PHD THESIS

Savant: An Accounting and Accountability Framework for Information Centric Networks

Author:

Diarmuid COLLINS

Supervisor:

Professor. Donal O'MAHONY

October 8, 2016

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Signed,

____

Diarmuid Collins

Date: October 8, 2016

Abstract

Content Provider i.e. entities that own or are licensed to sell and distribute content e.g., HBO, Netflix, Amazon Prime, which use the IP-based Internet model for content distribution, consume a large percentage of network bandwidth that is expected to almost treble between 2014 and 2019 [Cisco, 2015]. This requires significant infrastructural investment by content providers, content distributors (i.e. fixed-infrastructure content distribution networks (CDNs)) and network operators e.g., AT&T. Consequently, content providers are looking for more efficient, cheaper, secure, scalable and accountable mechanisms for the distribution of content to end-users.

Many existing and proposed future content distribution architectures offer desirable elements that may lead to less bandwidth usage, reduced network congestion, higher content availability and reduced costs e.g., multicast IP, peer-to-peer (P2P), and so forth. For example, the Information Centric Networking (ICN) paradigm offers solutions to many of these challenges by decoupling user trust in content from where it is obtained by enabling the content to *self-verify* i.e. the user can establish integrity, trust and provenance in content received from trusted or untrusted infrastructure. However, these architectures typically span domains of trust that lack central administration. Consequently, it is difficult to gather reliable accounting and accountability information for the content distribution process, which we argue is a fundamental business requirement for many content providers.

Content accounting refers to any information that a content distributor needs to track, relating to the delivery of content to its intended consumers. In contrast, content accountability refers to the willingness of the communicating infrastructure to produce accurate and verifiable information about the content distribution process. The primary difference between an accounting architecture and accountability architecture is that when trust fails the latter has the tools to pinpoint the responsible entity with non-repudiable evidence.

In this thesis, we develop two tools to help identify the drawbacks and merits of existing architectures. The first is a taxonomy for accounting information based on our analysis of logging information gathered from the surveyed systems. The second is a generic model for content distribution based on a synthesis of desirable elements from the surveyed architectures. Utilising these tools, we propose an ICN architecture extension for content accounting and accountability called the Savant framework, which we apply to the Named Data Networking (NDN) architecture. Savant naturally supports efficient content distribution while gathering non-repudiable near real-time information efficiently from NDN clients and NDN caches. This is supported using NDNs natural support for security, integrity and trust and by maintaining hash chains of logs and commitment to log integrity between communicating nodes.

Our proof-of-concept implementation, which is based on an NDN video content distribution session, demonstrated that accounting and accountability information can be gathered for an ICN packet-level architecture. Our analysis also showed that the overhead on the system is very small if several extensions and improvements are applied to the architecture for efficiency.

As described, ICNs/NDN with Savant support could eventually complement or replace today's CDN infrastructure with a scalable, trustworthy, reliable accounting and accountability framework for content distributed in trusted and untrusted environments meeting the diverse requirements of content providers, network operators and end-users.

Acknowledgements

First, I would like to thank my supervisor, Prof. Donal O'Mahony. He has been an excellent advisor throughout the PhD journey giving me the flexibility to explore many research areas and topics while providing me with extensive support, knowledge and guidance. I am also very grateful for the personal support and guidance he has given me over the last four years.

I would also like to acknowledge all the help, encouragement and support I have received from staff and students in CTVR/CONNECT Centre for Future Networks over the last four years. They have helped make this PhD journey a lot easier.

I would also like to acknowledge the help, support and feedback I have received from several other Trinity College Dublin students particularly Elwyn Davies, Emanuele Di Pascale, Paul Duggan and Andriana Ioannou.

I want to thank the anonymous reviewers at various different workshops, conferences and journals for their excellent comments and feedback, which helped define and crystallise my research topic.

I want to thank mam, dad, my siblings and friends for all their encouragement, love and support over the last four years.

Finally, I want to thank my wife Monica and sons Mateo and Miguel for being my big rock and my little rocks. Without your love and support none of this would have been possible.

Contents

List of Figures

List of Tables

Glossary

Accountability Content accountability refers to the willingness of the communicating infrastructure to produce accurate and verifiable information about the content distribution process. 1, 2

Accounting Content accounting refers to any information that a content distributor needs to track relating to the delivery of content to its intended consumers. 2

Content Delivery Network (CDN) A set of centrally managed, geographically dispersed, strategically located cache servers that replicate and deliver popular content close to end-users. 19

Content-Centric Networking (CCN) A future Internet architecture that supports ICNs principles. 49

Future Content Distribution Architectures These include emerging networking approaches such as Information Centric Networking, MobilityFirst, Nebula, and so forth, that address the problems and limitations of the current IP based Internet model for content distribution. 2

Information Centric Networking (ICN) The ICN future Internet paradigm seeks to resolve the problems and limitations of the current IP based Internet model for content distribution by proposing solutions for the future Internet. 1

Internet Service Provider (ISP) Is an entity that provides services and infrastructure to end-users for accessing the Internet, e.g., Eir, Vodafone, AT&T, and BT. 1

Minimal Accounting The minimal accounting scenario supports the collection of industry standard QoE metrics from client and cache infrastructure such as rate of buffering events, frames displayed per second on the client, and average uplink or downlink bitrate. 110

Near Real-Time Near real-time refers to data collected and processed within several seconds of an events occurrence. 3

Real-Time A real-time event is collected and processed within milliseconds of the events occurrence. 12

Self-verify The user can establish integrity, trust and provenance in content received from trusted or untrusted infrastructure. 1

Trusted Infrastructure Any trusted cache element located in the network e.g., a CDN cache server, content provider server, and so forth . 43

Untrusted Infrastructure Any untrusted cache element located in the network, e.g., P2P, ISP cache, ICN cache, and so forth. 43

Chapter 1

Introduction

Internet video traffic from *content providers* i.e. entities that are licensed to sell and distribute content[1] such as Netflix and HBO, has seen exponential growth in the last fifteen years and is predicted to make-up 80%-90% of total Internet traffic by 2019 [Cisco, 2015]. The majority of this content is delivered to end-users by fixed-infrastructure content distribution networks (CDNs) (referred to as *Content Distributors*) such as Akamai and Level3. CDNs are typically located geographically close to end-users and offer high data availability and performance. However, they are expensive to deploy and maintain, generate a lot of over-the-top (OTT) traffic for Internet service providers (Internet Service Provider (ISP)) and many are not close enough to the end-user to provide a quality content delivery experience [Aditya et al., 2012][Ganjam et al., 2015][F. Chen et al., 2015]. These problems are exacerbated by IP's limitations, which include: poor security and Accountability; lack of data integrity, reliability and trust; mobility challenges; bad multihoming support; unnecessary data retransmissions and network traffic; network congestion; lack of user privacy; and so forth.

The Information Centric Networking (Information Centric Networking (ICN)) future Internet paradigm seeks to identify solutions to these kinds of problems and limitations by proposing solutions for the future Internet [Xylomenos et al., 2014]. ICN architectures achieve this goal by treating names or identifiers (such as content, services and devices) as first-class citizens. This is achieved by decoupling a user's trust in content from where it is obtained by enabling the content to *Self-verify* i.e. the user can establish integrity, trust and

[1]We refer to *content* and *data* interchangeably throughout this thesis.

provenance in content received from trusted or untrusted infrastructure. This facilitates more efficient distribution of data in an ICN network as copies of data can be supplied from any infrastructure element that has an available copy. ICN architectures support a paradigm shift towards a data-centric network architecture that focuses on content distribution resolving many of the limitations and problems of the current IP-based Internet model.

However, existing ICN architectures do not provide natural support for content Accounting and Accountability. Instead, many of these architectures make a virtue out of not providing it, claiming to offer natural privacy to users. Content *accounting*[2] refers to the information tracked by content distributors related to the delivery of content to its intended consumers. This includes content views, end-user quality of experience (QoE), user demographics, and so forth. In contrast, content *accountability* refers to a content providers ability to produce accurate and verifiable information about the content distribution process. This involves the ability to establish authentication, integrity, provenance, auditability and non-repudiation in the accounting information received [Yumerefendi & Chase, 2005]. The primary difference between these concepts is that when trust fails, the latter has the tools to pinpoint the responsible entity with non-repudiable evidence. Consequently, accountability is an important concept when the primary form of communication is between nodes that span domains of trust.

Today, centrally managed fixed infrastructure CDNs offer the best option for the production of trustworthy accounting information by guaranteeing the integrity of the information gathered. However, this is not a sufficient model for content that can be distributed cheaply with better efficiency (at least for popular content) from primarily untrusted ICN infrastructure. The aim of the Savant framework is to address this challenge for ICNs, specifically the *named data networking* (NDN) architecture.

1.1 Background and Requirements

In our initial investigations, we studied and compared current and Future Content Distribution Architectures identifying different components utilised for efficient and successful

[2]We refer to *accounting* and *feedback* interchangeably throughout this thesis.

content delivery. Our analysis helped identify the need for a trustworthy accounting and accountability framework when distributing content. Further examination helped us isolate generic elements required for the collection of this information across trusted and untrusted infrastructure. These include:

- A strong identity management system, which cryptographically binds actions to nodes.
- A methodology for producing trustworthy accounting information, which captures client and cache actions and that can be audited by trusted infrastructure components.
- A protocol, which supports *commitment* to log integrity between communicating nodes.
- Distributed infrastructure components to support information gathering in Near Real-Time[3], which is scalable to 10s of millions of nodes.

The main usage scenario for Savant involves a client requesting data and a cache delivering the requested content object in response. This transaction should be supported with the collection of accounting and accountability information for the content distribution process. For example:

- an end-user requests a movie from Netflix,
- the request gets directed to a local ISP cache that has a copy of the requested movie,
- the cache delivers a movie to the end-user,
- both the client and cache produce accounting and accountability information for the content distribution process, which is collected and processed by Savant and presented to the content provider.

[3]Near real-time refers to data collected and processed within several seconds of an incidents occurrence. This is in contrast to a real-time event, which is collected and processed within milliseconds of the events occurrence. In Savant, real-time data is not required because a clients video buffer contains enough content, typically 30 seconds of buffered data, to support recovery if problems are encountered.

1.2 Savant Accounting and Accountability Framework

Savant is designed based on the premise that content providers want visibility over the content being distributed primarily for control and analytics purposes, irrespective of the infrastructure (trusted or untrusted) used to distribute the content. It achieves this goal by pushing primary responsibility for accounting and accountability out to NDN clients and NDN caches (collectively referred to as *NDN agents*) distributing and receiving content. Additionally, it incorporates components and principles from many different systems and architectures to support a near real-time accounting and accountability framework for NDN content distribution. These include [Clarke, 2001][Xylomenos et al., 2014][Aditya et al., 2012][Haeberlen et al., 2007][Cugola & Margara, 2012]:

1. a small-world model of trust supported by the Simple Public Key Infrastructure/Simple Distributed Security Infrastructure (SPKI/SDSI) public key infrastructure (PKI),
2. using ICN cryptographic techniques to produce log entries (referred to as *published log objects*), which provide natural support for data security, integrity, authenticity and provenance,
3. tamper-evident logs are maintained for each individual content provider by NDN agents using an append-only hash chain between log entries,
4. a log commitment protocol to support log integrity and authenticity between NDN agents for interests and data sent and received,
5. big data mechanisms to support the scalable near real-time collection, processing and aggregation of published log objects with capabilities to increase or decrease the amount of accounting and accountability information collected at NDN agents when required.

We implemented many of Savant's components to support a proof-of-concept implementation for use in an NDN video content distribution session. This implementation demonstrated the feasibility of an accounting and accountability framework for an NDN content distribution architecture. Finally, Savant's general principles can be applied to any ICN architecture.

1.3 Key Contributions

- An analysis of existing and proposed future content distribution architectures[4], the components they use for efficient and successful content delivery and the identification of the elements required for the collection of trustworthy accounting and accountability information for content distributed across trusted and untrusted infrastructure.
- A scalable near real-time framework to support the collection, auditing, processing, aggregation and alerting of trustworthy accounting and accountability information produced on trusted and untrusted NDN infrastructure when required.
- Demonstration of the feasibility of the Savant system in a real-world implementation for NDN video content distribution.

1.4 Organisation of this Work

This dissertation is structured as follows. This chapter provides a high-level introduction to some content distribution problems, providing the motivation for our work. We have also identified key contributions from this work for accounting and accountability research in ICN future Internet architectures.

Chapter 2, initially provides an analysis of existing and proposed future content delivery architectures detailing their methodologies for providing efficient low-cost content distribution, accounting and accountability across trustworthy and untrustworthy infrastructure. We also provide background into the desirable architectural elements and principles based on existing content distribution architectures, many of which Savant adopts to support near real-time accounting and accountability for NDN content distribution.

[4]Future Content Distribution Architectures include emerging networking approaches such Information Centric Networking, MobilityFirst, Nebula, and so forth, that address the problems and limitations of the current IP based Internet model for content distribution.

In chapter 3, we develop and use two tools to help analyse the drawbacks and merits of the content distribution architectures discussed in Chapter 2. First, a generic model for content distribution that leads to efficient low-cost distribution of content. Second, a taxonomy for analytic information based on a survey of the logging information gathered by existing systems. Using these tools we developed a table of elements required for efficient content distribution. This helped identify the lack of an adequate accounting and accountability component for ICN architectures.

Chapter 4, describes the design and security model of the Savant accounting and accountability framework in detail, which is supported by the tools from Chapter 3 and desirable architectures and elements from existing systems in Chapter 2. Additionally, we outline Savant's extensions and improvements to support accounting and accountability in a packet-level architecture such as NDN.

Chapter 5, provides a description of our implementation of the Savant system. We also carried out simulations using ndnSIM to evaluate the scalability of the architecture supporting millions of NDN agents. Additionally, we demonstrate Savant's ability to detect, isolate and resolve content distribution problems. Finally, we provide a list of lessons learned during implementation.

Chapter 6, concludes our work on this dissertation, summarising the main contributions and provides direction for future work.

1.5 Publications Arising from this Work

- Ó Coileáin, D. and O'Mahony, D. (2014). Savant: Aggregated Feedback and Accountability Framework for Named Data Networking. In Proceedings of the 17th Royal Irish Academy Research Colloquium on Communications and Radio Science into the 21st Century.

- Ó Coileáin D. and O'Mahony, D. (2014). Savant: A Framework for Supporting Content Accountability in Information Centric Networks. In Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), 2014 10th International

Conference on (p. 188-190). Rhodes, Greece: IEEE.

- Ó Coileáin D. and O'Mahony, D. (2014). SAVANT: Aggregated Feedback and Accountability Framework for Named Data Networking. In Proceedings of the 1st International Conference on Information-Centric Networking (pp. 187–188). Paris, France: ACM. Available from http://doi.acm.org/10.1145/2660129.2660165

- Ó Coileáin D. and O'Mahony, D. (2015). Accounting and Accountability in Content Distribution Architectures: A survey. ACM Computing Surveys (CSUR), 47(4):59:1–59:35.

Chapter 2

Background and Related Work

The Savant framework has two main goals: provide auditable non-repudiable accounting information to content providers for content distributed from trusted or untrusted ICN infrastructure, and to do this in a scalable and efficient manner. Savant consists of several architectural elements which achieve these goals. These are adopted from existing content distribution architectures. Throughout this chapter, we will discuss the background of each of these elements. Moreover, this analysis will help us derive a set of requirements to support content delivery in ICN networks while trying to satisfy the sometimes-conflicting demands of content providers, ISP's and end users.

We begin by defining existing content distribution models in Section 2.1, analysing them in three ways. First, we briefly outline the content distribution methodologies adopted by each architecture, some of the requirements they satisfy and highlight some of their advantages/disadvantages. Second, we show that the trust model adopted by each architecture has an impact on the integrity of the accounting information collected, which we believe has an impact on the adoption of the architecture for data distribution by content providers. Third, these elements form a basis for the development of the generic model for content distribution outlined in Section 3.1 and the taxonomy for accounting information in Section 3.2, which help guide the development of the Savant Framework in Chapter 4.

The rest of the chapter will provide an overview of ICN architectures in Section 2.2, identify name resolution and security mechanisms to support Savant in Section 2.3, state of the art accounting frameworks for IP and ICN systems in Section 2.4 and outline scalable information processing and control systems in Section 2.5.

2.1 Accounting and Accountability in Distribution Systems

All of the content distribution architectures we discuss in this section are designed to support efficient content delivery. For example, some of the early content distribution architectures (such as multicast IP and web caching) were created to help content providers and ISPs reduce bandwidth usage and network congestion while offering higher content availability and reduced costs. These architecture and many others discussed in this section were developed to accommodate a myriad of requirements from content providers, ISPs and end-users operating across trustworthy and untrustworthy infrastructure related to mobility, security, efficiency, cost, trust, privacy, accounting, accountability, and so forth.

The premise of our work states that content providers want visibility over the content being distributed primarily for control and analytic's purposes. Consequently, we have two aims in this section. First, we briefly outline the content distribution methodologies adopted by each architecture, some of the requirements they satisfy and highlight some of their advantages and disadvantages. Our aim is to ascertain what are the desirable characteristics of an efficient and effective content distribution architecture? Second, we identify what accounting information they produce, the trust model used and if there are mechanisms to support accountability. Our aim is also to determine what accounting and accountability information is necessary to support the content distribution process across trusted and untrusted infrastructure? To achieve these goals, at the end of each section we have added a table to help define what attributes these architectures contain. This is achieved by answering the following questions:

- Trustworthy Accounting: Is the accounting information produced by the architecture trustworthy i.e., not open to tampering?
- Span Trust Domains: Does the architecture span domains of trust?
- Decentralised: Is the architecture centralised or decentralised?

- Data Integrity: Is it possible for a malicious entity to pollute the architecture with bad content? Can the receiver establish data integrity?
- Data Provenance: Can we determine the origin or source of the content?
- Scalable: Is the architecture scalable?
- Content Availability: Is content always available?
- Proximity Awareness: How close is the cache to the data receiver?
- Transient Nodes: Do nodes typically stay connected to the network for long periods of time?
- Inexpensive: Is the architecture expensive or inexpensive?
- Searchable: Can the architecture be searched?
- Guaranteed (Content) Control: Does the architecture guarantee control of content?
- Guaranteed Quality: Will content be delivered with high quality?
- Guaranteed Reliability: Does it provide good reliability?
- Guaranteed Performance: Is performance guaranteed?
- Use ISP Cache: Can content be cached and retrieved from an ISP cache?
- Use P2P Cache: Is it possible to get content from a P2P cache?
- Use Multicast: Is multicast supported?

Desirable elements and attributes identified in this section form a basis for the development of the Savant Framework in Chapter 4.

2.1.1 Multicast IP and Web Caching

Accounting and accountability were not a priority in early content distribution architectures such as multicast IP and web caching. Multicast IP facilitates the distribution of IP datagrams to multiple simultaneous receivers in a network, which was pioneered by Stephen Deering at Stanford University in 1988 [Deering, 1988]. In contrast, a web cache improves on the distribution of popular time-shifted content, which was developed independently by both CERN and the Harvest project (which was funded by DARPA), and released in 1994. Both architectures were very effective solutions to the content distribution problem offering bandwidth savings, reduced origin server load, higher content availability and a reduction in network latency for users [Barish & Obraczke, 2000][Deering, 1988]. For example, web caching offered bandwidth savings of between 40%-50% of data transferred from a central server [Wang, 1999]. However, in the late 90's this lack of accounting and accountability was identified as being a significant barrier to their adoption and utilisation [Makofske & Almeroth, 1999][Wang, 1999].

Multicast IP Accounting

Several tools were developed to help gather accounting information in multicast IP environments. This information, which includes QoE, user behaviour information and join/leave times of participants, is gathered using two types of multicast monitoring: passive and active monitoring. Passive monitoring can be achieved by listening for packet information e.g., Real-Time (Control) Protocol (RT(C)P) traffic [Schulzrinne et al., 1996] or NORM positive acknowledgements (NORM - Negative-acknowledgment Oriented Reliable Multicast) [Adamson et al., 2009]. In contrast, active monitoring actively queries nodes in the network, e.g., mtrace (a multicast traceroute tool that traces the reverse path to source from a group member) or the Simple Network Management Protocol (SNMP) (available from routers and other network components on the multicast path in a domain) [Makofske & Almeroth, 1999][Almeroth & Ammar, 1996][Al-Shaer & Tang, 2004]. Tools that utilised some of these technologies to gather accounting information in multicast systems include:

- MListen (developed in 1996) used to monitor the join/leave times and session duration using RTCP of multicast group members on the Multicast Bone (MBone) overlay network [Almeroth & Ammar, 1996];

- MHealth combined the active and passive monitoring of RTCP and mtrace data to provide real-time health and topology of the multicast distribution tree [Makofske & Almeroth, 1999];

However, these tools lacked security, privacy, integrity, provenance, auditability and scalability of accounting information produced, factors that contributed to multicast's lack of adoption [Makofske & Almeroth, 1999].

Web Cache Accounting

Recent improvements to web caching architectures such as the *transparent cache* (see Section 2.1.5) seek to maintain the end-to-end relationship of traditional client-server communication. A transparent cache is an ISP managed distributed network of cache servers located close to end-users that cache and deliver the most popular web content. However, they encounter many of the same challenges as web cache systems related to content accounting and accountability as they span domains of trust [Yumerefendi & Chase, 2005]. These include lack of log integrity and ability to conceal information such as user QoE from content providers. Additionally, content providers have challenges in maintaining control over content being distributed. This results in some content providers encrypting content, which cannot be easily cached resulting in just 22%-34% of bandwidth savings for ISPs [Woo et al., 2013].

2.1.2 Peer-to-Peer (P2P)

Peer-to-peer (P2P) systems, which are self-organising systems designed to share data through direct peer communication with optional support from peer-assisted infrastructure [Androutsellis-Theotokis & Spinellis, 2004], were one of the first architectures to take steps towards providing content accountability [Haeberlen et al., 2007][Yumerefendi & Chase, 2005]. This was primarily born out of necessity. For example, a major challenge in distributing content on P2P networks is the inability to directly observe untrusted peer interactions, which results in parameters like end-user download experience, content served, and so forth going unmeasured [Aditya et al., 2012]. Additionally, many P2P systems are specifically designed to preserve user anonymity [Androutsellis-Theotokis & Spinellis, 2004]. Furthermore, the interests of peers in a P2P network are typically not aligned with each other [Clark & Blumenthal, 2011], so relying on them for efficient content distribution is unwise. For example, several BitTorrent clients exist that use the BitTorrent P2P protocol for selfish downloading by either not contributing to the BitTorrent download process (e.g., BitThief [Locher et al., 2006]) or by gaming the BitTorrent protocol into contributing minimal resources (e.g., BitTyrant [Piatek et al., 2007]). Additionally, as P2P systems span domains of trust, they are also susceptible to Byzantine faults, software bugs, malicious users, and so forth which can be difficult to identify and isolate [Yumerefendi & Chase, 2005].

Consequently, trying to establish trust in peers in a P2P system is important for building a dependable and trustworthy content distribution system. This resulted in early research into trust and accountability in P2P systems taking model designs from centralised reputation systems built around online communities such as the eBay auction site [Dellarocas, 2001]. For example, the EigenTrust algorithm uses a transitive trust value to determine the reputation of a peer. This value is assigned by other peers [Kamvar et al., 2003]. However, reputation systems such as EigenTrust are typically only effective when nodes repeatedly offend [Haeberlen et al., 2007]. Additionally, reputation systems are susceptible to Sybil attacks, where a host forges multiple identities that are used to increase the reputation of specific peers [Douceur, 2002]. As a result, reputation systems are not a good enough solution when there is not a penalty for misbehaviour [Yumerefendi & Chase, 2005], a financial incentive for good behaviour [Sirivianos et al., 2007], or a business requirement for correct analytical information [Aditya et al., 2012]. In these situations undeniable and non-repudiable evidence of a node's actions is required [Yumerefendi & Chase, 2005]. Consequently, there is a requirement to get the untrusted infrastructure to produce evidence of their actions if the integrity of the content distribution process is required.

There are two generations of P2P system *unstructured* and *structured* [Lua et al., 2005].

The first, unstructured P2P, is characterised by loose joining rules, decentralised downloads, decentralised search capabilities and the random placement of data across the P2P network [Lua et al., 2005][Rodrigues & Druschel, 2010]. Content is typically located by random walks or flooding requests across the P2P network with non-specific search terms e.g., Gnutella and BitTorrent [Androutsellis-Theotokis & Spinellis, 2004]. Additionally, many of these systems have a dependence on centralised infrastructure to locate data and other peers (e.g., Napster or SETI@home). In contrast, structured P2P is a particular kind of P2P application, characterised by the use of structured mechanisms such as distributed hash tables (DHT's) to place content (or pointers to content) not at random peers but at specific locations which facilitate fast lookup for future queries [Lua et al., 2005] [Androutsellis-Theotokis & Spinellis, 2004]. Examples include Chord, Pastry and Kademlia [Lua et al., 2005][Maymounkov & Mazières, 2002]. Several implementations of structured P2P system exist such as the Kad Network and Mainline DHT.

In the remainder of this section, we give a brief overview of several unstructured and structured P2P architectures such as Napster, BitTorrent and the Kad network (based on Kademlia DHT) and their methodology for collecting accounting information. We also identify how accountability information can be gathered and utilised by P2P systems based on the PeerReview [Haeberlen et al., 2007] system. Finally, we identify the weaknesses of P2P content distribution.

Unstructured P2P: Napster

Napster, launched in 1999 by Shawn Fanning, is credited with being the first peer-to-peer file-sharing system. It is an unstructured P2P system, which is dependent on centralised infrastructure for file indexing, searching and peer discovery [Androutsellis-Theotokis & Spinellis, 2004]. As Napster is a proprietary system, we looked at OpenNap (a reverse engineered open source Napster server and protocol [Drscholl, 2001]) to understand the accounting information transmitted between Napster servers and peers. The OpenNap server collects configuration information about peers when they log in such as IP address, port and available bandwidth [Drscholl, April 2000]. Peers notify the Napster server when they have files available for sharing. These files are then indexed. When a peer submits a search request for a file, the Napster server responds with a list of peers hosting copies of the file. The Napster client then sorts the results (such as server IP, port, peer bandwidth and file details) and makes them available to the user who then decides from which server to download. Once the file is selected for download, direct peer-to-peer communication takes place. Napster was forced to shut down in 2001 due to its part in facilitating the illegal distribution of copyrighted material following a court case filed by the Recording Industry Association of America (RIAA) [Lua et al., 2005].

Unstructured P2P: BitTorrent

BitTorrent is a simple P2P protocol designed for peer-to-peer file sharing which runs over the HTTP protocol. It depends on a centralised infrastructure component known as a tracker to support file downloads. When a user wants to download a particular file, they search for a *torrent* of the file. A torrent contains information about the file including a cryptographic hash of the data pieces (pieces are fixed file size of 256KB, 512KB or 1MB), tracker URLs and file length [B. Cohen, 2003]. A BitTorrent client then contacts the tracker or trackers using the tracker URLs. A tracker's primary responsibility is to help peers with complete and partial copies of a file to locate each other. Consequently, trackers gather information from peers such as file location and total bytes uploaded and downloaded. After an initial list of peers has been received from a tracker (i.e. the swarm - a list of participating peers that download and upload content to each other), peers can operate without further contact.

Structured P2P: The Kad Network

The Kad network is a structured P2P overlay implementation of the Kademlia distributed hash table (DHT), designed in 2002 by Petar Maymounkov and David Mazières. DHT's (e.g. Kademlia, Pastry, Tapestry, and so forth) use a hash function (e.g. SHA1, and so forth) to generate a globally unique key for both host machines and data objects. The key for a data object is then stored on the host with the closest corresponding key. This method facilitates efficient lookup for a key typically in $O(logN)$ hops (where N corresponds to the number of hosts in the system) and guarantees results. Hosts use an API implementation to access the DHT to perform operations for generating, adding, updating, locating (routing) and removing keys in the overlay DHT system. The DHTs differ in their approach to these operations.

Kademlia, for example, identifies hosts and data objects using a 160-bit key. It determines the numerically closest host to a data object key using the XOR metric, which defines the distance between two points in a key space as their bitwise exclusive or (XOR), interpreted as $d(x,y) = x \oplus y$ [Maymounkov & Mazières, 2002]. When a Kademlia host sends a message to another host, it also transmits its <IPAddress, UDP port, hostID>, which can be added to a list called a *k-bucket* on the receiving host. The most popular client that supports the Kad network and uses an implementation of the Kademlia protocol is eMule, an open source peer-to-peer file sharing application, in active development since 2002.

Accounting, Accountability and Efficiency: P2P Systems

Napster, BitTorrent and the Kad network gather accounting information related to server configuration and performance, network performance, user engagement, user demographics and user QoE. Every P2P application has a specific use for information gathered. For example, several studies exist (such as Napigator and Xnap [Whitman & Lawrence, 2002][Mauri et al., 2004]) that use the OpenNap protocol to measure file popularity, music artist demand, and numbers of users. Similarly, BitTorrent trackers can derive a peer sharing ratio (calculated by dividing the total data uploaded by the total data downloaded to penalise freeriding. Comparably, Kademlia k-buckets keep track of node uptime using a least-recently seen eviction policy and low-latency paths. However, the accounting information collected by these applications lacks any form of trust or accountability, a general characteristic of early P2P systems. PeerReview [Haeberlen et al., 2007], a system for providing accountability in distributed systems that span domains of trust, offers a solution to this problem.

PeerReview [Haeberlen et al., 2007] maintains a secure tamper-evident log on each node for all application messages sent and received in chronological order. Peers are assigned a public key π_i and private key σ_i pair bound to a unique node identifier, which is used to

Table 2.1: Attributes of PeerReview: An Accountable P2P Architecture

Attribute	PeerReview
Accountability	x
Public/Private Key	x
Tamper-Evident Log	x
Log Consistency	x
Trusted Auditing	-
Non-Deterministic	-
Challenge/Response Protocol	x
Rapid Fault Detection	-
Defend Accusations	x
Statistical Sample	x
Centralised	-

digitally sign each message sent. Each node n has log entry $e_i = (s_i, t_i, c_i, h_i)$ with a sequence number s_i, a type t_i and type-specific content c_i [Haeberlen et al., 2007]. Each log record also has a recursively defined hash h_i, which creates the hash chain making the log tamper evident. An authenticator $\alpha_i^n = \sigma_n(s_i, h_i)$, which is sent as part of the communication process to node $n2$, is a signed statement that node n has updated its log entry e_i with hash value h_i. To ensure commitment, the receiver node $n2$ must acknowledge the message received. Furthermore, a single linear log, which contains all communication between peers, must be maintained by each node to ensure consistency. This is verified by a set of independent *witnesses* gathering all authenticators sent by node n. Each node has a number of witnesses who check its correctness and distribute the results to other nodes in the system. A witness can request node n to return all log entries within a certain range. It then audits node n's actions to ensure they conform to a reference implementation of expected behaviour based on a deterministic state machine. Moreover, PeerReview's challenge/response and evidence protocols help ensure that all correct nodes eventually receive evidence of faulty nodes. Additionally, using PeerReview, correct nodes are always able to defend themselves against false accusations. Finally, while the PeerReview system provides irrefutable evidence of a node's actions using hash chains of logs, it does not scale well for large systems (hundreds of nodes) if every fault is to be detected [Haeberlen et al., 2007]. However, using probabilistic guarantees PeerReview can dramatically increase scalability at the expense of relaxed completeness guarantees. A summary of PeerReview attributes is available in Table 2.1.

Table 2.2: Attributes of P2P Architectures: Napster, BitTorrent and Kad

Attribute	Napster	BitTorrent	Kad Network
Trustworthy Accounting	-	-	-
Span Trust Domains	x	x	x
Decentralised	-	-	x
Data Integrity	-	x	x
Data Provenance	-	-	-
Scalable	x	x	-
Content Availability	x	x	-
Proximity Awareness	-	-	-
Transient Nodes	x	x	x
Inexpensive	x	x	x
Searchable	x	x	x
Guaranteed (Content) Control	-	-	-
Guaranteed Quality	-	-	-
Guaranteed Reliability	-	-	-
Guaranteed Performance	-	-	-
Use ISP Cache	-	-	-
Use P2P Cache	x	x	x
Use Multicast	-	-	-

While systems like PeerReview propose solutions for accountability in distributed systems, P2P systems still have other challenges to overcome. A summary of attributes of P2P architecture covered in this section is available in Table 2.2. For example, many unstructured P2P systems e.g., BitTorrent, have difficulty locating less popular content while some search mechanisms for structured P2P e.g., the Kad Network, can have implications for data scalability, availability and persistence [Androutsellis-Theotokis & Spinellis, 2004]. Moreover, many P2P systems are susceptible to having geographically dispersed neighbours in addition to transient node populations joining and leaving the network. CDNs offer solutions to some of these challenges however.

2.1.3 Content Delivery Networks (CDNs)

Accounting information is more prevalent in Content Delivery Network (CDN) systems, which is a set of centrally managed, geographically dispersed, strategically located cache servers that replicate and deliver popular content close to end-users. CDNs started to appear in 1998 when companies realised the benefits they offered in terms of content control, delivery quality, reliability, scalability, performance and accounting [Pallis & Vakali, 2006].

Since then, they have become one of the most important advances in content distribution on the Internet with Akamai alone claiming to deliver between 15%-30% of the worlds total web traffic [F. Chen et al., 2015]. The main components of the CDN architecture include: the content origin server, surrogate/edge servers, software for content replication, request routing infrastructure (RRI) and logging (data collection and analysis). In the remainder of this section, we give a brief introduction to these components (followed by examples) as they form a key element in understanding the requirements of content providers. Moreover, they have a direct impact on the design of the Savant framework.

Main Components

A CDN is a tree architecture with the content origin server acting as the root of the tree. Data is pushed to surrogate/edge servers, which are located geographically close to end-users. Determining how many copies of content to replicate and to which servers - known as content outsourcing - is a major challenge. There are three content outsourcing practices that CDN systems commonly use. These include [Pallis & Vakali, 2006]: cooperative push-based outsourcing, non-cooperative pull-based outsourcing and cooperative pull-based outsourcing. In cooperative push-based outsourcing the content distributor pushes content to the CDN surrogate servers (or edge servers) that cooperate with each other to proactively replicate content before client requests are handled [Pathan & Buyya, 2008]. However, this is a theoretical approach not used much in practice [Pathan & Buyya, 2008][Pallis & Vakali, 2006][Passarella, 2012]. Traditionally, CDN systems have preferred to use the non-cooperative pull-based outsourcing approach, which redirects user requests using DNS redirection or URL rewriting. Cooperative pull-based outsourcing is reactive, only caching content once it is requested, but cooperates with other caches if a cache miss occurs [Pallis & Vakali, 2006]. The RRI component transparently redirects a user's DNS request based on using a set of metrics such as content location, latency to server, proximity to server and surrogate server load. The final, and arguably one of the most important components in the CDN architecture, is logging [Aditya et al., 2012]. CDN customers depend on logging information for analytics to help determine what content is popular, user demographics, user behaviour, and so forth. Additionally, content providers want to ensure CDN systems are meeting service level agreements (SLA's), as poor end-user QoE can have repercussions such as lower revenues and lower user subscriptions. CDN systems also depend on logging information to help troubleshoot and debug system and network faults, resolve security vulnerabilities and bill customers [Repantis et al., 2010].

We have selected the following three CDN systems: Akamai, CoralCDN and YouTube to illustrate how each system utilises these core CDN components and outline how accounting and trust is provided to content providers.

Akamai

Akamai, founded in 1998 by a group at MIT as a solution to the flash crowd problem, is now the biggest content delivery platform in the world with over 170,000 servers located in 102 countries [F. Chen et al., 2015]. Its architecture consists of a virtual overlay network of edge servers comprising of the origin server, edge servers, mapping system (RRI), content outsourcing (transport) system, logging infrastructure, control software and a management portal [Nygren et al., 2010]. Most of these components (edge servers, mapping system, content outsourcing system and logging) act in accordance with full or partial content delivery based on the non-cooperative pull-based outsourcing technique [Passarella, 2012].

Content replication information from the content outsourcing system acts as input for RRI metrics. Additionally, edge server logs are mined for information about the server and network performance, debugging and troubleshooting network faults, raising alarms and billing customers. Akamai uses these logs to develop historical metrics based on server performance and global network conditions for use in their RRI. Furthermore, Akamai's management portal allows customers to view analytics related to end-user demographics, user behaviour, network QoE, and so forth [Nygren et al., 2010]. However, the amount of logging information gathered can consume massive resources such as in Akamai's *Query* system [Repantis et al., 2010] (see Section 2.5.2). For example, in 2010 Akamai processed over 100TB of logs per day collected from edge server's [Nygren et al., 2010].

In contrast to Akamai's commercial use of the non-cooperative pull-based outsourcing technique, CoralCDN uses the cooperative pull-based outsourcing technique, which is supported by a distributed hash table (DHT) [Pathan & Buyya, 2008][Passarella, 2012].

CoralCDN

CoralCDN, is an open source academic CDN based on P2P technologies, which has been publicly available across the PlanetLab test bed since 2004. In 2011, it was serving between 25-50 million requests per day [CoralCDN, 2011]. The system was originally designed so that participating peers could contribute resources to the CoralCDN network. However, due to security implications (such as data integrity and susceptibility to pollution attacks) it continues to be run on a centrally administered trusted test bed of primarily PlanetLab server's [Freedman, 2010].

The architecture itself is based on three major components: an indexing layer called Coral, HTTP proxies and DNS servers. The Coral indexing layer offers abstract improvements to existing DHT systems by helping locate nearby copies of data and avoiding server hotspots. Coral achieves this by allowing lists of pointers (i.e. a pointer to a host with a copy of the data) to be stored with the same key across multiple hosts. These lists offer weaker consistency because Coral stores lists of pointers to many nodes that host data in contrast to traditional DHT systems. A traditional DHT has only one value stored per key at each host. This technique is known as distributed sloppy hash table (DSHT) by the authors [Freedman & Mazières, 2003]. Furthermore, locality awareness is improved across the CDN architecture by introducing hierarchical levels of DSHT called clusters, which have a maximum desired round trip time (RTT) called a diameter [Freedman et al., 2004]. CoralCDN, in its current implementation, uses a three-level cluster hierarchy: regional (30ms), continental (100ms) and global (∞) [Freedman et al., 2004] and each host becomes a member of each cluster.

The most appealing aspect of CoralCDN's design (which we have incorporated into the generic model in Section 3.1) is its support for P2P content distribution. However, the inability to distribute content from untrusted infrastructure (due to threats to pollution, data integrity, and so forth) limits CoralCDNs ability to utilise this resource. Furthermore, pos-

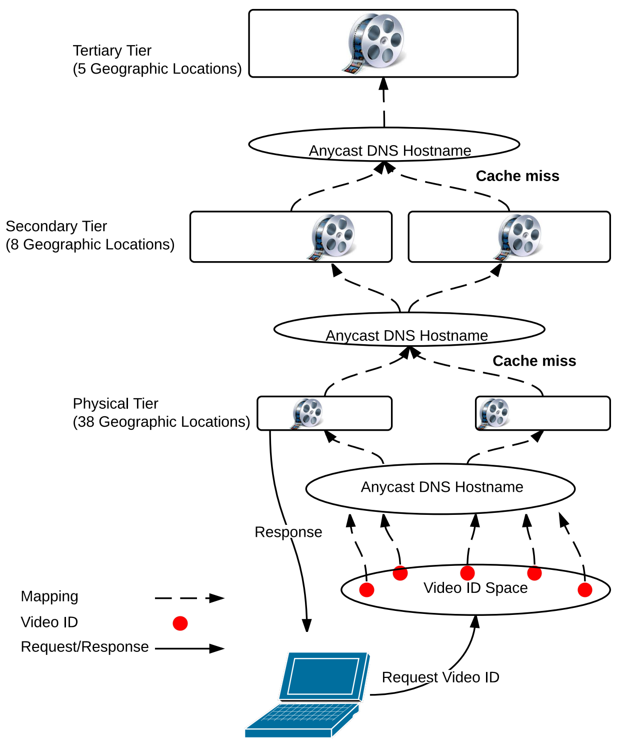


Figure 2.1: YouTube Architecture[adapted from [Adhikari et al., 2012]]

sibly due to CoralCDN being an academic CDN, it does not offer access to server logging for analytics and reporting purposes. However, CoralCDN allows the origin host to track the originator of the content request (i.e. does not interfere with absolute URL's) to which CoralCDN delivers the content.

In contrast to CoralCDN's use of the cooperative pull-based outsourcing technique, YouTube uses the non-cooperative pull-based outsourcing technique, which is supported by a distributed network of trustworthy hierarchical cache servers [Adhikari et al., 2012].

YouTube

YouTube, founded in 2005 by a group of ex-PayPal employees and acquired by Google in 2006, is a video sharing website that allows users to play videos on demand. There is no public information available from Google about the YouTube CDN architecture, however it was reverse engineered in a 2012 paper (architecture depicted in Figure 2.1) [Adhikari et al., 2012]. According to [Adhikari et al., 2012], the YouTube CDN is based on three major components: a flat video id space, multiple layers of anycast DNS hostnames that map to geographically dispersed servers in a 3-tier cache hierarchy.

The YouTube 3-tier cache hierarchy is comprised of a physical tier (in 38 geographic locations), a secondary tier (in eight geographic locations) and a tertiary tier (in five geographic locations) [Adhikari et al., 2012]. YouTube uses a flat 11 character string literal identifier to uniquely identify videos e.g. 'gqGEMQveoqg'. The total collection of video ids is referred to as the video id space [Adhikari et al., 2012]. YouTube uses a fixed mapping technique (using consistent hashing) to map video ids in the video id space to an anycast hostname at the physical tier that has responsibility for serving the video e.g., v1.lscache1.c.youtube.com [Adhikari et al., 2012]. Using the anycast hostname and multiple rounds of DNS resolutions and HTTP redirections it is possible to locate a surrogate server geographically close to the end-user that is responsible for the video delivery. Content distributed over the YouTube infrastructure closely follows the Pareto Principle (80-20 rule) where the top most popular 10% of viewable items is viewed 80% of the time [Cha et al., 2007]. Popular video's (i.e. videos with more than 2 million views) have a much higher probability of being served by the physical tier than unpopular videos, which need to be retrieved from secondary or tertiary tiers [Adhikari et al., 2012].

YouTube provides extensive analytics to all content providers for advertising, user engagement and user demographics. However, because Google controls the entire CDN infrastructure used to distribute YouTube content, no content distribution metrics are made available for user QoE or network performance to content providers.

