Modeling and analysis of techniques to increase robustness in distributed systems

Riccardo Loti

Modeling and analysis of techniques to increase robustness in distributed systems.

Riccardo Loti

Università degli Studi di Torino, Dipartimento di Informatica

Université de Nice Sophia Antipolis, c/o INRIA Sophia Antipolis Méditerranée

Thesis submitted for the degree of

Doctor of Philosophy

(26 May 2014)

Advisor UNITO:
Prof. Rossano Gaeta,
Associate Professor
Università degli Studi di Torino

Advisor UNS:
Dr. Luigi Liquori,
Research Director
INRIA Sophia Antipolis Méditerranée

Thesis reviewers:

Prof. John C.S. Lui,	*CUHK*
Prof. Giovanni Chiola,	*UNIGE*

Defense jury:

Prof. Giovanni Chiola,	*UNIGE*
Prof. Rossano Gaeta,	*UNITO*
Prof. Hung Nguyen Chan,	*VIELINA*
Prof. Marko Vukolić,	*EURECOM*
Dr. Luigi Liquori,	*INRIA*
Dr. Roberto Roverso,	*PEERIALISM AB*

Abstract

The original "selling point" for decentralized networks has been file exchange, using the decentralized approach to diffuse the bandwidth cost between all the participating nodes, augmenting the robustness by avoiding single point of failures and control by sharing the responsibility between all nodes.
While the most decentralized approaches are very efficient in terms of resilience, they are, by the same reason, more performance limited and harder to analyze.
This analysis is usually the prerogative of simulation tools, a quite inefficient way to analyze the possibility space. We thus developed and present here a mathematical model for network interconnection, enabling the study and exploration of equilibriums and, by virtue of the abstraction of the model, perfectly applicable to any interconnection of networks, be them communication networks, social networks, or physical distribution networks.
We also focused on decentralized networks, called MANETs, where communication between mobile nodes is purely ad-hoc based, exploiting rateless coding to minimize data loss due to transmission unreliability, and detecting malicious nodes sending corrupted packets, a hard to detect and prevent problem in a strongly distributed environments, using SIEVE, a custom developed algorithm.

Abstract

Le point de départ pour les systèmes decentralisées a eté l'échange des fichiers, en utilisant cet approche i) pour distribuer la bande passante entre tous les nodes concernés et ii) pour augmenter la robustesse en eliminant autant que possible les points individuels de défaillance et de contrôle et iii) en partageant également les responsabilités entre les nodes.

Si les approches le plus décentralisés sont très efficaces en termes de résilience aux pannes, pour la même raison, les performances sont limités et difficiles à analyser quand on observe plusieurs réseaux interconnectés entre eux, configurations qui peuvent être analysés à travers des outils de simulation, souvent peu efficaces dans l'analyse de l'espace de possibilités. Dans cette thèse on a développé un modèle matematique pour la modélisation de l'interconnexion des réseaux en permettant l'étude et l'exploration d'équilibres qui - grace à l'abstraction du modèle - peuvent s'appliquer à l'interconnexion des réseaux de communications, réseaux de distribution de marchandise ou réseaux de distribution d'eau.

La thèse se focalise aussi sur les réseaux décentralisés MANET, où la communication entre noeuds mobiles est purement « ad-hoc » (ex: deux voitures communiquant entre eux quand ils sont proches) en utilisant i) des « rateless coding » pour augmenter la robustesse et minimiser la perte ou la corruption de données causées par la non fiabilité du moyen de transmission et ii) des algorithmes de « pollution detection », par exemple de detection de noeuds malveillants ou de paquets corrompus, cette detection et prévention étant très difficile dans des environnements fortement distribués.

Abstract

L'iniziale "trampolino di lancio" per le reti decentralizzate è stato la distribuzione di file, un approccio decentralizzato per distribuire i costi di banda tra tutti i nodi partecipanti, aumentando la robustezza evitando singoli punti di fallimento e controllo, dividendo la responsabilità tra tutti i nodi. Nonostante gli approcci decentralizzati siano molto efficienti in termini di robustezza, per le medesime ragioni, sono più ostacolati nelle prestazioni e di più difficile analisi.
Questo tipo di analisi è solitamente prerogativa di strumenti simulativi, piuttosto inefficienti per analizzare lo spazio delle possibilità. Abbiamo quindi sviluppato un modello matematico per l'interconnessione di reti, permettente lo studio e l'esplorazione di equilibri e, in virtù dell'astrazione del modello, perfettamente applicabile a qualsiasi interconnessione di reti, siano esse di comunicazione, sociali o, di distribuzione fisica.
Ci siamo inoltre focalizzati su reti decentralizzate, chiamate MANET, dove la comunicazione tra nodi mobili è puramente ad hoc, sfruttando il "rateless coding" per minimizzare la perdità di dati a causa dell'inaffidabilità della trasmissione, e rilevando nodi malevoli invianti pacchetti corrotti, un problema difficile da identificare e prevenire in ambienti fortemente distribuiti, tramite SIEVE, un algoritmo appositamente sviluppato.

Contents

List of Figures

Part I

Overview

Chapter 1

Introduction

Chapter abstract:

In this Chapter we introduce the works this Thesis is composed of and the needs the proposed solutions arise from.

Each topic is connected to each other in an overall effort to increase robustness and availability in distributed networks and such underlying "common thread" is analyzed in the final Section of this Chapter, as a way to see the whole Thesis homogeneously.

Continuing Part I, Chapter 2 briefly introduces the state of the art for each topic, Chapter 3 will give a perspective of each work contribution to each topic. In Part II, Chapters 4 to 6 more deeply analyze each topic, offering a technical analysis of the contribution and results obtained, and in Part III, Chapter 7 summarizes the conclusions obtained from the results and future evolutions.

Finally, Appendix A acts as reference for the tools used in various topics.

1.1 Opportunities and challenges

In computer usage evolution, and its ecosystem in modern society, a particular place is occupied by communication networks. Especially in the last 5 years, with the development of smaller and more powerful devices, there has been a substantial push of the interconnection and mobility aspect of those devices.

New use cases migrate from the traditional single node concept with an op-

tional connection to the outside world, to a cloud of nodes, each one part of a more complex system, usually distributed, which see each participant only as a functional part of the whole, making the interconnection between them a fundamental part of the system.

Such decentralization of the concept of a "computer" opens the road to a whole range of new applications, usually impossible to realize before using a non-distributed approach. A list of a few common examples could be:

- a P2P file exchange, where the bandwidth resources load is distributed and scales with the whole network meanwhile giving the ability to avoid censorship and publishing delays of traditional channels;
- sensor networks, where every node, usually a low-power nano-sensor, exchanges data with others resulting in an exhaustive and robust autonomous coverage of an area;
- backup and data storage, for a totally distributed, set of files and data that can be preserved from physical harm and remotely accessed from every authorized device, everywhere on the globe;
- computational power aggregation, in which research or business institutions can distribute their computational workload among peers in a network;
- smart-grid / smart-city approach, where services and/or physical devices are made to talk with each other exchanging information and coordinating independently, offering a digital representation and access of the physical world;
- ad-hoc opportunistic wireless networking, like Delay Tolerant Networks (DTNs) and mesh networks, where interconnection between usually mobile nodes is made on the fly.

As a continual push to decentralization there is the perennial development of cheaper and more powerful platforms, moving the focus from the hardware, to the service itself, enabled by the software and this time with the hardware as just a medium to enable its usage.

This interaction between computers rather than on the computer itself introduces a series of problems, some of which were of minor importance before, while other are completely new, for example:

- moving externally something that is traditionally considered internal to the computer, such as processing, storage, and retrieval of data, forces a particular attention on the performance aspect, where delays and latencies tend to grow by orders of magnitude, and are hardly estimable in advance, needing robust and adaptive protocols;
- the protocol having to manage a bigger probability that data will get lost or corrupted during the transit meanwhile having only minor control over the process, to the point of not being able to directly act even if negative behaviors are known;
- the necessity of data security mechanisms, where the presence in the system of various actor, seldom trusted or known, makes imperative to implement access limits to transiting data and related control mechanisms, this way guaranteeing privacy and correctness.

In the scope of augmenting the performance, as well as the efficiency and robustness of networks, this Thesis adopts a holistic approach, offering a set of different techniques to be optionally applied both in the *planning phase* of a network design and in the *utilization phase* of such network. Such techniques are applicable or related to a multitude of network topologies and structures, common denominator between them all the decentralized approach used.

1.2 Random graph modeling of overlay networks interconnection

First we describe a random graphs based mathematical model we developed for analyzing network interconnections, where independent networks are connected together by bridge nodes, belonging to two or more networks.

While well known mathematical models exist for modeling individual networks [1], approaches modeling independent networks interconnected through shared nodes are few, and focuses on a hierarchical approach between the networks [2], or proposes a merging into a single equivalent network.

Both approaches are limiting, in the first case networks are considered not on the same level to each others, and in the second case the networks are either very similar to start with, or deeply modified by a merging like operation on all the networks.

The motivations for such model are to be searched in the computational and time costs associated with validating a specific configurations using simulation tools or limited scale prototyping, while variance in the building parameters can greatly influence performances and behaviors and as such justifies optimization analysis beforehand.

In interconnecting various networks together we are faced with a large number of combinations of parameters, such as the probability distributions describing the number of neighbors a node has, the number of nodes in each network belonging to more than one network, and so on. We focus only on a flooding routing strategy, common to many unstructured approaches (e.g.: P2P networks), so we add as parameters, the probability a node will forward a message (a query usually in communication networks).

Such large number of permutations easily renders any simulative approach quickly limited in its scope or completely inefficient at treating the optimization problem extensively enough, especially considering that many interesting behaviors emerge only when a considerable amount of nodes is present (in the range of 100 000 to 1 000 000 in our simulations), raising exponentially the resource consumption for any simulator or deployment solution.

Essentially, in the simulative approach, each node and each message, or at least a simplified model of them, must be kept in memory, and many parallel virtual events are coalesced in fewer processes leading to very intense use of computational resources like memory and computational power.

In our original testings with the the OmNET++ [3] simulator each set of pa-

rameters run, on a quad core Intel Xeon workstation with 16GB of RAM, took about two real-time days for 5 simulated networks of 100 000 nodes each, while a heavily simplified and parallelized custom simulator written in Erlang [4], optimized only to treat message passing and no transmissions details or contents, took, on the same machine, a few hours for each run, both software completely using the machine and using up to 8GB or RAM. Also this was only for a single run, while to have statistically significant data, at least a few runs for each parameters set have to be executed and averaged, linearly multiplying the time needed for the number of run repetition, then multiplying again for each set of parameters in the selected permutations, easily leading to weeks and weeks of continuos computing time for just exploring a small subset of all the possible permutations, all with the statistical relevance offered by the number of repetition for each run allowed.

Clearly this is not the best way to proceed, to this reason we developed a mathematical model representing the probability a node in a given network will find the resource he wants (meaning his query will reach a node owning the resource in a given number of jumps) and the average number of messages sent, given the number of networks, the probability distribution for neighbor nodes and for bridge nodes and the forwarding probability for the routing.

It may be noted that the number of nodes is not given as a parameters, as the model finds the probability a node will be reached in a number of steps, so it is agnostic to the number of nodes in the real implementation. This aspect is important to point out as the behavior of the model, explained before, shows up in the simulator and deployment only when a significative number of nodes is involved.

The gain in efficiency and pervasiveness is clearly visible as executing the model over a specific set of parameters takes only a few milliseconds and returns average values, not needing multiple runs, rendering possible to explore the same parameters that the simulators took weeks in less than a few hours on the same machine, with more precise (and reproducible) results obtained.

1.3 Polluter identification in MANETs

Another topic of our Thesis focuses on mobile, distributed, ad-hoc networks, and the inherent challenges in implementing a communication network which is both decentralized and restructured as possible and at the same time robust and efficient.
To this effect we initially approached the problem from related points of view:

- from perspective of the used coding, where we introduce the use of rateless coding to avoid data loss and availability problems in the network;
- from a pollution detection perspective, where we devise an efficient way of identifying malicious nodes in a completely distributed environment, especially important while used with the above encodings;

These two works are inherently connected to each other and are thus published, and presented here, together.

A central step in our study is the utilization of rateless coding techniques in completely distributed and unstructured networks, where knowledge of neighbors nodes is not exploited and only probabilistic flooding routing is available, thus rendering any centralized control or organization unneeded as long as two or more nodes are in range to communicate.
The chosen encoding is LT Code [5] where encoding *server* nodes divides a message chunk in n blocks, then proceeds to encode packets by combining together d distinct blocks by *logical XOR-ing* them, where $d < n$ and is randomly chosen by a Robust Soliton Distribution (RSD), guaranteeing that the chunk will be decoded in $n \cdot (1+\varepsilon)$ packets received, where ε tends to zero as n tends to infinite.

The server nodes than proceeds to wirelessly send the encoded packets around (similar to a fountain distributing water streams, from which the fountain coding synonym) which are received, stored and decoded when possible by decoder *client* nodes, which can, based on the scenario parameters, repeat the packets to their neighborhood to achieve better data diffusion at the cost of higher bandwidth usage and more difficult detection of eventual malicious nodes.

While the transmission overhead in respect to a sequential transmission of each block would be wasted in a reliable, non-lossy channel scenario between nodes, the repetition of data in the encoding helps circumvents the shortcomings of an ad-hoc wireless connection between wireless nodes, where node availability and transmission correctness can both not be guaranteed, nor expected.

Such peculiarities makes LT Codes a very well fit for all cases where there is a multitude of nodes, loosely or not hierarchically organized, in which the communication quality and reliance cannot be assured, such as wireless networks, P2P communications, and mobile ad-hoc wireless networks. All those topics represents in some way or another a strong point of this Thesis, making rateless coding a very helpful tool to have.

A topic which is a natural pairing to our rateless code based protocol is an efficient way to identify polluters, intended as nodes that voluntarily submit maliciously modified packets to corrupt the decoding process.

The more the network is lacking a central or local organization and collaboration, the clearer it becomes the threat that such malicious actors represents to the robustness and efficiency of the whole network and the harder it is to identify reliably bad acting nodes, especially recently appeared ones, either because they are fast moving or changing identity periodically.

A communication network like the one presented before, or in general any ad-hoc distributed network, is vulnerable to many attacks at many layers.
While protection against physical and transmission level attacks are out of the scope of this work and a few solution are available, we focuses on protecting against attacks on the decoding process by means of malicious nodes voluntarily sending malformed packets to pollute the overall chunk for unaware listening honest nodes.

As with many decentralized networks there is a clear advantage for a pollution detection algorithm to be completely distributed and all computation local to each node, which SIEVE [6] is, listening to other nodes reports, but constructing its beliefs only inside the node itself, without depending on any centralized authority.

Also, to be usable in a mobile environment such algorithm execution must be energy efficient, another strong fit for SIEVE being based on simple Belief Propagation iterations.

Finally the approach, while here mated to our LT Code exploiting protocol, is not by any mean bound to it. It is usable with any protocol that at any point of the decoding process, not necessarily at the end, is able to return a *check* containing a status (polluted or clean) and a list of all the nodes that contributed to that chunk and how much they contributed. The usage with an erasure fountain code can be seen as a worst case scenario, as the *check* can be produced only at the end of the decoding process and usually a lot of nodes contributes to a single chunk.

1.4 CCN and CCN-TV

Another topic that we studied in our Thesis is a simulative study of CCN-TV, a CCN based approach to TV service via internet video streaming.
Content-Centric Networking (CCN) [7] is a promising approach to data-centric architecture, in which the routing is name based (instead of address based) and the search is initiated by the data user rather than the data provider, leveraging intermediary nodes caching techniques to enhance performances exploiting data locality.

The need for CCNs comes from an average utilization of communication networks more and more to deploy and consume contents, be them video, documents, music or generic data.
In such context the ability to directly research and access wanted contents rather than having beforehand to know which nodes will serve which contents and then accessing their address, is indeed welcome.

Consider the case where someone wishes to access a video stream of a given social event, in the classic internet case he will have to beforehand known a provider of such service (and its address) or a service offering search services (and its address) such as a search engine. Once he has obtained the video service

provider address he will contact it and request the desired content, awaiting (and hoping) for a positive reply and only then beginning the data transfer, usually directly between.

In comparison the CCN approach does not need any previous knowledge as he would send an *Interest* packet, stating that he wishes to receive such service, and the packet is routed, based on a categorization of names, after a certain amount of hops to a machine who does provide such services.

The machine will send back the data, using previously touched nodes as intermediaries, so that if a neighbor of the requester asks for the same content he will not have to reach back the remote server again, but the first intermediary node will respond positively to his *Interest* request.

While a complete and general implementation overlaying current internet is still very difficult, having to face problems such as devising and adopting an efficient ontology of contents, to permit efficient routing, a more limited scope implementation, such as video on demand and real-time video services, is very promising and robust, integrating resource discovery and local caching directly in the routing protocol.

1.5 A unifying view of the Thesis

The Thesis flows around the central concept of enhancing the robustness and resilience of distributed networks through distinct techniques, each facing the problems from a different, sometimes complementary, point of view.
Two main directions are present in this work:

- the first one is using random graph modelization to exhaustively study the behavior of a system of interconnected networks over a set of common nodes called bridges; such study works toward efficiency and resilience by offering the tool needed to accurately study a network topology and its effects before deployment or modification;

- the second one is focusing on distributed networks by studying the effect of

novel routing and content discovery protocol (CCN), and polluter identification techniques coupled with rateless coding content distribution.

Both contributions aim to make distributed networks more efficient and robust, one in the planning and analyzing phase, the other during execution.
Our contribution tackle the problems following both an *off-line* and *on-line* approach, with a set of conceptual tools to be exploited as needed.

Our original contributions, as better explained in the following Chapters, span over the two main topics.

For the first topic contribution we exploited random graphs to develop a mathematical model of network interconnection [4, 8], where the model can be use to efficiently investigate the behavior of a given set of bridge-node interconnected networks, regarding the probability of finding a resource and the average number of query packets generated, given a certain number of hops permitted and the networks defined by a set of parameters.

The second line of work focuses on distributed mobile networks and ways to maximize their resilience during execution, such as developing a completely distributed polluter identification scheme called SIEVE [6, 9] which works over a P2P protocol we implemented base on rateless coding, or exploiting novel routing and content identification protocols such as CCN, focusing on real-time video streaming by adapting and studying the protocol in our CCN-TV approach [10], and successively extending the study to include crowd-sourced generated contents [11].

Chapter 2

State of the art

Chapter abstract:

In this Chapter we analyze, for each topic, the state of the art and related comparable works. While a more in depth analysis is present in each topic dedicated Chapter in Part II, the present Chapter helps in pointing out the background of the field as well as strong and lacking points of related published works.

2.1 Random graph modeling of overlay networks interconnection

Network interconnection and cooperation is indeed of growing importance in the future.

The bulk of related modeling literature focuses on evaluating single network behavior, which limits the statistical analysis to the merging of networks together, a very expensive and rarely feasible approach.

Inter-cooperation of network instances has been identified in [12, 13] as one of the future trends in the current Internet architecture development. When discussing logical networks, various techniques to achieve inter-communication among them have been presented.

Synergy [14] is an architecture for the inter-cooperation of overlays which provides a cooperative forwarding mechanism of flows between networks in or-

der to improve delay and throughput performances. Co-located nodes are, in the authors' opinion, good candidates for enabling such mechanisms and reduce traffic.

With a similar goal, authors in [15] propose algorithms tailored to file sharing applications, enabling a symbiosis between different overlays networks. They present hybrid P2P networks cooperation mechanisms and provide interesting observations on the appropriate techniques to perform network join, peer selection, network discovery, etc. Their simulations showed the effect of the popularity of a cooperative peer on the search latency evaluation, that is the more a node has neighbors, the better, as well as the effect of their caching mechanism which reduces (when appropriately adjusted) the load on nodes (but interestingly does not contribute to faster search).

Authors in [16] model an interconnected system by considering spaces with some degree of *intersection* between one another. They focus on different strategies to find a path between two overlays, and compare various routing policies analyzing which trade-offs lead to the best results. Trade-offs are considered in terms of number of messages, number of hops to find a result and state overhead. They provide a comparative analytical study of the different policies. They show that with some dynamic finger caching and with multiple gateways tactfully laid out in order to avoid bottlenecks due to the overload of a single gateway, they obtain good performances. Their protocol focuses on the interconnection of DHTs, while we focus on unstructured overlays.

Finally, [17] studies the co-existence of multiple overlay networks, namely Pastry and an unstructured overlay that uses a gossip protocol to improve its performance.

The topic of social network interconnection is a specific application of the network interconnection through bridging which is attracting a lot of interest due to the availability of multiple, expanded and existing social networks already deployed and in use.

The advantages offered by social inter-networking are centered about transparently "de-isolate" the single networks, exploiting the preexisting connections, which are compatible between the networks by mean of being based on the same

social structures and relationships, effectively enabling a virtual "over-network" extending and exploiting all the single social networks contents.

A summary of various works in [18] analyses the social inter-networking from the point of view of bridge interconnection, yet to the best of our knowledge no specific modeling tool exists to better study the various behaviors of the overall system.

2.2 Polluter identification in MANETs

Research in the field of rateless coding is well established in literature, with major examples being LT Codes [5] and their conceptual evolution Raptor Codes [19]. There have been interest in coupling rateless coding, and the relative concept of fountain encoding, with various fields interested by communication by many nodes and variably unreliable transmissions, like video streaming [20] or extending P2P BitTorrent file transfer [21, 22] and the techniques have been integrated in part of the error control protocol for LTE mobile networks.

Works in the field have mainly focused on finding more efficient ways of decoding (referring to performance to energy consumption ratio) and implementing encoding and decoding over non fixed sliding-window chunks of data [23].

The next generation of rateless codes implementations after LT codes is commonly agreed to be Raptor Codes [19], having linear time encoding and decoding, but computationally more intensive by using two encoding stages for encoding, where the second stage is an LT encoding.

MANET are vulnerable to attacks at any layer of the Internet model [24, 25]. In the area of network coding several efforts have been devoted to devise on-the-fly verification techniques carried out by participants [26, 27, 28, 29, 30, 31, 32] to identify the sources of corrupted data. The major drawback of these elegant methods is the high computational costs for verification and the communication overhead due to pre-distribution of verification information. In [33, 34, 35] previous works have been extended to limit the communication overhead and achieve high level of robustness to pollution in the context of network coding applied to static wireless mesh networks.

Error correction (and algebraic) approaches have been devised to deal with data corruption attacks in network coding [36, 37, 38]; these methods introduce coding redundancy to allow receivers to correct errors but their effectiveness depends on the amount of corrupted information. Recently, [39] has proposed an extension of previous work which aims at limiting the communication overhead required for verification of coded packets.

The data corruption attack we consider in this contribution is a well-known plague in peer-to-peer streaming systems. Unfortunately, all solutions developed in that research are not easily adoptable in MANET. The work by Wang et al. [40] proposes a detection scheme where each peer is able to detect receipt of corrupted blocks by checking the adherence of the decoded chunk to the specific formats of the video stream. Peers detecting polluted chunks send alert messages to the video server and the tracker. Upon receipt of an alert the server computes a checksum of the original chunk and disseminates it to all peers in the overlay. The checksum is used by peers to identify which uploader actually sent a corrupted block. Peers report their suspects to the server and true polluters cannot lie (the authors developed a non repudiation protocol to ensure that peers cannot lie when reporting suspects to the servers). Sequence numbers are used to tag alerts to deal with cycles in the overlay. This solution requires a centralized monitoring and management point that is not available in MANET. Therefore, it is hard (if not impossible) to adopt in the context we consider in our work.

The work by Li and Lui [41] presents a distributed detection algorithm and analyzes its performance. The technique is based on simple intersection operations performed by peers: each peer starts with a set of suspects that is equal to the entire neighborhood that is shrunk as long as chunks are downloaded from a random subset of uploaders independently chosen from the entire set of neighbors. The scheme allows malicious nodes to send corrupted blocks using a pollution probability. The technique is analyzed when the number of malicious nodes in the neighborhood is known in advance and an approximation is proposed when this quantity is unknown. The same approach has been also adopted in network coding wireless mesh networks [42, 43], where the analysis is restricted on the backbone network consisting of stationary (with minimal mobility) mesh routers. The technique is attractive thanks to its simplicity and fully distributed nature

although performance deteriorates when multiple polluter exist. Nonetheless, the technique works when then neighborhood of a node does not vary with time. Also in this case a comparison with our work is quite difficult.

The work by Jin et al [44] proposed a monitoring architecture to build and maintain a reputation system that peers use to select neighbors. The focus of the paper is on reputation computation, storage and load balancing among monitoring nodes. The results show that the system is able to detect malicious nodes up to a certain degree of lies. Nevertheless, the technique relies on the assumption that each peer is able to compute the amount of corrupted blocks received by each uploader during a monitoring period. Unfortunately, this capability is not available in the system we consider in our work: the original chunk can only be obtained if there are no malicious nodes among the chunk providers and if corruption is detected the honest node only knows that at least one of the chunk providers is malicious. In this case, comparison with our work is simply not possible.

Our work instead has been shown here to be very efficient, even with an unreasonably high number of malicious nodes, and computationally cheap, also being not limited to MANETs networks, but easily applicable to other more common P2P networks [45, 46].

2.3 CCN and CCN-TV

Internet usage patterns changed a lot in the last decade, where an ever-expanding multitude of sites and contents offers shifted the focus to content sharing and consumption, still offered on the IP underlying protocol.

Many *Future Internet* templates offer evolutions to what the current technical and social structure is, usually by shifting focus on content itself, rather than on the content provider, increasing role of P2P connectivity models, enabled by a greater ever-present connectivity between a plethora of devices, and considering the paradigm-breaking concept of pervasive mobility.

Many proposals move the data in a primary position, by adopting models like Publish-Subscribe [47], where a service provider publishes its offers in terms

of datas and services and interested nodes subscribes and are then given the requested service, or caching approaches of Store-and-Forward, original of DTN networks [48].

Amidst the multitude of proposals a common idea links many of them, where the contents are made central and are offered by providers and requested by consumers, adequately routed to each other by intermediary nodes based on their knowledge of the current offers of services.
Local cache is used to avoid repeated traffic as much as possible if the contents are already "in the neighborhood".
The main proposal is this field are CCN, sometimes referred also as Named Data Networks (NDN), [7].

While promising however the field is still loosely defined by the general rules of Interest and Data packets exchange and caching usage, but a lot of study is currently focusing on routing algorithms [49, 50] and even in less or more radical modifications of the protocol to specific scopes, like our present work on CCN-TV.

A reference software implementation also exists, called CCNx [51], yet works have mostly been theoretical so far, with yet to assess the success and efficiency of a real-world deployed in the wild implementation of any kind.

Chapter 3

Thesis contributions

Chapter abstract:

In this Chapter we present the original contributions of our works and this Thesis.

While we reserve the more technical details to the Chapters in Part II, we describe the approach used in each work and, where relevant, the differences with the mainstream implementation (eg. Section 3.3).

With the aim of making clear how each contribution is constructed we delegate the details to the specific Chapters, focusing on shortly, but clearly, explaining the whole process.

3.1 Random graph modeling of overlay networks interconnection

We describe the network by means of its degree distribution p_k, i.e. the probability a randomly chosen node has k connections in the same network, and a probability distribution s_i, meaning the fraction on nodes belonging to i distinct overlay, with $1 \leq i \leq X$, where X is the number of networks present.

The considered routing algorithm for the queries is *flooding base*, i.e. a node starting a search forwards the query (with probability $p_f(i)$) to a random subset of its adjacent neighbors.

They then forward the query to a random subset of their one-hop neighbors and so on, until the maximum number of hops, defined as *Time To Leave (TTL)*,

eventually forwarding the query up to the TTL if the receiving node has the resource, depending on the variation of the search algorithm chosen.
To model the search we define the goal as to reach at least one node containing the desired resource with the queries, which popularity is defined as $0 \leq \alpha \leq 1$, meaning the probability a random node will have the resource.

