Security & Scalability of Content-Centric Networking

Sicherheit und Skalierbarkeit von Content-Centric Networking

Master-Thesis von Tobias Lauinger aus Schwetzingen

September 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Peer-to-Peer Netzwerke

Security & Scalability of Content-Centric Networking
Sicherheit und Skalierbarkeit von Content-Centric Networking
Sécurité et extensibilité de Content-Centric Networking

Master's thesis by Tobias Lauinger from Schwetzingen, Germany

Submitted in September 2010 to TU Darmstadt and Eurécom/Télécom ParisTech

Filière (Eurécom): Sécurité des systèmes de communication
Studiengang (TU Darmstadt): Master of Science Informatik

Confidentiality: No

Advisors:
Prof. Ernst Biersack (Eurécom)
Nikolaos Laoutaris (Telefónica)
Pablo Rodriguez (Telefónica)
Prof. Thorsten Strufe (TU Darmstadt)

Work done during an internship at Telefónica Investigación y Desarrollo, Barcelona, Spain. Thesis jointly supervised by Technische Universität Darmstadt, Darmstadt, Germany and Eurécom, Sophia-Antipolis, France as part of a Double Master agreement between TU Darmstadt and Télécom ParisTech, Paris, France.

Please cite this document as:
Tobias Lauinger: "Security & Scalability of Content-Centric Networking," Master's Thesis, TU Darmstadt, Darmstadt, Germany and Eurécom, Sophia-Antipolis, France, September 2010.

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 19. September 2010

(T. Lauinger)

Abstract

By suggesting radical changes to the current Internet, approaches to clean-slate architectures run the risk of introducing new opportunities for attacks. These attacks can range from new forms of denial-of-service to attacks against other users' privacy. In this thesis, we analyse the architecture proposed by *Content-Centric Networking* from a security perspective.

One security-critical feature of Content-Centric Networking is the introduction of general-purpose caches that are shared by a small number of users. We show how attackers can leverage these caches to monitor what content its users are retrieving. More generally, we argue that there is a tradeoff between network efficiency and user privacy. Countermeasures against cache-based privacy attacks need to carefully explore this tradeoff.

Résumé

En changeant Internet d'une manière radicale, une refonte de son architecture court le risque d'introduire de nouvelles possibilités d'attaques. Entre autres, ces attaques peuvent être des attaques par déni de service ou des atteintes a la vie privée des utilisateurs. Cette thèse analyse l'architecture de *Content-Centric Networking* sous l'angle de la sécurité.

Une particularité critique vis-à-vis de la sécurité de Content-Centric Networking est l'introduction de tampons universels qui sont partagés par de petits groupes d'utilisateurs. Des attaquants peuvent exploiter ces tampons pour surveiller les contenus auxquels les utilisateurs accèdent. Les mesures contre ce type d'attaques passent par un équilibre entre l'efficacité du réseau et la vie privée des utilisateurs.

Zusammenfassung

Wegen der grundlegenden Veränderung der Architektur laufen Ansätze zur kompletten Neuentwicklung des Internet Gefahr, neue Arten von Angriffen zu ermöglichen. Solche Angriffe können von Denial-of-Service bis zu Angriffen auf die Privatsphäre der Nutzer reichen. Diese Masterarbeit analysiert die Architektur von *Content-Centric Networking* unter dem Aspekt der Netzwerksicherheit.

Eine sicherheitskritische Eigenschaft von Content-Centric Networking ist die Einführung von Allzweckcaches für kleine Gruppen von Benutzern. Angreifer können diese Zwischenspeicher ausnutzen um zu überwachen, auf welche Inhalte die Benutzer zugreifen. Daraus folgt, dass die Effizienz des Netzwerks und die Privatheit der Nutzer zueinander in einer Austauschbeziehung stehen. Dementsprechend müssen Maßnahmen gegen diesen Angriffstyp beide Faktoren gegeneinander abwägen.

Contents

List of Figures

List of Tables

1 Introduction

When the Internet was originally built, the main concern of its designers was to connect and share expensive resources such as mainframe computers [1]. Since then, the Internet has evolved from an academic research network to a global infrastructure that is used for business and entertainment. In terms of bytes, the major use of the Internet today is content retrieval [2, 3]. Although truly end-to-end protocols such as VoIP, SSH and chat only make up for a tiny fraction of the traffic, the Internet's mechanisms and primitives are still optimised for addressing end systems at fixed locations instead of catering for location-independent content.

The scientific world is divided over how to handle this situation [4]: One side defends an evolutionary approach and argues that the current Internet should be improved in an incremental and backwards-compatible way. The other side favours a clean-slate approach and advocates a hard switch to a next-generation, unconstrained Internet architecture.

Among the clean-slate approaches, Content-Centric Networking (CCN) [5] is one of the most recent and most vividly discussed proposals. CCN proposes to focus the network's main mechanisms on content names instead of content locations, to follow a receiver-based communication model, and to introduce generalised caching in potentially every network device. Section 2 of this thesis gives an introduction to CCN, reviews its current state of research, and identifies future research directions related to CCN's scalability and security.

Changing the architecture of the Internet involves the risk of introducing new opportunities for attacks. For instance, CCN routers need to keep per-communication state that can be abused for denial-of-service attacks. Caches contain potentially sensitive communication traces that can be extracted by attackers, thereby endangering users' privacy. In Section 3, we investigate the architectural features of CCN with respect to their effect on network security. We illustrate our findings with attacks that exploit these novel characteristics.

In the main part of this thesis, we focus on how attackers can exploit network-level caches that are located close to the users. CCN breaks down any type of communication into independent, named content objects that can be cached. Even though these content objects might be encrypted, they still leak information through side-channels such as the content name, timing, and their size. Because only few users share such a cache, the amount of personal information revealed by communication traces can be very high. To substantiate this threat, we explain in Section 4 how attackers can list the contents of a cache, monitor requests for a particular content object, and duplicate the conversation that two clients are having via a cache. More generally, we argue that countermeasures against this privacy threat need to carefully trade off communication privacy against network performance. While we consider these attacks in the context of CCN, we believe that this problem applies to all network architectures in which general-purpose caches are shared by small groups of users.

The feasibility of these cache-based attacks depends on how much attack traffic is required to carry them out, and on how precise the conclusions are that the attacker can draw. In Section 5, we evaluate the attack in a scenario with one hundred users who are connected to the same DSLAM, and who are concurrently downloading and watching Youtube-like videos. We find that the attack traffic required to monitor requests for one video is so low that the attack could be extended to monitoring a large set of videos at the same time.

The overall goal of this work is to identify how basic building blocks and concepts impact security and scalability of future Internet architectures, using CCN as a case study. Our hope is that the lessons learned in this work will be useful to improve the security of current and future architectural proposals. To this end, the contributions of this thesis can be summarised as follows:

- We identify several promising research problems in the context of CCN's security and scalability.
- We analyse the impact of architectural features and design decisions on network security by describing a range of attacks that exploit these features. In particular, we argue that more powerful routers (in terms of per-communication state, computation, and cache) increase the attack surface of the network, and we show how caches can be misused as *storage*, for *denial-of-service* and for *privacy-related attacks*.
- For the privacy attacks, we develop algorithms that exploit the timing side channel to continuously monitor requests for objects, even if the exact characteristics of the cache and its environment are not previously known, and that require attack traffic of only 37.3 bit/s in our Youtube/DSLAM scenario.

2 Content-Centric Networking

This chapter gives an overview of the state of research in Content-Centric Networking (CCN). We first review the available literature on CCN before we discuss problems that have not yet been addressed in current research.

2.1 State-of-the-Art of CCN

Content-Centric Networking [5] is a new communication paradigm, together with a new network architecture, that has been designed to complement, and ultimately to replace the current Internet. More specifically, compared to the current TCP/IP architecture and communication model, CCN differs in the following ways:

- *Receiver-based communication model:* Receivers pull information by sending an Interest message. At most one data message is delivered in response to an Interest. Communication is *unreliable* and *soft-state*: The applications on the receiver side have to re-express interest for content if previous Interests have timed out.
- *Hierarchical content naming scheme:* CCN does not address hosts, but *location-independent* content objects. Content is given arbitrary, user-defined names organised in a hierarchy similar to URLs. Interests are matched with content, or with routes to content, by doing *longest-prefix matching*. Because of these properties, receivers can express interest in names that do not yet exist. These Interests will be routed to a content source capable of generating the corresponding content.
- *Cache-based architecture:* Every participant in the system, such as end nodes and routers, may cache content objects and use them to serve future requests.
- *Content security:* Every content message exchanged in CCN is digitally signed. In this way, the content publisher certifies the binding between the content and its name to ensure integrity and authenticity. Encryption can be used if confidentiality is required.
- *Stateful, more powerful routers:* Content routers in CCN need to keep per-interest state to avoid routing loops, and to send back data responses on the same path that the corresponding Interests took. Routers can verify the content objects' signatures to avoid content spoofing attacks. CCN also supports a limited query language for Interests that routers must implement.

When a CCN node receives an Interest, it first attempts to satisfy it with content from its local content store (cache). Typically, such a cache implements a Least Recently Used (LRU) or Least Frequently Used (LFU) replacement policy.

If no suitable content can be found, the Interest will be forwarded. The Forward Interest Base (FIB) is used to find the outgoing interface for forwarding, similar to a routing table. The FIB may contain routes to applications running on the same host, or to physical interfaces. CCN supports multiple connectivity with parallel routes to the same content prefixes.

Every outgoing Interest is added to the Pending Interest Table (PIT), to know on which interface to send back incoming replies, and to avoid forwarding identical Interests several times. This aggregation of Interests is used for efficiency, and to tolerate loops in the CCN topology.

A strategy layer is used to adapt CCN's parameters and behaviour to different transmission media or paradigms, such as fixed, wireless, or delay-tolerant networking.

The CCNx project [6], sponsored by PARC, develops an open-source implementation of CCN. This early prototype implements the basic functionality of a CCN node, including FIB, PIT, a main memory cache, and forwarding using static routes. Content signing and signature verification, key management, optional encryption and decryption are also supported. The prototype also contains a persistent, disk-based content repository that can be used to serve files, and some example programs such as an implementation of Voice over CCN [7], a chat application, and several command line tools for data transfer.

2.2 Research Agenda for CCN

At the current state of research, the papers about CCN present general architectural principles, and the CCNx prototype defines APIs and local communication. Applications that would benefit heavily from CCN, large-scale evaluation of CCN and routing proposals are among the issues that are left for future work.

This section summarises open CCN research problems by discussing ideas proposed in the literature and new ideas that need further investigation. Many of these issues are interrelated. First, we discuss problems that are related to scaling CCN to widespread use; then we focus on security-related research issues.

2.2.1 Scalability Research Topics

CCN's routers need to keep state for every Interest that they forward, CCN nodes may forward their Interests to several nodes at once, and routing happens on content names instead of location. All these differences to TCP/IP might have an influence on CCN's scalability and prompt for research. This section discusses the research problems of suitable applications, routing, and router design as well as some protocol details of CCN.

Applications

Deployment of CCN will require significant investments into equipment and education of networking staff and programmers. In addition to bandwidth savings, CCN is expected to enable a new range of applications that make use of CCN's flexible naming/addressing & discovery. Future research should identify these applications.

As to existing applications, it is assumed that content-oriented applications can work on top of and benefit from CCN. The Voice-over-CCN prototype [7] focusses on demonstrating that traditional, conversation-oriented protocols can also be ported to CCN. The assumption is that both users involved in a conversation are reachable under a globally unique prefix, such as `/domain/user`. Assume that Alice wants to call Bob. Then the following steps are carried out to establish a call:

1. A SIP Invite message is used to request a call. In VoCCN, Alice encodes it into the name she requests in an Interest to call Bob. The name can be encrypted for confidentiality: `/domain/sip/bob/encoded-invite-message`.
2. Bob sends his SIP response back in a data message with the requested name.
3. The RTP voice data stream is encapsulated in CCN data messages. Alice sends Interests to request voice data from Bob, and vice versa. Interests follow the scheme `/domain/user/call-id/rtp/sequence-number`.

While this scheme shows that end-to-end conversations are feasible, it also shows that pushing data is not an easy task in CCN: If data is pushed to a destination by encoding it into the name, the destination should not simply send a data message as a response to this Interest, because the data message would send

the whole data back as part of its name. Therefore, push-scenarios require both parties to be reachable under a globally unique name[1].

A second issue about such end-to-end protocols is that the messages exchanged between the participants will be cached, although it is unlikely that there will be any cache hit in the future. In this way, those messages use cache space that could be used better for other, reusable data. This raises the question if applications such as voice or remote login should be using CCN.

Versioning is another issue in this context. Assume a newspaper with its front page being reachable under `/newspaper`. Obviously, the front page will change from time to time. CCN allows content names to carry special version markers. However, clients cannot be assumed to know in advance what the latest version number is. Therefore, there needs to be an algorithm to find out the latest version number in presence of older copies that might still be cached.

On the CCNx mailing list [8], an algorithm was proposed that we list in a slightly modified version:

1. Request the content without version marker, or with the latest version that the client knows of.
2. Extract the version number from the response that has been received, increase it by one, and request the same content again.
3. Iterate until no content can be found: The highest available version number has been determined.

This solution seems unsatisfactory because it might be inefficient, particularly if the algorithm has to be run frequently. Furthermore, it is challenging to implement this algorithm in a way that provides a guarantee that the latest version has indeed been found: If the version number requested by the client is too high, the content source has to reply with a "content not available" message that has a very short lifetime so that replay attacks are unfeasible. In absence of this reply, it is not possible for the client to distinguish this case from a simple timeout.

Broadcast or multicast protocols, such as those used for IPTV, need more investigation about how they could be implemented over CCN. By default, a CCN node may accept a data response only if a previous Interest has been sent. A CCN node connected to a broadcast link could overhear Interests sent by other nodes and suppress its own Interest to reduce the traffic on the link. However, this still means that an Interest must be sent for every data message transmitted to the multicast group, which increases the overhead compared to IP-based protocols.

Architecture

An important question relating to content routing, and to caching in general, is at which layer this functionality is best implemented. Running everything over CCN, i. e. replacing IP, has the advantage of providing a generalised caching infrastructure. On the other hand, some applications might not benefit from caching, and end-to-end communication, as discussed above, might be less efficient than with IP.

Alternatively, CCN could run as an overlay on top of IP, with only selected applications making use of CCN. This would raise the question of how much more efficient CCN would be as compared to traditional content distribution networks, and redundancy elimination techniques [9, 10].

Routing

In their CoNext paper [5], Jacobson et al. show as an example how CCN can be mapped on existing routing protocols: Intra-domain routing with link-state protocols uses the prefixes of local resources (such as persistent content repositories) and neighbour adjacencies to build a topology. For inter-domain routing,

[1] Pushing data in Interests that are never followed by a data response might confuse attack detection heuristics as the one described in Section 3.3.2

the authors suggest a scheme to establish tunnels between CCN-capable domains in an incremental deployment scenario of CCN. The gateway of one domain would look up the first component of the requested CCN name as a DNS name. The result would be the IP address of a content router in the other domain, with which an UDP tunnel can be established. For a full CCN deployment, the authors propose to integrate CCN prefixes into BGP announcements. However, it is not clear how big routing tables would grow, and how much "content source mobility" would be supported by this scheme.