Accounting, Accountability and Efficiency: Content Distribution Networks

Currently, centrally managed CDN infrastructures offer the best option for the production of trustworthy accounting information by guaranteeing the integrity of the information gathered. However, this CDN model for accounting is based on trust. Content providers need to trust the CDN will deliver content with adequate quality, speed and reliability, which is verified through the provision of accounting information. Moreover, content providers need to trust the accounting information CDNs produce because they lack non-repudiable evidence of actions taken i.e. accountability. However, CDN vendors depend on their reputations for providing these services to keep existing customers and attract new ones [Clark & Blumen-

Table 2.3: Attributes of Repeat and Compare: An Accountable CDN Architecture

Attribute	Repeat and Compare
Accountability	x
Public/Private Key	x
Tamper-Evident Log	x
Log Consistency	x
Trusted Auditing	x
Non-Deterministic	-
Challenge/Response Protocol	-
Rapid Fault Detection	-
Defend Accusations	x
Statistical Sample	x
Centralised	x

thal, 2011].

Moreover, the business model used by a CDN also determines the amount of information it will share with content providers. For example, Akamai allows content providers to view information such as user engagement, user QoE, advertising, user demographics and network performance metrics [Nygren et al., 2010]. Furthermore, the Akamai *Query* system gathers extensive information about distributed surrogate servers to detect system and network anomalies, monitor system usage, raise alarms and troubleshoot problems, which is available to Akamai staff and indirectly to Akamai customers [Repantis et al., 2010]. In contrast, YouTube only provide accounting information on: advertising, user engagement and user demographics, while CoralCDN provides no accounting information at all to content providers.

Lack of accountability i.e. the ability to establish integrity, authenticity, provenance, trust and audit accounting information received, is also an issue for CDNs. However, *Repeat and Compare* proposes a framework for providing accountability in untrusted peer-to-peer CDN environments for static and dynamically generated content [Michalakis et al., 2007]. This is supported by *verifiers*, which are randomly selected replica servers that deterministically *repeat* the content generation process and *compare* the results with *attestation records* received from the client. *Attestation records* provide undeniable evidence of replica server behaviour by cryptographically binding them to content generated. Moreover, they bind replica servers to code and external inputs used to deliver static and dynamic content. As

Table 2.4: Attributes of CDN Architectures: Akamai, CoralCDN and YouTube

Attribute	Amakai	CoralCDN	YouTube
Trustworthy Accounting	x	-	x
Span Trust Domains	-	-	-
Decentralised	-	-	-
Data Integrity	x	x	x
Data Provenance	x	x	x
Scalable	x	x	x
Content Availability	x	x	x
Proximity Awareness	x	x	x
Transient Nodes	-	-	-
Inexpensive	-	x	-
Searchable	x	x	x
Guaranteed (Content) Control	x	-	x
Guaranteed Quality	x	-	x
Guaranteed Reliability	x	-	x
Guaranteed Performance	x	-	x
Use ISP Cache	-	-	-
Use P2P Cache	-	-	-
Use Multicast	-	-	-

a result, an attestation record contains inputs e.g., client requests and original content, and execution environment information e.g., API library and configuration parameters used. Attestation records are included with all client requests and server responses and form a chain of accountability from the client to the server. Each client and replica server has a public key π_i and private key σ_i pair generated by a trusted certification authority, which is used to digitally sign each attestation record sent. Secure authentication is fundamental to the *Repeat and Compare* system and if private keys are compromised a client or replica may be held falsely accountable for illicit actions taken. Furthermore, a list of suspected replica servers is distributed by a centralised set of trusted verifiers or using a decentralised trust model based on off-line trust relationships between clients and replica servers. *Repeat and Compare* offers eventual detection guarantees as clients forward sample attestation records to verifiers, which supports scalability. A summary list of Repeat and Compare attributes is available in Table 2.3.

CDNs offer content distributors a centralised element that supports data control, integrity, provenance, availability, scalability and gives trustworthy accounting information for content distributed. In most cases, such as large CDNs like Akamai and YouTube, they almost always guarantee quality, reliability and performance for content delivered. A summary list of the attributes of CDN architectures covered in this section is available in Table 2.4. However, there are also several disadvantages to using CDNs in comparison to other content distribution architectures such as web caching, P2P, and so forth. For example, they are expensive to deploy and maintain, generate a lot of accounting information, and many CDNs are not close enough to the end-user to provide a quality content delivery experience [Bitar et al., 2012][Frank et al., 2013]. Some solutions to these challenges already exist however. For example, multi-CDN optimisers such as C3 provided by Conviva (see Section 2.5.4) use primitive QoE metrics in order to redirect clients to the best available CDN server if QoE problems are being encountered [Ganjam et al., 2015]. Similarly, some CDN systems based on the non-cooperative pull-based outsourcing technique (e.g., Akamai and YouTube) can provide at least some content delivery over the last mile due to peering agreements and cache servers located inside ISP networks (see Network CDNs in Section 2.1.6). Moreover, hybrid CDN-P2P systems offer solutions to some of these challenges.

2.1.4 Hybrid CDN-P2P

In a hybrid CDN-P2P architecture, the CDN edge server can use the resources of P2P peers such as CPU, memory, storage and bandwidth [Yin et al., 2009] while acting as a regular server to clients not participating in P2P distribution, acting as a tracker (a centralised server that supports file downloads) for new P2P peers joining the network and acting as a seed (a peer who has a complete copy of a file) for P2P peers already on the P2P network [Yin et al., 2010]. Research has shown that using the resources of peers in this way can reduce the load on edge servers by up to two-thirds [C. Huang et al., 2008]. For example, PPLive - a P2P *video on demand* (VoD) streaming system popular in China - reports serving 1.5 million users each with a 400 kbps stream using less than 10 Mbps server bandwidth [Hei et al., 2007]. Similarly, Akamai NetSession (Akamai's hybrid CDN-P2P system) reports offloading 70%-80% of traffic to peers from CDN infrastructure without any impact on QoE [Zhao et al., 2013].

There are two types of P2P streaming models used by CDN-P2P: tree-based and mesh-based. In tree-based P2P, streaming peers are organised into groups that construct an overlay spanning tree for efficient data distribution, which is typically self-organising and self-improving [Chu et al., 2002]. The tree-based streaming model delivers data to peers with low delay and there is a reduction in packets travelling across the network compared to unicast or mesh based P2P. Examples of tree-based CDN-P2P system include: PeerCast, End System Multicast (ESM) [ESM, 2007] and SplitStream [Castro et al., 2003].

Mesh-based P2P streaming inspired by BitTorrent's file swarming process, receives streams from multiple parent peers by requesting (pulling) advertised (available) chunks of data. It can also send streams to multiple child peers [Magharei et al., 2007]. Mesh-based P2P is robust to peers joining and leaving with fair uploading but needs to maintain a large buffer cache in case of late arrival of video segments [Yin et al., 2010]. Examples of mesh based CDN-P2P system include PPLive [Y. Huang et al., 2008], AntFarm [R. S. Peterson & Sirer, 2009] and Akamai NetSession [Zhao et al., 2013]. A hybrid streaming approach combines the push tree-based model with the pull mesh-based model, which leads to efficient delivery with robustness to churn [Yin et al., 2009]. LiveSky [Yin et al., 2010] is example of the hybrid approach.

Four hybrid CDN-P2P streaming systems: ESM [ESM, 2007], SplitStream [Castro et al., 2003], PPLive [Y. Huang et al., 2008] and LiveSky [Yin et al., 2010], which demonstrate tree-based, mesh-based and the hybrid streaming approach to content delivery are discussed in the remainder of this section. Moreover, we briefly discuss SAAR [Nandi et al., 2007], a control overlay for cooperative endsystem multicast (CEM) systems. Finally, we identify what accounting information is collected to support the content distribution process.

End System Multicast (ESM) and SplitStream

The ESM research project, started in 1999 at Carnegie Mellon University [ESM, 2007], pioneered research into real-time P2P overlay multicast streaming. ESM is not currently implemented in any production environments, however some of the ESM team went on to found Conviva (which develops tools to support online video broadcast) in 2006 [ESM, 2007]. ESM, which is modelled on multicast IP, uses end-hosts to provide support for packet replication and group membership instead of using dedicated IP infrastructure [Chu et al., 2002]. Research has shown that ESM can provide a 14-fold reduction of packets travelling over the same link when compared to unicast traffic while using about twice as much bandwidth as multicast IP [Chu et al., 2002].

New end nodes join an ESM broadcast by contacting the broadcast source, which replies with a random list of nodes currently participating in the overlay communication. Each node maintains a partial list of nodes in the network for future parent selection, and periodically learns about new members using a gossip protocol [J. Liu et al., 2008]. This list includes the path (parents, grandparents, and so forth) back to data source, which is used to prevent routing loops by enabling nodes to notify parents that they are descendants during probing. A host joining the group (say host "B"), runs a parent selection algorithm that probes hosts B knows about. These hosts reply with their current throughput and delay from source, whether they are saturated, whether they are a descendant of B and based on the response, B is able to determine the RTT to the source [Chu et al., 2004]. The new host B then selects a parent with the best throughput and least delay. The parent selection algorithm helps construct an overlay tree (tree-based P2P) rooted at the source that is optimised primarily for bandwidth and secondarily for delay, which is self-organising, incrementally self-improving (as new network information becomes available) and adaptive to change [Chu et al., 2002].

However, ESM and similar tree-based P2P streaming systems use a single multicast tree when distributing content [Castro et al., 2003]. Moreover, they depend on a small number of peers for content distribution and can perform poorly when peer parents leave the system due to packet loss. However, the SplitStream system improves on these problems by facilitating the creation of efficient and scalable multi-trees (multicast trees) that reduce the bandwidth demands on individual peers. Multi-trees split the content into independent disjoint stripes of equal size and balance the forwarding load between participating peers [Castro et al., 2003]. Stripes are content encoded using techniques such as Multiple Description Coding (MDC), which splits content into multiple prioritised descriptions. Content can be reconstructed from any subset of stripes received at participating peers. The more stripes received the more video quality improves. SplitStream is implemented using the Pastry DHT [Lua et al., 2005] and Scribe [Castro et al., 2002]. Pastry is a structured self-organising peer-to-peer overlay network based on distributed hash tables with proximity awareness [Lua et al., 2005]. Scribe is a decentralised application-level multicast system built upon Pastry with resource discovery that can support large groups and an arbitrary number of members with highly dynamic membership [Castro et al., 2002]. This combination of technologies enables the SplitStream algorithm to construct multi-tree forests in a decentralised, scalable, efficient and self-organising environment facilitating increased fault tolerance, path diversity and forwarding load with low delay and link stress in comparison to ESM.

PPLive

The PPLive system is a high quality P2P protocol and application for streaming live video and VoD - developed in 2004 at Huazhong University of Science and Technology in China. It was commercialised in 2005 into a popular P2P based IPTV application. When a peer joins the PPLive network, it interacts with several main centralised architectural components using a gossip protocol (over UDP), which include: a bootstrap server, a content index server, a tracker server (to find local peers) and a logging server [Y. Huang et al., 2008]. Once a peer finds local peers sharing the requested content, it can also begin participating in file sharing using the mesh-based P2P streaming model. PPLive uses its TV Engine buffer to reassemble received video chunks, distribute chunks to other peers, reduce download rate variations in the P2P network and feed the media buffer for video playback. To compensate for the distribution of time-shifted content in the PPLive VoD system, peers contribute additional disk storage to the TV Engine buffer (e.g., 1GB [Y. Huang et al., 2008]). PPLive initially used TCP for streaming, but changed to UDP in 2007 [Y. Liu et al., 2009].

The PPLive system collects metrics that enable the content provider to determine the health of the PPLive system. In AntFarm, which is also a mesh based P2P system, a centralised coordinator collects similar metrics and utilises them to optimise bandwidth allocation to reduce average download latencies across all peers [R. S. Peterson & Sirer, 2009]. Furthermore, SwarmServer [Sharma et al., 2014] analyses the benefits of coordinator (or controller) performance in mesh based P2P protocols by comparing similar metrics gathered across a broad range of client-assisted content delivery systems. The LiveSky and SAAR systems gathers similar accounting information but in a hybrid streaming environment.

LiveSky and SAAR

LiveSky is a commercially deployed CDN-P2P live streaming system developed by ChinaCache. The LiveSky CDN-P2P architecture is comprised of [Yin et al., 2010]: a management centre; *service nodes* that distribute content to end hosts; end hosts that are either peer-to-peer nodes or legacy client-server nodes. LiveSky utilizes a hybrid tree-based and mesh-based streaming service over UDP for P2P content distribution [Yin et al., 2010]. Service nodes and peer nodes both aggregate and submit summarised logs regularly to a log server in the management centre. Information gathered includes [Yin et al., 2009]: users joining and leaving the system, bytes uploaded and downloaded, list of peer interaction, playback quality and buffer fill-up time.

SAAR [Nandi et al., 2007], a control overlay for cooperative endsystem multicast (CEM) systems, gathers similar metrics to LiveSky but separates data dissemination and control into separate overlay networks. Consequently, the control overlay can be shared among multiple data plane structures (e.g., tree, multi-tree, mesh-based), each distributing different content or channels (e.g., IPTV). Participating peers in each data overlay or *group* maintain a set of *state variables* and proactively send updates to the control overlay. Updates include: peer forwarding capacity, current load, stream loss rate, tree-depth, and so forth. Group variables are aggregated by the control plane, which is managed by a spanning-tree rooted at a random member of the control overlay. Utilising aggregate system state information, SAAR can support efficient peer selection using an anycast primitive, which takes a constraint and an objective function as arguments. For example, a constraint might include all nodes with spare forwarding capacity, while the objective function might seek to limit distance from parent nodes. Using this methodology, SAARs offers improved content delivery, reduced channel-switching time and reduced peer overhead while supporting multiple different data plane structures for content dissemination.

Accounting, Accountability and Efficiency: Hybrid CDN-P2P Systems

Based on our analysis, hybrid CDN-P2P systems typically gather accounting information on server configuration and performance, network performance, user engagement and user QoE [Yin et al., 2009][Y. Huang et al., 2008]. This information can tell a content publisher about each participating host, the current overlay tree, the bandwidth of each overlay link, current group membership and host performance [Chu et al., 2004]. Information gathered can also be used to calculate additional metrics. For example, rebuffering rate i.e. the number of clients that go into a buffering state per minute, start-up delay, aggregate quality indices, and so forth [Yin et al., 2009]. Consequently, this information enables the CDN-P2P systems to select the best service node for each peer, facilitates localised P2P interaction, keeps CDN costs to a minimum while ensuring a good QoE for end-users.

To our knowledge none of these systems utilise accountability when collecting accounting information. However, the Reliable Client Accounting (RCA) system [Aditya et al., 2012] proposes an accountability framework for hybrid CDN-P2P systems adopting similar principles to PeerReview [Haeberlen et al., 2007] by maintaining a hash chain of logs between communicating peers (e.g., log entry $e_j = (h_j, s_j, t_j, c_j)$). However, it differs in several fundamental ways. First, the CDN infrastructure can audit logs in order to check for integrity, consistency and plausibility i.e. modelled as an abstract state machine. This is in contrast to PeerReview, which uses witnesses or peer infrastructure to audit logs based on deterministic state machine. Furthermore, RCA's focus is on reliable client accounting rather than fault detection. Additionally, trusted RCA infrastructure use mechanisms to control client paring, quarantining problematic nodes (using statistical anomaly detection) and issue public key π_i and private key σ_i pairs to nodes. Moreover, RCA enforces resource limits with the intention of constraining/minimising the impact of different types of attack (e.g., Sybil). This information can also be used during auditing to verify clients are performing as expected. Additionally, RCA's tamper-evident logs are designed with the intention of reducing processing overhead on CDN infrastructure. Consequently, each client maintains a hash sub chain along with one pair of authenticators $\alpha_j^i = (s_j, h_j, \sigma_i(s_j || h_j))$ for each peer it communicates with. This represents a commitment of log integrity to the latest log entry

Table 2.5: Attributes of RCA: An Accountable Hybrid CDN-P2P Architecture

Attribute	RCA
Accountability	x
Public/Private Key	x
Tamper-Evident Log	x
Log Consistency	x
Trusted Auditing	x
Non-Deterministic	x
Challenge/Response Protocol	x
Rapid Fault Detection	-
Defend Accusations	x
Statistical Sample	x
Centralised	x

and hash chain state (signed by i's private key σ_i). All RCA authenticators are cumulative as each message or acknowledgement can be used to verify all pervious log entries. This is in contrast to PeerReview, which maintains a single linear log for all communication-taking place between peers.

Enforcing resource limits in RCA is achieved by issuing resource certificates to clients joining the network based on physical attributes. Attributes bound to clients include: public key π_i, client GUID G_i, maximum throughput C_i (assigned based on a bandwidth test), IP address A_i, and expiration time T_i i.e. several hours. The issued certificate is signed by the RCA CDN infrastructures private key σ_P and are reissued when they expire or when the client IP address changes. As a result, a certificate has the following properties: $\tau_i = \sigma_P(\pi_i, G_i, C_i, A_i, T_i)$. The certificate τ_i issued to a client is sent back to CDN infrastructure along with tamper-evident logs that are signed by the nodes private key σ_i and a set of authenticators A_j, which provide non-repudiable evidence of a node's actions. Once auditing is complete, logs can be checked for consistency, plausibility and statistical anomalies. A summary of RCA's attributes is available in Table 2.5.

Hybrid systems have several limitations including not being able to utilise cache resources or multicast infrastructure in ISP networks. Moreover, they have issues producing trustworthy accounting information. Fortunately, the RCA system offers solutions to the reliable accounting problem. A full summary of hybrid CDN-P2P architecture attributes covered in this section is available in Table 2.6. Transparent cache systems, network CDNs

Table 2.6: Attributes of Hybrid CDN-P2P Architectures: ESM, SplitStream, PPLive, LiveSky and SAAR

Attribute	ESM	SplitStream	PPLive	LiveSky	SAAR
Trustworthy Accounting	-	-	-	-	-
Span Trust Domains	x	x	x	x	x
Decentralised	-	-	-	-	-
Data Integrity	-	-	-	-	-
Data Provenance	-	-	-	-	-
Scalable	x	x	x	x	x
Content Availability	x	x	x	x	x
Proximity Awareness	-	x	x	x	x
Transient Nodes	x	x	x	x	x
Inexpensive	x	x	x	x	x
Searchable	x	x	x	x	x
Guaranteed (Content) Control	x	x	x	x	x
Guaranteed Quality	x	x	x	x	x
Guaranteed Reliability	x	x	x	x	x
Guaranteed Performance	x	x	x	x	x
Use ISP Cache	x	x	x	x	x
Use P2P Cache	x	x	x	x	x
Use Multicast	x	x	x	x	x

and CDN Interconnection standardisation efforts propose solutions some of these challenges.

2.1.5 Transparent Caching

Similar to a web cache, the transparent cache is an ISP managed distributed network of cache servers located close to end-users that cache and deliver the most popular web content. However, they maintain the end-to-end relationship of traditional client-server communication. ISPs have had to deal with the rapid growth in so-called over-the-top (OTT) data traffic instituted by CDNs and services like YouTube and Netflix without yielding any additional revenue for the network operator. Transparent caching offers a partial solution to the bandwidth burden of OTT traffic by caching content close to end-users. For example, AT&T stated in a 2011 IP&TV forum presentation that content cached close to end-users needs to travel less distance (about 90% less) across a network than content peered from an upstream CDN [Skytide, Decemeber 2012].

A transparent cache works by redirecting web traffic to a cache server using deep packet inspection (DPI), Policy Based Routing (PBR) and the Border Gateway Protocol (BGP) without the need to configure a user's web browser [Barish & Obraczke, 2000]. Transparent caching has been an area of research since the early 90's, but previous transparent caches failed to maintain sufficient state for some application logic, which resulted in reduced functionality for users such as during user login and delivering device specific content e.g., iPad, iPhone, Nokia Lumia, and so forth. Recent improvements to transparent caching seek to maintain the end-to-end client-server relationship, which has the added benefit of always delivering fresh content, facilitating copyright compliance, and introducing no security vulnerabilities while being transparent to the user and the content provider.

There are two types of new transparent cache architecture: in-line and out-of-band. The in-line transparent cache uses DPI to inspect all data requests and responses, replying with cached data or caching requested data for later reuse. PeerApp is an example of an in-line transparent cache system. The out-of-band transparent cache is located off the main network and only certain types of traffic such as CDN and P2P are redirected to it. Examples of out-of-band transparent cache systems include OverSi and Quilt. These types of transparent cache are able to handle multiple forms of data such as HTTP, Real-Time Messaging Protocol (RTMP), BitTorrent and automatically update themselves to store popular content.

PeerApp: UltraBand

PeerApp's UltraBand provides a caching solution for the delivery of virtually any service or application including OTT video, software downloads, web browsing and P2P content delivery. UltraBand's architecture consists of a cache engine, which is used to detect repeatedly requested content; storage disks, which hold the content and a management centre, which is used for reporting, configuration and management [PeerApp, 2014b]. UltraBand uses DPI to intercept all traffic destined for the Internet and serves the requested content locally if it is available. One of the key features of UltraBand is that it maintains the relationship with the content provider by waiting for them to confirm delivery before serving the requested content. This technique, known as prefix-based web caching, compares the first N-bytes (the prefix-key) and the content-length of a requested cacheable, dynamic or un-cacheable web object with the prefix-key and content-length of cached web objects [Woo et al., 2013]. If they both match the download is cancelled and instead delivered from the local cache. This enables UltraBand to comply with US (DMCA 1998) and EU (E-Commerce Directive 2000) digital protection laws as the application business logic is not interfered with and content is always up-to-date.

Oversi's: OverCache

Oversi's transparent cache solution OverCache, provides caching for OTT video and P2P content delivery. OverCache maintains the client-server relationship with the CDN content provider for content requests but delivers the video file from a local cache server using the same prefix-based web caching technique as PeerApp [Woo et al., 2013]. OverCache consists of several architectural components including: a control plane, which offers a centralised view of subscriber demands; a data plane, which includes a network of distributed content caches; a management plane, which offers centralised management for the whole system.

The OverCache control plane intercepts traffic using DPI, PBR and BGP destined for popular CDN and P2P networks and diverts packets to the data plane cache servers. The data plane collects and stores HTTP and P2P traffic and provides better QoE to end-users when delivering requested content.

Accounting, Accountability and Efficiency: Transparent Cache Systems

The transparent cache systems neglect to provide any accounting information to the content distributor or content provider. They also fail to provide any accountability information for content distributed. Moreover, a transparent cache's inability to handle many streaming protocols or encrypted traffic results in just 22%-34% of bandwidth savings [Woo et al., 2013]. However, the transparent cache system does offer many desirable attributes for content distribution. These include: cheap resources close to end users in ISP networks that offer low latency, high performance, large cache resources and support content control. A summary of transparent cache architecture attributes covered in this section is available in Table 2.7.

In combination with CDN Interconnection (CDNI), network CDN (NCDN) and virtualisation architectures, transparent cache software and infrastructure are evolving towards a

Table 2.7: Attributes of Transparent Cache Architectures: UltraBand and OverCache

Attribute	UltraBand	OverCache
Accounting	-	-
Span Trust Domains	x	x
Decentralised	x	x
Data Integrity	-	-
Data Provenance	-	-
Scalable	x	x
Content Availability	x	x
Proximity Awareness	x	x
Transient Nodes	-	-
Inexpensive	x	x
Searchable	x	x
Guaranteed (Content) Control	x	x
Guarantee Quality	x	x
Guarantee Reliability	x	x
Guarantee Performance	x	x
Use ISP Cache	x	x
Use P2P Cache	x	x
Use Multicast	-	-

fully featured framework for content distribution [PeerApp, 2014a]. This is explored further in the next section.

2.1.6 Network CDN, CDNI and Virtualisation

The following architectures, which each have desirable attributes but different underlying objectives are evolving towards a fully featured framework of content distribution. This can be attributed to network CDNs (NCDNs), CDNI standards and the IETF Network Function Virtualisation (NFV) framework. In the remainder of this section, we give a brief overview of NCDNs, CDNI and virtualisation standards. Furthermore, we give a brief overview of how PeerApp's transparent cache system has evolved to support NFV. Finally, we outline the provision for accounting and accountability in these architectures.

Network CDN (NCDN)

A network CDN (NCDN) is a CDN deployed and managed inside an ISPs network at the network's *point of presence* (PoPs) i.e. where subscribers connect to the Internet. There are two types of NCDN: *managed* and *licensed* [Sharma et al., 2013]. In a managed NCDN, a CDN provider such as Akamai or the Google Global Cache place their own servers at PoPs inside the ISP's network, which they maintain on behalf of the ISP. In contrast, a CDN provider such as Akamai or EdgeCast license CDN software to ISPs enabling them to deploy and manage their own CDN. Consequently, ISPs can determine where to place content, route and redirect user requests. This enables NCDNs to enhance end-user QoE in comparison to traditional CDNs and reduce the traffic on the backhaul and core network. For example, recent research indicates that a mobile operator managed CDN with optimal positioning of cache servers in their network can reduce content retrieval costs by more than 50% when 80% of content is cached in the CDN close to users [Spagna et al., 2013].

There has been a lot of recent research in this area. For example, several projects investigate content placement, routing and redirection strategies [Spagna et al., 2013][Sharma et al., 2013]. Moreover, several projects are investigating NCDN and virtualisation in areas of resource sharing and providing a dynamic and scalable architecture that is easily managed [Jain & Paul, 2013][ETSI, 2013]. However, there are still outstanding challenges with regards to accounting and accountability, controlling content, request routing and sharing metadata between upstream and downstream CDNs. Consequently, interconnection between independent CDNs is an area of active research, which try to address many of these challenges.

CDNs and Interconnection

The aim of the various on-going CDNI standardisation efforts (IETF CDNI [L. Peterson et al., 2014]; ETSI Media Content Distribution (MCD) and Telecommunications and Internet Converged Services and Protocols (TISPAN) CDNI [ETSI, 2011]; and ATIS Cloud Services Forum (CSF) [ATIS, June 2011]) is to overcome many of the shortcomings of existing architectures by establishing a model for communication between separately administered CDNs. A CDNI standard will enable an authoritative upstream CDN to utilise the infrastructure of a standalone CDN that can offer the best service for content distribution. This involves defining how content is to be added, updated, deleted, pushed, pulled, converted (conversion or adaptation), blocked (geo-blocking), logged, and so forth between one or multiple CDNs.

Utilising CDNI infrastructure in this way will help reduce content delivery costs, facilitate better quality of experience for end-users and increase delivery robustness for network operators and CDN systems [Bitar et al., 2012]. Because the primary goals of each standardisation effort are the same, we focus on IETF's CDNI standardisation effort as a sample illustration of CDNI architecture.

IETF Content Distribution Network Interconnection (CDNI)

The IETF CDNI standardisation effort was established in 2011 with the intention of providing a model for CDN interconnection to industry, which develops on guidelines and requirements gathered by the IETF Content Internetworking (CDI) group [Day et al., 2003]. The CDNI model specifies interfaces and functionality for [Bitar et al., 2012][L. Peterson et al., 2014]: a control interface, a request routing interface, a metadata interface and a logging interface. The CDNI control interface facilitates CDN communication by helping interconnected CDNs with initial bootstrap, agreeing what logging information to share, purging old content, updating new content, and so forth. The request routing interface ensures the user is redirected to the correct downstream CDN for service delivery. The metadata interface enables the downstream CDN to request metadata information e.g., actor, genre, rating, content resolution, aspect ratio, and so forth, from the upstream CDN [Bitar et al., 2012]. The logging interface facilitates the distribution of logging information between interconnected CDNs for billing, reporting and analytics purposes.

A major challenge for the CDNI group is the provision of an adequate solution for the sharing of logging information between CDNs while maintaining log integrity. Logging information is used by CDNs to charge content providers and perform analytics related to system and network performance, troubleshoot issues, track audiences and determine quality of service (QoS) to end-users. For example, an upstream CDN depends on these logs to ensure the user achieves adequate QoS and that SLA targets were achieved by the downstream CDN. However, a downstream CDN may want to conceal system outages. Additionally, a downstream CDN may want to hide the topology of their network, the number of distribution servers and their location. Therefore, some filtering or obfuscation of the logging information is required by CDNs before sharing logs, but not enough to conceal critical information. The question of how to filter logs while maintaining their integrity remains unanswered by the CDNI architecture. One of the suggested solutions involves utilising a trusted third party CDN to filter the logs [L. Peterson et al., 2014].

Virtualisation

In addition to using NCDN principles and infrastructure and CDNI standards, CDNs are utilising virtualisation services inside ISP networks for content distribution. In these environments CDNs relinquish control over physical computation resources to ISPs with the objective of being closer to end-users (i.e. subscribers) so that server delay and network traffic are reduced. Moreover, virtual servers can be allocated in hours or minutes in different locations in an ISPs network depending on end-user demands, application requirements and resource availability [Frank et al., 2013]. For example, the *Network Platform as a Service* (NetPaaS) framework supports CDN-ISP collaboration by allowing CDNs to expand or shrink virtual server resources inside ISP networks on-demand [Frank et al., 2013].

Furthermore, the IETF Network Function Virtualisation (NVF) framework advocates evolving virtualisation technology to combine networking hardware functions in software that can be run on standard high volume servers, switches and storage infrastructure [ETSI, 2013][Mijumbi et al., 2016]. This results in reduced power consumption, service innovation and automation, operational efficiencies, manageability, flexibility and efficiencies for ISP's, content providers and content distributors [ETSI, 2013][Jain & Paul, 2013]. For example, these principles have been utilised recently by transparent cache providers (e.g., PeerApp) and CDN providers (e.g., EdgeCast and Limelight) to distribute high quality video (with 4K (4000 pixel) resolution) reliably to end users while reducing costs and network traffic and increasing end-user QoE [PeerApp, 2014a].

Table 2.8: Attributes of NCDN, CDNI and Virtualisation Architectures

Attribute	NCDN	CDNI	Virtualisation
Trustworthy Accounting	-	-	x
Span Trust Domains	x	x	x
Decentralised	x	x	-
Data Integrity	-	-	-
Data Provenance	-	-	-
Scalable	x	x	x
Content Availability	x	x	x
Proximity Awareness	x	x	x
Transient Nodes	-	-	-
Inexpensive	x	x	x
Searchable	x	x	x
Guaranteed (Content) Control	x	x	x
Guaranteed Quality	x	x	x
Guaranteed Reliability	x	x	x
Guaranteed Performance	x	x	x
Use ISP Cache	x	x	-
Use P2P Cache	-	-	-
Use Multicast	-	-	-

Accounting, Accountability and Efficiency: NCDN, CDNI and Virtualisation Environments

NCDN, CDNI and virtualisation architectures offer cheap mechanisms of distributing content with guaranteed performance, quality, reliability, and so forth. Additionally, content control is maintained by the content provider. These environments can also collect all accounting information, which is shared with content providers. However, they do not provide accountability capabilities to track and audit file history, integrity or provenance of logging information produced for physical or VM servers used [Ko et al., 2011]. Moreover, they cannot utilise P2P or multicast infrastructure. A summary of architecture attributes covered in this section is available in Table 2.8.

In environments where the customer does not directly control the physical infrastructure, it can be difficult to determine who is responsible, customer or NCDN/cloud provider, for problems encountered in the absence of accountability [Haeberlen, 2010]. Consequently these environments are susceptible to abuse, accidental faults, malicious users, software bugs, data loss, outages, and so forth, which impacts user adoption [Haeberlen et al., 2010]. However, several proposals outline the challenges and requirements for accountability in these environments [Ko et al., 2011][Haeberlen, 2010]. For example, a basic log primitive called AUDIT$(A,S,t1,t2)$ (i.e. an agreement A for service S during time interval $t1...t2$) could be offered [Haeberlen, 2010]. This primitive could be invoked by a customer to detect and provide non-repudiable evidence of faults, which can be verified independently by a third party.

For example, the *accountable virtual machine* (AVM), which is an ordinary virtual machine (VM) with accountability capabilities, uses a similar log entry (e.g., $e_i = (s_i, t_i, c_i, h_i)$). It records non-repudiable information about software running on the VM that can be audited by replaying it against a VM reference copy [Haeberlen et al., 2010]. AVM uses similar principles to PeerReview's tamper-evident logging [Haeberlen et al., 2007], which is extended to add non-deterministic asynchronous inputs (such as virtual hardware interrupts, mouse clicks, and so forth) from the VM instance. All application inputs, outputs, non-deterministic events and tamper-evident logs are maintained by an accountable virtual machine monitor (AVMM) process that runs in the VM. An authenticator, which is a commitment to log integrity, is attached to all outgoing messages: $a_i := (s_i, h_i, \sigma(s_i || h_i))$. Consequently, authenticators can be used as verifiable evidence that the log has been tampered. In order to perform log auditing and replay in AVM, a copy of the tamper-evident log and authenticators produced during the execution are required. Additionally, the original software image, to replay the whole log, or a snapshot of the virtual machine state, to replay the log from a log segment, is required. Authenticators are verified against log segments. If this step succeeds the log is genuine. Furthermore, the segment is replayed on the reference software or snapshot to check whether or not the execution described by the segment matches the events in the tamper-evident log [Haeberlen et al., 2010]. These tools allow users to detect faults, identify faulty nodes and produce non-repudiable evidence of a VM's actions. AVM could be extended to provide *evidence of time* for SLAs by including precise program timing during replay [A. Chen et al., 2014]. A summary of AVM attributes is available in Table 2.9.

VMs have additional levels of complexity in comparison to physical servers if tracking the VM to physical server mapping is required. These include [Ko et al., 2011]: links between the VM and physical machine operating system; VM location and physical machine

Table 2.9: Attributes of AVM: An Accountable Virtual Machine Architecture

Attribute	PeerReview
Accountability	x
Public/Private Key	x
Tamper-Evident Log	x
Log Consistency	x
Trusted Auditing	x
Non-Deterministic	x
Challenge/Response Protocol	-
Rapid Fault Detection	-
Defend Accusations	x
Statistical Sample	-
Centralised	x

location; how files are written in VM and physical machine memory. Current systems, such as AVM [Haeberlen et al., 2010], provide accountability for the VM instance only, not physical server to VM mapping.

2.1.7 Summary

Many of the architectures discussed in this section offer desirable attributes for efficient content distribution such as multicast, caching, using distributed resources, and so forth. However, a content distribution architectures successful adoption depends on the requirements of content providers, ISPs and end-users related to mobility, security, efficiency, cost, trust, privacy, accounting, accountability, and so forth. In this section, we observed that content providers primarily utilise content distribution infrastructure that offers extensive accounting and accountability information for the content distribution process e.g., fixed infrastructure CDNs. This observation forms the basis of our hypothesis, which states that the level of adoption and utilisation of a content distribution architecture in commercial environments relies on the level of content accounting and accountability they offer. Combining this observation with the *information centric networking* (ICN) future Internet paradigm, which utilises many of the attributes discussed in this section for efficient content distribution, can lead to a scalable, trustworthy, reliable, accounting and accountability framework for content distributed in Trusted Infrastructure and Untrusted Infrastructure environments.

2.2 Information Centric Networking (ICN)

The primary objective of the ICN paradigm is to identify the limitations of the current Internet architecture and propose solutions for the future Internet [Xylomenos et al., 2014]. These limitations include [Xu et al., 2006]: poor feedback and accountability; lack of data integrity, reliability and trust; mobility challenges; bad security; inefficient caching and mirroring; unnecessary data retransmissions and network traffic; network congestion; high distribution costs; poor content availability; and lack of user privacy. This is achieved by treating names or identifiers such as content, services and devices as first-class citizens. Key ICN design principles include: routing, caching, mobility, security, content protection, trust, privacy and accountability. ICNs are being actively investigated by many projects funded by the National Science Foundation (NSF) in the USA and the European Commission in the EU [Venkataramani et al., 2014][Han et al., 2012][Zhang et al., 2010][Trossen et al., 2008][Trossen, 2011].

ICNs naturally support the opportunistic caching of self-verifying content in the network offering increased data availability, less bandwidth demands, reduced origin server load and better QoS for content delivery [Fayazbakhsh et al., 2013]. For example, the MultiCache ICN architecture demonstrated traffic reductions of between 53%-62% for inter-domain and intra-domain traffic and better download times (over 62% lower) in comparison to BitTorrent P2P content distribution [Katsaros et al., 2011]. Caching can take place on-path i.e. directly in the path of network traffic such as Named Data Networking (NDN) (NDN) [Zhang et al., 2010], or off-path i.e. available from an off-path cache server such as Network of Information (NetInf) [Ahlgren et al., 2012]. On-path caching works in conjunction with name-based routing i.e. a name is looked up in a routing table and then forwarded towards the relevant data store. Data can be stored along the reverse request path in a router cache (e.g., NDN). In off-path caching, data is resolved to a location and data availability announced using routing protocols, which can then be resolved by a name resolution system (e.g., NetInf).

In the remainder of this section, we briefly describe several of the main ICN architectures including Data-Oriented (and Beyond) Network Architecture (DONA), Publish-Subscribe Internet (PSI), Network of Information (NetInf), Named Data Networking (NDN), eXpressive Internet Architecture (XIA) and MobilityFirst.

2.2.1 A Data-Oriented (and Beyond) Network Architecture (DONA)

The Data-Oriented Network Architecture (DONA) was proposed by a research team at UC Berkeley in 2007. It was the first ICN architecture to propose a clean-slate redesign of Internet naming and name resolution by advocating an approach based around flat-namespaces for content, services, and so forth [Koponen et al., 2007]. These flat-namespaces take the following form: <*P:L*>, where *P* is a cryptographic hash of the principal's public key where the principal represents a host, domain, person, organization, and so forth, and *L* represents the label or human readable hierarchical name [Ghodsi et al., 2011][Koponen et al., 2007]. Every principal must be associated with a public-private key pair and every datum, service, host, domain, and so forth must be associated with a principal [Koponen et al., 2007]. DONA proposes that data is named in the following format [Ghodsi et al., 2011]:

<data, *P*, *L*, metadata, signature>

The data received in this format is self-certifying and can be verified as having come from the principal by checking that the public key hashes to *P* and the key generated the signature, which associates the object with the name [Koponen et al., 2007].

DONA uses *resolution handlers* (RHs), which are servers located within an autonomous system to identify and locate data. There is at least one RH in each autonomous system. Additionally, they are interconnected as an overlay over the Internet to RHs in Tier 1 and peering autonomous systems supporting a hierarchical name resolution service [Xylomenos et al., 2014]. The RH's are aware of data location on a network due to REGISTER(P:L) messages, which act similar to link state advertisements (e.g., OSPF). These messages are sent to RHs by content publishers (i.e. the principal). Moreover, RHs propagate these messages to parent and peering autonomous system domains. Tier 1 autonomous systems are aware of all registrations in the entire network [Xylomenos et al., 2014].

A client seeking data will issue a FIND(P:L) or FIND(*:L) message (where * means *any* purveyor) to determine the location of an object named <P:L>or <*:L>. The RH's will route this request to the most appropriate copy of the data. RH's use longest prefix matching on <P:L>or <P>or <L>to yield the entry, which determines the location of data. Once a FIND record has been resolved by a RH, a standard transport-level TCP response ensues and data is routed using normal IP routing and forwarding mechanisms [Koponen et al., 2007]. Moreover, mobility is supported by clients simply issuing a new FIND request from their next location [Xylomenos et al., 2014].

DONA supports both on-path and off-path caching. If a RH wants to keep a cache of the data it needs to change the FIND source address to be its own address. This ensures that the data will traverse this RH on the return journey to the client. The RH cache can now serve future FIND requests for this data. Once an RH has a copy of the content, it can advertise this content using REGISTER(P:L) to other RH's on the network, assuming that the machine is authorized to service this content. DONA also provides an UNREGISTER command to notify RH's of expired data or when a mobile publisher has moved location (before re-register), which will be relayed to other RH's [Xylomenos et al., 2014]. If any client in the system or autonomous system cannot service a FIND request, an error message is returned to the source of the FIND [Koponen et al., 2007].

DONA also proposes using a routing mechanism similar to source-routing called *path-labels*, which are constructed domain by domain as a packet travels from the client to the server. The local-id of a domain is appended to packets as they cross-different domains by routers. When the packet is on the return journey the local-id is removed from the source address. This methodology has several benefits including: smaller inter-domain routing tables, no-globally meaningful addresses, difficulty to spoof source addresses, a symmetric path across domains, supports multicast distribution trees and importantly an ability for ISP's to identify DOS attacks [Koponen et al., 2007].

2.2.2 Publish-Subscribe Internet (PSI)

The PSI architecture realises the design principles of the EU funded projects PURSUIT (September 2010 to February 2013) and its predecessor Publish-Subscribe Internet Routing Paradigm (PSIRP) (January 2008 to June 2010) for a clean-slate, information oriented Internet architecture. PSI supports caching, native multicast, multihoming and mobility of information objects including files, content chunks, streaming media and services [Xylomenos et al., 2012]. A unique ID pair identifies information objects consisting of: scope identifier

(SID) and rendezvous identifier (RID). SIDs group related information while RIDs uniquely identify the information object itself. A RID must be unique within a scope(s) and a SID must be unique within a parent scope. Information objects can belong to multiple scopes, however they must be associated with at least one scope [Xylomenos et al., 2014]. Information consumers *subscribe* to objects in a scope or information objects directly based on RID.

The PSI architecture is supported by several key components. These include a rendezvous node (RN), which matches a subscriber's request to an information object publisher; a topology manager (TM) that determines the best route to a subscriber; and a forwarder node (FN), which delivers the requested object [Xylomenos et al., 2014]. RNs, or a collection of RNs in a rendezvous network (RENE) act as a hierarchical DHT to locate data based on (SID,RID) pair [Xylomenos et al., 2012]. Information object producers announce the availability of *published* information objects to their local RN. TMs utilise link-state routing to compute the best path to a consumer, in which RNs and FNs also participate. The TM uses the LIPSIN source-routing scheme to compute the forwarding path to consumers. LIPSIN is a multicast forwarding mechanism that uses a Bloom filter to encode source-route-style forwarding instructions into packet headers [Jokela et al., 2009]. Moreover, PSI supports security in the architecture using Packet Level Authentication (PLA), a technique for encrypting and signing individual packets [Xylomenos et al., 2014]. Consequently, the information consumer can easily establish data integrity, accountability and confidentiality. Additionally, an RID can optionally be a hash of the information object facilitating easy self-certification at the consumer. Consumer mobility is supported using multicast and caching. This is achieved by sending data to multiple caches simultaneously (sometimes based on prediction) with the goal of reducing handoff latencies [Xylomenos et al., 2014]. Finally, PSI supports both on-path and off-path caching and managed content replication (comparable to a CDN) [Xylomenos et al., 2012].

2.2.3 Network of Information (NetInf)

The Network of Information (NetInf) architecture was proposed and implemented by the *Architecture and Design for the Future Internet* (4WARD) project, active from January 2008 to December 2010 and the *Scalable and Adaptive Internet Solutions* (SAIL) project active from August 2010 to February 2013. Both were funded by the EU under FP7. Named *information objects* are flat-ish taking the form: *ni://A/L*, where *A* is the authority and *L* is the local part relative to the authority [Xylomenos et al., 2014]. Both *A* and *L* can be a hash (of the authorities public key and/or information object) allowing self-certification or a hierarchical string [Dannewitz et al., 2010]. Hierarchical names are useful for routing as longest prefix matching can be used. However, matching information objects to *A* and *L* are always treated as if they are flat identifiers i.e. they must match content exactly.

