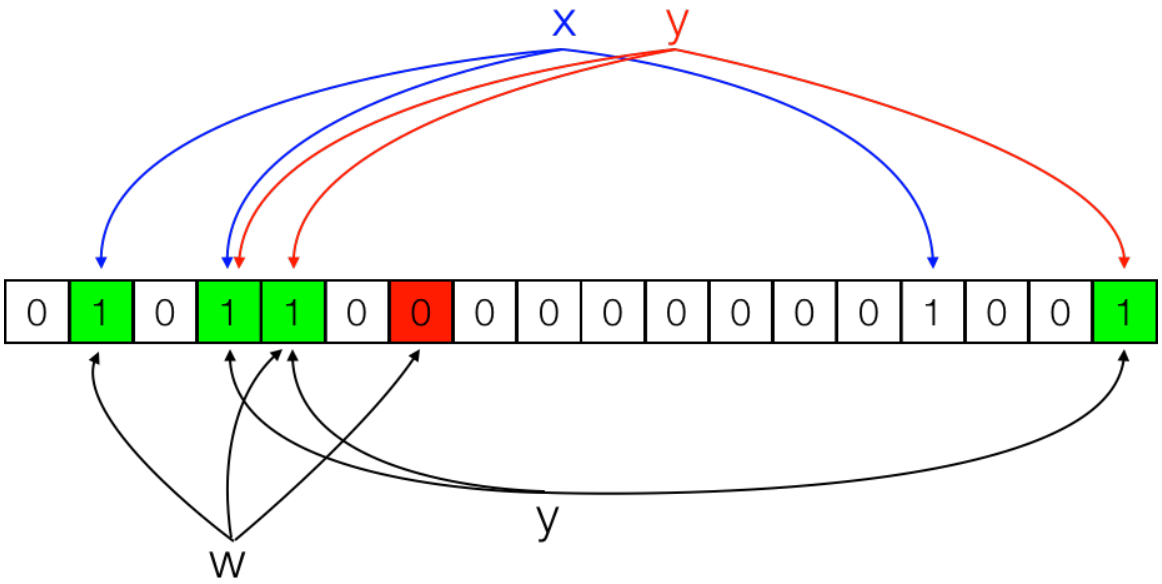


*Figure B.1 A Bloom filter where data elements $x$ and $y$ are inserted into the bit array — one bit is set for each hash function. Elements $w$ and $y$ are checked against the filter — $y$ is present with a certain probability, $w$ is decidedly not since one of the bits is false.*

The probability of a false positive in a Bloom filter is the same as the probability of all the bits for a data element already being set.

We can calculate the probability $p$ that one specific bit is not set by a specific hash function in a bit array of size $m$ with:

$$p = 1 - \frac{1}{m}$$

Further, we can calculate the probability $p$ of a specific bit not being set by $k$ hash functions after inserting $n$ elements with:

$$p = \left(1 - \frac{1}{m}\right)^{kn}$$

Therefore, the probability $p$ that $k$ hash functions set $k$ specific bits to true after inserting $n$ data elements can be calculated with:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-k\frac{n}{m}}\right)^{k} \tag{B.1}$$

Equation B.1 gives us the probability that the bits corresponding to the hash values of a new data element are already set to true — or in other words, the probability of false positives.

Ideally, $p$ should be kept as small as possible, but it can also be desirable to keep $k$ or $m$ small, to save computing resources or storage capacity — or in our case, bandwidth. Intuitively, we can see that $p$ will increase when $n$ increases, and decrease when $m$ increases. A high $m/n$ will give a smaller probability of false positives.

The optimal number of hash functions $k$ for a number of given data elements $n$ in a bit array of size $m$ can be determined by:

$$k = \frac{m}{n} \ln 2 \tag{B.2}$$

[58] propose using a stream of short Bloom filters to gradually reduce the false positive probability until the sets are equal. This approach has the advantage of requiring less computational complexity compared to error correcting codes at the expense of some additional bandwidth. Also, the approach is simple to implement and is therefore an interesting alternative to more complicated approaches. As the mechanism requires little state information to be transferred, any node may reply to a synchronization request after receiving only a single message. This enables fast synchronization in dynamic networks, as well as simultaneous synchronization of multiple nodes that are within radio broadcast range. These properties are desirable for many applications in mobile environments, for example in mobile tactical networks. Thus, attempts at set reconciliation in such environments could benefit from attempting this approach.