The routing approach summarised above is quite conservative in that it closely matches routing on the current Internet. CCN, however, is more flexible because it tolerates loops, and it supports multi-path routing, forwarding at most one data response back to the requestor. Many different routing protocols for CCN are conceivable, and their choice depends mainly on what the system should be capable of: Support content source mobility, enable disconnected operation, or look for cached content everywhere in the system as opposed to only following the (default) path to the content source.

One approach to routing could argue that popular content is likely to be found in nearby router caches, thus flooding-like routing protocols would be appropriate. However, in this approach it would be difficult to locate unpopular content, and to locate content sources for initial seeding.

More traditional approaches could oppose that main-memory-based router caches are likely to be relatively small compared to the total content size, thus they would not be a reliable source for content, but rather a dynamic and possibly short-term storage. Therefore, it would require less effort to find content at a relatively stable location, such as the original content producer.

To save transit bandwidth, ISPs could build large server farms with a lot of disk storage to serve cached content according to their caching policies. Because this would happen entirely inside their domain, ISPs could configure this system as they wish and thereby force their customers to use it. However, disk-based caches might induce a higher latency as compared to accessing the content directly at a more distant content source that has the data in main memory.

CDNs are unlikely to become obsolete because content providers wishing to reduce latency may need to deploy their own application logic close to their customers. CCN's caching cannot provide custom application logic and dynamic content creation. However, CCN's content name routing could be configured to accommodate CDNs much more transparently than the current Internet.

Caching

Caches are a central element of CCN's architecture. CCN might not only carry static content that can be shared between users, but also dynamic, individual content that is unlikely (or impossible) to be shared. Given these circumstances, we identified the following research questions with respect to caching:

- The topology of the caches, i. e. do caches cooperate, and how are they organised (hierarchy, mesh).
- The size of each cache as a function of the line speed, or the position in the hierarchy.
- The replacement policy used by the cache.
- How to assess the utility of data in the cache, i.e., what to cache and what not. For instance, dynamic content might not be cached.
- How the caches should interact with the routing algorithm: What content is announced to external parties, and shall caches cooperate beyond domains/ISPs?
- CCN splits large files into chunks—should caches be aware of this when making insertion and eviction decisions?

Network and Transport Protocol Issues

CCN's primitives work at the network level to provide unreliable content routing. Research is required to define policies for Interest forwarding that enable priorities for certain kinds of traffic, if desired, and that ensure fair sharing of network resources between all participants.

To date, CCN lacks a transport protocol that ensures reliability and fairness. Some of the CCNx tools pipeline Interests to achieve a higher throughput, similar to TCP, but a protocol still needs to be formally defined.

Interests time out if no reply has been received in a given time window. It is not clear how these timeouts are to be chosen, and which entities have to re-send an Interest that timed out: If intermediate routers retry, the protocol might be more efficient than if the original requestor has to reissue the Interest. However, this would also make it more difficult for the original requestor to set a timeout for the whole process.

Router Design

CCN requires more powerful routers than IP: CCN routers need to verify signatures, process Interests with complex query features, and keep state for every pending Interest. Routers might also have to run algorithms to detect and prevent denial-of-service attacks. And most importantly, CCN routers have to work at speeds comparable to today's IP routers.

To date, it is not sure how such routers can be built, and how much they would cost. An intermediate question towards solving this goal would be to determine the processing power and memory requirements for CCN routers as a function of the line speed. In particular, how do Interest aggregation and cache hits affect the amount of memory needed for the PIT? This question is of particular concern because state in CCN routers is proportional to the number of Interests forwarded. In addition to potentially impeding scalability, state in routers also poses security-related challenges because it could be abused for denial-of-service attacks, as we will discuss in Section 3.4.3.

Furthermore, CCN contains cryptographic algorithms that might have to be replaced if compromised or approaching the end of their lifetime. This raises the question of how to implement them, in hardware or in software [11].

Depending on the targeted size of caches, CCN routers might need considerable amounts of fast random-access memory. It might even be necessary to back main memory with a large, but slower disk. This raises the question of how disks can be integrated into the architecture without increasing the total request latency (compared to fetching the data from the content source that is distant, but has the data available in main memory).

Energy Efficiency

In [12], Lee, Rimac and Hilt compare the energy consumption of content distribution networks, nano data centres and CCN architectures. Their main finding is that a CCN architecture reduces the number of hops, thus reduces the energy consumption. However, the authors use a simplified system model in which routers do not consume any additional power due to increased computation requirements. Furthermore, the authors assume caches of considerable size, and assume that requested data is always cached.

We believe that these assumptions are oversimplified. CCN routers need additional computation power to verify signatures and to process special query features in Interests. Furthermore, caches might be quite limited in size if implemented with main memory to avoid the latency increase imposed by harddisk-based caches. Different routing protocols might also have an influence on energy efficiency. We believe that all these reasons motivate a more differentiated investigation into CCN's energy efficiency.

Statistics Infrastructure

Content providers are interested in statistics about how their content is being accessed, and by which user population. This information is essential to determine pricing of advertisements, and to optimise the service delivered to users. Because CCN does not provide endpoint identifiers in Interests, because multiple Interests can be aggregated, and because content can be served from caches instead of the content source, it is very difficult for content providers to establish such statistics.

A solution could consist in using a digital rights management system for the content, which would require users to interact with the content source to retrieve a decryption key that permits to decrypt the requested content. The content, in turn, could be served from caches. We elaborate on this below in Section 2.2.2.

If the content source needs geographical information about its users, it could require users to register and to provide this information before accessing content. However, the content provider would need to verify this data through a secondary channel. Especially for geographically restricted content, such as sports events, this procedure might not be deemed secure or efficient enough. In this case, a CCN implementation could force users to sign their Interests with a key that contains geographical information. The correctness of this information could be certified by the users' ISP. This approach is discussed below in Section 2.2.2.

2.2.2 Security-Related Research Topics

The change of architecture and communication paradigm imposed by CCN also changes many aspects about network security. This section focuses on unsolved problems from a high-level systems and design perspective, whereas attacks will be discussed in greater detail in Chapter 3.

Use of Cryptography

The extensive use of cryptographic techniques is at the heart of CCN: Because content is not bound to a location, it must be authenticated and integrity-protected to avoid basic spoofing attacks. Furthermore, access control in CCN is provided by encryption. That is, an attacker can rather easily obtain an encrypted version of the content that he is interested in. CCN's security is based on the fact that the attacker cannot make any conclusions about the plaintext of the content, and that he cannot exploit side channels.

However, even encrypted data tends to leak some information, such as the size of the content, the time when it was requested, and the name of the content. We will discuss these issues in more detail in Chapter 4. Another issue with the use of cryptography stems from the fact that cryptographic algorithms have a limited lifetime, which is generally assumed to be around 30 years. After that time, available computing power will have increased to such an extent that brute force attacks on encrypted content become practical. This means that an attacker with access to encrypted data that will still be valuable 30 years after its creation might simply keep the data for later analysis. We will discuss in Section 3.4.4 how attackers can clone the conversations of other participants.

Furthermore, once content has been released, it is unclear how it could be revoked: Copies of that content can be cached in many places in the network. To prevent more users from accessing revoked content, a revocation list could be published and enforced by routers. Alternatively, content could be released with relatively short lifetimes, after which routers would not serve cached copies any more. However, this would mean a higher load on the content publisher because it would have to serve copies of its content more frequently.

The issue of revocation is particularly important in the context of keys that are used for content signatures. For instance, if a key is found to have been compromised, the key should be marked as invalid. This information has to be spread to all consumers of the compromised data, and maybe even to all routers, in a timely manner.

Finally, use of cryptography requires the exchange of key material, and particularly the use of a trust model (some suggestions can be found in [5, 13]). Policies need to be defined to prevent multiple parties from claiming the same name space.

Denial of Service

A recurring topic in networking security research have been mechanisms to make the Internet more denial-of-service proof. Because attackers, especially if they control a large number of machines, can simply craft IP packets and send them to the victim, no matter their location, this type of attack is relatively easy to carry out.

CCN has some properties that make denial-of-service attacks more difficult. Firstly, data is not routed; it always follows previously sent Interests, and on each interface, at most one data packet will be accepted for a previous Interest. Thus, if an attacker wants to drown a distant victim in data messages, he would have to make the victim generate Interests for an amount of data that is larger than what the victim can handle, which is a quite unrealistic scenario.

Interests, on the other hand, are routed. If the attacker chooses the prefix of the victim content source, and a different suffix for every Interest, responses from caches will be avoided and the resulting Interests will be routed to the victim. However, according to [5], a second property of CCN makes it possible to develop countermeasures: Data messages travel on the same path as Interests. Thus, routers can observe the fraction of Interests per prefix that never resulted in transmission of a data reply. If this fraction is too high, an attack could be detected and the Interest could be dropped.

This algorithm would have to be deployed in intermediate routers. An efficient implementation of this algorithm and optimal attack detection thresholds have still to be researched. Knowing how the detection algorithms work, an attacker could simply choose an attack rate that lies below the threshold to avoid detection, but that is still high enough to achieve the attacker's goals. This stems from the fact that intermediate routers have no knowledge about application semantics: Only the final destination can define whether it is under attack. To this end, Jacobson et al. suggest using a mechanism in which "the attacked domain can ask downstream routers to throttle the number of Interests they forward by name prefix".

A target for denial-of-service attacks that has not yet been discussed are the intermediate routers. Because they offer more powerful functionality than IP routers, and because they have to keep per-Interest state, CCN routers have an increased attack surface and might become victims of denial-of-service attacks themselves. We discuss such attacks in Chapter 3.

Privacy

CCN messages are self-contained, named objects. As such, they carry more semantic information than an IP packet taken from an opaque TCP stream. Furthermore, these objects are (temporarily) kept in caches without any access restriction, which means that communication traces linger around all over the network.

Attackers can determine if a given data object is cached at a node by measuring the response time when requesting that item. Furthermore, even encrypted data leaks information through meta-data such as its name or the publisher's public key, and through side channels such as the content length, and the request time. This type of attack, called cache snooping, will be discussed in Chapter 4, where we also outline countermeasures.

Another threat to user privacy are malicious ISPs. ISPs wanting to profile their customers can easily do so by monitoring the access routers and by logging which customer requests which names. It is not clear how this could technically be prevented without disabling caching: Assume a hypothetical cryptographic or obfuscation construct that can map a clear-text name component c to n obfuscated names $o_1, ..., o_n$.

Let this construct be one-way, that is, these n names can be inferred from the clear-text name, but the opposite is not true. If a malicious ISP sees such a name, it cannot tell directly which real-world content source this name belongs to, thus one might claim that some level of "privacy" has been achieved.

In order to allow forwarding of such obfuscated Interests, and to allow cache hits, there must be a function that can detect a match between two obfuscated names for the same object, o_i and o_j. However, if such a function exists, a malicious ISP can build a dictionary of obfuscated names for the content sources of which the use is to be monitored, and match incoming Interests with the dictionary. Even worse, because a router always sees the data response to an Interest, the malicious ISP could build a dictionary of content hashes to fingerprint the data payload. Hence, the privacy gain would be minimal. On the other hand, if such a function does not exist, i. e. if names and payload are encrypted individually for every user, the privacy gain would be maximal. Yet, this would imply that caching is operational on a per-user basis only, which considerably limits its effectiveness.

Given these thoughts, the research question is if a mechanism can be found that makes it *impractical* for a malicious ISP to build a dictionary of obfuscated names, but that still allows (prefix) matching for forwarding and caching. Furthermore, one could investigate whether a tradeoff can be found between user privacy and network efficiency. More generally, research is needed into what privacy needs users have: Do they trust their ISP or do they need technically enforced guarantees? Do they care less about access to popular content being profiled, and more about unpopular (but more personal) content?

Finally, a point could be made that logging of user access to content should not be prevented at all, but should be made mandatory for law enforcement purposes, as we discuss below.

Accountability Infrastructure

Under TCP/IP, if a host is under attack, it has some approximate knowledge about where the attack comes from. Sender IP addresses can often be faked, but if the attacker needs to receive answer packets, the address must have at least some truth to it.

CCN, in contrast, does not have endpoint identifiers; only the previous hop is known when a message is received. Data responses follow back the path left by the Interests in the PITs, but this trail disappears as soon as a data message has been forwarded. Consequently, CCN content sources that are under attack cannot trace back where the attack came from, and they cannot selectively block that origin. They could disable the link on which the troublesome packet was received, but this would potentially disconnect a large number of innocent users.

The reason for this problem is that CCN abolishes endpoint identifiers and true end-to-end connection semantics, but emulates something similar: Individual services, such as dynamically generated content, still require access to a specific, physical machine. While users (and attackers) can route Interests to this machine, it is not possible for the machine to identify the other end of the "connection" any more. The new communication paradigm is more flexible in cases of intermittent connectivity or mobility of the requestor, but this comes at the cost of the new problems discussed above.

Law enforcement agencies could monitor access routers or DSLAMs, because at this location the previous hop leads to the customer. However, it might not be an easy task to correlate an attack observed at the victim site with information gathered on the network edge. Especially, this only works if all access providers worldwide are monitored. The other way around, if a malicious Interest is observed at an access router, without a complaint from a "victim", it might be difficult to proof where the Interest was routed because the state of the network (routing tables, cache contents) can quickly change.

A potential solution would be to force users to digitally sign their Interests, or to automatically sign Interests at the access router. The signing key could be provided by the ISP, and could be changed on a regular basis, such as every day, to provide privacy comparable to today's situation with dynamic IP addresses. The signing keys could contain information such as contact information of the ISP, and the name of the region or country. This solution would allow content sources to selectively block a specific, but still anonymous user, or to contact the user's ISP in case of severe problems to reveal the user's identity

and commence prosecution. However, aggregation of signed Interests in routers is much more difficult, if possible at all.

Protecting Content

Content providers are interested in making their content available under controlled circumstances. Sponsored content, for instance, shall be displayed only on their own page with advertisements, and third parties shall be prevented from deep-linking such content, i. e. from using it on other pages without the original authors' consent. Providers of subscription content need to make sure that the content is viewed only during the subscription period, and only on authorised devices with authorised software.

On today's WWW, content providers serve the content from a machine that they control, thus they can enforce a set of conditions before they admit a user's request for content. For instance, for free content they can check that the user is requesting the content from a page hosted on the same server, or that the user is logged on.

Subscription content, such as on iTunes, can be reproduced only with authorised software that enforces the access restrictions imposed by the content provider. After payment, the software downloads the content, which is encrypted with a single master key. The software also obtains the master key to decrypt the content, but this key is hidden from the user. Using a single encryption key per file instead of individually encrypting the file per user reduces the computational load of the servers, and facilitates the use of CDNs.