One of the fundamental differences between NetInf and other ICN architectures is that the public key/secret key (PK/SK) pair in NetInf is bound to the information object rather than to any particular owner or company. This means that the owner of a data object can change while keeping the information object ID persistent [Dannewitz et al., 2010]. The *metadata* field provides additional information to help validate the data e.g., public keys for information object and owner identification, content hashes, certificates and a data signature authenticating the content.

NetInf supports two methods of information object retrieval: 1) using a *name resolution system* (NRS) or 2) using *name-based routing* [Xylomenos et al., 2014]. The NRS uses a distributed hash table (DHT) such as Multilevel DHT (MDHT) or SkipNet, which is a hierarchy of DHT's for name resolution. The NRS resolves information objects (i.e. *ni://A/L*) to a globally-available resource (Global NRS) using the authority *A* part; or locally available resource (Local NRS) using the local part *L* relative to the authority. Moreover, the MDHT architecture also proposes a Resolution Exchange (REX) system that keeps track of all data object owners in the NetInf Internet, which provides redirects to the primary resolution system. REX principles are similar to DNS where a trusted third party will run distributed REX systems. The other object retrieval method proposed by NetInf is name-based routing. Content routers (CRs) are populated by a routing protocol used by publishing sources to advertise available information objects, which are propagated to neighbour routers. Information object requests are sent hop-by-hop towards the publisher or cache. Furthermore, a hybrid approach using the NRS (to reach the general location) and name-based routing mechanisms or vice versa can be chosen freely, which keeps routing table sizes small.

Like the other ICN designs, caching is fundamental to the NetInf's ability to provide enhanced information dissemination. NetInf supports both on-path and off-path caching. It employs a *storage engine* to manage requests by end nodes to STORE() data on the network. Once data has been stored on the network, the storage engine will register the published data into the NRS using the PUBLISH() primitive. Data is retrieved using the GET primitive. Finally, mobility is supported by the NRS maintaining topological information about each connected node, which is updated by each host as it moves through the network [Xylomenos et al., 2014].

2.2.4 Content-Centric Networking (CCN)

The Content-Centric Networking (CCN) (CCN) architecture, which supports ICNs principles, was initiated by Van Jacobson at the Palo Alto Research Centre (PARC) and described in a Google tech talk in 2006 [Jacobson et al., 2009]. The *Named Data Networking* (NDN) project, which is led by Lixia Zhang (UCLA) and funded by the National Science Foundation (NSF) in the US under its Future Internet Architecture (FIA) program, is an implementation of CCN principles to improve on the current Internet architectures limitations. The primary goal of the NDN project is to request an object by name and receive it from the network rather than from a particular end-point on the Internet. Information flow thus becomes receiver driven (*pull-based* model) and the network (i.e. the NDN routing infrastructure) directs the content request to the best available resource. This is supported by digitally signed self-verifying content.

The main infrastructure in the NDN architecture is the NDN router, which is comprised of several components including: a Pending Interest Table (PIT); a Forward Information Base (FIB), which is populated by a name-based routing protocol (e.g., based on OSPF) or with static routes; and a Content Store, which can store requested data for future requests

[Y. Liu et al., 2009]. The NDN client retrieves content by sending an *Interest* packet that contains the name of requested content to the local NDN router, which is added to the PIT. The NDN router performs longest-prefix match lookup for content name on the FIB table and forwards the *Interest* out the relevant interface. Once a copy of the data is located, it is sent back to the requesting client by checking the PIT entry of each router along the *Interest* reverse request path. NDN routers create a multicast tree of frequently requested content aggregated by the PIT table. Future requests for the same content can be delivered directly from the NDN router content store (on-path caching) without having to return to the content source for the data. Subscriber mobility is supported using the Listen First Broadcast Later (LFBL) routing protocol, caching and multicast nature of the NDN architecture [Xylomenos et al., 2014]. Finally, like other ICN architectures, NDN does not provide natural support for content accounting and accountability; instead it offers natural privacy to users [Jacobson et al., 2009].

Our work on the Savant framework outlined in Chapters 4 and 5 is based on NDN architecture principles.

2.2.5 eXpressive Internet Architecture (XIA)

The XIA clean-slate architecture, also funded by the NSF's FIA program since 2010, aims to provide an evolvable Internet architecture with functionality to support expressiveness and intrinsic security [Anand et al., 2011]. XIA utilises the concept of *principals*, referred to as *eXpressive identifiers* (XIDs), to support expressiveness in the architecture. Every principal (or XID) uniquely identifies a host (HID), administrative domain (AD), service (SID), content (CID), and so forth. These principals support an evolvable network that can easily introduce new functionality such as the incremental deployment of new Internet architectures over time. This is supported by a *fallback* option, which is a *directed acyclic graph* (DAG) (used by XIA's eXpressive Internet protocol (XIP)) containing the destination principal and additional legacy principals that are known to existing routing infrastructure [Han et al., 2012]. This fallback option specifies alternative action(s) if a legacy router does not know how to deal with the primary principal. For example, AD:HID:SID:CID defines the route to CID, where AD (i.e. a fallback option) is globally routable. Moreover, this offers the network more flexibility to perform in-network optimisations as it can observe intent (based on principals) and act on them directly. For example, XIA supports on-path and off-path caching of content objects [Han et al., 2012]. As a result, if a user requests content AD:HID:SID:CID, the network can direct the request to a local cache server with an available copy of the requested content i.e. CID.

Intrinsic security is supported in the architecture by unique XIDs derived from the hash of a public key or a hash of the content itself. As a result, communicating entities can independently ascertain the integrity of the communication process without support from third-party infrastructure [Anand et al., 2011]. Moreover, applications can utilise a name-resolution service to resolve human readable names to addresses (i.e. XID or DAGs) [Han et al., 2012]. Finally, these mechanisms support an evolvable Internet architecture that also offers support for ICN content distribution.

2.2.6 MobilityFirst

MobilityFirst, also funded by the NSF's Future Internet Architecture (FIA) program since 2010, proposes a clean-slate Internet architecture with mobility and trustworthiness as key architectural design principles [Venkataramani et al., 2014]. These design goals are achieved by separating names or identifiers from addresses or network location. This is supported by a logically centralised *global name service* (GNS), which dynamically binds names to a flexible set of attributes (including but not limited to a network address). Globally unique identifiers (GUIDs) define names for principals such as devices, users, services, networks, content, and so forth. These are derived using a one-way hash function of the entities public key or from the content itself. Self-certification and authentication of GUIDs between communicating entities is achieved without third-party support using a simple bilateral cryptographic challenge-response procedure [Venkataramani et al., 2014]. MobilityFirst supports on-path content storage and retrieval for popular content [Xylomenos et al., 2014]. Routing is supported by hierarchical forwarding tables, the GNS and routable content addresses (which are encoded as a two-tuple [PID, CID] where PID corresponds to the publisher's GUID and CID

Table 2.10: Attributes of ICN Architectures

Attribute	ICNs
Accounting	-
Span Trust Domains	x
Decentralised	x
Data Integrity	x
Data Provenance	x
Scalable	x
Content Availability	x
Proximity Awareness	x
Transient Nodes	-
Inexpensive	x
Searchable	x
Guaranteed (Content) Control	-
Guarantee Quality	x
Guarantee Reliability	x
Guarantee Performance	x
Use ISP Cache	x
Use P2P Cache	x
Use Multicast	x

is the content's GUID) [Venkataramani et al., 2014]. MobilityFirst supports a dynamic, secure and mobile architecture with many endpoint principals and location-independent communication primitives supported by a logically centralised GNS. Finally, these mechanisms offer support for ICN content distribution.

2.2.7 Summary

This section outlines some common ICN research architectures and their respective methodologies to resolve the problems and limitations of the current IP based Internet model. A summary of ICN architecture attributes covered in this section is available in Table 2.10. Content naming plays an important role in each architecture determining if support of a PKI or a name resolution service is required. Naming also plays an important part in routing, routing aggregation, security (e.g., self-verifying content) and so forth. Moreover, privacy is identified as an important issue for all architectures. However, caching data on untrusted ICN infrastructure has consequences for accounting and accountability, as the content provider cannot directly observe or trust analytic's associated with content distributed. This is in contrast to the current CDN content distribution model, which provides extensive analytic's to content providers. Moreover, initial ICN architectures give no indication about how to gather this information from trusted or untrusted infrastructure. Instead, many of these architectures make a virtue out of not providing it claiming to offer natural privacy to users. Consequently, the goal of our work on Savant is to provide a framework for collecting accounting and accountability information for content distributed from trusted and untrusted ICN infrastructure. However, first we identify existing systems to support ICN architectures and help avoid security threats and attacks mechanisms. Moreover, we will focus on security mechanisms specifically related to the NDN ICN architecture as this is the architecture we have utilised for the Savant framework.

2.3 Security Mechanisms

ICN content can be retrieved from any infrastructure element (trusted or untrusted) that has an available copy. This is supported by digitally signed self-verifying content to help establish integrity, authenticity, provenance and trust in content received. However, securing the content itself is not sufficient to ensure that an end user will receive the exact piece of content they requested. Additionally, securing content rather than the path it travels across has exposed many new and legacy types of security issues in ICNs [AbdAllah et al., 2015]. For example, a user can encounter network delays due to denial of service DoS attacks, corrupt data, confidentiality issues, and so forth. Consequently, ICNs depend on a variety of infrastructure, algorithms and techniques to ensure the requested content item is delivered to end-users. Furthermore, NDN's model of trust does not rely on any centralised public key infrastructure (PKI), instead advocating a distributed model of trust. Trust in keys is typically established using a PKI-like certificate chain based on the content naming hierarchy [Xylomenos et al., 2014]. However, individual applications are responsible for managing their own trust models. Consequently, the elements outlined in this section are required to support Savant framework. We start with a brief description of DNS and PKI as several of the ICN security implementations utilise these components in their design.

2.3.1 DNS

The domain name system (DNS) is a globally distributed hierarchical name service database for the Internet that is administered by Internet Assigned Numbers Authority (IANA), which is in turn managed by Internet Corporation for Assigned Names and Numbers (ICANN). Its most popular use is mapping human readable domain names to IP addresses. The original specification was released in 1983 as a replacement to the *hosts.txt* file, which was centrally maintained and distributed via FTP by the Stanford Research Institute Network Information Centre (SRI-NIC) [Mockapetris & Dunlap, 1988]. DNS was designed to be easily updatable, scalable, distributed, fault tolerant and utilise caching to improve performance. The architecture consists of three major components [Mockapetris, 1987]: domain namespace and resource records (RRs), name servers and resolvers.

The *domain namespace* forms a tree-like structure, with "." acting as the root of the tree. Each node or leaf in the tree is associated with a label. A nodes domain name is a concatenation of labels, separated by ".", which represent a path from a node (left or farthest point from root) to the root of the tree. Data for each domain name is organised as a set of *resource records* (RRs). Each RR is composed of several fields, which include: type (e.g., address record (A)), time to live (TTL) (i.e. length of time in seconds to cache a RR) and RDATA (the data e.g., IP address) [Mockapetris & Dunlap, 1988]. By formatting data in this way, queries can be limited to RRs of a specific type, which can also be cached by name servers. *Name servers* are repositories of information that answer queries with data they possess i.e. RRs. *Resolvers* interface with clients and use algorithms to resolve client queries to name servers with RRs. These algorithms are supported by local DNS resolvers that are preloaded with a list of root or top level domain (TLD) servers e.g., .ie, .org, .com. As a result, they can direct unknown requests to a resolver that can provide an answer.

DNS's scalability is supported by the concept of zones, which are sections of the global database controlled and managed by specific organisations and that can grow to arbitrary size. An organisation can gain zone control (become the *authoritative* name server for a domain) by persuading a parent zone to assign it subzone control. Moreover, a parent can further delegate subzones under its control to other organisations. Each parent is responsible for maintaining the zone's data and providing redundant servers [Mockapetris & Dunlap, 1988]. Utilising this model, DNS can easily scale to provide a distributed and fault tolerant name service for the Internet. Finally, *registrars* are entities authorised to manage and reserve domain names within TLDs.

DNS is a popular globally distributed hierarchical name service database for the Internet that maps human readable domain names to IP addresses. Its architecture is outlined briefly here as many ICN security implementations borrow components from its design.

2.3.2 PKI

Public key infrastructure (PKI) provides strong authentication on the Internet by binding a public/private key pair to an entity e.g., person, organisation, account, and so forth, using a digital certificate. A PKI is a set of components including users, certificate authorities (CAs), software, physical infrastructure and policies for producing, distributing, storing, using, authenticating and revoking digital certificates and managing the life cycle of public keys. Digital certificates are issued by a CA based on a certificate standard for interoperability such as X.509, Pretty Good Privacy (PGP) and Simple Public Key Infrastructure/Simple Distributed Security Infrastructure (SPKI/SDSI) [Buchmann et al., 2013]. These standards specify formats for digital certificates, certificate revocation lists, path certification algorithms, and so forth.

There are several trust models used by these certificate standards including hierarchical trust (e.g., X.509), web of trust e.g., PGP, and small-world model of trust e.g., SPKI/SDSI [Buchmann et al., 2013][Clarke, 2001]. In the hierarchical model, the CA acts as a centralised *trust anchor*, or root of trust, for all digital certificates produced. Similarly, the web of trust model establishes trust in keys by getting trusted users of the network to certify key ownership. In each model, trust in an entity can be established by building a certificate path from a trusted entity to the certificate of the entity requiring trust. For example, in the hierarchical model, trust in an entity can be reduced to trusting the anchor i.e. the CA, and following the certificate path or chain of trust to the entity requiring trust.

Several ICNs propose using the SPKI/SDSI model where trust originates from a local principal [Zhang et al., 2010][Ghodsi et al., 2011]. We also use this model to support Savant. Consequently, we provide a broader overview of it main components here.

2.3.3 SPKI/SDSI

SPKI/SDSI supports a scalable local namespace architecture using public keys based on a small-world model of trust. It binds authorisations i.e. privileges, delegation of rights, and so forth, to local identifiers i.e. local names in the certificate issuers local namespace, that are valid globally [Clarke, 2001]. In SPKI/SDSI a principal can also act as a CA that can issue name certificates and *authorisation certificates* for their local namespace.

A SPKI/SDSI name certificate binds a local name (e.g., user1) in the certificate issuer's namespace (e.g., /netflix) to a public key. It has four fields including [Clarke, 2001]: *issuer*, which is the public key that signs (or authorises), the name certificate; *identifier*, which contains the issuer's public key and a local name that is being defined; *subject*, the thing being empowered by the certificate e.g., the public key of the entity receiving a namespace; and *validity period* i.e. certificate start and end date.

A SPKI/SDSI authorisation certificate grants a specific authorisation to a certificate subject. It has five fields including [Clarke, 2001]: *issuer's* public key that signs, or authorises the certificate; *subject*, the public key or group of the entity receiving authorisation e.g., the end-user; *tag*, the authorisation being granted e.g., allowed publish in namespace /netflix/user1; *delegation bit*, a Boolean value; and *validity period* i.e. certificate start and end date.

Moreover, SPKI/SDSI access control lists (ACLs) have a similar syntax to SPKI/SDSI authorisation certificates. ACLs restrict access to a resource, which is set up and managed by a *guardian* process responsible for protecting it. ACL fields include [Clarke, 2001]: *issuer*, *subject*; *tag*; *delegation bit* and *validity period*. A request to access a resource is contained in a signed tag, which authenticates the request and a chain of certificates that prove authorisation to perform a request [Clarke, 2001].

The ability to trust data received and ensure the network delivers valid content is a fundamental design requirement for all ICNs. Consequently, in the remainder of this section we give a brief overview of some of the security considerations in ICNs, with a specific focus on NDN.

2.3.4 Security Attacks in ICNs

One of the primary philosophies of the ICN paradigm is to secure content itself rather than the path it travels across. This is achieved by decoupling a user's trust in content from where it is obtained by enabling the content to *self-verify* i.e. the user can establish integrity, trust and provenance in content received from trusted or untrusted infrastructure. This facilitates more efficient distribution of data in an ICN network as copies of data can be supplied from any infrastructure element that has an available copy. However, ICNs are susceptible to many new and legacy types of attack in part as a result of this change, which have an impact on the following [AbdAllah et al., 2015]:

- Privacy and Censorship: As data is named, users are susceptible to attacks such as monitoring and censorship.
- Cache Poisoning: Caches intentionally polluted with corrupt or unpopular content.
- Resource Exhaustion: Infrastructure susceptible to large amounts of requests or flooding attacks.
- Path Infiltration: Attackers can announce invalid routes for content.
- Network Congestion: Attacker directing data through heavily congested networks.
- Denial of Service (DoS): Any of the attacks outlined so far can lead to denial of service i.e. user not receiving requested content.
- Unauthorised Content Access: Viewing content without permission or authorisation.
- Masquerading: Gaining access to the content producer's private key and pretending to be them.

The CCN-KRS framework, outlined in Section 2.3.6, can help prevent issues such as cache poisoning and DoS attacks. Others, such as user privacy, preventing unauthorised access to content or circumventing path infiltration attacks will be covered later as part of the Savant framework in Section 4.3.1. Additionally, the Savant framework itself was designed to detect problems during the content distribution process. For example, in Section 5.8.2, we identify several use cases where Savant can detect infrastructure faults and resource exhaustion scenarios. Moreover, Savant could be utilised to redirect clients to alternative trusted cache resources for content distribution using authenticated interest commands (see Section 2.3.7). Finally, while we offer solutions to many of these attacks, there are still issues (e.g., privacy related), which may go undetected or unresolved by the architecture.

2.3.5 Naming and Security

Data security is achieved in ICNs by binding content to an entity that created it using public key cryptography. Consequently, all entities (or principals) that publish content (e.g., videos, web pages, and so forth) must be associated with a public/private key pair. Using public keys, the following security goals can be derived from data received: authenticity and authentication (data provenance), confidentiality and privacy, integrity (data not modified) and non-repudiation (cannot deny producing data). The two most common methodologies for naming content in ICNs include: hierarchical human-readable names and self-certifying names.

Hierarchical Names

Hierarchical names are composed of arbitrary length typically human-readable strings similar to DNS, which are delimited by '/' e.g., /ie/tcd.ie/cs/index.html. Names are bound to entities e.g., organisation, person, and so forth, by an inherent relationship between the content name and the publisher. Content *metadata* provides the location of public keys needed to establish integrity, provenance, trust and authenticity in the content received. This can be supported by traditional PKI mechanisms to establish trust in keys such as hierarchical trust or web of trust. However, ICN architectures are free to define new trust models as trust is between the content publisher and consumer.

For example, CCN advocates using SPKI/SDSI, which uses a distributed trust model that does not require support from a globally trusted (root) authority. Additionally, as all CCN/ICN entities that publish content are bound to a principal, they already have the components (i.e. public keys) required to become a CA and support the SPKI/SDSI trust model [Clarke, 2001]. Consequently, publishers (i.e. the *certificate issuer*) can define local names in their namespace and bind these names to public keys of other CCN publishers (i.e. the *certificate subject*) [Jacobson et al., 2009]. For example, the /tcd publisher can define the local name /tcd/cs, which can be bound to the public key of Trinity College Dublin's computer science department. Similarly, authorisations (such as read/write privileges) can be granted from the certificate issuer to the certificate subject. As CCN's names are hierarchical and content is published as a named content object, a trust relationship can be expressed as published CCN content, which effectively act as a digital certificate.

Flat-namespace

Flat self-certifying namespaces consist of **<P:L>**, where **<P>** is a cryptographic hash of a principal's public key and **<L>** is a unique label with respect to the principal, chosen by the principal. For example, the DONA architecture [Koponen et al., 2007][Ghodsi et al., 2011] distributes content in the form **<Data, P, L, Metadata, Signature>**. Data received can be verified as having come from an original source (i.e. the principal) by checking that the public key hashes to **<P>** and the signature matches the data received. Consequently, data integrity can be assured. However, a label can also be a cryptographic hash of the data itself, which end users can use to establish data integrity. In this case, **<P>** can be assigned to any purveyor of the content such as a CDN. In both cases data itself is self-verifying. Consequently, they do not require the services of a PKI to establish data integrity. However, as namespaces are flat, long and not human readable, it can be difficult to associate names to content. Consequently, some external mechanism to bind human-readable names to flat names is required. However, users can easily learn flat names via some external trusted mechanism such as a search engine, social network or from friends.

The naming methodology used by ICNs has an impact on whether or not a PKI or some external mechanism to bind human-readable names to flat names is required. NDN requires the support of a PKI.

2.3.6 CCN-Key Resolution Service (CCN-KRS)

The CCN architecture requires content to be signed by its publisher. Content metadata provides the location of public keys needed to establish integrity, provenance, trust and authenticity in the content received. However, the CCN network can have multiple copies of signed content with the same name from different content providers. As a result, an end-user cannot trust that the *content name:public key* binding in the metadata, which is valid, reflects the data they requested. Moreover, network infrastructure cannot easily differentiate between valid or polluted content chunks based on the content name only. As a result, a CCN end-user can specify additional security parameters in an interest such as a *publisher's public key digest* (PPKD) or a content digest along with the content name. However, no framework exists for end-users to learn this information prior to requesting content. The CCN-key resolution service (CCN-KRS) aims to solve this problem. It is a DNS-like service for resolving authorised content publisher security information such as the public key certificate and content digests for a CCN namespace [Mahadevan et al., 2014]. CCN-KRS guarantees (assuming network and cache compliance) an end-user will receive content matching the name and public key requested, which can be verified against the digital certificate received from KRS.

Key Resolution Service (KRS)

The requirements of a global DNS-like key resolution service for CCN include [Mahadevan et al., 2014]: It must be secure; scalable to 10^{14} content names; have a fast response time, with similar performance to DNS; flexibly resolve a content name to a content hash, publisher certificate, public key certificate chain or a future parameter; operate transparently to end-users; and be easily discoverable. As KRS and DNS have similar requirements, KRS adopts DNS-like components in its design. This includes using *local KRS* and *authoritative KRS* servers. Moreover, KRS uses CCN interests and digitally signed content objects (i.e.

KRS records) for all communication. Local KRS servers can be discovered transparently in the network by end-users using a DHCP-like CCN service or using a predictable/predetermined namespace such as */krs*.

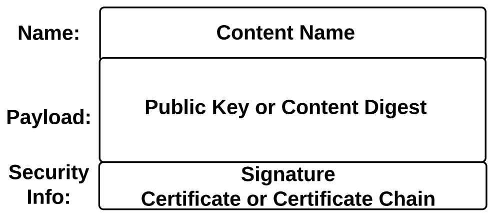

Name:
Content Name
Payload:
Public Key or Content Digest
Security Info:
Signature
Certificate or Certificate Chain

Figure 2.2: KRS record [adapted from [Mahadevan et al., 2014]]

A KRS record is a published CCN content object composed of (see Figure 2.2): a content name, which matches the interest received; payload, which contains security information associated with the KRS request i.e. content providers public key or content digest; signature for the KRS content object produced by the KRS server; and a certificate chain of trust, which is anchored at a globally trusted entity. After a client receives a KRS record for a content object and verifies its integrity and authenticity, it can specify the content name along with the publishers public key digest or content digest in all future interests.

For scalability purposes a KRS zone is assigned a name prefix to manage. Like DNS, each zone has one or more authoritative KRS servers, which store and manage KRS records associated with a name prefix and sub-prefixes. Requests that cannot be resolved locally (or by cached KRS records) are forwarded to the KRS authoritative zone server or to a root or top level domain (TLD) server if the authoritative zone is unknown. As CCN names are hierarchical, KRS servers use longest prefix-matching (LPM) algorithms when looking for the next hop KRS server to send a request. LPM also supports a KRS server resolving cached KRS record requests for sub-prefixes (e.g., /tcd/cs) to an authoritative key for that prefix (e.g., /tcd). Consequently, KRS record requests for the same or similar keys can be resolved quickly.

To support CCN principles, if an intermediate KRS server receives a KRS record from an authoritative KRS server, it needs to decapsulate the record before re-encapsulating it as a response to interest received. For trust and authentication purposes, all KRS servers and end-users must be able to trace a KRS server's certificate chain to a trusted entity i.e. a global trusted authority. Additionally, it is envisioned that KRS will adopt a federated methodology similar to DNS where publishers use a *registrar*, which has naming and certification authority for top level domains. This registrar is accredited by a global entity like ICANN, which coordinates namespace allocation and manages KRS root servers.

The model of trust used by CCN-KRS assumes KRS servers form a chain of trust that anchors at a global trusted authority. After checking a KRS records integrity and authenticity, a client can specify the publishers public key digest or content digest along with content name for all interests sent. Specifying this information in an interest guarantees (assuming network and cache compliance) an end-user will not receive fake or incorrect content from the network.

2.3.7 Authenticated Interests

The concept of *authenticated interests* was originally proposed to increase security across IP networks for building automation systems (BAS) [Burke et al., 2012]. The aim was to provide a simple, secure, access controlled and low-latency framework for communicating between controllers and low-powered lighting fixtures in untrusted IP networks. It was demonstrated using NDN, a popular implementation of CCN principles, by the NDN lighting control system [Burke et al., 2013][Burke et al., 2012].

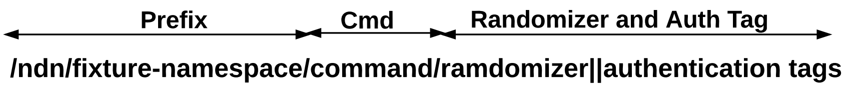

Prefix
Cmd
Randomizer and Auth Tag
/ndn/fixture-namespace/command/ramdomizer||authentication tags

Figure 2.3: Authenticated Interest [adapted from [Burke et al., 2013]]

Authenticated interests add commands and authentication tags, i.e. digital signatures or message authentication codes (MACs), to NDN interests [Burke et al., 2013]. They are composed of three parts, as illustrated in Figure 2.3. The *prefix* is used for routing, a fixture specific *command* and a *randomizer* (nonce, timestamp, and so forth), which is concatenated with an *authentication tag* computed over the rest of the interest. Authentication tags use nonce, timestamps and estimated RTT to ensure uniqueness and prevent timing and replay attacks. Commands are transparent to NDN routing infrastructure and can be interpreted and executed by the destination infrastructure. Furthermore, commands and acknowledgements can be encrypted by either the receivers public key or a shared symmetric key.

NDN Lighting Control System

The NDN lighting control system, which demonstrates a concrete NDN-based security architecture for BAS systems, has four components [Burke et al., 2012]: configuration manager (CM), fixture (Fix), application (App) and authorization manager (AM). The AM manages access control, public/private key pair generation and acts as a trusted third party (TTP) or root of trust for all keys. All lighting fixtures (Fix's) are assigned a specific namespace, a public key owned by the AM and access control permissions at initial start-up by the CM. A long-term secret key is also generated by the Fix at this time to support optional use of application-specific symmetric keys. For performance reasons, the AM can communicate the signing key pair to the Fix. Additionally, a Fix can generate and manage key-pairs under its sub-namespace.

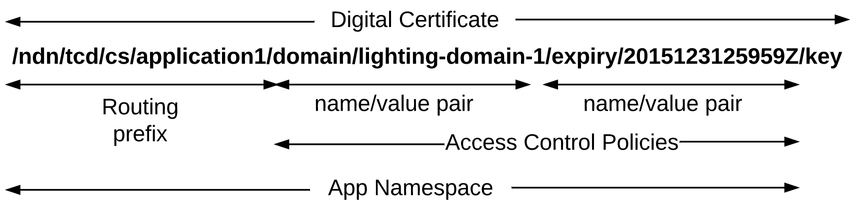


Figure 2.4: Application's namespace, name/value pair access control policies and digital certificate location.

When the AM creates a signing key pair for the App's namespace, it also specifies the App's access control policies using attributes (i.e. *name/value* pairs) such as domain, expiry date, and so forth in the namespace (see domain and expiry date *name/value* pairs in Figure 2.4). The AM associates an App with a namespace, which is composed of the App's prefix and access control policies, by publishing a content object containing the public key under */namespace/key*. This effectively acts as digital certificate (see *App Namespace* and *Key* in Figure 2.4). When an App sends an authenticated interest command to a Fix, it includes its namespace, which contains access control policies defined by the AM.

When an authenticated interest is received by a Fix it: examines if the command is valid; checks the App is allowed run the command i.e. based on *namespace* and access control policies installed at start-up by the CM; inspects the randomizer information; verifies authentication tag; executes the command; and finally responds with signed content as an acknowledgement. No specific commands are specified, but they can be as simple as *on* or *off* or more complex like *"intensity/+10/rgb-8bit-color/F0FF39"*.

Finally, a content producer (i.e. App or Fix) can prove its ownership of a key with a simple challenge-response protocol. When a challenger sends an interest with random nonce for data, only the content producer, one of its ancestors or a TTP can respond with valid (signed) data.

Auditing and Feedback

When auditing and feedback for accountability purposes is required, digital signatures are recommended [Burke et al., 2012]. However, for performance purposes, MACs are preferred when collecting data from low powered devices such as sensors. Consequently, the NDN lighting control system also proposes a mechanism utilising an application-specific symmetric key, which is generated by Fix and shared with App after verifying its public key, for sending commands and generating auditable acknowledgements supporting accountability [Burke et al., 2013]. This is supported by a hash chain, which is generated (when required) by the Fix or a TTP with input x and length ℓ. An anchor $H^{\ell}(x)$ valid for ℓ signatures is sent to the App as a certificate along with other parameters. For each authenticated interest command sent (assuming a reliable link with no lost packets), App includes the last $H^{i}(x)$ received from Fix. Fix's acknowledgement must contain $H^{i-1}(x)$ to ensure accountability, which can be easily verified by App.

The model of trust used by the NDN lighting control system assumes a namespace prefix is bound to a public-key signature, which is issued by a TTP or an ancestor. Consequently, an entity can only publish in its namespace or the namespace of its children.

2.3.8 Summary

The analysis of these security mechanisms leads to two important conclusions. First, all ICNs regardless of the naming scheme utilised, depend on external trusted mechanisms of binding (hierarchical or human readable) names to public keys or flat names respectively. Second, every ICN requires a PKI to manage the life cycle of public keys. This is a required piece of infrastructure for any content distribution model that depends on public key cryptography.

2.4 Accounting Frameworks: State of the Art

In this section, we give a brief overview of several IP and ICN frameworks that seek to maintain visibility and control of network elements and/or data. Our aim is to provide some background into IP-based frameworks, while highlighting the current state of the art in ICNs. These frameworks offer insight into the importance of gathering accounting information related to the content distribution process and understanding some of the requirements of content providers and ISP's. However, the model for accounting typically used by these systems is based on trust i.e. they lack accountability. Consequently, the content provider needs to trust the infrastructure to deliver content with adequate quality, speed and reliability.

2.4.1 SNMP

The Simple Network Management Protocol (SNMP) is a low-overhead protocol standard designed to manage IP-based network devices such as routers, servers, computers, and so forth. Several versions of the protocol were developed. The first IETF RFCs for SNMPv1 started in 1988. SNMPv1 proposes the basic network management components, protocols and operations of the framework, which include [Stallings, 1998]: management station, management agent, SNMP proxy, management information base (MIB), network management protocol and trap-directed polling. All three versions of SNMP utilise these basic concepts.

The *management station* uses a protocol for information exchange and a database containing summary information. These standardised components support human monitoring, control, analysis, fault recovery, and so forth of managed devices. A *management agent* runs on managed devices and responds to the management stations requests and actions. Agents can also send unsolicited information called *traps* to the management station based on local events. Additionally, an agent can act as a proxy to support one or more devices such as modems, bridges and computers that do not implement SNMP agent software. Each agent maintains a collection of standardised objects in a *management information base* (MIB) that are accessible to the management station. Objects are essentially data variables representing some feature of the managed system. The *network management protocol* can *GET* or *SET* objects values and agents can notify the management station of a significant event via a *TRAP* message [Stallings, 1998].

The management station can use the *trap-directed polling* technique to collect key baseline information (e.g., interface characteristics, performance statistics, packets sent/received, and so forth) from agents at periodic intervals (e.g. once daily) [Stallings, 1998]. Agents are responsible for notifying the management station about unusual events (such as a restart, link failure, overload conditions, and so forth) using SNMP *TRAP* messages. The management station can perform further investigation on agents using *GET* or *SET* SNMP messages after notification. The trap-directed polling operation reduces the overhead on agents, the network and the management station.

SNMPv2 (work started in 1993) addressed limitations in the original design, which included the inability to query network entities efficiently for information. This was facilitated by bulk data transfer capabilities in SNMPv2. Moreover, decentralised network management was supported by manager-to-manager cooperation features (e.g., alerts and a manager-to-manager MIB [Stallings, 1998]). The primary contribution of SNMPv3 (work began in 1997) addresses security issues in previous versions related to privacy, authentication and access control. These functions are supported by symmetric-key cryptography and utilise preconfigured access control policies to prevent unauthorised users running commands at agents [Stallings, 1998].

2.4.2 Ccndstatus

The ccnd daemon is the forwarder/router process required for CCNx protocol communication, which must be run by every CCNx node. It is based on an implementation of the CCN/NDN principles outlined in Section 2.2.4. Its primary functions include maintaining a forwarding information base (FIB) table, a pending interest table (PIT) and a content store (CS). The ccndstatus command returns internal state from the local or remote ccnd daemon, which includes information about the FIB, PIT and CS and can help explain its behaviour. It can be run via command line or HTTP request against routers. Information returned includes [PARC, 2015]: ccnd identity (e.g., ID, start time, current time), content object statistics (e.g., stored, stale, duplicate, sent), interests (names, pending, propagating, accepted, sent, dropped), configured interfaces (e.g., ID, IP address, bytes in/out, content objects in/out, pending interests) and forwarding information (e.g., name prefix, ID, expiry time).

However, the requestor has to trust the metrics produced as no accountability model exists in ccndstatus to establish integrity or authenticity of information produced. Moreover, it provides no information about transmission delay, RTT from consumer to content, and so forth. Additionally, commands need to be run against individual routers. As a result of these design decisions, it is difficult to trust metrics produced, trace content paths efficiently and troubleshoot or monitor CCN networks.

2.4.3 Contrace

Contrace is a tool for measuring and tracing content in CCN/NDN networks running over IP [Asaeda et al., 2015]. It aids with performance evaluation, troubleshooting issues, estimating content popularity, investigating routing paths and caching conditions in CCNs. Network tools such as IP's ping and traceroute and multicast IP's traceroute facility inspired its design. These tools cannot directly trace data, forwarding paths (including multipath) or cache status for name prefix's in CCN as they are based on host centric rather than data centric content distribution models. Moreover, similar tools do not currently exist to evaluate CCN networks or protocols. See Figure 2.5 for a high-level overview of contrace.

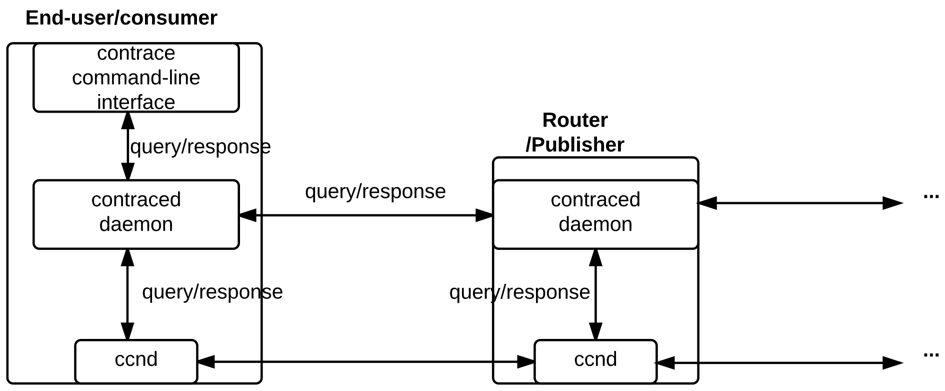


Figure 2.5: The Contrace CCN network analysis tool

A *contraced* daemon runs on every CCNx router and interfaces with the local ccnd daemon, which is a required component for CCNx communication. The ccnd forward information base (FIB) table (see Section 2.2.4) is populated by a name-based routing protocol (e.g., OSPF). In contrace, a *query* message is invoked by a user (via a command-line interface), which is passed to the contraced daemon. The local ccnd process responds to the query request with cache and forwarding path information. The contraced daemon can then forward the request to a neighbour router's contraced daemon, which queries the local ccnd and responds or forwards the request on to the next router. A *response* is generated by the contraced daemon running on the content forwarder and sent back to the user. Router policies defined by the network administrator can limit the information in the response. However, if no policies are defined, cache efficiency, performance and state information captured by contrace includes [Asaeda et al., 2015]:

- Cached content size.
- Number of cached content chunks.
- Amount of received interests for the content (i.e. content popularity).
- Lifetime and expiration time of cached content.
- Node name or IP address of the publisher or NDN cache.
- RTT between content consumer and NDN cache or content publisher.

- Cache state per name prefix in NDN cache.
- Prefix forwarding path information.

Contrace is a powerful network measurement tool that can be used to help design and test new routing protocols and forwarding/caching strategies in CCN/NDN. However, the contrace accounting model depends on trust. It assumes that routers will not return invalid or inflated metrics as they lack accountability mechanisms.

2.4.4 LIRA

This architecture is described in an article published in late 2015 in the arXiv preprints repository, which is a source of eprints of scientific papers. It shares some of the underlying goals of the Savant architecture such as controlling content and collecting accurate accounting information for the content distribution process. To our knowledge it has not been published in any peer reviewed conferences or journals to-date.

The objective of the Location-Independent Routing Layer (LIRA) framework is to support the deployment of ICNs by providing a network architecture that aligns with the goals of content providers and ISP's. LIRA identifies several obstacles hindering ICN deployment by these entities as follows [Psaras et al., 2015]: absence of a scalable name resolution service, lack of content control, no data access logging i.e. content accounting information, and the need for backward compatibility with the current IP Internet model. LIRA achieves these goals using several mechanisms.

First, content providers are directly involved in the name resolution process. This is achieved using the Internet's current standardised location-dependent mechanisms based on URLs, HTTP, DNS and IP addresses. All consumers are required to consult the content provider and *ask* for content name/ID (cID) before content transfer takes place. The content provider responds with the cID (but not content) to the consumer. Additionally, they can also add a list of up-to-date cIDs for subsequent content in the response. This avoids the client having to make requests for every individual content chunk. Consumers use these cIDs to retrieve content from any local cache that supports LIRA principles and has content availability. By requiring all consumers to *ask* the content provider for content, consumers are prevented from accessing content transparently to the content provider facilitating the production of content access logs (i.e. accounting information related to content views).

However, consumers or search engines could distribute known cIDs to other users. To prevent this, LIRA uses flat self-certifying ephemeral names for content that periodically expire (after some time period proportional to content popularity). Changing cIDs invalidates existing copies of content located in network caches, which will be evicted based on a cache replacement or predefined eviction strategy. This effectively enables content providers to purge LIRA caches of old content and actively control content delivered to consumers. Moreover, this mechanism offers more control over content than specifying a TTL value in published content. A TTL that is too long results in the delivery of out-dated content while a TTL that is too short results in the unnecessary redelivery of content.

To support these mechanisms, LIRA transparently adds an extra layer on the protocol stack above the network (IP) layer and below the transport layer. This layer supports the ICN philosophy of a location independent data distribution network. All LIRA nodes implement a *Content Forwarding Information Base* (C-FIB) table and content cache [Psaras et al., 2015]. The C-FIB manages incoming and outgoing content based on name and supports native multi-source routing and off-path cached content delivery (e.g., from neighbour nodes). Moreover, content cached locally (on-path caching) can be delivered directly to consumers. Furthermore, LIRA nodes can be deployed incrementally with backward compatibility for IP using traditional IP location-based routing. Evaluations show a performance gain for ISPs with a subset of LIRA nodes deployed in the network while content providers continue to maintain full control of content.

The LIRA content accounting model requires all consumers to *ask* the content provider for ephemeral data names (that periodically change). Moreover, it needs to trust caches will evict expired content and ISP's, search engines and consumers will not share cIDs with other users.

2.4.5 Encryption-Based and Push Interest Based Accounting in CCN

This architecture was also described in papers published in late 2015 in the arXiv preprints repository. It shares many of the design goals of the Savant architecture such as gathering accounting information for content distributed and the need for accountability in information produced. However, it fails to resolve issues related to per-consumer specific accounting and preventing several types of ICN attacks. In contrast, Savant seeks to specifically address these types of challenges, which are outlined in Chapter 4 and Chapter 5. To our knowledge this architecture has not been published in any peer reviewed conferences or journals to-date.

Encryption-based and push interest (*pInt*) based accounting are two practical secure schemes for gathering real-time feedback information for content distributed in CCN networks (specifically using the CCNx 1.0 protocol). The encryption-based scheme works by encrypting content (a form of access control), which is transparent to networking infrastructure as it works at the application layer. In contrast to the encryption-based scheme, the pInt-based scheme operates at the network-layer requiring router participation and modifications to the CCNx architecture. Both schemes are based on the identification of three types of accounting information. These include [Ghali et al., 2015]:

- Individual: information is bound to a unique consumer e.g., name or public key. User-specific information needs to be added to the interest payload[1].
- Distinct: the same information as individual but identities are not revealed.
- Aggregate: this is an aggregate of the set of unique consumers.

To facilitate probabilistically accurate accounting, the authors recommend that distinguishing information such as nonces and timestamps should be added to interests as payload for all three types of accounting [Ghali et al., 2015].

Encryption-Based Accounting

The encryption-based scheme works by encrypting each content chunk and getting consumers to retrieve decryption keys, which are the same for all users, directly from the content

[1]CCNx 1.0 interests can carry payload information in an interest that is signed by the producer. This is in contrast to the NDN protocol outlined in Section 2.2.4, which does not currently support interest payloads.

producer [Ghali et al., 2015]. This model facilitates per chunk accounting at the application layer by the content producer. To support individual accounting, consumer specific data needs to be specified in interests sent to the content provider for data decryption keys. However, if aggregate or distinct accounting is needed, no additional consumer information is required when requesting decryption keys. While this scheme is transparent to the routing infrastructure, it is based on an access control model. Moreover, it is an inefficient mechanism as at least two interests are sent per content chunk requested i.e. one interest for data and one for decryption keys. Additionally, all content chunks need encrypting by producers and decrypting by consumers.

pInt-Based Accounting

The pInt-based scheme is distinct from access control and requires half the number of messages per content chunk compared to the encryption-based scheme. *pInt messages* are generated by routers to inform the content producer that their content has been requested. This occurs when a cache hit occurs on a router or when *collapsed* (or aggregated) interests in the pending interest table (PIT) are satisfied. To support accounting at router caches, the pInt-based scheme requires a new flag in the content header called *ACCT*. Flag values include [Ghali et al., 2015]: *NONE*, *AGGREGATE*, *DISTINCT* or *INDIVIDUAL* corresponding to the types of accounting available.

