Quelques défis posés par l'utilisation de protocoles de Gossip dans l'Internet

Alessio Pace

▶ **To cite this version:**

Alessio Pace. Quelques défis posés par l'utilisation de protocoles de Gossip dans l'Internet. Autre [cs.OH]. Université de Grenoble, 2011. Français. NNT : 2011GRENM046 . tel-00636386

HAL Id: tel-00636386

https://theses.hal.science/tel-00636386

Submitted on 27 Oct 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Alessio Pace

Thèse dirigée par **Jean-Bernard Stefani**
et codirigée par **Vivien Quéma**

préparée au sein de **Inria Grenoble - Rhône-Alpes**
et de **Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Quelques defis posés par l'utilisation de protocoles de Gossip dans l'Internet

Gossiping in the wild – Tackling practical problems faced by gossip protocols when deployed on the Internet

Thèse soutenue publiquement le **4 octobre 2011**,
devant le jury composé de :

Dr. Anne-Marie Kermarrec
Inria Rennes, Présidente
Prof. Rachid Guerraoui
EPFL, Rapporteur
Prof. Pascal Felber
Université de Neuchâtel, Rapporteur
M. Jean-Bernard Stefani
Inria Grenoble - Rhône-Alpes, Directeur de thèse
Prof. Vivien Quéma
INP Grenoble, Co-Directeur de thèse

Abstract

Peer-to-peer (P2P) systems are very popular today. Their usage goes from instant messaging to file sharing, from distributed backup and storage to even live-video streaming. Among P2P protocols, gossip-based protocols are a family of protocols which have been the object of several research works in the last decade. The reasons behind the interest in gossip-based protocols are that they are considered robust, easy to implement, and that they have interesting scalability properties. They are then appealing candidates for implementing dynamic and large-scale distributed systems.

This thesis tackles two problems faced by gossip-based protocols when deployed on a practical scenario as the Internet. The first problem is coping with Network Address Translators (NATs) in the context of gossip-based peer sampling protocols. These protocols make the assumption that, at any moment, each node is able to communicate with any other node of the network. This assumption is false when some nodes use NATs. We present Nylon, a peer sampling protocol which works despite the presence of NATs. Nylon introduces a low overhead to cope with NATs and fairly balances this overhead among nodes using a NAT and those which do not.

The second problem that we study is the possibility to limit the dissemination of "spam" messages in gossip-based dissemination protocols. These protocols are in fact ideal vectors to spread spam messages due to the fact that there is no central authority in charge of filtering messages based on their content. We propose FireSpam, a gossip-based dissemination protocol which allows limiting the dissemination of "spam" messages. FireSpam implements a decentralized filtering mechanism (each node participates to the filtering). Moreover, it works despite the presence of a fraction of malicious nodes (also called "Byzantine" nodes) and despite the presence of so called "rational" nodes (also called "selfish" nodes). These latters are willing to deviate from the protocol if they have an interest in doing so.

Keywords. peer-to-peer (P2P) systems, gossip-based protocols, peer sampling service, Network Address Translators (NATs), large-scale information dissemination, fault tolerance.

Résumé

Les systèmes pair-à-pair (P2P) sont aujourd'hui très populaires. Leur utilisation va de la messagerie instantanée au partage de fichiers, en passant par la sauvegarde et le stockage distribué ou encore le streaming video. Parmi les protocoles P2P, les protocoles basés sur le "gossip" sont une famille de protocoles qui a fait l'objet de nombreux travaux de recherche durant la dernière décennie. Les raisons de l'engouement pour les protocoles basés sur le "gossip" sont qu'ils sont considérés robustes, faciles à mettre en oeuvre et qu'ils ont des propriétés de passage à l'échelle intéressantes. Ce sont donc des candidats intéressants dès lors qu'il s'agit de réaliser des systèmes distribués dynamiques à large échelle.

Cette thèse considère deux problématiques rencontrées lorsque l'on déploie des protocoles basé sur le "gossip" dans un environnement réel comme l'Internet. La première problématique est la prise en compte des pare-feux (NAT) dans le cadre des protocoles d'échantillonnage basés sur le "gossip". Ces protocoles font l'hypothèse que, a tout moment, chaque noeud est capable de communiquer avec n'importe quel noeud du réseau. Cette hypothèse est fausse dès lors que certains noeuds utilisent des NAT. Nous présentons Nylon, un protocole d'échantillonnage qui fonctionne malgré la présence de NAT. Nylon introduit un faible surcoût pour gérer les NAT et partage équitablement ce surcoût entre les noeuds possédant un NAT et les autres noeuds.

La deuxième problématique que nous étudions est la possibilité de limiter la dissémination de messages de type "spam" dans les protocoles de dissémination basés sur le "gossip". Ces protocoles sont en effet des vecteurs idéaux pour diffuser les messages de type "spam" du fait qu'il n'y a pas d'autorité de contrôle permettant de filtrer les messages basés sur leur contenu. Nous proposons FireSpam, un protocole de dissémination basé sur le "gossip" qui permet de limiter la diffusion des messages de type "spam". FireSpam fonctionne par filtrage décentralisé (chaque noeud participe au filtrage). Par ailleurs, il fonctionne malgré la présence d'une fraction de noeuds malicieux (aussi appelés "Byzantins") et malgré la présence de noeuds dits “rationnels” (aussi appelés "égoïstes"). Ces derniers sont prêts à dévier du protocole s'ils ont un intérêt à le faire.

Mots-clés. systèmes pair-à-pair (P2P), protocoles basé sur le "gossip", service d' échantillonnage, pare-feux (NAT), dissémination d'information à large échelle, tolérance aux fautes.

Acknowledgments

First of all I would like to thank Anne-Marie Kermarrec for being president in my jury, and Pascal Felber and Rachid Guerraoui for accepting to review this document.

A special thanks to Vivien Quéma for having been a guide during these years. It has been a real pleasure to work with him.

I would like to thank Jean-Bernard Stefani for the possibility to work in his team, and for giving me freedom in the research activities.

Thanks also to all the people I collaborated with during these last years: Anne-Marie Kermarrec, Valerio Schiavoni, Sonia Ben Mokhtar, Roberto Baldoni, Leonardo Querzoni, Arshad Jhumka.

I would like to thank all the current and former members of the SARDES team that I had the pleasure to interact with, and who really contributed to make me enjoy the time I spent in the team. In particular I would like to thank Renaud for always being helpful and supportive, Willy for being a great office mate and for sharing many P2P discussions, Claudio for being not only a team mate but also a good friend, Fabien for the amusing coffee times at the kfet of Inria, and all the other former or current floor mates: Pierre-Louis, Baptiste, Fabien, Sylvain, Gautier, Nicolas, Brice, Lionel.

Thanks to all the good friends I met in Grenoble, expecially the ones I spent most time with: Pierre, Salvatore, Davide, Simone, Daniele, Aurelie and Daniel.

I would like also to thank the good old times Roman friends, Daniele, Giulia and Roberto, for being good friends all along these years.

Finally, I really wish to thank my family, relatives and friends for supporting me all along these years, it really mattered to me to know I could count on you.

Preface

This thesis presents the research conducted in the SARDES team of the Inria Grenoble - Rhône-Alpes research institute to pursue the Ph.D. in the speciality "Informatics" from the Doctoral School "Mathématiques, Sciences et Technologies de l'Information, Informatique" of the Université de Grenoble. The research activities have been carried out under the supervision of Jean-Bernard Stefani and Vivien Quéma.

The thesis focuses on two problems faced by gossip-based protocols when deployed on a practical context as the Internet: *(i)* coping with Network Address Translators (NATs) in gossip-based peer sampling protocols, *(ii)* limiting the dissemination of "spam" messages in gossip-based dissemination protocols, while accounting for malicious and selfish behaviors of the nodes.

Novel contributions to tackle the aforementioned problems have been the content of the following published works:

- **NAT-resilient Gossip Peer Sampling**. Anne-Marie Kermarrec, Alessio Pace, Vivien Quéma, Valerio Schiavoni. *In Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*, June 2009.
- **FireSpam: Spam Resilient Gossiping in the BAR Model**. Sonia Ben Mokhtar, Alessio Pace, Vivien Quéma. *In Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems (SRDS)*, October 2010.

Common terminology

In the rest of this document, usually a single mostly used term is employed to refer to a given concept. Nevertheless, some alternative terms or expressions might be found sometimes. Table I provides a summary of the most common cases.

Concept	Common term used	Possible alternative term(s) used
A participant in a distributed system or (more specifically) in a peer-to-peer system	node	peer
Node arrivals and departures	churn	-
Full membership knowledge of the nodes in the system	global view	complete view
Partial membership knowledge of the nodes in the system	local view	partial view
The node returned by the peer sampling service, which is the node to gossip with	target node	destination node
The data type stored in a node's view, which references and contains info about a node's neighbor	(view) entry	(view) neighbor descriptor
A node arbitrarily or maliciously deviating from the protocol	Byzantine node	malicious node
A node deviating from the protocol to increase its net benefit from the participation in the system	rational node	selfish node

Table I – Common used terminology in this document.

Note: even if *node* remains the most frequently used term in the document to designate a participant in a distributed system, the term *peer* is preserved in naming the *Peer Sampling Service* (PSS), as this is in fact the name used in literature to call this service.

Contents

1

Introduction

Contents

Internet applications and services are used by millions of people every day. People might for instance participate and benefit from them using a desktop computer when they are at home, their tablets when they are in a meeting or traveling, or they can access them via their smartphones whey they are on the go. As common usage scenarios, we may, for example, list surfing the Web, catching up with a friend on an instant messaging chat application, watching a movie trailer, posting a picture when back from (or ever more, while being in) vacation on an online photo album, or changing the status on an online social network or micro-blogging service.

Albeit different in purpose, all of the above are examples of distributed systems. In each of them, some nodes of the network *provide* a service, and some other nodes *consume* this service and benefit from it. The Internet is the common communication channel which allows the nodes providing services (the *server* nodes) and the nodes consuming services (the *client* nodes) to be connected and interact.

Despite the simple concepts illustrated above, distributed systems are not as simple when dealing with their actual design, implementation and deployment. In fact, distributed systems must cope with the *dynamicity* of the environment: they must take into account the possible continuous arrival and departure, as well as the possible failure, of the nodes. For example, in an Internet TV broadcasting service, nodes could join or leave the channel at will and at any time. Distributed systems must also account for the growing or shrinking —possibly in a little amount of time— of the number of

nodes, as well as the overall volume of information generated or consumed by them. Still considering an Internet TV broadcasting service, a massive amount of nodes could join (resp., leave) the service approximately at the same moment, that is, around the announced start (resp., end) time of it. And this number of nodes could go from hundreds to thousands of nodes during the course of the broadcast, which means distributed systems must be *scalable*.

Given this scenario, the traditional *client/server* model shows soon its limitations. In fact, in such a model the server is the only node providing the service and, if it is not capable of doing so effectively, it becomes the bottleneck and single point of failure of the whole distributed system. Moreover, this centralized architecture, put aside the possible important costs to keep it up and running, does not leverage the resources of client nodes, which passively participate and benefit from the provided service.

1.1 Peer-to-peer systems

The *peer-to-peer* (P2P) model has consolidated in the last decade or so as an effective approach to deal with the dynamicity and scalability requirements of today's distributed systems. Here nodes act as *both* clients and servers, thus contributing to the service they participate in. This has led to the term *servent* (*serv*er and cli*ent*) [1] to indicate their twofold role (cfr. Figure 1.1b). Each node has then the same responsibility, and the resources shared with the other nodes to collaborate in the functioning of the distributed systems can be of different types: CPU cycles, network bandwidth, memory, disk storage, and so on.

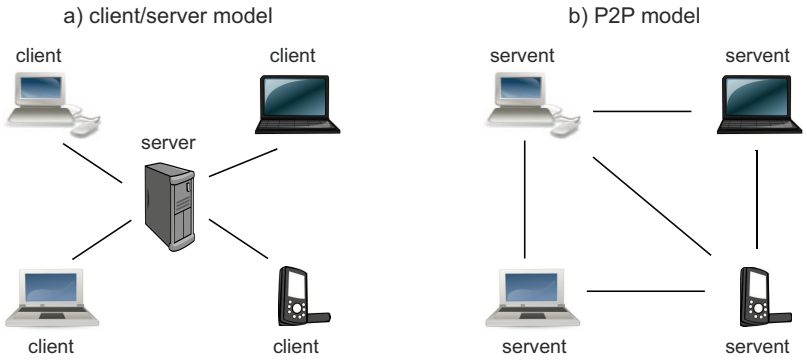


Figure 1.1 – (a) Client/server model: nodes acting only as clients; (b) P2P model: nodes acting as both clients and servers (*servents*).

To convey the popularity of P2P systems, a recent study conducted by Cisco [2] has shown how peer-to-peer file sharing alone accounts to roughly 25% of the US Internet traffic, slightly below the amount of traffic generated by streaming on video sites like YouTube [3] or Vimeo [4]. But P2P file sharing (nowadays mainly represented by the BitTorrent [5] protocol), is not the only widespread application of peer-to-peer. Other services range from applications like instant messaging (e.g., ICQ [6], Jabber [7]) to voice/video call (e.g., Skype [8]), from cooperative backup/storage (e.g., Wuala [9]) to video streaming (e.g., SopCast [10], PPLive [11], Veetle [12]) or even online social

networks (e.g., Diaspora [13]).