The security of this scheme is based on the assumption that only the authorised software is used. If attackers manage to understand the encryption scheme and extract the master decryption key, they can circumvent any restrictions imposed on the content. If such cases become public, content providers usually change their protection mechanisms and force all legitimate users to upgrade to the newest version of their client software by refusing connections from earlier versions.

With CCN, cached content is retrievable without restrictions. For subscription content, this means that once a protection scheme has been compromised, all content that is cached somewhere is not protected any more. Thus, it is important for CCN to have a solid revocation scheme for cached content as discussed in the beginning of this section[2]. Furthermore, even the semi-free (sponsored) content needs to be protected with such a digital rights management system to enforce the relatively loose rules on the playback environment. CCN's caching means less data traffic for content providers, but at the same time they lose direct communication with the users, thus the providers also lose some degree of control.

2.3 Conclusion

The CCN papers define the principles of a new networking architecture and the CCNx project develops a prototype implementation of CCN. Many details need to be investigated:

- Which applications should use CCN and at which layer should CCN operate;
- what capabilities should CCN's routing support;
- how should the caching infrastructure be organised;
- how can scalable CCN routers be built, and
- how can content providers control usage of their content and gain statistics about its use?
- CCN makes denial-of-service attacks more difficult, but at the same time the enhanced architecture provides new attack opportunities.

[2] The alternative of individually encrypting the content for each user would remove all benefit from caching and would be more costly than the solutions used today.

- The widespread use of caches can be exploited by attackers to extract privacy-relevant information about other users.

In the following chapter, we continue our study of CCN with a systematic security analysis.

3 Security of CCN

In this chapter, we undertake a systematic security analysis of CCN. We begin by discussing a simplified system model in Section 3.1, and continue with a threat analysis related to this model (Section 3.2). Section 3.3 reviews denial-of-service attacks that have already been reported in the CCN literature, whereas Section 3.4 introduces new attacks with a focus on denial-of-service and cache-related attacks.

3.1 System Model

Our system model encompasses the main participants and components in a potential CCN deployment. Because there is no fully functional and deployed implementation of CCN yet, we focus our analysis on architectural and conceptional weaknesses in the novel aspects of CCN's design. Implementation weaknesses, such as buffer overflows, incomplete verification of parameters, or flawed use of cryptography, are considered out of scope. The goal of this analysis is to predict system-inherent challenges with respect to securing CCN, so that these issues can be resolved before a large-scale deployment is started.

Participants:

- *End users* request content by generating Interests.
- *Content sources* produce content by generating data in response to Interests.
- *ISPs* connect end users and sources by forwarding Interests and data.

Components:

- *End user equipment* might include home routers and a set of personal computers; for simplicity, we assume that every customer of the ISP corresponds to one single device.
- *Routers* forward Interests and forward back the corresponding data. They are generally assumed to have caches built in. These caches can differ in size and implementation technology. In the following, we assume that caches are implemented in fast random access memory. Consequently, their size is in the order of tens of gigabytes, or less.
- *Content sources*, or content generators, have available or generate upon request the content that is solicited in incoming Interests. While in reality large server farms and even geographically distributed CDNs might be used to serve content, we simplify our model by assuming that one single machine is serving all requests.
- *Links* connect all the components listed above. Real-world links differ in technology and speed, but we do not distinguish this in our model.

We envision a CCN deployment as an open system similar to the current Internet, where participants do not necessarily fully trust each other.

3.2 Threat Analysis

Any of the participants listed above can become an attacker. We use the term attacker in its broadest sense, i.e., we would consider an ISP to be an attacker if they spy on their users or otherwise reduce users'

privacy. Consequently, depending on its type, an attacker might be in control of any of the components listed above. In the same way, the targeted victim can be any of the participants.

Table 3.1 identifies several *types of attackers* that differ in their knowledge, budget, capabilities, and goals. The *risk* of a given attack depends on how difficult it is to exploit, i.e. which attackers are able to carry out the attack, how many potential attackers exist, and what investment these attackers would have to make.

Type	Knowledge	Budget	Capabilities	Motivation
Simple user	little (use attack tools)	small	own connection	reputation, revenge, boredom
Professional attacker	medium to high	medium	botnet, hacked routers	money (paid attacks)
Malicious ISP, Malicious content src.	insider	high	access own equipment	money, law & regulations, competitive advantage
Cyber warrior	up to insider	very high	botnet, hacked routers	retaliation, deterrence, influence of public opinion

Table 3.1: Types of attackers.

3.2.1 Architectural Risks & Comparison to TCP/IP

CCN has a different architecture than TCP/IP. The most notable difference is that CCN enables systematic caching of all data in routers. **Caching, however, means that there is a tradeoff between efficiency and privacy:** Users leave communication traces in caches. These traces can be retrieved by anyone, either by using Interests with special query features, or by probing caches and using the timing of replies to find out whether an item was cached. In particular, it might be possible to extract the whole sequence of packets that a third party exchanged in a conversation. This fact considerably simplifies attacks, because the attacker does not need the capability of sniffing the victim's link in real-time any more as it would be the case with TCP/IP. The attacker only needs to be connected to a common cache, and the attack might even be carried out retroactively. Furthermore, caching requires encryption for data confidentiality. Cryptography, however, has a limited lifetime that is considered to be around 30 years. Thus, an attacker might collect data and keep it for the future when the encryption algorithms cannot withstand brute force attacks any more.

CCN's communication model is purely content-oriented. CCN addresses content instead of location, thus **users trust content and its original author, not its current origin** (which could be an arbitrary, potentially untrusted cache). Consequently, it is necessary to protect the content's integrity as well as the binding to its name and author.

Because TCP/IP enforces direct communication with a (trusted) content source, being up-to-date is not an issue in the Internet. In contrast, CCN requires to **prevent replay of old content**, and to have a secure means to gain assurance that some given content is the latest version released by its author.

Denial-of-service attacks against content sources are more difficult to carry out in CCN, but they **are still feasible:** Because only Interests are routed, only Interests can be used to overwhelm a content source. (Data always follows back the path that the Interest took.) Aggregation of Interests and caching implies that attackers must generate Interests that will be routed to the same content source, but that request different names. In contrast, an attacker on the Internet can generate an arbitrary amount of identical IP packets that will all be routed to the target. Detection algorithms in CCN routers can leverage the fact that requests and replies always take the same path. Nevertheless, they are complicated by the fact that Interests do not carry an identifier that would enable identification of their origin.

Location identifiers are available in TCP/IP and can be used to defend against attacks or to implement end-to-end protocols. Communication in CCN is "one-hop" because of Interest aggregation and lack of endpoint addressing. Without additional mechanisms, **a CCN node cannot identify the originator of a**

query, as it would be required to trace back an attack to its origin, for instance. Conversation-oriented protocols that have end-to-end semantics will need to implement an explicit way to make this information available. The risk is that such an "optional" extra layer might not be adopted by everyone, and could therefore be evaded by attackers to achieve a high level of anonymity.

CCN routers are more powerful, both in state and computation power, than IP routers. This is largely an implementation choice: A loop-free topology and introduction of source routing backwards to the Interest issuer would make the PIT redundant. This choice has consequences on the system's security, as **more powerful devices inside the network increase its attack surface** and could therefore lead to Denial-of-Service attacks, for instance.

3.2.2 Attack Tree for Denial-of-Service Attacks

Figure 3.1 on page 21 shows an attack tree for denial-of-service attacks in CCN. An attack tree shows different ways of achieving the goal of the attack. The leaves of the tree represent basic steps of the attack; inner nodes contain intermediate subgoals. The root of the tree stands for the ultimate goal of the attack. In this example, all subtrees are alternatives, that is, a path from a leaf to the root explains one potential attack instance.

One way to deny service to clients is to *make content unreachable for requests*:

- A *source* can be disrupted by flooding it; to achieve this an attacker can send a large number of Interests to the source (Sections 3.3.2 and 3.3.3), or he can decrease the efficiency of caching so that the regular traffic is entirely forwarded to the source and overloads it (Section 3.4.2).

- *Routing* can be disrupted by malicious (or compromised) routers that do not forward requests. We believe that economic forces make ISPs forward Interests as expected by customers. Preventing attackers from gaining administrative access to routers is considered out of scope in this discussion.

  Every router (PIT) has a timeout after which it deletes pending Interests and does not forward responses any more. There can be configurations of different timeout settings in different routers that prevent content from being retrieved; however, it is not clear how an attacker could enforce such a configuration.

- There are two types of *intermediate devices* that can be disrupted: *Links* can be flooded with a large number of artificial Interests so that the link reaches its capacity limit and legitimate traffic is queued in routers (Sections 3.3.2, 3.3.3 and 3.4.2). *Routers* can be slowed down by sending requests that require the router to carry out expensive computations (Section 3.3.1). Alternatively, the router's memory for state can be exhausted by overflowing the PIT (Section 3.4.3).

The other way to deny service to clients is to *serve fake responses:*

- Valid content can be *blocked by routers that believe that the content is invalid:* Attackers can replay (or generate) a "content does not exist" response.

- Content can be spoofed by generating fake responses that are not signed or signed with a wrong key, hoping that the client accepts the response. For instance, the client might not know the correct key of the content source. Alternatively, old content signed with the right key can be replayed, or the attacker might have gained access to the source's signing key and is now able to spoof content.

The latter attacks are possible if CCN's cryptographic protocols are not correctly implemented, or if clients have an unsafe configuration with respect to their use of cryptographic mechanisms. Because these risks are already present in current network architectures, we will not consider them further.

3.3 Known Attacks

In this section, we summarise some attacks related to denial-of-service, and their countermeasures, that have already been discussed in the CCN papers [5, 13], or on the CCNx mailing lists [14].

3.3.1 DoS by Forcing Expensive Computations

If routers verify content signatures, an attacker can request many data items that require the router to do expensive computations, which ultimately might negatively affect the service delivered to other users.

Assumptions: Content routers systematically verify the signatures of the content objects that they receive. Routers may cache keys, but they do not have every key readily available. The attacker controls one or more machines, and colludes with a malicious content source.

Attack: The attacker requests data from the malicious content source. Every data item is different and signed with a different key. In order to verify the signature, the router has to retrieve the key from the location indicated by the content source. To delay this operation, the content source can artificially slow down its response. Furthermore, knowing the cryptographic algorithms and their implementation that is being used, the attacker could choose the keying material and signatures such that they execute in the worst case time.

Impact: The router is busy verifying signatures, and may slow down service delivered to legitimate users. Alternatively, the router might stop verifying signatures to keep up with the demand.

Countermeasures:

- Stop verifying signatures when the load becomes too high. In any case, CCN receivers are expected to verify all signatures themselves.
- In order not to admit spoofed content into the cache, signatures should be verified. However, verification can be delayed until processing power is available. Alternatively, signatures could be verified only if the content has been cached for some time (i.e., avoid investing computing power for content that is immediately evicted from the cache).
- Detect content sources that use a suspiciously high number of different keys. Either do not verify these signatures, or retrieve the keys and verify the signatures only when sufficient spare processing power is available.

Risk: Low. The attack can be detected, requires a colluding content source and potentially considerable attack traffic (signature verification might be implemented in hardware to reach high speeds).

3.3.2 DoS Against Content Sources

Attackers can avoid cache hits and send Interests directly to the source with the goal of overwhelming the source and denying service to legitimate clients.

Assumptions: It is possible to build names of content that does not exist, and route Interests directly to the content source. The attacker is assumed to have access to one or more machines, such as a botnet.

Attack: The attacker constructs a large set of different names for content that does not exist. This guarantees that no cache hit can occur, and the Interests will be routed directly to the content source.

Impact: If many compromised machines carry out this attack at the same time, the content source can be flooded with Interests, so that legitimate clients are denied service.

Countermeasures:

- Routers can keep track of how many responses are received for the Interests that have been forwarded per prefix. If routers detect a suspiciously high fraction of Interests without response, they can slow down the rate of Interests forwarded for this prefix.

Risk: Low-medium. Detection is possible although it might be evaded; the attack requires access to a botnet and high attack traffic.

3.3.3 DoS with Special Bits

Some special Interest bits allow attackers to avoid cache hits. These can be abused to flood the content source with Interests and deny service to legitimate users.

Assumptions: CCN Interests can specify that the response may not be retrieved from a cache. The attacker is assumed to control one or several machines, such as a botnet.

Attack: The attacker sends a large number of Interests to a given content source, specifying that the responses may not be fetched from caches.

Impact: The source will be flooded with Interests. Legitimate users are denied service.

Countermeasures:

- Limit the use of this special Interest option. For instance, it might be permitted only for Interests with local scope (that are not forwarded beyond the local machine or network). Alternatively, only digitally signed Interests might be able to use this option. The hope is that the key certificate would reveal the identity of the attacker. However, the attacker could steal keys from unsuspecting users to carry out the attack under a false name.

Risk: Low. A perfect countermeasure is available; the attack requires access to a botnet and high attack traffic.

3.4 New Attacks

In this section, we introduce new attacks that have not yet been discussed in the context of CCN.

3.4.1 Keeping Unwanted Data Available in the Caches

By leveraging the widespread router caches, this attack allows an attacker to gain free and uncontrolled storage for illegal content, or to keep deleted content available in router caches.

Assumptions: CCN makes widespread use of caches, where the size of the caches might be substantial. The attacker is assumed to have access to one or several machines, such as a botnet, that are connected to a cache.

Attack: The attacker continuously issues Interests for the file that is to be kept in the cache. This artificially high number of requests creates a *false locality*, as described in [15, 16]: To the router, the popularity of the file looks higher than it actually is.

An attacker needs to issue an Interest shortly before the file would be evicted from the cache. In case of LRU caches, an item remains in the cache for at least the cache's *characteristic time* [17] t_c. This means that the request frequency must be higher than or equal to t_c^{-1}.

If caches cooperate in a mesh architecture, it is enough for the attacker to infiltrate one cache, and the item will be available to all connected clients. In the case of a tree-like hierarchy, the attacker ideally would like to infiltrate the root in order to make the file available to the maximum number of clients. In a LRU hierarchy, higher-level caches make sense only if they have a strictly higher characteristic time than the caches at the lower levels [17]. This is because a LRU cache works like a low-pass filter on (aggregate) request frequencies with a cut-off frequency of t_c^{-1}. Thus, if an attacker wants to infiltrate the root, he only needs to determine the characteristic time of the root cache, and adapt the request rate accordingly. We describe a measurement algorithm in Section 4.4.2. While measuring the characteristic time of the root is challenging, targeting the root instead of a leaf requires a lower request rate.

Impact: An attacker can use this technique to keep files that have been deleted from the origin server available to a broad range of users by using the CCN infrastructure as a content distribution network. In particular, publishers of illegal content can seed their content temporarily from a web server and then keep the files inside the CCN network, which means that shutting down the web server is not sufficient to prevent the content from being retrieved.