A pInt message is the same as a regular interest as it contains interest name and payload information, which corresponds to the name and payload of the interest for data served. However, pInt messages do not leave state behind in routers and interests are not aggregated or multicasted. As a result, a content producer is guaranteed to receive real-time information about content requested in the network (i.e. assuming no network failures, maliciousness, and so forth).

The pInt-based scheme also provides mechanisms to mitigate forgeries and replay attacks for the individual accounting type by authenticating users and cryptographically binding consumer specific data to a distinct interest, a nonce and a timestamp. A producer can detect replayed nonces by maintaining a list of all recently received nonces for a consumer.

Furthermore, the authenticity of public keys and symmetric keys can be established using some distribution mechanism prior to verification.

However, preventing forgery and replay attacks is not possible with aggregate or distinct accounting types as consumer-specific data is required. Additionally, the pInt-based mechanism cannot differentiate between routers not generating pInt messages (when they should), causing inflation attacks in distinct or aggregate accounting types or when genuine network packet loss events occur. The authors conclude that per-consumer specific accounting is not possible without application support when consumers are dishonest, because routers do not verify consumer specific information before distributing content.

The encryption-based and pInt-based accounting mechanisms identify some of the requirements and challenges of designing an accounting and accountability framework for ICN architectures.

2.4.6 Summary

We have covered a variety of systems that seek to maintain visibility and control of network elements or data in IP and ICN environments. Many of these systems were designed with a specific purpose (e.g., network measurement), which is achieved by gathering accounting information. We began with an overview of SNMP, which is used to maintain and control IP-based network devices. Ccndstatus offers similar information to SNMP (but no control) and is limited to querying one CCNx network device at a time. Contrace extends ccndstatus functionality by measuring and tracing content in CCN/NDN networks running over IP. LIRA uses self-certifying ephemeral names for content that periodically expires to support content control and content accounting. The encryption-based and pInt-based accounting schemes offer two practical secure frameworks for gathering real-time accounting information for content distributed in CCN networks. The pInt-based scheme highlights the importance of gathering accountability information for content distributed in untrusted environments.

However, many of these architectures lack proper support mechanisms to avoid new and legacy types of attacks in ICNs as identified in Section 2.3.4. Moreover, if integrity, authenticity and non-repudiation can be established in accounting information produced for content distributed in trusted or untrusted environments in near real-time and in a scalable manner, it would be a useful tool or to support the requirements of content providers, ISPs and end-users. To support this near real-time capability, an architecture would require the support of a scalable information processing and control system.

2.5 Information Processing Systems

The collection of accounting information for analytics purposes is common in many content distribution architectures such as P2P, fixed-infrastructure CDN, hybrid CDN-P2P and network CDN (NCDN) systems [Bitar et al., 2012] [Androutsellis-Theotokis & Spinellis, 2004][Nygren et al., 2010][Repantis et al., 2010]. The information gathered by these systems is used for monitoring and alerting on distributed system and network health, content location and access logging, user engagement and demographics, and so forth. This information is typically presented in raw, aggregated and summarised form based on locally or globally defined metrics. Moreover, with the advent of big data platforms large volumes of accounting information can now be processed in real-time (or near real-time). The mechanisms and systems to perform these actions have evolved over the last fifteen years, some of which we discuss in the following section. Moreover, many of the attributes from architectures discussed in this section have been used to support the Savant framework. We begin with a brief overview of the SDIMS system, which provides some background into distributed aggregation frameworks that can operate in untrusted P2P environments.

2.5.1 Scalable Distributed Information Management System (SDIMS)

The Scalable Distributed Information Management System (SDIMS) is a hierarchical data aggregation framework for large scale distributed networked systems. The SDIMS philosophy is to provide detailed information locally and summary information globally instead of providing all information to all nodes, which supports the architectures scalability. SDIMS

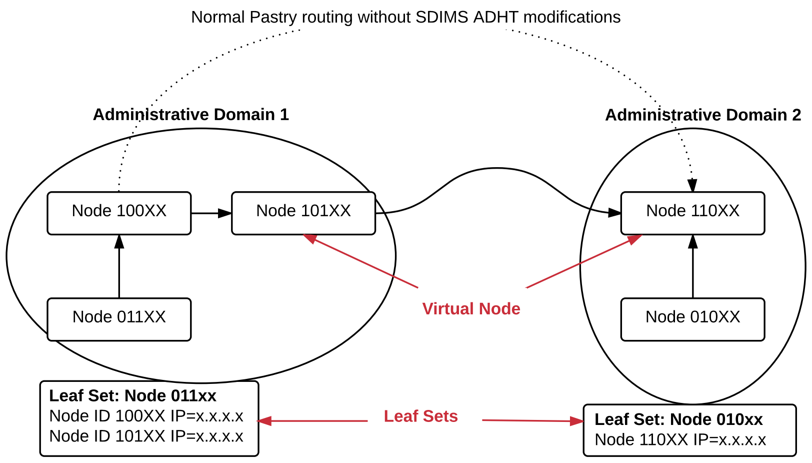


Figure 2.6: SDIMS using the Pastry DHT with support for administrative isolation [adapted from [Yalagandula & Dahlin, 2004]]

achieves these design goals using the Pastry[2] distributed hash table (DHT) with modifications to support local domain awareness using *administrative isolation* and *scalable aggregation trees*.

First, physical nodes are inserted into the Pastry DHT and assigned a globally unique key. Administrative isolation is achieved by modifying Pastry's routing tables so that each node maintains a separate routing table (called a *leaf set*) to identify groups of nodes for each administrative domain (e.g., a university department) they are a member (see *leaf sets* in Figure 2.6). A leaf set might be statically configured by a network administrator.

Second, *scalable aggregation trees* (implemented by every node) store and manage dynamic attribute key values, perform attribute aggregation using aggregation functions and propagate aggregate values to *virtual nodes*[3] in different administrative domains (see Figure 2.7). Each node stores raw attributes as a set of tuples <*attributeType*, *attributeName*, value>. For example:

- <uplink-bitrate, avg-uplink-bitrate, 5mb/s>
- <downlink-bitrate, avg-down-link-bitrate, 10mb/s>

[2]Pastry (outlined briefly in Section 2.1.4) is a structured self-organising P2P overlay network based on DHTs with proximity awareness (i.e. local or near by nodes) [Lua et al., 2005]).

[3]A *virtual node* acts as a convergence point or *root node* for an administrative domain. The virtual node is selected by finding the numerically *highest* node in the administrative domain (see *virtual nodes in Figure 2.6*). The virtual node in an administrative domain maintains aggregate values representative of the local domain and aggregate data received from other administrative domains. An abstract view of this configuration is depicted in Figure 2.7.

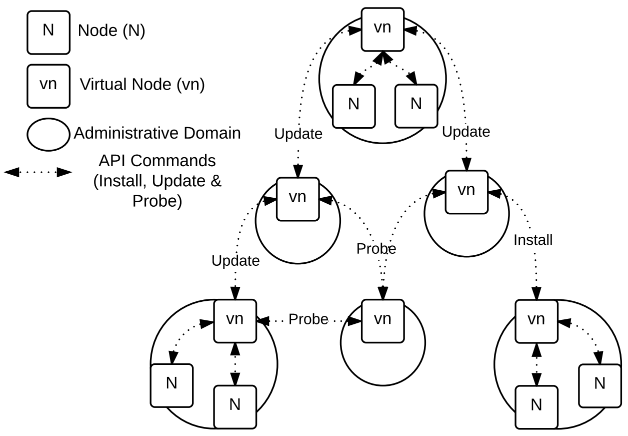


Figure 2.7: The SDIMS API runs Install, Update and Probe commands across different Administrative Domains for aggregation and propagation purposes.

- <buffering, avg-buffering-events, 2>

Aggregation takes place globally on *attributeType* values and locally on *attributeKey* values (where attributeKey is a secure hash function (such as SHA-1) over <attributeType, attributeName>) [Yalagandula & Dahlin, 2004]. Each node in the DHT has an obligation to manage aggregation for values that map to its key. Moreover, every node in an administrative domain will have some responsibility for performing data aggregation.

To support flexible distributed aggregate computation and propagation of aggregate values between administrative domains in the DHT, SDIMS provides an API for *installing*[4] aggregation functions and *updating* and *probing* aggregate information. This is depicted in Figure 2.7. These functions can be invoked by any of the nodes in the tree. After a node applies an aggregate value update e.g., adding a raw attribute update for buffering interruptions during a video session, the SDIMS aggregation API may trigger re-computation of aggregate values up the tree and down different sub-trees using the *update* function. These functions enable applications to monitor, query and react to state changes or issues in the distributed system.

Finally, SDIMS only guarantees eventual consistencies for large distributed systems as aggregate values are not recalculated each time a global probe occur.

[4]How far *up* or *down* different subtrees an *update* will propagate depends on the install parameters for each aggregation function. Install parameters include: attributeType, aggregation function, what to update (e.g., all nodes), domain restriction information and expiry time.

2.5.2 *Query* System

Query is the monitoring system for Akamai's EdgePlatform, which is a purpose built CDN of edge servers and optimised protocols that offers accelerated delivery for applications and content to end-users. *Query* provides near real-time information about Akamai's globally distributed network and services, which handle between 15% to 30% of the Internet's traffic each day [F. Chen et al., 2015]. It includes over 170,000 servers located in 102 countries and 1,300 autonomous systems running over one million software components [F. Chen et al., 2015][J. Cohen et al., 2010]. It has been in constant development and use since Akamai's inception in 1998 supporting user services such as live or on-demand media streaming, static and dynamic HTTP content, and so forth.

Query's design goals include [Repantis et al., 2010]: low latency for data and queries, scalable, reliable, consistent, fault-tolerant, with complete and synchronised data. These goals are achieved using several mechanisms and components in a hierarchical architecture, which is depicted in Figure 2.8. First, a *Query process* runs on every Akamai owned and managed machine i.e. not end-user machines, which collects and combines metrics from local software components into database tables [J. Cohen et al., 2010]. A set of tables collected at short intervals (i.e. every minute or two depending on configuration) is called a *generation*. Generations are collected and combined into tables by *cluster proxies*, which process information from all edge machines in a geographical location or *cluster*. Cluster proxies provide combined data to *top-level aggregators* (TLAs). TLAs collect and aggregate all generations from portions or the entire network, which are again added to database tables. TLAs send copies of these tables to *SQL parsers*, which compute and answer user queries.

Data is accessible to users via SQL queries. All queries are executed immediately or periodically, depending on the availability of tables in SQL parsers. If data is needed regularly, machines *prewarm* (or cache) tables based on a predefined list of required data. This information is collected, combined and aggregated as part of a generation. If some tables don not exist in the SQL parser, a request is sent to the TLA for the tables. If tables are still missing, data is requested from cluster proxies and then from edge machines. Tables missing at all layers will be delivered during the succeeding generation. This supports the architecture's

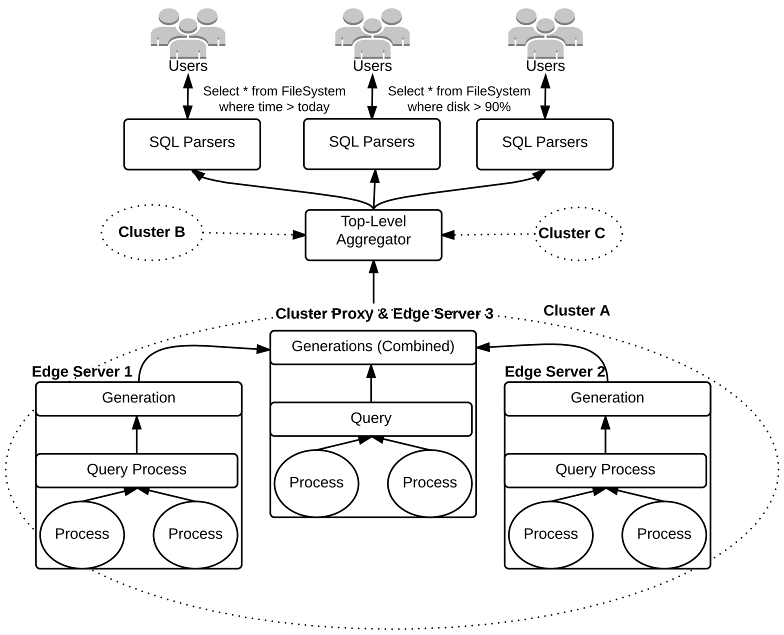


Figure 2.8: Akamai's *Query* System [adapted from [J. Cohen et al., 2010]]

scalability, as only required data and tables are collected, combined and aggregated.

It takes *Query* about two to three minutes to collect, decode and aggregate data from edge machines and offer a *best-effort* synchronized view of Akamai's network and services [Repantis et al., 2010]. Users and applications that use *Query* include [J. Cohen et al., 2010]: Akamai's alert system, which monitors and alarms on servers, outages, and so forth; historical data collection, which is used to track quantitative metrics over time; Akamai's customers (i.e. the content providers) providing estimates of recent traffic patterns for content delivered.

Data consistency, completeness and synchronisation are *best effort*. Moreover, data latency can be up to a few minutes. Finally, these resources are highly replicated for scalability, reliability and fault tolerance purposes

2.5.3 Google Analytics

Google Analytics is a service that collects and measures website traffic data and provides a tool to study a user's online experience. Information collected includes [Clifton, 2012]: traffic volumes, geographic distribution, latency, user behaviour, social media traffic, and so forth. Google acquired Urchin Software Corp in April 2005, which developed a web statistics software program called Urchin to analyse server log files. Urchin went on to become Google Analytics, which was released in November 2005. However, the architecture has evolved significantly since Urchin. As well as providing analytics for web traffic based on predefined queries, it has also developed into a big data analytics platform. Figure 2.9 shows an illustration of the Google Analytics architecture.

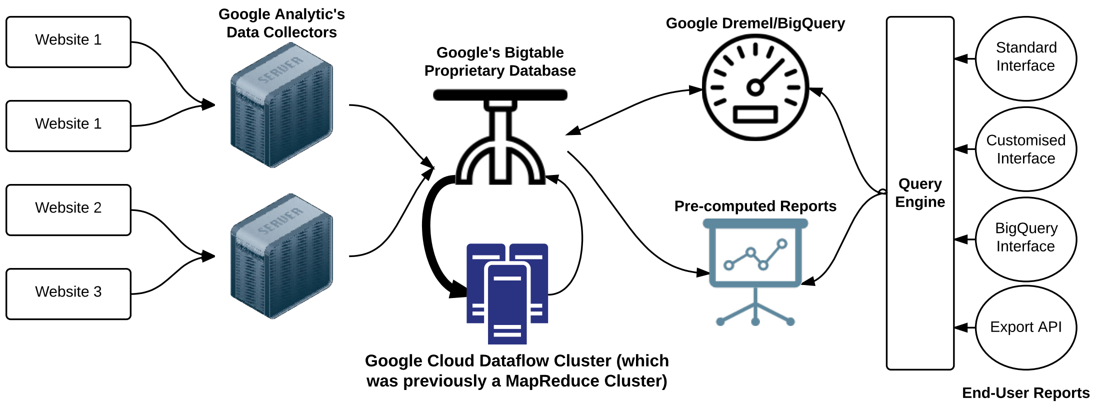


Figure 2.9: Google Analytics architecture [adapted from [Clifton, 2012]]

The architecture is composed of several components to support data collection, scalability and big data. First, JavaScript code needs to be installed on every web page to enable the collection and reporting of visitor interactions. Second, the Google Analytics JavaScript library (analytics.js) needs to be downloaded by all end-users accessing the website. Additionally, JavaScript needs to be enabled on the end-user's browser, otherwise metrics will not be sent to Google's collection servers. Third, all information sent to the collection servers is stored in Bigtable (see Figure 2.9), which is an internal distributed (database-like) storage system (in development at Google since 2004) designed for scalability and high performance [Chang et al., 2008]. Fourth, Google Analytics uses two tables in Bigtable. One is associated with raw clicks, 200TB in size, while the other is a summary table, 20 TB in size, containing predefined summaries for each website [Chang et al., 2008]. In the past, the summary table was generated from the raw clicks table by periodically scheduled MapReduce functions. MapReduce is a parallel programming model for batch processing and generating large datasets [Chang et al., 2008]. However, it was announced in June 2014 that MapReduce has been completely abandoned by Google in favour of *Cloud Dataflow* [Salapura et al., 2015]. Like MapReduce, the Dataflow processing model, hides the complexity of large scale distributed processing from users (such as coordinating workers, data sets, dealing with failures, and so forth) by providing an easy to use programming model allowing users to focus on the logical composition of the data processing job [Akidau et al., 2015]. However, unlike MapReduce, the Dataflow model can easily process petabytes of data, which caused a performance issue for MapReduce, deal with both batch processing and ad hoc streaming of unsorted data, and offer real-time monitoring capabilities [Akidau et al., 2015]. This process is complemented by *Dremel* (BigQuery is the public implementation of Dremel), which is a scalable distributed massively parallel interactive query system that can run aggregation queries against very large data sets over shared clusters in seconds (e.g., trillion row tables) [Melnik et al., 2010]. Finally, data is made available to users via the Google Analytics Query Engine.

2.5.4 C3

Conviva (briefly mentioned in Section 2.1.4), was founded in 2006 and develops tools that support online video optimization, performance and analytics for content providers and end-users. Conviva's C3 controller is a centralised scalable network model for optimising video delivery on the Internet. It uses global knowledge and a real-time view of network state as well as content provider policies and objectives to choose the optimal CDN server for content delivery to a client. Additionally, it seeks to detect and resolve video QoE issues experienced by users such as start-up delays, low bitrate, buffering, frame dropping, and so forth in near real-time.

The C3 controller evolved in three phases [Ganjam et al., 2015]. The first phase started in 2006 when P2P content distribution was a cheap alternative to CDNs, which typically cost in the region of 40 cents/GB. The primary goal of the centralised C3 controller was to compute optimal overlay trees (based on client location, volume, churn, and so forth) to deliver CDN quality live video streams over P2P. In this stage clients were largely homogenous (e.g., desktop machines) and utilised the Flash/RTMP protocol. Additionally, the number of clients was relatively small (tens of thousands).

The second phase started in 2009 with the arrival of cheaper CDN distribution (5 cents/GB), the entry of big players such as Apple and Hulu (seeing the potential for video distribution)

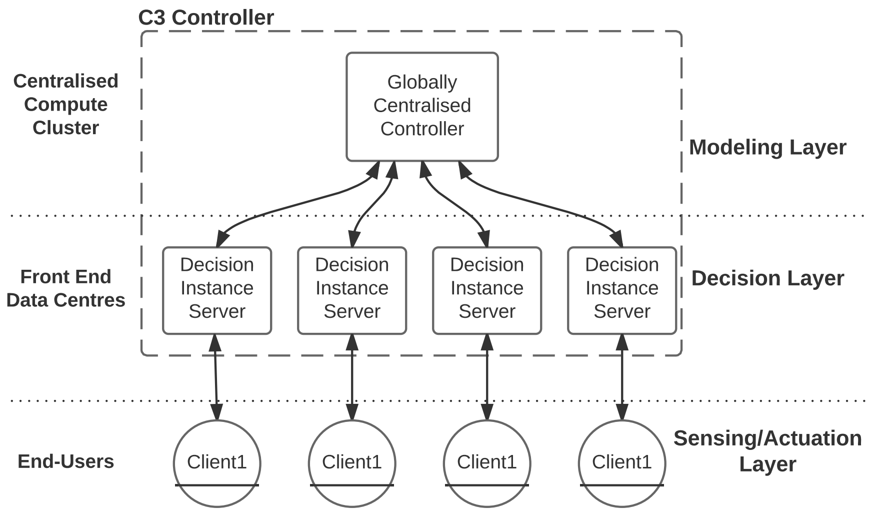


Figure 2.10: The Conviva Phase III C3 architecture [adapted from [Ganjam et al., 2015]]

and the emergence of HTTP and chunk-based video streaming protocols. The centralised C3 controller evolved to calculate the optimal CDN and bitrate for a client. This was achieved using a player plugin, which was downloaded by the client when the session started allowing C3 to modify the player logic and select the best CDN server. However, later adaptations during the video session (such as bitrate switching) could not be supported by C3 and relied on local client adaptation logic. Consequently, there was a lot of QoE metric computation and summarisation logic defined in the player plugin.

The third phase started in 2011 and saw a massive increase in client heterogeneity, which involved supporting different protocols (e.g., proprietary and HTTP chunking), devices (e.g., mobile, set-top-box, TV), application frameworks (e.g., Akamai, Ooyala, PrimeTime) and the emergence of big data platforms. These trends and advances supported the latest development of C3, a diagram of which is available in Figure 2.10. First, the QoE data computation and summarisation logic moved from the client (in phase two) to the C3 controller, which was supported by a big data platform e.g., Apache Spark [Zaharia et al., 2012]. The C3 *sensing/actuation layer*, which runs on the client, provides several functions: it sends raw video quality metrics to the C3 controller; it receives and implements decisions from the controller e.g., change CDN or increase/decrease frequency of metric collection; and it has built in fault tolerance if connectivity is lost to the C3 controller. Second, the C3 controller was split into two layers [Ganjam et al., 2015]: the centralised *modelling layer* offers a global view of the network state, it's decisions are pushed to a scalable globally distributed *decision layer* that operates on a per-client basis. The global modelling layer operates with stale information i.e. up to tens of seconds or minutes, based on feedback metrics from clients. This layer operates based on the premise that CDN metrics are stable for minutes at a time [Ganjam et al., 2015]. The decision layer, which is horizontally scalable up to 100s of millions of users, operates geographically close to clients performing calculations and updates on a per-client basis in milliseconds. The decision layer combines information from: the global (stale) modelling layer; up-to-date client metrics received, aggregated and monitored at intervals of 5-20 seconds; and global policies to make optimal CDN and bitrate selections for clients. These mechanisms support a scalable, responsive and up-to-date view of the global network state.

The architecture provides scalability by centralising the global model layer, which has a stale picture of network state and using a distributed layer located close to end-users to provide decisions quickly to clients.

2.5.5 Summary

In this section, we give a brief overview of several information processing and aggregation frameworks based on a centralised or decentralised models. SDIMS provides some background into distributed aggregation frameworks that can operate in untrusted P2P environments. Akamai's *Query* system uses principles of hierarchical aggregation from SDIMS, collecting near real-time information about Akamai's globally distributed network and services. In contrast to SDIMS which can operate on untrusted infrastructure, all information collected and aggregated by *Query* takes place on trusted infrastructure. Google Analytics provides a centralised platform that collects data from untrusted client infrastructure, which is aggregated and presented to content providers. Similarly, Conviva gathers client-side metrics that support online video optimization and performance. This is supported with a centralised global model layer that operates with stale information and a horizontally scalable decision layer that operates on a per-client basis in milliseconds and is located geographically close to clients.

The advent of big data analytics supports the collection and processing of near real-time information from end-users. However, all of these systems depend a model of trust, which assumes perfect integrity of the accounting information received for aggregation. Moreover, it is typically based on the observation of one entity (i.e. the end-user).

2.6 Chapter Summary

This chapter introduces a number of systems and concepts related to content distribution, security, accounting, accountability and information processing. Each section deals with a specific aspect of research we believe is required to support effective accounting and accountability for content distribution in ICN environments. We use many of these systems, concepts and methodologies to support the design of the Savant framework in the following chapters.

Chapter 3

Tools for Efficient Content Delivery

In Section 2.1 and 2.2, we surveyed existing and proposed future content delivery architectures detailing their methodologies for providing efficient low-cost content distribution, accounting and accountability across trustworthy and untrustworthy infrastructure. We also surveyed several architectures and systems to help identify desirable elements that can lead to reduced network traffic, less network congestion, lower distribution costs and smaller origin server load. Moreover, we identified which architectures and frameworks increase content availability, scalability, content control, security, accountability and that can help manage accounting information produced.

In this chapter, we develop and use two tools to help analyse the drawbacks and merits of the content distribution architectures discussed in Section 2.1 and 2.2. The first is a generic model for content distribution, developed by synthesising the most desirable elements that lead to efficient low-cost distribution of content. The second is a taxonomy for analytic information based on a survey of the logging information gathered by existing systems. Using these tools in combination with the other systems, mechanisms and frameworks surveyed in Chapter 2, we identify a table of elements required for efficient content distribution. Finally, we highlight outstanding challenges for content distribution.

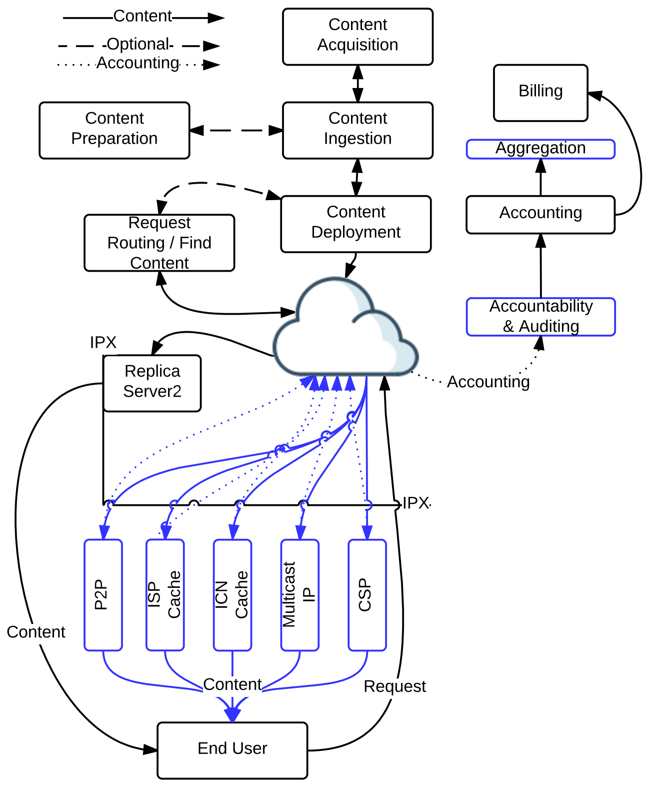


Figure 3.1: Generic Content Distribution Model: components in black are adapted from a high level functional CDN model [ETSI, 2011] while the blue components, which are optional, were added for the Generic Model

3.1 A Generic Model for Content Distribution

The generic content distribution model in Figure 3.1 is a synthesis of the most important elements in existing content distribution architectures discussed in chapter two. We have used a high level functional CDN architecture model [ETSI, 2011] (with components for content acquisition, ingestion, preparation, deployment, request routing, replica servers, accounting and billing) as the baseline for our generic model. This is due to the CDNs prevalence and success for content distribution on the Internet [Nygren et al., 2010][Pathan & Buyya, 2008]. However, the generic model adds the functional elements: accountability, ISP cache, P2P, cloud service provider (CSP), ICN cache (store-and-forward) and multicast IP to complement existing CDN elements. Consequently, the generic models functional elements identify more efficient and cost effective methods of content delivery than the traditional CDN model [ETSI, 2011]. Elements of the generic model utilised by existing content distribution architectures are summarised at the end of this chapter in Table 3.1.

Overview of the Generic Model Components

The generic model uses the following elements for content delivery, which can be mapped to one or more functional CDN activities [ETSI, 2011]. User generated content (UGC) and commercial licensed content acquired from content providers is uploaded into the content delivery system. The ingestion element prepares content for distribution to many different users, devices and networks, preforming tasks such as transcoding, resolution conversion, encryption and incorporating publisher controls. Ingested content is then pushed or pulled to CDN replica servers based on content popularity. In the generic model this information can also be pushed or pulled to ISP cache, ICN cache, CSP, multicast IP and P2P systems. The request routing element interacts with the deployment element to keep an updated view of content availability across CDN replicas, ISP cache, multicast IP, CSP and P2P infrastructure. Request routing is also responsible for directing users to copies of content using adaptive or non-adaptive routing algorithms that depend on heuristics or metrics based on network conditions, server load and server health. The generic model's accounting element is responsible for monitoring events such as user engagement, network and system performance and user demographics, which also provide input for reporting, billing, analytical information and request routing elements. Our model enhances accounting with accountability, which can provide non-repudiability for content accounting information collected from trusted or untrusted infrastructure [Haeberlen et al., 2007].

However, the ability to monitor all the logging information generated across distributed trusted and untrusted infrastructure is a major challenge [Ko et al., 2011][Repantis et al., 2010]. Consequently, a major obstacle is to identify what accounting and accountability information should be gathered based on the content distribution infrastructure (trusted or untrusted)? In the next section we develop a taxonomy of accounting and accountability based on information gathered by existing content distribution architectures.

3.2 Taxonomy of Accounting and Accountability

The following taxonomy is based on our analysis of the accounting and accountability information collected from existing content distribution systems: multicast IP, web cache, CDN, P2P, transparent cache, CDN-P2P, NCDN, CDNI, virtual environments and ICN systems. We have grouped the information into the following categories by purpose. Each sub-category is likely to be of special interest to sub-groups within an organisation such as marketing, IT and customers.

- **Server Configuration and Performance**: Details of system information relating to the distributing server or peer. For example, CPU-type, available RAM, RAM-occupancy, IP address, open ports and port numbers, CPU usage, OS version, temperature, power usage, content location, content availability, and so forth.

- **Network Performance**: Information about the performance of the network when delivering the requested content. For example, bytes uploaded, bytes downloaded, total bytes transferred, speed, reliability, round-trip time (RTT), latency, and so forth.

- **Advertising**: Record of ads pushed to user. For example, impressions (ad displayed to user), cost per impression, cost per click, click through rate, gross revenue, ad type (e.g., reserved in-stream ads (non-skippable), banner ads, auction ads, and so forth) [Clifton, 2012].

- **User Engagement**: User involvement and interaction with the content. For example, unique-views (i.e. distinct individuals requesting content), subscribers (users who prearrange access to a service), total play time (minutes watched), content marked as favourite, user likes and dislikes, user comments, user shares e.g., via Facebook, Google+ and Blogger [Clifton, 2012][Pathan & Buyya, 2008].

- **User Quality of Experience (QoE)**: Measuring the kind of experience the user has with the content. For example [Dobrian et al., 2013]: join time (i.e. time it takes for the buffer to fill up and start playing), buffering ratio - percentage of session time (i.e. playing time + buffering time) spent buffering, rate of buffering events (i.e. frequency of buffering interruptions), average bitrate (e.g.,100 kbps), rendering quality and rate of bitrate switching.

- **User Demographics**: Data characterising users. For example, age, location (by country, city, ISP), gender, device type (computer, tablet, mobile, game console, TV) and operating system.
- **State Machine**: A log of the sequence of events (deterministic or non-deterministic) that a node is expected to follow (based on a protocol) when sending, receiving and processing messages between CDN systems [Haeberlen et al., 2006][Haeberlen et al., 2007].

Organisationally sub-groups have different motivations for analysing information in each of the taxonomy categories. For example, information gathered by the server configuration and performance and network performance categories can be useful for engineering teams to determine if the system and network are performing adequately and have enough resources to scale. Similarly, marketing teams are interested in analytical information related to advertising, user engagement and user demographics to analyse user behaviour. There are also other factors determining sub-group interest in these accountability categories. For example, content producers might be interested in network performance and user QoE if the distribution infrastructure is untrusted e.g., P2P.

3.3 Outstanding Challenges for Content Distribution

The objective of Chapter 2 was to illustrate the value that can be gained by using alternative architectures (trusted and untrusted) for content distribution to reduce costs and increase performance to end-users, while continuing to provide adequate levels of accounting and accountability information. However, many of the systems discussed are research systems (such as ESM [Chu et al., 2002], SplitStream [Castro et al., 2003], CoralCDN [Freedman, 2010] and all ICN systems [Xylomenos et al., 2014]) with no paying customers and as a consequence have a tendency to lack accounting and accountability capabilities. However, one

Table 3.1: Elements for content distribution based on existing and proposed future Internet architectures

Element	P2P	CDN	TC	CDN-P2P	NCDN & CDNI	ICN
Content Acquisition	x	x	x	x	x	x
Content Ingestion	x	x	x	x	x	x
Content Preparation	x	x	x	x	x	x
Content Deployment	x	x	-	x	x	x
Routing	x	x	x	x	x	x
Accountability	x	x	-	x	-	-
Accounting	x	x	-	x	x	x
Accounting Aggregation	-	x	-	x	x	-
Billing	-	x	-	x	x	x
Security	-	x	-	x	x	x
Use Replica Servers	-	x	-	x	x	x
Use P2P	x	x	x	x	-	x
Use ISP Cache	x	-	x	-	x	x
Use ICN cache	-	-	-	-	-	x
Use multicast IP	-	-	-	-	-	x
Use CSP	-	x	x	x	x	x

Note: P2P=Peer-to-Peer; CDN=Content Distribution Network; TC=Transparent Cache; CDN-P2P=Hybrid CDN-P2P; NCDN&CDNI=Network CDN and CDN Interconnection; ICN = Information Centric Networking; CSP=Cloud Service Provider.

of the goals of Chapter 2 was to identify desirable elements for content distribution, which can be used for the design and development for future content distribution architectures. To this end, we created the generic model (see Figure 3.1), which combines the advantageous elements from each architecture identified in Section 2.1 for efficient low-cost content distribution. Many of the architectures discussed utilise one or more generic model elements. However, no architecture provides support for the whole model (see Table 3.1 and Table 3.2).

Table 3.2: Summary of Architecture Attributes Surveyed in Chapter 2

Attribute	Napster	BitTorrent	Amakai	Kad	CoralCDN	YouTube	ESM	SplitStream	PPLive	LiveSky	SAAR	UltraBand	OverCache	NCDN	CDNI	Virtualisation	ICNs
Trustworthy Accounting	-	-	-	x	-	x	-	-	-	-	-	-	-	-	-	x	-
Span Trust Domains	x	x	x	-	-	-	x	x	x	x	x	x	x	x	x	x	x
Decentralised	-	-	x	-	-	-	-	-	-	-	-	-	-	x	x	-	x
Data Integrity	-	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	x
Data Provenance	-	-	-	x	x	x	-	-	-	-	-	-	-	-	-	-	x
Scalable	x	x	-	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Content Availability	x	x	-	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Proximity Awareness	-	-	-	x	x	x	-	x	x	x	x	x	x	x	x	x	x
Transient Nodes	x	x	x	-	-	-	x	x	x	x	x	-	-	-	-	-	-
Inexpensive	x	x	x	-	x	-	x	x	x	x	x	x	x	x	x	x	x
Searchable	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Guaranteed (Content) Control	-	-	-	x	-	x	x	x	x	x	x	x	x	x	x	x	-
Guaranteed Quality	-	-	-	x	-	x	x	x	x	x	x	x	x	x	x	x	x
Guaranteed Reliability	-	-	-	x	-	x	x	x	x	x	x	x	x	x	x	x	x
Guaranteed Performance	-	-	-	x	-	x	x	x	x	x	x	x	x	x	x	x	x
Use ISP Cache	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	x
Use P2P Cache	x	x	x	-	-	-	x	x	x	x	x	x	x	-	-	-	x
Use Multicast	-	-	-	-	-	-	x	x	x	x	x	-	-	-	-	-	x

As identified above, centrally managed CDN infrastructure is the most popular content distribution architecture [Nygren et al., 2010] but it is also the most expensive and arguably one of the least efficient [Greenberg et al., 2008][Mathew et al., 2014]. However, the CDN infrastructure offers the best option for the production of trustworthy accounting information by guaranteeing the integrity of the accounting information gathered. Consequently, it is only rational to assume that any future Internet architecture that spans domains of trust must provide the same level of accounting and accountability (or trust) as a CDN. However, CDN logging has its own list of challenges such as requiring trust in the CDN provider, large log size, associated processing costs and poor availability of real-time log information. Consequently, we propose re-evaluating the following research question: how best to provide scalable trustworthy content accountability for accounting information produced on trusted and untrusted infrastructure? Additionally, identifying what taxonomic information to collect to support the content distribution process is also required. Finally, while the production of scalable and reliable accounting information is an issue for most of the architectures surveyed, it is not an inherent problem in either existing or future Internet architectures.

3.4 Savant System Model

Based on the generic model, research question, distributing content that spans trust domains, and the reliable trustworthy accounting requirement, we have developed the following systems model for Savant. It supports the production and collection of trustworthy accounting information in an ICN environment. The model has four main components with five main functions, which are depicted in Figure 3.2. These include:

1. Content Provider: A set of entities that are licensed to sell and make content available for distribution. This involves performing activities described in the generic model (see 3.1) such as content acquisition, ingestion, preparation, deployment, and directing end-users to copies of available content (request routing / find content). Companies that fall into this category include: Netflix, YouTube, HBO, BBC, and so forth.

2. ICN Cache: Popular content is cached in an ISPs network, which can be any trusted

or untrusted ICN component that can cache data, respond to subscriber requests for available content, and deliver data to end-users. The following elements in the generic model correspond to ISP cache components: replica servers, P2P, ISP cache, ICN cache, multicast IP, and CSP. The ICN cache element also collects and sends accounting and accountability information to the accountability engine. Organisations that fall into this category include: Eir, Vodafone, AT&T, BT, and so forth.

3. Subscriber: Content is requested by a subscribers device. A subscriber is any end-user that interacts with content available from a content provider. This component displays and renders delivered data to the end-user. It also reports industry standard QoE metrics associated with the content distribution process to the accountability engine element.

4. Accountability Engine: This element is owned and operated by the content provider. It collects and aggregates data from cache and subscriber entities. It can also send commands (Cmds) or instructions to these elements. For example, commands include: collect additional CPU consumption statistics; increase/reduce the frequency of data collection; instruct a client to download content from an alternative cache resource. These commands are outlined in greater detail in Section 4.2.7.

5. Content Provider: The accountability engine provides summary data back to the content provider. The content provider is responsible for the generic model elements for collecting accounting information and billing customers. They can also send commands to ICN Cache or Subscriber elements via the accountability engine. For example, instruct all subscribers in Ireland to blacklist a certain cache element.

There are several assumptions to make this architecture function. As outlined in Section 2.1, the premise of our work states that content providers want visibility over content distributed for control and analytic's purposes. Based on ideas and principles explored in multicast and web caching in Section 2.1.1, P2P in Section 2.1.2, hybrid CDN-P2P in Section 2.1.4, and transparent caching in Section 2.1.5, we assume that content cached in an ISP's network can be delivered with high quality, reliability and performance, and at low

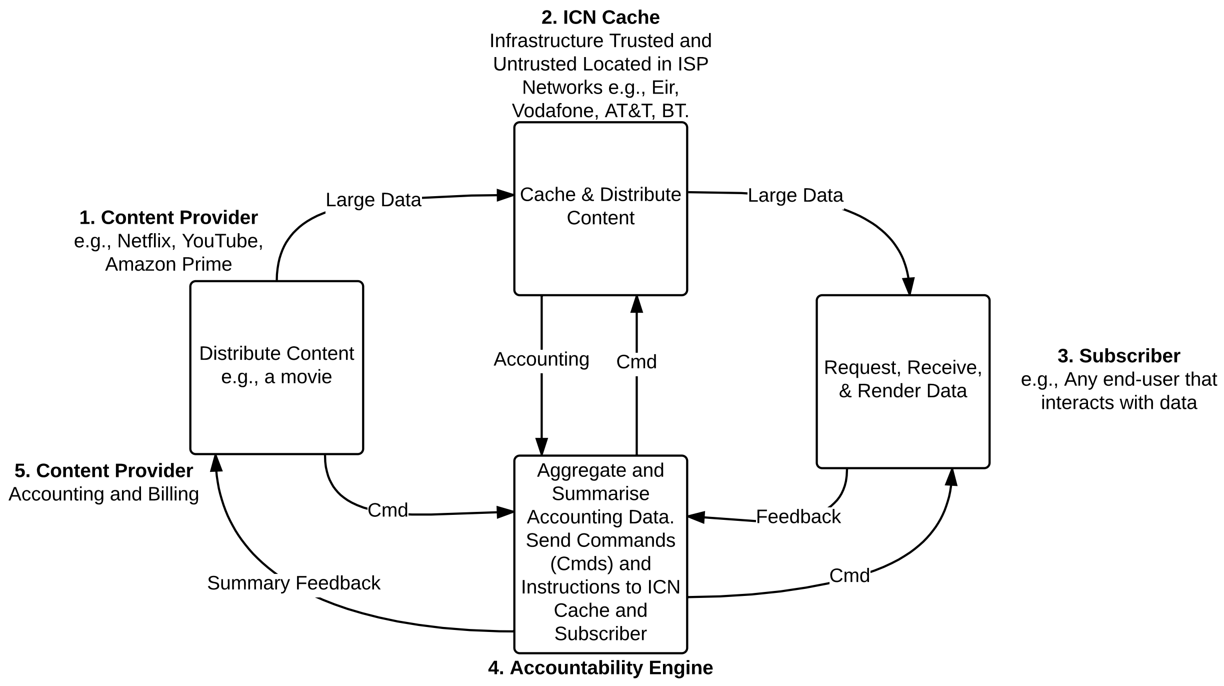

2. ICN Cache
Infrastructure Trusted and Untrusted Located in ISP Networks e.g., Eir, Vodafone, AT&T, BT.
Cache & Distribute Content
1. Content Provider
e.g., Netflix, YouTube, Amazon Prime
Large Data
Large Data
Distribute Content e.g., a movie
Accounting
Cmd
Request, Receive, & Render Data
3. Subscriber
e.g., Any end-user that interacts with data
5. Content Provider
Accounting and Billing
Cmd
Aggregate and Summarise Accounting Data. Send Commands (Cmds) and Instructions to ICN Cache and Subscriber
Feedback
Cmd
Summary Feedback
4. Accountability Engine

Figure 3.2: System System Model: 1. Content Provider, 2. ICN Cache, 3. Subscriber, and 4. Accountability Engine

cost. Additionally, ICN principles are used by cache and subscriber elements to produce accounting information.

The Savant accounting and accountability element provides trustworthy accounting information about the content distribution process to the content provider for content distributed from trusted and untrusted elements. In Chapter 4 we outline how this is achieved with Savant based on ICN principles.

3.5 Chapter Summary

All of the content distribution architectures and systems surveyed in Section 2.1 and 2.2 use at least some combination of desirable elements identified in the generic model for content distribution. Only ICN architectures potentially offer support for the whole model. However, they lack adequate mechanisms to provide accountability for accounting information produced on trusted or untrusted infrastructure. At present, the CDN architecture offers the most comprehensive support for efficient content distribution while also assuring integrity and trust in the accounting information gathered. It is the technology that has been most successful commercially but also the most expensive to use. While this model is adequate for centrally managed and centrally controlled content distribution architectures, it is an inadequate model to use when distributing content from both trusted and untrusted infrastructure. Consequently, we identified a major challenge to solve before all generic model elements can be utilised for efficient content distribution. This involves providing adequate levels of accounting and accountability information during the content distribution process to content providers and content distributors. Two research issues are identified arising from this chapter. The first is how best to provide scalable trustworthy content accountability for content accounting information produced on trusted and untrusted infrastructure? The second is the identification of what information to collect as part of the content distribution process. Finally, the taxonomy and the generic model used in this chapter may be used as foundations for the design and development of future Internet architectures that can utilize trusted and untrusted infrastructure for efficient low-cost content distribution, accounting and accountability.