We model a single network as a random graph degree distribution $G_0(z)$ and derive $G_1(z)$ as the probability generating function (p.g.f.) of the adjacent neighbors connections excluding the edge we came from, chaining it to obtain the p.g.f. of a given TTL neighborhood, e.g. $G_0(G_1(G_1(z)))$ for $TTL = 3$.

To then model the interconnection we consider that some peers will be *bridges*, called *Synapses* in the original work [52], enabling the query to diffuse in other networks neighborhoods too. The related p.g.f., relative to probability distribution s_i, is $F(z)$, and by combining it with $G_0(z)$ we obtain $M(z)$, the p.g.f. for the number the neighbors considering the bridges too. By extension, combining with $G_1(z)$ too we obtain $N(z)$ the p.g.f. for the neighborhood, excluding the selected edge. We are then finally able to define the p.g.f. for the number of neighbors t hops away as $N_t(z)$, such that $N_1(z) = M(z)$, $N_2(z) = M(N(z))$, $N_3(z) = M(N(N(z)))$, and so on.

Once we modeled the interconnections, we combine our p.g.f. with our p_f probability the query will be forwarded to define various p.g.f. leading to $Q_t(z)$, the p.g.f. for the probability distribution of the number of neighbors t hops away from a randomly chosen node that received a query, which product for each $1 \leq t \leq TTL$ gives us $T(z)$, yielding us the average number of queries $m = T'(1)$.

Finally by introducing the α resource distribution probability we define the p.g.f. $H_t(z)$ for the probability distribution of the number of neighbors t hops away for a randomly chosen node that received a query and also possess a copy of the requested resource, which product for each $1 \leq t \leq TTL$ gives us, as before, $H(z)$, giving us the search hit probability $p_{hit} = 1 - H(0)$. We also alternatively defined a different $R(z)$ to also experiment with the variant in which a node owning the resource will not forward the query if it is received. A better description of the standard version and this variant is present in Section 4.3.3 and 4.3.5.

3.2 Polluter identification in MANETs

We exploited the non sequential and redundancy properties of erasure codes, LT Codes in particular, to raise the robustness and efficiency of unstructured ad-hoc wireless networks.
Avoiding any kind of routing assumption and only basing our transmission on flooding patterns makes the approach readily applicable to any topology of networks or degree of mobility between the nodes, yet creates new efficiency and reliance problems, as there are no guarantees that the packet will be received or that the percentage of lost or corrupted ones will be under a certain threshold.

A server node, i.e. a node that already possesses the chunk of data to be transmitted and is willing to share it with others, will start encoding the packets to be sent out continuously. Differently from a direct transmission there is a sequential flow of packets after which the transmission ends, instead each packet payload is composed of 1 to n blocks encoded together using a XOR function, where n is the number of equal size blocks the chunk is divided into and the number of blocks a packet contains is called *degree* or d.

Necessary to the decoding in the packet header there is an hash h, identifying the chunk, the number n of blocks it has been divided into, the degree d of the current packet and a list L of the d distinct blocks that are encoded in the payload.

If a particular probability distribution is used to randomly choose the degree of the generated packets the decoding is mathematically proven to be possible after having received $n \cdot (1 + \epsilon)$ packets, where ϵ tends to 0 when n tends to ∞, meaning also that the overhead grows smaller the more blocks the chunk is divided into. Such distribution is called *Robust Soliton Distribution* (RSD) and it is a variant of the *Soliton Distribution* modified to guarantee the previous decoding assertion [53].

It is to be noted that a packet with $d = 1$ and $L = \{k\}$ (where of course $1 \leq k \leq n$) contains in the payload the exact k-th block and is the only packet that is readily decodable (not depending on other blocks to be decoded). For all $d > 1$ packets the payload needs at least one already decoded block form its list L, the exact number depending on the decoding algorithm used, to decode the

remaining information (blocks in L not already known), but it can be stored for later usage, when additional information has been received.

Finally the server broadcasts wirelessly the packet to all nodes who can hear it (hence the term fountain coding), not expecting any reply or acknowledge and thus keeping on encoding and sending packets until a certain prefixed condition has been met, like a timeout or a number of packets sent.

Client decoding works by collecting all the packets received by the client, then subdividing them by the h identifier, indicating different decoding sessions and data. Focusing on a specific hash generally the payload and relative metadata is extracted (degree d and list of blocks L) and passed to the used algorithm, in order of efficiency:

- *Sequential*, a very inefficient test-only algorithm only keeping distinct $d = 1$ blocks and discarding all the others, effectively negating LT Code benefits;
- *Belief Propagation*, a basic algorithm crossing known decoded blocks with stored still un-decoded data to progressively trim down blocks;
- *Gaussian Elimination*, a more efficient comparison algorithm then BP, permitting a faster decoding, even when there is scarcity of degree one blocks.

Then depending on the parameters passed to the nodes a client can, once obtained the decoded chunk, become a server for that chunk, simply repeated heard packets, be their chunk already decoded or not or simply do nothing other than receiving.

As seen previously, decoding is an incremental process in which various nodes collaborate in seeding the decoding pool, and which result is known only when a certain number of packets has been received, an average of $n \cdot (1 + \epsilon)$, where n is the number of blocks the original chunk has been divided into if the decoding and ϵ an overhead dependent on n.

The whole SIEVE [6] approach is based on the concept of checks, which are produced by the receiving node and can be *negative* or *positive*, respectively if the decoding is successful or fail as soon as an inconsistency in the underlying system of linear equations for the decoding is found.

Each node internally maintains a list $\mathcal{L}_n$ of all internally created checks and, to raise the availability of checks data, broadcasts the check each time it is produced and periodically, once every T_s seconds, a set of random Q checks in $\mathcal{L}_n$.

The checks in $\mathcal{L}_n$ and all received checks are used to build the bipartite factor graph $\mathcal{G}_n=(\mathcal{U}, \mathcal{C}, \mathcal{E})$, where vertex set $\mathcal{U}$ is the set of uploader nodes, vertex set $\mathcal{C}$ is the set of checks, and an undirected edge $\{i, I\} \in \mathcal{E}$ exists if and only if check $I \in \mathcal{C}$ depends on uploader $i \in \mathcal{U}$. We also refer to the set of uploaders involved in check I as $\mathcal{U}_I$ and the set of checks that node i contributes as an uploader as $\mathcal{C}_i$.

The goal of the inference problem of identifying malicious nodes from a given number of checks is to estimate the hidden state of the nodes (malicious or not) and we adopt the *Belief Propagation* (BP) algorithm for the task [46].

From a neutral value of 0.5 for the probability of each node of being malicious and honest each check works towards implying the set uploader nodes as honest or malicious depending if *negative* or *positive*, finally extracting an estimate after a reasonable amount of iterations has been collected.

To face the constantly changing environment the BP estimation is designed to be incremental, rather than a single iterative instance, utilizing a *sliding window* approach over a certain amount of time. Each time a given node is considered to be a suspect polluter over a certain threshold the node is included in the list of probable polluters or, if already present, its counter increased by one, then, the overall ranking, is produced based on sorting the counter value of each node in the probable polluter list.

We currently do not integrate a way to phase out probable polluters over time, if ever this is to be considered a feature it would be easy to decrease the counter based on some heuristic, for example every given amount of time.

3.3 CCN and CCN-TV

In keeping in line with the content centric CCN vision the communication is initiated by the client, who generates an *Interest* message and expects to receives a *Data* message, the only two type of messages used.

The issued *Interest* packet serves to request a specific content and is routed by successive nodes until a provider for that content is found. The server will then reply with *Data* packets which will follow the inverse path of the *Interest* packet back to the original client requesting them. All the message exchanges happens between the nodes *faces*, which are the CCN equivalent of a network interface.

Every intermediate node is allowed, and usually expected if able, to keep in its cache the forwarded content, such that a subsequent independent *Interest* packet for the same content reaching the node won't have to be forwarded up to the server, but will be replied by the cached *Data* packets, thus easily resolving congestion on popular contents and raising data availability and speed for locally available contents.

In this scheme a few issues are of great importance and directly influences the overall performance of the protocol. The first and foremost is *routing* and by extension *namespace definition*, as the *Interest* packet is routed node by node by using the content requested as an "address", where usually a hierarchy of categories (e.g.: */media/video/sport/curling/olympics2006*) is defined in a namespace and such layers are used to route to nodes which are in the specific routing Tables responsible for that root name, in a manner reminiscent of subnet addressing in common IP routing.

Our specific approach, aimed at realtime video streaming, and called CCN-TV, specifies a few aspects of the base CCN protocol and adds a few fields to the *Interest* and *Data* to manage client bootstrap and requirements of live video feeds. Such addition are anyway easily adaptable to classic CCN message structure by merging them as additional fields in the content name and specifying a no-caching, direct forward policy on participating nodes for *Interest* packets working as *bootstrap* messages for a client.

Internally the node keeps the same structures as a standard CCN node:

- a *Content Store (CS)* which is a cache memory for received contents and managed by a chosen replacement policy;
- a *Forwarding Information Base (FIB)* which is a forwarding Table, different from the IP one as a given content name can have more *faces* to allow

Interests to be forwarded to more potential sources;

- a *Pending Interest Table (PIT)* which is a Table keeping track of the forwarded *Interest* packets and the *faces* they come from, to be able to route back *Data* replies.

At the reception of an *Interest* the node searches its CS if the requested content is already available, and if found is sent back to the requesting node.
Otherwise the PIT is consulted to see if the node has already received other *Interest* packets for the same content, if yes the received *Interest* arrival *face* is added to the PIT and nothing more.
If not the FIB is consulted to find the a matching entry to forward the *Interest* to, if one is found the *Interest* is added to the PIT and sent to the appropriate node, otherwise it is dropped, not being able to either satisfy of forward the request.

Before starting the real data transfer in the CCN-TV approach a client node bootstraps the channel between him an the server, by sending a particular *Interest* with the desired *domain* (the content provider) and *channelID* (the specific channel), the *Status* and *Nonce* (the added custom fields we noted before) respectively set to "BOOTSTRAP" and a random, but reasonably unique, numerical identifier, ensuring each distinct bootstrap request for the same content will be distinguishable.

The bootstrap *Interest* is specially treated by the intermediate nodes by simply forwarding it up to the server node, ignoring the caches for the same content they may have in the CS. This is to avoid sending the client a video frame not suitable to start the playing, i.e.: a frame which is not a key frame (called in jargon I-frame) or a stale (already expired) I-frame. The *chunkID* of the chunk containing the last generated not expired I-frame and the index for that frame is then sent back to the client with a *Data* packet, which will start the standard requesting exchange, with the only difference that the *Interest* is sent only once for a stream of *Data* packets to reduce network load, acting more like a subscription.

A mechanism based on the delay on the bootstrap (and subsequent request) messages works as flow control, setting the size for the sliding window buffer of received frames, essential in live-streaming video, which must be big enough to avoid emptying it by delayed *Data* packets, but not too big to put un unreasonable

delay in the video playing.

The last addition to the standard CCN scheme to conform to real-time video streaming is the ability to request a retransmission of a given chunk. By setting the *Status* field of an *Interest* packet to "Retransmission" the intermediate nodes replies with the content if the have it in their CS, otherwise they forward the *Interest* (if they a matching *face* in the FIB) regardless, eventually adding it to their PIT.

Part II

Detailed works

Chapter 4

Random graph modeling of overlay networks interconnection

4.1 Overview

Interconnection of multiple P2P networks has recently emerged as a viable solution to increase system reliability and fault-tolerance as well as to increase resource availability. In this Thesis we consider interconnection of large scale unstructured P2P networks by means of special nodes (called *Synapses*) that are co-located in more than one overlay. Synapses act as *trait d'union* by sending/forwarding a query to all the P2P networks they belong to. Modeling and analysis of the resulting interconnected system is crucial to design efficient and effective search algorithms and to control the cost of interconnection. To this end, we develop a generalized random graph based model that is validated against simulations and it is used to investigate the performance of search algorithms for different interconnection costs and to provide some insight in the characteristics of the interconnection of a large number of P2P networks.
Useful applications of interconnecting unstructured P2P networks spans from interoperability of existing data-bases ensuring backward compatibility, to the "hot" topic of Typed Smart Grid where users can be aware of the "type" of energy they want to produce and buy.

The last decade has seen the rise of peer to peer networks with a variety of ap-

plications, such as file sharing, resource lookup, real time services, up to the most recent research in SmartGrids. Common issues which affect all P2P systems, such as scalability, fault tolerance and security, arise from the different peculiarities each class of applications might expose. Increasing the locality properties of such systems, be it geographical, semantic, network, or social-based locality, is one of the most valued approaches to face such challenges: by grouping together peers representing users, and increasing their connections with one another, one can improve scalability, fault tolerance, and security (consider the possible creation of a "circle of trust" amongst nearby peers).

In this Thesis, we consider the interconnection of large-scale unstructured P2P networks by means of special nodes called *Synapses* [52], which are co-located in more than one network, and act as *connectors* by sending or forwarding a query to some or all the P2P networks they belong to. Modeling and analysis of the resulting interconnected system is crucial to design efficient and effective search algorithms and to control the cost of network interconnection. Yet, simulation and/or prototype deployment based analysis can be very difficult - if not impossible - due to the size of each component (we consider large scale systems that can be composed of millions of nodes) and to the complexity arising from the interconnection of several such complex systems.

Our contribution

To overcome this strong limitation, we develop a generalized random graph based model to represent the topology of one unstructured P2P network, the partition of nodes into Synapses, the probabilistic flooding based search algorithms, and the resource popularity. We validate our model against simulations and prove that its predictions are reliable and accurate. We use the model to investigate the performance and the cost of different search strategies in terms of the probability of successfully locating at least one copy of the resource and the number of queries as well as the interconnection cost. We also gain interesting insights on the dependency between interconnection cost and statistical properties of the distribution of Synapses. Finally, we show that thanks to our model we can analyze the performance of a system composed of a large number of P2P networks.

To the best of our knowledge, this is the first work on model-based analysis of interconnection of large scale unstructured P2P networks.

The topic is organized as follows: Section 4.2 describes our system, Section 4.3 presents the mathematical derivation of the generalized random graph model we develop, Section 4.4 contains model validation through simulation, as well as model exploitation to study the performance of three search algorithms, Section 2.1 discusses related works, and in Section 7.1, we draw conclusions and outline ongoing activities that extend the current work.

4.2 System description

In this Thesis, we focus on unstructured P2P networks where peers organize into an overlay network by establishing application level connections among them. The topological properties of an overlay network are represented by the number of connections of any of its participants. To this end, we describe an overlay by means of the degree distribution $\{p_k\}$ that can be interpreted as the probability that a randomly chosen peer has k connections in the overlay ($\sum_{k=1}^{\infty} p_k = 1$).

We consider a set of X unstructured P2P networks that are interconnected thanks to a subset of peers that belong to multiple overlays (these special peers are denoted as *Synapses*). Any peer may then belong to $i \in \{1, \ldots, X\}$ overlays: we denote i as the *Synapse degree* of a peer. The interconnected system is then described by $\{s_i\}$ ($i \in \{1, \ldots, X\}$) where s_i is the fraction of peers belonging to i overlays ($\sum_{i=1}^{X} s_i = 1$).

The search algorithm we consider is *flooding-based*. A peer starting a search sends *queries* to a randomly chosen subset of its one-hop neighbors. These nodes forward the queries to a randomly chosen subset of their one-hop neighbors, excluding the query originator, and so on until the maximum number of allowed hops, i.e. the query time-to-live (TTL). A simple schema of interconnection through Synapses and search is depicted in Figure 4.1.

We also consider a variation of this search algorithm where a query is not forwarded by peers that own a copy of the resource. We focus on probabilistic versions of this general algorithm where any peer flips a coin before sending or forwarding a query to a specific neighbor. We allow the weight of this coin to

be dependent on the Synapse degree of a peer; hence, a peer that belongs to i overlays sends/forwards a query to a particular neighbor with probability $p_f(i)$ ($i \in \{1, \ldots, X\}$). Please note that $\{p_f(i)\}$ ($i \in \{1, \ldots, X\}$) *is not* a probability distribution hence in general $\sum_{i=1}^{X} p_f(i) \neq 1$.

The goal of a search is to localize at least one resource related to the key we are looking for. There could be more replicas of the same resource hosted by different peers for two reasons: a resource is popular and/or is owned by peers located in different P2P networks. We represent resource popularity by $0 \leq \alpha \leq 1$, the average fraction of nodes that globally hold a copy of a given resource, and interpret it as the probability that a randomly chosen peer owns a copy of the resource.

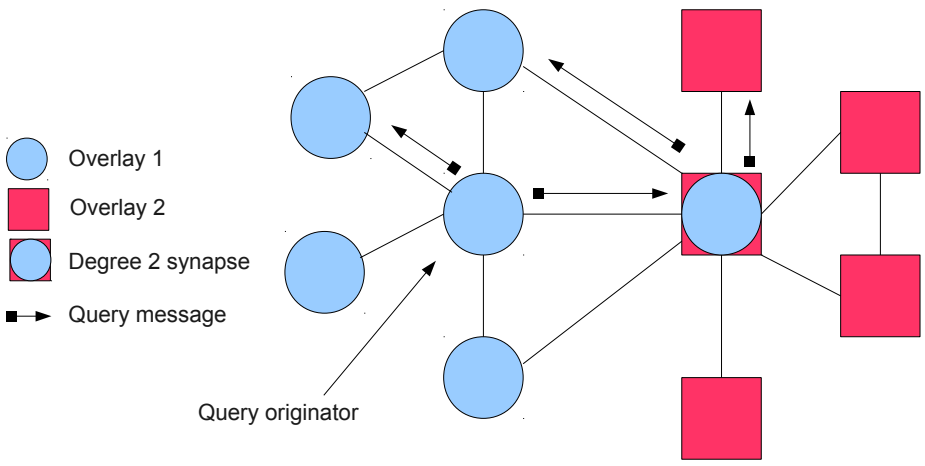


Figure 4.1: Example of two P2P interconnected networks ($X = 2$) and one degree 2 Synapse that belongs to both

4.3 System model

This Section illustrates the random graph modeling approach to represent one overlay topology, the interconnection of X P2P networks, the search algorithm, and resource popularity as described in Section 4.2.

4.3.1 One overlay topology

Each P2P network is organized into an overlay that we model as a generalized random graph whose degree distribution is $\{p_k\}$ that can be interpreted as the probability that a randomly chosen peer has k connections in the overlay. The random graph degree distribution is a probability distribution therefore we consider its probability generating function (henceforth denoted as p.g.f.) that is equal to $G_0(z) = \sum_{k=0}^{\infty} p_k \, z^k$. To correctly characterize the neighborhood of a randomly chosen peer we also need to characterize the probability distribution of the number of connections of a peer reached by randomly choosing an *edge* of the overlay. This probability is proportional to the degree of the peer (kp_k) and it can be proved that its p.g.f. is given by $\frac{\sum_k kp_k z^k}{\sum_k kp_k} = z\frac{G_0'(z)}{G_0'(1)}$ where $G_0'(z)$ denotes the first derivative of $G_0(z)$ with respect to z and $G_0'(1)$ yields the average value of distribution $\{p_k\}$. Finally, to characterize the number of connections *excluding* the edge we chose we obtain the p.g.f. $G_1(z) = \frac{G_0'(z)}{G_0'(1)}$. Starting from Equations defining $G_0(z)$ and $G_1(z)$ we can compute the p.g.f. for the number of two hops neighbors of a randomly chosen peer as $G_0(G_1(z))$. Similarly, the p.g.f. for three hops neighbor is given by $G_0(G_1(G_1(z)))$, and so on.

For a detailed overview on analyzing generalized random graphs using generating functions, we refer the reader to [54].

4.3.2 Interconnection of multiple P2P networks

To interconnect multiple overlays we consider some peers as Synapses nodes: these peers belong to multiple P2P networks hence the interconnected system can be modeled by the probability distribution $\{s_i\}$ (with $i \in \{1, \ldots X\}$). The elements of this distribution describe the fraction of nodes belonging to multiple P2P networks: s_i is the fraction of nodes that belong to k P2P networks. Its

p.g.f. is given by $F(z) = \sum_{i=0}^{\infty} s_i \, z^i$. If we consider one of the X P2P networks including the Synapse nodes then the p.g.f. for the number of connections of a randomly chosen peer can be written as

$$M(z) = s_1 G_0(z) + s_2 G_0^2(z) + \ldots + s_X G_0^X(z) = F(G_0(z))$$

that is, if the chosen node is a degree 1 Synapse (this event has probability s_1) then the number of connections is represented by $G_0(z)$. If the node is a degree 2 Synapse (this event has probability s_2), then the number of connections is represented by the sum of two independent random variables whose p.g.f. is $G_0(z)$; it is well-known that the generating function of the sum of two independent random variables is equal to the product of the respective generating functions yielding the $G_0^2(z)$ factor in the equation for $M(z)$. The same reasoning is valid for Synapses whose degree is greater than 2.

A similar expression can be written for the neighborhood of a node reached by following one randomly chosen edge *excluding the selected edge*:

$$N(z) = s_1 G_1(z) + s_2 G_1(z) G_0(z) + \ldots + s_X G_1(z) G_0^{X-1}(z) = \frac{G_1(z)}{G_0(z)} F(G_0(z)).$$

If we denote as $N_t(z)$ the p.g.f. for the probability distribution of the number of neighbors t hops away from a randomly chosen node we have that: $N_1(z) = M(z)$, and $N_2(z) = M(N(z))$, and $N_3(z) = M(N(N(z)))$, and so on. From these p.g.f. the average number of neighbors can be computed by evaluating their first derivative w.r.t. z in $z = 1$.

As such, each probability distribution $\{s_i\}$ induces an *interconnection cost* that we define as the average number of P2P networks a randomly chosen node belongs to:

$$f = F'(1). \tag{4.1}$$

4.3.3 Search algorithm

To model a flooding-based search in the interconnected system, we consider the set of probabilities $\{p_f(i)\}$, where $i \in \{1, \ldots X\}$. A peer belonging to i overlays

sends/forwards a query to a particular neighbor with probability $p_f(i)$, where $i \in \{1, \ldots X\}$). Therefore, $\{p_f(i)\}$ *is not* a probability distribution.

We denote as q_h the probability that h first hop neighbors received a query from the peer that started the search. If the peer belongs to i overlays, it sends a query to one of its neighbors with probability $p_f(i)$. Therefore, the number of neighbors that receive the query follows a binomial distribution with parameter $p_f(i)$. Therefore, it is well known that the probability distribution $\{q_h\}$ has p.g.f. given by [54] $Q(z) = \sum_{i=1}^{X} s_i G_0^i(1 + p_f(i)(z-1))$.

Similarly, for the p.g.f. of the probability distribution describing the number of queries sent by a node reached by following a randomly chosen edge, we obtain:

$$R(z) = \sum_{i=1}^{X} s_i \, G_1(1 + p_f(i)(z-1)) \, G_0^{i-1}(1 + p_f(i)(z-1)). \tag{4.2}$$

If we denote as $Q_t(z)$ the p.g.f. for the probability distribution of the number of neighbors t hops away from a randomly chosen peer that received a query, we have that: $Q_1(z) = Q(z)$, $Q_2(z) = Q(R(z))$, and $Q_3(z) = Q(R(R(z)))$, etc. As a special case, we may consider constant forwarding probabilities, *i.e.* $p_f(i) = p_f, \forall i \in \{1, \ldots X\}$. In this case, we would obtain: $Q(z) = M(1 + p_f(z-1))$ and $R(z) = N(1 + p_f(z-1))$. Since the p.g.f. of the probability distribution of the sum of independent random variables is given by the product of the corresponding p.g.f., the total number of queries generated by a search issued by a randomly chosen peer is described by: $T(z) = \prod_{t=1}^{TTL} Q_t(z)$ yielding the *average number of queries*

$$m = T'(1). \tag{4.3}$$

4.3.4 Hit probability

We model resource popularity by $0 \leq \alpha \leq 1$ that is the average fraction of peers that globally hold the given resource. We interpret this parameter as the probability that a randomly chosen node owns a copy of the resource.

If we denote as w_h the probability that h first hop neighbors hold a copy of the requested resource *and* received a query from a peer that belongs to i overlays we note that the number of such neighbors follows a binomial distribution

with parameter $\alpha p_f(i)$. If we denote as $H_t(z)$ the p.g.f. for the probability distribution of the number of neighbors t hops away from a randomly chosen peer that received a query *and* hold a copy of the requested resource then we have that: $H_1(z) = Q_1(1 + \alpha(z-1))$, $H_2(z) = Q_2(1 + \alpha(z-1))$, $H_3(z) = Q_3(1 + \alpha(z-1))$, and so on. Therefore, the total number of search hits is described by a probability distribution whose p.g.f. is given by: $H(z) = \prod_{t=1}^{TTL} H_t(z)$ yielding the search *hit probability*

$$p_{hit} = 1 - H(0). \tag{4.4}$$

4.3.5 A variation of the search algorithm

To model a search algorithm where peers that own a copy of the resource do not forward a query message it suffices to redefine $R(z)$ in Equation 4.2. In particular, when a peer owns a copy of the resource the number of its neighbors that receive the query is equal to 0: this happens with probability α. In Equation 4.5 this is represented by the term α that can be written as $\alpha \cdot p_0 \cdot z^0$ with $p_0 = 1$. With probability $1 - \alpha$ Equation 4.2 holds, therefore we obtain the p.g.f. of the probability distribution describing the number of queries sent by a node reached by following a randomly chosen edge as:

$$R(z) = \alpha + (1-\alpha) \sum_{i=1}^{X} s_i G_1(1 + p_f(i)(z-1)) G_0^{i-1}(1 + p_f(i)(z-1)). \tag{4.5}$$

The definition of $Q_t(z)$, and $T(z)$, and m remains unchanged.

4.4 Results explained

In Sections 4.5 and 4.6, we will first show the results of the model validation, performed via a heavily multi-threaded simulator, written in Erlang, that reproduces, in terms of message routing, the exact behavior of a system described by our model. Also, we will show the results of some broad system evaluations made possible by the use of our model to compute metrics that would otherwise, if performed by means of simulations, require too much in terms of simulation time and computational power.

In our analysis, we consider different routing policies that can be employed in our scenarios, modeled by defining the $p_f(i)$ mentioned in Section 4.2. Those are:

- $p_f(i) = \frac{1}{i}$, henceforth referred to as *1/i*, i.e. the probability of selecting a neighbor is inversely proportional to the number of overlays a node is connected to. This routing tends to maintain a constant number of messages, but "flattens" the interconnected topology, not allowing Synapse nodes to exploit the extended neighborhood.
- $p_f(i) = min(1, \frac{z_{max}}{zi})$, henceforth referred to as $zmax$, where $z = E[\{p_k\}]$ is the average number of neighbors for a node based on the current degree distribution and z_{max} is a system parameter, specified upon design, indicating the upper bound for the average number of forwarded messages. This policy allows for a better exploitation of Synapse nodes, while still finely limiting the number of messages in the system. In our evaluations, z_{max} has been set to $2z$, twice the average number of neighbors per node.
- $p_f(i) = 1$, henceforth referred to as *flood*, i.e. a routing where every node selects forwards a message to *every* neighbor, regardless of the number of connected overlays.

In both simulations and evaluations, the individual overlays have been modeled following the neighbors degree distribution measured in [55] from real world applications and used already in [56], in order to have an accurate overlay model.

4.5 Model validation

In order to evaluate the accuracy of our model in predicting the performance indexes of a real network, we validated the obtained results by means of simulation. The simulator employs standard statistical procedures to estimate 68% and 95% confidence intervals for the p_{hit} and m indexes defined in Section 4.3.

4.5.1 Simulation methodology

The simulator has been developed from scratch in Erlang. The choice of Erlang has been driven by its native multi-threading capabilities and inter-process communication model based on the message passing paradigm embedded in the language, thus allowing for a rapid implementation of an accurate network model made of node processes running independently and exchanging messages with one another. Each process has a list of other processes it can exchange messages with, that constitutes its neighborhood.