A variant of this attack just increases the popularity of legitimate content to offload the servers. For instance, a content provider could link its unpopular files to its most popular files so that they load together. By increasing the request frequency to be slightly higher than the characteristic time, the items will be retrieved from the cache instead of being fetched from the origin server. Thus, a content provider can abuse its popularity to obtain free content distribution services.

Countermeasures:

- To prevent retrieval of illegal content, a list of censored names could be defined. Routers would not deliver content that is on the list.
- Caches could be forced to periodically revalidate content with the original content source. However, it is not clear how a cache could distinguish a response from another cache from a response that has been fetched directly from the source.
- False locality attacks can be detected with a heuristic algorithm described in [15].

Risk: Low. The attack has a high complexity compared to easier means of achieving the same effect, and countermeasures are available. Potentially, a high request rate would be necessary.

3.4.2 DoS by Decreasing the Efficiency of Caching

By modifying the popularity distribution of requests observed by a cache, an attacker can decrease the efficiency of caching. If many of these attacks take place at the same time, the increased bandwidth requirements could lead to denial-of-service against network infrastructure or content sources.

Assumptions: Caching enables considerable bandwidth savings. To save costs, the network/server infrastructure is designed for a workload with caching enabled. The attacker has access to one or several machines (e.g., a botnet) that are connected to a cache.

Attack: The attacker selects a set of unpopular files U, and requests them periodically to increase their popularity. The request frequency of U should be similar to the request frequency of the truly popular files P. The size of U has to be chosen such that the observed overall popularity distribution at the cache becomes considerably more uniform. Because the cache is not large enough to keep all content, caching becomes less efficient and more requests have to be forwarded upstream. This attack is known as *locality disruption* attack [15].

Impact: The hit rate of the caches decreases, which increases the load on network links (cost for the ISP) and also increases the number of files that are directly retrieved from the content source (cost for the content provider). If the infrastructure has been designed for a workload with functional caches, the infrastructure might become congested. Ultimately, this might result in a denial of service.

Countermeasures:

- Provide infrastructure for the worst case, that is, for an assumed cache hit rate of zero. However, this would result in a high cost and question why CCN should be used at all.
- A heuristic algorithm to detect locality disruption attacks is described in [15].

Risk: Medium. Only detection heuristics are available; the attack requires a botnet and potentially high attack traffic, but it is feasible.

3.4.3 DoS by Filling Available Memory of a Router

By filling up the memory that routers reserve for keeping the communication state, attackers can degrade or disrupt the service delivered to other users.

Assumptions: CCN routers need to keep in memory the Pending Interest Table that contains every Interest that has been received and forwarded, but not yet answered. The memory for these entries is limited. We assume the attacker to have access to one or several machines, such as a botnet. The attacker has a colluding content source under his control to participate in the attack.

Attack: The basic attack consists in requesting a large number of different, inexistent names in order to fill up the PIT. However, a high fraction of Interests without reply could be detected by the router. Therefore, the attack should be refined by colluding with a content source: The content source generates replies, but delivers them only shortly before a PIT entry in the router would time out. In this way, the number of requests necessary to fill up the PIT is minimised. Routers could still detect the frequent accesses to the colluding content source's prefix, thus the attacker might use several different prefixes to evade detection. In particular, if an attacker controls a botnet, he could use every machine to serve content, meaning that there would be as many requestors as colluding content sources.

Every pending Interest has a size of several hundred Bytes for the name and additional fields such as `exclude`, `scope` etc. As the attacker controls the Interests that are generated, he can craft Interests that maximise memory consumption. For instance, the attacker can accelerate the attack by choosing very long names.

Impact: If the PIT is full, legitimate clients can be denied service.

Countermeasures:

- Do not store the full requested name in the PIT, but use hashing techniques to save memory and prevent an attacker from speeding up the attack.
- Drop Interests at the head of the PIT instead of at the tail. That is, a new Interest is always admitted, and replaces the oldest pending Interest. This makes it more difficult to fill up the PIT by delaying responses from the content source. (CCN is unreliable, thus routers are allowed to drop Interests.)
- Develop a mechanism to detect a large number of requests to a very low number of name prefixes.
- To implement reverse forwarding, use a data structure, potentially similar to a bloom filter, that stores exact information under normal conditions, and slowly degrades to flooding as it reaches its capacity limit.
- Design stateless routers without PIT: Use a loop-free topology and add reverse source routing fields to Interests and data packets.

Risk: Medium. The attack targets central components of the network, thus it has high impact. However, it requires a botnet and high attack traffic.

3.4.4 Cache Snooping: List Cache Contents, Monitor Object Access, Copy Conversations

By requesting data just like a regular user, and by examining whether the response was cached, attackers can extract privacy-relevant information from caches. Furthermore, if the naming scheme permits to identify items belonging to the same message exchange, it is possible to clone the full conversation.

Assumptions: Caches are the heart of CCN. In a typical network, every device that carries out forwarding tasks can be expected to be equipped with a cache: Core Internet routers, access routers, DSLAMs, and also routers in corporate networks. This means that there will be caches very close to the end user, and because of the limited number of users that access these caches, they will contain very specific content. We assume that the attacker is directly connected to the same cache as his victims.

Attack: Cache snooping is a technique used to determine the contents of a cache. To find out if an item with a given name is cached, the attacker sends a request for the data item to the cache and measures the response time. The attacker can send a second request for another data item that is guaranteed not to be cached, and compare the response times to determine where the first data item came from (from the probed cache, from any intermediate cache, or from the content source).

Alternatively to using timing, there might be other means for an attacker to find out information about a cache's content. For instance, CCN's prefix matching and the `exclude` field in Interests allow an attacker to retrieve the entire contents of a cache without knowing the names beforehand.

If the attacker can find out the scheme that is used to name subsequent data items in a conversation, he can retrieve them from the cache and reassemble the entire conversation. Even if the content is encrypted, the attacker might be able to find out valuable information from unsecured side channels [18, 19].

Impact: The potential privacy threat ranges from finding out that two entities are communicating to drawing conclusions about the contents of the conversation.

Countermeasures:

- Use a minimum response delay for every answer provided by an access cache. While this delays legitimate requests, it still preserves the bandwidth saving and limits the privacy leak in timing attacks.
- Encrypt names of dynamic content so that only authorised parties can attribute a data item to a conversation.
- Disable complex Interest features such as `exclude` or limit its use over time.

Risk: High. Every user can carry out the attack and potentially gain access to a large range of data.

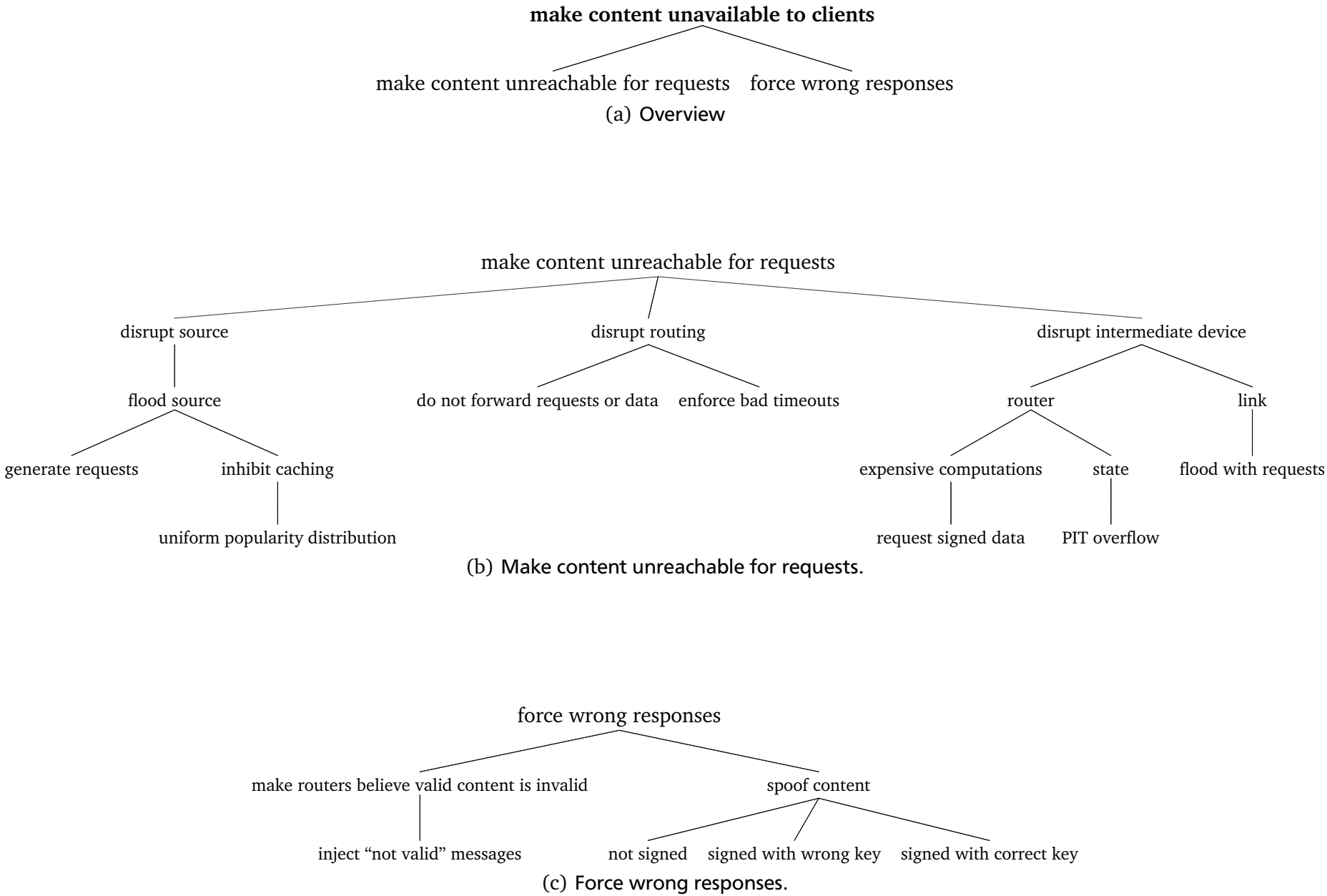
We discuss this attack, and more countermeasures, in more detail in the next chapter.

3.5 Conclusion

In this chapter, we have shown that the architectural differences of CCN compared to TCP/IP have consequences on network security:

- Caching implies a tradeoff between network efficiency and user privacy.
- More powerful routers increase the network's attack surface.
- CCN makes it difficult to identify the origin of a request.
- Denial-of-service attacks require more effort on CCN than on TCP/IP, but they are still feasible.
- Per-request state in routers can be abused for denial-of-service attacks.
- Caches can be misused in the following ways:
  - Attackers can use caches as storage to make their own content available.
  - The efficiency of caches can be decreased by attackers with the goal of denial-of-service attacks.
  - Content can be extracted by any attacker connected to the cache, putting users' privacy at risk.

In the next chapter, we will analyse in detail cache-based attacks on user privacy.



(a) Overview

(b) Make content unreachable for requests.

(c) Force wrong responses.

Figure 3.1: Attack tree for denial-of-service attacks in CCN.

4 Cache Snooping

In networking systems, caches are used to increase efficiency by storing a copy of requested data in a cache. Subsequent requests can be satisfied with the local copy found in the cache instead of fetching the content from the content source. This efficiency gain is twofold: Firstly, the data traffic to the content source is reduced because some requests can be satisfied locally. Secondly, caches can reduce the response time for cached content because caches are usually located closer to the clients than the content source.

However, caching introduces a *fundamental tradeoff between efficiency and privacy*: While cached content increases the efficiency of the system, the presence of content in a cache can be abused to violate users' privacy. The types of privacy-relevant information that attackers might be interested in can be related to:

- who is communicating with whom (*meta-information*), or
- what is the data being exchanged, i.e. the actual *contents* of the communication.

The attack that aims at extracting communication traces from a cache is called the *cache snooping attack*. In this chapter, we examine three distinct attacks:

1. *Obtaining a copy of the cache's contents* (Section 4.5). The attacker is interested in a snapshot of the cache, revealing the objects that are currently being requested by the cache's users. By comparing the cache's contents with a list of "interesting" objects, the attacker can find out if someone of the cache's users has visited a particular page, or seen a particular object. Furthermore, an attacker can prepare subsequent attacks by determining the protocols that are being used by the user population.
2. *Analysing access to a given name* (Section 4.6). The attacker checks if any of the cache's users has recently accessed an object with a given name. By regularly repeating these probes, an attacker can obtain more precise access time information, and even infer the request rate. Periodic probing also enables an attacker to observe a cache and trigger an alarm if some specific object has been served to any of the cache's users.
3. *Cloning conversations* (Section 4.7). The attacker uses the cache to obtain a copy of all data exchanged between two participants, and reassembles their conversation. Even if the communication is encrypted, side channels such as message timing and message size can leak privacy-relevant information.

Previous works have targeted communication traces in local DNS caches [20, 21, 22, 23]. They permit to remotely inspect which web sites are being accessed by the client population, what applications are being used, and with which relative popularity. Another work [24] described how malicious web sites can infer users' browsing histories from their web browser caches. We discuss these works in more detail in Section 4.1.

From an attacker's point of view, caching has several advantages. Firstly, caching relaxes attacks in the time domain: Large caches might keep the data for a long time after the original conversation has ended. Thus, an attacker does not need to carry out his attack at the same time as the victim's conversation takes place. Secondly, caching decouples attacks in the space domain: An attacker only needs snooping, but not sniffing capabilities. In other words, the attacker only needs to be connected to the same cache as the victim; direct access to the wire (or the ether) used by the victim is not necessary any more.

Cache-based network architectures potentially increase privacy leakage. Current deployments of caches, such as DNS caches, web caches, and CDNs, are application-specific and serve a large population. For

instance, DNS caches are typically deployed at a high aggregation level inside a provider's network, such as on a per-city basis. Cache-based network architectures, in turn, claim to be application-independent and deploy caches at a lower level in the aggregation hierarchy: CCN suggests to have cache memory available in every network device, including routers and DSLAMs. The lower number of potential users per cache means that the information leaked by the content found in the cache is more significant as there is less ambiguity in attributing data to users. Furthermore, CCN's application-independence potentially gives an attacker access to a wider range of information, including types of content that are not cached today.

Although previous work has identified several cache-based attacks for specific applications, as we summarise in Section 4.1, we are not aware of any work that discusses the general impact of caches on user privacy in network architectures such as CCN. As we have argued above, generalised caching exacerbates the privacy threat that users are exposed to. We believe that it is important to study cache-based attacks in the context of network architectures to raise awareness for the problem, and to develop countermeasures before the architecture is eventually deployed. While we use CCN as a case study, as far as its commands and protocols are already known, we suspect that the problems that we reveal in this chapter apply as well to other proposed Internet architectures.

For the attacks described in this chapter, we assume that the attacker's objective is to find out sensitive information about his neighbours. We base our description of these attacks on the system model outlined in Section 4.2, and follow the way that an attacker would typically proceed:

1. Firstly, the attacker needs to define what sensitive information is (Section 4.3). The output of this step should be a list of interesting object names. Examples include called phone numbers, or the names of web sites or videos.