Chapter 4

Savant Design

In chapter one, we identified the problems we wish to solve and the requirements for the Savant architecture. In chapter two, we surveyed existing content distribution models, architectures and accounting methodologies. Moreover, we identified mechanisms and systems we will utilise in our solution that perform such functions as accountability, data aggregation and security. In chapter three, we synthesised the most important elements in existing content distribution systems identified in Section 2.1 and 2.2 into a generic model for content distribution. Additionally, we developed a taxonomy for analytic information based on a survey of the logging information gathered by these systems.

Based on our analysis in chapter three, we observed that it is difficult to gather reliable accounting and accountability information for architectures that span domains of trust and that lack central administration such as P2P or ICN architectures. We also noticed that there is a correlation between the efficiency of the content distribution architecture and the integrity of accounting information produced. Moreover, there is a relationship between the success of a content distribution architecture and the amount of accounting information it produces. For example, we observed that content providers typically utilise arguably inefficient distribution infrastructure that offers extensive accounting and accountability information for the content distribution process e.g., CDNs.

Consequently, one of the primary philosophies of this thesis is that accounting and accountability information is a fundamental business requirement for many content providers. Moreover, it should be supported in future Internet architectures such as ICNs as they can potentially support the whole generic model by offering efficient low-cost content distribution from trusted or untrusted infrastructure. However, existing ICN architectures do not provide natural support for content accounting and accountability. Instead, many of these architectures make a virtue out of not providing it claiming to offer natural privacy to users. Consequently, Savant is designed for ICN architectures based on the premise that content providers want visibility over the content being distributed primarily for control and analytics purposes. This goal was achieved by combining cryptographic techniques, PKI security processes, ICN principles and information flow processing mechanisms utilised in existing systems to develop a scalable, secure, near real-time accounting and accountability framework for an ICN packet-level content distribution architecture [Haeberlen et al., 2007][Aditya et al., 2012][Cugola & Margara, 2012][Burke et al., 2013] .

This chapter is divided into several sections. First, we identify Savant's architectural requirements. Second, we give an overview of the main components, attributes and processes in the Savant framework. Third, we outline Savant's security and trust model, providing details about threats, attacks and defence mechanisms. Fourth, we provide details of Savant's global and local namespace, which is based on the trust and security models adopted. Fifth, we outline Savant's protocols: agent discovery and local identity management protocol, log commitment protocol and challenge response protocol. Sixth, we give an overview of the extensions and improvements needed to make Savant scalable. Finally, we analyse our solution based on Savant's architectural requirements and conclude the chapter.

4.1 Requirements

We identify several requirements to support accountability in an untrusted accounting environment. These include:

1. Identification of the accounting information to be collected from NDN client and cache infrastructure during content distribution.
2. Provide auditable, tamper-evident and non-repudiable accounting and accountability information for content distributed from trusted and untrusted infrastructure.

3. The architecture must reduce the accounting information overhead in contrast to existing infrastructure such as CDNs. However, it should also guarantee fault detection and the ability to diagnose problems in near real-time. Consequently, the architecture should be able to increase or decrease accounting and accountability collection when required.

4. The architecture must be able to scale both in terms of content distribution and accounting and accountability collection.

5. The architecture should provide a model to prevent security threats and attacks such as polluting data, cache poisoning, sending unauthorised authenticated interest commands, and so forth.

4.2 Overview of The Savant Framework

Existing ICN architectures do not provide organic support for content accounting and accountability. Instead, many of these architectures make a virtue out of not providing it claiming to offer natural privacy to users. Content *accounting* refers to the information tracked by content distributors related to the delivery of content to its intended consumers. This includes content views, end-user quality of experience (QoE), user demographics, and so forth. In contrast, content *accountability* refers to a content providers ability to produce accurate and verifiable information about the content distribution process. This involves the ability to establish authentication, integrity, provenance, auditability and non-repudiation in the accounting information received [Yumerefendi & Chase, 2005]. The primary difference between these concepts is that when trust fails, the latter has the tools to pinpoint the responsible entity with non-repudiable evidence. Consequently, accountability is an important concept when the primary form of communication is between nodes that span domains of trust, which is the case in ICN architectures.

Savant is designed based on the premise that content providers want visibility over the content being distributed primarily for control and analytics purposes. This is achieved by pushing primary responsibility for accounting and accountability out to the NDN clients and NDN caches, collectively referred to as *NDN agents*, sending and receiving content. A high-level overview of the Savant architecture is depicted in Figure 4.1.

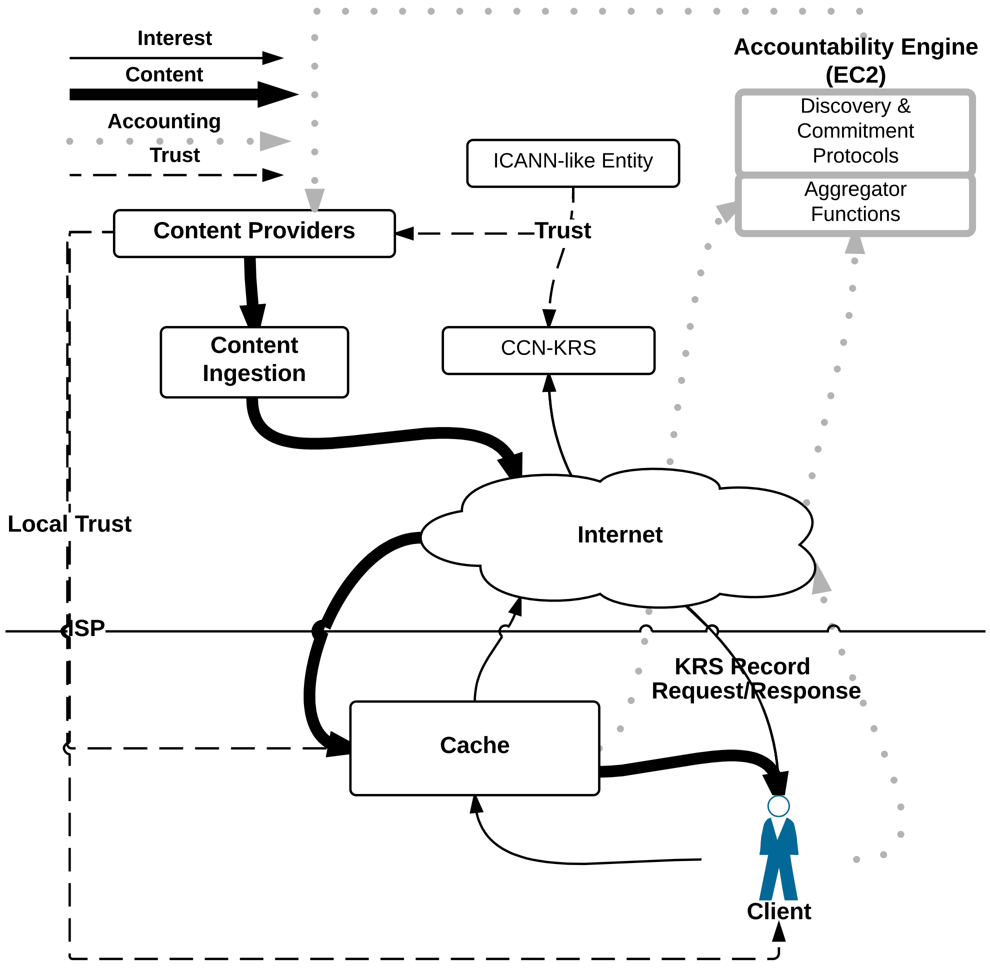


Figure 4.1: High-level Overview of the Savant Architecture

Content providers are responsible for collecting and auditing published log objects associated with content emanating from them. This is supported by a set of *accountability engines*, which are owned and operated by the content provider or a trusted third party. NDN agents are directed to a local accountability engine by the NDN network based on contact information contained in the content *metadata*. This is specified during the content ingestion process. The accountability engine runs *aggregator functions* based on event processing languages to aggregate, summarise and alert upon data collected [Cugola & Margara, 2012]. Additionally, it performs log auditing (see Section 4.2.9), supports NDN agent discovery (see Section 4.5.1) and log commitment between NDN agents (see Section 4.5.2).

Moreover, content providers and NDN agents depend on two security models based on the SPKI/SDSI small-world model of trust (see Section 2.3.2). The first model is supported by a globally trusted ICANN-like entity, which coordinates namespace allocation and digital certificates with content providers. This entity also administers and regularly updates a collection of globally distributed CCN-KRS servers (described in Section 2.3.6), which resolve authorised content publisher security information for a CCN namespace. NDN agents get the latest security information for content providers from CCN-KRS servers. In the second model, content providers bind trust to NDN agents by assigning them name and authorisation certificates using the SPKI/SDSI certificate standard.

In the remainder of this chapter, we identify components and attributes needed to support the scalable collection of accounting and accountability information in an untrusted decentralised ICN environment. Moreover, for simplicity purposes, we assume that a content provider (such as Netflix) will be the entity managing the content distribution process and the Savant framework. However, this function can be delegated to a third party by applying the same principles.

4.2.1 Content Provenance and Authentication

In ICNs, data is bound to the entity that created it. As a result, all entities (or principals i.e. entities capable of generating a digital signature) that publish content (e.g., movies, web pages, log objects, and so forth) must be associated with a public key π_i and private key σ_i pair. This is to ensure that integrity, trust, authenticity and non-repudiation can be established in the data received. To establish a trust context in Savant, all NDN agents sending and receiving content must be identifiable to content providers. Moreover, to ensure authenticity and non-repudiation in content and log objects received, a security model is required. As discussed above, Savant uses a small-world model of trust to support the architectures scalability. We will provide details of Savant's security model in Section 4.3.

4.2.2 Content Ingestion

Commercially licensed content acquired from content producers is uploaded into the content delivery system. The ingestion element prepares content for distribution to many different users, devices and networks, performing tasks such as encryption, adding metadata and producing signed chunks of content. Similar to the generic model outlined in Section 3.1, in Savant this information can then be pushed or pulled to CDN replica servers, ISP cache, ICN cache, CSP, multicast IP and P2P systems. Moreover, content can be cached opportunistically at any trusted or untrusted ICN cache resource. Most ICN architectures support either on-path or off-path caching. On-path caching refers to information cached along the *interest reverse request* path taken by the ICN routing service. Off-path caching directs requests to cache servers not on the requested path. Moreover, content is typically cached according to some cache replacement strategy such as Least Frequently Used (LFU), Most Recently Used (MRU), Most Frequently Used (MFU), and so forth.

4.2.3 NDN Agents

If content is cached on untrusted infrastructure in the network, content providers and distributors cannot monitor the data distribution process. As a result, the primary responsibility of NDN caches and NDN clients (collectively referred to as *NDN agents*) is to produce accounting and accountability information for the content distribution process. An NDN cache is a component that holds a copy of requested content and is prepared to deliver it to any entity that requests it. In contrast, an NDN client downloads, views and interacts with the content received from an NDN cache.

An NDN cache creates a log entry for content if it is retrieved from its local content store. It also creates log entries for *collapsed* interests in its PIT table. A collapsed interest refers to an interest that is not forwarded upstream to the content provider. The original interest that reached the content provider already triggered the creation of a log entry. Collapsed interests support content objects being multicast to multiple simultaneous receiver's in NDN. This scheme has similarities to the push interest based accounting mechanism outlined in Section 2.4.5.

It is envisaged that NDN agents will have some monetary agreement with content providers. For example, the NDN cache might distribute content and produce accounting and accountability information for a fee, while the NDN client might pay a subscription to view content.

4.2.4 Content Metadata

In ICNs, content *metadata* is associated with information such as content author, creation date, expiry date, data encoding rate, security, search keywords, and so forth [Ahlgren et al., 2012]. This information is specified during the content ingestion process and is customisable by the ICN architecture or content provider. As outlined above in Section 2.3.5, a digital signature is computed over the content object's name, payload and the metadata in ICN architectures. Consequently, metadata is bound to content and creator and cannot be changed without affecting the integrity of the content object.

In Savant, content providers are responsible for collecting published logs associated with content emanating from them. This is supported by a set of *accountability engines*, which are owned and operated by the content provider or a trusted third party. To support accountability engines collecting logs from NDN agents, the content provider specifies a *feedback* field in the metadata during the content ingestion process. This field directs NDN agents to the accountability engine responsible for collecting accounting and accountability information for that content provider. For example, Netflix's accountability engine may be contactable via the metadata *feedback:/netflix/acc-engine*, which is directed by the NDN network to the local /netflix/acc-engine resource. Moreover, it is bound to the content and cannot be changed by a malicious third party. The only entity that will receive accounting and accountability information for the content distribution process is specified in the metadata. If the content provider wants to allow content anonymity, they can just omit the feedback metadata during content ingestion.

4.2.5 Published Log Objects

Savant NDN agents use NDN technology to publish log objects. Consequently, all log objects act as digitally signed *commitments* to log integrity and provenance. A published log object e_i has the following properties: $e_i = (h_i, s_i, t_i, c_i)$. This corresponds to a sequence number s_i (always increasing e.g., timestamp), a type t_i (e.g., *Interest*, Content or Event), type-specific content c_i (e.g., content name, QoE, and so forth) and a hash h_i of the current log object.

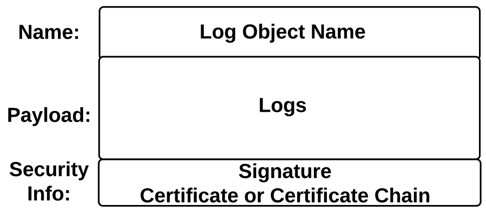


Figure 4.2: Published Log Object

Figure 4.2 illustrates the fields in a published log object. The log object name reflects the local namespace convention outlined in Section 4.4.1. The payload contains log entries such as $e_i = (h_i, s_i, t_i, c_i)$. Each log object is cryptographically signed and carries a link in the metadata to the associated public key and certificate chain. In the Savant security model, the accountability engine receiving a published log object will only trust the NDN agent if the digital certificate anchors at the content provider. Content providers use the SPKI/SDSI certificate standard, which enables local principals (i.e. content provider) to issue digital certificates to NDN agents. Savant's security model is outlined in more detail in Section 4.3.

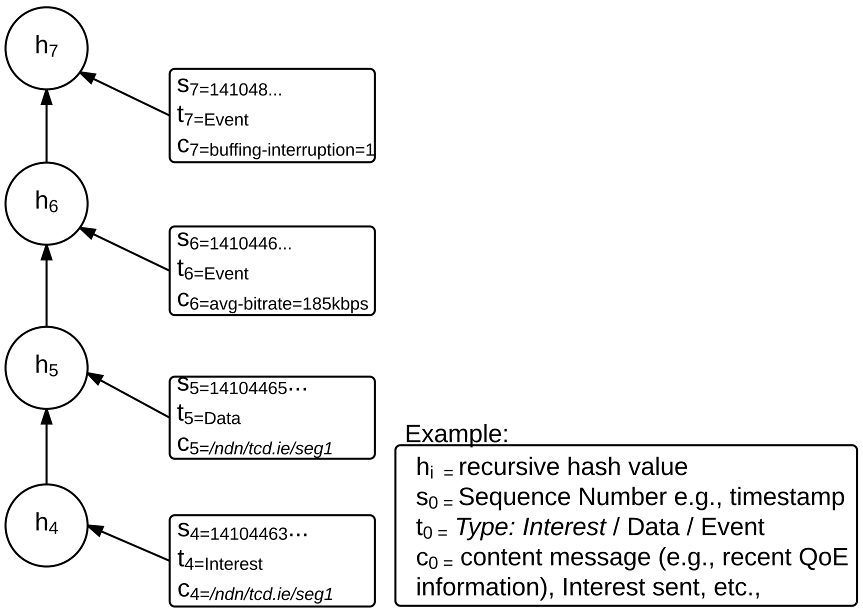

h7
S7=141048...
t7=Event
C7=buffing-interruption=1
h6
S6=1410446...
t6=Event
C6=avg-bitrate=185kbps
h5
S5=14104465...
t5=Data
C5=/ndn/tcd.ie/seg1
h4
S4=14104463...
t4=Interest
C4=/ndn/tcd.ie/seg1
Example:
hi = recursive hash value
s0 = Sequence Number e.g., timestamp
t0 = Type: Interest / Data / Event
c0 = content message (e.g., recent QoE information), Interest sent, etc.,

Figure 4.3: A hash chain of published log objects

Tamper-Evident Logs

A recursively defined hash chain h_i linking the current log object with the previously published log object h_{i-1} makes the log tamper-evident. This is depicted in Figure 4.3. By maintaining hash chains between published log objects, an NDN agent commits to the integrity of the current log object and to all previously published log objects. Moreover, a single linear append-only log, which contains all communication between NDN agents for a particular content provider, must be maintained by each NDN agent to ensure consistency.

4.2.6 Accounting

Content accounting enables a content provider and a content distributor to track the content, quality of service (QoS) and end-user QoE. This information can help to determine what content is popular, the geographical location of users and what kind of performance clients are receiving from the network [Aditya et al., 2012]. Content providers can use this information to maximise revenue and minimise distribution costs, while providing the user with a quality viewing experience for the least cost [Balachandran et al., 2012]. As can be seen in the taxonomy in Section 3.2, there is a broad spectrum of accounting and accountability information that can be produced by NDN agents. The following illustrate two possible accounting scenarios. We utilise the minimal scenario in Section 5.7 to demonstrate Savant's capabilities. The minimal and maximal accounting scenarios collect different amounts of accounting metrics related to content distributed, client QoE and network QoS.

Minimal

This scenario collects industry standard QoE metrics based the on the player state machine in Figure 4.4 (adapted from [Dobrian et al., 2013]). Client metrics monitored include [Dobrian et al., 2013]: join-time, which is the time it takes for the buffer to fill up and start playing; buffering ratio, which is the fraction of total session time (i.e. playing time plus buffering time) spent buffering; rate of buffering events, which is the frequency of buffering interruptions; rendering quality, which is the frames displayed per second on the client; and average uplink or downlink bitrate (e.g., 200 kbps). Additionally, a sample of content chunks distributed is gathered.

It is envisaged that this accounting scenario will be the default for video content distribution in Savant, as the overhead for accounting and accountability is small on NDN agents

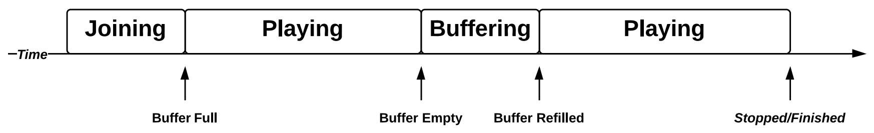


Figure 4.4: Video Player State Machine (adapted from [Dobrian et al., 2013]).

while the utility of the metrics gathered is high. Savant's performance under this scenario is demonstrated in Section 5.7.

Maximal

There is a broad spectrum of accounting information to gather in the maximal scenario. However, we limit the information collected to the following metrics on the client and cache server. These include: CPU usage, uplink and downlink bandwidth, user demographic information and all content chunks distributed. This is in addition to minimal metrics such as join-time, rate of buffering events, buffering ratio and rendering quality. Moreover, additional metrics can be inferred from the information gathered. For example, the time taken to serve all individual content chunks, chunk delivery failures, cache hit ratios and content popularity. Additionally, content quality can be determined based on chunks distributed, received and frequency of end-user random access such as Play/Stop/Seek.

4.2.7 Authenticated Interest Commands

Savant's scalability and potential to detect, isolate and resolve content distribution problems depends on its ability to run commands whenever required at NDN agents. We realise this using authenticated interest commands, which add commands and digital signatures to NDN interests [Burke et al., 2013]. Savant's authenticated interest commands are based on the same principles as the NDN lighting control system outlined in Section 2.3.7. They provide a secure, access controlled and non-repudiable methodology for communicating with untrusted infrastructure. Additionally, authenticity and provenance of commands received can be easily established by NDN agents based on security information received for the content provider using the CCN-KRS framework.

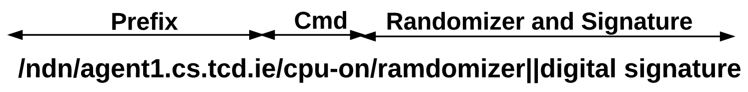


Figure 4.5: Authenticated Interest [adapted from [Burke et al., 2013]]

To recap Section 2.3.7, an authenticated interest is composed of three parts, as illustrated in Figure 4.5. The *prefix* is used for routing commands to NDN agents while the *cmd* carries the directive. The *randomizer* contains a nonce and timestamp to prevent timing and replay attacks, which is concatenated with a digital signature computed over the rest of the interest. The content provider or one of its delegates such as an accountability engine signs interests. Savant uses simple commands such as *on* or *off* to enable or disable accounting functionality. However, more sophisticated commands can be run, such as redirecting clients to alternative cache resources or blacklisting cache servers from content distribution. See Table 4.1 for a list of sample commands used by Savant.

Table 4.1: Savant Authenticated Interest Commands

Command	Comment
cpu-on/cpu-off	Turn off/on CPU statistics at NDN agent
framedrop-on/framedrop-off	Turn off/on frame drop accounting on client
uplink-on/uplink-off	Turn off/on average uplink bitrate
downlink-on/downlink-off	Turn off/on average downlink bitrate
blacklist=[ip-address]	Blacklist cache server
/namespace=[ip-address]	Add new route for /namespace
logging-period=15	increase/decrease logging periodicity
sample-frequency=325	chunk statistical sample frequency

Savant uses a similar methodology to the NDN lighting control framework described in Section 2.3.7 to produce and verify authenticated interest commands. However, as NDN agents are expected to run on non-resource constrained devices, we recommend using public keys and digital signatures for authentication. This is in contrast to the NDN lighting control system, which advocates using symmetric keys to produce message authentication codes (MACs), on resource constrained devices [Burke et al., 2012]. Moreover, as the collection of additional accounting metrics can have associated costs (both system and monetary), all commands received from content providers must be acknowledged. Consequently, accountability and non-repudiation is supported in the architecture as both interests and data responses are digitally signed.

When an NDN agent receives an authenticated interest from a content provider, it determines whether or not to execute the command, as follows:

1. Obtain and verify the authenticity of the content providers public key. If the key is not available locally, it can be retrieved using the CCN-KRS framework [Mahadevan et al., 2014], which is outlined in detail in Section 2.3.6.

2. Verify the digital signature in the authenticated interest received is from the content provider.

3. Check if the content provider is allowed to run the command, which is determined by the access control policy described in Section 4.3.5.

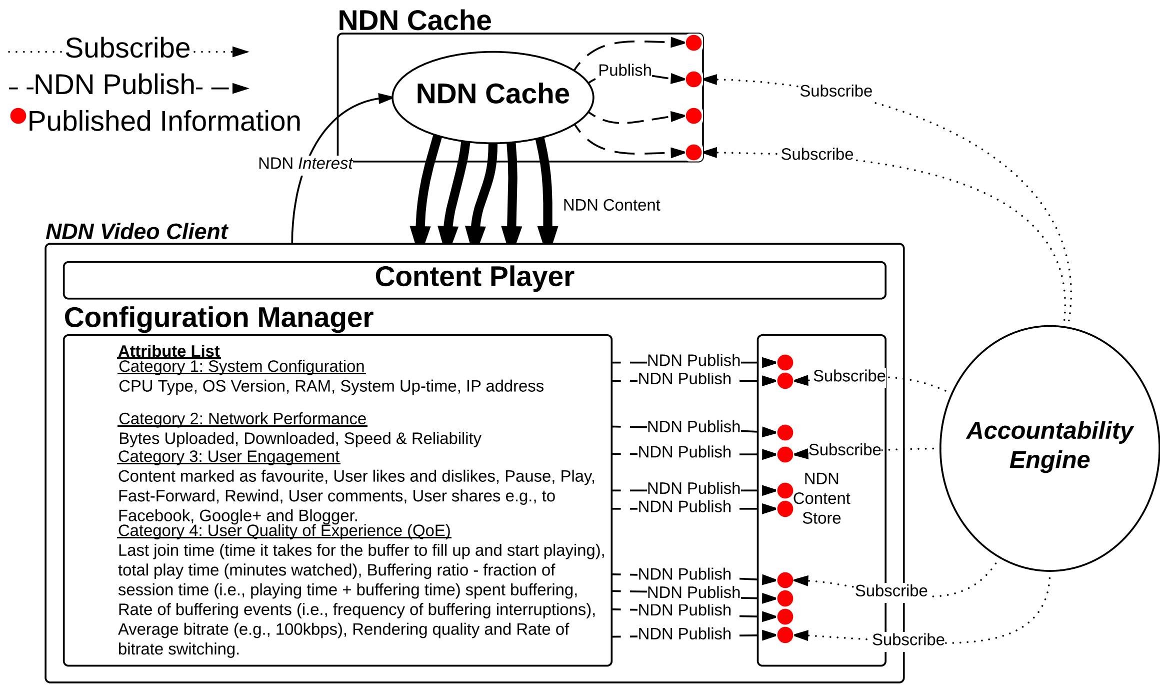
4. Verify the command is well formed.

5. Verify the randomizer information is current e.g., timestamp.

If these steps are completed successfully, the NDN agent executes the command and generates a data response to the content provider. Moreover, the command executed is published as a log object.

4.2.8 Accountability Engine

The accountability engine is a critical piece of infrastructure for the Savant framework. A high-level overview is depicted in Figure 4.6. It is infrastructure owned and operated by the content provider or a trusted third party. Its primary function is to support content providers collecting, aggregating and auditing accounting and accountability information collected from trusted or untrusted NDN agents. However, it also supports NDN agents with discovering accountability engine infrastructure based on metadata specified during content ingestion. Moreover, it supports log commitment (see Section 4.5.2), which ensures an NDN



Attribute List
Category 1: System Configuration
CPU Type, OS Version, RAM, System Up-time, IP address

Category 2: Network Performance
Bytes Uploaded, Downloaded, Speed & Reliability
Category 3: User Engagement
Content marked as favourite, User likes and dislikes, Pause, Play, Fast-Forward, Rewind, User comments, User shares e.g., to Facebook, Google+ and Blogger.
Category 4: User Quality of Experience (QoE)
Last join time (time it takes for the buffer to fill up and start playing), total play time (minutes watched), Buffering ratio - fraction of session time (i.e., playing time + buffering time) spent buffering, Rate of buffering events (i.e., frequency of buffering interruptions), Average bitrate (e.g., 100kbps), Rendering quality and Rate of bitrate switching.

Figure 4.6: High-level Overview of the Savant Accountability Engine

agent cannot create log entries for interest or content it never sent or received. Furthermore, during the agent discovery process (see Section 4.5.1), the accountability engine provides NDN agents with digital certificates based on the SPKI/SDSI certificate standard to support the production of accounting and accountability information in local namespaces. This is based on the local trust model outlined in Section 4.3.1.

When NDN agents are authorised to publish accounting and accountability information in a content provider's namespace, the collection of accounting and accountability information can start. NDN agents produce published log objects using sequential data names based on the hierarchical naming scheme. As a result, it is easy for accountability engine infrastructure to formulate the name of the next interest to send for a published log object. Once the log object is collected, the following happens on the accountability engine:

1. The integrity and authenticity of the published log object is established using the NDN agents public key.

2. The log object is decapsulated.

3. The integrity of the logs hash chain is determined.

Table 4.2: Log Auditing Event and Type

Event	Type
Uplink bandwidth	Deterministic (every second)
Downlink bandwidth	Deterministic (every second)
Buffering ratio	Non-deterministic
Join-time	Deterministic (occurs once)
Rate of buffering events	Non-deterministic
Rendering quality	Non-deterministic
Interest	Deterministic
Content	Deterministic

4. Logs are audited based on the accounting scenario used, e.g., typically using the Minimal Accounting scenario.

5. Logs are presented to aggregator functions for data aggregation, summarisation and alerting.

4.2.9 Log Auditing

Each log is audited according to a logging scenario defined by the content provider and based on a record of authenticated interest commands received. Combined, they identify what information is expected to be in the log during auditing e.g., QoE metrics, interests and content sent/received, and so forth. For example, based on the minimal accounting scenario outlined in Section 4.2.6, the log should contain some of the information in Table 4.2. While certain information is deterministic by predictably occurring at intervals such as every second (e.g., uplink or downlink bandwidth), other metrics such as buffering interruptions or rendering quality are non-deterministic and can occur due to network fluctuations or client-side issues.

Regardless of the order of data in the log, it should be compatible with what was defined in the accounting scenario used and based on authenticated interest commands received. For example, if an authenticated interest command is received to turn on CPU metric gathering, then we would expect to see CPU metrics occurring deterministically every second in the log. Any evidence of log subversion, log-tampering (i.e. based on checking for the consistency of hash chains between published log objects) or inconsistencies (i.e. logs deviate from the expected logging scenario) can be reported directly to the content provider or investigated further using authenticated interest commands.

4.2.10 Aggregator Functions

Aggregator functions based on information flow processing mechanisms are utilised by Savant-NDN agents and accountability engines to aggregate, summarise and alert on data collected [Cugola & Margara, 2012][Zaharia et al., 2012]. Some aggregator functions embody functionality similar to data stream management systems, which run queries continuously and provide updated answers when new data arrives. Savant's accountability engine is configured to produce a summary of events received at ten second intervals[1] using the following *generic* Apache Spark SQL command:

Listing 4.1: Apache Spark SQL Command

```
SELECT hostname, eventtype, AVG(value) as Average
FROM records
GROUP BY hostname, eventtype
ORDER BY eventtype desc
```

The command in Listing 4.1 produces a table in the accountability engine every ten seconds with an average of all events received grouped by NDN agent. Other functions can perform complex event processing such as monitoring event patterns for multiple related events in tables produced, then notifying interested parties on occurrence. Consequently, aggregator functions can determine what feedback information to collect and the frequency at which it is collected. Furthermore, they can combine the outputs from other aggregator functions, supporting compositional processing. Utilising support from high-level accountability engine infrastructure, aggregation is performed in the distributed architecture incrementally and hierarchically and does not require large volumes of logs to be aggregated nightly, which is common in existing CDN, CDNI and hybrid CDN-P2P systems such as Akamai and Akamai Netsession [Nygren et al., 2010] [Aditya et al., 2012]. Furthermore, Savant supports decentralised aggregation in order to aggregate information as close as possible to the data

[1]This interval can be configured by the content provider and changed easily when required. For example, if the log size is very large and gathered every 1, 10, or 60 minutes, the SQL can be configured to reflect this interval.

source and minimise the amount of bandwidth used when collecting logging information. Finally, summarised information is published as a log object, which is available for collection by higher-level accountability engines or the content provider.

4.2.11 On-Demand Accounting and Accountability

One of the defining benefits of the Savant framework is its ability to monitor the content distribution process by collecting a subset of the information available in the taxonomy in Section 3.2 in near real-time, while maintaining the flexibility to gather additional accounting information when required. This is achieved utilising *authenticated interest* commands (outlined in Section 4.2.7) and choosing the right subset of accounting metrics to monitor. In the current implementation of Savant, we use the minimal accounting scenario (defined in Section 4.2.6) by default, which is based on industry standard QoE metrics [Dobrian et al., 2013]. These metrics are summarised in a table in the accountability engine (currently at ten second intervals using the Apache Spark framework [Zaharia et al., 2012]). This is supported by aggregator functions (outlined in Section 4.2.10) based on event processing languages, which intelligently aggregate, monitor and alert on arriving data in near real-time. Using aggregator functions and authenticated interest commands, Savant can easily collect additional metrics if anomalies are detected. For example, if the downlink bandwidth decreases on a client, this could initiate collection of additional accounting metrics on the client and cache to try determine the cause of the fault. In Section 5.8.2, we present an example of Savant detecting a fault scenario.

In the current version of Savant, the frequency that NDN agents produce and accountability engine infrastructure collect published log objects is configurable by the content provider using authenticated interest commands. This interval, which is currently set to 15 seconds by default for all NDN agents, can be increased or decreased when needed at a single pair of communicating NDN agent or all NDN agents by the content provider.

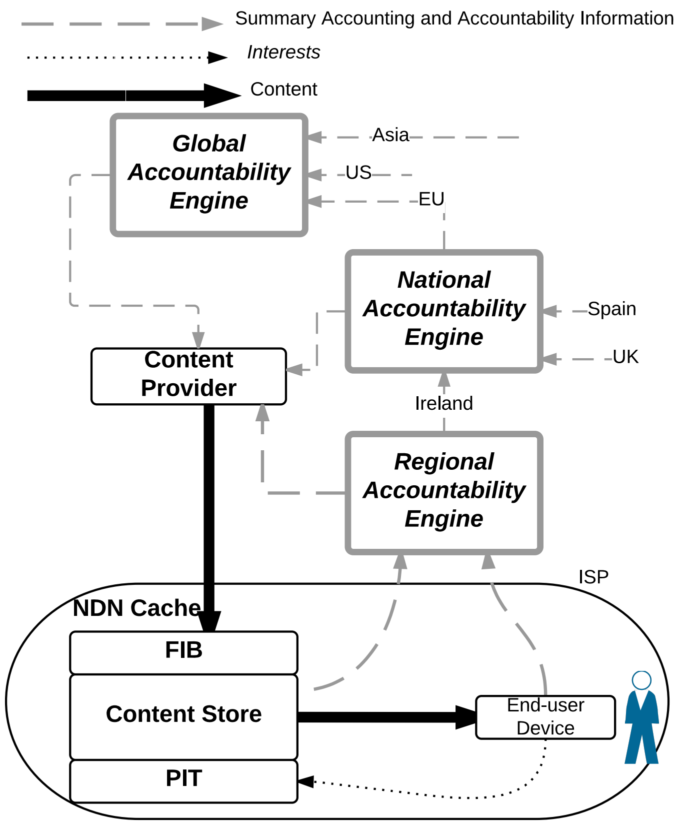

Summary Accounting and Accountability Information
Interests
Content
Global Accountability Engine
Asia
US
EU
National Accountability Engine
Spain
UK
Content Provider
Ireland
Regional Accountability Engine
ISP
NDN Cache
FIB
Content Store
PIT
End-user Device

Figure 4.7: High-level Overview of the NDN architecture with support from hierarchical accountability engines.

4.2.12 High-level Accountability Engine

Savant's scalability is supported by hierarchical accountability engines, which collect and aggregate summary data from multiple low-level accountability engines. Summary information is propagated back to the content provider as published log objects. The Savant architecture forms a tree-like structure where leaves represent NDN agents, accountability engines correspond to branches and the content provider acts as the root of the tree. A high-level overview of the NDN architecture with support from hierarchical accountability engines is depicted in Figure 4.7.

4.3 Security Model

NDN's trust model does not rely on any centralised public key infrastructure (PKI), instead advocating a distributed trust model. Trust in keys is typically established using a PKI-like certificate chain based on the content naming hierarchy [Xylomenos et al., 2014]. However, the CCN network can have multiple copies of published content with the same name from different content providers, which are bound to valid public keys. Savant aggravates this trust problem further as both NDN agents and content providers can publish content in any namespace they choose. Consequently, a key resolution service is required (such as CCN-KRS [Mahadevan et al., 2014], which is adopted by Savant) to resolve authorised content provider security information for a CCN namespace (see Section 2.3.6). Moreover, Savant requires a scalable and flexible security model that can manage namespace allocation and the integrity and authenticity of published content for both content providers and NDN agents. This should be supported with a PKI for producing, distributing, storing, using, authenticating and revoking digital certificates and managing the life cycle of public keys. Finally, known and unknown threats and attack methods should be mitigated in the best possible way.

4.3.1 Savant's Trust Model

In Savant, different levels of trust are required between content providers, NDN agents and accountability engine infrastructure. For example, if a private key is compromised in any ICN architecture, an attacker can publish valid content under the publishers namespace. This is a serious threat for content providers as they are responsible for copyrighted material such as movies, TV shows, and so forth. Additionally, an attacker with a publisher's private key in Savant can send authenticated interest commands to NDN agents (see Section 4.2.7), which can impact on client performance and have associated monetary costs. In contrast to content providers publishing copyrighted material, NDN agents produce accounting metrics related to the content distribution process, which while important, are less critical to secure. Moreover, these metrics are typically only of interest to the content provider.

Consequently, we envision Savant using at least two PKI trust models to support the architectures scalability, see Figure 4.8. In the first PKI model, trust is anchored at a globally trusted ICANN-like entity, which supports content providers publishing copyrighted material and end-users getting authorised content for a namespace. In the second PKI model, trust originates from the content provider (local principal) to NDN agents publishing accounting and accountability information. Both trust models are based on the SPKI/SDSI PKI cer-

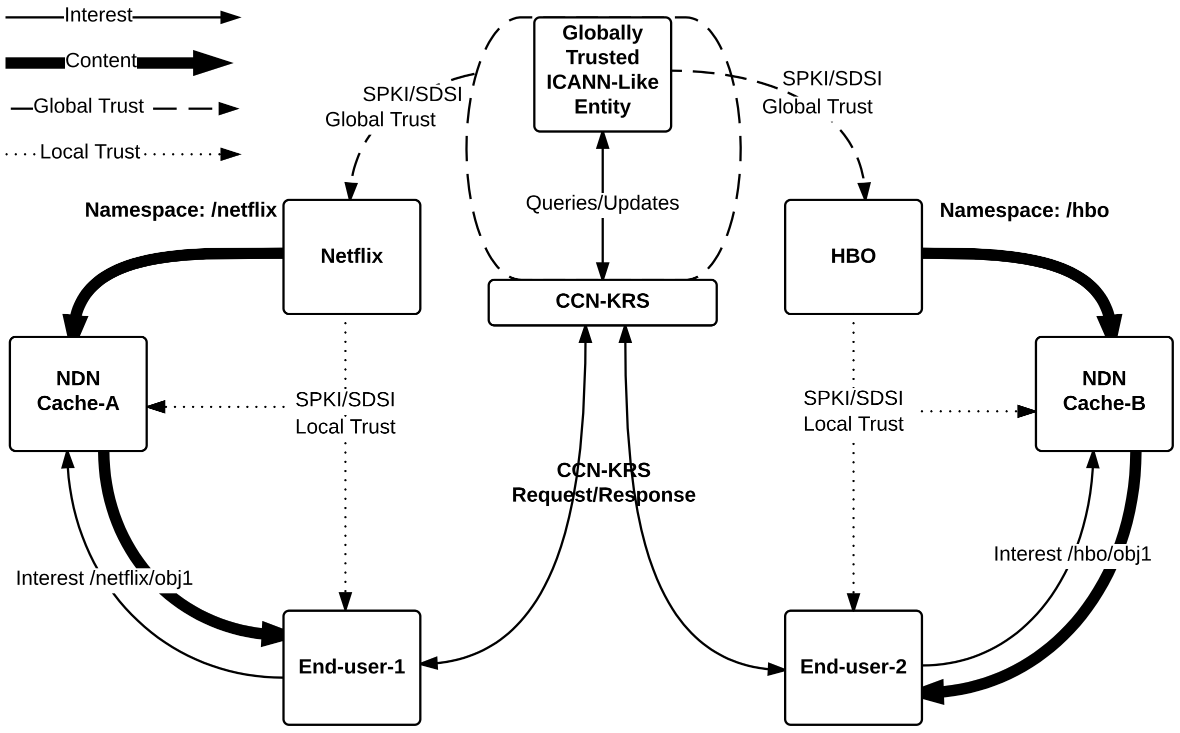


Figure 4.8: This figure shows two trust models (hierarchical and local) used by the Savant framework, which are both supported by the SPKI/SDSI PKI certificate standard.

tificate standard (see Section 2.3.2). End-users receiving content can easily establish the authenticity of a content providers public key by following the certificate path from the trust anchor to the content provider's digital certificate.

Global Trust Model

In the global PKI model, we envisage content providers such as Netflix and HBO using the services of a globally trusted ICANN-like entity to coordinate namespace allocation and manage associated digital certificates. Additionally, one of its secondary responsibilities is to update the CCN-KRS service (outlined in Section 2.3.6) with the latest security information (such as new, updated or revoked digital keys) for content providers. Consequently, end-users can easily resolve up-to-date authorised content publisher security information for a CCN namespace. This entity can also act as a CA that produces digital certificates, manages the life cycle of public keys and certifies and accepts liability for the authenticity of these keys if required. However, trust can originate from different sources based on the SPKI/S-DSI certificate standard. As a result, its main responsibilities include managing namespace allocation and updating CCN-KRS servers.

This ICANN-like entity can also delegate capabilities to third party registrar entities (similar to those used in DNS in Section 2.3.1), which are authorised to manage and reserve prefixes. To ensure the architecture's scalability, registrars are also authorised to produce digital certificates using the SPKI/SDSI certificate standard for content providers.

Local Trust Model

In the *local trust* model (i.e. small-world model of trust), NDN agents are given digital certificates by the content provider (e.g., Netflix) based on the SPKI/SDSI certificate standard. In this model trust originates from the content provider (i.e. local principal), which defines a local name in the issuers namespace using SPKI/SDSI name certificate (outlined in Section 2.3.2) and assigns these names to NDN agents. Moreover, the local principal grants authorisations using SPKI/SDSI authorisation certificate (outlined in Section 2.3.2) to publish log objects under a namespace.

Granting name and authorisation certificates to NDN agents can also be delegated (using the authorisation certificate delegation bit) to an accountability engine (or other entity) by the content provider. For example, a Netflix end-user is granted a name certificate for namespace /netflix/acc-engine/user1 by a Netflix accountability engine, which also issues an authorisation certificate to publish log objects under that namespace.

SPKI/SDSI supports a secure, scalable and flexible authorisation model managed by a content provider or accountability engine infrastructure, which can build distributed trust among local communities without requiring the support of a globally trusted entity (root). Moreover, every accountability engine or content provider is free to adopt an independent model for managing the life cycle of NDN agent public keys, which can expire periodically requiring new certificates to be issued. NDN agents require valid digital certificates to publish in a content providers namespace.

Adopting the SPKI/SDSI trust model can also help mitigate some ICN threats and attacks identified next while continuing to support the architecture's scalability.

4.3.2 Threats and Attack Scenarios

Several threats and attacks were identified related to the Savant accounting and accountability model. These include [AbdAllah et al., 2015]:

1. *Corrupted Data*: Data has not been modified or tampered with and comes from an original source. Data integrity, provenance and authenticity can be established.
2. *Unauthorised Access*: End-users viewing content without permission or authorisation.
3. *Cache Poisoning*: Caches intentionally polluted with corrupt or unpopular content.
4. *Path Infiltration*: Attackers announcing invalid routes for content.
5. *Unauthorised Commands*: Unauthorised (authenticated interest) commands being run against NDN agents such as turning off or on accounting and accountability functionality.
6. *Privacy* - Content Monitoring and Censorship: A third party eavesdropping, intercepting and modifying published log object information or authenticated interest commands.
7. *Collusion*: Agents suspected of colluding to perform malicious or deceitful behaviour. This includes an inflation or deflation attack where the client and cache collude to over or under report upload or download activity [Aditya et al., 2012]. Similarly, a Sybil attack, when a user forges multiple identities that join a system in order to perform one or multiple attacks while subverting the system authority.
8. *Denial of service (DoS)* Attacks: This includes interest flooding and content/cache poisoning [Afanasyev et al., 2013]. Additionally, content providers overloading the cache or client with requests for accounting and accountability information using authenticated interest commands.
9. Poor *Quality of Service (QoS)*: Caches refusing, degrading or aborting data transfers to end-users and reporting otherwise.