We consider N_s independent realizations for the interconnected overlay topologies (in our experiments $N_s = 30$); each interconnected topology is used to obtain one realization of m and p_{hit}. The h^{th} realization is obtained as follows:

- We first generate a new topology, made of X overlays interconnected by Synapse nodes, using as input parameter the number of nodes $N = 500000$, the nodes degree distribution $\{p_k\}$ [55], and the $\{s_i\}$ to be validated;
- From the generated topology file, the simulator instantiates N node processes and assigns each the corresponding list of neighbors;
- One or more resources are then seeded in the system, according to their respective popularity α, by sending a `PUT(value)` message to $N\alpha$ random nodes;
- Separate worker processes take care of sending a query message `SEARCH(value,TTL)` to each node process in the network.
- Meanwhile, a listener process receives then the responses, either the resource being found or the TTL being reached, and of computes the statistics.

4.5.2 Topology generation

The generation of a network made of interconnected overlays mainly consists of generating first X individual overlay topologies, and then connecting them by "merging" nodes from different overlays in one Synapse node, thus creating nodes with extended neighborhoods spanning across all the connected overlays.

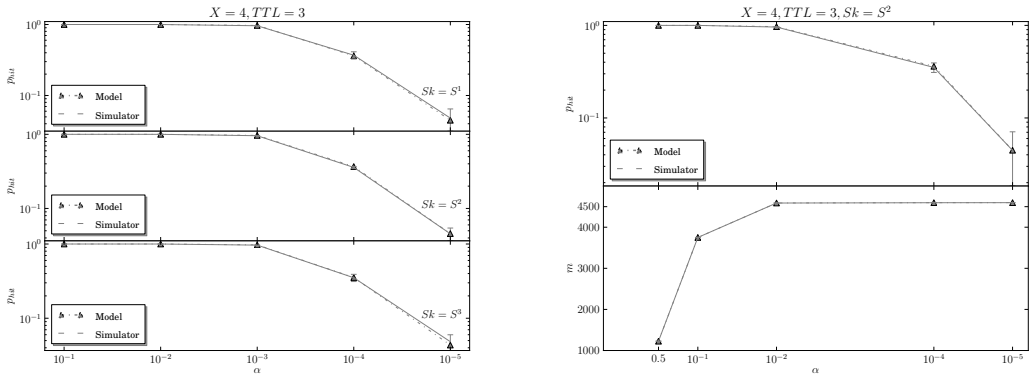


Figure 4.2: p_{hit} for different α and s_i (left) and alternative search algorithm (right)

In order to generate X random graphs with a specified degree $\{p_k\}$ we relied on the algorithm presented in [57], that provides short generation times while guaranteeing the respect of the specified degree.

4.5.3 Validation results

The first validation we performed was conducted for a system with only one overlay ($X = 1$). For the sake of brevity we only show the results for the *flood* routing strategy, $\alpha = 0.0001$, and $TTL = 3$. Table 4.1 shows the model is very accurate and faithfully predicts results when compared to the simulation output.

We then validated various scenarios with a higher number of interconnected overlays ($X = 4$), at $TTL = 3, 4$ and with different values of α, different routing policies and different distributions $\{s_i\}$. We considered the distribution for the degree of Synapses summarized in Table 4.2.

Table 4.1: m for different s_i distributions: comparison between model and simulation

	Model	**Simulation (95% C.I.)**
p_{hit}	0.3733	0.373552 ± 0.003852
m	4822.63	4821.57 ± 0.0498

Figure 4.2 (left) shows a comparison between the computed p_{hit} for different values of α and the corresponding simulation results, while Table 4.3 summarizes the same comparison for m. The results show that both performance metrics fall within the confidence interval of the simulation results.

Furthermore, we validate the system against the alternative search algorithm detailed in Subsection 4.3.5. For the sake of brevity, we are showing results only for S^2 since the same conclusions can be drawn for S^1 and S^3. Figure 4.2 (right) shows both p_{hit} and m against different values of α, since with this algorithm the number of message is dependent of the resource popularity. Even in this scenario, the model results fall within the confidence interval estimated by the simulator.

Therefore, we can safely conclude that our model is accurate in predicting the behavior of the performance indexes in a broad range of different scenarios. Furthermore, while simulations required hours of CPU time to complete solving our model took less than one second with a solver implemented in C.

4.6 Model exploitation

After validating the model we conducted a few analysis to show its usefulness in the design phase of the interconnection of several peer-to-peer networks.

4.6.1 Comparison of different routing policies

A first evaluation concerns the choice of a specific routing policy in the system, i.e. the definition of different $p_f(i)$. In this case, we want to compare for values of α down to 10^{-6}, the performances in terms of p_{hit} and m for the distribution of degree of Synapses S^1 (results for the other two distributions suggested similar considerations and are omitted for the sake of brevity), $X = 10$, and $TTL = 3$. Please note that to achieve a reliable measurement via simulation for $\alpha = 10^{-6}$

Table 4.2: Definition of the $\{s_i\}$ distributions used for validation

S^1	$s_1 = 0.7, s_2 = 0.1, s_3 = 0.1, s_4 = 0.1$
S^2	$s_1 = 0.4, s_2 = 0.3, s_3 = 0.2, s_4 = 0.1$
S^3	$s_1 = 0.1, s_2 = 0.2, s_3 = 0.3, s_4 = 0.4$

we would need to conduct complex simulations (at least 1000000 nodes) for a long simulation time (ideally each of them to be queried individually for multiple topology realizations).

Figure 4.3 show the values of p_{hit} for the 3 different policies and different resource popularities, while Figure 4.4 depicts the average number of messages for the 3 policies in the case of propagation of queries up to TTL hops (Figure 4.4(b)) and for the query propagation that stops when reaching a node holding a copy of the resource (Figure 4.4(a)) modeled in Subsection 4.3.5. In the former case, the number of messages is independent of the resource popularity while in the latter case we note that reduction of the number of query messages can be obtained for popular resources, i.e., for $\alpha > 0.01$.

In this case, the model allows for a simple cost/benefit evaluation, based on the expected popularity of a resource. For one, we can notice an almost tenfold increase in the number of messages between the *zmax* and the *flood* policy, to which it does not correspond a proportional increase in the p_{hit}.

4.6.2 f-cost based evaluation

In a cost/benefit analysis of the interconnected system, we consider p_{hit} as our benefit metric whereas m and f are considered as costs. Another kind of evaluation we performed consists of fixing the f cost and analyzing which distributions $\{s_i\}$ lead to better performances (p_{hit}) and minimum cost (m).

To this end we considered all distributions $\{s_i\}$ that can be defined for $X = 5$ where the individual probabilities are non-zero multiple of 0.05. We considered 3 values of f (namely, $f = 2, 3, 4$) and compared the performances of every distribution $\{s_i\}$ with given f for $TTL = 2$. Again, please note that this analysis

Table 4.3: m for different s_i distributions: comparison between model and simulation

	Model	**Simulation (95% C.I.)**
S^1	4598.02	4596.77 ± 2.38
S^2	4701.82	4700.96 ± 0.49
S^3	4449.57	4453.58 ± 3.41

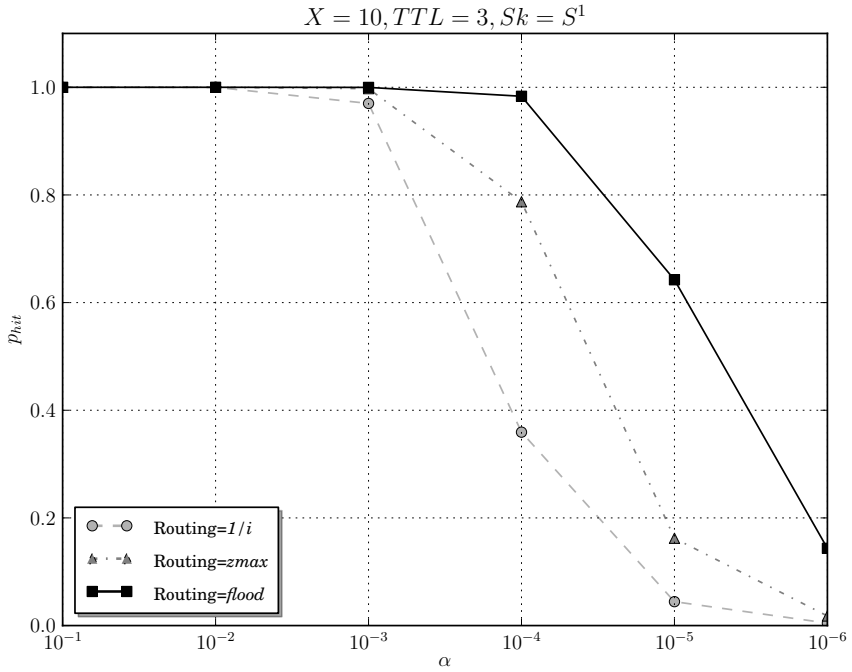
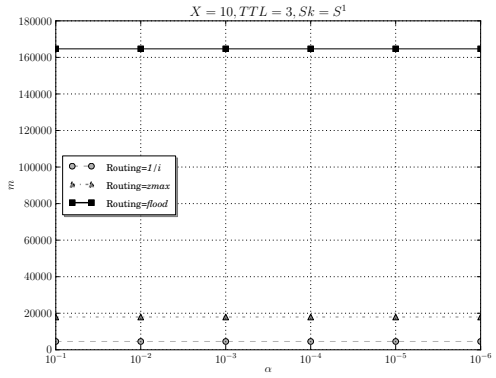


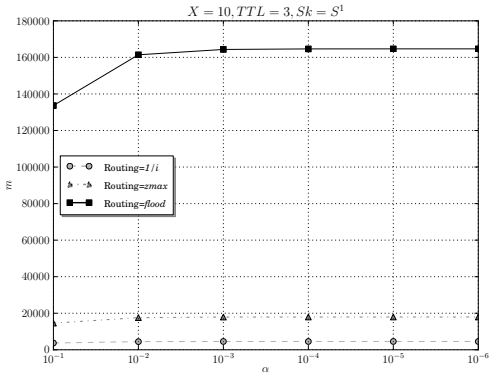
Figure 4.3: Routing policies comparison: p_{hit} for different resource popularities α

would have required days of CPU time to be completed by means of simulation since even with a coarse granularity in the definition of $\{s_i\}$ (0.05) we tested hundreds of different distributions. This analysis required only a few seconds to complete with our model.

Figures 4.5(a) and 4.5(b) show a subset of these distributions (each point in the graph corresponds to a particular distribution $\{s_i\}$). We only plotted the ones with the highest p_{hit}; it appears that the interconnection cost f alone is not directly bound to an increase in performances. There are, as a matter of fact, different configurations with $f = 3$ that perform equally (sometimes very slightly better) than those with a $f = 4$. Furthermore, within the configuration with $f = 2$ some are better than others in terms of performance and costs. Nevertheless, a clear relation exists between message cost m and p_{hit}: the larger the average number of messages the higher the p_{hit}.



(a) Query propagation for TTL hops



(b) Query propagation of Subsection 4.3.5

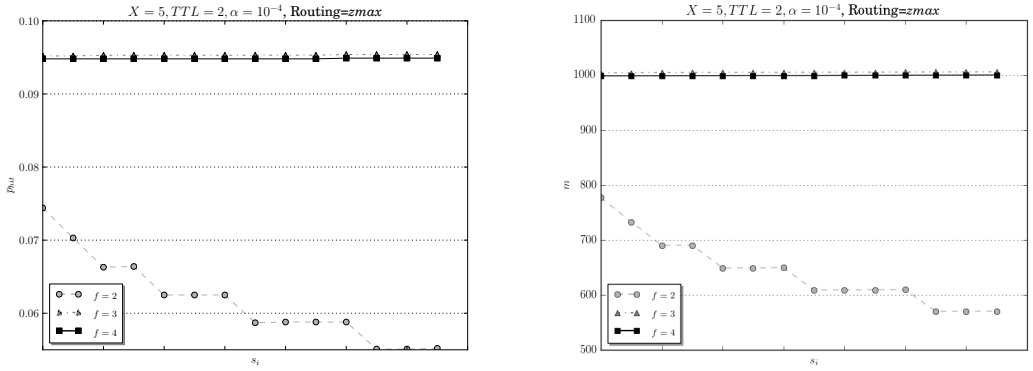
Figure 4.4: Average number of messages for different routing policies

The behavior shown in the Figures can be explained as following: the routing policy called *zmax* limits the number of messages that can be issued by a node to the value z_{max}, which is set in our evaluations to $2z$. Therefore, increasing the number connections in the interconnected system (f) beyond certain values does not lead to a significant performance increase. That is why we observe a proportionally higher increase in the p_{hit} from $f = 2$ to $f = 3$ than from $f = 3$ to $f = 4$.

4.6.3 Effects of granularity

Another aspect we analyze is a performance comparison as the number of overlays to interconnect increases. In this case we chose to analyze the behavior of the *zmax* routing policy, in a system with $TTL = 3$ and $\alpha = 0.0001$, for an increasing number of overlays (X) and for different distributions $\{s_i\}$, characterized by an increasing percentage of non-Synapse nodes s_1, while the remainder of the distribution is equally distributed across the remaining s_i.

Figures 4.6(a) and 4.6(b) show four different configurations, with an increasing number of non-Synapse nodes in the system. The parameter s_1 indicates the share of non Synapses nodes, while the remaining part $(1 - s_1)$ is equally distributed among the remaining $X - 1$ values, i.e., $s_i = \frac{1-s_1}{X-1}$ for $1 < i \leq X$. It can be noted that at each given ratio of Synapses vs non-Synapses nodes the system behavior is



(a) Hit probability p_{hit}

(b) Number of messages m

Figure 4.5: s_i comparison at different f

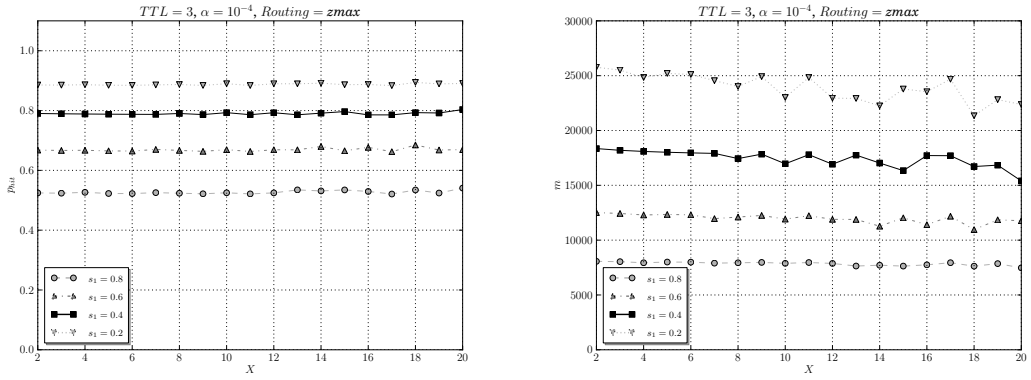
roughly the same regardless the number of overlays. The efficiency is still tightly bound to the number of messages and both increase as s_1 decreases.

4.6.4 System design with minimum requirements

Thanks to the high number of different configurations that can be evaluated with our model in a relatively short time, we conduct a further analysis to support the design of the interconnection of several peer-to-peer networks.

For instance, we set the number of overlays X and the resource popularity α; by setting a bound for the minimum desired p_{hit}, we can compare different routing policies and TTL values and find the one that minimizes the average number of messages m.

Figures 4.7 and 4.8 (left) show a classification of distributions $\{s_i\}$ for two different routing policies and two different TTL values with respect to p_{hit} and m for $X = 10$ and $\alpha = 0.0001$ (each point in the graphs represents a particular distribution $\{s_i\}$). In the first case (Figure 4.7), we decided to fix a cost factor and set $f = 4$, whereas in the second case (Figure 4.8 left), the fixed factor is the ratio of expected non-Synapse nodes in the system s_1. We are able to discriminate immediately those distributions $\{s_i\}$ that do not satisfy the imposed criteria of having $p_{hit} > 0.9$. We also discriminate among those that do the distributions $\{s_i\}$ that minimize the number of messages m, as shown in Figure 4.7(b).



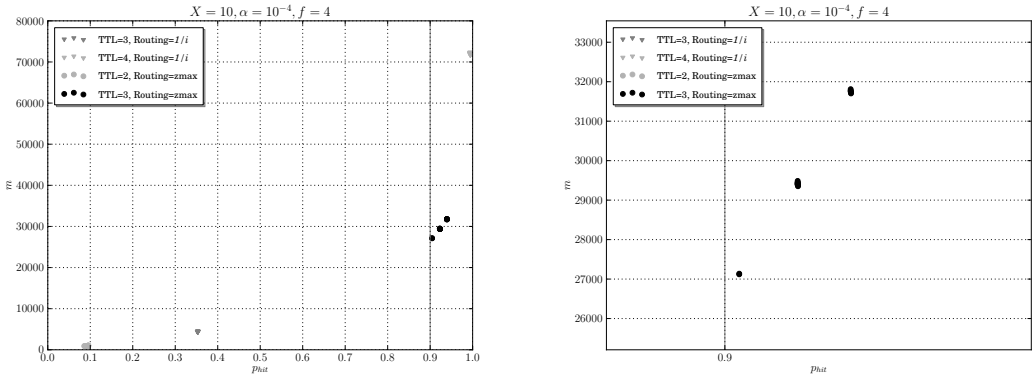
(a) p_{hit} vs. X (b) m vs. X

Figure 4.6: Performance evaluation with different numbers of overlay X

4.6.5 Routing without propagation

We briefly present some evaluation results based on the model variation presented in Subsection 4.3.5. In the first version of our model, the routing of a message is assumed to continue until the TTL expires, regardless of a resource being found or not. This leads to an $H_t(z)$ able to describe different cases, such as the probability of finding *multiple* copies of a resource. However the system is not optimal message-wise. In case we are interested only in the first hit of a search query, and we want to optimize the number of messages employed, with the variant of $R(z)$ described in Subsection 4.3.5 we are able to evaluate the system under the conditions that the routing in a node stops whenever a resource is found.

Figure 4.8 (right) shows the trend of m for different α, and two routing policies for $X = 10$, $TTL = 3$, and distribution S^1. While the number of messages was unrelated to the resource popularity before, here we see that, as routing stops upon first hit, the more popular a resource, the lower the number of messages per query.



(a) Overall view

(b) Zoomed view

Figure 4.7: Distribution of different routing policies with fixed f

4.7 Published works

The mathematical model was originally presented as a poster and short paper titled *"Modeling and analysis of large scale interconnected unstructured p2p networks"* [8].

The full paper was subsequently published as *"Interconnection of Large Scale Unstructured P2P Networks: Modeling and Analysis"* [4].

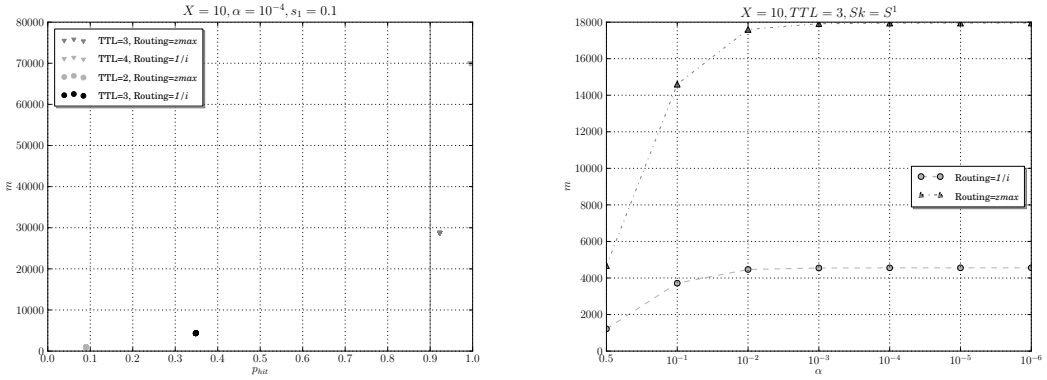


Figure 4.8: Distribution of different routing policies with fixed s_1 (left) and message evaluation at different α, for different routing policies (right)

Chapter 5

Polluter identification in MANETs

5.1 Overview

We consider the following problem: nodes in a MANET must disseminate data chunks using rateless codes but some nodes are assumed to be malicious, i.e., before transmitting a coded packet they may modify its payload. Nodes receiving corrupted coded packets are prevented from correctly decoding the original chunk. We propose SIEVE, a fully distributed technique to identify malicious nodes. SIEVE is based on special messages called *checks* that nodes periodically transmit. A check contains the list of nodes identifiers that provided coded packets of a chunk as well as a flag to signal if the chunk has been corrupted. SIEVE operates on top of an otherwise reliable architecture and it is based on the construction of a *factor graph* obtained from the collected checks on which an incremental belief propagation algorithm is run to compute the probability of a node being malicious. Analysis is carried out by detailed simulations using ns-3. We show that SIEVE is very accurate and discuss how nodes speed impacts on its accuracy. We also show SIEVE robustness under several attack scenarios and deceiving actions.

Mobile Ad-hoc Networks (MANET) are characterized by open, distributed

and dynamic architectures, built on top of the shared wireless medium; all features contribute to make MANET very vulnerable to attacks at any layer of the Internet model [24, 25].

We consider a particular type of active, non-cryptography related attack, where insider nodes corrupt data at the application level (this is also known as *pollution attack*). In this Chapter we deem as a use case a data dissemination application over a MANET. Nodes generate data chunks to be disseminated to all participants using rateless codes; some malicious nodes deliberately modify coded packets of a chunk before relaying them to prevent honest nodes from obtaining the original information.

In this Chapter we propose SIEVE a decentralized, accurate and robust technique to identify malicious nodes on top of an otherwise reliable and attacker-free architecture. Each node in SIEVE dynamically creates a bipartite graph (*factor graph*) whose vertexes are *checks* and uploading nodes. A check is a report created by a node upon decoding a data chunk; a check contains a variable length list of nodes identifiers that provided parts of the data as well as a flag to signal if the data chunk has been corrupted. Detection of the compromised chunks is achieved exploiting the constraints imposed by linear channel coding.

The factor graph is periodically and independently analyzed by each node running an incremental version of the Belief Propagation (BP) algorithm [58, 59, 60, 61]. The proposed algorithm allows each node to compute the probability of any other node being malicious; these latter probabilities are used to derive a *suspect ranking* of nodes in the MANET. Each node updates its local factor graph using the checks obtained by its own decoding operations as well as checks that are periodically gossiped by neighbor nodes.

The major contributions of the Chapter are the recasting of the problem of malicious nodes identification in terms of the estimation of the marginal probabilities on a bipartite graph and the proposal of a decentralized and accurate solution based on the BP algorithm. It is worth pointing out that the selected data dissemination application is just a quite popular use case [62, 63], whereas the proposed approach is by no means constrained to a single scenario. In particular, SIEVE can be used in any application that uses multi-party download

or collaboration, provided that is possible to detect that a given set of collaborating entities is compromised by at least one malicious node. As opposed to cryptographic/algebraic techniques proposed in the area of network coding based wireless mesh networks, e.g., [33, 34, 35, 39] SIEVE does not rely on verification tools to check the integrity of every coded block. In SIEVE the BP algorithm is used to infer the identity of the malicious nodes resting upon only on a simple pollution detection mechanism; as an example, in our reference scenario, pollution detection is achieved as a by product of the data dissemination protocol based on rateless codes. Furthermore, the aforementioned solutions may not be suitable for MANET since mobility is likely to affect key predistribution, routing mechanisms, attack behavior, etc.

SIEVE fits well two key MANETs features that must be accounted for when devising any security solution: it is fully decentralized and does not rely on any infrastructure (as opposed to some solutions in the area of peer-to-peer streaming where special well known nodes are necessary, e.g., [40]). Furthermore, SIEVE requires small computational, storage, and communication costs for implementation.

Chapter contributions are completed by a comprehensive experimental investigation of SIEVE capabilities. Our analysis is carried out by detailed simulations using ns-3; we show that SIEVE is accurate in letting each honest node identify all malicious nodes under several scenarios. We analyze the sensitivity of SIEVE performance to the nodes speed and we stress test SIEVE under several deceiving actions, colluding attacks launched by malicious nodes, and increasing number of malicious nodes. We also discuss an interesting trade-off between rateless code efficiency and SIEVE performance. The SIEVE technique has been partially presented in [64]. This Chapter includes a richer set of experimental results and a more detailed analysis of the obtained performance indexes, worked out on a larger set of system settings.

The Chapter is organized as follows: Section 5.3 describes the system model we consider, Section 5.4 presents the SIEVE technique, Section 5.5 discusses the simulation methodology and the accuracy, reactivity, and robustness results we obtained, Section 2.2 summarizes other works related to SIEVE, finally Section 7.2 draws conclusions and outlines directions for future developments.

5.2 LT codes

LT codes have been proposed in [53] and represents one of the first embodiment of the class of rateless codes: these are a particular family of erasure codes where the rate is not fixed by design, so that the number of coded packets can be decided and changed on the fly. LT codes are rateless based on the binary Galois field GF(2), i.e., coded packets are computed with simple binary XOR of random subsets of the K original data blocks. In [53] it is shown that selecting the number of blocks to be combined, termed as the packet degree d, according to the Robust Soliton Distribution (RSD)[1], one gets optimal asymptotic decoding performance. By optimal performance we mean that the so called decoding overhead, i.e. the number of coded packets to be received in excess of K, turns to be negligible for asymptotically large K. In particular, the original chunk can be obtained by any node able to collect any set of $M = K \cdot (1 + \epsilon)$ coded packets (on average), where ϵ is defined as the code overhead. The decoding algorithm can be viewed as the solution of a system of linear equations with K unknowns (the K original data blocks) and $M \geq K$ equations. In [53] it is shown that a simple method based on the recursive cancellation of equations corresponding to packets with degree 1, i.e. representing one original data blocks, guarantees the desired asymptotic performance.

5.3 A use case for SIEVE

In this Chapter we consider a MANET composed of N wireless nodes moving in a given area. A set of N_{source} nodes periodically produces a new data *chunk* to be disseminated to all others once every h seconds. All nodes cooperate to the diffusion of the data chunks by running a distributed dissemination algorithm based on *Luby Transform* (LT) codes [53]. Data is transmitted by source nodes using LT codes [53]: a chunk (whose size in bytes is fixed and is denoted as S) is divided in K equally sized blocks. The source node then creates and forwards

[1] RSD is derived by the Ideal Soliton Distribution and depends on two free parameters usually identified with constants c and δ.

coded packets using LT codes [53], combining random subsets of the K blocks; the size of each coded packet is $S_{cb} = \frac{S}{K}$.

5.3.1 LT based dissemination protocol

The rateless principle and randomness of LT codes (see Section 5.2) is used for spreading data in the MANET by letting source nodes transmit novel coded packets that can be generated randomly and on the fly. A coded packet conveys the XORed payloads of the corresponding original packets as well as a header signaling the indexes of the combined packets. The original chunk can be obtained by any node able to collect any set of $K \cdot (1 + \epsilon)$ coded packets without requiring any coordination among the source nodes. In turn, whenever a node is able to successfully decode a chunk, it can behave as a new source for the chunk, generating and disseminating novel coded blocks.

In Figure 5.1 the algorithmic steps of the LT based dissemination protocol implemented by a node are graphically sketched. Each node can simultaneously collect coded packets for different chunks; to this end a window based mechanism is used where the w_r most recent chunks can be concurrently downloaded and progressively decoded. As soon as a given chunk is decoded (this event happens on average when $K \cdot (1 + \epsilon)$ coded blocks of the chunk have been buffered), the corresponding data blocks are moved into a buffer storing the w_t data chunks that have been decoded most recently. A simple round robin scheduling policy is used on the transmitter side, where one of the data chunk is selected, a novel LT coded packet is encoded and transmitted. A new coded packet is transmitted every T_{tx} ms using UDP over an 802.11g wireless communication interface yielding an average transmission range of r meters.