As illustrated in [14], regardless of the kind of specific application or service, peer-to-peer systems can be defined in terms of their *properties* and *characteristics*, and also in terms of how nodes are linked to each other forming the underlying communication graph, which is usually referred to as the *overlay network*.

Properties and characteristics. We list here a few distinguishing properties and characteristics of peer-to-peer systems, as recently presented in a survey by Rodrigues and Druschel [14].

High degree of decentralization. Nodes act as both clients and servers, so that the state and the work of the distributed system are spread on the nodes. They in fact contribute to the computing power, network bandwidth and disk storage capacities of the system. "Pure" (fully decentralized) peer-to-peer systems have no dedicated nodes keeping a centralized state, while other systems might allocate a few nodes in charge of it. While this might simplify certain tasks (e.g., keeping an index of live nodes or of the resources shared by them), it falls shortly into similar limitations as those of the aforementioned client/server model.

Self-organization. When a node wants to enter the system, very little (or even no) configuration is required to make the node join and to have the system adjust accordingly. Usually in fact the joining node needs to know only the address of one or more nodes which are already part of the system.

Multiple administrative domains. Nodes do not usually belong to the same organization. Instead, they are typically individual nodes joining the system at their will.

Low barrier to deployment. Differently from the client/server model, P2P systems do not demand a dedicated infrastructure, thus reducing the initial costs to run a service.

Organic growth. Peer-to-peer systems can grow almost with unbounded limit, and do not require a suspension/replacement/upgrade/restart of the infrastructure to cope with increased node population.

Resilience to faults and attacks. Given the fact that there is generally no single central node in peer-to-peer systems, they are by nature more resilient to faults and attacks than centralized systems. To degrade the quality of a P2P system, an attacker might have to target or compromise a significant fraction of the nodes.

Abundance and diversity of resources. Only few organizations in the world can dispose of the amount of resources that popular P2P systems have. Furthermore, these resources differ in hardware and software architecture, network capabilities, storage capacity, geographical location and so on.

Moreover, P2P systems have to ensure the properties of the system as a whole despite the fact that typically nodes only have a small local knowledge of the entire system. To ensure the desired system properties, nodes then communicate and collaborate via a distributed protocol which should not make assumptions on the (well) behavior of nodes or on the reliability of communications.

Operating principles: overlay networks. We have previously discussed how *fully decentralized* P2P systems provide appealing scalability properties. They can be defined in terms of the P2P *overlay network* which characterizes them. An overlay network is a logical network sitting on top of another network (e.g., the IP network), connecting the nodes of a P2P system. In the simplest scenarios, two nodes connected through the same overlay can communicate between each other via classical IP routing mechanisms. In the context of overlays, such two nodes are usually called *neighbors*. Neighbors relationship might not necessarily be symmetric: a node might know the existence and address of another node, while this latter being unaware of the first. This is why the term *view* is usually employed to indicate the current set of nodes known or stored as neighbors by a given node. The actual topology of an overlay network is then determined by considering the neighbors relationships among the views of all nodes.

With respect to the topology, there exist two main classes of overlay networks: *structured* and *unstructured* overlay networks. The first class is best represented by Distributed Hash Tables (DHTs), while the second class by gossip-based communication protocols. In the next sections we briefly present their operating principles before delineating the objectives and contributions of this thesis.

1.2 Distributed Hash Tables

In a Distributed Hash Table (DHT), nodes are logically arranged in a large numerical identifier space. The identifier of a node determines its position in the overlay network, whose topology is specific to the given DHT implementation. Among the most known overlay topology examples, there are those where identifiers are logically placed on a circular ring (Chord [15], Pastry [16], Bamboo [17]), on a d-dimensional coordinate space (CAN [18]), or based on XOR metrics (Kademlia [19]).

Each DHT topology defines a *distance metric* to evaluate the distance among two given identifiers. This distance metric comes into play in the *overlay maintenance* protocol run by nodes and in the *lookup* algorithm described next. The overlay maintenance protocol essentially accomplishes two tasks. First, nodes pick as their neighbors the "closest" nodes according to the distance metric. For example, on a ring-based topology, they pick the immediate successor and predecessor nodes on the ring (e.g., [15, 16]). This set of neighbors is usually called *neighbors set* (or *leaf set*), and it ensures connectivity of the overlay and basic routing capabilities, as it will be illustrated next. Second, they pick other "distant" nodes according to a criterion which is specific to the given DHT implementation. For example, the criterion might be choosing nodes with increasingly longer common prefixes with respect to the identifier of the node (e.g., [16]). This second set of neighbors is usually called *routing table*, as it is used to speed up the lookup routing algorithm described next.

A DHT provides a variant of consistent hashing [20] to deterministically map a *key* (i.e., an identifier, for instance obtained by hashing the content of an object) to a node, being the node in charge of the key typically the one whose position immediately follows the key on the identifier space (e.g., [15]). The procedure to lookup for this node is known as key-based routing (KBR) [21] and roughly works accordingly to the following greedy algorithm. Once a node receives a lookup request for a given key k,

two scenarios may arise: if the node has in its neighbors set or routing table a node with closer identifier to k, then it routes the lookup to that node; otherwise, the node concludes it must be the node to be in charge of k and responds to the lookup. DHT implementations (e.g., [15,16]) are designed such that this key-based routing is efficient, typically in the order of $log(N)$ hops, N being the number of nodes in the overlay.

This $lookup(k)$ primitive is the block on top of which the actual Distributed Hash Table API is built: $put(k, v)$ and $get(k)$. The $put(k, v)$ function allows assigning a value v to the key k, which means making the node in charge of k store the value v. Analogously, the $get(k)$ function allows retrieving the value(s) associated to a given key k by previous invocation(s) of the $put()$. To increase resiliency to node arrivals and departures (*churn*), DHTs usually employ a replication strategy (e.g., [22, 23, 24]) to add redundancy to the copies of a given $\{k, v\}$ stored pair.

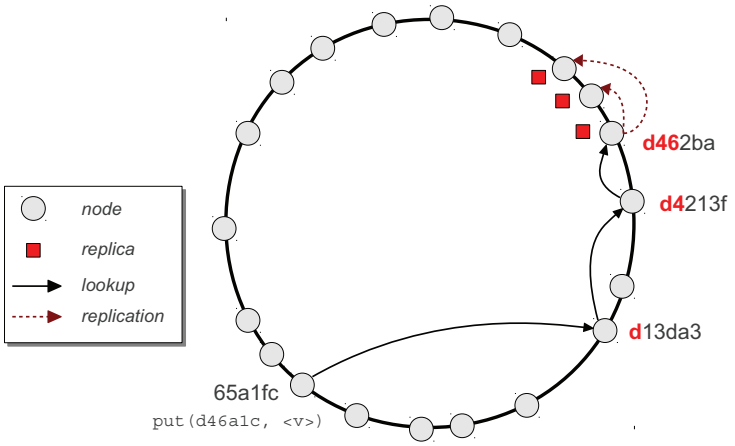


Figure 1.2 – DHT with ring topology: $put()$ with {key, value} replication on the most immediate successors of the node in charge of the key. (*Based on:* [14])

Figure 1.2 illustrates a graphical example of what we explained so far. In the example, the DHT has a ring topology, and the node with id $65a1fc$ issues a $put()$ for the key $d46a1c$. This translates in the request being routed along various hops with increasingly longer common identifier prefix, until the request reaches the node whose identifier is the closest to the given key. This turns out to be node $d462ba$, which also replicates the stored key/value pair on its two most immediate successor nodes.

1.3 Gossip-based protocols

As described previously, Distributed Hash Tables are characterized by a given *structured* topology. This topology remains *static* when there are no node arrivals or departures. In fact, in absence of churn, the overlay network maintenance protocol will eventually make the neighbors relationships converge to the ones which best satisfy the distance metric function and routing table criteria. Once the overlay has converged to this state, nothing changes if no node joins or leaves. The overlay will only *reactively*

adjust its links in case of network dynamics.

Gossip-based overlay networks, instead, reside at the other side of the spectrum. In fact, gossip-based protocols (e.g., [25, 26, 27]) aim at building and maintaining an *unstructured* topology with random graph [28] properties, for that they fairly balance the load among the nodes, and are highly robust to node failures. Furthermore, this random topology is *dynamic*, as nodes *proactively* keep continuously changing neighbors.

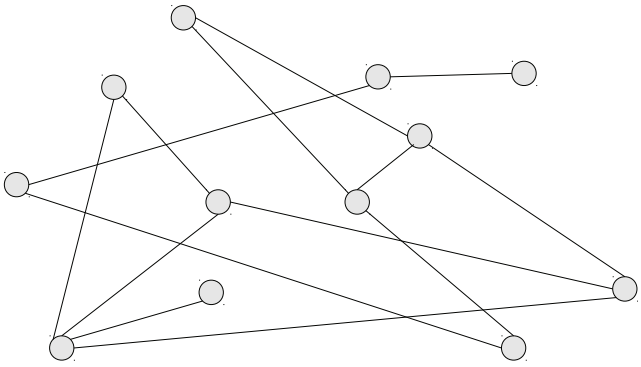


Figure 1.3 – Gossip protocols: random graph topology.

In a gossip-based protocol each node periodically exchanges ("gossips") information with one (or generally, more than one) node of the system, selected from its view. This turns out to ensure the progressive and reliable diffusion of the information to all nodes with high probability (e.g., [29, 30]). Due to this approach, gossip protocols are also referred to as *rumor mongering protocols* [31] (due to the similarities with the spreading of a rumor), or —very commonly— as *epidemic protocols* [29] (due to the similarities with the diffusion of a virus).

The gossip paradigm was first proposed in distributed systems by Demers et al. [29], in the context of the Xerox Clearinghouse service [32], to maintain database replicas consistent by having replicas gossip updates among them. Since this seminal paper, gossip-based protocols have been used in various contexts and for different purposes: data broadcasting [25, 30, 33], failure detection [34], data aggregation (e.g., network size estimation) [35, 36], slicing of the node population based on some node property [37, 38, 39], topology management (i.e., clustering nodes based on certain criteria) [40, 41], and even live video streaming [42, 43], to name a few.

A generic framework to describe gossip protocols has been provided by Kermarrec et al. [44], and its simple and concise pseudo-code[1] is reported in Figure 1.4. Each node runs both a periodic *active* part which selects the target node of the gossip and the data to exchange with it, and a *passive* part which handles the reception of the data and (possibly) which takes care of sending the response to it. The main aspects of a generic gossip protocol are thus: *(i)* peer selection, *(ii)* data exchange, and *(iii)* data processing. Whereas the last two aspects are specific to the application context, the first one, peer selection, turns out to be a common underlying fundamental building block of every

[1]The pseudo-code of the gossiping framework presented in [44], as well as other pseudo-codes presented in the rest of this document, have been adapted to follow similar style conventions.

gossip protocol: the so called *peer sampling service* (PSS) [45].

A gossip protocol in fact periodically picks one (or more than one) *random* node to gossip with by means of a $selectPeer()$ call to the peer sampling service. The possibility to rely on a *uniform random sample* of nodes, returned by the peer sampling service, has been shown to ensure the reliability and scalability properties of several gossip-based protocols (e.g., [30, 36, 38, 46]).

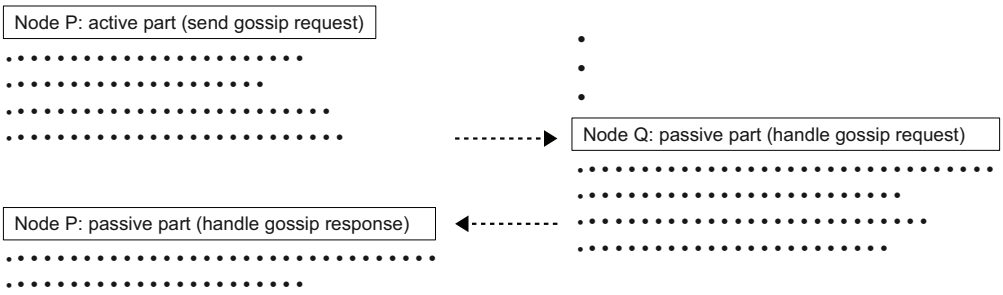


Figure 1.4 – Functioning of a generic gossip protocol.

Such a random peer sampling service is also typically implemented in a decentralized fashion via gossiping, by so called gossip-based peer sampling protocols [26,27,47]. In these protocols each node locally maintains a small view of the system, and periodically exchanges its view with a node selected from the view itself. This approach has been shown experimentally [26, 27] and analytically [48] to result in views which represent a continuously changing random sample of all nodes.

1.4 Objectives and contributions of this thesis

This thesis settles its grounds in the area of gossiping protocols given their appealing characteristics for building large-scale distributed systems: they are scalable, robust, highly resilient to churn, lightweight and simple to implement.

Little attention has been devoted to their behavior in real-world deployment scenarios, where these protocols might face practical problems. Specifically, this thesis addresses two problems. The first problem we address is coping with Network Address Translators (NATs) in gossip-based peer sampling protocols. The second problem we address is limiting the dissemination of "spam" messages in gossip-based dissemination protocols, while accounting for malicious and selfish behaviors of the nodes.