10. *Resource Exhaustion*: Infrastructure susceptible to large amounts of requests or flooding attacks.

4.3.3 Defence Mechanisms

To illustrate Savant's defence mechanisms against these kinds of threats and attacks, we consider how a content provider similar to Netflix might use the NDN model for content distribution in conjunction with the Savant framework for accounting and accountability.

- To support Savant, each end-user device, cache server and content provider must be associated with identities that can be cryptographically asserted. Public keys are used to establish the integrity and provenance in content and log objects received from content providers and NDN agents. In Savant, a globally trusted ICANN-like entity issues namespaces to content providers and digital certificates (if required). Additionally, content providers adopt a local trust model using the SPKI/SDSI certificate standard to issue name and authorisation certificates to NDN agents for publishing accounting and accountability information.

- To force NDN agents to produce accounting and accountability information, content providers can encrypt content before distribution to protect it from unauthorised access. Decryption keys are shared with authorised users. Consequently, all end-users viewing content must be accessible to the content provider. Additionally, this content can be set to periodically expire (e.g., daily) requiring replacing at caches with new encrypted content.

- In CCN/NDN, establishing trust in content is an application-dependent concept. Consequently, it is difficult for the CCN network to differentiate between valid or polluted content based on content name only. As a result, Savant employs the CCN-key resolution service (CCN-KRS), which is a DNS-like service for resolving security information (such as public key certificates) for a CCN/NDN namespace [Mahadevan et al., 2014]. As outlined in Section 2.3.6, CCN-KRS supports the restriction of content requests to that which is signed with a content providers public key or content digest

[Mahadevan et al., 2014]. Consequently, public keys or content digests can be specified in the interest sent, which guarantees (assuming network compliance) an end-user will not receive fake or polluted content from a cache. Additionally, content provider security information is maintained and updated in CCN-KRS in cooperation with a globally trusted CA.

- A related attack involves caches advertising invalid routes for content. Using CCN-KRS security information, the CCN network will only deliver valid content for an interest request (assuming network compliance).

- NDN agents can also establish the integrity and provenance of authenticated interest commands received from content providers using the CCN-KRS framework [Burke et al., 2013]. Additionally, they include mechanisms (such as timestamps, estimated round-trip-time, sequence number, and so forth) to protect against an adversary attack that records, drops, modifies, replays, injects or delays commands [Burke et al., 2013].

- Several threats exist whereby accounting information is disclosed to a competitor. This involves eavesdropping on information sent between client, cache and accountability engine. Encrypting all published log object information with the content providers public key or a symmetric key shared with the accountability engine can rectify this issue. Moreover, authenticated interest commands and data response can optionally be encrypted with the receiver's public key.

- A related threat involves a competitor running a hostile cache node where they have a complete view of the accounting process for a particular content item in a specific region. This falls into an area of collusion attacks if the competitor also runs client nodes to create fake transactions. The use of a strong identity management system, which binds actions to agents, should act to discourage this kind of behaviour, assuming the cost of getting caught outweighs the gains [Yumerefendi & Chase, 2005].

- Savant is susceptible to several types of denial of service (DoS) attack including: content pollution and interest (and authenticated interest) flooding. As discussed in Section 2.3.6, the CCN-KRS framework limits the impact of content pollution attacks by guaranteeing the client will not receive corrupted or fake content. To mitigate interest-flooding attacks, we advocate using proven statistical algorithms that use per packet router state, interest satisfaction ratios (i.e. interests sent versus satisfied) and explicit interface limits to restrict the amount of malicious interests forwarded by routers [Afanasyev et al., 2013]. Moreover, authenticated interest commands received can help isolate the attack source as commands are signed by the attacker, which is identifiable to the system. Finally, we suggest content providers and caches use some sort of monetary model based on reimbursement for different amounts of accounting and accountability information produced. These choices require future research.

4.3.4 Savant Supported Defence Mechanisms

Other attack methods exist that cannot be mitigated by the mechanisms outlined above. However, the faults generated by these attacks can be recognised by the Savant framework and some action taken (e.g., raise an alarm or redirect clients to alternative infrastructure for content distribution using authenticated interest commands). These include:

- Savant can monitor the quality of experience of experience (QoE) between NDN agents using aggregator functions based on event processing languages. Consequently, it is able to recognise issues such as caches refusing, degrading or aborting data transfers to NDN clients. Moreover, published log objects provide non-repudiable evidence of actions taken by NDN agents, which can be audited. Alarms can be raised and some action initiated such as automatically redirecting clients to alternative trusted cache infrastructure for content distribution using authenticated interest commands.

- Authenticated interest commands could also be used to redirect client infrastructure to alternative trusted cache resources if NDN agents are suspected of colluding to perform malicious or deceitful behaviour. This is similar to the hybrid CDN-P2P approach adopted by the RCA system [Aditya et al., 2012], which redirects (quarantines) nodes suspected of causing malicious or deceitful behaviour to trusted infrastructure for content distribution.

- As Savant has the capability to process and aggregate client and cache performance metrics it can monitor for types of DoS attacks such as resource exhaustion or network congestion. This is supported by authenticated interest commands, which turn off an on accounting and accountability information at NDN agents. Moreover, as Savant knows the location of multiple copies of content, it can redirect client infrastructure to an alternate location for content distribution.

This is not an exhaustive list of attacks methods that can be identified by Savant. It serves only to illustrate some of Savant's capabilities. In Section 5.8.2, we engineer a fault into the NDN content distribution process to demonstrate Savant's ability to detect and isolate some of these kinds of attack.

The defence mechanisms outlined above are supported by the Savant trust model, asymmetric and symmetric cryptographic techniques, the CCN-KRS framework, a strong identity management system and proven statistical algorithms (to mitigate interest flooding attacks). Moreover, Savant components such as accountability engines, aggregator functions and authenticated interest commands provide additional mechanisms to recognise faults and perform some action to help mitigate content distribution problems.

4.3.5 Access Control Lists

An NDN agent can decide whether or not to execute authenticated interest commands received from content providers based on locally defined access control lists (ACLs). NDN agents can optionally create their own ACLs using the SPKI/SDSI certificate standard (see Section 2.3.2). These ACLs can determine what authenticated interest commands content providers are allowed to run on NDN agents. Consequently, they are expected to be based on policies and agreements between content providers and NDN agents. For example, an NDN cache may adopt a monetary model based on reimbursement for different amounts of accounting and accountability information produced and shared with content providers.

Every NDN agent contains a complete list of access control policies for all accounting and accountability information listed in the accounting taxonomy in Section 3.2. However, for scalability purposes, all of these are disabled by default except for the minimal accounting scenario metrics outlined in Section 4.2.6. However, the other metrics can be turned off and on when required using authenticated interest commands. It should be noted that the more access control policies activated, the more accounting and accountability information generated. This can have a significant impact on available client, cache and accountability engine resources. Consequently, it is recommended that this functionality be turned on only when necessary.

4.4 Naming

4.4.1 Global and Local Namespace

The global and local namespace used by content providers and NDN agents in the Savant architecture is dependent on the trust model used. As outlined in Section 4.3, Savant uses hierarchical and local trust models supported by the SPKI/SDSI certificate standard.

The hierarchical model is rooted at a globally trusted ICANN-like entity that manages namespaces and optionally, digital certificates. This entity designates content providers such as Netflix and HBO to be the authoritative publisher for a prefix e.g., /netflix or /hbo. This is reflected in the global CCN-KRS service, which like DNS today, can be easily queried by NDN agents. Moreover, a content provider is free to manage the sub-prefix in any way they choose. For example, Netflix movies may be published under /netflix/movies while TV shows might be published under /netflix/tv.

To support the architecture's scalability, content providers are responsible for managing trust relationships between themselves and NDN agents. This is supported by Savant's small-world model of trust based on the SPKI/SDSI certificate standard [Clarke, 2001]. As outlined in Section 4.3, name and authorisation certificates are issued by the content provider or an accountability engine designated by the content provider to NDN agents. An NDN agent's namespace is local to the namespace of the entity granting it a name certificate.

For example in Figure 4.9, an ICANN-like entity designates Netflix as the authoritative publisher for the */netflix* prefix. Netflix in turn grants several accountability engines authority to delegate authorisation certificates and publish in a local namespace in the certificate

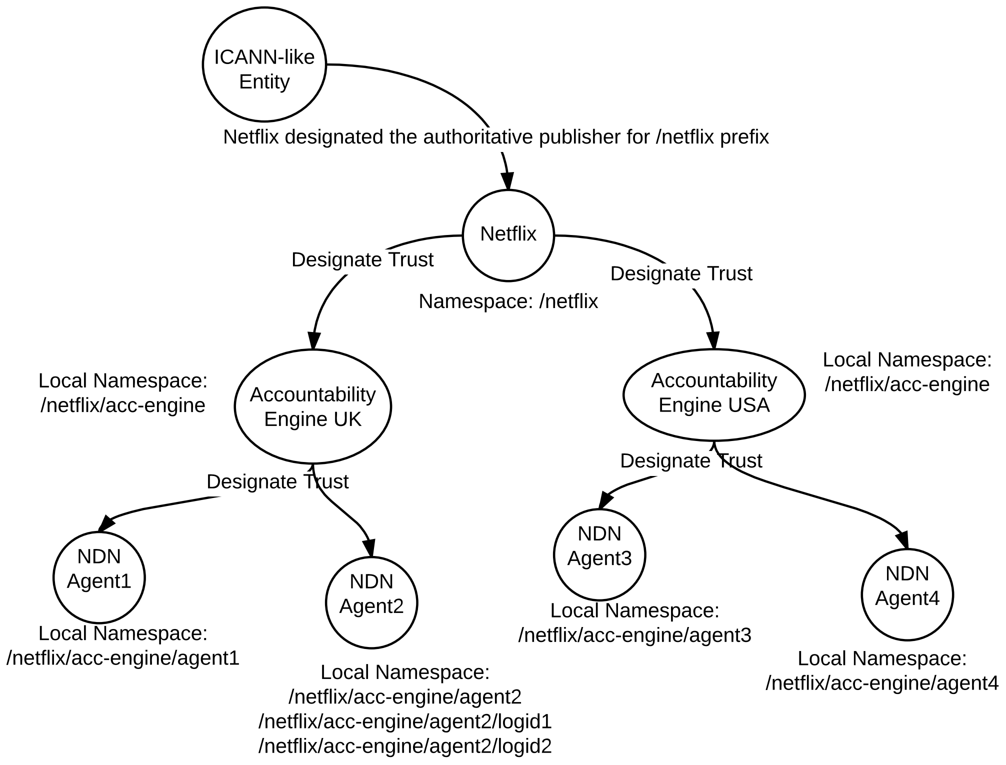

ICANN-like Entity
Netflix designated the authoritative publisher for /netflix prefix
Netflix
Namespace: /netflix
Designate Trust
Designate Trust
Local Namespace: /netflix/acc-engine
Accountability Engine UK
Accountability Engine USA
Local Namespace: /netflix/acc-engine
Designate Trust
Designate Trust
NDN Agent1
Local Namespace: /netflix/acc-engine/agent1
NDN Agent2
Local Namespace: /netflix/acc-engine/agent2 /netflix/acc-engine/agent2/logid1 /netflix/acc-engine/agent2/logid2
NDN Agent3
Local Namespace: /netflix/acc-engine/agent3
NDN Agent4
Local Namespace: /netflix/acc-engine/agent4

Figure 4.9: Savant's SPKI/SDSI small-world trust model

issuer's namespace (e.g., /netflix/acc-engine). Accountability engines can, via succession, grant NDN agents authority to publish in a local namespace to the accountability engine (e.g., /netflix/acc-engine/agent1).

Moreover, NDN agents publish log objects for a content provider under their designated namespace e.g., /netflix/acc-engine/agent2/logid1. If NDN agents interact with multiple content providers, they will publish log objects under multiple namespaces e.g., /netflix or /hbo or /bbc. Accountability engines send interests to NDN agents for published log objects. Moreover, accountability engines and hierarchical accountability engines can publish aggregate data under the namespace designated to them by the content provider e.g., /netflix/acc-engine/logid1.

4.5 Savant Protocols

Several protocols are required to support the Savant framework. The first underpins agent discovery and identity management and is supported by the accountability engine and the SPKI/SDSI certificate standard. The second supports log commitment between NDN agents, which is also aided by the accountability engine. The third ensures a content publisher (content provider, NDN agent, and so forth) can prove its ownership of a key, which is achieved using a simple challenge response protocol. We will outline these protocols in the remainder of this section.

4.5.1 Agent Discovery and Local Identity Management Protocol

Content providers specify the accountability engine responsible for collecting accounting and accountability information using *feedback* metadata during content ingestion. For example, Netflix's accountability engine might be contactable via the following metadata: *feedback:/netflix/acc-engine*. NDN agents read metadata when sending or receiving content. If the feedback field is present and the NDN agent does not have a valid SPKI/SDSI name or authorisation certificate for the content provider, it will contact the local[2] accountability engine by sending it an authenticated interest command. The authenticated interest contains the following information (see Figure 4.10): the feedback prefix; a command request for a name or authorisation certificate; the NDN agents IP address and public key (generated by the NDN agent if required), which will be used to publish log objects; randomizer and authentication tags.

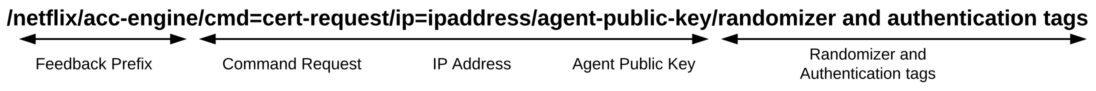

/netflix/acc-engine/cmd=cert-request/ip=ipaddress/agent-public-key/randomizer and authentication tags
Feedback Prefix
Command Request
IP Address
Agent Public Key
Randomizer and Authentication tags

Figure 4.10: Authenticated interest command supporting NDN agent discovery

After verifying the integrity of the authenticated interest received using the digital signature and NDN agent's public key, the accountability engine responds with a published content object (see Figure 4.11). This content object acts as a SPKI/SDSI name certificate. It contains the accountability engines public key i.e. the issuer, a local namespace in the accountability engines namespace i.e. the identifier e.g., /netflix/acc-engine/agent1, the NDN agents public key i.e. the subject, and a certificate start and expiry date i.e. the validity

[2]The NDN network forwards the request to the closest (i.e. local) accountability engine responsible for collecting accounting and accountability information for that content provider. Interests are forwarded based on NDN router FIB table entries, which are populated by a name-based routing protocol.

period.

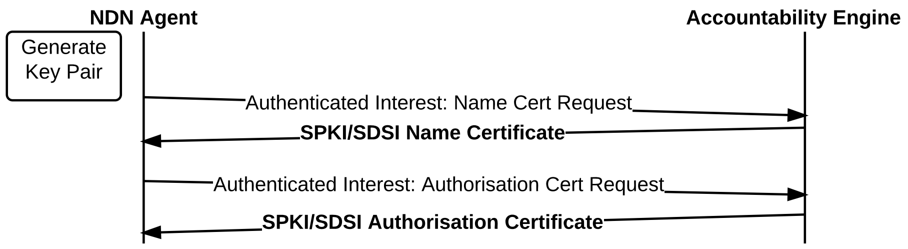

NDN Agent
Accountability Engine
Generate Key Pair
Authenticated Interest: Name Cert Request
SPKI/SDSI Name Certificate
Authenticated Interest: Authorisation Cert Request
SPKI/SDSI Authorisation Certificate

Figure 4.11: SPKI/SDSI local identity management protocol between client and accountability engine

On receiving the name certificate, the NDN agent sends another authenticated interest command requesting an authorisation certificate to publish content under its assigned local namespace. The accountability engine responds with a second content object, which acts as an SPKI/SDSI authorisation certificate (see Figure 4.11). It contains: the accountability engine's public key i.e. the issuer; the NDN agents public key i.e. the subject; the authorisation to publish under the local namespace /netflix/acc-engine/agent1 i.e. the tag; and a certificate start and expiry date i.e. the validity period.

When both SPKI/SDSI name and authorisation certificates have been received, the NDN agent can publish log objects under its designated local namespace until the associated certificates expire or are revoked. Published logs are collected and audited in near real-time by the accountability engine. Moreover, content providers have direct control over the security model used and can manage the life cycle of public keys associated with NDN agents. Finally, the agent discovery and local identity management protocol runs only when the name or authorisation certificates do not exist or expire on NDN agents.

4.5.2 The Log Commitment Protocol

Log commitment between NDN agents ensures that an NDN agent cannot create log entries for interests or content it has never sent or received. This is accomplished by adding the digital signature of the published log object for the interest or content *sent* to the corresponding agents log for interest or content *received*.

The log commitment protocol in Figure 4.12 is an improvement on previously pub-

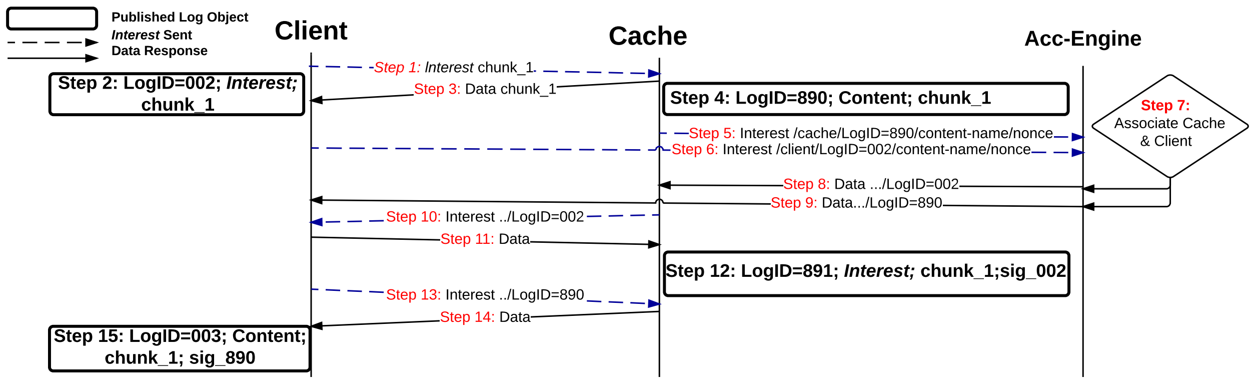

Published Log Object
Interest Sent
Data Response
Client
Cache
Acc-Engine
Step 1: Interest chunk_1
Step 2: LogID=002; Interest; chunk_1
Step 3: Data chunk_1
Step 4: LogID=890; Content; chunk_1
Step 5: Interest /cache/LogID=890/content-name/nonce
Step 6: Interest /client/LogID=002/content-name/nonce
Step 7: Associate Cache & Client
Step 8: Data .../LogID=002
Step 9: Data.../LogID=890
Step 10: Interest ../LogID=002
Step 11: Data
Step 12: LogID=891; Interest; chunk_1;sig_002
Step 13: Interest ../LogID=890
Step 14: Data
Step 15: LogID=003; Content; chunk_1; sig_890

Figure 4.12: Savant commitment protocol with accountability engine support

lished versions [Ó Coileáin & O'Mahony, 2014b], [Ó Coileáin & O'Mahony, 2014a] and [Ó Coileáin & O'Mahony, 2014c]. As a result of these changes, less network and system resources are required to support the log commitment and accountability process between NDN agents. The updated protocol works as follows:

- Log objects are published at the client (Step 2) for interest sent (Step 1) and the cache (Step 4) for Data response (Step 3).
- The client (Step 6) and the cache (Step 5) send the content name (in interest or data sent/received), nonce and ID of recently published log objects to the accountability engine.
- The accountability engine associates communicating agents using the content name and nonce (Step 7).
- It then responds to each agent with the opposing agents LogID (Step 8 to Step 9).
- The client and the cache share published log object information (Step 10 and Step 11; Step 13 and Step 14).
- Log commitment occurs when the digital signature for the interest or content sent is appended to the receiving agents log (Step 12 and Step 15).

Step 10 to Step 12 and Step 13 to Step 15 typically occur in parallel.

This protocol supports log integrity and authenticity between the clients and caches by getting NDN agents to acknowledge having sent or received interests and data messages. Moreover, they provide non-repudiable evidence of their actions by digitally signing a published log object.

It should be noted that in early versions of the Savant architecture we proposed performing log commitment and verification for every individual content chunk distributed. However, it quickly became apparent that gathering accounting and accountability information for all content chunks distributed in a packet-level ICN architecture is a major challenge. As a result, we take a statistical sample of content chunks distributed. This process is covered in greater detail in Section 5.2.2.

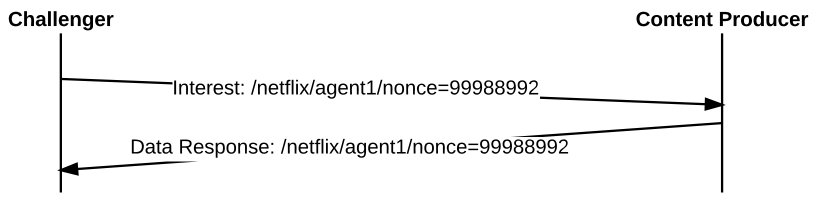

Challenger
Content Producer
Interest: /netflix/agent1/nonce=99988992
Data Response: /netflix/agent1/nonce=99988992

Figure 4.13: Challenge Response Protocol

4.5.3 The Challenge-Response Protocol

A content producer i.e. a content provider, accountability engine, trusted third party or NDN agent, can prove its ownership of a private key using a simple challenge response protocol adopted from the NDN lighting control system [Burke et al., 2012]. A challenger sends an interest to a content provider with a random nonce (see Figure 4.13). For example, /netflix/agent1/nonce=99988992. Based on NDN principles, only a content producer with the secret key can respond to the interest received with a valid content object.

4.6 Extensions and Improvements

During Savant's implementation, we encountered many problems for video distribution related to the volume of content objects created in a packet-level ICN architecture such as NDN. Additionally, there were also issues related to the size of published log objects and the overhead of producing and verifying content received. Consequently, the remainder of this section outlines several modifications we made to Savant to improve the architectures scalability.

4.6.1 Statistical Sample

Savant can monitor all content chunks distributed between NDN agents. However, there is a significant overhead performing log commitment in a packet-level architecture such as NDN for every individual content chunk distributed. As a result, we use a systematic sampling technique to flag content chunks for log commitment using metadata e.g., flagged-content=true, during content ingestion. When a published log object, linked via a hash chain to the previous log entry, is produced by both the client and cache infrastructure that matches flagged metadata, the log commitment protocol is utilised. As a result, the accountability engine and the content provider have a non-repudiable tamper-evident view of the content distribution process between NDN agents based on a statistical sample, which can be audited, aggregated, summarised and alerted upon.

4.6.2 Public/Private Key Length

In the initial design, content and log objects were both published with at least 2048 bit RSA keys. However, there is a significant overhead for NDN agents and accountability engine infrastructure producing, verifying and auditing published log objects produced. In the updated design, content providers such as Netflix continue to publish copyrighted content using at least 2048 bit RSA keys, while NDN agents now use at most 1024 bit RSA keys. As outlined in Section 4.3.1, we justify this change by the level of trust required different entities. For example, we believe stronger keys are required to support copyrighted material such as movies and TV shows provided by content providers. In contrast, NDN agents publish local accounting and accountability information using digital certificates provided by the content provider or accountability engine. Consequently, a compromised key only puts one NDN agent's integrity in danger.

4.6.3 Published Log Object Size

Chunk size varies across ICN architectures due to the overhead required for asymmetric cryptography. In NDN, this is about 650 bytes per chunk, which is independent of the chunk size [Salsano et al., 2012]. Consequently, there is an argument in favour of adding more than one log entry, which is typically between 50 bytes to 130 bytes in size, to a published log object. This would reduce the system and network overhead of producing, distributing and verifying published log objects at both NDN agents and accountability engine infrastructure. To achieve this in Savant, all interests and content sent and received and all accounting metrics produced during a fifteen second interval are aggregated into one published log object by both client and cache infrastructure. Additionally, hash chains are maintained between each log entry. Consequently, when logs are collected by the accountability engine, hash chain integrity between log entries and published log objects requires verification.

4.7 Analysis

In section 4.1, we outlined several requirements that Savant must fulfil. We now analyse how we believe these requirements have been satisfied by Savant's design.

The first requirement was to identify what accounting information should be collected from NDN agents. In the taxonomy in Section 3.2, there is a broad spectrum of accounting and accountability information that can be produced by NDN agents. However, the most useful information is defined in the minimal accounting scenario in Section 4.2.6, which is based on industry standard QoE metrics for video distribution. It is envisaged that this would be the default scenario for video content distribution in Savant, as the overhead for accounting and accountability is small on NDN agents while the utility of the metrics gathered is high.

Savant also needs to provide auditable, tamper-evident and non-repudiable accounting and accountability information for the content distribution process. This was achieved by getting NDN agents to use NDN principles to publish log objects. This requires all infrastructures that can publish content or log objects to be associated with identities that can be cryptographically asserted. Moreover, by maintaining hash chains between published log objects, an NDN agent commits to the integrity of the current log object and to all previously published log objects. Furthermore, logs are audited according a logging scenario (e.g., minimal) defined by the content provider and based on a record of authenticated interest commands received. Combined, they identify what information is expected to be in the log during auditing e.g., QoE metrics, interests and content sent/received, and so forth.

The next requirement was to reduce the amount of accounting information collected and processed in contrast to existing infrastructure such as CDNs [Repantis et al., 2010], while maintaining the ability to detect faults and diagnose problems in near real-time. This was achieved by collecting the minimal accounting metrics defined in Section 4.2.6 and allowing content providers to run authenticated interest commands on NDN agents. This is supported by aggregator functions running on accountability engine infrastructure that monitor, aggregate and alert on data collected. These mechanisms support Savant gathering additional accounting metrics when required, which helps with problem investigation, fault detection, and so forth during the content distribution process.

Savant also requires scalability, which has been achieved with several mechanisms. The first utilises accountability engines, which aggregate and summarise data produced at NDN agents. Moreover, hierarchical accountability engines summarise data collected from multiple low-level accountability engines. Summary information is propagated back to the content provider as published log objects. We also identified several extensions and improvements such as statistical sampling and size of published log objects needed to ensure the volume of log objects produced did not overload NDN agents or accountability infrastructure.

Savant requires a security model to ensure the architectures scalability and prevent various types of ICN threats and attacks. We achieved this by developing two trust models based around content providers and NDN agents using the SPKI/SDSI certificate standard. Content providers depend on a globally trusted ICANN-like entity to manage namespace allocation and digital certificates. This entity provides updates regularly to a globally distributed CCN-KRS service, which helps NDN agents resolve authorised content provider security information for a CCN namespace. Additionally, NDN agents use a local trust model rooted at the content provider supported by the SPKI/SDSI certificate standard. These models provide scalable mechanisms to manage global and local namespace allocation, and manage the life cycle of public keys for content providers and NDN agents. Furthermore, we provide details of Savant's protocols to manage agent discovery and identity management, and log commitment between NDN agents. This is in addition to mechanisms for detecting log tampering using hash chains, log auditing, and proving key ownership using the challenge response protocol. Savant also provides various processes to encrypt authenticated interest commands and published log objects, prevent timing and replay attacks, determine access permissions using ACLs, and so forth.

In the next chapter, we analyse the behaviour of the Savant system we implemented and also the architecture's scalability based on simulations using ndnSIM, an ns-3 module that allows discrete-event network simulation for Internet systems.

4.8 Chapter Summary

This chapter outlines the design of the Savant accounting and accountability framework for NDN content distribution. It grew out of the requirements for effective content distribution identified in Chapter three, which was based on our analysis of content distribution architectures in Chapter two. Existing effective models for content distribution rely on centralised trust to deliver content with adequate quality, speed and reliability. However, this is not a sufficient model for content distributed from primarily untrusted infrastructure. The aim of the Savant framework is to address this challenge for an ICN packet-level architecture, where the content is distributed from primarily untrusted infrastructure. We identified the architectures requirements in Section 4.1. The remainder of the chapter outlines how we satisfy these requirements. This was accomplished using cryptographic techniques, PKI security processes, information flow processing mechanisms and ICN principles to collect, control, audit and aggregate accounting and accountability information in near real-time in a scalable and efficient manner.

Chapter 5

Savant: An Implementation

This chapter discusses our approach to implementing the Savant framework outlined in Chapter 4. To develop the whole architecture would be a very substantial undertaking. However, we have implemented several components to support a proof-of-concept accounting and accountability framework for video distribution using the NDN ICN architecture. Components implemented include: client and cache accounting and accountability functionality; the log commitment protocol outlined in Section 4.5.2; accountability engine processes such as log auditing, hash chain verification, aggregator functions and authenticated interest commands; and basic security functionality. We outline how we implemented these components and experiments carried out to validate the architecture in the remainder of this chapter.

5.1 Savant Design and Implementation

We implemented Savant[1] accounting and accountability components using Java SE 7, which runs on the Ubuntu 14.04.3 LTS operating system. We used the NDN v0.3.4 library [NDN, 2015] for security, routing, caching, and so forth and GStreamer 0.10[2], which is a pipeline based multimedia framework for playing, recording, streaming, and editing media content.

[1]The full implementation consists of over 50 Java classes and about 7,000 lines of code (including about 3,000 lines of my own code), which includes elements for NDN, security, GStreamer, Apache Spark and accounting and accountability. The library produces independent jar files for the NDN client, NDN cache and accountability engine components. Access to bitbucket code available on request from collindi@tcd.ie.

[2]GStreamer 0.10 is available from https://gstreamer.freedesktop.org/documentation/gstreamer010.html. GStreamer-Java was also required as the client player was written in Java https://github.com/gstreamer-java

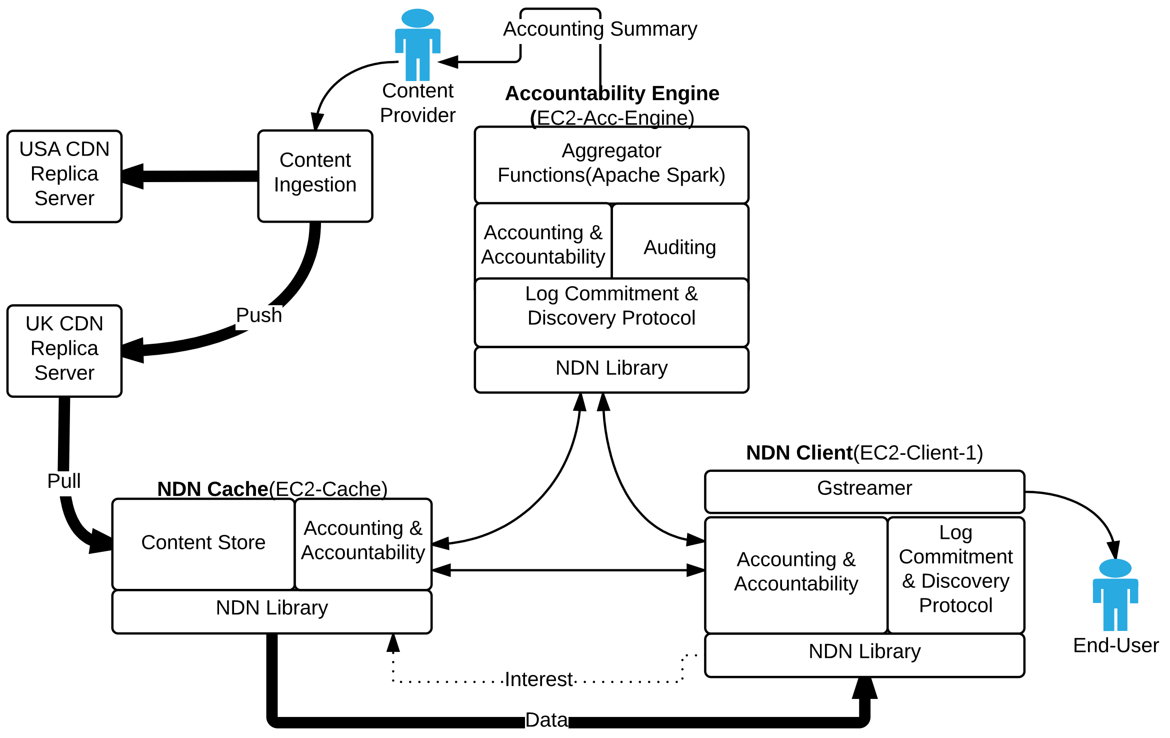


Figure 5.1: A High Level View of Communicating Entities and Components in Savant

Moreover, accountability engine components use the Apache Spark library[3], which is a framework for real-time event stream processing and data aggregation [Zaharia et al., 2012]. Furthermore, all client, cache and accountability engine elements ran on Amazon EC2 instance machines with the following configuration: m3.medium, CPU=1, memory=3.75GB. These libraries and configuration allowed us to focus exclusively on producing, collecting, aggregating and auditing accounting and accountability information for the NDN content distribution process rather than solving NDN routing, video player or data aggregation issues.

5.1.1 High Level Components

The Savant architecture is relatively simple. A high-level view of libraries and components utilised by Savant is depicted in Figure 5.1. All entities use the *NDN Library* for requesting and publishing content. Additionally, they all use an *Accounting & Accountability* process to publish log objects using NDN principles.

[3]Available from http://spark.apache.org

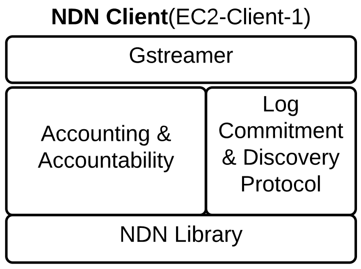

NDN Client(EC2-Client-1)
Gstreamer
Accounting & Accountability
Log Commitment & Discovery Protocol
NDN Library

Figure 5.2: NDN Client

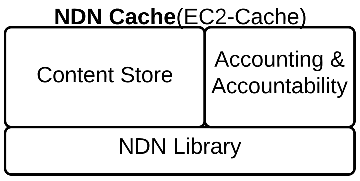


Figure 5.3: NDN Cache

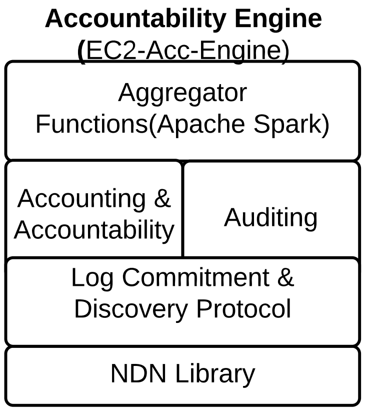

Accountability Engine (EC2-Acc-Engine)
Aggregator Functions(Apache Spark)
Accounting & Accountability
Auditing
Log Commitment & Discovery Protocol
NDN Library

Figure 5.4: Accountability Engine

NDN Client

The NDN client (in Figure 5.1 and Figure 5.2) renders video to the end-user using the *GStreamer* multimedia player. GStreamer sends interests for content using the *NDN Library* when the buffer is empty or goes below a certain threshold. The NDN cache replies to interests received with content objects. Every time an event occurs on the GStreamer player based on the minimal accounting scenario defined in Section 4.2.6 and Section 5.4, a *log entry* is created and linked to the previous log entry with a hash chain by the accounting and accountability component. Additionally, a log entry is created each time an interest or content chunk matching the statistical sample (methodology described in Section 5.2.2) is encountered by the player. Every 15 seconds[4] a batch of log entries is produced as one NDN *published log object* (see Section 4.2.5) by the NDN client.

NDN Cache

The primary function of the NDN cache (in Figure 5.1 and Figure 5.3) is to accept interests from the NDN client and respond with chunks of content. To simplify the implementation, all content is retrieved from the local NDN cache *content store* i.e. it is not forwarded to an upstream cache or content provider. We used the standard NDN cache component (in NDN

[4]Fifteen seconds is currently the default interval. However, it can be reconfigured on-demand by sending an authenticated interest command to a pair of NDN agents or to all NDN agents by the content provider. This process is outlined in more detail in Section 4.2.11 and Section 5.6 for additional information

v0.3.4 library) with code modifications to gather a statistical sample (methodology described in Section 5.2.2) when content is retrieved. Sampled data is sent to a Java process, which performs hash chaining, accounting and accountability functionality. This component also collects and processes accounting metrics related to uplink bandwidth usage. Like the client, the cache publishes a log object at intervals of 15 seconds, which contain a batch of log entries for the content provider.

Accountability Engine

The accountability engine (in Figure 5.1 and Figure 5.4) is built as a standalone Java process. We implemented functionality to collect published log objects from NDN agents (which is supported by the NDN library) and aggregate data received (supported by the Apache Spark framework (see Aggregator Functions in Section 4.2.10)). We also developed functionality to *audit* hash chain state and verify the integrity and provenance of published log objects received. Furthermore, the accountability engine assists NDN clients and caches with accountability. This component was developed based on the statistical sampling technique (outlined in Section 5.2.2) and the log commitment protocol outlined in Section 4.5.2.

To support the log commitment protocol, NDN agents send the *interestId* or *contentId* and the *logId* of the locally created published log object to the accountability engine using a standard NDN interest. The accountability engine responds with a published content object in response to the interest received, which contains the corresponding agents *logId*. NDN agents perform log commitment by requesting the opposing agents published log object and committing the associated digital signature to their log.

Finally, the collection of additional accounting metrics at NDN agents is achieved using authenticated interest commands. In the current Savant implementation, authenticated interest commands are generated and sent manually when an issue is detected within NDN agent infrastructure.

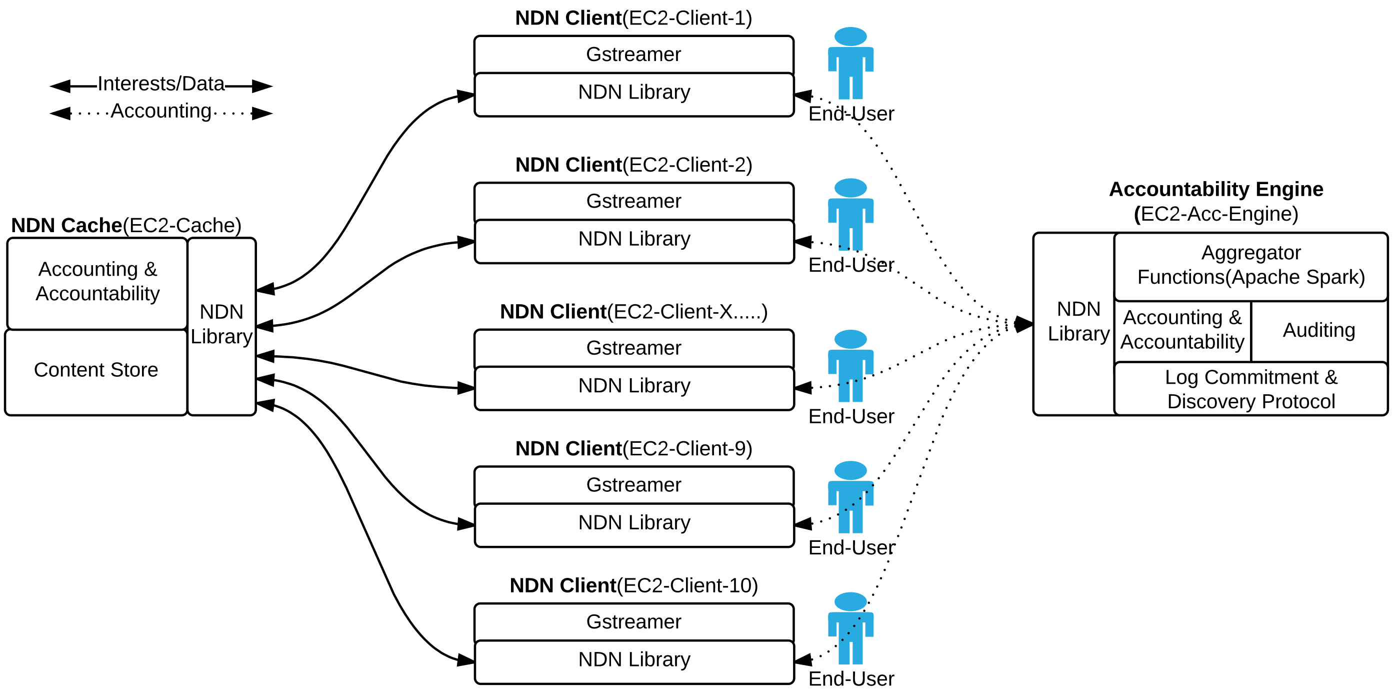

Interests/Data
Accounting
NDN Client(EC2-Client-1)
Gstreamer
NDN Library
End-User
NDN Client(EC2-Client-2)
NDN Client(EC2-Client-X.....)
NDN Client(EC2-Client-9)
NDN Client(EC2-Client-10)
NDN Cache(EC2-Cache)
Accounting & Accountability
Content Store
NDN Library
Accountability Engine (EC2-Acc-Engine)
Aggregator Functions(Apache Spark)
Accounting & Accountability
Auditing
Log Commitment & Discovery Protocol

Figure 5.5: Ten Independent NDN Agents Running Gstreamer, One NDN Cache Distributing Content and One Accountability Engine Collecting Published Log Objects.

5.2 Considerations for Accounting and Accountability

In this section, we identify several considerations for collecting accounting and accountability information for content distributed in a packet-level ICN architecture such as NDN. In the NDN v0.3.4 library, chunk size typically ranges between 1024 bytes to 8800 bytes[5]. Many of the following considerations would not be necessary if content chunks were larger than a maximum of 8.8KB e.g., 500KB, 1MB, and so forth. Additionally, it proved useful to define the accounting information to collect based on existing standards because the volume of logs collected can get very large. Many of the considerations in this section were the result of problems experienced with early versions of the Savant architecture. In some cases issues were discovered when we deployed several NDN agents running full client functionality on Amazon EC2 instances, similar to the overview depicted in Figure 5.5.

[5]NDN uses a default chunk size of 4KB (roughly three IP packets of 1500 bytes), which limits the negative effect on performance if one fragment is lost due to congestion. However, as CCN is an evolving Future Internet Architecture (FIA), the current CCN 1.0 implementation allows chunk size up to 64KB [PARC, 2015]. Other ICNs, such as MultiCache, support 16MB chunks [Katsaros et al., 2011].

5.2.1 Packet-Level Architecture

As mentioned in Section 4.6.3, chunk size varies across ICN architectures due to the overhead required for asymmetric cryptography, which is about 650 bytes per chunk regardless of chunk size [Salsano et al., 2012]. Savant uses chunk sizes of 8800 bytes, which is the maximum allowed by the NDN v0.3.4 library. This is to try and reduce the accounting and accountability overhead associated with the log commitment protocol (see Section 4.5.2).