5.3.2 Malicious nodes

The proposed dissemination protocol is an example of a distributed and collaborative approach that has the potential to simplify and accelerate the spreading of the information in the MANET thanks to node mobility. On the other hand, few malicious nodes may try to break the system by polluting, i.e. modifying some coded packets. In this Chapter we assume that a subset of P nodes, with

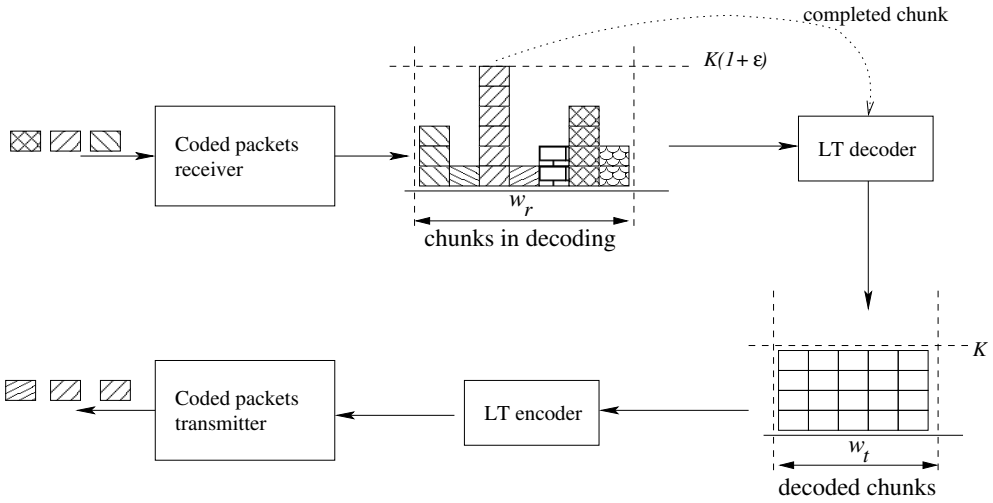


Figure 5.1: Node operations: LT encoding, decoding and dissemination protocol

$P < N$, is composed of *malicious nodes*, that deliberately modify the payload of the coded packets to prevent honest nodes from correctly reconstructing the original chunk. In the presence of coding even a single corrupted coded packet can prevent an honest node from decoding the original chunk.

5.4 The SIEVE protocol

SIEVE uses LT codes decoding mechanism to detect modified chunks and exploits the *Belief Propagation* (BP) algorithm [58] to identify malicious nodes.

5.4.1 LT codes verification mechanism

According to the dissemination strategy described in Section 5.3 every node keeps collecting from different uploaders sets of coded blocks corresponding to different chunks. LT codes can be exploited to detect if modified blocks have been collected without the need of any supplementary verification mechanism.

Indeed, a node is able to detect pollution as soon as an inconsistency is found in the solution of the underlying system of linear equations. In particular, according to the procedure [53] the decoder keeps canceling out all the already known

data packets. This is achieved by observing that a coded packet with a degree 1 equation represents a data packet in the clear. Such data packet can be simplified from all the incoming equations. Since LT codes have a certain overhead some coded packets that are linearly dependent on the ones received previously are always collected before successful decoding; this amount to the reception of some equations whose terms are all already known. As soon as this condition is met the LT decoder can check the consistency of the payload carried by the coded packet; in other words, the same linear combination must be obtained combining a set of already known packets. If this constraint is violated the whole chunk is recognized as corrupted. Please note that the receiver node is not able to identify the corrupted block(s) but only that at least one of them has been maliciously manipulated.

5.4.2 Check construction and reporting

SIEVE is based on the concept of *checks* that are reports created by nodes upon decoding a chunk. A check contains the list of the identifiers of nodes that provided coded blocks of a chunk and a flag to label such chunk as corrupted or not. A check describing a corrupted chunk is called a *positive check* while it is termed a *negative check* otherwise. Each nodes n maintains a list of all checks created that is denoted as $\mathcal{L}_n$.

Each node, besides accumulating the checks from its local decoding operations, gossips them in the neighborhood. Each node n in the MANET transmits its checks in two cases:

- as soon as n decodes a chunk it inserts it in $\mathcal{L}_n$ and broadcasts it;
- once every T_s seconds n randomly selects Q checks in $\mathcal{L}_n$ and transmits them.

5.4.3 Identification based on belief propagation

The checks in $\mathcal{L}_n$ and all checks received by n are used to build a *factor graph* $\mathcal{G}_n$=$(\mathcal{U}, \mathcal{C}, \mathcal{E})$. $\mathcal{G}_n$ is a bipartite graph where the vertex set $\mathcal{U}$ is the set of uploader

nodes, the vertex set $\mathcal{C}$ is the set of checks, and an undirected edge $\{i, I\} \in \mathcal{E}$ exists if and only if check $I \in \mathcal{C}$ depends on uploader $i \in \mathcal{U}$.

In the following we will refer to the set of uploaders involved in check I as $\mathcal{U}_I$ and the set of checks that node i contributes as an uploader as $\mathcal{C}_i$. An example of factor graph with four uploaders (circles) and two checks (squares) is show in Figure 5.2. The factor graph can be progressively created while decoding the chunks. In Figure 5.2 we assume that a node has decoded two chunks corresponding to two checks in its local factor graph. The filled square represents a *positive check* due to the detection of a modified chunk concurrently downloaded by three nodes (in the Figure we assume that the rightmost node is malicious). The remaining check is a negative one, appended when a chunk has been decoded successfully. From the point of view of any decoding node, each uploader i can be in one of two hidden states $x_i = 1$ or $x_i = 0$, depending on whether uploader i *is* or *is not* a malicious node. Each check can report one of two observations $c_I = 0$ or $c_I = 1$ in case of negative or positive pollution detection, respectively.

The problem of identifying the malicious nodes from a given number of checks can be recast as an inference problem. The goal of the inference is the estimation of the hidden state of the nodes, i.e. being malicious or not, given a set of observations corresponding to the checks. Each check can be interpreted as an accusation raised against a set of uploader nodes by a witness node. In this Chapter we adopt the BP algorithm [58, 59, 60, 61], that has been used to solve a number of inference problems in many different fields, e.g., iterative channel decoding [65], Bayesian networks [58] and computer vision [66] to mention a few.

The BP algorithm can be used to estimate from the factor graph the so called variable marginals $P(x_i)_{i\in\mathcal{U}}$, i.e. the probability of node i being malicious. BP is an iterative algorithm based on the exchange of probability estimates (also called messages or beliefs), along the edges of the bipartite graph $\mathcal{G}_n$. In case of a Bayesian network BP represents a closed-form solution for the marginals. Nonetheless, the same algorithm has proven to be a robust estimator for the variable marginals of general factor graph [60, 61].

In our setting it is convenient to distinguish between two classes of messages: message from uploader i to check I, m_{iI}^x, that is meant to be the probability that uploader i is in state x, given the information collected via checks other

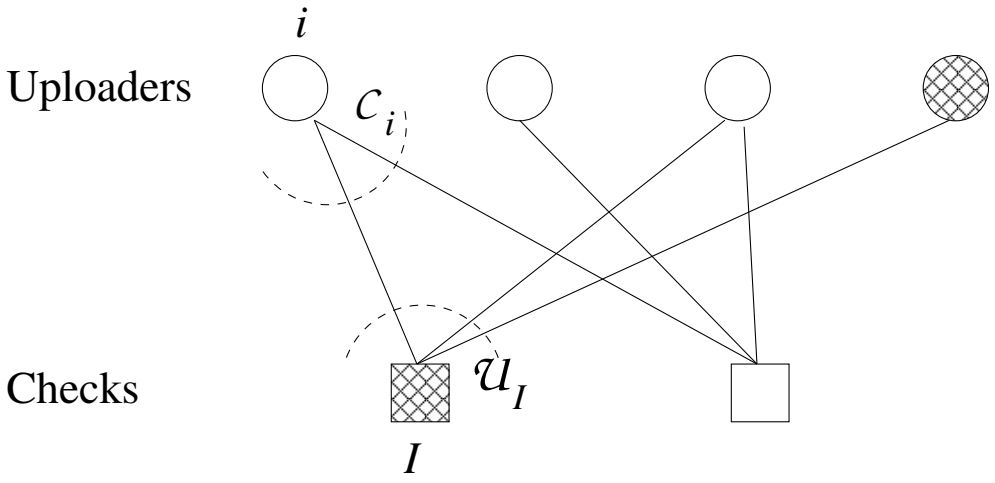


Figure 5.2: Example of factor graph

than check I ($\mathcal{C}_i \setminus I$); message from check I to uploader i, m_{Ii}^x is defined as the probability of check I having value c_I if uploader i is considered in state x and all the other uploader states have a separable distribution given by the probabilities $\{m_{i'I}^x : i' \in \mathcal{U}_I \setminus i\}$.

The BP algorithm is based on iterative refinements of the check messages m_{Ii}^x based on the current values of the node messages m_{iI}^x (check pass), followed by updating of m_{iI}^x as a function of m_{Ii}^x (node pass). For the first check pass the messages are initialized to the values $m_{iI}^0 = m_{iI}^1 = 0.5$, since we assume that no prior information on the number and identity of the malicious nodes is available.

The check pass is based on the estimation of the probabilities m_{Ii}^x as follows:

$$m_{Ii}^x = \sum_{\{x_{i'}: i' \in \mathcal{U}_I \setminus i\}} P\left(c_I | x_i = x, \{x_i' : i' \in \mathcal{U}_I \setminus i\}\right) \prod_{i' \in \mathcal{U}_I \setminus i} m_{i'I}^{x_i'} \tag{5.1}$$

Equation (5.1) depends on the probability of observing a certain check value c_I, given the states of the uploaders of such check.

Given that a check turns out to be positive as soon as at least one of the uploaders is a malicious node, we can write:

$$P\left(c_I = 1 | \{x_i : i = 1, \ldots, k\}\right) = \begin{cases} 0, & \text{if } x_i = 0, \forall i \\ 1, & \text{otherwise} \end{cases} \tag{5.2}$$

Analogously, observing that a check can be negative if and only if all the uploaders are not malicious, we get

$$P\left(c_I = 0 | \{x_i : i = 1, \ldots, k\}\right) = \begin{cases} 1, & \text{if } x_i = 0, \forall i \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

Plugging the last two expressions into Equation (5.1) we can simplify the check pass computation as follows:

$$m_{Ii}^x = \begin{cases} \prod_{i' \in \mathcal{U}_I \setminus i} m_{i'I}^0 & \text{if } c_I = 0, x = 0 \\ 0 & \text{if } c_I = 0, x = 1 \\ 1 - \prod_{i' \in \mathcal{U}_I \setminus i} m_{i'I}^0 & \text{if } c_I = 1, x = 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.4}$$

The next BP step is constituted by the updating of the probabilities m_{iI}^x, using information from previous computation in the checks (node pass), according to (5.5):

$$m_{iI}^x = \prod_{I' \in \mathcal{C}_i \setminus I} m_{I'i}^x \tag{5.5}$$

After any node pass it is possible to get an estimate of the marginal $P(x_i = x)$ according to:

$$P(x_i = x) = \prod_{I' \in \mathcal{C}_i} m_{I'i}^x \tag{5.6}$$

We recall that the probability estimates given by Equation (5.5) and (5.6) are not guaranteed to be normalized; in the practical implementation we use proper normalization constants to avoid numerical issues as suggested in [59].

To conclude, the BP algorithm initializes the values of m_{iI}^x, then keeps iterating using Equations (5.4) and (5.5). A reliable estimate of $P(x_i = x)$ is computed after a certain number of such iterations using Equation (5.6). In all the experiments reported in this Chapter 3 iterations have been used to strike a balance between estimation accuracy and complexity.

5.4.4 BP complexity

The computational cost of single message update using Equation (5.4) amounts to $|\mathcal{U}_I| - 1$ multiplications, where operator $|\cdot|$ evaluates the cardinality of a set. Since messages are associated to each edge of the bipartite graph we can conclude that the overall check pass takes on average $|\mathcal{E}|(z_{\mathcal{U}} - 1)$ multiplications, $z_{\mathcal{U}}$ being the average number of uploaders per check.

Using the same reasoning from (5.5) one can compute the cost of the node pass as $|\mathcal{E}|(z_{\mathcal{C}} - 1)$ multiplications, where $z_{\mathcal{C}}$ is the average number of checks per node. Analogously, the final estimate in (5.6) takes $|\mathcal{E}|z_{\mathcal{C}}$ multiplications.

Finally, taking into accounts all the BP steps and assuming 3 iterations the overall computational complexity amounts to $3(|\mathcal{E}|(z_{\mathcal{U}} - 1) + |\mathcal{E}|(z_{\mathcal{C}} - 1)) + |\mathcal{E}|z_{\mathcal{C}}$ multiplications.

5.4.5 Incremental BP estimation

In the previous description of the BP we have assumed that the factor graph $\mathcal{G}_n$ is known in advance and kept fixed for all the iterations. In practice this assumption is not met in the proposed scenario. Nonetheless, the proposed algorithm can be implemented using an incremental (or sliding window) approach as follows.

Each node n keeps receiving checks from the other ones and it is allowed to run the BP on $\mathcal{G}_n$ every T seconds considering only the checks received and created in a time window of the past w seconds. At time t, depending on the checks stored during a time window w, an updated factor graph $\mathcal{G}_{n,t,w}$ is obtained by removing the old checks and adding the new ones; then, the corresponding estimates $P_{t,w}(x_i = x)$ are computed through BP. The belief values m_{iI}^x are initialized to 0.5 when the i-th node is met for the first time; then, the partial estimates of m_{iI}^x are stored in memory and propagated to all the iterations and computation windows where such values are needed.

After every BP estimation of $P(x_i = x)$, a list of suspect nodes is obtained by setting a threshold on the probability $P_{t,w}(x_i = 1) \geq \eta$. Each node n keeps a counter for each of its uploader nodes i: the nodes in the list of suspects after the BP run have their counter increased by 1. Finally, a *suspects ranking* $\mathcal{R}_n$ over uploader nodes is defined by sorting their counters in decreasing order. As an

example, the first node in the suspects ranking at time t is the uploader that more often has been included in the list of suspects after all the BP runs performed up to time t.

5.5 Results

In this Section we describe the simulation methodology and the indexes we defined to evaluate the accuracy, reactivity, and robustness of SIEVE. In particular, we show how the structural properties of the factor graph $\mathcal{G}_n$ and mobility impact the performance of SIEVE as well as how robust it is with respect to several deceiving actions operated by malicious nodes. Finally, we show a trade-off between coding efficiency and SIEVE accuracy.

5.5.1 Factor graph and performance

According to [67] when short cycles are absent in the factor graph $\mathcal{G}_n$ it is best to have high degree nodes (i.e., uploaders that contribute to many checks) and low degree checks (i.e., checks obtained from a small number of uploaders). High degree nodes (both malicious and honest) tend to their correct marginal probability $P(x_i)_{i\in\mathcal{U}}$ (i.e. the probability of node i being malicious) quickly; on the other hand, the lower the number of uploaders in a check the more valuable the information it conveys.

Unfortunately, the actual factor graphs computed by nodes do contain cycles and it is well known that cycles in the factor graph negatively impact on the accuracy of the probability estimates yielded by the BP algorithm [67]. The shortest length of a cycle in a bipartite graph with at most one edge between any two nodes is 4 and is due to the presence of at least two common uploaders in a pair of checks, i.e. the cardinality of the intersection between the uploaders of a pair of checks is at least 2. If the cardinality of the intersection is $x > 2$, then each of the $\binom{x}{2}$ pairs defines one length 4 cycle. Equivalently, a cycle forms if any two nodes contributed to provide blocks of at least two common checks. It is easy to note that the higher the degree of nodes and/or checks the higher the number of cycles in the factor graph.

In this context, the benefits of high degree nodes and low degree checks are counter-balanced by the negative effects of cycles. Indeed, increasing the average number of checks per node improves the performance of the BP algorithm up to a point where the number of cycles increases too much and lowers the accuracy of the BP algorithm. Similarly, the benefits of a lower average check degree are offset by a high number of cycles.

5.5.2 Simulation methodology

The performance of SIEVE have been investigated by simulations using ns-3 version 3.12 [68] on a Red Hat 4.4.6-3 machine using standard variable settings. In particular, we developed a node object implementing the sender and receiver functionalities according to the protocol described in Sections 5.3 and 5.4. The MANET is composed of N wireless nodes placed in a square area whose side length is l meters. Nodes move using different average speeds. N_{fast} nodes moving at V_{fast} m/s (e.g., cars, buses, motorcycles), N_{slow} nodes at V_{slow} m/s (e.g., pedestrian, bikes), and N_{still} fixed nodes (e.g., sensors, relay stations, shops). As a consequence we have $N = N_{fast} + N_{slow} + N_{still}$. All non-fixed nodes movements are described by the same mobility model and moving nodes are randomly distributed across the simulation area at the beginning of the experiment. Still nodes are distributed on a grid to avoid clustering in a particular region. We used the ns-3 default transmission model composed of a propagation delay model (a constant value model) and a propagation loss model (based on a log distance model with a reference loss of 46.677 dB at a distance of 1 meter). Under these settings we derived the maximal transmission range r as 175 meters.

We conducted simulation experiments that terminate after 1 hour of simulated time. We used the independent replication method to obtain $N_{EXP} = 30$ executions and computed 95% confidence intervals. Each execution is obtained with different streams of the random number generator provided by ns-3. The simulation output is made of a log file containing all checks received by each node that is post-processed by our analysis software to run the SIEVE protocol to detect malicious nodes.

All results have been obtained by setting as initial data sources only the N_{still}

Parameter	Description	Value
N_{fast}, N_{slow}	Number of fast/slow nodes	100
N_{still}	Number of still nodes	50
P_{fast}, P_{slow}	Number of fast/slow polluters	5
P_{still}	Number of still polluters	0
K	Number of blocks	100
S_{cb}	Size of blocks	500 Bytes
T_{tx}	Time between transmissions of blocks	100 ms
c, δ	RSD parameters for LT encoder	0.01
l	Square area side length	2000 m
h	Time between data chunk generation	600 s
w_r	Window size for concurrent chunk downloading	50
w_t	Window size for concurrent chunk uploading	6
r	Maximal transmission range	175 m
V_{fast}	Fast nodes speed	[10-40] m/s
V_{slow}	Slow nodes speed	[1-5] m/s
-	Mobility model	2D RW

Table 5.1: Parameter values for the reference scenario

still nodes; all N nodes contribute to the spreading of the information according to the policy described in Section 5.3. A subset of nodes ($P = P_{fast} + P_{slow} + P_{still} < N$) are simulated as malicious nodes that modify the coded packets. All the system parameter values of the reference scenario considered in the following are summarized in Table 5.1.

5.5.3 Performance indexes

For each node n in the system we define:

- $p(n)$ an indicator function whose value is equal to 1 if node n is not malicious and 0 otherwise;
- $c_n(t)$ an indicator function whose value is equal to 1 if $\mathcal{R}_n$ is non empty at time t and 0 otherwise;
- $a_n(t)$ the number of actual malicious nodes correctly identified by node n at time t. More precisely, $a_n(t) = x$ if x nodes in the top P positions of $\mathcal{R}_n$ are truly malicious;
- $r_n(t)$ the number of actual malicious nodes correctly identified in the top positions of $\mathcal{R}_n$ at time t. More precisely, $r_n(t) = x$ if *top* x nodes in $\mathcal{R}_n$ are all actually malicious. If SIEVE at time t has identified $P-1$ out of P

malicious nodes ($a_n(t) = P - 1$) but the first node in $\mathcal{R}_n$ is not malicious then $r_n(t) = 0$;

- $t_{n,trigger}$ the time node n received the very first positive check that triggered the SIEVE activation;

- $t_{n,hit}(x) = min_t\{t : r_n(t) \geq x\}$, that is, the minimum time to identify at least x malicious nodes in the top x positions of $\mathcal{R}_n$.

In each independent replication trial we compute the following averages:

- $c^{(i)}(t) = \frac{\sum_{n=1}^{N} p(n)c_n(t)}{\sum_{n=1}^{N} p(n)}$, i.e., the fraction of honest nodes that detected the attack and have a non empty $\mathcal{R}$;

- $a^{(i)}(t) = \frac{1}{P}\frac{\sum_{n=1}^{N} p(n)c_n(t)a_n(t)}{\sum_{n=1}^{N} p(n)c_n(t)}$, i.e., the average accuracy of honest nodes with non empty $\mathcal{R}$. When $a^{(i)}(t) = 1$ it means that *all* honest nodes with non empty $\mathcal{R}$ have correctly identified *all* malicious nodes;

- $tsi^{(i)}(x) = \frac{\sum_{n=1}^{N} p(n)(t_{n,hit}(x) - t_{n,trigger})}{\sum_{n=1}^{N} p(n)}$, i.e., the average time honest nodes require to unambiguously identify x malicious nodes.

Finally, we computed averages and 95% confidence intervals over N_{EXP} executions and considered $c(t)$, $a(t)$, and $tsi(x)$ that we call completeness, accuracy, and time to identification, respectively. In all computations we set $\eta = 0.99$ (η is the threshold on the probability $P_{t,w}(x_i = 1)$ for suspect identification) and iterated each run of the BP algorithm on a specific factor graph three times.

5.5.4 Sensitivity results

The first analysis we conducted is on the sensitivity of $c(t), a(t)$, and $tsi(x)$ to parameters w, T, T_s, and Q used by SIEVE as defined in Section 5.4. To this end we considered integer multiples of 10s values for w ranging from 10s to 90s, $T = \{5, 10\}$s, $T_s = \{5, 10, 15\}$s, and $Q = \{5, 10\}$. We analyzed all 108 combinations and observed that SIEVE performance is most sensitive to the value w.

Figure 5.3 shows the performance indexes $a(t)$, $c(t)$, and $tsi(x)$ for some values of w, when $T = T_s = 10$ s and $Q = 10$ (all 108 configurations yielded similar results so we selected a representative set of curves). It can be noted that too

w	$\lvert\mathcal{C}\rvert$	$z_{\mathcal{U}}$	$\lvert\mathcal{U}\rvert$	$z_{\mathcal{C}}$	length 4 cycles
10	83.1688	8.53656	173.7	3.8633	1175.57
30	247.617	8.55307	216.824	9.29914	8899.69
60	492.807	8.5262	231.687	17.3395	32021.6
90	735.944	8.49615	236.444	25.3386	68064.4

Table 5.2: Structural properties of $\mathcal{G}_n$ for increasing values of w

small or too large values for w yield the worst performance for all the performance indexes. Indeed, small window sizes do not allow the factor graph to include enough checks and nodes to accurately infer the node status; on the other hand, large values of w provide more checks and nodes in $\mathcal{G}_n$ but increase the number of cycles in the factor graph which in turn impact on the accuracy of the probability estimates yielded by the BP algorithm, as discussed in Subsection 5.5.1.

We summarized the structural properties of $\mathcal{G}_n$ in Table 5.2. It can be noted that the average number of nodes per check $z_{\mathcal{U}}$ is not dependent on w (it mostly depends on the overall number of nodes, the mobility patterns, and the data exchange protocols). On the contrary, the size of $\mathcal{G}_n$ ($\lvert\mathcal{C}\rvert$ and $\lvert\mathcal{U}\rvert$) increases as w increases although the number of nodes saturates since it is upper bounded by N. Accordingly, the average number of checks per node $z_{\mathcal{C}}$ increases and it positively impacts on SIEVE accuracy up to the value $w = 60$; for larger values the average number of cycles in the factor graph increases and nullifies the increase of $z_{\mathcal{C}}$.

It can also be noted that $c(t)$ approaches 1 after about 4 minutes; the behavior of $c(t)$ clearly depends on the delay after which a node has collected a sufficient amount of checks to start its own suspect ranking. As for SIEVE reactivity, we observe that $w = 60$s yields the lowest values for the time to identification $tsi(x)$ (all graphs for $tsi(x)$ are computed for the maximum x such that for each honest node n it is true at the end of the simulation experiment that $r_n = x$). Based on the above discussion on the impact of w on the performance of SIEVE, we selected $w = 60$s as the value for all the following experiments.

Figure 5.4 shows a comparison among different choices of the values for parameters T, T_s, and Q (please recall that each node executes a BP estimation once every T seconds and transmits a selection of Q checks once every T_s sec-

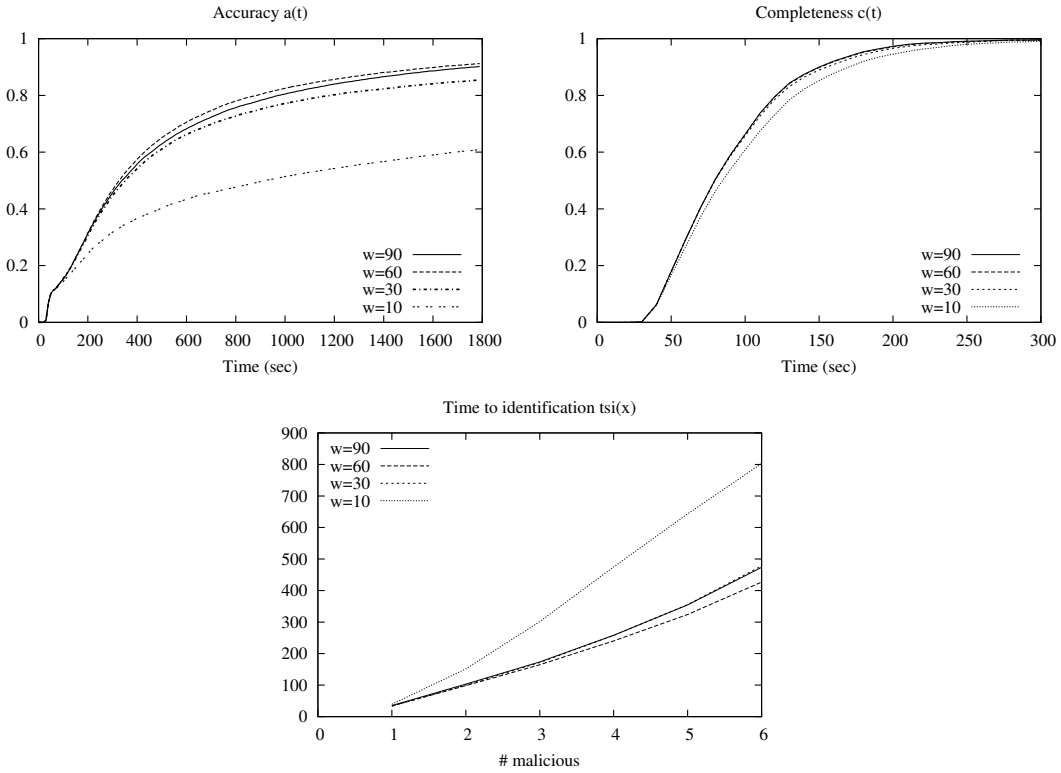


Figure 5.3: Representative accuracy (left), completeness (middle), and time to identification (right) of SIEVE as a function of w

onds). In particular, we selected three settings for three different values of the number of checks transmitted per seconds ($\frac{Q}{T_s} = 2, 1, 0.33$). It can be noted that performance of SIEVE achieve their best when the BP estimation is run more frequently ($T = 5$s) and the the number of checks transmitted per seconds is the highest. Nevertheless, improved performance come at the cost of increased computation and communication burden on nodes; this may cause concerns for battery operated nodes whose lifetime could be shortened. Based on the above reasoning, we selected a compromise between performance and energy consumption awareness using the following setting for the rest of the work: $w = 60$s, $T = T_s = 10$s, $Q = 10$.