NAT-resilient Gossip Peer Sampling. In gossip peer sampling protocols (e.g., [27]), each node keeps a local set of neighbors (its local view) which it periodically exchanges with another node of the system that is selected from the local view itself. This view is expected to be a sample of the nodes picked uniformly at random among all the nodes of the system. These protocols rely on the implicit assumption that any node is able to communicate with any node contained in its local view. But, it is a well known fact that nowadays a large fraction of nodes in the Internet are behind Network Address Translators (NATs) [49, 50, 51, 52]. NAT devices allow several nodes with a

private IP address to share a single public IP address. NATs implement firewall-like functionalities which drop unsolicited incoming messages. Consequently, the presence of NATs may prevent communication among nodes. Whereas the issue of NATs has been considered in the case of structured P2P overlay networks [49, 53, 54], it has been almost ignored in the area of gossip protocols.

Our first contribution in this area is to show how the mere presence of NATs significantly impacts the properties of gossip-based peer sampling protocols with respect to the connectivity of the overlay network and the randomness of the returned sample. In fact, we show how above a given threshold of nodes sitting behind a NAT, the overlay network gets partitioned, a condition that a peer sampling protocol precisely should avoid. Also, we show how the local views of nodes present many references to nodes which are actually unreachable. This implies that an application using the peer sampling service could effectively use only a subset of the entries in the view, corresponding basically to the nodes which are not behind a NAT.

Our second contribution in this area is the design and evaluation of $\mathcal{N}ylon$ [55], a fully decentralized NAT-resilient gossip-based peer sampling protocol. $\mathcal{N}ylon$ is built on the gossip-based peer sampling framework [27] and it is based on a decentralized hole punching [56, 57] approach to traverse NATs and establish paths of relay nodes in order to initiate the communication towards nodes behind a NAT device. We show how $\mathcal{N}ylon$: *(i)* ensures the properties of the peer sampling service, despite the presence of large fractions of nodes sitting behind NATs, *(ii)* fairly balances the load among nodes (be they behind or not a NAT device), *(iii)* efficiently handles churn.

Spam-resilient Gossiping in the BAR model. Peer-to-peer systems can be used to build decentralized communication and content distribution services as, for example, Internet forums. In such a context, an appealing and effective way to disseminate the content produced by a node to all other participants is by means of gossip-based dissemination protocols [25, 30]. In a gossip-based dissemination protocol, once a node receives a new message, it forwards it to a random subset of its neighbors. The advantages of such a scheme are it simplicity and its proved reliability [30].

Our first contribution in this area is to show how such a gossiping approach turns out to be also an ideal vector for the dissemination of *spam* content in the system. We consider as spam a message whose content is inappropriate for participating users (e.g., false information diffused on the forum). We base our analysis on the notion of *filtering capability* (also called "pollution awareness" in [58]) and we show how a simple strategy consisting in having each node locally filter the messages it detects as spam does not work. In fact, gossip-based dissemination protocols are highly redundant and random: each node receives messages multiple times from different nodes.

Our second contribution in this area is the design and evaluation of $\mathcal{F}ire\mathcal{S}pam$ [59], a gossiping protocol which is able to limit the spam dissemination. In $\mathcal{F}ire\mathcal{S}pam$, nodes are organized in a ladder topology according to their capability to filter spam: nodes having a high (resp. low) filtering capability are located at the top (resp. bottom) of the ladder. Messages are disseminated from the bottom to the top of the ladder, which acts as a progressive spam filter. The rationale behind this topology is that nodes that actively filter spam (i.e., those with a high filtering capability) progressively

climb the ladder and will eventually be less overwhelmed by spam messages. While organizing nodes in a ladder topology can easily be achieved if all nodes in the system behave correctly (e.g., by clustering nodes [40, 41]), it appears challenging to build such a topology in the presence of nodes acting selfishly or maliciously. Nodes with such behaviors are common in P2P systems and they do thus need to be taken into account. We designed $\mathcal{F}ire\mathcal{S}pam$ in the BAR model [60]. This model states that there are three kinds of nodes: *altruistic* nodes that strictly follow the protocol, *Byzantine* nodes that can behave arbitrarily, and *rational* nodes that are willing to deviate from the protocol if there is a gain in doing so. In order to tolerate rational nodes, $\mathcal{F}ire\mathcal{S}pam$ encompasses a set of incentive-compatible mechanisms that make the protocol a strict Nash equilibrium [61]. More precisely, the incentive mechanisms used in $\mathcal{F}ire\mathcal{S}pam$ ensure that it is in a rational node's best interest to always follow the protocol. Moreover, $\mathcal{F}ire\mathcal{S}pam$ encompasses a set of mechanisms guaranteeing that Byzantine nodes are detected and evicted from the system. These mechanisms assume a known upper bound on the number of Byzantine nodes in the system.

1.5 Research methodology

We have conducted an experimental evaluation of the two protocols presented in this thesis: $\mathcal{N}ylon$ and $\mathcal{F}ire\mathcal{S}pam$. For $\mathcal{F}ire\mathcal{S}pam$, we provided also a more formal study and proof of its properties.

The experimental evaluation of the two protocols consisted in studying their behavior through simulations, which allowed to systematically evaluate them using different configuration parameters. For each of the two protocols, a dedicated lightweight round-based P2P simulator inspired by PeerSim [62] has been developed and used to run the experiments. The two P2P simulators share the same architecture and general design principles, they diverge in the implementation language and specific characteristics:

- for $\mathcal{N}ylon$: the simulator has been written in Java, and it has support for simulating the functioning of Network Address Translators. This is used to simulate dropping or receiving messages due to (in)valid NAT filtering rules.
- for $\mathcal{F}ire\mathcal{S}pam$: the simulator has been written in C++ and tuned for performances due to the relatively (with respect to $\mathcal{N}ylon$) higher computational cost of the simulations (periodic creation and dissemination of messages).

1.6 Organization of this document

The rest of this document is organized in the following chapters.

Chapter 2 - Peer Sampling Service and Network Address Translators. This chapter first describes the peer sampling service and illustrates two approaches: random walk and gossip-based. Afterwards, it describes the functioning of Network Address Translators. Focusing on the case of gossip peer sampling [27], the chapter studies how NAT devices, by limiting connectivity among nodes, strongly impact the properties of the gossip peer sampling service.

Chapter 3 - Nylon: NAT-resilient Gossip Peer Sampling. This chapter first describes NAT traversal techniques [56,57] allowing communication towards nodes behind a NAT: *relaying* and *hole punching*. It then presents and evaluates $\mathcal{N}ylon$ [55], a NAT-resilient gossip peer sampling protocol built on the gossip peer sampling framework [27]. $\mathcal{N}ylon$ is based on a decentralized hole punching approach to traverse NATs and establish paths of relay nodes in order to initiate the communication with nodes behind a NAT device.

Chapter 4 - Spam and the presence of Byzantine and rational nodes. This chapter first describes gossip-based dissemination as a lightweight, robust and reliable approach for information dissemination in an overlay network. Focusing on the case of the protocol described in [30], the chapter shows how the randomness and redundancy of gossiping make it an ideal vector for disseminating *spam*. Then, the chapter presents the system model under which we target designing a solution to the spam dissemination. This model is made of Byzantine nodes, which can arbitrarily fail, and rational nodes, which want to maximize their benefit while reducing their contribution. A few practical systems taking into account Byzantine faults are illustrated, as well as a few systems dealing with both rational and Byzantine behaviors.

Chapter 5 - FireSpam: Spam-resilient Gossiping in the BAR Model. This chapter first proposes a *ladder* organization of the nodes of the system based on their spam filtering capability. The ladder can be used to reliably deliver good messages among nodes via gossiping, while limiting the diffusion of spam messages via the progressive filtering exercised by the nodes themselves along the ladder dissemination. Then, the chapter presents and evaluates the $\mathcal{F}ire\mathcal{S}pam$ protocol [59]. The protocol guarantees the ladder properties (correct nodes placement in the ladder, reliable dissemination of messages), while: *(i)* tolerating a (configurable) bounded number of Byzantine nodes, and *(ii)* encouraging all non-Byzantine nodes to follow the protocol by means of a set of incentives.

Chapter 6 - Conclusions. This chapter concludes this document, providing a summary of the results of this thesis, and discussing elements for possible future research.

Part I

NAT-resilient Gossip Peer Sampling

2

Peer Sampling Service and Network Address Translators

Contents

Gossip-based protocols rely on the ability for a node to periodically exchange ("gossip") information with other random nodes. The service providing every node with this random sample of nodes is called *peer sampling service* (PSS) [45].

Nevertheless, an implicit assumption of gossip protocols is that any node is able to communicate, and hence perform the gossip exchange, with any other node in the

system. This is generally not the case on the Internet, where a significant proportion of nodes might be behind Network Address Translators (NATs) [49, 50, 51, 52]. NAT devices in fact, by employing firewall-like mechanisms, limit the connectivity of nodes and as such they can harm the properties of a random peer sampling service, and consequently of the protocols relying on it for their functioning.

This chapter is organized as follows. Section 2.1 presents two approaches to random peer sampling: one based on random walk methods and one based on gossiping. Section 2.2 illustrates the basic functioning of Network Address Translators and the limitations they impose on node to node communication. Section 2.3 studies the impact of NATs on gossip-based peer sampling protocols. Section 2.4 discusses two existing solutions for gossiping in presence of NATs. Section 2.5 concludes this chapter.

2.1 Peer sampling service

In gossip-based protocols, each node periodically exchanges information with one (or generally, more than one) node of the system. The peer sampling service [45] is the distributed abstraction which provides every node with this sample of nodes to gossip with. As depicted in Figure 2.1, the same peer sampling service can generally be used by more than one gossip protocol at once, thus constituting the common underlying building block of a gossip-based network application.

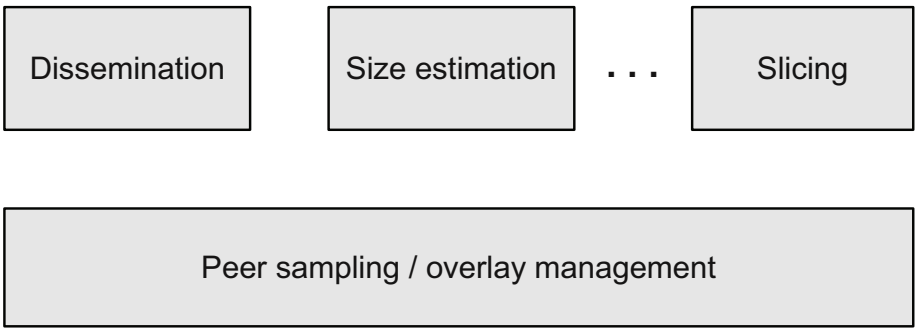


Figure 2.1 – Gossip protocols: importance of the peer sampling service as underlying building block.

A key aspect in gossiping is the uniform random sample of nodes to gossip with. Several gossip protocols (e.g., [30, 36, 38, 46]) in fact owe their reliability and scalability properties to the randomness of such a sample. In [46], Pittel proves that if each node at periodic rounds picks a random node to gossip information with, then with high probability the number of gossip rounds to have the information spread among the *whole* set of nodes of the system is in the order of $log_2(N) + ln(N) + O(1)$, N being the total number of nodes in the system.

The first gossip protocols (e.g., [29, 33]) relied on the assumption that each node could sample nodes uniformly at random by keeping a complete knowledge of all the other nodes of the system, that is, a *global view*. In a P2P system, which can be composed of a huge number of nodes and be highly dynamic, keeping a global view up-to-date can impose important communication and storage costs on nodes. Scalable

peer sampling services have then be proposed afterwards, relying on nodes only having a small, *partial view* of the system.

In the following, we describe then two existing approaches for implementing a random peer sampling where nodes only have a partial view of the system. First, we describe a peer sampling service based on *random walk* methods. Then, we describe instead a generic gossip-based peer sampling framework.

2.1.1 Random walks based peer sampling

The random peer sampling algorithms presented in [63, 64, 65] rely on random walk methods [66]. *RaWMS* [63] targets wireless ad-hoc networks, while instead the algorithm presented in [64] allows to sample a node at random in a Distributed Hash Table. We focus here on the *Sample&Collide* algorithm [65], for it allows performing random walks to return a random sample of the nodes using a method which is not tied to a specific networking context or a specific overlay network structure.

In the following paragraphs, we first illustrate the operating principles of random walks on a graph. Then, we describe the *Sample&Collide* peer sampling protocol, which is based on random walk methods.

2.1.1.1 Random walks on a graph

A P2P overlay network can be seen as a graph G=(V,E), where V is the set of nodes in the system and E are the edges among these nodes. In particular, an edge from a node p to a node q exists if q is present in p's local view. Roughly speaking then, a random walk is a sequence of hops from one node (the "initiator node") initiating the random walk, to a given "end node" (which can be the initiator node itself), and where the next hop along the path is randomly chosen from the neighbors in the local view of the current node.

Figure 2.2 illustrates a possible random walk on a graph, starting from some initiator node and, after three steps, ending at a different node of the graph.

2.1.1.2 The *Sample&Collide* algorithm for nodes counting and sampling

The *Sample&Collide* (in the rest, *S&C*) [65] is a protocol to measure system size, that is, the number of nodes in the system. The *S&C* protocol, in order to measure the system size, relies on a sub-routine which is actually a random peer sampling based on a random walk, and that is hence our focus here. The *S&C* protocol in fact provides a system size estimation which is based on the number of uniform random samples that a node picks before reaching a given number of redundant samples. We illustrate the functioning of the *S&C* peer sampling, and discuss its properties, based on the study conducted by Le Merrer et al. in [67].