2. Secondly, the attacker must gain some information about the topology. He should find out approximately how many users are connected to the same cache as he is, how many users are aggregated at the next level, and so on. Some of the techniques require the attacker to have estimates on the round-trip time to caches at the different levels of the hierarchy. Some techniques also require knowledge of the average lifetime of an object in the cache, if it is not requested again after it has been added. Measurement algorithms to gather this knowledge can be found in Section 4.4.

3. Thirdly, the information has to be retrieved from the cache. At this point, the attacker carries out one of the three attacks introduced above. We describe them in detail in Sections 4.5, 4.6 and 4.7.

After having described these three attacks, we conclude this chapter with a discussion of potential countermeasures in Section 4.8.

4.1 Related Work

DNS snooping [20] permits to find out from a DNS cache if someone with access to this cache has recently resolved a given domain name, for instance to visit a web page or to send email. This is achieved by querying the cache to determine if information about the domain name is locally available. DNS caches can be queried in two ways:

- *Non-recursive queries* are always answered with local knowledge of the cache. In case no data is locally available, an error message is returned.

- *Recursive queries* are forwarded to other caches if necessary to supply a valid response. Although recursive queries always return a response, there are two ways for an attacker to find out if the information was originally cached locally: Firstly, the response is slower if the data has to be fetched from somewhere else. Secondly, DNS responses always carry the remaining time-to-live (TTL) of the data in the cache. The initial value is defined by the domain's authoritative DNS server. If the

TTL returned with the reply is low compared to the initial value, there is a high probability that the data was cached.

In addition to some scenarios in which the information gained from the cache is used for further attacks, the author gives recommendations for countermeasures:

1. To prevent distant attackers from gaining access to a DNS cache, every cache should allow requests from local clients only.

2. To make the attack more difficult, a DNS cache should not allow non-recursive queries.

3. A DNS cache can add entropy to the TTL by adding some randomness to the initial TTL when inserting an entry into the cache.

For speed optimisation, several web browsers pre-fetch domain names appearing in links on web pages. Krishnan and Monrose [25] show how to exploit pre-fetching of the DNS names found on search engine result pages to infer the keywords that have been used during searches. Attackers can build a list of domain names that would be resolved for a given keyword, and probe DNS caches for these domains. By comparing the initial TTL values set by the authoritative DNS server and the remaining TTL of the cached entries, it is possible to detect if these domain names have been inserted into the cache at approximately the same time. If this is the case for the majority of the expected names, the attacker can conclude that someone has used the corresponding keyword.

The authors show that with a 30 minutes probing delay, they could obtain an accuracy of 85 % in detecting searches for a given keyword. They note, however, that in the general case, inferences can be made only about the cache population as a whole, but not about individuals. CCN, in comparison, potentially provides more information: Firstly, whole object names are accessible and discoverable, not only domain names. Secondly, the user population per CCN cache is much smaller than the population of a DNS cache.

Several works [21, 22, 23] measure the time a DNS entry is not cached, and use this information to infer its request rate. They make the following simplifying assumptions:

- The time that an item stays in the cache is constant and equal to the TTL defined by the respective authoritative name server. In particular, the number of cache hits while the item is cached has no influence on the item's lifetime in the cache, and no replacement takes place in the cache.

- Client requests are independent and identically, exponentially distributed, and so are the gaps during which the item is not cached.

In the case of low popularity, the hit rate obtained during cache snooping is a sufficiently good approximation of the request rate. As the popularity of the item grows, client cache hits are more likely to occur. In this case, the request rate λ can be computed as the average over the time gaps between eviction of an item (assuming that the last request took place shortly before), and the next time the item is inserted into the cache. The authors show that it is sufficient to probe the cache once every authoritative TTL. Non-recursive queries are preferred as they directly reveal whether the data was cached, and because they do not modify the state of the cache. Because the sum of Poisson processes is a Poisson process, the number of clients can be computed if the per-client request rate is known.

Felten and Schneider [24] show how malicious web sites can probe client-side caches, such as the web browser's cache, to find out if the user has recently visited another, unrelated web site. The authors discuss the following countermeasures:

1. *Disable caching*. Due to the obvious performance penalties, they discard this solution.

2. *Alter the hit/miss performance of the cache*. Cache hits can be made slower, or random, in an attempt to conceal them. However, the authors dismiss this possibility: "If we make hits as slow as misses, then attackers will be handcuffed; but in that case we might as well have turned off caching." However, this statement seems to ignore the bandwidth savings that could still be made.

3. *Compartmentalisation*. Every domain would have its own virtual cache, so that web sites cannot probe the state of other sites. Yet, this countermeasure would considerably decrease the cache hit rate, and provide only small efficiency gains.

4.2 System Model

We use the following network model and assumptions in our discussion:

- Caches are arranged in a *tree-like hierarchy*, where the leaves correspond to DSLAMs. All users are connected directly to one leaf cache. Requests are routed upwards in the hierarchy; there is no collaboration between neighbouring caches. If no cache hit occurs, the root cache forwards the request to the corresponding content source.

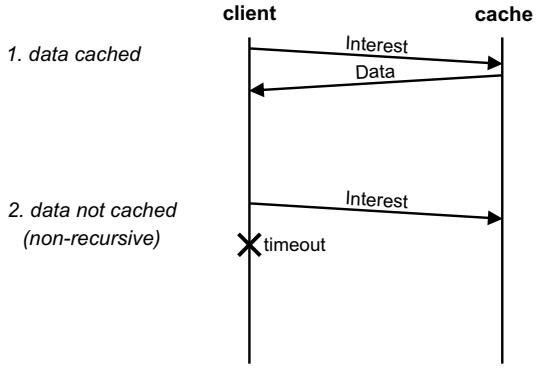
- Every cache is limited in size, using LRU as replacement policy.

- If a cache hit occurs, the memory read latency is the same for all caches, no matter where in the hierarchy the hit occurs. That is, all caches are implemented in the same technology (using either main memory or disk), but not a combination of both.

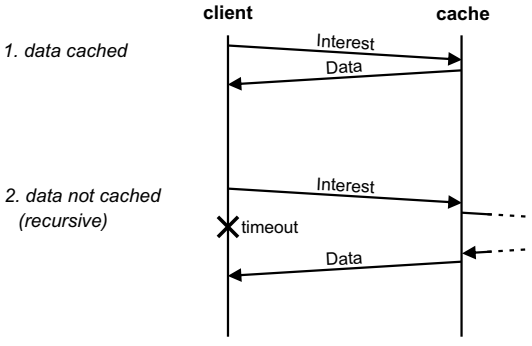
- Attackers are interested in the best possible bound on the number of users who might have requested data, i.e. they will always target the first cache that the attacker and the victim have in common.

An attacker can use either recursive or non-recursive queries to implement his attacks. This choice is orthogonal to the attack type, and depends on what mechanisms the system under attack makes available. We simplify our discussion by assuming that the attacker and the victim are directly connected to the same cache[1].

There are two ways of discovering the presence of an object in the cache:



(a) Non-recursive: After a cache miss, the client does not receive any reply.



(b) Recursive: Client receives a reply after a cache miss.

Figure 4.1: CCN message exchange in different request modes.

[1] Alternatively, the attacker can connect to the victim's leaf cache by tunnelling CCN in direct UDP packets.

- A *non-recursive* query to a cache, as shown in Figure 4.1(a), is answered with local knowledge only. It times out if the data item is not contained in the cache.

  In CCN, non-recursive queries are possible by sending an Interest message with the `scope` field set to 0. However, this functionality might be deactivated for security reasons.

- A *recursive* query (Figure 4.1(b)) always returns the queried data, as long as it can be found somewhere in the system. The attacker exploits the timing side channel to infer from the response delay which cache in the system had the item stored. The assumption behind this is that the (distribution of) response delays of caches at different levels of the hierarchy is significantly different, so that the attacker can tell them apart.

  Recursive queries are the normal behaviour of CCN. Therefore, it is much harder to develop countermeasures against this type of attack.

A non-recursive query is less intrusive than a recursive query: After a non-recursive cache miss, the cache remains unmodified, while a recursive query would insert the missing item. Both methods behave in the same way for a cache hit: The lifetime of the object in the cache will be increased[2]. This means that with any query type, if repeated queries are necessary, special measures have to be taken to make sure that the attacker does not incorrectly observe a state that he caused himself.

A clear advantage of non-recursive queries is that the result is always exact. With recursive queries, there is some remaining probability that the attacker makes a wrong guess because of variations in the response delays.

If the attacker cannot directly connect to the victim's cache, he can target the first common cache. However, this approach makes the attack more difficult to carry out: The attacker needs more precise latency estimates and must enhance the algorithms to take into account a multi-cache hierarchy.

4.3 Attack Goals

Before an attacker can start the actual attack, he has to define his goals. In particular, the attacker needs to decide on who the potential victim is going to be, and which application or content is to be targeted. The output of this preparatory step is a list with the names of interesting, potentially sensitive content objects.

When it comes to selecting victims, the attacker can either try to gather aggregate information about the neighbourhood *in general*, without identifying any individual. For instance, he could be interested in knowing if someone in the neighbourhood downloads a video with, say, instructions on how to build a bomb, or if anyone is looking for information about a given disease. In this case, the attacker searches for the content himself and compiles a list with the names of the objects that he found.

Alternatively, the attacker can target a *specific* person in the neighbourhood. Examples include knowing when a given user has last connected to some web site, what data he entered into that web site, or when he has last received a phone call. To do this, the attacker needs knowledge about the naming schemes used by the protocols of the respective applications. For instance, if an application's data always carries the prefix `/application-name/user-name/`, then the attacker can add this prefix to his list.

The privacy impact of the attacker finding out about content present in the cache depends largely on the content's popularity. Static, popular content such as the latest music video carries little information that might be of interest for an attacker; it has low entropy. Static, but rare content such as a terrorist training video leaks some more information about the neighbourhood, although it might not be possible for the attacker to attribute the video to the person who downloaded it. Dynamic content, such as individual web browsing sessions or phone calls, are not usually shared between users. If the attacker can retrieve some

[2] This is an important difference between LRU caches as used in CCN, and the related work about DNS caches. We will come back to the cache lifetime in Section 4.4.2.

information from the content itself (if it is unencrypted) or from a side channel (such as the name, or the size), then the entropy is probably highest, because the information is highly individual.

While the cases described above assumed that the attacker needs to know interesting names prior to the attack, it might even be possible for the attacker to gain complete knowledge of a cache's contents. This type of attack is the most powerful one: The attacker would be unconstrained in the heuristics that he can apply to analyse the data.

4.4 Topology Intelligence

Before an attacker can start the actual attack, he must find out some basic information about the cache topology that is required by the attack algorithms:

- The *latency* to a cache can be used to tell if a data item requested by the attacker is cached there, or somewhere else in the hierarchy.
- The *characteristic time* of a LRU cache [17] is the time that an object remains in the cache, given that it is not accessed any more after its initial request.

  As discussed in Section 4.2, probes to LRU caches can modify the state of the cache. This metric indicates how long the effect of probing persists on average, and gives the minimum sensible probing interval.

In the following, we describe two algorithms to measure these characteristics. At a first time, we consider only the leaf cache, i.e. the cache at the DSLAM level.

4.4.1 Latency Measurement

Latency estimates are required if the attacker wants to exploit the timing side channel to attribute a response to the cache that most likely delivered it. The algorithm depicted in Figure 4.2 measures the latency of the attacker's leaf cache and works in two phases:

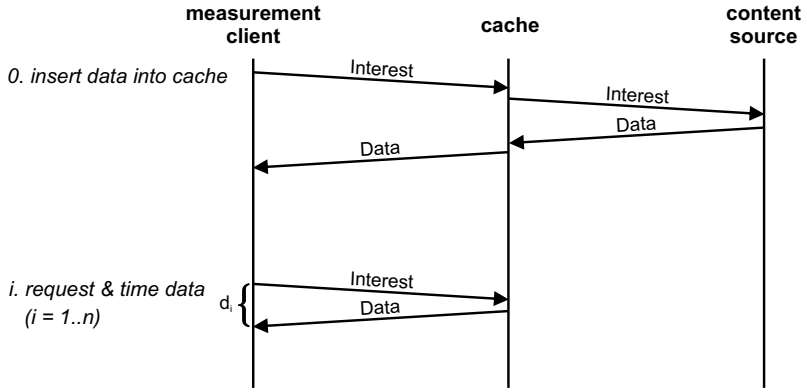


Figure 4.2: Measuring the latency of the first CCN cache.

1. Before the measurements begin, the attacker must identify an arbitrary data item that is contained in the cache under measurement. Alternatively, he can insert one into the cache as shown in Step 0 in Figure 4.2.
2. The attacker now requests this data item, and measures the time difference between the Interest message and the incoming response. The attacker can repeat this step n times and calculate the mean and standard deviation to obtain a better estimate.

The delay between two successive probes should be low enough to make sure that the data has not been evicted from the cache in the meantime. At the same time, the delay should be high enough so that a representative sample is taken (with respect to background traffic in the cache's queues).

Using this latency estimate, the attacker can guess if data he requested was delivered from the local cache or fetched elsewhere: Let $\overline{d} = \text{mean}\{d_1, ..., d_n\}$ the mean latency of the first cache measured with the algorithm above, σ its standard deviation, $d_{request}$ the latency of the request of unknown origin, and α a parameter to influence the weight of the probe variation. If $d_{request} \gg \overline{d} + \alpha \cdot \sigma$, then it can be assumed that the request has been answered by a cache other than the local one.

4.4.2 Cache Lifetime Measurement

The goal of this section is to measure how long an item that is requested only once remains in a cache, as shown in Figure 4.3(a). Che et al. [17] call this the *characteristic time* of the (LRU) cache, because it defines the minimum access frequency that an item needs to have in order to be served from the cache. Items with a lower access frequency (i.e., requests in time intervals longer than the characteristic time) will not be found in the cache and have to be served from caches or content sources further up in the hierarchy. The authors show that for a LRU cache with a Zipf-like item popularity distribution and a Poisson request arrival process, the characteristic time can be approximated to be constant for each item, and to be the same for all items.

In the context of cache snooping attacks, the characteristic time implies the minimum time that an attacker has to wait before he can use the same probe item again. If the attacker reuses a probe item before the characteristic time has passed, he risks measuring a state that he has caused himself with his previous request.

To measure the lifetime of an object in the cache, the attacker uses items that are guaranteed to be used only by him. This can be achieved by using a colluding content source, or by otherwise generating items with unique names. To verify a given lifetime guess, the attacker first requests the item to insert it into the cache, and subsequently verifies after some time if the item is still contained in the cache.

The measurement algorithm that we propose works in two phases: First, exponentially increasing guesses are used to find an upper bound on the cache lifetime. Then, the technique of nested intervals is used to refine the guess.