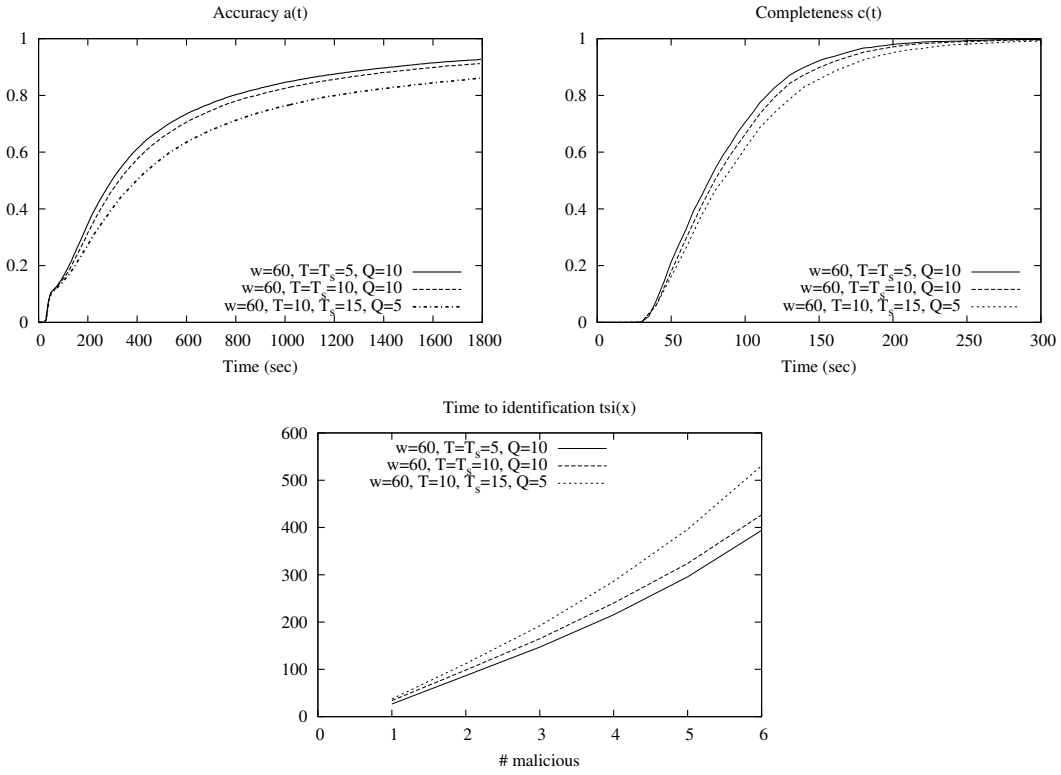


Figure 5.4: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for best performance, lower energy consumption, and compromise setting

5.5.5 Mobility and SIEVE performance

The first interesting observations we made is that mobility affects SIEVE performance. In particular, the performance depends not only on the speed but also on the mobility model adopted for nodes.

Figure 5.5 shows the performance of SIEVE in the reference scenario with different mobility models available in ns-3, i.e. 2D random walk, random direction, Gauss-Markov and steady state random waypoint; clearly, all mobility models have been compared for the same node speeds. It can be noted that the 2D random walk mobility model selected for the reference scenario yields slightly worse performance with respect to other mobility models that more closely represent mobility in a urban environment.

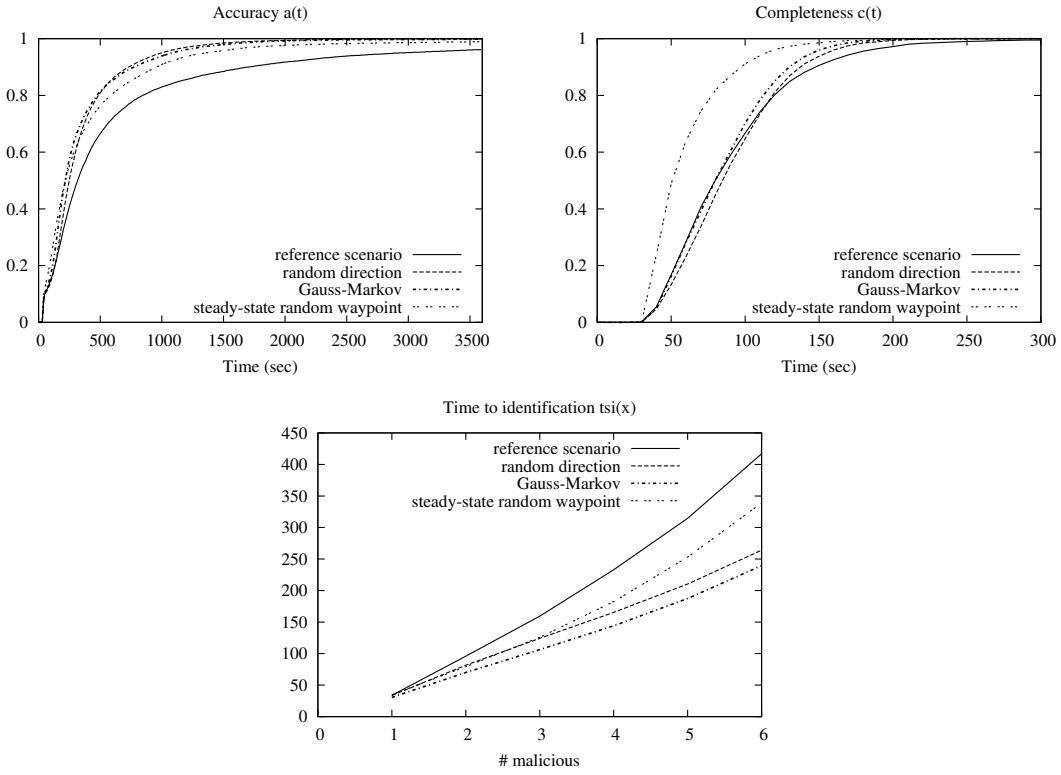


Figure 5.5: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different nodes mobility models

Figure 5.6 shows the performance of SIEVE in the reference scenario but averaged over nodes in the same speed class (fast, slow, still). It can be noted that accuracy of fast nodes is higher with respect to slow and still nodes. Reaction times are also lower for fast moving nodes.

Figure 5.7 shows the overall performance of SIEVE in the reference scenario and in scenarios with different mixes of fast and slow nodes. The system where all moving nodes are fast yields much higher accuracy and much lower reaction times with respect to other extreme case where all nodes move slowly. Nevertheless, SIEVE accuracy is 0.83 at the end of the one hour long experiments and approaches 1 for longer runs.

Figure 5.8 shows results in the reference scenario and in scenarios where all malicious nodes are either fast or slow. This results show that malicious nodes can

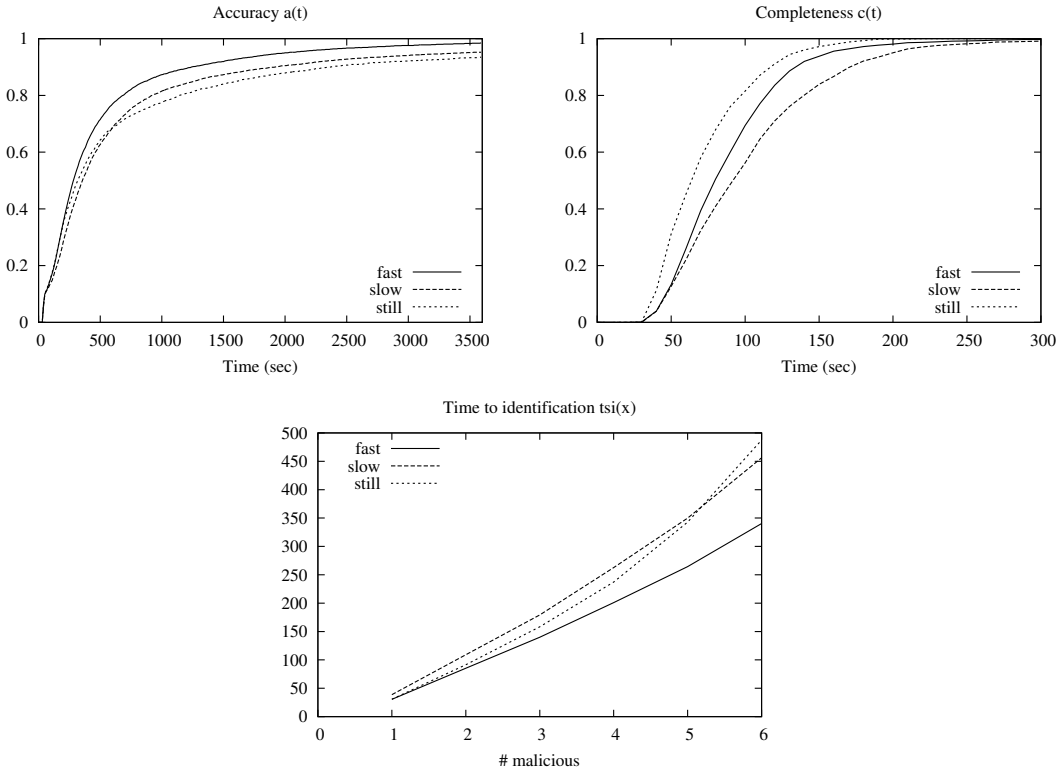


Figure 5.6: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different nodes speeds in the reference scenario

delay their identification if they move slowly; on the other hand if all malicious nodes move fast all of them are quickly identified.

Although SIEVE is able to identify all malicious nodes in the long run in any setting, in all three cases we can conclude that *high speed is key to obtain both high accuracy and low reaction delays by honest nodes.*

Why is higher speed beneficial for SIEVE performance of honest nodes but detrimental for malicious nodes? Consider the case where different mixes of fast and slow nodes (Figure 5.7) are compared and assume an extreme scenario composed of all still nodes. Since nodes do not move it is possible to define a static geometric graph $\mathcal{O}$ describing connections among nodes where vertexes are nodes and an undirected edge between two vertexes exists if the corresponding nodes are within the transmission radio range r. For any node $n \in \mathcal{O}$ let $\mathcal{N}(n)$

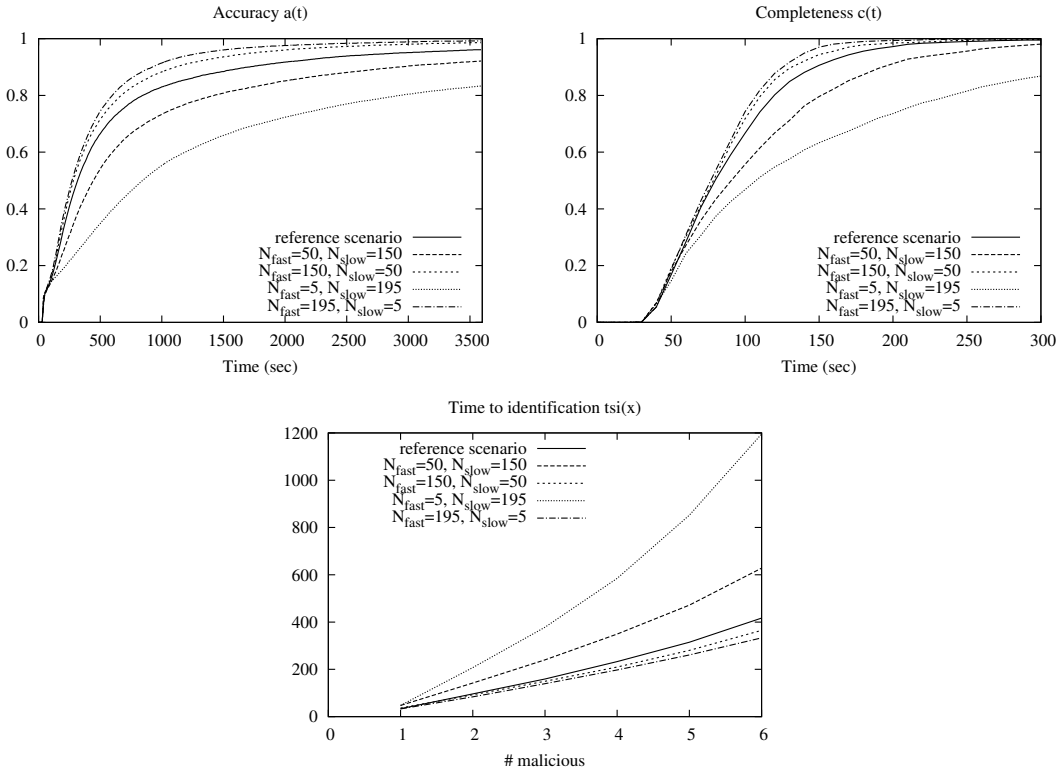


Figure 5.7: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE where moving nodes range from all fast to all slow

denote the set of nodes connected to n (the neighborhood of n).

Consider one data source s and a chunk c; clearly, when s has transmitted $K \cdot (1 + \epsilon)$ coded packets, c is decoded by all nodes in $\mathcal{N}(s)$; in turn, these nodes start transmitting fresh coded packets for c and, if $\mathcal{O}$ is connected, all nodes will be able to decode c after some time. It can be noted that all coded packets produced by s will follow the same paths in $\mathcal{O}$. This means that for any node $n \in \mathcal{O}$ chunks originating from s will be provided to n by *the same set of uploaders* $\mathcal{U}_s(n)$ that is a subset of $\mathcal{N}(n)$. Of course, the same reasoning is valid for all checks describing chunks produced by any data source other than s. The final effect is that the set of checks created by each node n upon decoding chunks ($\mathcal{L}_n$) is such that strong intersections exist among the uploaders of different checks (given that $\mathcal{O}$ is static and $\mathcal{N}(n)$ does not change over time); this translates into

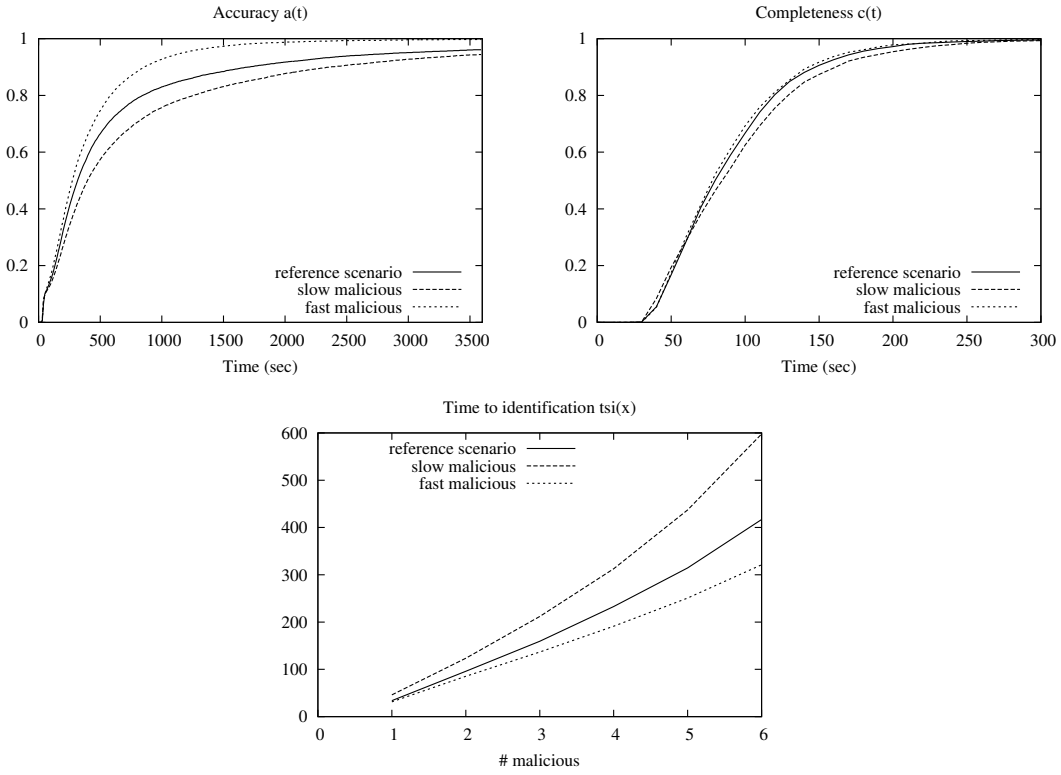


Figure 5.8: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE where malicious nodes are either all fast or all slow

a high number of short cycles in the factor graph used by SIEVE that is a well known cause of poor performance of the BP algorithm, as discussed in Subsection 5.5.1.

Instead, when nodes move the geometric graph $\mathcal{O}$ becomes dynamic, i.e., for any node n its $\mathcal{N}(n)$ varies with time. Speed translates into higher rate of changes in $\mathcal{N}(n)$ that, in turn, reduces the number intersections among the uploaders of different checks (hence of short cycles in $\mathcal{G}_n$) and increases the accuracy of BP algorithm. For the reference scenario we computed the structural properties of $\mathcal{G}_n$ that are summarized in Table 5.3. It can be noted that the average number of nodes per check ($z_{\mathcal{U}}$) and the average number of checks per node ($z_{\mathcal{C}}$) for still nodes would suggest better performance with respect to fast and slow nodes. Nevertheless, the average number of length four cycles is *much* higher for still

speed	$\lvert\mathcal{C}\rvert$	$z_{\mathcal{U}}$	$\lvert\mathcal{U}\rvert$	$z_{\mathcal{C}}$	length 4 cycles
fast	434.14	8.6891	232.594	15.8041	23240.5
slow	428.937	8.91507	230.292	15.8437	29242.7
still	737.88	7.42264	232.663	23.4016	55141.4

Table 5.3: Structural properties of $\mathcal{G}_n$ for different nodes speed in the reference scenario

nodes and this is the main reason for worse performance.

5.5.6 Deceiving actions and SIEVE robustness

Besides following the data dissemination and SIEVE protocols, malicious nodes may also implement disturbing actions aiming at preventing or delaying their identification. In the following a number of deceiving actions are investigated.

- Reduced pollution intensity at the coded packet level: a malicious node modifies a coded packet with probability p_{poll} (the reference scenario is analyzed with $p_{poll} = 1$). Figure 5.9 shows how SIEVE performs when nodes lower their pollution intensity to $p_{poll} = 0.5, 0.25, 0.125$ in the attempt of making their identification harder. It can be noted that SIEVE is able to spot malicious nodes in all cases although very low pollution intensities delay identification. Nevertheless, it is worth pointing out that tuning the pollution intensity yields a natural trade-off between speed of identification and average damage caused to honest nodes, i.e., the lower p_{poll} the lower the number of polluted chunks and the longer the time to identification.

- Reduced pollution intensity at the chunk level: a malicious node pollutes a chunk with probability c_{poll} and the corresponding coded packets are modified with probability p_{poll}. In Figure 5.10 we show results for $c_{poll} = 0.75, 0.5$ and $p_{poll} = 1$, comparing them to the reference scenario (analyzed with $c_{poll} = p_{poll} = 1$). Also in this case, we observe that malicious nodes are identified by SIEVE albeit at the price of some delay for low values of c_{poll}.

- Check status falsification: with probability p_{lie} a positive check obtained by a malicious node is forwarded as negative and viceversa. Figure 5.11 shows

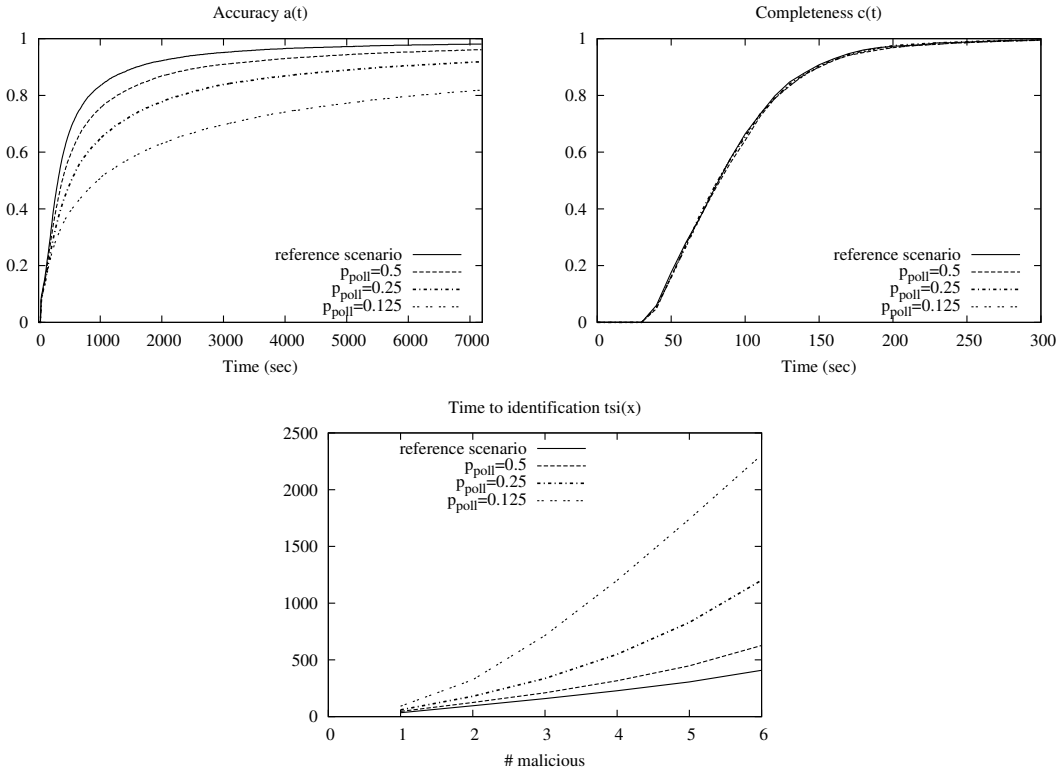


Figure 5.9: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different pollution intensities at the coded block level (p_{poll})

that this trick is ineffective: indeed, the only effect is a limited increase in the values of $tsi(x)$. Figure 5.12 also shows that SIEVE is able to achieve high accuracy even in extreme cases when half of the moving nodes is malicious. Indeed, provided that the system operates for long enough time, SIEVE is able to allow every node to identify all existing malicious nodes.

- Honest nodes disparaging: malicious nodes always produce dummy positive checks involving a set of honest nodes. In this case malicious nodes flip a coin and, with probability $p_{disparage}$, they replace the actual uploaders of a check with a set of honest nodes; the detection flag is marked as positive. Figure 5.13 shows SIEVE performance when $p_{disparage} = 1$ and honest nodes are either randomly chosen or chosen in the same fixed order followed by

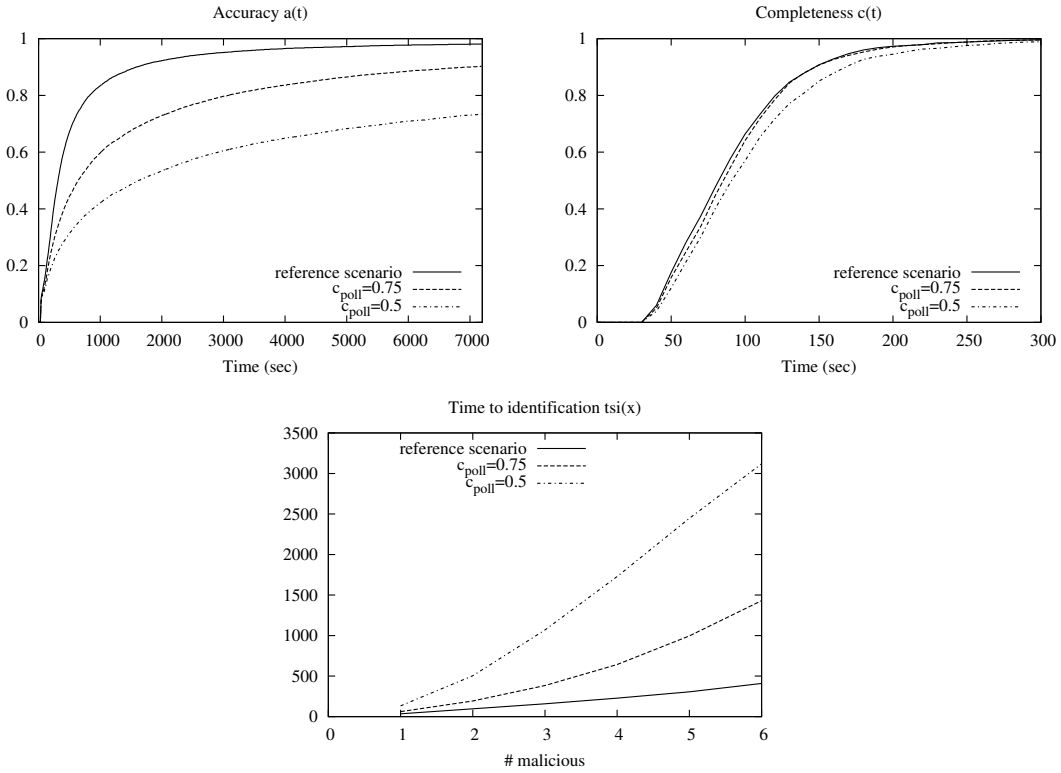


Figure 5.10: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different pollution intensities at the chunk level (c_{poll})

all malicious nodes (this is a colluding attack to honest nodes). It can be noted that, by colluding, malicious nodes only succeed in slightly delay their identification. As a matter of fact, SIEVE is able to correctly identify all malicious nodes anyway at the end of the simulation experiments.

- Increasing the number of malicious nodes: the last stress test for SIEVE is to consider its performance for an increasing number of malicious nodes that coordinate to launch the colluding attack. In Figure 5.14 we show results for up to $P = 100$ malicious nodes in the system (in all cases, $P_{fast} = P_{slow} = \frac{P}{2}$). It can be noted that SIEVE reaches high accuracy at the end of the simulation experiments with an increase of the $tsi(x)$ values, provided that the system runs for a long enough time.

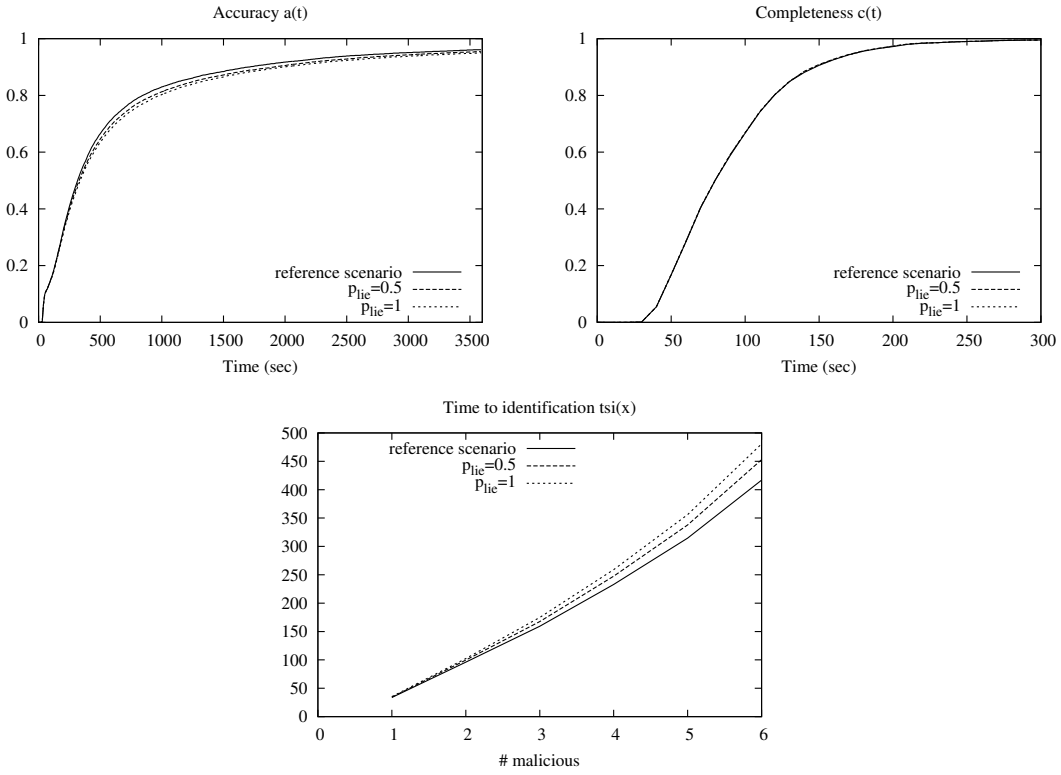


Figure 5.11: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different lying intensities

5.5.7 Coding efficiency vs. SIEVE performance

Coding efficiency is defined as the capability of decoding with the smallest possible overhead ϵ. It is well known that LT codes overhead decreases as K increases [53]; therefore high values of K are preferred in most cases. We compared the SIEVE performance for different values of K and S_{cb} to keep the chunk size constant; in particular, Figure 5.15 shows results for $K = 50, S_{cb} = 1000$ and $K = 200, S_{cb} = 250$. It can be noted that small values of K yield better accuracy and lower reaction times.

Table 5.4 summarizes the structural properties of $\mathcal{G}_n$ for different values of K. It can be noted $K = 50$ yields both lower average number of nodes per check and length four cycles.