The *S&C* approach is based on the *inverted birthday paradox* described in [68]. The original *birthday paradox* [69] problem proves what is the probability $P(N, K)$ of having, within a group of K people, at least two of them born on the same day of the year, assuming the probability of a person to be born on a given day is uniform across

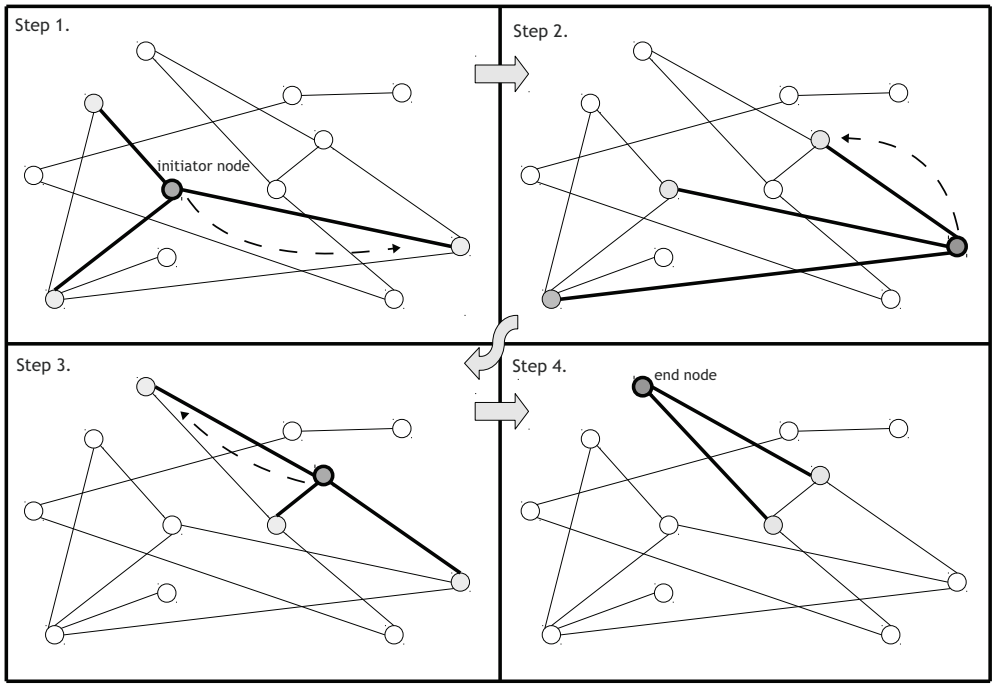


Figure 2.2 – A possible random walk on a graph.

all the days of the year. The *inverted birthday paradox* method [68] takes the inverted approach to the problem. That is, it seeks determining the probability distribution of the number $X(N)$ of people which have to be consecutively selected at random, before finding two people who are born on the same day of year. The value of $X(N)$ tends to be around $\sqrt{2N}$ when N is large. The *S&C* then leverages this result in the following way: the nodes of the system correspond to the dates of the stated problem, and the algorithm continues sampling a random node until it finds a duplicate sample. If there have been X samples before seeing a duplicate sample, then the estimation $\hat{N}$ of the system size is $\hat{N} = X^2/2$.

The *S&C* algorithm improves the above inverted birthday paradox method in providing a uniform sample of nodes even if nodes have different *in-degrees*, the in-degree of a node being the number of nodes having it as neighbor in their local views. The algorithm works as follows. An initiator node i starts a timer with some predefined value $T > 0$, and the timer is enclosed in a sampling request message which is meant to be disseminated in a random walk fashion. In fact any node j, either because it is the initiator node itself or because it has received the sampling message, executes the following tasks. First, it decrements the timer by $log(1/R)/d_j$, where R is a randomly thrown value in $[0, 1]$ and d_j is the *out-degree* (i.e., the number of neighbors) of node j. Afterwards, if at this point $T > 0$, then node j continues the random walk by forwarding the sampling request message to a random node chosen among its neighbors. Otherwise, the random walk ends at node j: it either sends to the initiator node i a sampling response message containing its identifier as sampling result, or handles locally this value if node j coincides with node i.

```
1  def selectPeer()
2    timer ← init_timer()
3    sampling_request ← ⟨REQUEST, self, timer⟩
4    process_sampling_request(sampling_request)

5  def process_sampling_request⟨REQUEST, initiator, timer⟩
6    rand_val ← rand()  // returns a random value in [0, 1]
7    timer ← timer − log(1/rand_val) / degree(self)
8    if timer ≤ 0 then
9      if initiator = self then
10       handle_returned_sample(self.id)
11     else
12       send ⟨RESPONSE, self.id⟩ to initiator
13   else  // else forward to a random neighbor
14     neighbor ← select_random(view)
15     send ⟨RESPONSE, initiator⟩ to neighbor

16 def handle_returned_sample(id)
17   // do something with the returned random sampled node id

18 on receive ⟨REQUEST, initiator, timer⟩
19   process_sampling_request⟨ REQUEST, initiator, timer⟩

20 on receive ⟨RESPONSE, id, timer⟩
21   handle_returned_sample(id)
```

Figure 2.3 – *Sample&Collide*: pseudo-code.

The pseudo-code in Figure 2.3 summarizes the functioning of the algorithm. The method `selectPeer()` initializes the timer and encloses it in a sampling request message. Any node, being it the initiator node or a node along the random walk which has received the sampling request message, processes this message by means of the call to the method `process_sampling_request()`. Eventually, the initiator node will receive the id of the node at which the random walk ended, and it will handle the result via the call to `handle_returned_sample()`.

The *Sample&Collide* peer sampling sub-routine provides random samples of nodes in both dynamic and static settings (i.e., with or without node arrivals and departures). The distribution of the returned sample tends towards the desired uniform distribution for increasing values of the system parameter T. The value of T to be chosen depends on the characteristics of the overlay network graph. Roughly speaking, an underlying overlay network with strong connectivity and with random graph properties is a favorable graph topology for the *Sample&Collide* peer sampling service quality.

In the next section, we illustrate a gossip-based peer sampling framework, which besides providing a uniform sample of the nodes of the system, is also able to build and maintain a dynamic overlay network with random graph properties, thus making it an appealing standalone solution.

2.1.2 Gossip based peer sampling

A generic gossip-based peer sampling framework [27] has been proposed, which can be used to implement existing (e.g., Cyclon [26], Newscast [47]) or novel gossip-based peer sampling protocols.

The approach of the framework is based on having each node maintain a small limited knowledge of the nodes of the system, that is, a *local view*. The framework then implements a random peer sampling by building and maintaining an overlay network with random graph properties, by means of continuously exchanging among nodes (via gossiping) their *local view entries*. The local views of nodes are in fact expected to: *(i)* store a continuously changing random sample of the nodes of the system, and *(ii)* quickly adapt in case of node arrivals and departures (churn).

A local view entry contains a reference to a neighbor node. This reference contains the neighbor node's identifier and its IP address and port number. Additionally, a local view entry contains an `age` field indicating the time spent in the local view. This information plays a role in the possible instantiation variants of the framework. Furthermore, the local view size at each node has a fixed maximum size chosen in the order of $ln(N) + c$ neighbors, where N is the number of nodes in the system and c is a constant parameter. In fact, if a node has $ln(N) + c$ (random) neighbors, then the probability of the so formed (random) overlay graph to be connected evolves as $exp(exp(-c))$ [70].

The functioning of the framework is summarized by the pseudo-code[1] of Figure 2.4. Periodically, each node (called the "source node") selects a node in its local view (called the "target node") to gossip with. This *peer selection* actually represents the $selectPeer()$ primitive of the peer sampling service. Depending on the *view propagation* strategy, the source and/or target nodes send (a subset of) their local view entries to the other node. A node receiving a non empty set of entries, merges them in its local view and truncates the result to the maximum view size. This whole gossip operation is typically called *view shuffling* (or *view exchange*), and the period at which the operation is performed is the *shuffling period*[2].

The framework captures then different variants with respect to:

- **Peer selection**. It specifies how the node to gossip with is chosen, via the method `selectPeer()`, among the ones currently present in a node's local view. Possible implementation strategies are selecting a random node (*rand* selection), and the one selecting the oldest node, i.e. the node with the highest age (*tail* selection).

  The authors point out that selecting the node with the lowest age is not considered, as it would make the two gossiping nodes keep a rather correlated local view among them, resulting ultimately in a fairly static overlay.

[1] The pseudo-code portrays a simplified description of the framework, the reader can refer to the original paper [27] for full details.

[2] Nodes are not required to synchronize their periodic gossip executions, but only to use the same shuffling period, which is typically set in the order of a few seconds, e.g., 5s.

```
1  every shuffling_period units do
2    // Increase by one the age of each reference
3    increase_view_age(view)
4    target ← selectPeer(view)
5    if push_pull or push then
6      // Insert self reference with fresh initial age
7      view_s ← select_neighbors(view) ∪ (self,0)
8    else // pull mode
9      view_s ← {}
10   send ⟨REQUEST, view_s⟩ to target

11 on receive ⟨REQUEST, view_s⟩ from source do
12   if push_pull or pull then
13     // Insert self reference with fresh initial age
14     view_t ← select_neighbors(view) ∪ (self,0)
15     send ⟨RESPONSE, view_t⟩ to source
16   if push_pull or push then
17     view ← merge_and_truncate(view, view_s)

18 on receive ⟨RESPONSE, view_t⟩ from target do
19   view ← merge_and_truncate(view, view_t)
```

Figure 2.4 – Gossip-based peer sampling framework: pseudo-code. The methods `selectPeer()`, `select_neighbors()` and `merge_and_truncate()` allow the different combination of implementation strategies of the framework.

- **View propagation**. It states which nodes participating in a gossip exchange actually send a subset of their local view entries to the other gossip partner. In the *push/pull* strategy (see Figure 2.5), both the source and the target node send entries to the other one. In the *pull*-only strategy (see Figure 2.6), the source node sends an empty set, while the target node replies with a subset of its local view entries. Conversely, in the *push*-only strategy (see Figure 2.7), only the source node sends its local view entries, and the target node is not even required to send back a message.

  The authors point out that neither the pull-only nor the push-only strategies are satisfactory. The pull-only strategy is not a suitable solution as it would prevent a node which has an empty local view to refill it. For the push-only strategy, they have experimentally shown it is a non adequate solution, as it tends to partition the overlay graph by forming clusters of nodes, even when there is no churn.

- **View selection**. It determines how to select the subset of local view entries to send to the gossiping partner (the method `select_neighbors()`), and how to merge and truncate the current local view using the entries received from the other node (the method `merge_and_truncate()`) in order to create the new local view for the node. In case of duplicates when merging the received entries with the local ones, the entry with the lowest age is kept. The subset size of local entries to be sent is usually called the *shuffle length* and it is typically chosen to be half of the maximum size of the local view.

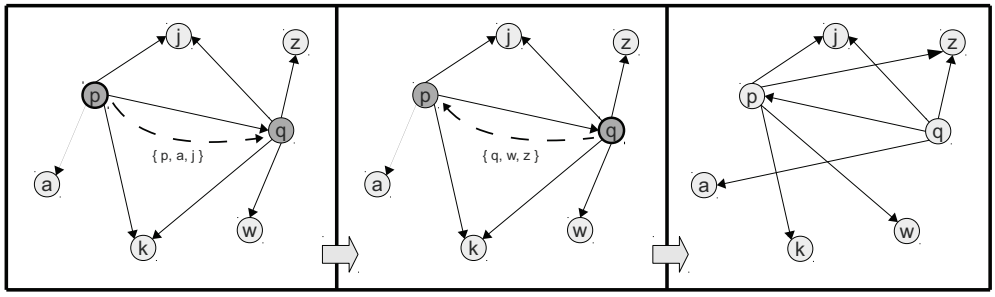


Figure 2.5 – View propagation: PUSH/PULL.

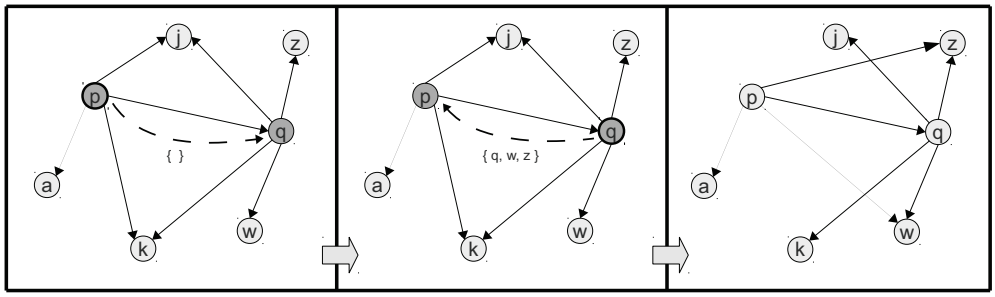


Figure 2.6 – View propagation: PULL-only.

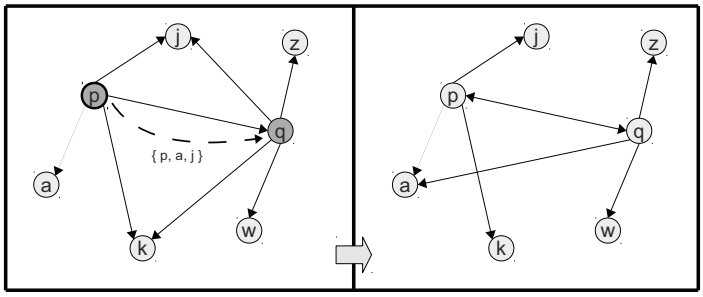


Figure 2.7 – View propagation: PUSH-only.