5.2.2 Statistical Sample

It is difficulty for scalability purposes to perform log commitment and verification for every individual content chunk distributed in a packet-level ICN architecture. As a result, we take a statistical sample of content chunks distributed. This was achieved using the content chunk number, a sampling interval, the modulo operation at NDN agents and support from accountability engine infrastructure. This process is supported by the following mechanisms:

- In NDN, every content chunk is numbered. Chunk numbers start at 0 and increment by 1 until the end of the file.

- We used the *systematic sampling* technique to select the sampling interval, where the si^{th} content chunk is selected. The sampling interval si, is calculated as: $si = \frac{P}{p}$, where P is the amount of NDN content chunks created by a file during the content ingestion process and p is the sample size.

- For ease of development, we statically configured the sampling interval at distributed NDN agents. However, this value can also be specified in metadata during content ingestion or configured remotely when required at NDN agents by accountability engine infrastructure using authenticated interest commands.

- The modulo operation finds the remainder after dividing the chunk number by the statistical sample size. If it equals zero, then it performs log commitment, as outlined in the protocol in Section 4.5.2

5.2.3 Security

All entities that publish content and log objects in the Savant architecture including NDN agents, accountability engine infrastructure, content providers, and so forth, require a public/private key pair. This enables integrity, trust and provenance to be established in content and log objects received, as discussed in Section 2.3.5. However, our security implementation only provides enough support for basic proof of concept. To simplify the development, testing and auditing process, all RSA keys are hard-coded at each entity and located in a local database table. In the current implementation, all entities utilise RSA 2048-bit encryption keys for signing content and log objects. Savant can be extended in the future to use the SPKI/SDSI PKI model [Clarke, 2001] and CCN-KRS framework [Mahadevan et al., 2014]. Additionally, tamper evident logs are supported using the SHA-1 hash function for linking log entries and published log objects.

5.2.4 Namespace

DNS is a distributed hierarchical name service database for the Internet, which is separate from the IP architecture. Similarly, namespace management is not a component of the NDN architecture. However, it is a fundamental element required for its successful operation [Zhang et al., 2014]. All publishing entities in the NDN architecture implicitly require a globally routable name. We envision using an ICANN-like entity to coordinate namespace allocation to content providers. As discussed in Section 4.3.1, this is supported by the SPKI/SDSI distributed trust model, which maps a small-world model of trust on to public keys and local namespaces. In Savant, this translates to trust originating from the content provider (local principal) to NDN agents. Consequently, NDN agents publish log objects in a sub-namespace of the content provider. Moreover, we expect NDN agents to interact with multiple content providers and publish log objects in multiple different namespaces. For example:

- /netflix/agent-id/logs
- /hbo/agent-id/logs

- /bbc/agent-id/logs

In this implementation, all entities publish under the */tcd* namespace. For example:

- /tcd/data
- /tcd/client-1/logs
- /tcd/ndn-router/logs
- /tcd/accountability-engine/logs

5.2.5 Prerequisites and Assumptions

To reduce the programming overhead associated with the following prerequisites and assumptions, we manually configure namespaces and routing prefixes at every NDN entity operating on Amazon EC2 in the current Savant implementation.

Full Agent Connectivity

Full agent connectivity implies that all NDN entities should be able to establish direct communication with all other NDN entities. NDN operates as an overlay network on top of the IP Internet using UDP. Consequently, all entities must have an IP address and several UDP ports open. For example, the NDN *Network Forwarding Daemon* (NFD), which is run by every NDN entity requires UDP port 6363 open.

All Entities Publish Under a Unique namespace

In order to publish content in Savant, all entities must publish under a unique namespace. However, any NDN entity can publish under any NDN namespace. As a result, when requesting published content, the requestor e.g., accountability engine or NDN agent, should also specify content publisher security information such as the public key certificate or content digests.

Routing

For routing purposes, all entities must be associated with a NDN prefix and an IP address. For example, all NDN agents need a route entry for the accountability engine. The following command creates a UDP tunnel to the remote accountability engine NFD at an NDN agent:

- nfdc register /ndn/accountability-engine udp://<IP-ADDRESS>

The accountability engine should also have a *back route* to NDN agents, which is supported by the agent discovery protocol. This enables accountability engine infrastructure to collect published log objects from NDN agents for auditing and aggregation purposes.

Agent Discovery Protocol

To access the Savant network, NDN agents need to make contact with the accountability engine, which is supported by the NDN agent discovery protocol (as outlined in Section 4.5.1). Metadata in content received specifies the accountability engine responsible for collecting accounting and accountability information for a content provider. This enables the accountability engine infrastructure to collect published log objects and support the log commitment protocol between NDN agents. However, to reduce the development overhead in this implementation, the accountability engine infrastructure is manually configured at each NDN agent.

5.3 Savant Business Model

The free business model utilised by web and transparent cache systems discussed in Section 2.1.1 and Section 2.1.5 is based on decreasing capital and operational costs by localising traffic to cache servers with available content in a company or ISPs network. We advocate using a similar model with a Savant ICN system, where cache servers take part in the content distribution process without requiring payment. Their reward is non-financial and delivers the following benefits to content providers:

1. Reliable trustworthy analytic information associated with content distributed from their network. This functionality is supported by the Savant framework.

2. Reduced costs associated with distributing data from expensive CDN infrastructure or content provider equipment, as data is now delivered from a local network cache server.

3. Network operators can offer better QoE to end-users then the content provider as data is located close to subscribers.

4. Using the web cache business model reduces the amount of relationships or business agreements a content provider needs to maintain with cache providers, which could potentially run into thousands worldwide.

We anticipate that these benefits will deliver financial savings within their own organisations that will compensate for the effort of taking part in Savant dialogues. In summary, the free web/transparent cache business model helps both network operators and content providers reduce costs and infrastructure overheads associated with requesting, distributing, and delivering frequently requested content to end-users, while continuing to offer reliable and trustworthy analytic information for the content distribution process.

5.4 Accounting Scenarios

Significant amounts of accounting metrics can be collected by Savant. Such metrics relate to (see Section 3.2): advertising, record of ads pushed to users; network performance, quality of service; demographics, data characterizing users; and so forth. This data can help diagnose system and network problems, determine the success of a marketing campaign or bill clients for movies watched. Listing each metric individually is outside the scope of this work. However, we did define and use one[6] ICN accounting and accountability scenario for use in the NDN video content distribution session (see Section 4.2.6): minimal accounting. We also define the *default* scenario, which serves as the control scenario.

[6]In previous versions of Savant we also defined and implemented a *maximal* accounting scenario, see Section 4.2.6. However, it quickly became apparent that gathering accounting and accountability information for all content chunks distributed in a packet-level ICN architecture is a major challenge. This is why we chose to gather a statistical sample of content chunks distributed, see Section 5.2.2

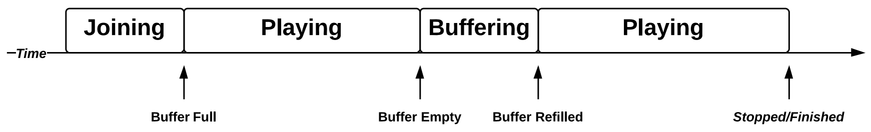


Figure 5.6: Video Player State Machine (adapted from [Dobrian et al., 2013]).

5.4.1 Control Scenario: Default

We compare the minimal accounting scenario against a set of base values for CPU consumption, uplink and downlink bandwidth usage. See Figure 5.9 for an example of metrics collected. We call this control scenario: *default*. The default scenario shows the overhead of the NDN content distribution process on NDN client and NDN cache infrastructure without any accounting or accountability functionality turned on. In the experiments in Section 5.8, we evaluate the Savant minimal accounting and accountability scenario against this scenario.

5.4.2 Minimal Accounting

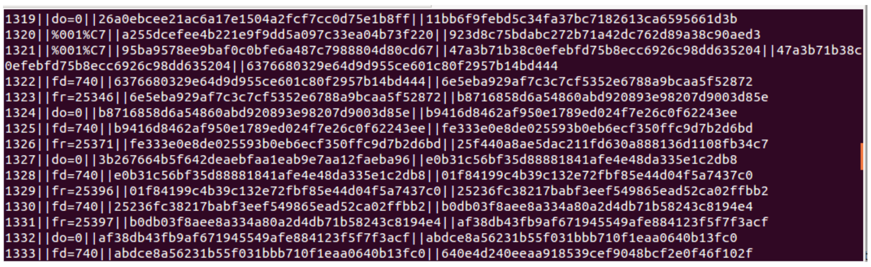
The *minimal* accounting scenario collects industry standard QoE metrics based the on the player state machine in Figure 5.6 (adapted from [Dobrian et al., 2013]). Client metrics monitored include [Dobrian et al., 2013]: join-time, which is the time it takes for the buffer to fill up and start playing; buffering ratio, which is the fraction of total session time (i.e. playing time plus buffering time) spent buffering; rate of buffering events, which is the frequency of buffering interruptions; rendering quality, which is the frames displayed per second on the client; and average downlink bitrate e.g., 200kb/s. We also collect a statistical sample of content chunks distributed, which is supported by the log commitment protocol outlined in Section 4.5.2. The collection, processing and aggregation of these metrics form the basis of our experiments in Section 5.8. The minimal scenario also has support for hash chaining (see Section 4.2.5), log commitment (see Section 4.5.2) and accountability (see Section 4.2.1). Table 5.1 shows the log metrics collected for the minimal accounting scenario while Figure 5.7 and Figure 5.8 show screenshots of log entries collected.

Table 5.1: Example of Log Entries Collected by NDN Client and Cache Elements

Log Entry (NDN Client)	Hash h_i	SeqNo. s_i	Content Type $t_i c_i$	h_{i-1}
Interest Sent	h_i	1327	interest_s=%00%02%C3	h_{i-1}
Content Received	h_i	1328	content_obj_r=%00%02%C3	h_{i-1}
Downlink BW	h_i	1329	do=80233	h_{i-1}
Frames Dropped	h_i	1330	fd=697	h_{i-1}
Frames Rendered	h_i	1331	fr=25346	h_{i-1}
Buffering Event	h_i	1332	be=1	h_{i-1}
Buffer Empty Event	h_i	1333	nd=1	h_{i-1}
Duration End of Stream	h_i	1334	st=end-of-stream	h_{i-1}

Log Entry (NDN Cache)	Hash h_i	SeqNo. s_i	Content Type $t_i c_i$	h_{i-1}
Interest Received	h_i	2100	interest_r=%00%02%C3	h_{i-1}
Content Sent	h_i	2101	content_obj_s=%00%02%C3	h_{i-1}
Uplink BW	h_i	2102	up=51040	h_{i-1}
CPU Usage %	h_i	2103	cpu=39	h_{i-1}

Note: Hash entries are quite large for the SHA-1 message digest algorithm. For clarity they have been removed from this table. However, the following is an example of both an h_{i-1} and h_i hash value e.g., b8716858d6a54860abd920893e98207d9003d85e

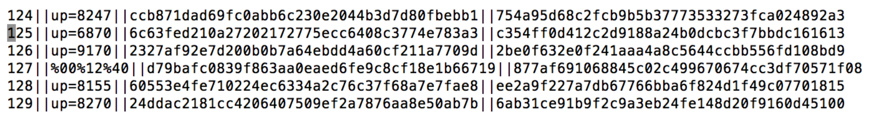


```
1319||do=0||26a0ebcee21ac6a17e1504a2fcf7cc0d75e1b8ff||11bb6f9febd5c34fa37bc7182613ca6595661d3b
1320||%001%C7||a255dcefee4b221e9f9dd5a097c33ea04b73f220||923d8c75bdabc272b71a42dc762d89a38c90aed3
1321||%001%C7||95ba9578ee9baf0c0bfe6a487c7988804d80cd67||47a3b71b38c0efebfd75b8ecc6926c98dd635204||47a3b71b38c
0efebfd75b8ecc6926c98dd635204||6376680329e64d9d955ce601c80f2957b14bd444
1322||fd=740||6376680329e64d9d955ce601c80f2957b14bd444||6e5eba929af7c3c7cf5352e6788a9bcaa5f52872
1323||fr=25346||6e5eba929af7c3c7cf5352e6788a9bcaa5f52872||b8716858d6a54860abd920893e98207d9003d85e
1324||do=0||b8716858d6a54860abd920893e98207d9003d85e||b9416d8462af950e1789ed024f7e26c0f62243ee
1325||fd=740||b9416d8462af950e1789ed024f7e26c0f62243ee||fe333e0e8de025593b0eb6ecf350ffc9d7b2d6bd
1326||fr=25371||fe333e0e8de025593b0eb6ecf350ffc9d7b2d6bd||25f440a8ae5dac211fd630a888136d1108fb34c7
1327||do=0||3b267664b5f642deaebfaa1eab9e7aa12faeba96||e0b31c56bf35d88881841afe4e48da335e1c2db8
1328||fd=740||e0b31c56bf35d88881841afe4e48da335e1c2db8||01f84199c4b39c132e72fbf85e44d04f5a7437c0
1329||fr=25396||01f84199c4b39c132e72fbf85e44d04f5a7437c0||25236fc38217babf3eef549865ead52ca02ffbb2
1330||fd=740||25236fc38217babf3eef549865ead52ca02ffbb2||b0db03f8aee8a334a80a2d4db71b58243c8194e4
1331||fr=25397||b0db03f8aee8a334a80a2d4db71b58243c8194e4||af38db43fb9af671945549afe884123f5f7f3acf
1332||do=0||af38db43fb9af671945549afe884123f5f7f3acf||abdce8a56231b55f031bbb710f1eaa0640b13fc0
1333||fd=740||abdce8a56231b55f031bbb710f1eaa0640b13fc0||640e4d240eeaa918539cef9048bcf2e0f46f102f
```

Figure 5.7: Savant Client Log Example

5.5 Log Entries

As discussed in 4.2.5, a log entry e_i has the following properties: $e_i = (h_i, s_i, t_i, c_i)$. This corresponds to a sequence number s_i (always increasing), a type t_i (e.g., *Interest*, Content or Event), type-specific content c_i (e.g., content name, QoE, and so forth). Moreover, Section 4.2.5 outlined how a tamper-evident log is maintained between each log using a recursively defined hash chain h_i. This links the current log object with the previously published log object h_{i-1}. Table 5.1 shows example log entries for client and cache elements such as uplink and downlink bandwidth consumed, a statistical sample of content chunks distributed, and so forth. Figure 5.7 and Figure 5.8 show screenshot examples of real client and cache log entries.



```
124||up=8247||ccb871dad69fc0abb6c230e2044b3d7d80fbebb1||754a95d68c2fcb9b5b37773533273fca024892a3
125||up=6870||6c63fed210a27202172775ecc6408c3774e783a3||c354ff0d412c2d9188a24b0dcbc3f7bbdc161613
126||up=9170||2327af92e7d200b0b7a64ebdd4a60cf211a7709d||2be0f632e0f241aaa4a8c5644ccbb556fd108bd9
127||%00%12%40||d79bafc0839f863aa0eaed6fe9c8cf18e1b66719||877af691068845c02c499670674cc3df70571f08
128||up=8155||60553e4fe710224ec6334a2c76c37f68a7e7fae8||ee2a9f227a7db67766bba6f824d1f49c07701815
129||up=8270||24ddac2181cc4206407509ef2a7876aa8e50ab7b||6ab31ce91b9f2c9a3eb24fe148d20f9160d45100
```

Figure 5.8: Savant Cache Log Example

5.6 Log Object Frequency

To minimise the amount of log objects produced, collected, audited, aggregated, and so forth, the logging component periodically every 15 seconds produces one *published log object*. This is a configurable interval that can be changed per content provider, data producer, or at autonomous system (AS) level. A published log object, which is a digitally signed commitment to log integrity and provenance (see Section 4.2.5), contains a collection of log entries of recent QoE metrics and events that have occurred on the NDN agent over the last 15 seconds. A hash chain is maintained between each log entry and between published log objects, which is audited by the accountability engine. This process is supported by a playback video buffer on the client, which is roughly 30 seconds in length. As a result of this configurable parameter, the accountability engine can receive an aggregate of log entries at intervals from NDN agents and perform some corrective or investigative action on the client or cache before the end-user encounters problems i.e., before the playback buffer empties. The interval with which published log objects are produced can be modified before data is published by the content provider or while data is being sent or received by the cache or subscriber elements using authenticated interest commands (see Section 4.2.7).

5.6.1 Authenticated Interest Commands

One of the defining benefits of the Savant framework is its ability to monitor the content distribution process by collecting a subset of the information available in the taxonomy in Section 3.2 in near real-time, while maintaining the flexibility to gather additional accounting information or change certain parameters on remote agents (such as log frequency) when required. This is achieved using authenticated interest commands. The commands implemented for the Savant framework are outlined in detail in Section 4.2.7. The idea here is that due to the expected scale of the system e.g., millions of nodes, basic or minimal accounting information should be collected from all NDN agents. However, if additional metrics are required to investigate issues outlined in the taxonomy in Section 3.2 such as network performance issues, server problems, advertising impressions, and so forth, then the content provider has the tools to activate the collection of additional metrics to help satisfy these requirements.

5.7 Experiment Setup, Evaluation and Results

Our main objective during implementation was to show that the Savant system could distribute NDN video chunks efficiently while producing accounting and accountability information, and scale to potentially millions of NDN agents. To achieved these goals, we ran the Savant client, cache and accountability engine components on separate Amazon Elastic Compute Cloud (EC2) instances[7] to test and evaluate their capabilities in a real-world content distribution environment. A complete list of the environment, software utilised and code to reproduce experiments is available in Section 5.1. We also used ndnSIM to simulate the production of aggregated accounting information by local accountability engine infrastructure representing data from thousands of NDN agents. This information is collected and aggregated further by a hierarchical accountability engine. In the remainder of this section, we provide an overview of our methodology to evaluate these objectives.

5.7.1 Experiment Overview

During the final two years of the PhD I ran thousands of Savant NDN experiments. Initially, I started using the python version of NDN video already developed by UCLA Computer Science Department [Kulinski & Burke, 2012]. However, it quickly became apparent that the industry standard QoE metrics outlined in the minimal accounting secnario in Section 4.2.6 (adapted from [Dobrian et al., 2013]) could not be captured easily using this system. This required building a new custom GStreamer Java player, which is outlined in Section 5.1, to support capturing these metrics. However, this was a complex undertaking and every

[7]As discussed above in Section 5.1, all Amazon EC2 instance machines refer to the following configuration: m3.medium, CPU=1, memory=3.75GB.

time the software was updated to support new Savant functionality, fix bugs, or change variable parameters, the results from old experiments effectively became useless. For example, typical variables that changed frequently included: collecting a larger/smaller sample size of content chunks distributed, using more powerful EC2 instance machines, changing the message digest algorithm e.g., between MD5, SHA-256, SHA-1, or collecting published log objects at intervals of milliseconds, 1 second, 10 seconds, 15 seconds, 20 seconds, and so forth.

5.7.2 Experiment Configuration

Experiments were scheduled to stream data from the NDN cache and render video[8] at the NDN client continuously at two minute intervals from the unix cron table for different durations of 24, 48, 72, and 96 hours corresponding to 720, 1440, 2160, and 2880 independent experiments respectively. There are several reasons why Savant's experiments were run at these intervals including:

1. The Savant client video software, cache component and accountability engine element had reliability issues. All three components needed to start up and run successfully for 1 minute 45 second periods[9], otherwise the experiment failed. If any element failed, the QoE metrics generated by all other components were also invalid and discarded.

2. Most fluctuations in the *GStreamer* multimedia player, client, cache, accountability engine and network that caused QoE events, typically happened within the first 80 seconds of the content distribution process. The accounting metrics collected by Savant are detailed in Table 5.1. Long periods of stability typically followed this initial start-up period, possibly due to an efficient EC2 environment, which resulted in minimal QoE activity.

[8]The video 'CLANNAD - Live At Christ Church Cathedral, Dublin' was just over 57 minutes long and available from https://vimeo.com/35666925

[9]The length of time each experiment ran during the two minute cron table interval was 1 minute 45 seconds. It takes 15 seconds to stop, restart and make sure the NDN client, cache, and accountability engine processes are up and running. This delay was primarily required to ensure the NDN cache element retrieved data from its long-term content store repository, and not the local cache store.

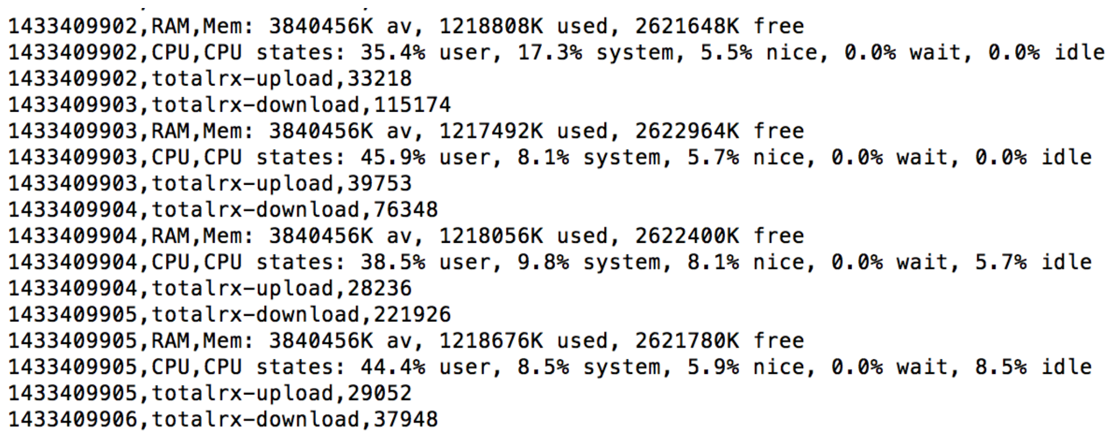

```
1433409902,RAM,Mem: 3840456K av, 1218808K used, 2621648K free
1433409902,CPU,CPU states: 35.4% user, 17.3% system, 5.5% nice, 0.0% wait, 0.0% idle
1433409902,totalrx-upload,33218
1433409903,totalrx-download,115174
1433409903,RAM,Mem: 3840456K av, 1217492K used, 2622964K free
1433409903,CPU,CPU states: 45.9% user, 8.1% system, 5.7% nice, 0.0% wait, 0.0% idle
1433409903,totalrx-upload,39753
1433409904,totalrx-download,76348
1433409904,RAM,Mem: 3840456K av, 1218056K used, 2622400K free
1433409904,CPU,CPU states: 38.5% user, 9.8% system, 8.1% nice, 0.0% wait, 5.7% idle
1433409904,totalrx-upload,28236
1433409905,totalrx-download,221926
1433409905,RAM,Mem: 3840456K av, 1218676K used, 2621780K free
1433409905,CPU,CPU states: 44.4% user, 8.5% system, 5.9% nice, 0.0% wait, 8.5% idle
1433409905,totalrx-upload,29052
1433409906,totalrx-download,37948
```

Figure 5.9: Metrics were Collected at 1 second intervals by an Experiment System Monitoring Script

5.7.3 Data Analysis

The data collected and averaged by the experiment monitoring script and outputted to a log file every second includes (see sample log data in Figure 5.9): CPU consumption, RAM used, uplink and downlink bandwidth usage. Every two minutes a new log file was created on the client, cache and accountability engine machines corresponding to the next experiment run. After each set of experiments completed, all log files were collected from all nodes and processed by a log-analyser[10] script. This script supports the removal of invalid data elements and outliers. For example, it was expected that the uplink and downlink throughput should average between 95 kbps and 100 kbps for a 1 minute 45 second period. If the average was outside these values then the log-analyser script assumed there was an experimental error during the content distribution process and the data was discarded. The log-analyser also computed the average for all experiments completed during the 24, 48, 72, or 96 hour period.

5.8 Experiments

The following experiments were run to:

1. Establish the overhead of the Savant accounting and accountability process.

2. Determine if the system can detect faults.

[10]The log analyser code is available on request from collindi@tcd.ie

	Scenario	CPU Utilisation %	Downlink(kb/s)	Uplink(kb/s)
1 Client	Default	88%	98.0	1.3
	Minimal	89%	98.4	1.4
1 Cache	Default	6%	1.1	99.5
	Minimal	10%	1.2	99.6
1 Acc-Engine	2 Agents	-	0.22	0.05

Table 5.2: Results for Default and Minimal Scenarios

3. Check if the accountability engine component is scalable.

4. Verify if Savant system can scale.

5.8.1 Savant Accounting and Accountability Overhead

To demonstrate Savant's capabilities, the first experiment compares accounting scenarios: *default* and *minimal*. The default or control scenario shows the overhead of the NDN content distribution process on client and cache infrastructure without any accounting or accountability functionality turned on. This serves as a baseline scenario to evaluate minimal accounting against. We implemented the minimal scenario defined in Section 5.4 and Section 4.2.6 with support for hash chaining (see Section 4.2.5), log commitment (see Section 4.5.2) and accountability (see Section 4.2.1). Metrics were gathered from independent elements running on dedicated EC2 instance machines. The three machines deployed on EC2 are conceptually similar to the client, cache and accountability engine elements illustrated in Figure 5.1.

The results in Table 5.2 shows the overhead created by the NDN content distribution process based on the default and minimal scenarios on the client and cache infrastructure. We see that there is a small increase in CPU utilisation on the client and the cache with an average increase of just over 2.5% for the minimal scenario in comparison to the baseline. This is good as it demonstrates the accounting and accountability overhead on the client and cache infrastructure is small. Additionally, we believe that implementing Savant in efficient C code would add significant improvements to overall performance. Furthermore, based on a sampling interval of 0.3% of content chunks distributed (see Statistical Sample in Section 5.2.2), the sum of all logging data generated for the minimal scenario i.e. uplink and downlink bandwidth usage, log commitment, and so forth, for client, cache and accountability

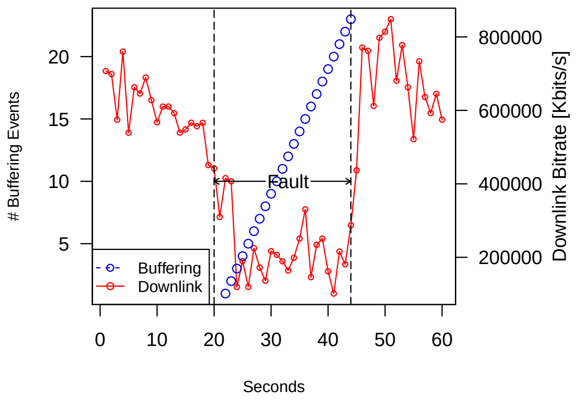


Figure 5.10: NDN Client experiencing a fault before, during and after its introduction

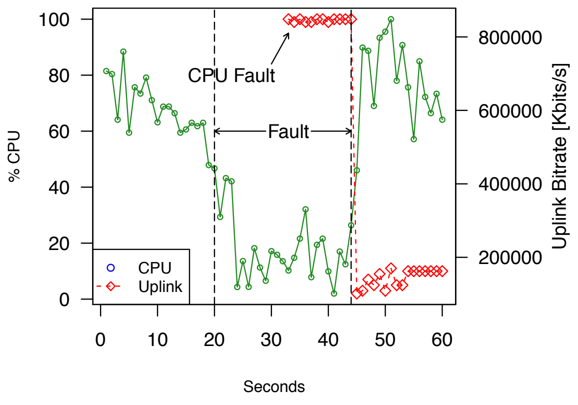


Figure 5.11: NDN Cache with fault before, during and after its introduction

engine entities comes to a total of 0.48% increase in comparison to the baseline scenario.

This experiment shows that Savant's overhead creates only a small increase in resource usage (i.e. CPU, uplink and downlink bandwidth) on client, cache, accountability engine and network infrastructure for the minimal accounting and accountability scenario.

5.8.2 Fault Detection

Utilising the prototype implementation outlined in Section 5.8.1, we engineering a fault into the NDN content distribution process to demonstrate Savant's ability to detect, isolate and resolve problems. During a normal content distribution session (such as outlined in Section 5.8.1), we introduced a rogue process on the NDN cache to consume all available CPU resources.

Figure 5.10 and Figure 5.11 show an NDN client and an NDN cache before, during and after the introduction of this rogue process. On the client in Figure 5.10, we see a drop in *downlink* bandwidth consumption and an increase in the number of *buffering interruptions* occurring after the faults introduction. Similarly, on the cache in Figure 5.11, we see a drop in *uplink* bandwidth consumption after the faults introduction. These anomalies were detected by the generic aggregator function defined in Section 4.2.10 running on the accountability engine instance on Amazon EC2. Based on the information gathered, we manually sent an authenticated interest command to activate collection of additional CPU statistics on all NDN agents suspected of being the source of the fault. Based on the latest metrics collected

Accountability Engine		
CPU	PLOPS	Clients
100%	109	1,635
58%	77	-
50%	59	-
28%	27	-
23%	13	-

Figure 5.12: PLOPS Processed by an Accountability Engine on EC2

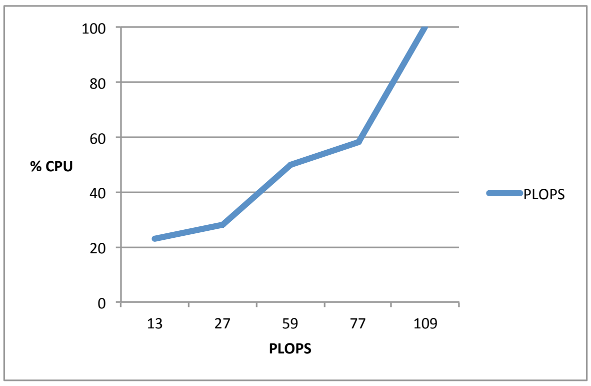


Figure 5.13: Graph of PLOPS from Figure 5.12

and aggregated by the generic aggregator function, we identified a CPU resource problem on the NDN cache (see Figure 5.11). To resolve the problem, we identified and killed the problematic process. This caused the downlink bandwidth and buffering interruptions to return to normal on the client, shown in Figure 5.10. Moreover, CPU utilisation and uplink bandwidth returned to normal on the cache, which is illustrated in Figure 5.11.

This scenario shows that the Savant framework can identify problems during the NDN content distribution process based on the minimal accounting scenario defined in Section 4.2.6 and using the aggregator functions described in Section 4.2.10. It also showed the collection of additional QoE metrics at NDN agents can be initiated easily using authenticated interest commands.

5.8.3 Accountability Engine Scalability

It was not feasible to take our working prototype and scale it to operate with a very large number of NDN agents. Consequently, in a separate experiment our goal was to determine how many NDN agents an accountability engine can support. This was achieved by measuring the amount of *published log objects per second* (PLOPS), based on simulated client and cache interactions with real published log objects, that an accountability engine can process before the instance saturates.

In this scenario, data is collected and content integrity and hash chain is state verified by

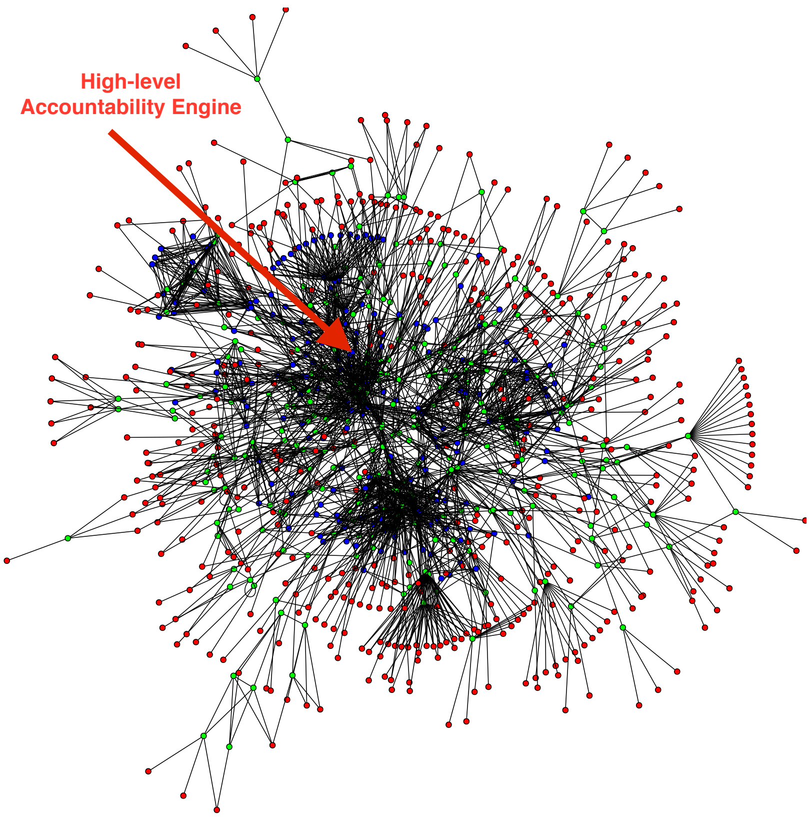


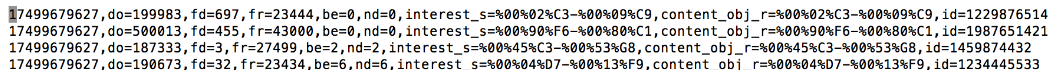
Figure 5.14: Rocketfuel's "Verio US" Internet topology with one high-level accountability engine that collects data from nine hundred and twenty low-level accountability engines.

the accountability engine element running on EC2. Figure 5.12 and Figure 5.13 show the increase in CPU resource usage the more PLOPS processed by the accountability engine, with instance saturation at 109 PLOPS. Based on this result, if each NDN agent produces data at 15-second intervals, then we can extrapolate that each accountability engine running on EC2 hardware can support about 1,600 concurrent NDN agents.

This experiment shows that even with a small EC2 instance machine, the overhead of collecting and validating published log objects is small, which should support the architecture's scalability.

5.8.4 NDNSim

Savant is designed to be scalable. This is supported by hierarchical (or high-level) accountability engine infrastructure (see Section 4.2.12), which collects and aggregates data from hundreds or thousands of low-level accountability engines that are located (geographically) close to end-users. To emulate this scenario, we used ndnSIM, an ns-3 module that allows



```
17499679627,do=199983,fd=697,fr=23444,be=0,nd=0,interest_s=%00%02%C3-%00%09%C9,content_obj_r=%00%02%C3-%00%09%C9,id=1229876514
17499679627,do=500013,fd=455,fr=43000,be=0,nd=0,interest_s=%00%90%F6-%00%80%C1,content_obj_r=%00%90%F6-%00%80%C1,id=1987651421
17499679627,do=187333,fd=3,fr=27499,be=2,nd=2,interest_s=%00%45%C3-%00%53%G8,content_obj_r=%00%45%C3-%00%53%G8,id=1459874432
17499679627,do=190673,fd=32,fr=23434,be=6,nd=6,interest_s=%00%04%D7-%00%13%F9,content_obj_r=%00%04%D7-%00%13%F9,id=1234445533
```

Figure 5.15: Sample NDN agent aggregate data received by a high-level accountability engine.

discrete-event network simulation for Internet systems and Rocketfuel's "Verio US" Internet topology. Rocketfuel is a tool that measures realistic router-level ISP topologies, while "Verio US" is an ISP topology that was mapped using the Rocketfuel technique [Spring et al., 2002]. The Verio US topology was selected for this simulation as it offers an environment with over nine hundred hierarchically connected nodes. This topology is illustrated in Figure 5.14.

In this simulation, we wanted to measure the uplink and downlink bandwidth overhead at *one* high-level accountability engine collecting aggregate data from over nine hundred low-level accountability engines. To achieve this goal using Rocketfuel's "Verio US" Internet topology, we classify one backbone node as a high-level accountability engine (see Figure 5.14), which collects summary data (represented as *InData* in Figure 5.16) using NDN interests (represented as *OutInterests* in Figure 5.16) from nine hundred and twenty low-level accountability engines (i.e. all other nodes) in the "Verio US" topology. Additionally, following the output from the previous experiment in Section 5.8.3, each low-level accountability engine simulates the production of summary statistics received at fifteen second intervals from 1,600 synchronous NDN agents running the minimal accounting and accountability scenario. A sample of summary statistics received from low-level accountability engines is depicted in Figure 5.15. These metrics are an aggregate of published log objects received from NDN agents including log entries from Table 5.1.

The result in Figure 5.16 shows over 73 Mbps of published log objects are received every second by the high-level accountability engine. This represents aggregate data received from almost 1.5 million synchronous NDN agents i.e. aggregate data received from nine hundred and twenty low-level accountability engines, each simulating the production of aggregate data from 1600 NDN agents. Moreover, interests sent i.e. *OutInterests* in Figure 5.16 appear constant at almost zero due to their low uplink bandwidth overhead.

This simulation shows that the Savant architecture can easily scale to millions of nodes.

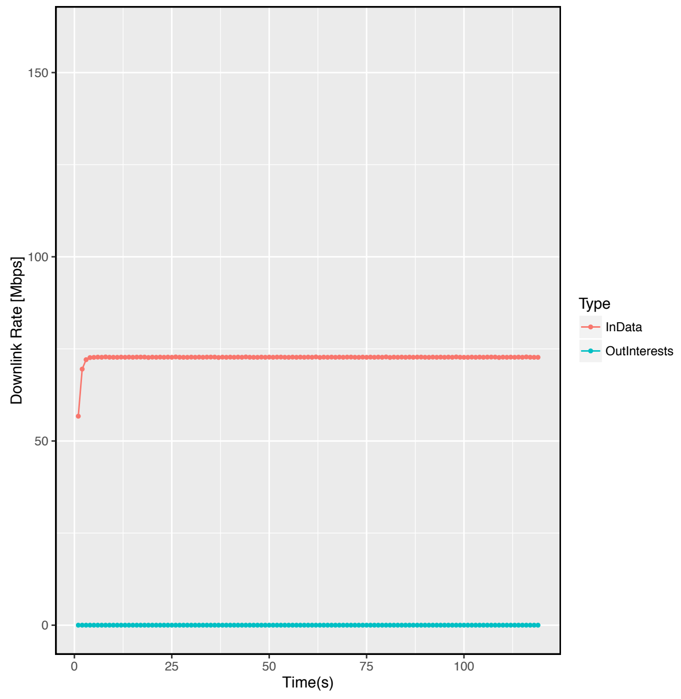


Figure 5.16: Total average downlink bandwidth consumed by a high-level accountability engine receiving summary statistics from almost 1.5 million NDN agents.

Moreover, utilising large EC2 instance machines, cloud auto-scaling capabilities and scalability through accountability engine hierarchy, we are confident the Savant architecture can scale to accommodate 10s of millions and possibly 100s of millions of NDN agents.

5.8.5 Accounting Overhead Projection

In 2010, Akamai processed over 100 TB of logs per day collected from edge server's handling over 10 million HTTP requests per second [Nygren et al., 2010]. Based on data gathered from the experiments in this section, we can estimate that Savant can support at least 1.5 million NDN agents sending and receiving around 8 million content objects per second. These agents generate about 6 TB of published log objects per day, which is collected and processed by Savant accountability engine infrastructure. This estimate is based on the data collected using the minimal accounting scenario described in Section 5.4. While not an ideal comparison, this is a large reduction in log size in comparison to the Akamai CDN model. We make some additional projections for larger numbers of subscribers in table 5.3.

Table 5.3: Estimated Accounting Overhead Based the Minimal Scenario

No. of NDN Agents	Content Objects (Millions)	Log Size (TB)
Akamai Unknown[Nygren et al., 2010]	10	100 TB
1.5 Million	8	6 TB
10 Million	53	40 TB
50 Million	267	200 TB
100 Million	533	400 TB
500 Million	2667	2000 TB

5.9 Lessons Learned

We end with a list of key lessons learned from implementing Savant.

First, collecting accounting and accountability information using the log commitment protocol outlined in Section 4.5.2 for every individual content chunk distributed in a packet-level ICN architecture is a major challenge. This is due to the volume of content chunks created. For example, in this implementation a 294MB video file was broken up into 35,004 independent content chunks of 8.8KB in size. Consequently, if accountability is required for every content chunk distributed it is worth considering using larger chunk sizes. Alternatively, gathering a random statistical sample of data chunks distributed is a viable option (which was demonstrated in this implementation).

Second, utilising cryptographic techniques to publish log objects and authenticated interests to send commands to NDN agents is a useful insight for ICN content distribution. These mechanisms give content providers the ability to control the content distribution process and gather in-depth reporting and analytics information with associated integrity and authenticity. They also offer the basic tools using public key cryptography for NDN agents to bill content providers for accounting and accountability information produced and commands executed. Furthermore, these tools offer a methodology to reduce the accounting overhead in comparison to current architectures such as CDNs, as discussed in Section 5.8.5.

Third, accountability engines offer detailed near real-time aggregate information locally while hierarchical accountability engine infrastructure offer summary information globally. Moreover, using the SPKI/SDSI PKI model where trust originates from a local principal, decisions that require support from authenticated interest commands (such as collecting additional accounting information, and so forth) can be created and sent by the accountability engine infrastructure located close to end-users. The scalability and responsiveness of a similar architecture that uses hierarchical infrastructure to support system monitoring while local infrastructure can execute control has already been validated by architectures such as C3 [Ganjam et al., 2015].

Fourth, the frequency of log production, collection and analysis should be almost proportionate to amount of video play time available (in seconds) in a clients buffer. However, this interval should provide enough time to support recovery if problems are encountered (e.g., before the buffer empties). Producing less published log objects also reduces the accounting and accountability overhead on client, cache and accountability engine infrastructure. Additionally, using authenticated interest commands, it is easy to increase or decrease the log production frequency when required. Again, near real-time log collection and analysis methods have been validated by Conviva's C3 platform when supporting remote clients [Ganjam et al., 2015].

Finally, if logging information is collected for every content chunk distributed, then it is easy to calculate the uplink and downlink bitrate at NDN agents. This simplifies client-side functionality as content providers only need to keep track of their own content and its interaction with the client-player (e.g., frame-rate, buffering events, and so forth). This is in contrast to the current method of collecting uplink and downlink bandwidth information from network interfaces.

5.10 Chapter Summary

In preceding chapters, we claim the Savant architecture can offer content providers the ability to use many different trusted and untrusted cache elements to distribute content while offering integrity and authenticity for reporting and analytics information produced. In this chapter we demonstrated Savant's capabilities via an implementation. We achieved our goals of determining: if the logging component was lightweight and robust; if it could help identify, isolate and correct faults; and if it could scale to accommodate 10s of millions of NDN agents. These attributes were demonstrated in several experiments described, which were run on Amazon EC2 instance machines and using ndnSIM. Moreover, this implementation was supported by several extensions and improvements to support the development of an efficient accounting and accountability framework for a packet-level ICN architecture. We concluded with a list of key lessons learned from implementing Savant in an ICN video content distribution environment. Finally, knowledge of these consideration and lessons learned should prove useful to future researchers implementing an accounting and accountability framework for an ICN Internet architecture.