As a side effect of using a smaller value for K, the probability of receiving a

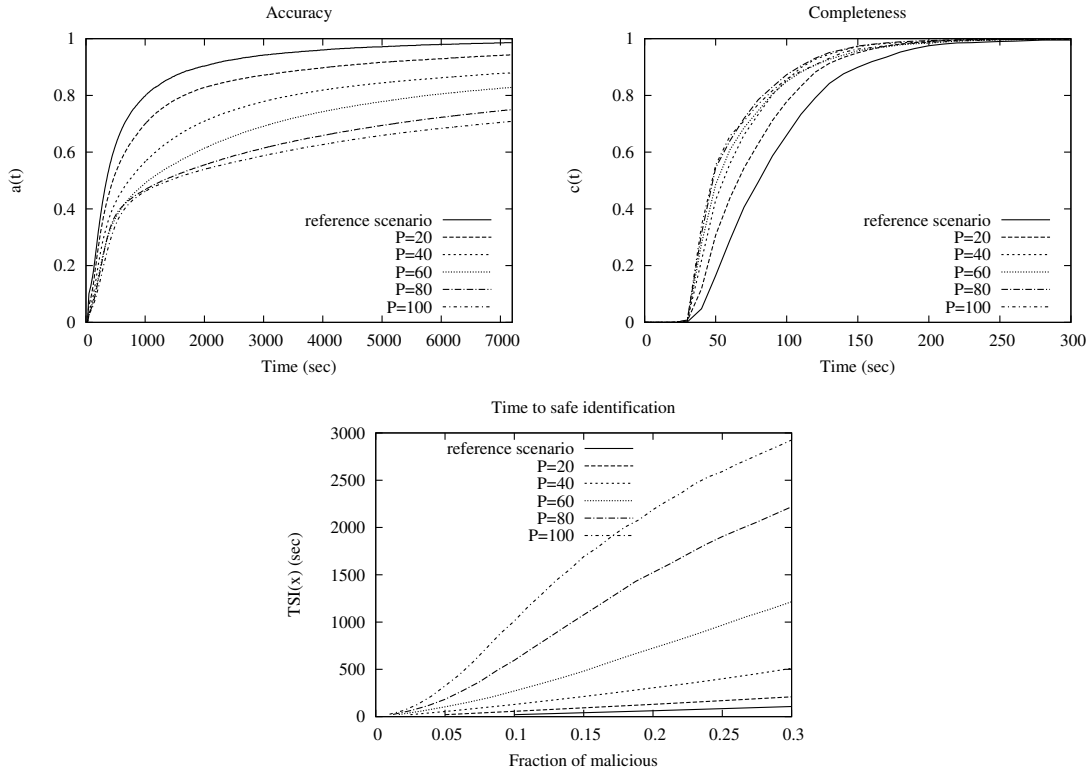


Figure 5.12: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for $p_{lie} = 1$ and increasing number of malicious nodes

coded packet from a malicious node is lower as confirmed by the attack damage, i.e., the average fraction of corrupted decoded chunks, that is equal to 0.20 for $K = 50$ and 0.33 for $K = 200$.

5.5.8 Bandwidth, memory and CPU costs

The bandwidth overhead required by a node to implement SIEVE is determined by the number of bits necessary to transmit a check when decoding a data chunk plus the bits required by the broadcasting of Q checks every T_s seconds. In our simulations a check I is represented by using a message whose payload size is equal to $b_I = 32|\mathcal{U}_\jmath| + 1$ bits; in this expression $32|\mathcal{U}_\jmath|$ bits are required to store the node identifiers represented as integers and one bit is used for the chunk corruption flag.

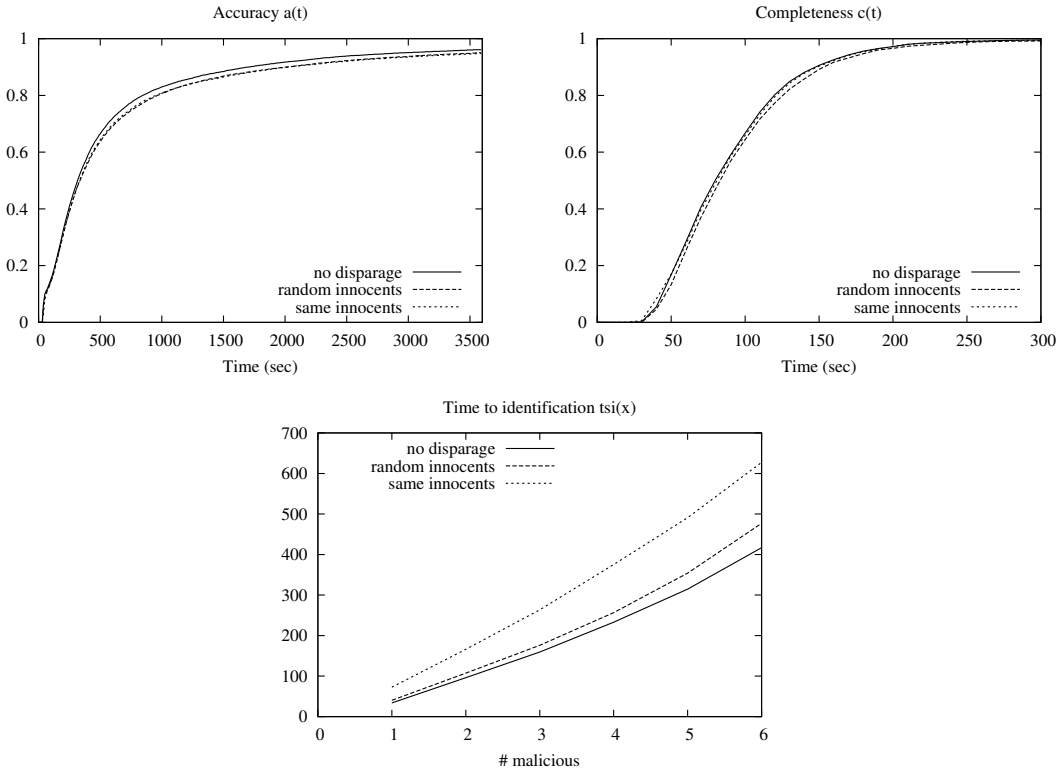


Figure 5.13: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different disparaging attacks

From the simulations of the reference scenario, we measured an average value of the bandwidth overhead equal to 301 bps per node (95% confidence interval $[193, 408]$). We also measured the average throughput, defined as the number of decoded bits (of data chunks) per second, that is equal to 20483 bps (95% confidence interval $[13162, 27803]$). It follows that SIEVE communication overhead, defined as the ratio between bandwidth overhead and throughput, is limited to about 1.5%. It is worth noticing that one contribution to the bandwidth overhead depends on Q and T_s and can therefore be traded-off. As an example, the SIEVE communication overhead for $Q = 5$ and $T_s = 15$s (see Figure 5.4) turns out to be about the 0.5%.

One of the main requirements a solution for MANET must satisfy is to have a low computational and memory cost. We measured the average CPU time

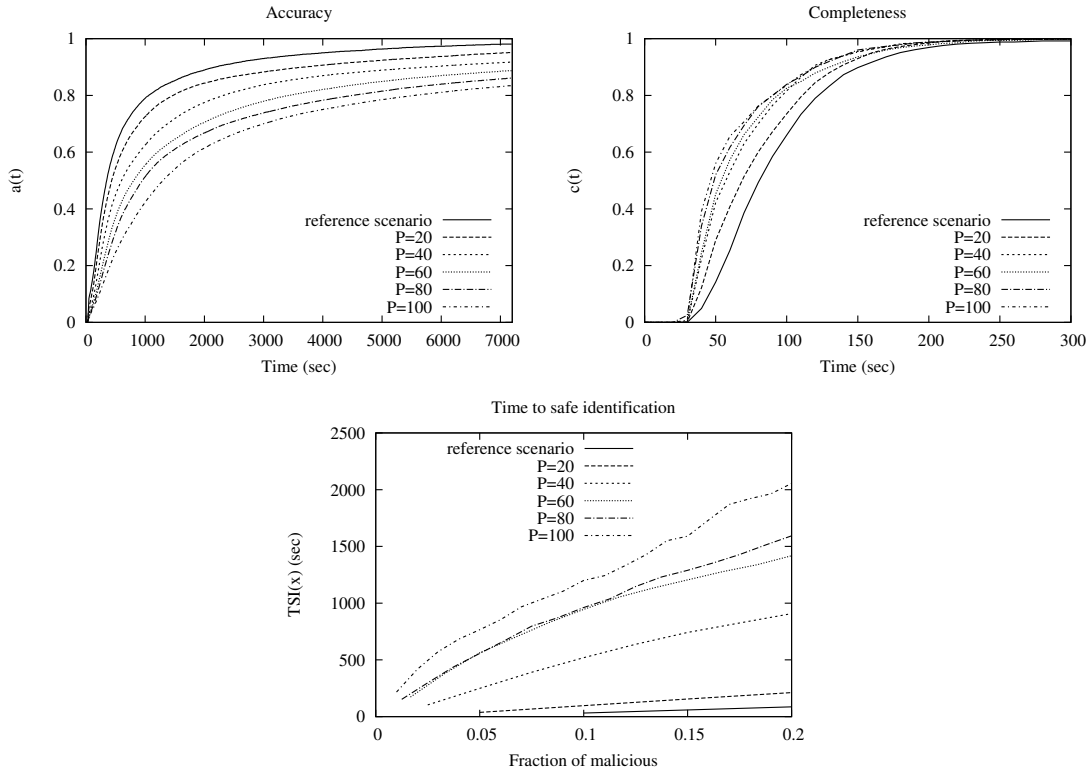


Figure 5.14: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for increasing number of colluding malicious nodes

experienced to run SIEVE on a single factor graph in our C++ implementation on an Intel(R) Core i5 2.80GHz CPU: we obtained 25ms. Of course, MANET nodes do not have the same computational power of a desktop PC but newer CPUs equipping Tablets and smart phones are reducing the gap, e.g., the ARM Cortex-A9 MPCore has up to 4 cores, 2 GHz clock, and 10,000 DMIPS.

Furthermore, storage requirements are very low: the average number of checks in $\mathcal{G}_n$ for fast, slow, and still nodes is 435, 423, and 742, respectively. The average number of nodes in $\mathcal{G}_n$ is equal to 227, 224, and 228, respectively. The factor graph must be represented as a dynamic undirected bipartite graph. Several data structures can be used. In our implementation for each check I (received during the last w seconds) we store the set of uploaders $i \in \mathcal{U}_I$, each one representing an arc of the factor graph (represented as the pair of identifiers (I, i)). Each

K	$\lvert\mathcal{C}\rvert$	$z_{\mathcal{U}}$	$\lvert\mathcal{U}\rvert$	$z_{\mathcal{C}}$	length 4 cycles
50	499.447	6.10647	230.457	12.9598	11508.5
100	492.807	8.5262	231.687	17.3395	32021.6
200	480.232	11.3097	229.752	22.1949	94461.8

Table 5.4: Structural properties of $\mathcal{G}_n$ for different values of K

node/check identifier is represented as a 32 bit integer; therefore, the number of bits for storing the factor graph is equal to $2 \cdot 32 \cdot |\mathcal{E}|$ bits. In our experiments, the average number of arcs of the factor graph is equal to 3823, 3962, and 5790 (for fast, slow and still nodes respectively). It follows that the average memory requirements is equal to 30.5 kB, 31.7 kB, and 46.3 kB, respectively.

5.6 Published works

The work on polluter identification, based on the SIEVE protocol and our FountainApplication module for the ns-3 simulator has been originally published in *"SIEVE: a distributed, accurate, and robust technique to identify malicious nodes in data dissemination on MANET"* [6].
Subsequently it evolved further and the expanded evolution has been publised in journal *"Exploiting rateless codes and belief propagation to infer identity of polluters in MANET"* [9].

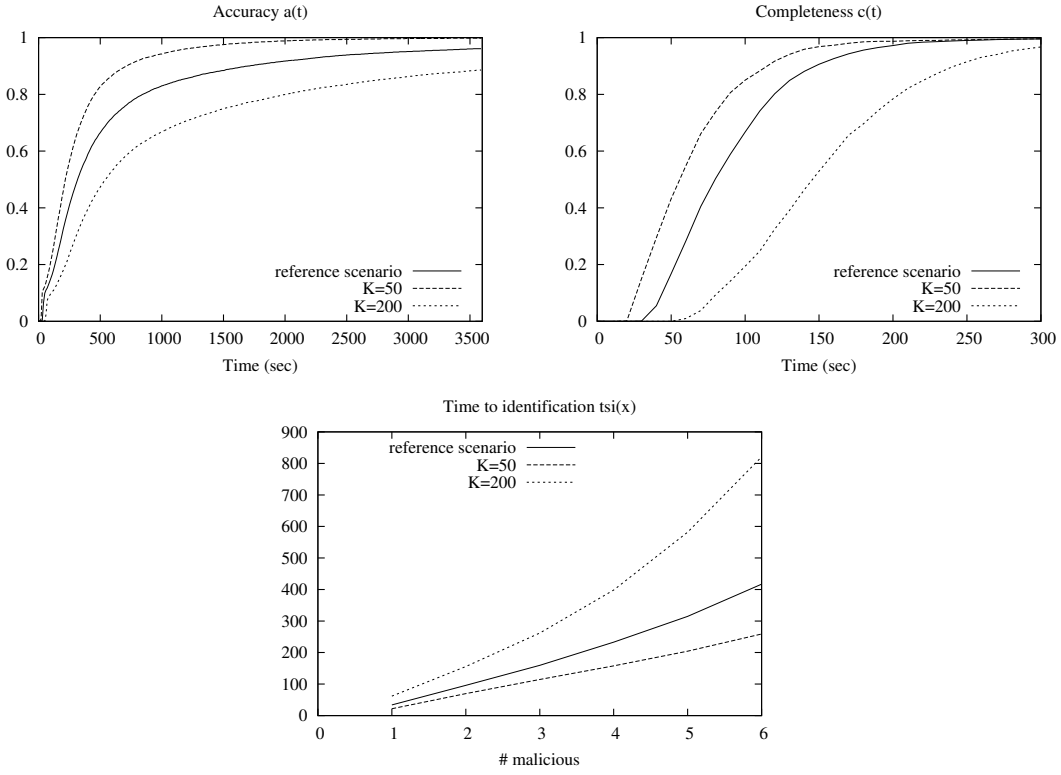


Figure 5.15: Accuracy (left), completeness (middle), and time to identification (right) of SIEVE for different values of K

Chapter 6

CCN and CCN-TV

6.1 Overview

Content-Centric Networking (CCN) [7] is a promising data-centric architecture, based on in-network caching, name-driven routing, and receiver-initiated sessions, which can greatly enhance the way Internet resources are currently used, making support for a broader set of users with increasing traffic demands possible.

The CCN vision is, currently, attracting the attention of many researchers across the world, since it has all the potential to become ready to the market, to be gradually deployed in the Internet of today, and to facilitate a graceful transition from a host-centric networking rationale to a more effective data-centric working behavior.

At the same time, several issues have to be investigated before CCN can be safely deployed at the Internet scale. They include routing, congestion control, caching operations, name-space planning, and application design. With reference to application-related facets, it is worth noticing that the demand for TV services is growing at an exponential rate over time, thus requiring a very careful analysis of their performance in CCN architectures.

To this end, in the present contribution we deploy a CCN-TV system, capable of delivering real-time streaming TV services, and we evaluate its performance through a simulation campaign based on real-world topologies.

Due to the relevant importance that content sharing applications are going to play in the upcoming future [69, 70], the Content Centric Networking (CCN) rationale [71] has been proposed as a possible way to drive the current *host-centric* Internet paradigm towards a novel *content-centric* behaviour. It is based on in-network caching operations, receiver initiated data exchange, hierarchical content naming, and native support to security and privacy.

In a CCN, contents are split in chunks which are requested using opposite *Interest* messages, generated at the client side. Each *Interest* is then routed until it reaches a node which has, in its own cache, a copy of the requested item. Then, this copy is sent, as a *Data* message, back along the path the *Interest* had gone through. Intermediate nodes can cache the *Data* before forwarding it to the next node (more details on the CCN working behavior is provided in Section 6.2).

Since its birth, the CCN vision has gained a warm attention from both scientific and industrial communities to discover the bounds of its real potential from different perspectives.

Many studies have focused on modeling and designing caching strategies and data-transfer performance such as in [72]-[77]. In that direction, it is now clear that the cache size may have a major impact on the overall performance of a CCN even if finding an optimal caching strategy is still an open problem to address. With respect to congestion control issues, instead, recent studies shown as the classic additive increase multiplicative decrease algorithm, at the foundation of TCP, could be inherited by CCN, provided that some countermeasures are employed to limit unfairness issues that could arise among contents with different popularities [78, 79]. Another very relevant topic in CCN research covers routing operations, which are essential to properly drive the dissemination of receiver generated *Interest* packets. To this end, the adoption of Bloom filters appears a promising solution [80, 81] that merits further investigations.

Starting from this premise, it is evident that all facets of CCN are going to be afforded in an ebullient panorama of activities that cover both the underlying mechanisms within the protocol architecture and the design of content oriented applications and services.

With reference to application-related features, it is worth to notice that the demand for TV services is growing at an exponential rate over the time [69], thus requiring a very careful analysis of their performance in CCN architectures. A preliminary study presented in [82] addresses time shifted applications only, whereas live streaming operations are currently under investigation as testified in the interesting contribution [83]. To complement the research efforts of the community in a so relevant domain, the present manuscript is intended to design a complete CCN-TV system encompassing all the main facets of typical live streaming video services.

The proposed CCN-TV has been tested through a solid simulation campaign based on real topologies. To this end, the *ccnSim* simulator [84] has been properly tailored to our purposes by adding window based flow control, handling of playout delay and real-time data, advanced logging functions, links with bandwidth constraints, and data session bootstrapping. Simulation results shown that in-network caching seems to play a minor role in live streaming video services, mainly because cached data looses its utility after the deadline is expired.

On the other hand, the way CCN handles client requests for TV contents helps improving the performance of the system with respect to a plain IP infrastructure.

The rest of the topic is organized as follows: Section 6.2 illustrates the core concepts of CCN. Section 6.3 describes the CCN-TV system we propose in this Thesis, which is then evaluated in Section 6.4. Finally, Section 7.3 summarizes the main findings of this contribution and draws future research.

6.2 Basic background on CCN

Internet usage has undergone a radical change during the last ten years: content-sharing applications are now dominant whereas the IP architecture still provides a connection-less service among remote hosts [70]. Users ask for contents, looking for *what* they intend to retrieve from the Internet while the language spoken by the underlying IP infrastructure provides answers on *where* a packet should be sent. This mismatch is actually overcome by a number of workarounds at

different levels of the protocol stack, which, indeed, limit the overall efficiency of the Internet.

The so-called *Future Internet* represents a family of possible solutions to the aforementioned issues, embracing novel communication models that can better accommodate and fulfill users' requirements related to efficiency, security, support to mobility, and integrated media experience [70].

At the present stage, many valid proposals for the Future Internet exist, such as the Publish Subscribe Internet Routing Paradigm, the 4WARD NetInf project, and the Cache-and-Forward Network Architecture, the Data-Oriented Network Architecture and the CCN approach [71, 85], having different levels of compatibility with the IP paradigm.

Among them, the CCN vision looks promising since, besides being "data-centric", it can be gracefully integrated with today's IP-based Internet. In a CCN, all content is unambiguously identified by a hierarchical name, allowing users to retrieve information without being aware about the physical location of servers (e.g. IP address). Also, commu-nication is receiver-driven and based on content chunk ex-change, name-based routing, and self-certifying packets [71].

Nevertheless, the real performance bounds of a CCN and the actual benefits it can bring to the Internet are still not entirely known, mainly because there are many open issues that surround the CCN architecture, such as those related to: (i) routing, (ii) congestion control, (iii) strategy layer design, (iv) name space definition, (v) semantic layer, (vi) accurate models, and (v) fairness among heterogeneous applications and contents having different popularities.

As specified before, CCN communications are driven by the consumer of data and only two types of messages are exchanged (namely *Interest* and *Data*). A user may ask for a content by issuing an *Interest*, which is routed within the CCN towards the nodes in posses of the required information, thus triggering them to reply with *Data* packets.

The routing operations are performed by the strategy layer only for *Interest* packets. *Data* messages, instead, just follow the reverse path to the requesting user, allowing every intermediate node to cache the forwarded content.

CCN adopts a hierarchical structure for *names*, which leads to a *name tree*. In particular, it is formed by several components, each one made by a number

of arbitrary octets (optionally encrypted), so that every *name prefix* identifies a sub-tree in the name space. An *Interest* can specify the full name of the content or its prefix, thus accessing to the entire collection of elements under that prefix.

Finally, since contents are exchanged based on their names, multiple nodes interested in a particular data can share it using multicast suppression techniques over a broadcast medium. Analyzing a CCN node, it is possible to identify three main structures [71].

- the *Content Store* (CS): a cache memory that can implement different replacement policies as Least Recently Used (LRU) and Least Frequently Used (LFU);
- the *Forwarding Information Base* (FIB): is similar to an IP FIB except for the possibility to have a list of *faces*[1] for each Content Name entry, thus allowing *Interest* packets to be forwarded towards many potential sources of the required Data;
- the *Pending Interest Table* (PIT): is a Table used to keep track of the *Interest* packets that have been forwarded upstream towards content sources, combining them with the respective arrival faces, thus allowing the properly delivery of backward Data packets sent in response to *Interests*.

When an *Interest* packet arrives to a CCN node, the CS is searched to discover whether a data item is already available as an answer to be sent immediately back to the requesting user. Otherwise, the PIT is consulted to find out if others *Interest* packets, requiring the same content, have been already forwarded towards potential sources of the required data. In this case, the *Interest*'s arrival face is added to the PIT entry. Otherwise, the FIB is examined to search a matching entry, indicating the list of faces the *Interest* has to be forwarded through. At the end, if there is not any FIB entry, the *Interest* is discarded.

On the other hand, when a Data packet is received, the PIT Table comes into play, which, keeping track of all previously forwarded *Interest* packets, allows to establish a backward path to the node that requested the data.

[1] In CCN it is used the term "*face*" instead of the "*interface*" because packets are also exchanged between application process, besides being forwarded only over real network interfaces.

6.3 CCN-TV architecture

Unlike Video-On-Demand, real-time video distribution has to deal with a specific class of problems to ensure the timely delivery of an ordered stream of chunks. Video chunks have to be received in playing order and within a given time interval (the *playout delay*), before they are actually played, thus "expiring". A chunk not delivered before its expiration will result in degradation of the rendered video, impacting the end user Quality of Experience; the extent of the video degradation may vary depending on the video codec and the type of the lost frame. To solve these challenges, client nodes implement a receiving buffer queue where the chunks are stored in order, that is emptied while the video is being played. Therefore, any chunk not received before its playing instant becomes useless. To reduce the chance of chunk loss several mechanisms can be put in place to retransmit requests for chunks close to expiration. Furthermore, in modern codecs, such as H.264 [86], there are different types of video frames, encoded using intra-frame or inter-frame techniques, each having a different level of importance. For example, the so called I-frames, derived using intra-frame compression techniques, actually represent a full video image, providing a fundamental reference for subsequent inter-frame encoded images.

With this in mind in CCN-TV we considered a network of nodes requesting different real-time video streams, identified by a *channelID*, served by one or more broadcast server.

Unlike canonical UDP/TCP-based streaming, in CCN-TV each video chunk, identified by a progressive *chunk number*, has to be requested individually, via a dedicated *Interest*.

Although this might look costly at a first sight, CCN's routing mechanisms ensure that *Interests* for the same chunk do not propagate twice along the same routing path (unless under specific conditions, as explained in Subsection 6.3.2), and the caching strategy implemented by every node can make sure that *Interests* for the most popular contents are served before going through the whole routing path. Moreover, the *Interest*/*data* exchange allows for a natural flow control mechanism, where each node can request for new chunks just when the old ones have arrived.

Herein we thoroughly describe the design rationale and all the details of the CCN-TV system this Thesis targets. Specifically, in what follows, we present: the bootstrap phase, the flow control strategy, and the management of retransmitted *Interest* packets.

6.3.1 Channel bootstrap

One challenge we are faced with in a real-time scenario is to bootstrap the channel to be received. Bootstrapping a channel involves the operations of finding a routing path to the nearest channel provider and locating the first valid chunkID of the video stream. Due to video codec requirements, the video stream can be visualized only once the first I-frame has been received. Therefore, a client has to first gather from the server the first chunk (and the corresponding chunkID) of the last generated I-frame. To do so, it sends an *Interest* packet for the URI: `[domain]/[channelID]`, with the *Status* field set to *BOOTSTRAP* and the *Nonce* field set by the client, as in in Subsection 6.3.4. An *Interest* with Status = BOOTSTRAP would travel unblocked until it reaches the first good stream repository (i.e. a node who can provide a continuous real-time flow of chunks, not just cached ones).
To this *Interest*, the server responds with a data message in the format `[domain] / [channelID] / [chunkID]`, with *chunkID* being the first chunk of the last generated I-frame, and the corresponding *Frame ID* field value. Upon receipt, the node starts asking for subsequent chunks, using the sliding window mechanism detailed in Subsection 6.3.2. The use of a *nonce* (a uniquely generated identifier) in the *Interest* URI allows the *Interest* to propagate to the server without being blocked along the routing path, as every bootstrap *Interest* for the same channel has a different nonce. It also avoids the retrieval of the data response from the cache of an intermediate node; the risk, in this case, is the retrieval of a bootstrap data message for a given channel from a cache containing an already expired chunk of an I-frame.

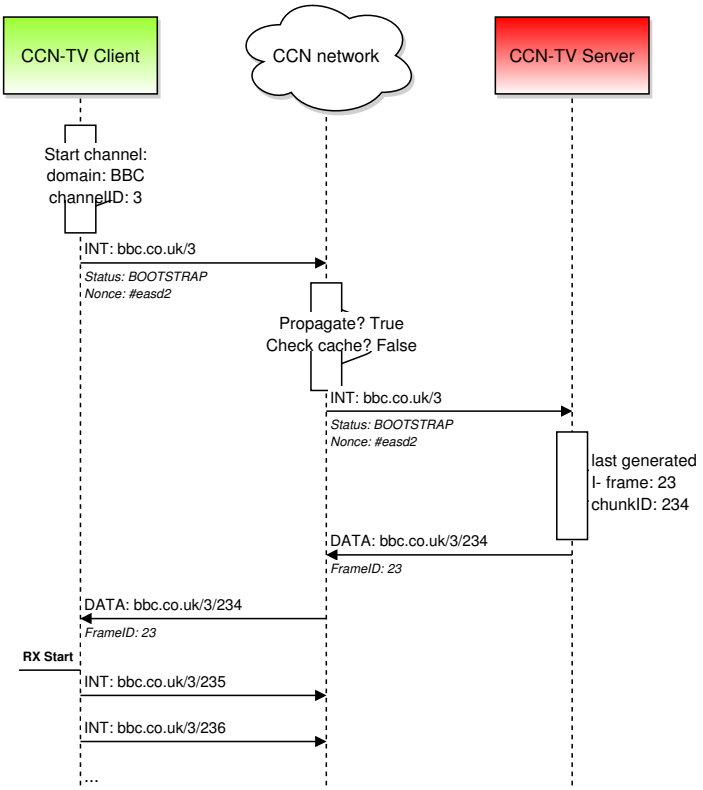


Figure 6.1: Bootstrap handshake

6.3.2 Flow control

From the moment a node receives the bootstrap data message, it can initiate the sliding window mechanism to request the subsequent chunks in an optimal way. Each node has a windows of size W to store W pending chunk. We define *pending chunk* a chunk whose *Interest* has been sent by the node, and the window containing the pending chunks a *Pending Window*. Together with the chunkID, we store in the pending window other information, such as the timestamp of the first request and the timestamp of the last retransmission. Whenever a new data message is received, the algorithm described in Figure 6.2 runs over the Pending Window, to perform the following operations:

1. Purge the Pending Window from all the chunks who are expired, i.e., who have already been played, to free new space in the sliding window.