Three general view selection strategies are considered: *(i)* random entries are selected upon sending and kept upon merging and truncating (*blind* strategy), or *(ii)* the youngest entries are selected and kept (*healer* strategy), or *(iii)* random entries are selected and the ones received from the other node are kept when truncating the local view to its maximum view size (*swapper* strategy).

View selection	`select_neighbors()`	`merge_and_truncate()`
blind	select random entries	keep random entries
healer	select freshest entries	keep freshest entries
swapper	select random entries	swap the sent entries with the received ones

Table 2.1 – Gossip-based peer sampling: view selection strategies.

The framework variants have been evaluated experimentally along different dimensions: randomness of the returned sample of nodes (considering the sequence of entries obtained from the `selectPeer()` call), load balancing (studying the distribution of the number of views in which a node is present), robustness in case of churn (studying the connectivity and size of the biggest connected graph in case of node departures). It turns out that no single variant is the best in all dimensions. In fact, whereas all variants provide a good uniform random sample of nodes returned at each node ("local randomness"), the *swapper* variant tends more closely to a random graph (having load more fairly balanced among the nodes), while instead the *healer* variant is more robust to churn. The trade-off choice depends hence on the application. Nevertheless, and very importantly, all variants manifest a self-organizing property: they convergence to their final topology regardless of the initial configuration of the overlay network.

2.1.3 Summary

We have presented the peer sampling service API and two existing implementations. In the sequel, we focus on the gossip-based peer sampling framework, as it is an appealing and standalone solution for implementing a random peer sampling service by building and maintaining a random overlay network graph. The gossip peer sampling assumes each node can communicate with any other node at any given time. Yet, Network Address Translators, employing firewall-like mechanisms, determine which (incoming) messages are accepted.

In the following sections, we first illustrate the behavior of NATs. Then, we study how NATs impact the properties of gossip peer sampling.

2.2 Network Address Translators

Network Address Translators (NATs) [71] were introduced as a (supposed) temporary solution for the shortage of IPv4 addresses, to allow several nodes with a private IP address to share a single public IP address (that of the NAT). NATs have become

nowadays commonplace, for instance used as the way to access the Internet at individual customers' homes.

But, a NAT also orchestrates the communication between nodes sitting behind it (in the following, "natted nodes") and nodes in the rest of the network (in the following, "external nodes"). To do so, it implements firewall-like functionalities, dropping unsolicited incoming messages. In fact, when a natted node opens an outgoing TCP or UDP session through a NAT, the NAT assigns the session a public IP address and port number to allow subsequent messages from an external node to be received. In addition, the NAT assigns the session a *filtering rule*, which specifies whether messages received from external nodes on the assigned public IP address and port should be forwarded or not to the natted node's private IP address and port. The public IP address and port mapping, as well as the filtering rule, only remain valid a limited time after the last message was sent (or received) in a session.

Furthermore, existing NATs differ in the way they assign public IP addresses and ports, as well as in the filtering rules they implement. In the rest, we assume the communication among nodes is based on UDP. In fact, in the context of a random peer sampling service, nodes only need to communicate once and transmit small data. This gossip exchange does not dictate a reliable communication channel. Also, a node does not need to keep persistent connections with the neighbors in its local view, which are in fact continuously changing after each gossip exchange. From the above arguments, in the following we concentrate on the classification and functioning of NATs as presented in STUN [72], which focuses on communication based on UDP. Nevertheless, similar reasoning can be applied to the case of TCP [56].

In the rest of this section, to help understanding the *four* NAT types described in [72], we illustrate their functioning through graphical examples. In each of them, a natted node has internal (private) IP address $10.0.0.1$ and initiates communication (towards one or more external nodes) from its source port 6789. The natted node is behind a NAT device which has public IP address $1.1.1.1$. Furthermore, all the external nodes in the examples are assumed to be not sitting behind a NAT device and accepting the incoming messages addressed to them.

2.2.1 Full Cone (FC) NAT

This is the most permissive type of NAT. The NAT assigns the same public IP address and port to all sessions started from a given natted node's private IP address and source port. These sessions all share the same filtering rule, which states that the NAT must forward all incoming messages directed to the (mapped) external IP address and port to the internal (private) IP address and source port.

Figure 2.8 illustrates the behavior of FC NAT. A natted node $10.0.0.1$ starts a communication session from its source port 6789, towards the external node $2.2.2.2$ on port 6789. The NAT assigns to this session the public address $1.1.1.1$ and port 7777. The filtering rule which is created is such that any incoming message to $1.1.1.1$:7777, regardless of its source IP and port, will be forwarded by the NAT to $10.0.0.1$:6789.

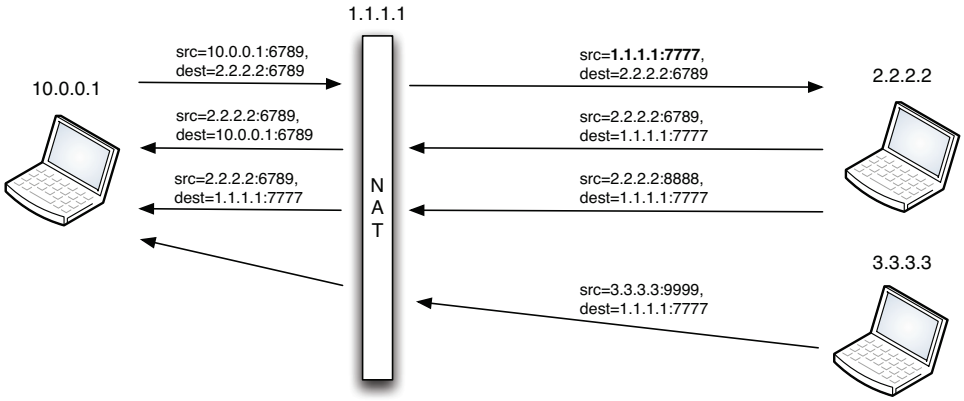


Figure 2.8 – Full Cone NAT. Node 10.0.0.1 is behind the NAT 1.1.1.1.

2.2.2 Restricted Cone (RC) NAT

This type of NAT imposes restrictions on the IP addresses of external nodes that can send messages to natted nodes. As for FC NATs, the RC NAT assigns the same public IP address and port to all sessions started from a given natted node's IP address and port. All the sessions started from a given natted node's IP address and port, and involving the same target IP address, share the same filtering rule: the NAT only forwards messages coming from this IP address.

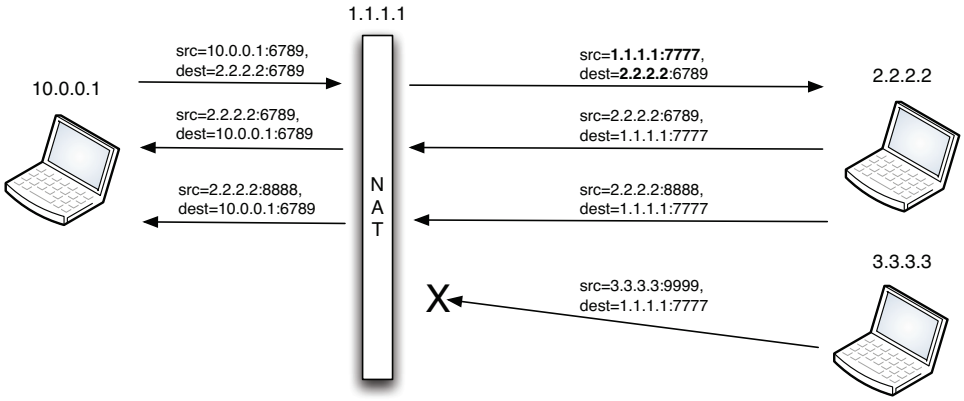


Figure 2.9 – Restricted Cone NAT. Node 10.0.0.1 is behind the NAT 1.1.1.1.

Figure 2.9 illustrates the behavior of RC NAT. A natted node 10.0.0.1 starts a communication session from its source port 6789, towards the external node 2.2.2.2 on port 6789. The NAT assigns to this session the public address 1.1.1.1 and port 7777. The filtering rule which is created is such that any incoming message to 1.1.1.1:7777 coming from the IP address 2.2.2.2, regardless of its source port, will be forwarded by the NAT to 10.0.0.1:6789. Incoming messages from any other IP address will be dropped by the NAT.

2.2.3 Port Restricted Cone (PRC) NAT

This type of NAT imposes restrictions on the IP addresses and ports of external nodes that can send messages to natted nodes. As for the previous NAT types, the NAT assigns the same public IP address and port to all sessions started from a given natted node's IP address and port. Nevertheless, each session started from a given natted node's IP address and port towards a target IP address and port, has its own filtering rule. This rule states that the NAT only forwards messages coming from the target IP address and port to which the session has been opened.

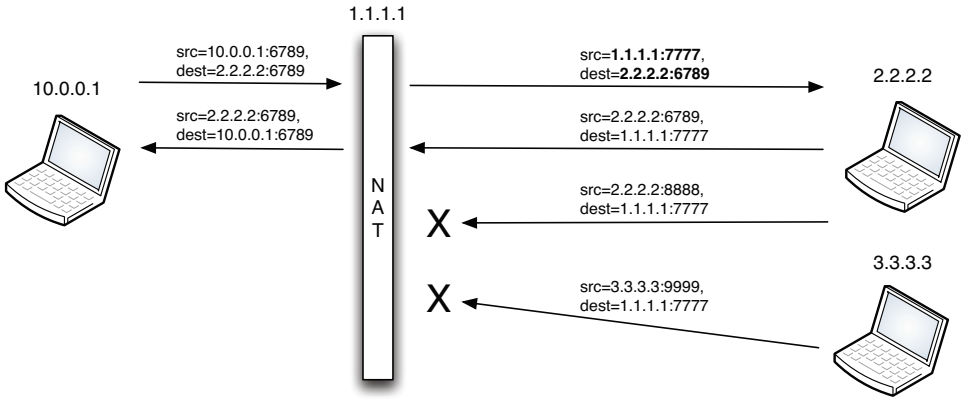


Figure 2.10 – Port Restricted Cone NAT. Node 10.0.0.1 is behind the NAT 1.1.1.1.

Figure 2.10 illustrates the behavior of PRC NAT. A natted node 10.0.0.1 starts a communication session from its source port 6789, towards the external node 2.2.2.2 on port 6789. The NAT assigns to this session the public address 1.1.1.1 and port 7777. The filtering rule which is created is such that any incoming message to 1.1.1.1:7777 coming from the IP address 2.2.2.2 and source port 6789 will be forwarded by the NAT to 10.0.0.1:6789. Incoming messages from any other port of the node 2.2.2.2, or from any other IP address and port, will be dropped by the NAT.

2.2.4 Symmetric (SYM) NAT

This is the most restrictive type of NAT. For every session started from a given natted node's IP address and port, the NAT always assigns the same public IP address but a different port. Note that contrarily to other NAT types, the mapping is destination (IP and port) dependent. The filtering rule is similar to the one used in PRC NATs: the NAT only forwards messages coming from the target IP address and port to which the session has been opened.

Figure 2.11 illustrates the behavior of SYM NAT. A natted node 10.0.0.1 starts three communication sessions, all of them from its source port 6789:

- one session towards the external node 2.2.2.2 on its port 6789. The NAT assigns to this session the public address 1.1.1.1 and port 7777.

- another session towards the external node 2.2.2.2 but this time to the port 9876. The NAT assigns to this session the public address 1.1.1.1 but this time on port 8888.

- one session towards the external node 3.3.3.3 on its port 6789. The NAT assigns to this session the public address 1.1.1.1 and this time again on a different port with respect to the other sessions, that is port 9999.

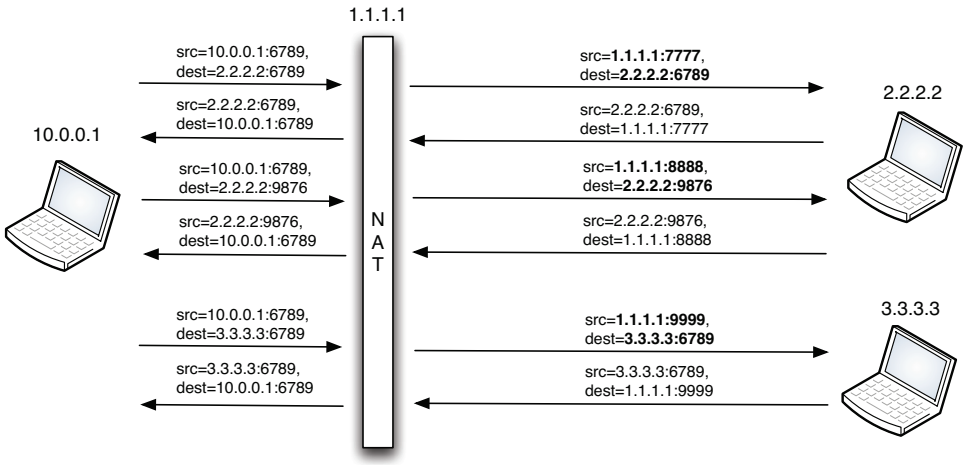


Figure 2.11 – Symmetric NAT. Node 10.0.0.1 is behind the NAT 1.1.1.1.

Note that, despite the fact that the natted node initiates a session always from its port 6789, the NAT device maps this port to a different public port for each different destination IP address and port pair.