Chapter 6

Conclusions and Future Work

In this thesis, we argue that the integrity and authenticity of the accounting information produced by a content distribution architecture is important as it provides valuable insight about the content viewed, network performance, end-users, and so forth. The premise of our work is that if an architecture does not provide this information it leads to its lack of adoption and utilisation by content providers and content distributors. For example, many architectures offer cheap and efficient content distribution mechanisms such as P2P, multicast IP, web caching, and so forth, but they fail to sufficiently address this accounting and accountability requirement leading to their limited adoption. In contrast, architectures such as CDNs that provide integrity for accounting information produced are being utilised extensively. However, CDNs have challenges related to end-user latency, large volumes of logging information produced, high costs, and so forth. Moreover, content providers need to trust the accounting information produced because they lack non-repudiable evidence of actions taken (i.e. accountability).

Furthermore, there has not been a comprehensive study of what elements and requirements constitute an efficient and successful content distribution system across current or future Internet architectures. In this thesis we developed and used two tools to help identify the drawbacks and merits of existing architectures based on systems surveyed. The first is a taxonomy for accounting and accountability information (summarised in Section 3.2) based on our analysis of logging information gathered from the surveyed systems. The second is a generic model (shown in Figure 3.1) for content distribution based on a synthesis of desirable elements from the surveyed architectures. This analysis (based on Table 3.1) helped identify the need for an architecture that supports the whole generic model by offering efficient low-cost content distribution from trusted or untrusted infrastructure while supporting content: control, security, routing, billing, accounting, accountability, and ubiquitous caching. The ICN future Internet paradigm offers solutions to many of these challenges by enabling content to self-verify. However, it fails to provide an adequate accounting and accountability framework for the content distribution process. Moreover, many of these architectures make a virtue out of not providing it claiming to offer natural privacy to users.

We propose an ICN architecture extension for content accounting and accountability called the Savant framework, which we apply to the NDN architecture. Savant is designed based on the premise that content providers want visibility over the content being distributed primarily for control and analytics purposes. This goal was achieved by combining cryptographic techniques, PKI security processes, ICN principles and information flow processing mechanisms utilised in existing systems to develop a scalable, secure, near real-time accounting and accountability framework for an ICN packet-level content distribution architecture. This architecture naturally supports efficient content distribution while gathering non-repudiable near real-time information efficiently from NDN clients and NDN caches.

Our proof-of-concept implementation (shown in Figure 5.1), which was based on an NDN video content distribution session, demonstrated that accounting and accountability information can be gathered for an ICN packet-level architecture. Our experiments demonstrated that the overhead on the system is very small if several extensions and improvements are applied to the architecture for efficiency. Further experiments on NDNSim, an ns-3 module, demonstrated that it could easily scale to accommodate 10s of millions of NDN agents. Furthermore, we demonstrated that Savant can gather additional accounting metrics when required, which helps reduce the accounting information overhead while still maintaining the ability to investigate problems, detect and isolate faults, and so forth during the content distribution process.

As described, ICNs/NDN with Savant support, could eventually complement or replace today's CDN infrastructure with a scalable, trustworthy, reliable accounting and accountability framework for content distributed in trusted and untrusted environments meeting the diverse requirements of content providers, network operators and end-users.

6.1 Summary of Contributions

The main contributions of this thesis include:

- an analysis of the problem of efficient content distribution in existing and future Internet architectures and recognising that the success of a content distribution architecture relies on the level of content accounting and accountability it offers; the identification of usage patterns and basic requirements an accounting and accountability framework must satisfy, regardless of the infrastructure (trustworthy or untrustworthy) it is deployed or implemented on;
- a system, Savant, that satisfies these requirements for future Internet architectures such as NDN by synthesising components, designs and principles from multiple different systems, architectures and environments into an accounting, accountability and near real-time scalable aggregation framework supported by a set of protocols to ensure log commitment between communicating agents and that can help identify, isolate and correct faults;
- a demonstration of the feasibility of adding an accounting and accountability element to the NDN ICN architecture, and verifying its applicability for video content distribution;
- a lessons learned during development of an accounting and accountability framework for a ICN packet-level architecture, which should be useful to future researchers designing and implementing a similar framework.

6.2 Future Work

The Savant framework and the accounting and accountability information it gathers holds a lot of potential for future work. These elements present a number of research questions that we will discuss in the remaining paragraphs.

The Savant architecture presents an opportunity to develop new routing algorithms to support the content distribution process. This is based on the observation that Savant accumulates knowledge about the locations of multiple copies of content at client and cache infrastructure. Similar algorithms for request routing are already being utilised by CDNs such as Akamai when directing users to surrogate servers based on a set of metrics such as content location, latency to server, proximity to server and so forth [Nygren et al., 2010]. Additionally, Conviva's centralised C3 controller [Ganjam et al., 2015] uses optimization algorithms to direct users to the optimal CDN server (from a list of servers from multiple CDN providers) for content delivery to a client. These algorithms are based on a set of metrics similar to CDN request routing but also based on content provider policies and objectives. Savant could take this further, not only incorporating policies and metrics for the optimal CDN infrastructure, but also P2P client infrastructure, network operator caches, multicast environments, and so forth.

The Savant architecture has the potential to decouple a content provider's dependence on a handful of global CDN providers by providing a framework that guarantees the integrity of the accounting information produced. However, a monetary model based on reimbursement for different amounts of accounting and accountability information produced is required. This model should consider traditional CDN accounting metrics such as proximity to users, content latency, volume of content delivered, and so forth. However, it should also consider how much money the content distributor or network operator is saving in terms of core and backhaul network bandwidth saved; no inter-ISP, Tier 1 or Tier 2 network traffic; and so forth. Moreover, some models could also be considered around reimbursement for P2P content distribution.

There is scope for developing an FCAPS (fault, configuration, accounting, performance, security) model for network management in ICNs. Savant has already been designed with many FCAPS requirements in mind. However, we have been focused on accounting for the content distribution process and developing a model to establish the integrity of the accounting information produced on trusted and untrusted ICN infrastructure. Moreover, our framework is designed for distributed collection and aggregation of logs produced across millions of NDN agents. However, many of the baseline principles and components already exist in Savant for collecting information from local network infrastructure such as a methodology to publish and collect logs, a security model, an aggregation and alerting system, and so forth. Additionally, configuration instructions can be sent to network devices using authenticated interests commands, which can also be used to help identify, isolate and correct faults. Even the taxonomy we developed could be expanded to identify different statistical metrics or objects to collect from ICN network infrastructure.

There is scope to investigate what accounting information can be produced when utilising a service such as Savant in ICN architectures. Most of the information collected in our implementation is based on existing industry standard QoE accounting metrics for video distribution. However, in Chapter 5 we established that if we account for every chunk of content distributed by client and cache infrastructure, then we can easily calculate the bitrate based on chunk size. As a result, we do not need to gather uplink or downlink bitrate from the client or cache infrastructure. Using Savant in conjunction with different ICN architectures and applications will change accounting requirements, which should be explored further.

To our knowledge, the SPKI/SDSI [Clarke, 2001] security model has not been investigated to any great depth in ICNs. Moreover, it has not had any significant development's in over sixteen years. Consequently, there is scope to investigate different security and trust models related to local namespaces and key management using this model based around ICN content, content providers, users, infrastructure, routing, and so forth [Zhang et al., 2010].

NDN cache components need to be located inside ISP networks and NDN traffic needs to be directed to these elements by network operators. The advent of network function virtualisation (NFV) frameworks inside ISP networks offers virtualised environments where NDN caches can be easily deployed and extended [Mijumbi et al., 2016]. However, at what location will these elements be most effective e.g., core network, backhaul network or cell tower [Spagna et al., 2013]? Moreover, NDN cache components should not be confined to ISP networks. They can also become part of wireless access points, P2P infrastructure, automobiles, drones, and so forth. Detailed analysis of the benefits Savant can offer in these environments is required.

References

AbdAllah, E., Hassanein, H., & Zulkernine, M. (2015). A Survey of Security Attacks in Information-Centric Networking. *Communications Surveys Tutorials, IEEE*, *17*(3), 1441-1454. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7009958&isnumber=7214344 doi: 10.1109/COMST.2015.2392629

Adamson, B., Bormann, C., Handley, M., & Macker, J. (2009, November). *NACK-Oriented Reliable Multicast (NORM) Transport Protocol.* RFC 5740. IETF. Retrieved from http://tools.ietf.org/html/rfc5740

Adhikari, V. K., Jain, S., Chen, Y., & Zhang, Z.-L. (2012, March). Vivisecting YouTube: An active measurement study. In *Proceedings of the IEEE International Conference on Computer Communications* (p. 2521-2525). Orlando, FL, USA: IEEE. Retrieved from http://www-users.cs.umn.edu/~zhzhang/Papers/youtube-tech-report.pdf doi: 10.1109/INFCOM.2012.6195644

Aditya, P., Zhao, M., Lin, Y., Haeberlen, A., Druschel, P., Maggs, B., & Wishon, B. (2012, April). Reliable Client Accounting for P2P-infrastructure Hybrids. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (pp. 8–8). San Jose, CA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=2228298.2228309

Afanasyev, A., Mahadevan, P., Moiseenko, I., Uzun, E., & Zhang, L. (2013, May). Interest Flooding Attack and Countermeasures in Named Data Networking. In *IFIP Networking Conference, 2013* (p. 1-9). Brooklyn, NY, USA: IEEE. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6663516&isnumber=6663488

Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., & Ohlman, B. (2012). A Survey of Information-Centric Networking. *IEEE Communications Magazine*, *50*(7), 26-36. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231276&isnumber=6231266` doi: 10.1109/MCOM.2012.6231276

Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., ... Whittle, S. (2015). The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-scale, Unbounded, Out-of-order Data Processing. *Proc. VLDB Endow.*, *8*(12), 1792–1803. Retrieved from `http://dx.doi.org/10.14778/2824032.2824076` doi: 10.14778/2824032.2824076

Almeroth, K. C., & Ammar, M. H. (1996, Aug). Collecting and Modeling the Join/Leave Behavior of Multicast Group Members in the MBone. In *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing* (p. 209-216). Syracuse, NY, USA: IEEE. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=546190&isnumber=11475` doi: 10.1109/HPDC.1996.546190

Al-Shaer, E., & Tang, Y. (2004, April). MRMON: Remote Multicast Monitoring. In *Proceedings of the IEEE Network Operations and Management Symposium* (Vol. 1, p. 585-598 Vol.1). Seoul, Korea: IEEE. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1317746&isnumber=29204` doi: 10.1109/NOMS.2004.1317746

Anand, A., Dogar, F., Han, D., Li, B., Lim, H., Machado, M., ... Steenkiste, P. (2011, November). XIA: An Architecture for an Evolvable and Trustworthy Internet. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (pp. 2:1–2:6). Cambridge, MA, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2070562.2070564` doi: 10.1145/2070562.2070564

Androutsellis-Theotokis, S., & Spinellis, D. (2004). A Survey of Peer-to-peer Content Distribution Technologies. *ACM Computing Surveys*, *36*(4), 335–371. Retrieved from `http://doi.acm.org/10.1145/1041680.1041681` doi: 10.1145/1041680.1041681

Asaeda, H., Matsuzono, K., & Turletti, T. (2015). Contrace: A Tool for Measuring and Tracing Content-Centric Networks. *Communications Magazine, IEEE*, *53*(3), 182-188. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7060502&isnumber=7060469` doi: 10.1109/MCOM.2015.7060502

ATIS. (June 2011). *CDN Interconnection Use Case Specification and High Level Requirements, Alliance for Telecommunications Industry Solutions. Technical Report ATIS-0200003* (Vol. ATIS-0200003). Retrieved from `http://webstore.ansi.org/RecordDetail.aspx?sku=ATIS-0200003`

Balachandran, A., Sekar, V., Akella, A., Seshan, S., Stoica, I., & Zhang, H. (2012, October). A Quest for an Internet Video Quality-of-Experience Metric. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (pp. 97–102). Redmond, WA, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2390231.2390248` doi: 10.1145/2390231.2390248

Barish, G., & Obraczke, K. (2000, May). World Wide Web Caching: Trends and Techniques. *IEEE Communications Magazine*, *38*(5), 178-184. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=841844&isnumber=18201` doi: 10.1109/35.841844

Bitar, N., Niven-Jenkins, B., & Faucheur, F. L. (2012, September). *Content Distribution Network Interconnection (CDNI) Problem Statement.* RFC 6707. IETF. Retrieved from `http://tools.ietf.org/html/rfc6707`

Buchmann, J. A., Karatsiolis, E., & Wiesmaier, A. (2013). *Introduction to Public Key Infrastructures.* Heidelberg, Germany: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-40657-7

Burke, J., Gasti, P., Nathan, N., & Tsudik, G. (2012). Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control. *CoRR*, *abs/1208.1336*. Retrieved from `http://arxiv.org/pdf/1208.1336.pdf`

Burke, J., Gasti, P., Nathan, N., & Tsudik, G. (2013, April). Securing Instrumented Environments Over Content-Centric Networking: The Case of Lighting Control and NDN. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*

(p. 394-398). Turin, Italy: IEEE. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6970725&isnumber=6562841 doi: 10.1109/INFCOMW.2013.6970725

Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., & Singh, A. (2003, October). SplitStream: High-bandwidth Multicast in Cooperative Environments. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (pp. 298–313). Bolton Landing, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/945445.945474 doi: 10.1145/945445.945474

Castro, M., Druschel, P., Kermarrec, A.-M., & Rowstron, A. (2002). Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications*, *20*(8), 1489-1499. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1038579&isnumber=22260 doi: 10.1109/JSAC.2002.803069

Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., & Moon, S. (2007, October). I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement* (pp. 1–14). San Diego, CA, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1298306.1298309 doi: 10.1145/1298306.1298309

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... Gruber, R. E. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.*, *26*(2), 4:1–4:26. Retrieved from http://doi.acm.org/10.1145/1365815.1365816 doi: 10.1145/1365815.1365816

Chen, A., Moore, W. B., Xiao, H., Haeberlen, A., Phan, L. T. X., Sherr, M., & Zhou, W. (2014). Detecting Covert Timing Channels with Time-deterministic Replay. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (pp. 541–554). Broomfield, CO, USA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=2685048.2685091

Chen, F., Sitaraman, R. K., & Torres, M. (2015, August). End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (pp. 167–181). London, UK:

ACM. Retrieved from `http://doi.acm.org/10.1145/2785956.2787500` doi: 10.1145/2785956.2787500

Chu, Y.-h., Ganjam, A., Ng, T. S. E., Rao, S. G., Sripanidkulchai, K., Zhan, J., & Zhang, H. (2004, June). Early Experience with an Internet Broadcast System Based on Overlay Multicast. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (pp. 12–12). Boston, MA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1247415.1247427`

Chu, Y.-h., Rao, S. G., Seshan, S., & Zhang, H. (2002). A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications*, *20*(8), 1456-1471. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1038577&isnumber=22260` doi: 10.1109/JSAC.2002.803066

Cisco. (2015). *Cisco Visual Networking Index: Forecast and Methodology 2014—2019.* Retrieved Accessed: 2015-06-02, from `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf`

Clark, D. D., & Blumenthal, M. S. (2011). End-to-End Argument and Application Design: The Role of Trust. *Federal Communications Law Journal*, *63*(3). Retrieved from `http://www.repository.law.indiana.edu/fclj/vol63/iss2/3`

Clarke, D. E. (2001). *SPKI/SDSI HTTP Server/Certificate Chain Discovery in SPKI/SDSI.* MA, USA: Massachusetts Institute of Technology. Retrieved from `http://hdl.handle.net/1721.1/72800`

Clifton, B. (2012). *Advanced Web Metrics with Google Analytics*. NJ, USA: John Wiley and Sons Inc.

Cohen, B. (2003, June). Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems* (Vol. 6, pp. 68–72). Berkeley CA, USA. Retrieved from `http://www.bittorrent.org/bittorrentecon.pdf`

Cohen, J., Repantis, T., McDermott, S., Smith, S., & Wein, J. (2010, November). Keeping Track of 70,000+ Servers: The Akamai Query System. In *Proceedings of the 24th International Conference on Large Installation System Administration* (pp. 1–13). San Jose, CA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1924976.1924999

CoralCDN. (2011). *Coralcdn.* Retrieved 2014-03-12, from http://www.coralcdn.org

Cugola, G., & Margara, A. (2012). Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys*, *44*(3), 15:1–15:62. Retrieved from http://doi.acm.org/10.1145/2187671.2187677 doi: 10.1145/2187671.2187677

Dannewitz, C., Golic, J., Ohlman, B., & Ahlgren, B. (2010, March). Secure Naming for a Network of Information. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010* (p. 1-6). San Diego, CA, USA: IEEE. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5466661&isnumber=5466609 doi: 10.1109/INFCOMW.2010.5466661

Day, M., Cain, B., Tomlinson, G., & Rzewski, P. (2003, February). *A Model for Content Internetworking (CDI).* RFC 3466. IETF. Retrieved from http://tools.ietf.org/html/rfc3466

Deering, S. (1988, May). *Host Extensions for IP Multicasting.* RFC 1054. IETF. Retrieved from http://tools.ietf.org/html/rfc1054

Dellarocas, C. (2001, October). Analyzing the Economic Efficiency of eBay-like Online Reputation Reporting Mechanisms. In *Proceedings of the 3rd ACM Conference on Electronic Commerce* (pp. 171–179). Tampa, Florida, USA: ACM. Retrieved from http://doi.acm.org/10.1145/501158.501177 doi: 10.1145/501158.501177

Dobrian, F., Awan, A., Joseph, D., Ganjam, A., Zhan, J., Sekar, V., ... Zhang, H. (2013). Understanding the Impact of Video Quality on User Engagement. *Commun. ACM*, *56*(3), 91–99. Retrieved from http://doi.acm.org/10.1145/2428556.2428577 doi: 10.1145/2428556.2428577

Douceur, J. R. (2002). The Sybil Attack. In *Peer-to-Peer Systems* (Vol. 2429, p. 251-260). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/3-540-45748-8_24 doi: 10.1007/3-540-45748-8_24

Drscholl. (2001). *OpenNap: Open Source Napster Server.* Retrieved 2014-03-12, from `http://opennap.sourceforge.net`

Drscholl. (April 2000). *Napster Messages (open specification).* Retrieved 2014-03-12, from `http://opennap.sourceforge.net/napster.txt`

ESM. (2007). *End System Multicast: Streaming for the Masses.* Retrieved 2013-09-19, from `http://esm.cs.cmu.edu/`

ETSI. (2011). *Telecommunications and Internet Converged Services and Protocols for Advanced Networking (TISPAN); Content Delivery Network (CDN) Architecture Interconnection with TISPAN IPTV Architectures, Technical Report ETSI TS 182 019* (Vol. ETSI TS 182 019). Retrieved from `http://www.etsi.org/deliver/etsi_ts/182000_182099/182019/03.01.01_60/ts_182019v030101p.pdf`

ETSI. (2013). *Network Functions Virtualisation (NFV); Use Cases, Technical Report ETSI GS NFV 001 V1.1.1 (2013-10)* (Vol. NFV 001 V1.1.1). Retrieved from `http://docbox.etsi.org/ISG/NFV/Open/Published/gs_NFV001v010101p-UseCases.pdf`

Fayazbakhsh, S. K., Lin, Y., Tootoonchian, A., Ghodsi, A., Koponen, T., Maggs, B., ... Shenker, S. (2013, August). Less Pain, Most of the Gain: Incrementally Deployable ICN. In *Proceedings of the ACM SIGCOMM Conference* (pp. 147–158). Hong Kong, China: ACM. Retrieved from `http://doi.acm.org/10.1145/2486001.2486023` doi: 10.1145/2486001.2486023

Frank, B., Poese, I., Lin, Y., Smaragdakis, G., Feldmann, A., Maggs, B., ... Weber, R. (2013). Pushing CDN-ISP Collaboration to the Limit. *SIGCOMM Computer Communication Review*, *43*(3), 34–44. Retrieved from `http://doi.acm.org/10.1145/2500098.2500103` doi: 10.1145/2500098.2500103

Freedman, M. J. (2010, April). Experiences with CoralCDN: A Five-year Operational View. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (pp. 7–7). San Jose, CA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1855711.1855718`

Freedman, M. J., Freudenthal, E., & Mazières, D. (2004, March). Democratizing Content Publication with Coral. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation* (Vol. 1, pp. 18–18). San Francisco, CA, USA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1251175.1251193`

Freedman, M. J., & Mazières, D. (2003). Sloppy Hashing and Self-Organizing Clusters. In *Peer-to-Peer Systems II* (Vol. 2735, p. 45-55). Springer Berlin Heidelberg. Retrieved from `http://dx.doi.org/10.1007/978-3-540-45172-3_4` doi: 10.1007/978-3-540-45172-3_4

Ganjam, A., Jiang, J., Liu, X., Sekar, V., Siddiqi, F., Stoica, I., ... Zhang, H. (2015, May). C3: Internet-scale Control Plane for Video Quality Optimization. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (pp. 131–144). Oakland, CA, USA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=2789770.2789780`

Ghali, C., Tsudik, G., Wood, C., & Yeh, E. (2015). Practical Accounting in Content-Centric Networking (extended version). *CoRR*, *abs/1510.01852*. Retrieved from `http://arxiv.org/abs/1510.01852`

Ghodsi, A., Koponen, T., Rajahalme, J., Sarolahti, P., & Shenker, S. (2011, August). Naming in Content-oriented Architectures. In *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking* (pp. 1–6). Toronto, ON, Canada: ACM. Retrieved from `http://doi.acm.org/10.1145/2018584.2018586` doi: 10.1145/2018584.2018586

Greenberg, A., Hamilton, J., Maltz, D. A., & Patel, P. (2008). The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Computer Communication Review*, *39*(1), 68–73. Retrieved from `http://doi.acm.org/10.1145/1496091.1496103` doi: 10.1145/1496091.1496103

Haeberlen, A. (2010). A Case for the Accountable Cloud. *Operating Systems Review*, *44*(2), 52–57. Retrieved from `http://doi.acm.org/10.1145/1773912.1773926` doi: 10.1145/1773912.1773926

Haeberlen, A., Aditya, P., Rodrigues, R., & Druschel, P. (2010, October). Accountable Virtual Machines. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (pp. 1–16). Vancouver, BC, Canada: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1924943.1924952

Haeberlen, A., Kouznetsov, P., & Druschel, P. (2006, November). The Case for Byzantine Fault Detection. In *Proceedings of the 2nd Conference on Hot Topics in System Dependability* (Vol. 2, pp. 5–5). Seattle, WA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1251014.1251019

Haeberlen, A., Kouznetsov, P., & Druschel, P. (2007). PeerReview: Practical Accountability for Distributed Systems. *Operating Systems Review*, *41*(6), 175–188. Retrieved from http://doi.acm.org/10.1145/1323293.1294279 doi: 10.1145/1323293.1294279

Han, D., Anand, A., Dogar, F., Li, B., Lim, H., Machado, M., … Steenkiste, P. (2012, April). XIA: Efficient Support for Evolvable Internetworking. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation* (pp. 309–322). San Jose, CA: USENIX. Retrieved from https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/han_dongsu_xia

Hei, X., Liang, C., Liang, J., Liu, Y., & Ross, K. W. (2007). A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, *9*(8), 1672-1687. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4378423&isnumber=4378421 doi: 10.1109/TMM.2007.907451

Huang, C., Wang, A., Li, J., & Ross, K. W. (2008, May). Understanding Hybrid CDN-P2P: Why Limelight Needs Its Own Red Swoosh. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (pp. 75–80). Braunschweig, Germany: ACM. Retrieved from http://doi.acm.org/10.1145/1496046.1496064 doi: 10.1145/1496046.1496064

Huang, Y., Fu, T. Z., Chiu, D.-M., Lui, J. C., & Huang, C. (2008). Challenges, Design and Analysis of a Large-scale P2P-vod System. *SIGCOMM Computer Communi-*

cation Review, *38*(4), 375–388. Retrieved from `http://doi.acm.org/10.1145/1402946.1403001` doi: 10.1145/1402946.1403001

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., & Braynard, R. L. (2009, December). Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies* (pp. 1–12). Rome, Italy: ACM. Retrieved from `http://doi.acm.org/10.1145/1658939.1658941` doi: 10.1145/1658939.1658941

Jain, R., & Paul, S. (2013). Network Virtualization and Software Defined Networking for Cloud Computing: A Survey. *IEEE Communications Magazine*, *51*(11), 24-31. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6658648&isnumber=6658638` doi: 10.1109/MCOM.2013.6658648

Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., & Nikander, P. (2009, August). LIPSIN: Line Speed Publish/Subscribe Inter-networking. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (pp. 195–206). Barcelona, Spain: ACM. Retrieved from `http://doi.acm.org/10.1145/1592568.1592592` doi: 10.1145/1592568.1592592

Kamvar, S. D., Schlosser, M. T., & Garcia-Molina, H. (2003, May). The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web* (pp. 640–651). Budapest, Hungary: ACM. Retrieved from `http://doi.acm.org/10.1145/775152.775242` doi: 10.1145/775152.775242

Katsaros, K., Xylomenos, G., & Polyzos, G. C. (2011). MultiCache: An Overlay Architecture for Information-centric Networking. *Computer Networks*, *55*(4), 936–947. Retrieved from `http://dx.doi.org/10.1016/j.comnet.2010.12.012` doi: 10.1016/j.comnet.2010.12.012

Ko, R. K., Jagadpramana, P., Mowbray, M., Pearson, S., Kirchberg, M., Liang, Q., & Lee, B. S. (2011, July). TrustCloud: A Framework for Accountability and Trust in Cloud Computing. In *Proceedings of the IEEE World Congress on Services (SERVICES)* (p. 584-588). Washington, DC, USA: IEEE. Retrieved from `http://ieeexplore.ieee.org/`

stamp/stamp.jsp?tp=&arnumber=6012795&isnumber=6012651 doi: 10.1109/SERVICES.2011.91

Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., & Stoica, I. (2007, August). A Data-oriented (and Beyond) Network Architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (pp. 181–192). Kyoto, Japan: ACM. Retrieved from http://doi.acm.org/10.1145/1282380.1282402 doi: 10.1145/1282380.1282402

Kulinski, D., & Burke, J. (2012). NDNVideo: random-access live and pre-recorded streaming using ndn. *University of California, Los Angeles, Tech. Rep. NDN–0007*, *0*(0), 0-17. Retrieved from http://www.named-data.net/techreport/TR007-streaming.pdf

Liu, J., Rao, S. G., Li, B., & Zhang, H. (2008). Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, *96*(1), 11-24. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4399973&isnumber=4404137 doi: 10.1109/JPROC.2007.909921

Liu, Y., Guo, L., Li, F., & Chen, S. (2009, June). A Case Study of Traffic Locality in Internet P2P Live Streaming Systems. In *Proceedings of 29th IEEE International Conference on Distributed Computing Systems* (p. 423-432). Montreal, QC, Canada: IEEE. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5158452&isnumber=5158393 doi: 10.1109/ICDCS.2009.50

Locher, T., Moor, P., Schmid, S., & Wattenhofer, R. (2006, November). Free Riding in BitTorrent is Cheap. In *Proceedings of the Fifth Workshop on Hot Topics in Networks (HotNets-V).* Irvine, CA, US: HotNets. Retrieved from http://distcomp.ethz.ch/publications/hotnets06.pdf

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys Tutorials*, *7*(2), 72-93. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1610546&isnumber=33817 doi: 10.1109/COMST.2005.1610546

Magharei, N., Rejaie, R., & Guo, Y. (2007, May). Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *Proceedings of the IEEE International Conference on Computer Communications* (p. 1424-1432). IEEE. Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4215750&isnumber=4215582` doi: 10.1109/INFCOM.2007.168

Mahadevan, P., Uzun, E., Sevilla, S., & Garcia-Luna-Aceves, J. (2014, September). CCN-KRS: A Key Resolution Service for CCN. In *Proceedings of the 1st International Conference on Information-centric Networking* (pp. 97–106). Paris, France: ACM. Retrieved from `http://doi.acm.org/10.1145/2660129.2660154` doi: 10.1145/2660129.2660154

Makofske, D., & Almeroth, K. (1999, June). MHealth: A Real-Time Multicast Tree Visualization and Monitoring Tool. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video.* Basking Ridge, NJ, USA: ACM. Retrieved from `http://www.nossdav.org/1999/papers/53-1441030863.pdf`

Mathew, V., Sitaraman, R. K., & Shenoy, P. (2014). Energy-efficient Content Delivery Networks using Cluster Shutdown. *Sustainable Computing: Informatics and Systems.* Retrieved from `http://www.sciencedirect.com/science/article/pii/S221053791400033X` doi: http://dx.doi.org/10.1016/j.suscom.2014.05.004

Mauri, J. L., Fuster, G., Santos, J. D., & Domingo, M. E. (2004). Analysis and Characterization of Peer-To-Peer Filesharing Networks. *World Scientific and Engineering Academy and Society Transactions on Systems*(7), 2574–2579. Retrieved from `http://personales.upv.es/jlloret/pdf/icosmo2004.pdf`

Maymounkov, P., & Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems* (Vol. 2429, p. 53-65). Springer Berlin Heidelberg. Retrieved from `http://dx.doi.org/10.1007/3-540-45748-8_5` doi: 10.1007/3-540-45748-8_5

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). Dremel: Interactive Analysis of Web-scale Datasets. *Proc. VLDB Endow.*, *3*(1-2), 330–339. Retrieved from http://dx.doi.org/10.14778/1920841.1920886 doi: 10.14778/1920841.1920886

Michalakis, N., Soulé, R., & Grimm, R. (2007, April). Ensuring Content Integrity for Untrusted Peer-to-Peer Content Distribution Networks. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (pp. 11–11). Cambridge, MA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1973430.1973441

Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., De Turck, F., & Boutaba, R. (2016). Network Function Virtualization: State-of-the-Art and Research Challenges. *Communications Surveys Tutorials, IEEE*, *18*(1), 236-262. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7243304&isnumber=7393921 doi: 10.1109/COMST.2015.2477041

Mockapetris, P. (1987, September). *Domain Names - Concepts and Facilities.* RFC 1034. IETF. Retrieved from https://www.ietf.org/rfc/rfc1034.txt

Mockapetris, P., & Dunlap, K. J. (1988, July). Development of the Domain Name System. In *Symposium Proceedings on Communications Architectures and Protocols* (pp. 123–133). Stanford, CA, USA: ACM. Retrieved from http://doi.acm.org/10.1145/52324.52338 doi: 10.1145/52324.52338

Nandi, A., Ganjam, A., Druschel, P., Ng, T. S. E., Stoica, I., Zhang, H., & Bhattacharjee, B. (2007, April). SAAR: A Shared Control Plane for Overlay Multicast. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (pp. 5–5). Cambridge, MA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1973430.1973435

NDN. (2015). *Named data networking.* Retrieved 2015-05-20, from http://named-data.net/

Nygren, E., Sitaraman, R. K., & Sun, J. (2010). The Akamai Network: A Platform for High-performance Internet Applications. *Operating Systems Review*, *44*(3), 2–19. Retrieved from http://doi.acm.org/10.1145/1842733.1842736 doi: 10.1145/1842733.1842736

Ó Coileáin, D., & O'Mahony, D. (2014a, Aug). Savant: A Framework for Supporting Content Accountability in Information Centric Networks. In *Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), 2014 10th International Conference on* (p. 188-190). Rhodes, Greece: IEEE. doi: 10.1109/QSHINE.2014.6928686

Ó Coileáin, D., & O'Mahony, D. (2014b, April). SAVANT: Aggregated Feedback and Accountability Framework for Named Data Networking. In *Proceedings of the 17th Royal Irish Academy Research Colloquium on Communications and Radio Science into the 21st Century*. Dublin, Ireland.

Ó Coileáin, D., & O'Mahony, D. (2014c, September). SAVANT: Aggregated Feedback and Accountability Framework for Named Data Networking. In *Proceedings of the 1st International Conference on Information-Centric Networking* (pp. 187–188). Paris, France: ACM. Retrieved from `http://doi.acm.org/10.1145/2660129.2660165` doi: 10.1145/2660129.2660165

Pallis, G., & Vakali, A. (2006). Insight and Perspectives for Content Delivery Networks. *Communications of the ACM*, *49*(1), 101–106. Retrieved from `http://doi.acm.org/10.1145/1107458.1107462` doi: 10.1145/1107458.1107462

PARC. (2015). Project ccnx. Retrieved 2015-11-23, from `http://www.ccnx.org`

Passarella, A. (2012). Review: A Survey on Content-Centric Technologies for the Current Internet: CDN and P2P Solutions. *Computer Communications*, *35*(1), 1–32. Retrieved from `http://dx.doi.org/10.1016/j.comcom.2011.10.005` doi: 10.1016/j.comcom.2011.10.005

Pathan, M., & Buyya, R. (2008). A Taxonomy of CDNs. In *Content Delivery Networks* (Vol. 9, p. 33-77). Springer Berlin Heidelberg. Retrieved from `http://dx.doi.org/10.1007/978-3-540-77887-5_2` doi: 10.1007/978-3-540-77887-5_2

PeerApp. (2014a). *Monetizing OTT Content and Service Delivery.* Retrieved 2014-09-15, from `http://www.peerapp.com/Solutions/cse.aspx`

PeerApp. (2014b). *UltraBandTM 6000 Series Product Overview.* Retrieved 2014-03-12, from `http://www.peerapp.com/products/UltraBand.aspx`

Peterson, L., Davie, B., & Van Brandenburg, R. (2014, August). *Framework for Content Distribution Network Interconnection (CDNI).* RFC 7336. IETF. Retrieved from `http://tools.ietf.org/html/rfc7336`

Peterson, R. S., & Sirer, E. G. (2009, April). Antfarm: Efficient Content Distribution with Managed Swarms. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation* (pp. 107–122). Boston, MA, USA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1558977.1558985`

Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., & Venkataramani, A. (2007, April). Do Incentives Build Robustness in BitTorrent. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation* (pp. 1–1). Cambridge, MA, USA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1973430.1973431`

Psaras, I., Katsaros, K. V., Saino, L., & Pavlou, G. (2015). LIRA: A Location Independent Routing Layer based on Source-Provided Ephemeral Names. *CoRR*, *abs/1509.05589*. Retrieved from `http://arxiv.org/abs/1509.05589`

Repantis, T., Cohen, J., Smith, S., & Wein, J. (2010). Scaling a Monitoring Infrastructure for the Akamai Network. *Operating Systems Review*, *44*(3), 20–26. Retrieved from `http://doi.acm.org/10.1145/1842733.1842737` doi: 10.1145/1842733.1842737

Rodrigues, R., & Druschel, P. (2010). Peer-to-peer Systems. *Communications of the ACM*, *53*(10), 72–82. Retrieved from `http://doi.acm.org/10.1145/1831407.1831427` doi: 10.1145/1831407.1831427

Salapura, V., Beaty, K. A., Bivens, A., Kim, M., & Li, M. (2015, May). Towards Building an Analytics Platform in the Cloud. In *Proceedings of the 12th ACM International Conference on Computing Frontiers* (pp. 49:1–49:8). Ischia, Italy: ACM. Retrieved from `http://doi.acm.org.elib.tcd.ie/10.1145/2742854.2747279` doi: 10.1145/2742854.2747279

Salsano, S., Detti, A., Cancellieri, M., Pomposini, M., & Blefari-Melazzi, N. (2012, August). Transport-layer Issues in Information Centric Networks. In *Proceedings of* *the Second Edition of the ICN Workshop on Information-centric Networking* (pp. 19–24). Helsinki, Finland: ACM. Retrieved from `http://doi.acm.org/10.1145/2342488.2342493` doi: 10.1145/2342488.2342493

Schulzrinne, H., Casner, S., Frederick, R., & Jacobson, V. (1996, January). *RTP: A Transport Protocol for Real-Time Applications.* RFC 1889. IETF. Retrieved from `http://tools.ietf.org/html/rfc1889`

Sharma, A., Venkataramani, A., & Rocha, A. A. (2014, January). Pros & Cons of Model-Based Bandwidth Control for Client-Assisted Content Delivery. In *Proceedings of the Sixth International Conference on Communication Systems and Networks* (p. 1-8). Retrieved from `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6734893&isnumber=6734849` doi: 10.1109/COMSNETS.2014.6734893

Sharma, A., Venkataramani, A., & Sitaraman, R. K. (2013, June). Distributing Content Simplifies ISP Traffic Engineering. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems* (pp. 229–242). Pittsburgh, PA, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2465529.2465764` doi: 10.1145/2465529.2465764

Sirivianos, M., Park, J. H., Yang, X., & Jarecki, S. (2007). Dandelion: Cooperative Content Distribution with Robust Incentives. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference* (pp. 12:1–12:14). Santa Clara, CA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1364385.1364397`

Skytide. (Decemeber 2012). *Combining cdn and transparent caching into a dynamic duo.* Retrieved 2014-03-12, from `http://www.skytide.com/content/resources/PDF/Combining_CDN_and_Transparent_Caching_into_a_Dynamic_Duo.pdf`

Spagna, S., Liebsch, M., Baldessari, R., Niccolini, S., Schmid, S., Garroppo, R., ... Awano, J. (2013). Design Principles of an Operator-Owned Highly Distributed Content Delivery Network. *Communications Magazine, IEEE*, *51*(4), 132-140. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6495772&isnumber=6495745 doi: 10.1109/MCOM.2013.6495772

Spring, N., Mahajan, R., & Wetherall, D. (2002, August). Measuring ISP Topologies with Rocketfuel. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (pp. 133–145). Pittsburgh, PA, USA: ACM. Retrieved from http://doi.acm.org/10.1145/633025.633039 doi: 10.1145/633025.633039

Stallings, W. (1998, Mar). SNMP and SNMPv2: The Infrastructure for Network Management. *Communications Magazine, IEEE*, *36*(3), 37-43. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=663326&isnumber=14500 doi: 10.1109/35.663326

Trossen, D. (2011). *PURSUIT Publish Subscribe Internet Technology FP7-INFSO-ICT-257217 Codifying Evolving Tussles For Tomorrow's Internet.* Retrieved from http://fp7pursuit.ipower.com/PursuitWeb/wp-content/uploads/2011/10/TR11-0001.pdf

Trossen, D., Tuononen, J., Xylomenos, G., Sarela, M., Zahemszky, A., Nikander, P., & Rinta-aho, T. (2008). *From Design for Tussle to Tussle Networking: PSIRP Vision and Use Cases, Technical Report TR08-0001* (Vol. PSIRP-TR08-0001). Retrieved 2014-03-12, from http://www.psirp.org/files/Deliverables/PSIRP-TR08-0001_Vision.pdf

Venkataramani, A., Kurose, J. F., Raychaudhuri, D., Nagaraja, K., Mao, M., & Banerjee, S. (2014). MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture. *SIGCOMM Computer Communication Review*, *44*(3), 74–80. Retrieved from http://doi.acm.org/10.1145/2656877.2656888 doi: 10.1145/2656877.2656888

Wang, J. (1999). A Survey of Web Caching Schemes for the Internet. *SIGCOMM Computer Communication Review*, *29*(5), 36–46. Retrieved from http://doi.acm.org/10.1145/505696.505701 doi: 10.1145/505696.505701

Whitman, B., & Lawrence, S. (2002, September). Inferring Descriptions and Similarity for Music from Community Metadata. In *Proceedings of the 2002 International Com-*

puter Music Conference (pp. 591–598). Gothenburg, Sweden. Retrieved from http://alumni.media.mit.edu/~bwhitman/whitman02inferring.pdf

Woo, S., Jeong, E., Park, S., Lee, J., Ihm, S., & Park, K. (2013, June). Comparison of Caching Strategies in Modern Cellular Backhaul Networks. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (pp. 319–332). Taipei, Taiwan: ACM. Retrieved from http://doi.acm.org/10.1145/2462456.2464442 doi: 10.1145/2462456.2464442

Xu, D., Kulkarni, S., Rosenberg, C., & Chai, H.-K. (2006). Analysis of a CDN-P2P Hybrid Architecture for Cost-Effective Streaming Media Distribution. *Multimedia Systems*, *11*(4), 383-399. Retrieved from http://dx.doi.org/10.1007/s00530-006-0015-3 doi: 10.1007/s00530-006-0015-3

Xylomenos, G., Vasilakos, X., Tsilopoulos, C., Siris, V. A., & Polyzos, G. C. (2012). Caching and Mobility Support in a Publish-Subscribe Internet Architecture. *IEEE Communications Magazine*, *50*(7), 52-58. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231279&isnumber=6231266 doi: 10.1109/MCOM.2012.6231279

Xylomenos, G., Ververidis, C. N., Siris, V. A., Fotiou, N., Tsilopoulos, C., Vasilakos, X., ... Polyzos, G. C. (2014). A Survey of Information-Centric Networking Research. *IEEE Communications Surveys Tutorials*, *16*(2), 1024-1049. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6563278&isnumber=6811383 doi: 10.1109/SURV.2013.070813.00063

Yalagandula, P., & Dahlin, M. (2004, August). A Scalable Distributed Information Management System. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (pp. 379–390). Portland, OR, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1015467.1015509 doi: 10.1145/1015467.1015509

Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., ... Li, B. (2009, October). Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky. In *Proceedings of the 17th ACM International Conference on Multimedia* (pp.

25–34). Beijing, China: ACM. Retrieved from `http://doi.acm.org/10.1145/1631272.1631279` doi: 10.1145/1631272.1631279

Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., … Li, B. (2010). LiveSky: Enhancing CDN with P2P. *ACM Trans. Multimedia Comput. Commun. Appl.*, *6*(3), 16:1–16:19. Retrieved from `http://doi.acm.org/10.1145/1823746.1823750` doi: 10.1145/1823746.1823750

Yumerefendi, A. R., & Chase, J. S. (2005, June). The Role of Accountability in Dependable Distributed Systems. In *Proceedings of the First Conference on Hot Topics in System Dependability* (pp. 3–3). Yokohama, Japan: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=1973400.1973403`

Zaharia, M., Das, T., Li, H., Shenker, S., & Stoica, I. (2012, June). Discretized Streams: An Efficient and Fault-tolerant Model for Stream Processing on Large Clusters. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing* (pp. 10–10). Boston, MA: USENIX Association. Retrieved from `http://dl.acm.org/citation.cfm?id=2342763.2342773`

Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., claffy, k., Crowley, P., … Zhang, B. (2014). Named Data Networking. *SIGCOMM Comput. Commun. Rev.*, *44*(3), 66–73. Retrieved from `http://doi.acm.org/10.1145/2656877.2656887` doi: 10.1145/2656877.2656887

Zhang, L., Estrin, D., Burke, J., Jacobson, V., Thornton, J., Smetters, D. K., … Yeh, E. (2010, October). *Named Data Networking (NDN) Project NDN-0001* (Tech. Rep.). Retrieved from `http://www.parc.com/content/attachments/named-data-networking.pdf`

Zhao, M., Aditya, P., Chen, A., Lin, Y., Haeberlen, A., Druschel, P., … Ponec, M. (2013, October). Peer-assisted Content Distribution in Akamai Netsession. In *Proceedings of the Internet Measurement Conference* (pp. 31–42). Barcelona, Spain: ACM. Retrieved from `http://doi.acm.org/10.1145/2504730.2504752` doi: 10.1145/2504730.2504752