In detail, the algorithm works as follows:

- **Initialisation:** Set $t_{min} = 0$, $t_{max} \to \infty$, and $i = 0$. Use $t_0 > 0$, $\beta > 1$, $0 < \gamma < 1$ and $n \in N^+$ as fixed, but tuneable parameters of the algorithm (as explained below).

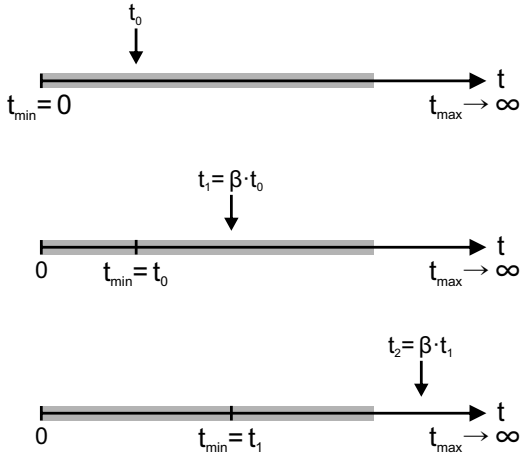
- **Find upper bound:** Insert a new item into the cache, and request it after time t_i. If the item is not contained in the cache any more, set $t_{max} = t_i$, $i = i + 1$ and jump to the next step of the algorithm. Else, set $t_{min} = t_i$, $t_{i+1} = \beta \cdot t_i$, $i = i + 1$ and repeat this step. This process is illustrated in Figure 4.3(b).

- **Refine estimate:** Insert a new item into the cache, and request it after time $t_i = t_{min} + \gamma \cdot (t_{max} - t_{min})$. If the item is not contained in the cache any more, set $t_{max} = t_i$. Else, set $t_{min} = t_i$. Increase i, and repeat this step until a predefined number n of iterations has been performed. Figure 4.3(c) shows two iterations.

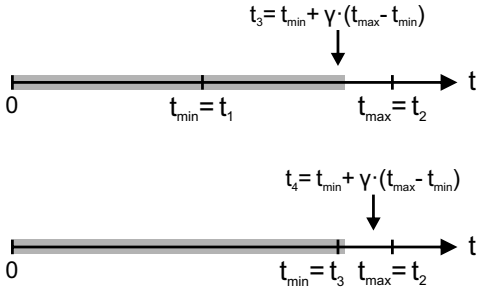
The start value t_0 can be set to a rough estimate of the characteristic time, if known. β defines how fast the guesses in the initial phase increase. γ defines which point of the interval will be probed next. Because the goal of the algorithm is to find an upper bound, we use $\gamma = 0.75$. For the other parameters, we use $t_0 = 1$ second and $\beta = 2$. The number of iterations n can be determined experimentally. Alternatively, this stopping condition could be replaced by a minimum change in the interval borders between iterations.

There are several ways how this algorithm could be improved. Firstly, it could be speeded up by not inserting a new item after a cache hit—because of the properties of LRU, it would be sufficient to view the hit time as equivalent to the insertion time. Secondly, this sequential algorithm can be speeded up even more by parallelising probes for several guesses. This might be useful especially for very long characteristic times.

(a) The life cycle of an item inserted into the cache. The time between insertion and eviction is called the *characteristic time*.



(b) Finding an upper bound on the characteristic time.



(c) Refining the interval around the characteristic time.

Figure 4.3: Estimating the characteristic time of a cache.

Both the latency and characteristic time estimation algorithm could be enhanced to target caches further up in the hierarchy: If caches use LRU, a hierarchy makes sense only if the characteristic time of the higher-level caches is higher than the characteristic time of the lower caches [17]. Under this assumption, once an item has been evicted from the leaf cache, it will still be available for some time in the higher-level caches, thus the probing algorithms can target the next cache in the hierarchy. Different caches can be differentiated by observing their response latencies. We leave this enhancement for future work.

4.5 Attack I: List Cache Contents

This attack allows an attacker to reveal what items are being requested by the client population of a cache. The attacker can use this information in various ways:

- He can infer than someone from the cache population requested a potentially sensitive item. This can be achieved by either manually inspecting the requested objects, or by comparing them to a previously established list of sensitive content. For instance, such a list could contain pages or videos that reveal political views or health conditions.
- The attacker can analyse what protocols are being used by the clients to prepare subsequent attacks. For instance, he could be looking for communication that is vulnerable to side channel attacks to prepare the conversation cloning attack of Section 4.7.

CCN defines a name enumeration protocol, which however seems to be aimed at persistent content stores (repositories), but not caches. For that reason, this protocol is very unlikely to be implemented in router caches. In the following, we describe an alternative algorithm that enumerates the contents of a CCN cache by recursively excluding previously seen names.

CCN uses prefix matching, which means that an attacker does not have to guess a full item name in order to obtain a response from a cache. Furthermore, CCN Interests allow to specify a list (or bloom filter) for name components that may not be contained in the item returned by the cache. Taken in combination, these two characteristics permit an attacker to recursively explore the contents of a cache: First, the attacker requests an arbitrary item, and then he successively sends further requests that have the previously delivered items on the exclude list. In detail, the algorithm works as follows:

- Input $p = /p_0/p_1/.../p_i/$, the prefix of the name space to be explored, and initially set $E = \emptyset$ as the set of excluded name components.
- Send an Interest with p and E to the cache. In case of a cache miss, the algorithm terminates. For a data response, let $n = /n_0/n_1/.../n_j$ the name of the response. Then $j \geq i$ and $p_0 = n_0$, $p_1 = n_1$, ..., $p_i = n_i$. If $j > i$, $\forall e \in E : n_{i+1} \neq e$. In other words, the response is an arbitrary cached item that has p as prefix, and none of the elements of E as the $(i+1)$th name component. The attacker now sets $E = E \cup \{n_{i+1}\}$ and repeats this step.

After the algorithm has terminated, E contains all name components that are available in the cache at level $i+1$ with prefix p. The attacker can further explore the cache by repeating the algorithm with prefix $p = /p_0/p_1/.../p_i/e/$, for one specific (or all) $e \in E$. For instance, to explore the entire cache, the attacker starts with $p = /$ and recursively repeats the algorithm for all $e \in E$ found in the respective instances of the algorithm. (A similar version of this algorithm is implemented in the command `ccnls` provided with the CCNx prototype.)

A drawback of this algorithm is that it has to download the data objects, thus the attack might be too slow to obtain a consistent snapshot of a large cache. However, it might prove useful to explore limited subspaces.

4.6 Attack II: Probe Specific Name

The attack outlined in this section assumes that the attacker knows the name that he wants to observe. We first describe an algorithm that allows to detect cache insertion and eviction times of an item with arbitrary precision (Section 4.6.1): Periodic probing of the cache can be used to timely detect accesses to a given item. For instance, an attacker could be interested in logging all the voice call events of one user, or all the log-in events of a user on, say, Facebook. Additionally, this information allows an attacker to infer the request rate of the observed item (Section 4.6.2).

4.6.1 Insertion & Eviction Time Detection

A single probing request to a cache reveals whether the item is currently cached. However, with a single probe, an attacker can only look into the past for at most the characteristic time t_c: If the item is currently cached (or not cached), the attacker only knows that there has been at least one request (or no request) during the past t_c time units. In particular, the attacker does not know when exactly an item was added to the cache, or when it was evicted.

Periodic probing of the same item is limited in accuracy by a probe interval of at least t_c time units because an attacker's probe can alter the state of the cache during t_c time units. In disk-based caches, t_c can grow to the order of hours or days, which would make the attack impractical. Additionally, the uncertainty in measuring a cache's characteristic time might be high, which would have the same consequences.

However, CCN splits large files into smaller, numbered chunks[3]. It is very likely that a client who accesses the first chunk of a file will also access the subsequent chunks. Similarly, an attacker might be able to identify distinct items that are always requested together, such as images embedded into a web page. (For simplicity, we assume that all such chunks are requested at exactly the same time.) The attacker can leverage the existence of these items to increase the probing frequency by requesting the chunks in parallel and shifted by the desired probing interval.

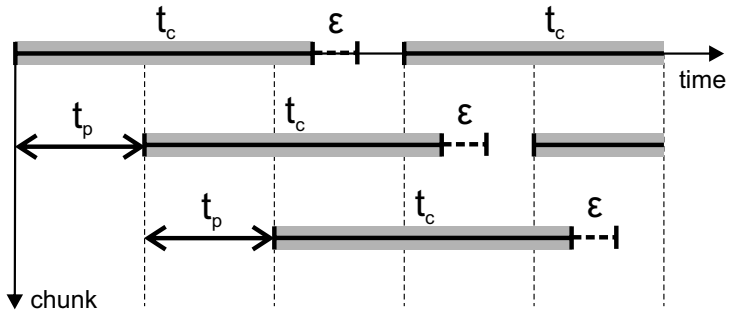


Figure 4.4: Parallel probing involving three chunks.

Let t_c be the characteristic time of the cache, ϵ the uncertainty of the measured characteristic time, and t_p the probe interval desired by the attacker. Then the number of chunks necessary to carry out the attack is $\lceil \frac{t_c+\epsilon}{t_p} \rceil$. Figure 4.4 illustrates the attack. It lets the attacker trade off lack in t_c precision (and timeliness of detection, if at most one access occurs during t_c) versus the amount of attack traffic and chunks that are necessary.

For caches with small t_c and with a large number of chunks, an attacker can set t_p to small values to obtain a precise estimate of the cache insertion and eviction times of an item. As we show in the next section, this data can then be used to infer the request rate of the observed item.

4.6.2 Infer Access Rate

The related work discussed in Section 4.1 proposed algorithms to infer the access rate of a DNS item from the gaps where the item is not cached. There are several differences between DNS and CCN caches that require these algorithms to be modified:

- The cache lifetime of a DNS item is always constant, while in a LRU cache the lifetime depends on the number of hits.
- No assumptions can be made about the number of accesses to a DNS item while it is cached, whereas in a LRU cache, a lower bound can be given by dividing the item's lifetime by the characteristic time of the cache. Furthermore, if a LRU cache evicts an item, it is clear that this item has not been accessed for at least the characteristic time.
- It is possible to infer the exact cache insertion time of a DNS object from any cache hit, while this is not possible in CCN.

Because of these characteristics, this kind of attack is possible in CCN only if a parallel probing algorithm similar to the one described in the previous section can be executed. Furthermore, the attack is feasible only if the gaps where the items are absent from the cache are sufficiently long compared to the precision of the probing algorithm. This implies that the access frequency of the probed items may not be too high. In particular, the attack is impossible if the access frequency is higher than t_c^{-1}, where t_c is the

[3] The default chunk size is 4 KB.

characteristic time of the cache. (To overcome a very low access frequency, several instances of the measurement algorithms could be executed in parallel, using separate sets of chunks, to obtain a sufficient number of samples in reasonable time.)

In the following, we assume that the cache insertion and eviction times have already been measured.

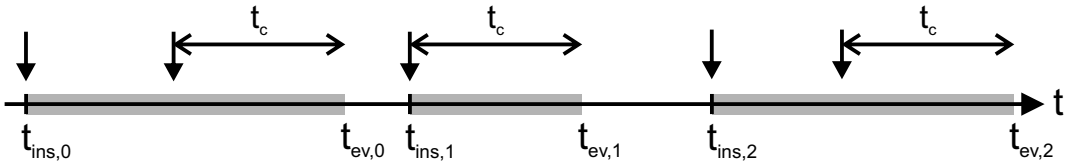


Figure 4.5: Measuring the request rate in a LRU cache.

Figure 4.5 shows the timeline of an object cached in a LRU cache. Every cached instance of the object is inserted at $t_{ins,i}$, and evicted at time $t_{ev,i}$. These are the only times known to the attacker.

Every request delays the eviction time by t_c from the request time. However, from $t_{ins,i}$ and $t_{ev,i}$ alone, an attacker cannot infer exactly how many requests occurred during this time span: The lower bound is $\lceil \frac{t_{ev,i}-t_{ins,i}}{t_c} \rceil$, but there is no upper bound. Yet, an attacker can infer the time span between two successive requests if the object is evicted between them: $t_{gap,i} = t_{ins,i+1} - t_{ev,i} + t_c$. If there is a large number of such gaps, the attacker can compute the overall request rate as follows: $\lambda \approx \text{mean}(t_{gap,i})$.

In this discussion, we neglected the effect of chunks not being requested at exactly the same time, and possible imprecisions in measuring the insertion and eviction times. Future work should study how these impact the accuracy of the request rate estimate.

4.7 Attack III: Clone Conversation

Caches keep a copy of all content exchanged by their users for at least some short time period. Because caches do not enforce any access control policies, any user that knows the name of a data item can retrieve it from the cache.

A data exchange, or conversation, between two users can be copied and reassembled by an attacker if he finds out the names of the items belonging to the same exchange, and if he can determine their sequence in the conversation. For instance, the Voice-over-CCN prototype [7] uses the naming scheme `/domain/user/call-id/rtp/sequence-number` for the voice data exchanged during a call. Once an attacker knows the prefix of a specific call instance, such as `/telefonica/alice/1234/rtp/`, the attacker can request these names with an increasing sequence number to obtain and reassemble a copy of the voice data stream.

CCN implements access control with encryption. That is, everyone can obtain a copy of the cipher text, but only the authorised parties possess the keys that are necessary to decrypt it. However, even encrypted data payload might leak information through side channels. One side channel is meta-data such as an item's name. The cloning attack described above is made easy because VoCCN does not encrypt the name and sequence numbers.

In [18], Wright et al. describe how to check an *encrypted VoIP conversation* for occurrence of a set of phrases. The attack is possible if the conversation is encoded with a variable bit rate codec together with a length-preserving stream cipher. The result of this is that the encrypted packet size depends on the speech input (the phonemes) during the sampled time interval. Using the packet sizes resulting from encoding the diphones/triphones of the pronunciation of a target phrase, the authors train a profile Hidden Markov Model to generate a speaker-independent model of the phrase. When trying to match an observed conversation with the model, they detect instances of the phrase with 50 % precision and recall on average. As a countermeasure, the authors suggest padding the packets to a constant size as a very promising approach.

The VoCCN prototype [7] uses constant-size data messages. It is not clear whether this behaviour comes from padding, or from sending data messages only if enough data has been accumulated in the outgoing buffer to fill up an entire message. In the latter case, an inter-message delay side channel would remain. However, such a side channel would be very challenging to exploit in the scenario assumed in this chapter, where messages have to be extracted from a cache. The reason is that an item retrieved from a cache does not reveal when it has been inserted. An attacker would have to find out this information in real time by periodic probing. However, the precision required to carry out a successful side-channel attack is likely in the order of milliseconds, and it is unclear whether such a precision could be reached in practice.