Furthermore, as in the case of the Port Restricted Cone NAT, for each of the three started sessions, a different filtering rule is created. Each rule is such that an incoming message to a mapped public IP and port pair is forwarded to the private IP and port composing the mapping only if the message comes exactly from the IP address and port towards which the session originally initiated.

2.3 Impact of NATs on gossip peer sampling

As described in the previous sections, the gossip-based peer sampling framework is an appealing and standalone solution for implementing a random peer sampling service by building and maintaining a random overlay network graph. We focus on such a framework in the sequel, and study the impact of NATs on its properties. In fact, the gossip peer sampling assumes each node can communicate with any other node at any given time. Yet, Network Address Translators, employing firewall-like mechanisms, determine which (incoming) messages are accepted.

To study the impact of NATs, we evaluated six different configurations of the generic gossip-based peer sampling framework described in Section 2.1.2. The six configurations result from the combination of the possible variants of the peer selection

(*rand* or *tail*) and view selection (*blind*, *swapper*, *healer*) strategies. In fact, as explained previously, the only suitable view propagation strategy is the $push/pull$ strategy, as it has been shown to consistently exhibit better performances than the push-only or pull-only gossip exchange modes [27]. This is then the one used in the all six configurations.

The experiments have been obtained through extensive simulations. The network size is set to 10,000 nodes, and the bootstrapping procedure is such that at the beginning of the simulation all nodes' local views are filled with randomly chosen public nodes. The initial graph is thus always connected. No churn was considered. Moreover, for the sake of simplicity, only PRC NATs are considered in the experiments presented in this section. We evaluated the protocols along the following metrics: *(i)* the resilience of the protocol with respect to network partitioning; *(ii)* the ratio of stale references in the views of nodes and; *(iii)* the randomness of the resulting views.

2.3.1 Overlay network partitions

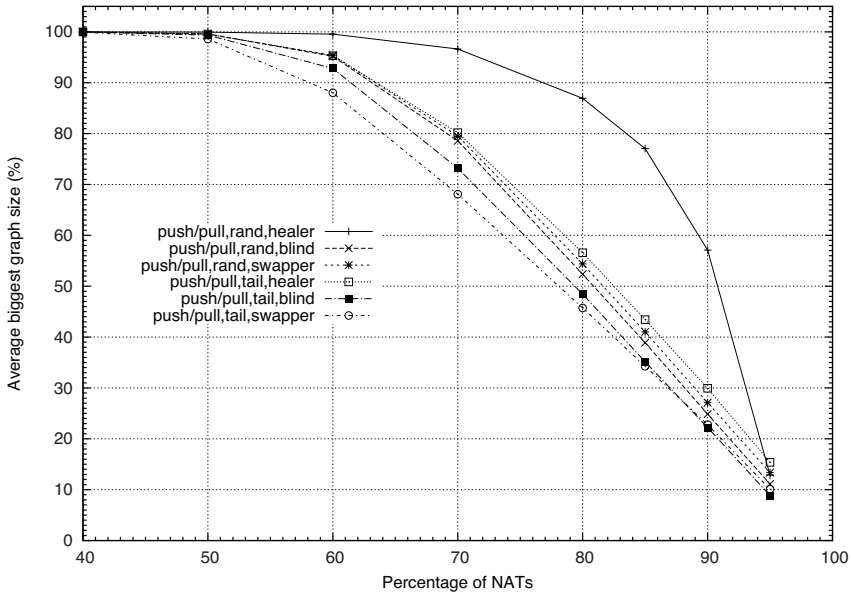
Figure 2.12 shows the size of the biggest connected graph as a function of the percentage of natted nodes for two chosen local view sizes (15 and 27). The biggest graph size is expressed as the percentage of nodes of the system belonging to it. As explained in the previous section, a gossip peer sampling protocol should ensure that the graph never partitions in the absence of churn. This means that the biggest connected graph should always contain 100% of the nodes. We clearly see that this property is not ensured when the percentage of natted nodes reaches a certain threshold (50% and 70% for the considered view sizes). We observe that, as expected, increasing the view size has a positive impact on the biggest connected graph size for all the six protocol configurations. One can legitimately consider that increasing the view sizes is enough to prevent partitions in the presence of NATs. This is actually what is proposed in the cache-based solution described in ARRG [73]. We show in the reminder of this section that increasing the view size is not a satisfactory solution with respect to the two other metrics: the randomness and ratio of stale references.

2.3.2 Stale references

Figure 2.13 shows the average percentage of stale references in the local views of nodes for two different view sizes (15 and 27). A reference to a neighbor is said to be *stale* when it is not possible to communicate with this node (due to the presence of NATs). We observe that a small percentage of natted nodes suffices to cause nodes to have stale references in their view, and that the percentage of stale references almost linearly grows with the percentage of natted nodes. This is not acceptable for applications relying on a peer sampling protocol. Moreover, we observe that: *(i)* the percentage of stale references increases when the view size increases, and *(ii)* the percentage of stale references decreases for view size 15 when the percentage of NATs reaches a certain threshold (85%).

These two observations can be easily explained by two facts. First, increasing the view size decreases the probability that two nodes shuffle with each other twice during the lifetime of a NAT filtering rule. Second, with a large percentage of NATs and view size 15, the network starts to significantly partition in many small clusters.

(a) Local view size: 15



(b) Local view size: 27

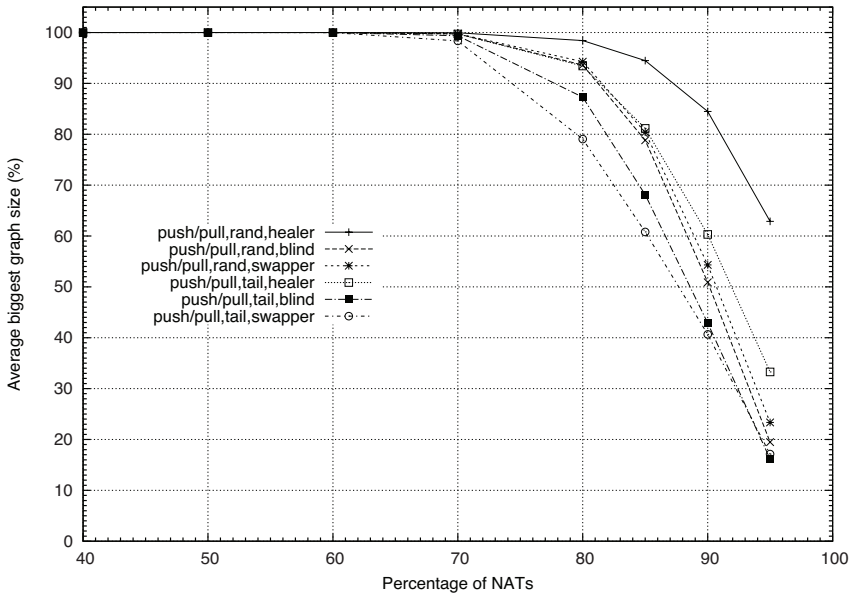


Figure 2.12 – Size of the biggest connected graph using two different local view sizes: (a) 15, (b) 27.

Consequently, two nodes within a cluster have a very high probability to shuffle with each other twice during the lifetime of a NAT filtering rule. They will thus have much fewer references than nodes in the biggest cluster, which in turn reduces the average number of stale references over all nodes. To illustrate this, consider the extreme case of a small cluster of two nodes: they will eventually only have one reference in their view (i.e., a reference towards the other node in the cluster). This reference will always be valid, as the two nodes continuously shuffle together, thus refreshing the validity of their NAT filtering rule.

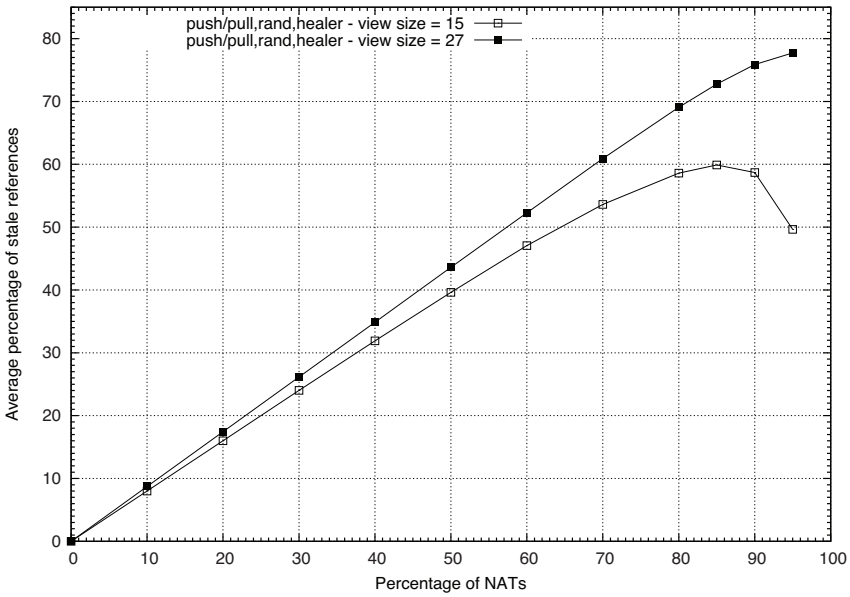


Figure 2.13 – Percentage of stale references.

2.3.3 Randomness

Figure 2.14 shows the average percentage of non-stale references that correspond to natted nodes. Again, we consider two different view sizes (15 and 27). The evaluations show that with 40% of natted nodes and a view of size 15, nodes have on average only 10% of their non-stale references that correspond to natted nodes. This typically means that 40% of the nodes are sampled only 10% of the time, which is obviously a non uniform random sampling. As in Figure 2.13, we observe that increasing the view size negatively impacts the protocol. We also observe that when the percentage of NATs reaches a certain threshold (70%), the average percentage of non-stale references increases. The explanation is similar to the one given for Figure 2.13. In fact, the percentage of non-stale references towards natted nodes increases when nodes selecting natted nodes as shuffling targets have higher probability to be able to communicate. This happens when the view size decreases and when the network partitions.

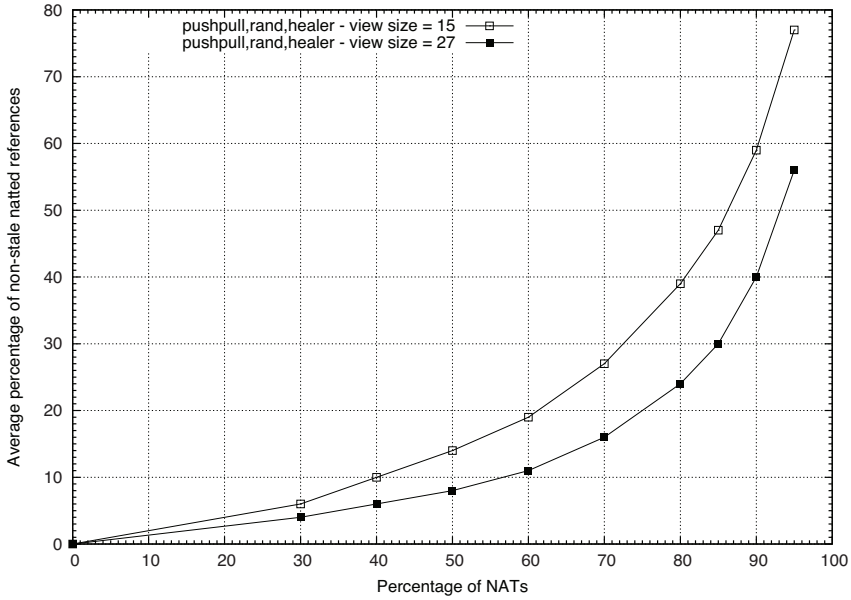


Figure 2.14 – Percentage of non-stale references towards natted nodes.

2.3.4 Summary

We have shown how the presence of natted nodes significantly impacts the properties of the peer sampling protocol with respect to both the randomness of the provided samples and the connectivity. Effectively, above a given threshold of natted nodes, potentially unreachable without specific actions, the network gets partitioned. Note that this is precisely what gossip protocols are expected to avoid. In addition, the mere presence of nodes behind NATs alters the properties of the protocol. Typically, many entries in the views of nodes are unreachable (stale references), and therefore the number of entries that an application running on top of the protocol can effectively use are mostly those which correspond to nodes which do not sit behind a NAT.

We describe in the following section two existing solutions aimed at accounting for NATs in gossiping protocols, and discuss their drawbacks.

2.4 Gossiping in presence of NATs: existing solutions

To the best of our knowledge, the only works dealing with NATs in gossip protocol are [73, 74]. We first describe the principles behind each of them. Then, we discuss why they do not represent a satisfactory solution for realizing a random peer sampling service.

2.4.1 ARRG: Actualized Robust Random Gossiping

In [73], a gossip protocol called ARRG is proposed. Together with the protocol, a *fallback cache* mechanism is proposed. This fallback cache is a mechanism which can be added to existing gossip-based protocols, without changing their basic behavior. The

goal of the cache is to make gossiping protocols more resilient to connectivity problems, i.e. firewalls or Network Address Translators.