2. Retransmit all chunks that have not been received for a given timeout (on-

ward denoted as *windowTimeout*.

3. Transmit, for each slot that got freed by the received or expired chunks, the *Interest* for a new one.

```
1: procedure SendInterests(PW, W, WinT, Now, LC)
2:     # Remove all expired Interest
3:     for all CID in PW do
4:         if CID is expired then
5:             remove CID from PW
6:         end if
7:     end for
8:     # Resend stale Interests
9:     for all CID in PW do
10:        if lastTx(CID) < Now − WinT then
11:            resend(Int(CID))
12:            lastTx(CID) ← Now
13:        end if
14:    end for
15:    # Send Interests for new chunk
16:    NNC ← W − size(PW)
17:    for i ← 1, NNC do
18:        send(Int(LC))
19:        lastTx(LC) ← Now
20:        add LC to PW
21:        LC ← LC + 1
22:    end for
23: end procedure
```

Figure 6.2: Sliding window algorithm

Furthermore, the same operations are performed if a node doesn't receive any data for at least *windowTimeout* seconds, in which case, all the *Interests* for non-expired chunks in the Pending Window are retransmitted, together with new chunks if new slots have been freed due to expired chunks.

Figure 6.2 details the implemented algorithm; for the purpose of brevity and readability, the variable names have been contracted: PW is the Pending Window, W is the aforementioned system parameter, indicating how many *Interests* should a node have ongoing, $WinT$ is the window timeout, after which *Interests* in

the Pending Window are resent, *Int* is a new *Interest* message, *CID* is a chunkID in the pending window, *lastTx* is the transmission time of the most recent *Interest* for a given chunkID, *LC* is the chunkID of the most recent requested chunk and *NNC* is the number of new chunks to request, after the pending window has been purged.

To provide a further insight, we reported in Figure 6.3 an example of the conceived sliding window algorithm, in which we have set the value of W to be equal to 3.

6.3.3 Interest routing

As described in Section 6.2, CCN nodes along the routing path of an *Interest* will stop the propagation of said *Interest*, if they have previously routed another *Interest* for the same resource, and the correspondent data has note been sent back yet; instead, they will simply update their Pending Interest Table adding the face from where this newcomer *Interest* was originated, so to reroute the data back recursively along the path the *Interest* has gone through.

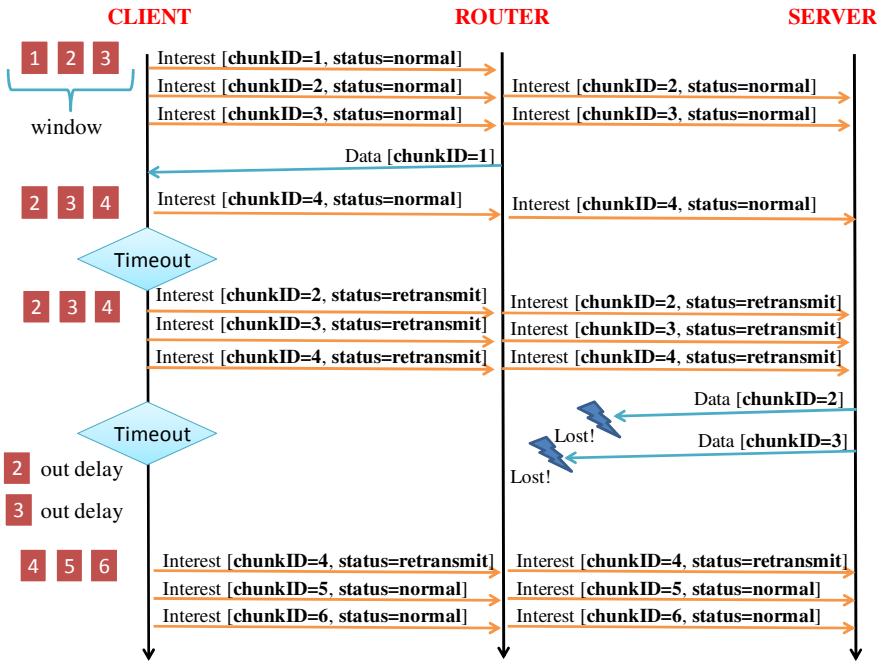


Figure 6.3: Sliding window example

This mechanism ensures flow control and limits the propagation of duplicate *Interests*, in case several nodes in the same network are watching the same channel.

However, to make the *Interest* retransmission mechanism effective, a retransmitted *Interest* needs to propagate all the way up to the server, or to the first node with the desired chunk in cache. Therefore, retransmitted *Interests* carry the *Status* field set to *Retransmission* to mark if the *Interest* is a retransmission or not, and each node along the routing path propagates the *Interests* marked as retransmitted, thus skipping the usual CCN mechanism, unless the correspondent chunk is found in the cache.

Table 6.1: Messages used in CCN-TV

Packet type	**Field**	**Content**
Chunk Interest	*Content Name*	[*domain*]/[*channelID*]/[*chunkID*].
	nonce	Used only for the bootstrap phase.
	Publisher Filter	Not used.
	Status	*Bootstrap*, *Normal*, *Retransmission*.
Chunk Data	*Content Name*	[*domain*]/[*channelID*]/[*chunkID*].
	Publisher ID	Optional.
	Signature	Optional (for increased content authentication).
	Stale Time	Set to the *frame time* of the frame the chunk belongs to.
	Frame ID	ID of the frame the chunk belongs to.
	Data	The request video chunk binary data.

6.3.4 CCN-TV messages

As detailed above, additional functionalities required by the system for real-time video streaming are implemented on top of existing CCN data and *Interest* messages via new fields carrying the required additional information.

However, should the situation require the system to conform to classical CCN messages, all additional fields can be easily replaced by additional fields in the content name.

Table 6.1 shows how we made use of the classical CCN message fields, together with the new fields and their use.

In particular, CCN-TV *Interests* carry an additional *Status* fields marking if the *Interest* is a *bootstrap Interest* (Subsection 6.3.1), a normal one or a *retransmission* (Subsection 6.3.3).

Concerning CCN data message, we extended the messages with an additional field, i.e., *Frame ID*, containing the ID of the frame to which the embedded chunk belongs to.

6.4 Simulation results

In this Section, we will evaluate performances of the proposed CCN-TV architecture. To this end, we implemented it within *ccnSim*, i.e., an open source and scalable chunk-level simulator of CCN [84] built on top of the Omnet++ framework [3], dedicated to the evaluation of Video On Demand systems on top of CCN.

By itself, *ccnSim* models a complete video distribution systems, with a high degree of fidelity concerning catalogs, requests and repositories distribution, and network topologies. Since, however, it did not support the real-time constraints required by our evaluations, it has been modified and improved in the following ways:

- we added support for links with bounded capacity and packets with a well defined size, which was missing in *ccnSim*, to be able and estimate the CCN behavior under some bandwidth constraints;
- due to the datarate channels, we implemented a transmission queue for each face of each node, in order to properly manage the packet transmission;
- we added the support for synthetic video traces, so to be able to transmit and receive chunk of real videos, and consequently being able to reconstruct the received video and evaluate its PSNR;
- due to possible expiration of *Interests*, we implemented a cleanup mechanism for each node's PIT, to avoid having in long term stale entries due to expired chunks;

- we improved and enriched the logging system, so to be able to record each node's received chunks and reconstruct the received video;
- we added more controls server-side, to send a data only for those chunks who have already been generated.

Furthermore, the following mechanisms, beyond the provided ones, have been implemented in the simulator:

- the sliding window mechanism described above, and all the related data structures;
- the *Interest* forceful propagation in case of retransmission;
- constant data reception, until a channel is changed.

The extended *ccnSim* simulator is available at [87].

The aim of our study is to evaluate how the behavior of the CCN-TV system is influenced by (i) the amount of the network bandwidth dedicated to real-time streaming services, (ii) the *windowTimeout* adopted by the sliding window mechanism, (iii) the playout delay, and (iv) the cache decision policy.

We focus the attention on the GEANT network, which interconnects the European research and education institutions and it is composed by 22 routers [88]. Every node of the network is considered to be a direct CCN node, i.e. no TCP or UDP encapsulation is implemented.

We assume the presence of only one small video streaming provider that offers 5 parallel real-time transmissions to remote clients.
It is connected to one of the nodes forming the GEANT topology. In every simulation round, each video content is mapped to a video stream compressed using H.264 [86] at a average coding rate randomly chosen in the range [250, 2000] kbps.

Clients, i.e., CCN nodes that download video contents from the server, are connected to remaining nodes (1 client per node).
In order to catch the behavior of people watching TV, we modeled two groups of

users: *faithful* and *zapping*.
Faithful users are attached to one video channel for the whole simulation.
Zapping users, instead, change frequently the channel among those offered by the server according to a Poisson process with parameter $\lambda = 0.0666$.

Further, the channel selection process has been modeled considering that contents popularities follow a Zipf distribution, according to [89], the most of works presented in literature set the parameter α of the Zipf distribution in the range [0.6, 2.5]. In line with these common settings, we set $\alpha = 1$.

Once a client decides to watch a specific channel, it performs the bootstrap process described in the previous Section and then starts sending *Interest* packets following the designed sliding window mechanism.

In our tests, we adopted the optimal routing strategy, already available within the *ccnSim* framework. According to it, *Interest* packets are routed towards the video server along the shortest path. On the other hand, three caching strategies have been considered in our study: *no-cache*, *LRU*, and *FIFO* [89]. When well known *LRU* or *FIFO* policies are adopted, we set the size of the cache to 100 chunks. The *no-cache* policy is intended to evaluate the performance of the CCN without using any caching mechanism.

The window size W has been set to 10, ensuring that faces of the server are almost fully loaded in all considered scenarios. Also, the transmission queue length associated to each face, Q, has been set, in order to be larger than

$$Q = 2 \cdot L_c \cdot P_D \tag{6.1}$$

where L_c and P_D represent link capacity and maximum propagation delay in the considered network topology. All simulation parameters have been summarized in Table 6.2.

6.4.1 Interest generation process

As a first step, we investigate the impact that the sliding window mechanism has on the amount of sent *Interest* packets, which is shown in Figure 6.4. From these plots it is evident that the highest *windowTimeout*, the lowest the total number

Table 6.2: Summary of simulation parameters

Parameter	**Value**
Topology	GEANT with 22 routers
Link capacity	40 Mbps and 100 Mbps
Number of real-time service provides	1
Number of clients	21
Catalog size	5 files
Chunk size	10Kbytes
Video average bit rate	250kbps, 600kbps, 1000kbps and 2000kbps
W (window size)	10
Playout delay	10s and 15s
Window timeout	1s, 3s, and 5s
Caching strategy	No cache, LRU, and FIFO
Cache size	100 chunks
Client zapping behavior	50% fixed, 50% changing on average every 15s
Simulation time	60s
Number of seeds	5

of *Interest* messages sent by end users. When the *windowTimeout* increases, the probability that a given client does not receive any chunks within such a time interval decreases and, as a consequence, also the number of retransmitted *Interest* packets decreases as well. As a further confirm of this result, Figure 6.5 shows that the percentage of duplicated *Interest* packets increases when the *windowTimeout* decreases due to a high number of chunks that are unable to reach the client within the expected timeliness.

As expected, the *playout delay* has a minor impact on the number of generated *Interest* messages, which, as is known from the theory on sliding window mechanisms [90], can be only influenced by window size (W) and *windowTimeout*.

Also, caching operations do not have any significant impact on the number of generated *Interest* messages. The main reason being that chucks stored in cache memories lose their effectiveness after their deadline is expired.

On the other hand, the link capacity greatly influence the *Interest* transmission rate. From Figure 6.4 emerges, in fact, that the number of *Interest* lowers

when the capacity of links decreases. This is because a limited bandwidth reduces the quota of received chunks, thus preventing a rapid advancement of the sliding window. In other terms, this result proves, once again, the effectiveness of the sliding window mechanism in CCN.

6.4.2 QoS and QoE

The first important parameter that describes how CCN-TV settings affect the quality of service offered to end users is the chunk loss ratio, which represents the percentage of chunks that have not been received in time (i.e., before the expiration of the *playout delay*) by clients.

From Figure 6.6, showing the chunk loss ratio measured in all considered network scenarios, we note that *playout delay* plays a fundamental role. When the *playout delay* increases, in fact, the client could receive a *Data* packet within a longer time interval, thus reducing the amount of chunks discarded because out of delay.

On the other hand, a slight increment of the chunk loss ratio can be registered by increasing the *windowTimeout*. If the client retransmits an *Interest* packet after long time, there is the risk that the *Data* packet will be reached by the destination after the expiration of the *playout delay*.

In addition, we note that a reduction of the link capacities leads to a higher number of lost chunks, due to increased latencies induced by network congestion.

It is very important to remark that the presence of the cache can guarantee only a small reduction of the chunk loss ratio. With our study, we found that, in the presence of real-time flows, the cache does not represent an important CCN feature.

On the other hand, we noticed that the PIT plays a more relevant role. In fact, in presence of live video streaming services, clients that are connected to a channel request the same chunks simultaneously. In this case, a CCN router has to handle multiple *Interest* messages that, even though sent by different users, are related to the same content.

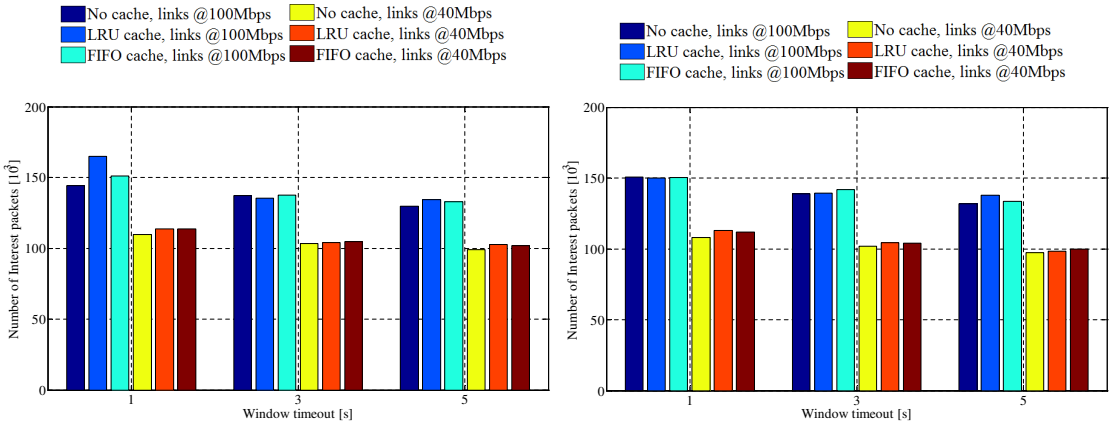


Figure 6.4: Total number of *Interest* packets sent by clients with playout delay of (a) 10s and (b) 15s

According to the CCN paradigm, such a node will store all of these requests into the PIT, waiting for the corresponding *Data* packet. As soon as the packet is received, the router will forward it to all users that have requested the chunk in the past.

Following to these considerations, the use of the cache will not produce a relevant gain of network performances. Indeed, the PIT helps reducing the burden at the server side by avoiding that many *Interest* packets for the same chunk are routed to the server.

To conclude our study, we have computed Peak Signal to Noise Ratio (PSNR), which is nowadays one of the most diffused metrics for evaluating user satisfaction, together with interactivity level, in real time video applications [91]. Results shown in Figure 6.7 are in line with those reported for chunk loss ratio (the PNSR is higher in the same case in which the chunk loss ratio is lower). Again, link capacity greatly influences the quality of the TV service provided to users. According to [92], the obtained PSNR values can be translated to a Mean Opinion Score (MOS) not less than 4, corresponding to satisfactory quality for almost all users.

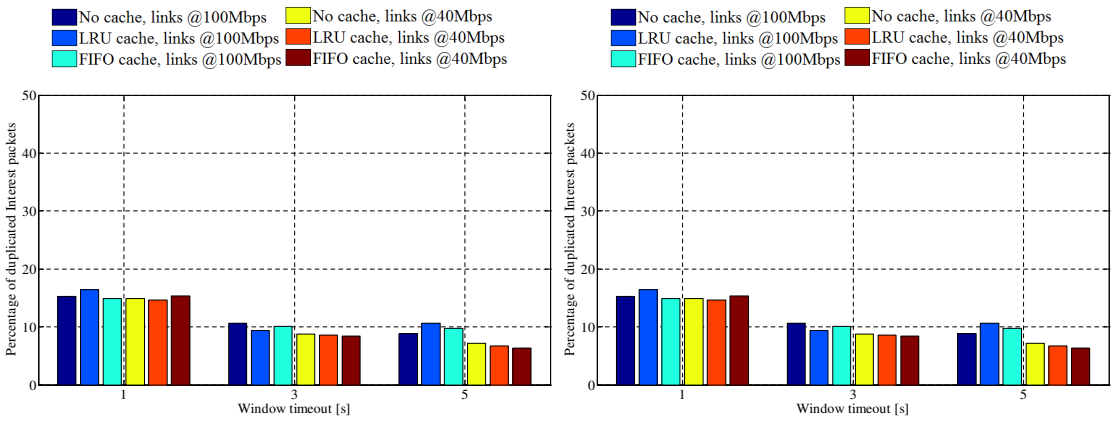


Figure 6.5: Percentage of duplicated *Interest* packets sent by clients with playout delay of (a) 10s and (b) 15s

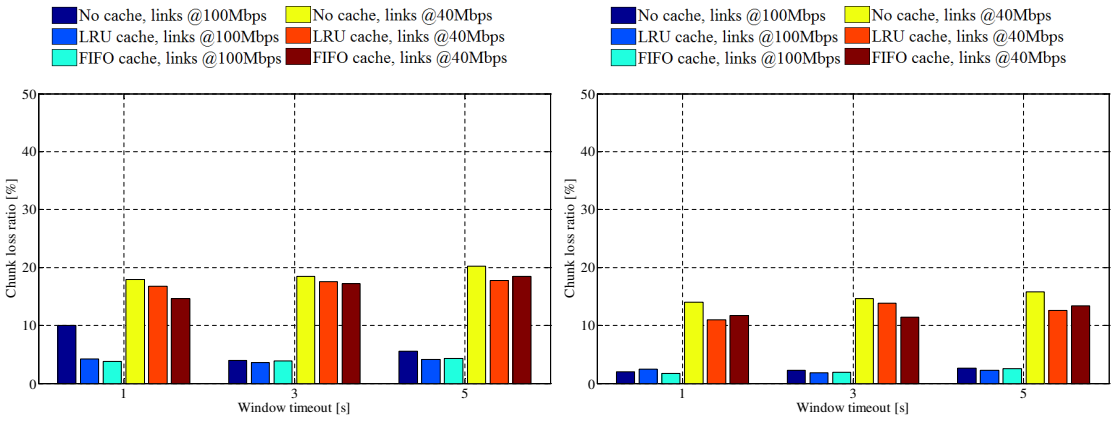


Figure 6.6: Chunk loss ratio with playout delay of (a) 10s and (b) 15s

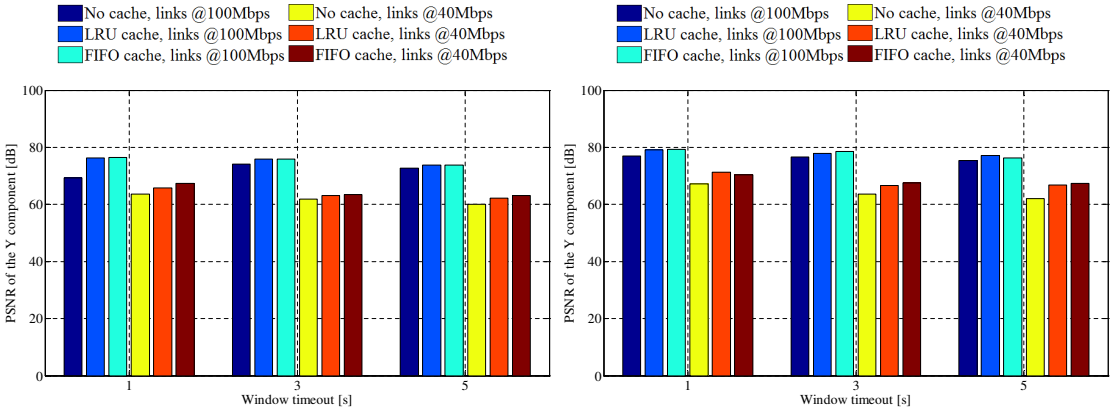


Figure 6.7: PSNR of the Y components of received videos with playout delay of (a) 10s and (b) 15s

6.5 Published works

Our implementation of the CCN protocol to implement a real-time video streaming architecture has been originally published in the article *"CCN-TV: a data-centric approach to real-time video services"* [10] and a subsequent extension has been requested as a book Chapter, integrating crowdsourced content in the infrastructure and analyzing the strong and weak points and performances.
The book chapter, *"Providing crowd-sourced and real-time media services through a NDN-based platform"* [11], has been accepted and is currently awaiting publication.

Part III

Summarizing results and future evolutions

Chapter 7

Summarizing results and future evolutions

Chapter abstract:

In this Chapter we summarize the obtained results in the examined topics and discuss possible and ongoing future evolutions of the works.

For each topic a paragraph presents a sum of the results and evolutions explained in more details in Chapters 4 for Section 7.1, 5 for Section 7.2, and 6 for Section 7.3.

Furthermore Section 7.4 details an ongoing work to unify the above works for a robust CCN-based mobile network coded approach.

7.1 Random graph modeling of overlay networks interconnection

Using our random graphs based model for network interconnection we were able, after validating it with results obtained with a simple simulator, to generate all results in a given interval of parameters so to analyze, for various configurations and interconnections sets, the most efficient parameters to use.

Using the homogenous initial version the results were promising, yet still limited in real case scenarios by the model own limits. We then removed those limits in developing the model heterogeneous version, providing for different configuration for each interconnected network.

While in the first case we were only able to demonstrate how, given the same configuration for all networks, the efficacy of the searches and the number of messages sent varied globally with changing parameters, with the second version of the model we were able to also study the influence of different interconnection topologies between the various network and more in depth the case where a single network contains the desired resource and is then connected to a previously established interconnected network, analyzing the resource diffusion and best parameters to use for maximize availability.

In this Thesis we considered interconnection of large scale unstructured P2P networks through co-located nodes called synapses: these nodes send/forward a query to all the P2P networks they belong to. We developed a generalized random graph based model to represent the topology of one unstructured P2P network, the partition of nodes into synapses, the probabilistic flooding based search algorithms, and the resource popularity. We validated our model against simulations and proved that its predictions are reliable and accurate. The model allowed the analysis of very large and complex systems: we believe that simulation and/or prototype deployment based analysis would be unfeasible in this case.

We are currently working to further extend our model in several directions. In particular, we are generalizing equations to represent heterogeneous topologies and resource availability. As a consequence, we are also extending the analysis to more refined partition of synapses, i.e., to consider the fraction of nodes that belong to a specific set of P2P networks. Furthermore, we are extending the model to represent nodes availability due to churning. Last but not least, we are generalizing the model to represent interconnection of both unstructured and structured P2P networks.

We are currently working, to be published following to this Thesis, to an heterogeneous evolution to the model, where each interconnected network does have its own interconnection degree distribution and resource availability.
As such queries can be initiated from each network i obtaining the pertinent hit probability p_{hit}^{i} and average number M^{i} of generated messages.
Due to the highly abstract modelization of the networks and their relations, the model, especially the heterogeneous one, can be used to investigate different sce-

narios. We choose to analyze in an ongoing work the interconnections of social networks, considering each social network as network in the model, the user's contacts as the interconnections between nodes and the same user on different social networks as the "bridge" nodes interconnecting the networks.
Each query TTL is the number of jumps through the user contacts, and contacts' contacts, until the desired content is reached.

7.2 Polluter identification in MANETs

The results obtained are very encouraging: even in a completely decentralized and destructured ad-hoc mobile network, the list of most probable polluters correctly identifies all the active polluters in a reasonable amount of time with a very high accuracy.

Even if the detection occurs after a few decoding sessions have gone bad and usually takes in the order of a few minutes we have to consider that in the chosen encoding scheme there is simply no way to know if a chunk is corrupted until its decoding, and even then there is no way to directly identify which packet or packets were responsible for poisoning the blocks pool.

The results are even more positive when we analyze the robustness of the protocol to coordinated attacks, basically polluters coordinating to send coherent false checks or even worse accusing one or more specific non polluter nodes to be in fact malicious. If no more than half of the total nodes are malicious (a quite reasonable assumption) the protocol maintains its efficiency, correctly identifying all and only the polluter with a relevant margin of certainty, and only taking more time to correctly guess all of the polluters, even in unrealistically negative situations (a significant percentage of the nodes are polluters and they all coordinate, while the honest nodes aren't even aware of the neighbor presence).

Besides the protocol, being based on the Belief Propagation algorithm of obtained decoding observations is low in computational complexity, so even well suited for mobile and battery limited devices, and easily adaptable to other dis-

tributed transmission protocols, as far as a relative check is produced during or after the decoding process.

In Chapter 5 we have proposed the novel SIEVE technique for the identification of malicious nodes performing a pollution attack within a MANET.
A data dissemination application based on the distribution of LT coded data packets has been considered as a use case, where malicious nodes aim at preventing the delivery of the information corrupting, i.e., polluting, coded packets.
In turn, the reception of a single corrupted packet may break down the decoding of a whole data chunk.

Fortunately, the LT decoding procedure can be used by each node to detect that a data chuck has been attacked; nonetheless, since parallel downloading form multiple nodes is used, such detection does not allow to identify the malicious nodes. This latter represents the core problem solved by SIEVE.

In SIEVE node collaborations are represented by a bipartite graph linking nodes and detection opportunities, the checks. It is worth pointing out that such representation is quite general and can be used in many other collaborative scenarios other than the data dissemination use case analyzed in this Chapter.

SIEVE is a completely distributed technique that, using statistical inference based on the belief propagation algorithm, allows each node to independently analyze local snapshots of the bipartite graph of the collected checks to estimate the probability of nodes being malicious and to perform a progressive ranking of the suspect nodes.

Our results, worked out using detailed ns-3 simulations, show that SIEVE is accurate in letting each honest node identify all malicious nodes under several scenarios. We have analyzed the sensitivity of SIEVE performance to the nodes mobility; we have discovered that SIEVE is very robust to several deceiving actions, colluding attacks launched by malicious nodes, and amount of malicious nodes inside the network.

Future works will be focused in two main directions. From the one hand we will complete the design and experimentation of a full system to counteract an

active attack within MANET, including both malicious node identification and their subsequent isolation or removal. The techniques adopted for the identification and the following removal of malicious nodes clearly require a joint and careful design to optimize the overall performance. The other research line will be devoted to better understand the features of the bipartite graph representation, e.g. presence of loops, average node and check degrees, that affect SIEVE accuracy. Such study is expected to let us further improve its performance by adding the checks in a smart and adaptive way guaranteeing better convergence of the BP algorithm.

The approach by itself is flexible and efficient, as shown by our results, so future works will work toward the two complementary directions of raising the SIEVE accuracy by better analyzing the underlying graph and devising ways to followup to the detection of polluters, so to pass the useful *pollution detection* informations generated by SIEVE to a *pollution avoidance* mechanism which will use such informations to actively avoid (or neutralize) polluters.

7.3 CCN and CCN-TV

Using a custom modified ccnSim simulator [87, 93], itself a modified version of the Omnet++ network simulator [3], we simulated a network, based on the structure of the GEANT network, composed of CCN-TV nodes (meaning no TCP or UDP encapsulation) with one video streaming server offering 5 different streams with specific video parameters and a variable number on clients of two types *faithful*, sticking to one channel, and *zapping*, moving from one channel to another with a given frequency.

Routing Tables, handled by ccnSim, are always shortest path optimal and we tested with three different caching strategies for the intermediate nodes *no cache*, *LRU* and *FIFO* and a cache size of 100 chunks.

The results permitted us to show that the approach is working and to evaluate the behavior at the variation of the sliding window timeout for the clients and its impact on the Quality of Service by measuring chunk-loss. We also found that in

the presence of real-time streams the cache actually reduces the chunk-loss only to a small amount, owning to the PIT structure already distributing multiple request over a single reply, and a very minute amount of subsequent requests for the same content.