S. Chen et al. [19] describe a similar attack on SSL- or WPA/WPA2-encrypted web traffic. Both protocols use stream ciphers that do not pad packets, so that the encrypted messages differ in size. With technologies such as Ajax, web applications often respond to a single user input such as a mouse click or a single keystroke. Because this user input has low entropy, the attacker can enumerate all possible inputs, test directly with the application the size of the response packets for every input possibility, and compare with the actual response received by the victim. There might remain some ambiguity in this comparison because some messages might have the same size. However, communication with the web application is stateful, which means that this ambiguity can be reduced in a later step of the conversation: One observed state of the application restricts the set of possible past states. The authors show how a health service leaks medical conditions, a search engine the keywords, a tax form the range of income, and an investment application the fund allocation made by the user. The authors argue that countermeasures must be application-specific. Padding as a generalised countermeasure does not seem attractive as it would need packet sizes of 512 B to drastically reduce the entropy (and utility) for the attacker. At the same time, the network overhead of this countermeasure would reach almost 33 %.

As there are no proposals yet as to the communication protocols over CCN to be used for web-like applications, it is impossible to state whether such implementations would be vulnerable to this type of attack. Rather, we see this as an opportunity to design such protocols in a secure way by identifying potential attacks beforehand.

4.8 Countermeasures

Countermeasures against this cache snooping attack fall into two categories: The attack should be *prevented* by technical measures, and it should be *detected* once it happens so that measures in the real world can be taken.

Prevention

Attackers should be prevented from *discovering the contents of a cache* by adopting the following measures:

- CCN's built-in cache enumeration protocol should not be implemented in caches.
- To prevent listing the cache contents using prefix matching and the exclude functionality, the protocol specification should be changed to make data objects specify on which parts of their name the exclude function may operate (none by default). In this way, content objects (and more generally communication protocols on top of CCN) implicitly define which parts of the name contain sensitive information and must be known in advance.

  Alternatively, the exclude functionality could be disabled and the responsibility for content discovery could be shifted to another layer of the protocol stack where it can be better secured.
- Furthermore, prefix matching could be restricted to a maximum number of unmatched components, which would imply that the attacker has to know a large part of the name.

- If cache space in routers is compartmentalised based on a per-user basis, attackers cannot easily gain information about data retrieved by other users because every user can access only their own cache compartment. However, the overall hit rate would greatly suffer from such a design, and practical aspects such as user identification and additional state in routers make this countermeasure very unlikely to be adopted.

Attacks that *probe the cache for a given name* can be made more difficult in the following ways:

- First of all, non-recursive queries should be ignored by access routers.
- If CCN is tunnelled over UDP, routers should admit queries only from their direct neighbours. That is, users should be able to access only their own access router, but not intermediate or root caches.
- To avoid timing-based attacks, CCN access routers could enforce a minimum response delay. This delay should correspond to the round-trip time to a router in the hierarchy that has a sufficiently large number of users behind it so that the privacy implications of detecting a hit in this cache are low. In other words, if the attacker detects a hit in this cache, at least k users could have requested the item. Because the CCN access router enforces a delay threshold that corresponds to a hit in this cache, the attacker cannot refine his approximation to less than k users. (This corresponds to the concept of k-anonymity [26].) However, this approach can only work if all other side channels in content objects have been closed, i.e. the object itself leaks no information as to who might have requested it. This requirement is difficult to satisfy in general because the list of side channels is potentially open-ended.

  Enforcing a minimum request delay maintains the bandwidth savings that can be obtained with caching, but it trades off the potential delay reduction against a higher level of privacy. Furthermore, there will be additional state in access routers to maintain the timers. Care has to be taken that the implementation of this countermeasure in routers does not introduce new side channels or attack opportunities.

Applications on top of CCN should impede *cloning of conversations*:

- Conversation-oriented protocols need sequence numbers to retrieve the next item in the conversation. If the numbering scheme is unknown to the attacker, for example because the sequence number part of the name is encrypted with a secret key, reconstructing the conversation retrospectively will be more difficult or even impossible.
- If the privacy-relevant part of a name cannot be encrypted, the naming scheme should at least make it difficult for an attacker to guess names. For instance, names could follow the scheme `/application-prefix/random-number/further-parameters/`. If the exclude and prefix matching functionalities are disallowed on the random number component, the attacker must find out the random number from other sources in order to retrieve the data item.
- Content exchanged in dynamic sessions should carry a very low expiry time. Currently, CCN caches are allowed to return stale content. This behaviour should be forbidden to reduce the time frame during which attackers can carry out their attacks. If dynamic, individual data is quickly evicted from caches, retroactive attacks become much more difficult because the attacker must carry out the attack simultaneously with the victim's session.
- Finally, caching of content that is unlikely to be shared could be disabled. Individual web browsing sessions and phone calls lose much of their value once they have ended. If caching is disabled entirely, attackers cannot retrieve the content from the cache any more and must fall back to classical sniffing attacks. To this end, content objects might set a do-not-cache bit. A router should evict such an item from its buffers immediately after all processing and forwarding has completed.

Detection

If an attack cannot be prevented, it should at least be detected. The attacks described in this chapter have typical characteristics:

- An access router might observe periodic polling from the same access link when an attacker is probing for a known name.
- Similarly, obtaining a snapshot of the cache involves querying for a large number of items in very short time, with a very high hit rate, and with high coverage of the corresponding name space in the cache.
- Cloning conversations results in a sequence of packets that arrive at the router being forwarded to two destinations (incoming content: from the uplink to two access links, and outgoing content: from one access link to the uplink and the attacker's access link).

However, detection needs to rely on heuristics. In this context, we see a large potential for false positives: Downloading a large, cached file with many chunks might be mistaken for a cache snapshot attack, and multicast protocols such as multi-party video conferences look similar to the cloning attack. Furthermore, we argue that heuristics can be evaded by very stealthy attacks that simply stay below the detection thresholds. We leave concrete detection algorithms for future work.

4.9 Conclusion

In this section, we discussed

- how characteristics of the cache topology can be measured: The *latency* and *characteristic time* of caches are necessary to prepare the following attacks.
- How attackers can obtain a snapshot of the cache's contents.
- How attackers can monitor access to content by their neighbours.
- How conversations can be cloned to carry out side channel attacks on them.

We proposed the following countermeasures:

- Make CCN queries less expressive.
- Tweak the privacy–efficiency tradeoff: We suggest to maintain the bandwidth savings provided by caching, but to restrict the latency gain by enforcing an artificial lower latency bound to hide which cache answered a query.

Future work should show that the above attacks are feasible with reasonable attack traffic requirements and investigate the accuracy of the algorithms. Our work could be extended to support caches at distinct layers in the hierarchy and to support caches that are implemented with a combination of fast random-access memory and slower disks. Finally, the impact and efficiency of our countermeasures should be investigated.

In the next chapter, we evaluate the algorithm to measure a cache's characteristic time.

5 Evaluation

In this chapter, we summarise our first results of an evaluation of the cache snooping attacks. All types of attacks discussed in the previous chapter are based on at least approximate knowledge of the characteristic time t_c: The characteristic time defines how long communication traces "survive" in the cache, i. e., how far the attacker can look back in time, and consequently how often he has to probe. Therefore, as a first step, we centre our evaluation around the characteristic time.

In the previous section, we assumed that the characteristic time of a LRU cache is constant. However, this assumption does not hold true if the workload changes, and even for stationary workload, it holds true only on the long term [27, 17]. The goal of this evaluation is to determine the order of magnitude of the characteristic time, and to find out to what extent it varies. Once the characteristic time is known, we can deduce the amount of traffic that an attacker needs to send in order to monitor all accesses to a given content object. Furthermore, we would like to evaluate how long the algorithm to measure the characteristic time (Section 4.4.2) takes to converge, and how precise its estimates are. Both results taken together show the feasibility of the cache snooping attacks.

As the evaluation scenario, we chose a DSLAM with a cache for Youtube-like video traffic. We assume that the attacker is interested in monitoring all accesses to one particular video. Section 5.2 contains the details of this scenario and the workload that we are using.

For our evaluation, we decided to implement a simplified version of CCN in a simulation environment. While a real-world testbed would provide us with more realistic results, it would be difficult to get hold of hardware and a corresponding software implementation that is able to execute CCN with a main-memory cache of several gigabytes and to operate at speeds of several hundred megabits per second. Section 5.1 describes our CCN simulator and the abstractions that we made.

After having discussed the results of our simulation in Section 5.3, we derive the amount of attack traffic that is necessary to carry out the attack in Section 5.4.

This should by no means been taken as a complete evaluation, but rather as a hint at what we plan to investigate in future work. Indeed, we intend to quantify the amount of privacy loss that an architecture such as CCN imposes on its users compared to other architectures or CDNs. We will elaborate more on future work in Section 5.5.

5.1 CCN Simulator

Our CCN simulator has been written in Python. It models the following components:

- *Clients* request and receive data. They are used to generate background traffic or to carry out measurement algorithms.
- *Interests* and *data chunks* are implemented as individual, independent units. That is, prefix matching or queries are not implemented; clients must know the exact name of the data to be retrieved, and they must know which chunks belong together to form a single file. Interests and data chunks are assumed to have a constant size of 100 B and 4 KB, respectively: This corresponds to an Interest with a medium-length name, and to a chunk with the default size used in the CCNx prototype.
- *Links* impose transmission delay on all Interests and data chunks transmitted over them as a function of their size. However, links do not implement propagation delay or loss: These can be neglected for the ISP-local scenarios that we envision.

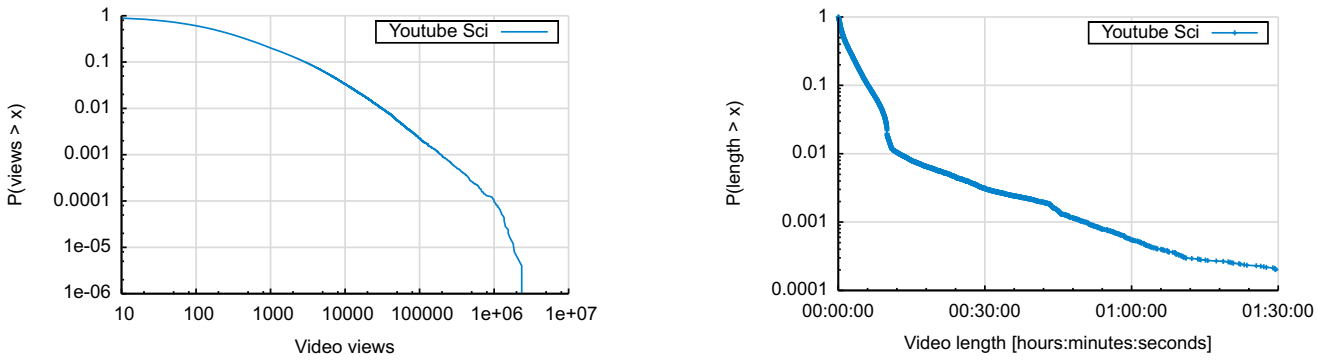
- *Routers* have a cache of limited size with a LRU replacement policy. Routers also use a PIT to aggregate Interests. Each link has a separate and infinite queue, and links can operate concurrently. Routing has not been implemented; instead each Interest is forwarded on a default uplink path. That is, routers always form a tree-like hierarchy.

- The *content source* is at the top of the router hierarchy. It is assumed to have all content available and therefore answers every incoming Interest with the corresponding data chunk.

5.2 Scenario

For this evaluation, we assume the scenario of a DSLAM with a CCN cache for video content. A DSLAM typically has around 1,000 users (e.g. *Free* in France [28]). We assume that during the "rush hours" of the video service, 10 % of the users are simultaneously watching videos: They download the entire video at 2 Mbit/s and watch the video while downloading (or longer if the download finishes earlier than the video playback). Users start downloading the next video once they have finished watching the first one.

The videos downloaded by the clients correspond to the Youtube Science & Technology Category as retrieved on January 15, 2007 [29] and analysed in more detail in [30]. The trace includes a video ID, the length of the video in seconds, and the number of views. We exclude 6 videos that are longer than one day, and keep the remaining 252,249 videos. We transform the number of views per video into a relative popularity, and compute the file size by assuming a constant bit-rate encoding of 384 kbit/s[1]. Figure 5.1(a) shows the CCDF of the views per video on a log-log scale. As [30] notes, the straight part over some orders of magnitude indicates a power law popularity. Figure 5.1(b) shows the CCDF of the video length in seconds on a simple log scale; the distribution appears to be light-tailed. Because each client requests a new video as soon as the old one has been watched, the video length distribution is equivalent to the inter-arrival time distribution of the client request arrival process.



(a) Video popularity CCDF.

(b) Video length CCDF.

Figure 5.1: The popularity and length of videos in the Youtube Science & Technology Category.

For the simulation, we assume that there are no bottlenecks, i. e. the DSL lines and DSLAM uplink have sufficient bandwidth (3 Mbit/s and 500 Mbit/s, respectively), and there is no cross traffic on the lines, but only the video traffic. Each DSL line has direct memory access to the cache and an individual queue on the respective line card. The cache has a capacity of 2,500,000 chunks (of 4 KB each), which results in a cache size of approximately 9.5 GB.

During the simulation, all 100 clients continuously download and watch videos. After 6 hours of warm-up time to populate the cache, we start our measurements during a period of additional 10 hours. We repeated the entire experiment 5 times to calculate the 95 % confidence intervals.

[1] As measured by [31] for some types of Youtube content.

5.3 Evaluation Results

During the measurement period, the router could satisfy a mean of 19.63 % of all received Interests with the local cache (±0.64 points for the 95 % confidence interval). Accordingly, out of the 38 Mbit/s of data transferred to the clients, only 30.54 Mbit/s were fetched from the content source.

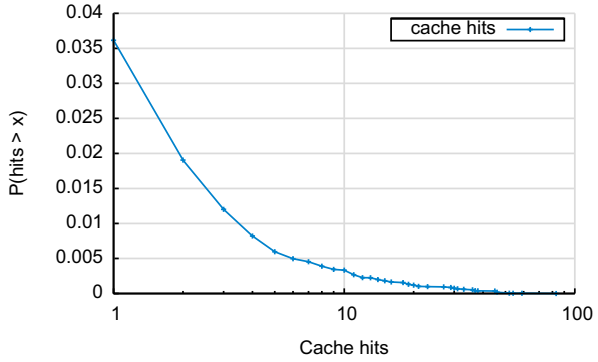


Figure 5.2: CCDF of the number of cache hits per object evicted from the cache.

For all data items evicted from the cache, we studied how many read accesses they experienced after their insertion: 89.39 % (±0.36 points) of all items were evicted without any hit, whereas 10.61 % had at least one hit[2]. 7.08 % (±0.18 points) of all evicted items had exactly one cache hit. Figure 5.2 plots the hit rate CCDF for evicted items of one simulation trace; the other traces are highly similar.

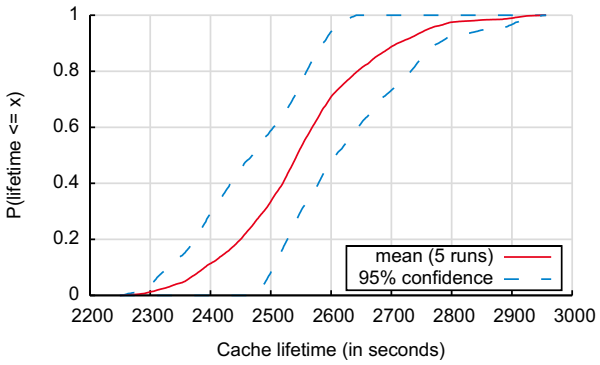


Figure 5.3: CDF of the characteristic time of the cache.