We describe the usage of this fallback cache by using the gossip-based peer sampling framework described previously in this chapter. To do so, we adapt a push/pull instance of the peer sampling framework to make use of the fallback cache. Figure 2.15 illustrates this variant of the gossip peer sampling framework using the cache.

```
1  every shuffling_period units do
2    target ← selectPeer(view)
3    view_s ← select_neighbors(view) ∪ (self,0)
4    send ⟨REQUEST, view_s⟩ to target
5    start receive_timer for target

6  on trigger receive_timer for target do
7    target_from_cache ← selectRand(fallback_cache)
8    view_s ← select_neighbors(view) ∪ (self,0)
9    send ⟨REQUEST, view_s⟩ to target_from_cache

10 on receive ⟨REQUEST, view_s⟩ from source do
11   fallback_cache.update_with(source)
12   view_t ← select_neighbors(view) ∪ (self,0)
13   send ⟨RESPONSE, view_t⟩ to source
14   view ← merge_and_truncate(view, view_s)

15 on receive ⟨RESPONSE, view_t⟩ from target do
16   stop receive_timer
17   fallback_cache.update_with(source)
18   view_t ← select_neighbors(view) ∪ (self,0)
19   send ⟨RESPONSE, view_t⟩ to source
20   view ← merge_and_truncate(view, view_t)
```

Figure 2.15 – Push/pull instance of the gossip-based peer sampling framework using ARRG fallback cache technique: pseudo-code.

The functioning of this modified instance of the gossip-based peer sampling framework protocol becomes thus the following. As in the original version, periodically each node selects from its local view a target node to gossip with. But, differently from the peer sampling framework, upon sending a request, the node also registers a timer (line 5) in order to detect a possible connectivity problem towards the target node. If no response from the target node arrives before the timer elapses, the source node retries (only once) by picking a random entry from its fallback cache (line 7). This cache is composed of a (fixed size) set of nodes with which the sender node has successfully communicated in the past. The presence of this cache is supposed to ensure that at any time a node has a high probability to know a node it can communicate with.

2.4.2 Balancing gossip exchanges in networks with firewalls

In [74], the authors first discuss how in a topology with connectivity limitations (due to firewalls or NATs), nodes do not typically participate in a fairly balanced number of gossip exchanges. In particular, public nodes —by being reachable by all nodes—

end up participating in more gossip exchanges, and thus consuming more resources, than natted nodes.

The authors propose then a gossip protocol which balances the number of gossip exchanges the nodes perform. Figure 2.16 provides a simplified pseudo-code description of the protocol. The protocol relies on two main components. First, the protocol assumes that each node relies on an external peer sampling service (line 2) which only returns public nodes or nodes sitting behind the same NAT device. This ensures that the target node can accept a message from the sender node. Second, each node keeps track of the number of gossip exchanges it actually initiated or answered to. Then, upon receiving a request, a node will handle it and send a response (line 9-12) if the number of initiated gossips exceeds the number of gossip requests it has responded to. Otherwise, the node does not respond to the gossip request, but instead forwards it to the last node it had previously communicated with in the past (line 15). The forwarding of the gossip request continues until a node with a balanced quota to accept the request is found, or when the maximum number of allowed hops is reached. The gossip response is then sent back to the node which sent the request by traversing backward the same path of nodes which carried the request.

```
1  every gossip_exchange_period units do
2    target ← peer_sampling.selectPeer()
3    path ← [] ∪ self
4    num_hops ← 1
5    send ⟨REQUEST, path, 1⟩ to target
6    quota ← quota+1

7  on receive ⟨REQUEST, path, num_hops⟩ from p do
8    if quota>0 or num_hops = MAX_HOPS or last_contacted = ⊥
9      quota ← quota−1
10     // handle the gossip request
11     backward_path ← path \ p
12     send ⟨RESPONSE, backward_path⟩ to p
13   else
14     // forward to last node it communicated with
15     send ⟨REQUEST, path ∪ self, num_hops+1⟩ to last_contacted
16   if hop = 1 then
17     last_contacted ← p

18 on receive ⟨RESPONSE, backward_path⟩ from q do
19   if backward_path = [] then
20     // handle the gossip response
21   else
22     next_hop ← last_of(backward_path)
23     send ⟨RESPONSE, backward_path \ next_hop⟩ to next_hop
```

Figure 2.16 – Pseudo-code for "Balancing gossip exchanges in networks with firewall".

.

2.4.3 Summary

We have illustrated two existing solutions to account for NATs in gossiping protocols. Neither of the two provides a satisfactory solution for realizing a random peer sampling service in presence of Network Address Translators.

In [73], the presence of the fallback cache is expected to ensure that at any time a node has a high probability to know a node it can communicate with. Needless to say, such a simple mechanism does not ensure a uniform random sample of the nodes. Moreover, we have previously shown how —above a given threshold of natted nodes— the local view of nodes end up containing many potentially unreachable references, causing the network to get partitioned.

In [74], albeit the number of gossip exchanges performed is balanced among public and natted nodes, the number of gossip exchanges requested is not. The solution relies on an external peer sampling service to return to a node either nodes picked among the public nodes or nodes sitting behind the same NAT device. This does not ensure a uniform random sample. Moreover, if there is a high percentage of NATs, and that there are very few nodes behind the same NAT device (as in the case of home boxes), the public nodes would have to receive most of the gossip requests initiated by all nodes of the system.

2.5 Conclusion

Gossip protocols have received an increasing attention in distributed computing over the past decade as they are robust, simple and highly resilient to failures and node arrivals and departures (aka churn). Gossip random peer sampling protocols are extensively used in this area to build and maintain unstructured networks and provide each node with a random sample of the network in a fully decentralized way. Such protocols provide a core building block for gossip based dissemination [25], data aggregation [36], slicing [39], etc.

In gossip peer sampling, each node typically maintains a set of neighbors (called its local view) which it periodically exchanges with another node, picked from its view. This view is expected to be a sample of nodes picked uniformly at random among all nodes. Such protocols rely on the implicit assumption that a node is able to communicate with any node of its view. Yet, it is a well known fact that today, a large number of nodes sit behind Network Address Translators (NATs) [49, 50, 51, 52]. NAT devices allow several nodes with a private IP address to share a single public IP address and implement firewall-like mechanisms that drop unsolicited incoming messages. Consequently, the presence of NATs between nodes may prevent them from being able to communicate directly. To the purpose, we have shown how the presence of NATs impacts the properties of the gossip peer sampling. The local views of nodes end up containing many stale references that hurt the randomness of the provided sample, as well as the connectivity of the overlay, even in the absence of churn.

While the problem of limited connectivity has been addressed in the context of structured P2P networks [49, 53, 54], it has been mostly ignored in the area of gossip protocols. To the best of our knowledge, the only works dealing with NATs in

gossip protocol are [73, 74]. In [73], a node stores in a cache the nodes with which it successfully communicated in the past. The presence of this cache is expected to ensure that at any time the node has a high probability to know another node it can communicate with. Needless to say, such a simple mechanism cannot ensure that the network will remain connected. In [74], the proposed solution ensures balancing the number of gossip exchanges *performed* by public and natted nodes. Nevertheless, it does not ensure that the actual number of gossip exchanges *received* is balanced among public and natted nodes.

The problem of limited connectivity has also been addressed in works [34,75,76,77] which rely on an explicit structure to route messages on top of a gossip protocol. These solutions use proactive mechanisms to ensure that communication between natted nodes is possible under the implicit assumption that the network is fairly static.

In the next chapter, we present and evaluate $\mathcal{N}ylon$, a novel gossip-based peer sampling protocol which takes into account NATs, while ensuring the desired properties of the random peer sampling service.

3

Nylon: NAT-resilient Gossip Peer Sampling

Contents

In the previous chapter we have presented various gossip protocols implementing the peer sampling API. These protocols all assume that every node can communicate with any other node. We have shown that this is not the case on the Internet, due to the presence of Network Address Translators, which prevent the direct communication towards natted nodes. As a result, the local views of nodes end up containing many stale

references that hurt the randomness of the provided sample, as well as the connectivity of the overlay, even in the absence of churn.

A straightforward cope out is to associate every natted node to a public node. Provided the natted node accepts incoming messages from its associated public node, the latter can act as a relay between this natted node and any other node. Obviously, this imposes a significant overhead on public nodes which is not acceptable. Furthermore, the failure of public nodes would severely impact the overall connectivity and functioning of the system. The popular Skype P2P VoIP and Video calling service suffered from this very same issue recently [78], causing several hours of downtime of the system for millions of users worldwide.

To the best of our knowledge, no existing gossip-based peer sampling protocol accounts for the presence of NATs. In this chapter, we present $\mathcal{N}ylon$, a novel NAT-resilient gossip peer sampling protocol that leverages "NAT traversal techniques" [56, 57] to build a peer sampling protocol that works despite the presence of NATs. This protocol ensures that the communication between a node and its neighbors is always possible, even for the case when a neighbor is natted. In fact, as soon as a node picks a neighbor n in its local view to initiate a gossip with, it uses as relay the node which gave it this specific reference to set up a communication with n, and becomes itself a relay to n. Note that the node might rely on more than one relay to set up a communication with n. We evaluated $\mathcal{N}ylon$ through an extensive simulation study, showing that: *(i)* it ensures that the properties of the peer sampling are preserved in the presence of NATs, *(ii)* it evenly balances the relay load between nodes (be they public or natted), and *(iii)* it is highly resilient to churn.

The rest of this chapter is organized as follows. Section 3.1 presents general techniques to traverse Network Address Translators in order to communicate towards natted nodes. These techniques are the foundations upon which the $\mathcal{N}ylon$ protocol relies, as illustrated in our NAT-resilient protocol description in Section 3.2. Section 3.3 presents a set of optimizations to the base protocol in order to help reducing the length of the chain of relays which are used to communicate towards natted nodes. We evaluate $\mathcal{N}ylon$ in Section 3.4, while Section 3.5 concludes this chapter.

3.1 NAT traversal techniques

In the presence of NATs, the public IP address and port mapping and the filtering rules determine how nodes can communicate. As previously illustrated in the example scenario of Section 2.2.1, as long as a node behind a Full Cone (FC) NAT regularly sends or receives messages through the public address and port the NAT device assigned to it, it will have a valid filtering rule forcing the NAT device to forward it all incoming messages, regardless of the IP address and port they are coming from. Consequently, nodes behind a FC NAT can almost behave like public nodes. Rather, if the target node is behind a Restricted Cone (RC) NAT, Port Restricted Cone (PRC) NAT, or Symmetric (SYM) NAT, the source node willing to communicate with it will have to apply a so-called "NAT traversal technique" to be able to do so effectively.

There exist two different NAT traversal techniques [56, 57] depending on the

combination of source and target node's NAT type. These techniques rely on the use of *rendez-vous peers* (RVPs) which are able to exchange messages with both the source *and* the target nodes[1]. Table 3.1 summarizes which one of the two techniques should be used for a given combination of source and target node's NAT type. As explained in Chapter 2, we consider UDP-based implementations of gossip protocols. Consequently, we describe in this section NAT traversal techniques designed especially for UDP message exchanges. Similar techniques for TCP messages exchanges are also described in [56].

source \ target	public	RC	PRC	SYM
public	direct	hole punching	hole punching	hole punching
RC	direct	hole punching	hole punching	hole punching
PRC	direct	hole punching	hole punching	relaying
SYM	direct	(mod.) hole punching	relaying	relaying

Table 3.1 – NAT traversal technique to use for a given combination of source/target node's NAT type.

In the following two sections, we present the hole punching and relaying NAT traversal techniques.

3.1.1 Hole punching

We illustrate the *hole punching* technique in Figure 3.1, where for completeness we consider the case where both the source and target nodes are behind NATs. In the example, we assume no prior recent message exchange has taken place between the source and target node. We also assume that the source node knows the target's node mapped public IP address and port (e.g., querying the RVP for example).

First the source node sends a PING message to the target node. The message is dropped by the target node's NAT. Nevertheless, the action of sending it makes the source node's NAT device create a filtering rule which allows to forward to the source node subsequent messages coming *from* the target node. In jargon, the source node has opened a hole in its NAT to allow communication *from* the target node. Then, the source node sends an OPEN_HOLE message to the RVP, to declare that it wants to communicate with the target node. The RVP forwards the OPEN_HOLE message to the target node.

As soon as the target node receives the OPEN_HOLE message, it sends a PONG message to the source node. At this point, also the NAT device of the target node has a valid filtering rule allowing incoming messages *from* the source node. That is, also the target node has opened a hole in its NAT. Finally, as soon as the source node receives the PONG message, it can start sending the messages to the target node. A little more care has to be taken when the source node is behind a SYM NAT. In fact, the hole punching technique needs to be slightly modified: because the target node does not

[1] In the generic description of a RVP, this is usually a public node to which the source and target node periodically send PING messages to keep their filtering rules valid for the RVP.

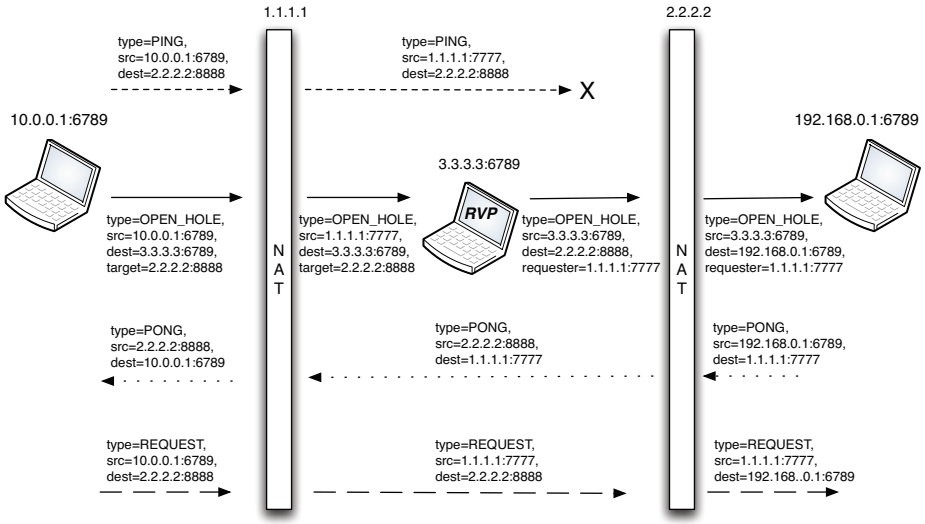


Figure 3.1 – NAT traversal via hole punching.

know the public IP address and port that has been assigned to the source node by the NAT (as they change on a per session basis), it uses the RVP to send the PONG message to the source node.