Confronted with a classical CCN approach the CCN-TV alternative of not sending *Interest*s for every *Data* allows a sensible amount of bandwidth saving, leading to better performances and user experience.

In this work, the effectiveness of TV services in a CCN has been investigated. To this end, the *ccnSim* simulator has been modified to add several relevant features such as window-based flow control, handling of playout delay and real-time data, advanced logging mechanisms, and data session bootstrapping.

Preliminary results reported herein clearly show that the most relevant CCN feature to TV services is the management of *Interest* packets through the PIT data structure. In fact, such a mechanism limits the number of requests for the same chunk at the server side for multiple clients watching the same TV channel, thus decreasing the link and the computational load at the server.

Further research will explore: wider scenarios with many users and available channels, advanced optimization techniques for TV services in CCN (including routing and congestion control), and the adoption of scalable video coding. Moreover, a more deeply investigation on the relevance of the cache and the PIT in live TV services will be also conducted.

The CCN protocol, while facing many problems to be solved before becoming a widespread general network protocol (the promised Future Internet Protocol), does indeed offers many advantages as presented in this work.
Future works in the real-time video streaming field will focus on better understanding the behavior of larger networks with heterogeneous characteristics (mobility of nodes, link speeds and latency, ...) and multiple simultaneous video streams.

7.4 Rateless coding over mobile CCN

All the previous approaches examine a single solution to the wider problem of increasing robustness and resilience in a distributed decentralized scenario, either by employing particular encoding not penalized by lack of structural informations or message losses, by exploiting pollution detection techniques suited for a completely decentralized and inherently non trusted network, and by more efficient routing techniques based on contents rather than addresses.

Uniting those techniques together, especially introducing mobility to the CCN paradigm, and studying their perks and flaws is indeed an interesting and very applicable evolution, one that we present here as an initial study in such direction.

We called our approach "Rateless coding over mobile CCN" as a mean to define an open protocol to let use define and analyze the interaction between mobility, particular encodings and the CCN protocol.

As pointed out, decentralized ad-hoc mobile networks are of growing importance and interest nowadays. Yet the troubles they face, both in performance and robustness, requires innovative approaches in order to overcome what are currently the most central obstacles to wide adoption and deployment, as for example in urban traffic information scenarios or rural spontaneous network formation where infrastructures are lacking.

Introducing the element of mobility in a communication network greatly complicates the strategies used to ensure a reliable communication with satisfying performances as it directly reduces to ability to create a durable routing tree and trust in the neighbor nodes.

If the mobility also comprises from seldom moving slow nodes like hand-held laptops to continuously unpredictable fast moving nodes, such as vehicles then we can see that many wireless mesh protocols gradually becomes less and less efficient by losing their ability to grow a continuos path from source to destination, even if for the duration of a single transmission.

Promising old and new technologies like network coding and decentralized polluters detection has been shown in Chapter 5 to be robust and efficient, especially given the specific constraints of the scenario, while other like CCN can relay

more on the contents rather than the inherently abstract address informations, as shown in Chapter 6.

Considering CCNs/NDNs the concept of introducing an underlying mobility aspect has been present since their conception [94].

Various relevant work has already been done on pairing mobility with CCNs, both as general application study [95], or specifically in MANETs [96, 97].

It is generally accepted that CCNs are well suited for mobile environments, needing few, if any, modification, and very resilient based on their focus on nodes IDs rather than addresses. For real-time, low latency applications however some structural decisions must be evaluated carefully during implementation, as they can impact heavily on the resulting performances.

Regarding coupling network coding with CCNs there are a few works available, all focusing on the central idea of augmenting data availability by multiple seeding from different sources [98, 99, 100]. All these approaches are shown to be valid and a substantial increase in performances over the non-coded case, yet to the best of our knowledge none of them considers a mobile topology, which could heavily influence the obtained results.

Focusing on pollution detection and malicious nodes attacks to the best of our knowledge the current literature is a lot smaller in size, concentrating on specific attacks similar to the IP-based ones [101, 102] or more inherent to the CCN protocol structure [103]. All of those approaches focuses on directly identifying corrupted data once received, which will not be usually possible using rateless coding, rendering them unapplicable in our case, at least without modifications.

We are analyzing, via a simulator, the behaviors and performances of different common ad-hoc routing protocols [104], specifically OLSR, DSDV, AODV, and a custom flooding based one, over a mobile nodes wireless network, while they are exchanging data using the CCN protocol of *Interest*/*Data* to request and receive packets encoded using *network coding*, via our FountainApplication module for the ns-3 simulator.

We are also implementing different scenarios, varying in the ratio of slow and

fast moving nodes, the number of different sources presents, as well as the caching and rebroadcasting policies adopted, to better identify strong and weak points of each routing protocol when underlying the CCN + coding protocol.

Finally we plan to briefly analyze the detection of malicious polluter nodes in our set using our SIEVE protocol, in a completely distributed and accurate way.

Preliminary results on the topic are positive, where the CCN model is confirmed to be well suited in distributed mobile environments, and initial implementation of network coding can greatly increase the data throughput, even in very sparse and fast moving scenarios.

Yet further work is still needed, and ongoing, to better evaluate the interaction between those aspects and the parameters influencing coding behaviors to avoid degraded performances or excessive noise.

Part IV

Appendix

Appendix A

Tools used

A.1 HSO - Heterogeneous Statistical Optimizer

- **Tool name:** HSO
- **Tool authors:** R. Loti, and R. Gaeta
- **License:** (Currently not released to the public)
- **Website:** Software available on request

To calculate the probability functions generating the the statistical model for network interconnection we have developed a dedicated software, called HSO (Heterogeneous Statistical Optimizer).

The tool, evolution of a simpler, also internally developed, nameless tool specific to the homogeneous networks case, takes as parameters a json formatted configuration file, the results file name and a few optional parameters regarding the specific variation of the mathematical model to be used, the verbosity of the console output and how to manage missing values in the given configuration file. Following is the help text showing the usage pattern:

```
Usage: ./HSO -c <conf_file> -o <result_file> [<param> <param_value>]
  -c or --config:  the json configuration file
  -o or --output:  the result output file
```

```
-m or --model:   the statistical model to use, 0 for independent,
                 1 for independent global, 2 for dependent [Default: 2]
-v or --verbose: a flag for having verbose output [Default: false]
-t or --test:    a flag for initiating selftest [Default:false]
-n or --noderiv: a flag for reading all parameters from the json file,
                 not calculating them as standard [Default: false]
```

The tool has been realized to read all the parameters regarding the instance from the json file, such as the number of distinct networks, the TTL values to be calculated, the α_i resource distribution, $C_{i,j}$ fraction of nodes that also belong to network j, the $P_{i,k}$ distribution for the number of connection of each node, and $pf_{t,i,j}$ or $pf_{i,j}$, depending if it varies with the TTL t or not, probability that a node will forward a message from network i to j, for each network i.

With the optimizer, so called as it permits to sweep the space of all parameters permutations in a feasible time in respect to a simulator and so to choose the optimal values range, an entourage of related small tools has been developed, such a configuration generator, able to generate ranges of json file iteratively, sets of shell scripts to automate as much as possible the "generate, calculate and parse" standard iter and also helping parse and group the obtained result files, and the relative tools and file format documentation.

A.2 Network simulator ns-3

- **Tool name:** ns-3
- **Tool authors:** M. Lacage, and various authors
- **License:** GNU GPLv2
- **Website:** http://www.nsnam.org

ns-3 is a discrete-event network simulator, targeted primarily for research and educational use.

The software is a C++ library which provides a set of network simulation models implemented as C++ objects and wrapped through Python.
Users normally interact with this library by writing a C++ or a Python application which instantiates a set of simulation models to set up the simulation scenario of interest, enters the simulation "main loop", and exits when the simulation is completed (i.e. no more events are planned in the schedule).

A through and detailed description of the various APIs is present on the official site, togheter with a very helpful tutorial for initially understanding the basic internal organization of the modules.

A.2.1 The FountainApplication application

- **Tool name:** FountainApplication (ns-3 module)
- **Tool authors:** R. Loti
- **License:** GNU GPLv2 (Currently not released to the public)
- **Website:** Software available on request

Over the ns-3 models system we wrote the FountainApplication module, as an Application in ns-3 terms.

A Node with a FountainApplication configured on it can, depending on the parameters passed, work both as client and server for receiving and decoding, and encoding and sending chunks, using a specified decoder:

- **0 (Fake):** no actual decoding happens, the payload is considered decoded after $n + \epsilon$ packets have been received, where the overhead ϵ is specified based on statistical patterns relative to n;
- **1 (Linear):** only packets of degree $d = 1$, so containing an uncodified block in the payload, are considered, waiting for one of each of the n blocks. Equivalent to a standard n-blocks sequential transmission;
- **2 (Belief Propagation):** the packets payload is decoded using the Belief Propagation algorithm, less efficient than the following, but also less computationally intensive;
- **3 (OFG):** the packets payload is decoded using the very efficient, but computationally intensive On-The-Fly Gaussian Elimination algorithm, based on the simpler, yet less flexible GE (Gaussian Elimination) algorithm;

or encoder:

- **0 (Fake):** no actual encoding happens and so the payload is actually of a specified size, but empty, only populating the header. Usually used in conjunction with the *fake* decoder to avoid loading the simulator with big quantities of data;

- **1 (Sequential):** only packets of degree $d = 1$ are generated, while the block order is random and not guaranteed to be sequential or non repeating before a full n blocks transmission is functionally similar to a standard, sequential data transmission;
- **2 (LT Code):** standard encoding for LT Codes, using a RSD distribution a random degree $1 \leq d \leq n$ is chosen and as many distinct blocks in n are randomly selected to be *xor*-encoded in the payload.

A.2.2 The SieveApplication application

- **Tool name:** SieveApplication (ns-3 module)
- **Tool authors:** R. Loti
- **License:** GNU GPLv2 (Currently not released to the public)
- **Website:** Software available on request

In parallel with FountainApplication, but independent from it, we developed the SieveApplication module, also as a ns-3 Application type.

The application access the methods of the FountainApplication class to retrieve received and decoding statistics from each Node that has the application installed to pass to the SIEVE algorithm to progressively build the probable polluter list.

A.2.3 The ndnSIM module

- **Tool name:** ndnSIM (ns-3 module)
- **Tool authors:** A. Afanasyev, I. Moiseenko, and L. Zhang
- **License:** GNU GPLv2
- **Website:** http://ndnsim.net

The ndnSIM is a ns-3 module that implements the Named Data Networking (NDN) communication model [105].

ndnSIM is specially optimized for simulation purposes and has a cleaner and more extensible internal structure comparing to the existing NDN implementation (NDNx) as it works internally to ns-3, not as an external library as the DCE-ccnx module (a similar module for ns-3 in which a software "cradle" integrates the real ccnx library, see Section A.3), and in which every part of the protocol is independent and implemented as a distinct, and easily modifiable, submodule.

A.3 The CCNx protocol and library

- **Tool name:** CCNx
- **Tool authors:** V. Jacobson, and various authors
- **License:** GNU GPLv2, and GNU LGPLv2
- **Website:** https://www.ccnx.org

The CCNx protocol is the transport protocol for a communications architecture called Content-Centric Networking (CCN) built on named data.

CCN has no notion of host at its lowest level as a packet address names content, not location. The CCNx protocol efficiently delivers named content rather than connecting hosts to other hosts.
Every packet of data may be cached at any CCNx router which combined with intrinsic support for multicast or broadcast delivery this leads to a very efficient use of the network when many people are interested in the same content.

A.3.1 The ccnSim simulator

- **Tool name:** ccnSim
- **Tool authors:** R. Chiocchetti, D. Rossi, and G. Rossini
- **License:** GNU GPLv3
- **Website:** http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.CcnSim

ccnSim is a scalable chunk-level simulator of Content Centric Networks (CCN), written in C++ using the Omnet++ simulator as a base, which allows to assess CCN performance in scenarios with large orders of magnitude for CCN content stores and Internet catalog sizes, due to the focus on higher level workings than, for example, ns-3 [106].

In our works on CCN-TV we extended the ccnSim simulator to include our extensions of the protocol needed for live-tv streaming, but we maintained the efficient definition and mechanisms for routing and network structure.

References

[1] M. Newman, *Networks: an introduction*. Oxford University Press, 2010.

[2] I. Martinez-Yelmo, R. Cuevas, C. Guerrero, and A. Mauthe, "Routing performance in a hierarchical dht-based overlay network," in *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pp. 508–515, IEEE, 2008.

[3] VV.AA., "Omnet++ home page." [Online] Available: http://www.omnetpp.org/, 2014.

[4] V. Ciancaglini, R. Gaeta, R. Loti, and L. Liquori, "Interconnection of large scale unstructured p2p networks: Modeling and analysis," in *Analytical and Stochastic Modeling Techniques and Applications*, pp. 183–197, Springer, 2013.

[5] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pp. 271–280, 2002.

[6] R. Gaeta, M. Grangetto, and R. Loti, "Sieve: a distributed, accurate, and robust technique to identify malicious nodes in data dissemination on manet," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pp. 331–338, IEEE, 2012.

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

[8] R. Gaeta, R. Loti, and V. Ciancaglini, "Modeling and analysis of large scale interconnected unstructured p2p networks," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pp. 710–711, IEEE, 2012.

[9] R. Gaeta, M. Grangetto, and R. Loti, "Exploiting rateless codes and belief propagation to infer identity of polluters in manet," *IEEE Transactions on Mobile Computing*, 2013.

[10] V. Ciancaglini, G. Piro, R. Loti, L. A. Grieco, and L. Liquori, "Ccn-tv: a data-centric approach to real-time video services," *In Proc. of IEEE AINA*, 2013.

[11] G. Piro, V. Ciancaglini, R. Loti, L. A. Grieco, L. Liquori, *et al.*, "Providing crowd-sourced and real-time media services through a ndn-based platform," *MODELLING AND PROCEESING FOR NEXT GENERATION BIG DATA TECHNOLOGIES AND APPLICATIONS*, 2013.

[12] M. Siekkinen, V. Goebel, T. Plagemann, K. Skevik, M. Banfield, and I. Brusic, "Beyond the Future Internet–Requirements of Autonomic Networking Architectures to Address Long Term Future Networking Challenges," in *International Workshop on Future Trends of Distributed Computing Systems*, pp. 89–98, 2007.

[13] A. Fonte, M. Curado, and E. Monteiro, "Interdomain Quality of Service Routing: Setting the Grounds for the Way Ahead," *Annals of Telecommunications*, vol. 63, no. 11, pp. 683–695, 2008.

[14] M. Kwon and S. Fahmy, "Synergy: an Overlay Internetworking Architecture," in *Proc. of International Conference on Computer Communications and Networks*, pp. 401–406, 2005.

[15] K. Junjiro, W. Naoki, and M. Masayuki, "Design and Evaluation of a Cooperative Mechanism for Pure P2P File-Sharing Networks," *IEICE Trans Commun (Inst Electron Inf Commun Eng)*, vol. E89-B, no. 9, pp. 2319–2326, 2006.

[16] P. Furtado, "Multiple Dynamic Overlay Communities and Inter-space Routing," *Lecture Notes in Computer Science*, vol. 4125, pp. 38–49, 2007.

[17] B. Maniymaran, M. Bertier, and A. Kermarrec, "Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks," in *Proc. of IEEE ICDCS'07*, pp. 33–40, 2007.

[18] F. Buccafurri, V. D. Foti, G. Lax, A. Nocera, and D. Ursino, "Bridge analysis in a social internetworking scenario," *Information Sciences*, vol. 224, pp. 1–18, 2013.

[19] A. Shokrollahi, "Raptor codes," *Information Theory, IEEE Transactions on*, vol. 52, no. 6, pp. 2551–2567, 2006.

[20] M. Grangetto, R. Gaeta, and M. Sereno, "Rateless codes network coding for simple and efficient p2p video streaming," in *Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on*, pp. 1500–1503, IEEE, 2009.

[21] A. Magnetto, S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno, "Fountains vs torrents: the p2p toroverde protocol," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pp. 417–420, IEEE, 2010.

[22] S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno, "Bittorrent and fountain codes: friends or foes?," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–8, IEEE, 2010.

[23] A. Fiandrotti, V. Bioglio, M. Grangetto, R. Gaeta, and E. Magli, "Band codes for energy-efficient network coding with application to p2p mobile streaming," 2013.

[24] B. Wu, J. Chen, J. Wu, and M. Cardei, "A survey of attacks and countermeasures in mobile ad hoc networks," in *Wireless Network Security* (Y. Xiao, X. S. Shen, and D.-Z. Du, eds.), Signals and Communication Technology, pp. 103–135, Springer US, 2007.

[25] H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang, "Security in mobile ad hoc networks: challenges and solutions," *IEEE Wireless Communications*, vol. 11, no. 1, pp. 38–47, 2004.

[26] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," *IEEE Symposium on Security and Privacy*, 2004.

[27] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE INFOCOM 2006*, 2006.

[28] Q. Li, D.-M. Chiu, and J. Lui, "On the practical and security issues of batch content distribution via network coding," in *14th IEEE International Conference on Network Protocols, ICNP '06.*, 2006.

[29] D. Kamal, D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *40th Annual Conference on Information Sciences and Systems, 2006*, 2006.

[30] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient signature-based scheme for securing network coding against pollution attacks," in *IEEE INFOCOM 2008*, 2008.

[31] E. Kehdi and B. Li, "Null keys: Limiting malicious attacks via null space properties of network coding," in *IEEE INFOCOM 2009*, 2009.

[32] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient scheme for securing xor network coding against pollution attacks," in *IEEE INFOCOM 2009*, 2009.

[33] S. Agrawal and D. Boneh, "Homomorphic macs: Mac-based integrity for network coding," in *Applied Cryptography and Network Security*, pp. 292–305, Springer, 2009.

[34] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks," in *Proceedings of the second ACM conference on Wireless network security*, pp. 111–122, ACM, 2009.

[35] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "Ripple authentication for network coding," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[36] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks with random network coding," *IEEE Transactions on Information Theory*, vol. 54, pp. 2798 –2803, june 2008.

[37] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros, "Resilient network coding in the presence of byzantine adversaries," *IEEE Transactions on Information Theory*, vol. 54, pp. 2596 –2603, june 2008.

[38] R. Koetter and F. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Transactions on Information Theory*, vol. 54, pp. 3579 –3591, august 2008.

[39] A. Newell and C. Nita-Rotaru, "Split null keys: A null space based defense for pollution attacks in wireless network coding," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, pp. 479–487, 2012.

[40] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "MIS: Malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *IEEE INFOCOM 2010*, pp. 1–5, march 2010.

[41] Y. Li and J. C. Lui, "Stochastic analysis of a randomized detection algorithm for pollution attack in P2P live streaming systems," *Performance Evaluation*, vol. 67, no. 11, pp. 1273 – 1288, 2010.

[42] Y. Li and J. Lui, "Identifying pollution attackers in network-coding enabled wireless mesh networks," in *20th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1 –6, aug 2011.

[43] Y. Li and J. Lui, "Epidemic attacks in network-coding enabled wireless mesh networks: Detection, identification and evaluation," *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, p. 1, 2012.

[44] X. Jin and S.-H. G. Chan, "Detecting malicious nodes in peer-to-peer streaming by peer-based monitoring," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 6, pp. 9:1–9:18, March 2010.

[45] R. Gaeta, M. Grangetto, L. Bovio, *et al.*, "Dip: Distributed identification of polluters in p2p live streaming," *ACM Transaction Multimedia Computing*, 2014.

[46] R. Gaeta and M. Grangetto, "Identification of malicious nodes in peer-to-peer streaming: A belief propagation-based technique," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 10, pp. 1994–2003, 2013.

[47] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[48] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," *RFC4838, April*, 2007.

[49] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi, "A survey on content-oriented networking for efficient content delivery," *Communications Magazine, IEEE*, vol. 49, no. 3, pp. 121–127, 2011.

[50] A. Baid, T. Vu, and D. Raychaudhuri, "Comparing alternative approaches for networking of named objects in the future internet," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pp. 298–303, IEEE, 2012.

[51] VV.AA., "Ccnx home page." [Online] Available: https://www.ccnx.org/, 2014.

[52] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini, and B. Marinkovic, "Synapse: A scalable protocol for interconnecting heterogeneous overlay networks," in *Proc. of Networking '10*, pp. 67–82, 2010.

[53] M. Luby, "LT codes," in *43rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2002*, pp. 271–280, 2002.

[54] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Phys. Rev. E*, vol. 64, no. 2, p. 026118, 2001.

[55] R. Bolla, R. Gaeta, A. Magnetto, M. Sciuto, and M. Sereno, "A measurement study supporting p2p file-sharing community models," *Computer Networks*, vol. 53, no. 4, pp. 485–500, 2009.

[56] R. Gaeta and M. Sereno, "Generalized probabilistic flooding in unstructured peer-to-peer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2055–2062, 2011.

[57] F. Viger and M. Latapy, "Efficient and simple generation of random simple connected graphs with prescribed degree sequence," in *Computing and Combinatorics*, vol. 3595 of *Lecture Notes in Computer Science*, pp. 440–449, Springer Berlin / Heidelberg, 2005.

[58] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[59] D. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[60] J. Yedidia, W. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282 – 2312, 2005.

[61] J. Yedidia, W. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," in *Exploring Artificial Intelligence in the New Millennium*, Science & Technology Books, ch. 8, Elsevier, 2003.

[62] T. Schierl, S. Johansen, A. Perkis, and T. Wiegand, "Rateless scalable video coding for overlay multisource streaming in manets," *Journal of Visual Communication and Image Representation*, vol. 19, no. 8, pp. 500–507, 2008.

[63] V. R. Syrotiuk, C. J. Colbourn, and S. Yellamraju, "Rateless forward error correction for topology-transparent scheduling," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 464–472, 2008.

[64] R. Gaeta, M. Grangetto, and R. Loti, "SIEVE: a distributed, accurate, and robust technique to identify malicious nodes in data dissemination on manet," in *IEEE ICPADS 2012*, pp. 331–338, 2012.

[65] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge: M.I.T. Press, 1963.

[66] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *International Journal of Computer Vision*, vol. 40, pp. 25–47, 2000.

[67] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, pp. 585 –598, feb 2001.

[68] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*, pp. 15–34, Springer Berlin Heidelberg, 2010.

[69] C. Systems, "Cisco visual networking index: Forecast and methodology, 2010-2015." White Paper, Jun. 2011.

[70] B. Ahlgren, P. A. Aranda, P. Chemouil, S. Oueslati, L. M. Correia, H. Karl, M. Sollner, and A. Welin, "Content, connectivity, and cloud: ingredients for the network of the future," *IEEE Commun. Mag.*, vol. 49, Jul. 2011.

[71] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM CoNEXT '09*, 2009.

[72] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Int. Teletraffic Congress, (ITC)*, 2011.

[73] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *ACM SIGCOMM workshop on Information-centric networking (ICN '11)*, 2011.

[74] M. Varvello, I. Rimac, U. Lee, L. Greenwald, and V. Hilt, "On the design of content-centric manets," in *Int. Conf. on Wireless On-Demand Network Systems and Services, (WONS)*, Jan. 2011.

[75] G. Carofiglio, V. Gehlen, and D. Perino, "Experimental evaluation of memory management in content-centric networking," in *IEEE ICC*, 2011.

[76] M. Tortelli, I. Cianci, L. A. Grieco, G. Boggia, and P. Camarda, "A fairness analysis of content centric networks," in *Proc. of Int. Conf. on Network of the Future, NOF*, (Paris, France), Nov. 2011.

[77] D. Rossi and G. Rossini, "On sizing CCN content stores by exploiting topological information," in *IEEE INFOCOM, NOMEN Worshop*, 2012.

[78] L. A. Grieco, D. Saucez, and C. Barakat, "AIMD and CCN: past and novel acronyms working together in the Future Internet," in *Capacity Sharing Workshop 2012 (CSWS'12) co-located with ACM SIGCOMM CoNEXT 2012.*, Dec. 2012.

[79] G. Carofiglio, M. Gallo, and L. Muscariello, "Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks," in *ACM SIGCOMM, ICNí12 workshop.*, 2012.

[80] M. Tortelli, L. A. Grieco, and G. Boggia, "CCN forwarding engine based on bloom filters," in *Proc. of ACM Int. Conf. on Future Internet Technologies, CFI*, (Seoul, Korea), Sep. 2012.

[81] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "Dipit: A distributed bloom-filter based pit table for ccn nodes," in *Computer Commu-*

nications and Networks (ICCCN), 2012 21st International Conference on, pp. 1 –7, 30 2012-aug. 2 2012.

[82] Z. Li and G. Simon, "Time-shifted TV in content centric networks:the case for cooperative in-network caching," in *Proc. of IEEE ICC*, Jun. 2011.

[83] H. Xu, Z. Chen, R. Chen, and J. Cao, "Live streaming with content centric networking," in *Proc. 3rd Int. Conf. on Networking and Distributed Computing, Hangzhou, China, 2012.*, 2012.

[84] G. Rossini and D. Rossi, "Large scale simulation of ccn networks," in *In Algotel 2012*, 2012.

[85] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, pp. 26 –36, july 2012.

[86] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, pp. 560 –576, Jul. 2003.

[87] VV.AA., "Ccn-tv home page." [Online] Available: http://telematics.poliba.it/ccn-tv/, 2014.

[88] VV.AA., "Geant project website." [Online] Available: http://www.geant.net/, 2014.

[89] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," in *Technical report, Telecom ParisTech*, 2011.

[90] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, 6th ed., 2012.

[91] G. Piro, L. Grieco, G. Boggia, R. Fortuna, and P. Camarda, "Two-level Downlink Scheduling for Real-Time Multimedia Services in LTE Networks," in *IEEE Trans. Multimedia*, vol. 13, pp. 1052 – 1065, Oct. 2011.

[92] J. Ohm, *Multimedia Communication Technology*. Springer, USA, 2004.

[93] VV.AA., "ccnsim home page." [Online] Available: http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.CcnSim, 2014.

[94] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[95] G. Tyson, N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe, "A survey of mobility in information-centric networks," *Communications of the ACM*, vol. 56, no. 12, pp. 90–98, 2013.

[96] Z. Zhu, A. Afanasyev, and L. Zhang, "A new perspective on mobility support," tech. rep., Technical report, Named-Data Networking Project, 2013.

[97] Y. Lu, X. Li, Y.-T. Yu, and M. Gerla, "Information-centric delay-tolerant mobile ad-hoc networks," *INFOCOM 2014, Workshop on Name Oriented Mobility*, 2014.

[98] W.-X. Liu, S.-Z. Yu, and P.-Y. Zhu, "Multisource dissemination in content-centric networking," in *Network of the Future (NOF), 2013 Fourth International Conference on the*, pp. 1–5, IEEE, 2013.

[99] S. Miyake and H. Asaeda, "Network coding and its application to content centric networking," *WITMSE 2013*, 2013.

[100] Q. Wu, Z. Li, and G. Xie, "Codingcache: multipath-aware ccn cache with network coding," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pp. 41–42, ACM, 2013.

[101] K. Wang, H. Zhou, J. Chen, and Y. Qin, "Rdai: Router-based data aggregates identification mechanism for named data networking," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, pp. 116–121, IEEE, 2013.

[102] K. Wang, J. Chen, H. Zhou, Y. Qin, and H. Zhang, "Modeling denial-of-service against pending interest table in named data networking," *International Journal of Communication Systems*, 2013.

[103] M. Conti, P. Gasti, and M. Teoli, "A lightweight mechanism for detection of cache pollution attacks in named data networking," *Computer Networks*, vol. 57, no. 16, pp. 3178–3191, 2013.

[104] S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, "A performance comparison study of ad hoc wireless multicast protocols," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 565–574, IEEE, 2000.

[105] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," Technical Report NDN-0005, NDN, October 2012.

[106] E. Weingartner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *Communications, 2009. ICC'09. IEEE International Conference on*, pp. 1–5, IEEE, 2009.