To determine the characteristic time of the cache, we analyse the length of the time span between the last read access and eviction of an item from the cache[3]. Figure 5.3 plots the CDF of this time span averaged over 5 simulation runs as well as the 95 % confidence intervals. Ideally, the curve would have the shape of a straight vertical line at the characteristic time. The actually observed sample differs from this expectation in two ways:

- The curve has an s-shape (the difference between the third and first quartile is 149 s). This means that the characteristic time was not constant over time. The median characteristic time was 2544 s or 42.4 minutes.
- The confidence intervals show that even the CDF curves for (slightly) different workloads differ. For instance, the 95 % confidence interval around the median is ±94 s (3.7 % of the median).

[2] The hit rate per item differs significantly from the cache hit rate computed above because the cache hit rate is expressed in terms of requests (workload in Interests) whereas the hit rate per item is relative to the cache object population. The difference means that a few objects accumulate a high number of hits.

[3] For items with no cache hit, the time span is from insertion to eviction.

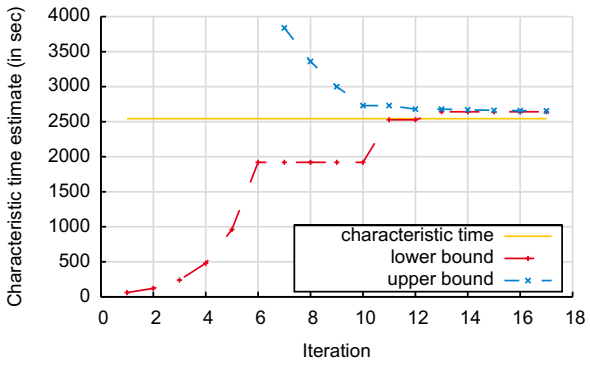


Figure 5.4: Progress of the measurement algorithm for the characteristic time.

We executed our algorithm to measure the characteristic time as defined in Section 4.4.2 to find out how it behaves if the characteristic time is not constant. Figure 5.4 shows the 17 iterations that could be completed during the 10 hours of measurement in one particular simulation run. During the first 6 iterations, the algorithm used a slow-start-like mechanism to find an upper bound on the characteristic time. The initial guess was 60 s, and this value was doubled in each iteration. In round 7, after 2.1 hours of execution time, an upper bound had been found. From there on, the next probes sent by the algorithm were at $\gamma = 75\,\%$ of the currently known interval width. The reason for this behaviour is that in the cache snooping attacks, only the upper bound is of interest.

As Figure 5.4 shows, the algorithm converged to an estimated characteristic time that lies 4.4 % above the median of that simulation run (it also lies outside the 95 % confidence interval). We explain this behaviour with the temporal variation of the characteristic time.

A second observation is that the algorithm took very long to complete: Round 12 completed after a total of 6.1 hours of execution time, round 15 after 8.3 hours, and round 17 after 9.8 hours. This is a consequence of the high characteristic time, and of the algorithm working strictly sequentially.

Following these two observations, the measurement algorithm can be improved in two ways: First, it can be speeded up by parallelising steps, i. e. by inserting many items at the same time and by requesting them in increasing time intervals to obtain a larger range of samples in less time. A close first guess of the characteristic time can also speed up execution. Second, the attacks only need an estimate of an upper bound of the characteristic time. Therefore, the value of γ could be increased even further to favour the upper bound. Furthermore, during the course of the attack, the attacker should constantly monitor if his value for the upper bound is still valid, and increase it if necessary.

5.4 Attack Traffic

A consequence of the high characteristic time is that the attack traffic required by an attacker to constantly monitor all accesses to a given object is quite low. Under ideal conditions, i. e. a constant characteristic time t_c, the attacker would need to probe once every $t_c + \epsilon$ seconds, where $\epsilon > 0$ is to be chosen such that the attacker does not hit on a probe previously inserted by him. The probability of not detecting an access is $1 - \frac{t_c}{t_c+\epsilon}$.

Under real conditions, the situation is more complex. As seen in Figure 5.3, the characteristic time is not constant; it might vary over time. This fact complicates the choice of the probing interval: Assume that the attacker has measured the instantaneous characteristic time t_c at time t_1 and sends probes at time $t_2 \gg t_1$. By probing once every t_c, the attacker might miss items that are evicted from the cache before t_c has elapsed if the actual characteristic time has decreased in the meantime (false negative). If the probing interval is shorter than the characteristic time, an item might still be cached because of a previous probing request sent by the attacker (false positive).

As we have described in Section 4.6.1, an attacker can probe for several chunks in parallel to overcome imprecise measurements of the characteristic time. In the same way, parallel probing can be used to tolerate variations of the characteristic time. For the algorithm to succeed, it is necessary to have an upper bound on the characteristic time, i. e. a delay after which the attacker can be sure that a requested item without any further request will have been evicted from the cache. Furthermore, the attacker needs a number of chunks of the same file that are requested at the same time. Their number depends on the imprecision in measuring the characteristic time (or on its variation).

For the video payload that we assume in this evaluation, the chosen encoding corresponds to 96 chunks per second. For a a low number of consecutive chunks, the chunks' request time will be approximately the same, and the parallel probing attack can be carried out. For instance, if the attacker assumes an upper bound on the characteristic time of 60 minutes and uses the first four chunks of a video for parallel probing, the resulting probing frequency is one request every 15 minutes. From Figure 5.3, it can be seen that no item without subsequent requests remained cached after 60 minutes, and no item was evicted within 15 minutes of its last read access. Assuming that this distribution does not change significantly, both the false negative and the false positive rate will be zero. The attack traffic corresponds to at most $4 \cdot (100\,B + 4096\,B)/1\,h = 37.3\,\text{bit/s}$ if every Interest is answered with a maximum-size chunk. We argue that the attack traffic required by this attack is so low that it is perfectly feasible for an attacker to constantly monitor accesses to a large number of different objects.

5.5 Conclusion & Future Work

In this chapter, we have shown that it is feasible to estimate the characteristic time of a cache. However, if the characteristic time is high, the measurement can take a long time to complete. Future work should aim at decreasing the measurement algorithm's execution time. Furthermore, a mechanism to constantly monitor and update the estimate of the characteristic time is required due to temporal variations of the characteristic time.

Using a scenario of videos on Youtube, we have evaluated how much traffic an attacker needs to constantly monitor and detect if anyone out of 100 active users watches a particular movie: For videos with a minimum length of $\frac{4}{96}$ s, the attack traffic is at most 37.3 bit/s with no false positives and no false negatives.

In future work, we are planning to compare different architectures such as CCN and CDNs. We would like to quantify the privacy loss that arises when caches are placed closer to the end users. For instance, a hit detected in a CDN node could be caused by a request of any user in the whole city, whereas a hit in a DSLAM cache indicates that one out of thousand neighbours requested the data.

6 Conclusion

In this thesis, we analysed the impact of CCN's architecture on network security. We identified several attacks related to caches, and we evaluated how often an attacker needs to probe the cache if he wants to detect every client access to a particular content object. Our work permits us to draw the following conclusions:

1. CCN provides better content security than the current Internet, and denial-of-service attacks against content providers are more difficult to carry out.
2. While CCN improves traditional security concerns of the Internet, its more advanced architecture introduces novel attack opportunities:
   - Routers are more powerful in terms of per-communication state and computation. This characteristic enables new denial-of-service attacks against routers.
   - General-purpose caches can be deployed in every network device. Ubiquitous caching jeopardises users' privacy because their communication leaves behind (ephemeral) traces.
3. The cache snooping attack enables attackers to monitor with very little attack traffic which content objects their neighbours are retrieving (37.3 bit/s in the Youtube-DSLAM scenario).
4. Caching implies a tradeoff between network efficiency and communication privacy.

Countermeasures against cache snooping are challenging to develop because they need to carefully explore this tradeoff: Perfect protection of users' privacy would prohibit caching. As a compromise, we suggested introducing an artificial minimum query delay in access routers. More research is required to determine how effective this countermeasure is, both in terms of efficiency and privacy. We believe that this tradeoff is an important concern in the design of future Internet architectures.

Having reported on the attack opportunity as such, we plan to *quantify* in future work the amount of privacy loss that an architecture such as CCN imposes on its users, and to compare it with other architectures and CDNs.

Glossary

Botnet A network of compromised machines that are controlled by a botmaster and that can be used for coordinated attacks.

Characteristic Time The time between the last access to an item in a LRU cache and its eviction from the cache. See [17].

Content Distribution Network (CDN) A set of machines used to serve content that are located close to the users, for instance in an access provider's network. Used for load balancing, and to speed up content delivery.

Denial-of-Service (DoS) An attack that aims at making the victim unavailable for requests, or to slow responses down considerably. Potential victims include content providers, network links, and routers. A Distributed Denial-of-Service (DDoS) attack leverages the aggregate bandwidth of a number of machines under the attacker's control, for instance a botnet.

Dictionary Attack Images of a function are given, and the function cannot easily be inverted. Precompute a list of (interesting) values and their corresponding images (the dictionary). Compare the observed images with those from the list to find out the preimages.

Forward Interest Base (FIB) Data structure on CCN nodes to make forwarding decisions; similar to a routing table. See Section 2.1 on Page 3.

ISP Internet Service Provider. In this work, mostly used to refer to Internet access providers for end users.

LRU Least Recently Used cache replacement policy. The item that has not been requested for the longest time is evicted from the cache.

Pending Interest Table (PIT) Data structure on CCN nodes to keep track of Interests that have been forwarded in the past. See Section 2.1 on Page 3.

Replay attack An attacker records data sent by an authorised source, and sends this data again at a later point in time. For instance, if a content source replies with "content not available" at time t_0, the attacker might replay this message at time $t_1 \gg t_0$ to make the victim believe that the content is still not available, although this might not be the case any more.

RTT Round-trip time.

Sniffing Listen in on a conversation, for instance capture all traffic on a wireless link.

Snooping Send requests to a cache to find out information about its contents.

Spoofing Provide fake information, for instance a wrong sender address in an IP packet.

Bibliography

[1] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A brief history of the Internet," *SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 22–31, 2009.

[2] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," in *SIGCOMM*, S. Kalyanaraman, V. N. Padmanabhan, K. K. Ramakrishnan, R. Shorey, and G. M. Voelker, Eds. ACM, 2010, pp. 75–86.

[3] Cisco Systems, Inc. (2010, Jun.) Cisco visual networking index: Forecast and methodology, 2009–2014. White Paper. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[4] J. Rexford and C. Dovrolis, "Future Internet architecture: clean-slate versus evolutionary research," *Communications of the ACM*, vol. 53, no. 9, pp. 36–40, 2010.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*. New York, NY, USA: ACM, 2009, pp. 1–12.

[6] Project CCNx. [Online]. Available: http://www.ccnx.org/

[7] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: voice-over content-centric networks," in *ReArch '09: Proceedings of the 2009 workshop on Re-architecting the internet*. New York, NY, USA: ACM, 2009, pp. 1–6.

[8] S. Barber. (2010, Jun.) Dynamic-content oriented applications. CCNx-Dev Mailing List. [Online]. Available: http://www.ccnx.org/pipermail/ccnx-dev/2010-June/000234.html

[9] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *SIGCOMM*, V. Bahl, D. Wetherall, S. Savage, and I. Stoica, Eds. ACM, 2008, pp. 219–230.

[10] A. Anand, V. Sekar, and A. Akella, "SmartRE: an architecture for coordinated network-wide redundancy elimination," in *SIGCOMM*, P. Rodriguez, E. W. Biersack, K. Papagiannaki, and L. Rizzo, Eds. ACM, 2009, pp. 87–98.

[11] M. Dobrescu, N. Egi, K. J. Argyraki, B.-G. Chun, K. R. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: exploiting parallelism to scale software routers," in *SOSP*, J. N. Matthews and T. E. Anderson, Eds. ACM, 2009, pp. 15–28.

[12] U. Lee, I. Rimac, and V. Hilt, "Greening the internet with content-centric networking," in *e-Energy '10: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. New York, NY, USA: ACM, 2010, pp. 179–182.

[13] D. Smetters and V. Jacobson, "Securing network content," PARC, Tech. Rep., Oct. 2009.

[14] CCNx mailing lists. [Online]. Available: http://www.ccnx.org/content/community#mailing-lists

[15] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic, "Pollution attacks and defenses for Internet caching systems," *Computer Networks*, vol. 52, no. 5, pp. 935–956, 2008.

[16] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen, "Internet cache pollution attacks and countermeasures," in *ICNP*. IEEE Computer Society, 2006, pp. 54–64.

[17] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305 – 1314, Sep. 2002.

[18] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2008, pp. 35–49.

[19] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 191–206.

[20] L. Grangeia, "DNS cache snooping," Feb. 2004.

[21] C. E. Wills, M. Mikhailov, and H. Shang, "Inferring relative popularity of Internet applications by actively querying DNS caches," in *Internet Measurement Conference*. ACM, 2003, pp. 78–90.

[22] H. Akcan, T. Suel, and H. Brönnimann, "Geographic web usage estimation by monitoring DNS caches," in *LocWeb*, ser. ACM International Conference Proceeding Series, S. Boll, C. Jones, E. Kansa, P. Kishor, M. Naaman, R. Purves, A. Scharl, and E. Wilde, Eds., vol. 300. ACM, 2008, pp. 85–92.

[23] M. A. Rajab, F. Monrose, A. Terzis, and N. Provos, "Peeking through the cloud: DNS-based estimation and its applications," in *ACNS*, ser. Lecture Notes in Computer Science, S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, Eds., vol. 5037, 2008, pp. 21–38.

[24] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *ACM Conference on Computer and Communications Security*, 2000, pp. 25–32.

[25] S. Krishnan and F. Monrose, "DNS prefetching and its privacy implications: When good things go bad," in *LEET '10: Proceedings of the 3rd Usenix workshop on large-scale exploits and emergent threats*. Usenix, 2010.

[26] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[27] N. Laoutaris, G. Zervas, A. Bestavros, and G. Kollios, "The cache inference problem and its application to content and request routing," in *INFOCOM*. IEEE, 2007, pp. 848–856.

[28] Dégroupage Free. [Online]. Available: http://francois04.free.fr/liste_dslam.php

[29] Youtube science & technology category traces. [Online]. Available: http://an.kaist.ac.kr/traces/IMC2007.html

[30] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. B. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *Internet Measurement Comference*, C. Dovrolis and M. Roughan, Eds. ACM, 2007, pp. 1–14.

[31] P. McFarland. Approximate youtube bitrates. [Online]. Available: http://adterrasperaspera.com/blog/2010/05/24/approximate-youtube-bitrates