Note that for most combinations (i.e., those not involving SYM NATs), after the hole punching technique has been performed, the target node can also successfully send messages directly to the source node.

3.1.2 Relaying

We illustrate the *relaying* technique in Figure 3.2, again assuming no prior recent message exchange has taken place between the source and target nodes.

This technique can be used as a fallback solution when the hole punching cannot be used, i.e. when the target node is behind a SYM NAT and the source node is either behind a PRC NAT or a SYM NAT, or when the target node is behind a PRC NAT and the source node is behind a SYM NAT. This is due to the fact that the SYM NAT device assigns a different port to every new session started by a node towards another node, and this port is generally not known in advance[2]. The only possibility for sending messages effectively to the target node is then to use the RVP as a relay, as illustrated in Figure 3.2.

The RVP is a node which can communicate with both the source and target nodes. The source node then sends to the RVP a REQUEST message indicating which node is actually the intended target node. Upon reception of the message, the RVP successfully relays it to the target node.

[2]Some SYM NAT devices implement predictable port assignment strategies, making it possible to use so called “port prediction techniques”. This is nevertheless not always the case.

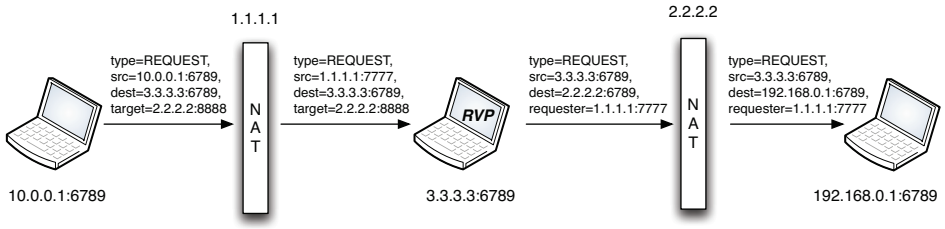


Figure 3.2 – NAT traversal via relaying.

3.2 The $\mathcal{N}ylon$ protocol

Before providing the description of the approach and pseudo-code of the $\mathcal{N}ylon$ protocol, we start by providing some preliminary observations which led to its design.

3.2.1 Preliminary observations

A possible approach for systematically allowing to traverse NATs is to employ public RVPs [79, 80]. This could allow to build a NAT-resilient peer sampling protocol as follows: a source node needing to communicate with a target natted node, would contact first the natted node's public RVP to forward an OPEN_HOLE message to the target node. This simple scheme suffers however from several drawbacks. First, the extra load induced by the presence of NATs is supported only by the public nodes. This creates an uneven distribution of the load, where public nodes contribute much more to the protocol than natted nodes. Another issue is the non uniform impact of failures of natted and public nodes. A public node's failure invalidates all references to the natted nodes bound to it. A possible solution would be to use several RVPs for each natted node. Nevertheless, this solution has its drawbacks as well. First, increased bandwidth consumption, caused by having to disseminate together with a node's reference also the list of its RVPs. Second, the additional need of having to maintain this list in presence of failures, or in order to fairly balance the RVPs bound to the natted nodes.

In order to overcome the limitations imposed by using only public RVPs, we designed a fully decentralized protocol that uses both natted and public nodes as RVPs. Relying also on natted nodes for implementing RVPs is challenging. Indeed, an RVP must be reachable by all nodes willing to communicate with nodes for which it acts as RVP. It is obviously impossible to ensure that a natted RVP will have valid NAT filtering rules for every nodes in the system.

The design of $\mathcal{N}ylon$ relies on the following two observations:

1. **In a gossip peer sampling protocol, it is not required that every node be reachable at any time by all nodes.** In fact, at a given time, the only nodes a given node *might* want to communicate with are those that are in its (continuously changing) local view. Consequently, each node only needs to be able to communicate with a very small, continuously changing, subset of nodes. Typically, in the gossip peer sampling protocols described in Section 2.1.2, this subset has a size in the order of *ln(N)*, *N* being the number of nodes in the system.

2. **In a gossip peer sampling protocol, although a node might want to communicate with any node in its local view at any time, it does not.** In fact, only a single node from its local view is picked upon each gossip operation. Moreover, it might even be the case that a node p in the local view of a node q is removed from q's local view without p and q actually gossiped with each other.

$\mathcal{N}ylon$ leverages these two observations to build a NAT-resilient gossip-based peer sampling protocol in which all nodes can act as RVPs. The first observation is taken into account by implementing a hole punching protocol for only a subset of the system. The second observation is taking into account by implementing a *reactive* hole punching protocol which consists in performing the actual hole punching protocol between two nodes only when needed, namely when a gossip between the two nodes is initiated. This avoids to systematically send an OPEN_HOLE message to all the natted nodes that a node adds in its view.

3.2.2 Protocol description

The main idea of $\mathcal{N}ylon$ is to implement *reactive* hole punching. Intuitively, this works as follows: a node only performs hole punching towards nodes it gossips with. Hole punching is implemented using a chain of RVPs that forward the OPEN_HOLE message until it reaches the gossip target node.

The chain of RVPs is built as follows. Consider the case of a node $n1$ shuffling with a node $n2$. After having performed hole punching towards $n2$ (using a chain of RVPs), nodes $n1$ and $n2$ can directly communicate with each other. Thus, they both become RVP for each other. Consider now that later, one of them, say $n2$, shuffles with a node $n3$ and gives it a reference to $n1$. Before shuffling, node $n2$ performs hole punching towards $n3$. Consequently, as between $n1$ and $n2$, node $n2$ and $n3$ both become RVP for each other. Finally, consider that $n3$ shuffles with a node $n4$ and gives it a reference to $n1$. A chain of RVPs has thus been created, as shown in Figure 3.3. This chain allows $n4$ to shuffle with node $n1$. For this purpose, it performs hole punching towards node $n1$ by sending an OPEN_HOLE message to $n3$ that will in turn forward it to $n2$, that will forward it to $n1$.

As we can see from the figure, an `RVP` field is added in the entry data type for the purpose of specifying the next RVP to use to establish communication towards a node. When this field is empty ("-"), it means that direct message exchange with the given node is possible (e.g., either because the given node is public, or because a hole punching has been performed).

Furthermore, a time to live (`TTL`) field is also added to the entry data type, which is only used for references to natted nodes. The time to live specifies the remaining duration of the filtering rule of the chain of RVPs towards the given natted node: being part of the entry data type, TTLs are in fact exchanged by nodes during shuffling. Also, the TTLs fields are decreased every shuffling period, and refreshed every time a message from a given RVP is received. Note that the TTL mechanism assumes that

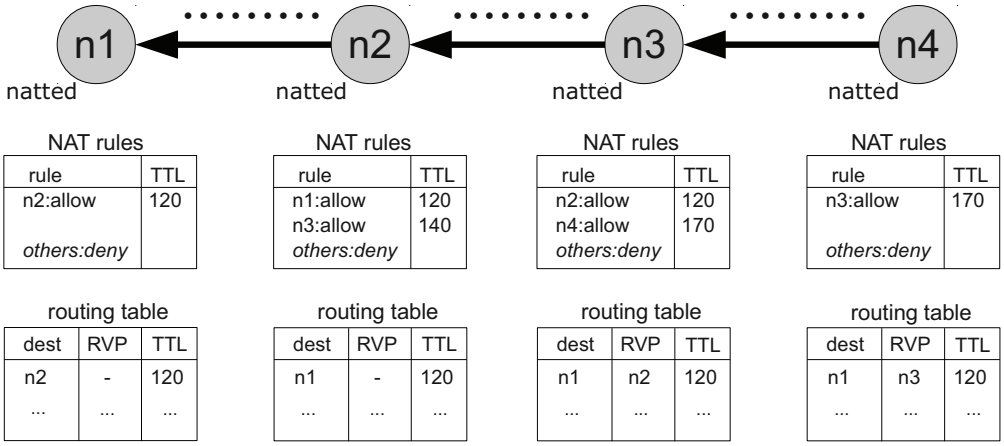


Figure 3.3 – $\mathcal{N}ylon$ operating principle.

there is a known upper bound on the latency between each pair of nodes[3].

Let us note that, in addition to maintaining entries in its local view, each node maintains also entries in a *routing table* data structure, in order to be able to route messages towards nodes no longer present in its local view. The routing table data structure is in fact maintained as follows. Each time an entry is added (or updated) in the local view, it is also added (or updated) in the routing table[4]. Whenever an entry is removed from the local view (e.g., as part of the merge and truncate procedure), it remains in the routing table as long as the TTL does not expire. This allows a node to route messages along the chain of RVPs even when the intended target is no more among its current local view neighbors. Moreover, if the TTL of an entry expires, the entry is removed from the routing table and from the local view (if it is currently present there as well). As en effect of continuously exchanging neighbors among nodes as part of the gossiping protocol, also RVPs in $\mathcal{N}ylon$ are constantly changing and following the reactive flavor of the protocol. Also, RVPs do *not* proactively refresh holes by sending keep alive messages.

3.2.3 Pseudo-code

The pseudo-code of the $\mathcal{N}ylon$ protocol is presented in Figure 3.4. The protocol is built on top of the gossip-based peer sampling framework presented in Section 2.1.2. The basis of the protocol is the (push/pull, rand, swapper) configuration, which in the original study [27] (without NATs) has been shown to obtain good load balancing among the nodes. The only additions to the protocol are for handling NAT traversal techniques and implementing the RVP chaining mechanism previously described.

[3] If the upper bound is not met, this could cause a node's reference in the routing table to become stale. We show in Section 3.4 that the protocol resists to the simultaneous departure of 40% of the nodes. This shows that the protocol would resist to 40% of the message exchanges simultaneously exceeding the upper bound.

[4] Technically, the two data structures contain the very same entry object.

```
1  every shuffling_period units do
2    decrease_routing_table_ttls()
3    target ← selectPeer(view)
4    view_s ← select_neighbors(view) ∪ (self)
5    // If souce node can communicate directly to gossip target
6    if (target is public or next_RVP(target) = target) then
7      send ⟨REQUEST, view_s, self, target⟩ to target
8    elif ((target is SYM and self is PRC) or self is SYM) then
9      // Else either NAT traversal via relaying..
10     send ⟨REQUEST, view_s, self, target⟩ to next_RVP(target)
11   else
12     // ..or NAT traversal via hole punching
13     send ⟨OPEN_HOLE, self, target⟩ to next_RVP(target)
14     // If source is natted, open hole on its NAT for the target
15     if self is not public then
16       send ⟨PING⟩ to target

17 on receive ⟨REQUEST, view_s, src, dest⟩ from p do
18   update_next_RVP(p,p,HOLE_TIMEOUT)
19   if dest ≠ self then
20     // Relaying request to the next RVP along the chain
21     send ⟨REQUEST, view_s, src, dest⟩ to next_RVP(dest)
22   else // self node is the gossip target node
23     view_t ← select_neighbors(view)
24     if (src is SYM and self is not public)
          or (self is SYM and src is not public) then
25       // Responding via relaying along the chain of RVPs
26       send ⟨RESPONSE, view, src⟩ to next_RVP(src)
27     else
28       // Responding directly to the gossip source node
29       send ⟨RESPONSE, view, src⟩ to src
30     view ← merge_and_truncate(view, view_s)
31     update_routing_table(view_s)

32 on receive ⟨RESPONSE, view_t, dest⟩ from p do
33   update_next_RVP(p,p,HOLE_TIMEOUT)
34   if dest ≠ self then
35     // Relaying response to the next RVP along the chain of RVPs
36     send ⟨RESPONSE, view, dest⟩ to next_RVP(dest)
37   else
38     view ← merge_and_truncate(view, view_t)
39     update_routing_table(view_t)

40 on receive ⟨OPEN_HOLE, src, dest⟩ from p do
41   update_next_RVP(p,p,HOLE_TIMEOUT)
42   if dest = self then
43     send ⟨PONG⟩ to src
44   else
45     send ⟨OPEN_HOLE, src, dest⟩ to next_RVP(dest)

46 on receive ⟨PING⟩ from p do
47   update_next_RVP(p,p,HOLE_TIMEOUT)
48   send ⟨PONG⟩ to src

49 on receive ⟨PONG⟩ from p do
50   update_next_RVP(p,p,HOLE_TIMEOUT)
51   send ⟨REQUEST, view, self, p⟩ to p
```

Figure 3.4 – The *Nylon* protocol: pseudo-code.

The routing table code is abstracted by four methods. The method `next_RVP()` returns the next RVP to be used to route a message to a given node: if the destination is directly reachable (because either the destination is public or the node acts as an RVP for the destination), the method returns the destination itself. The method `update_next_RVP()` is used to create a new entry or refresh the TTL of an existing entry in the routing table for the node from which a message is received. The method `update_routing_table()` is called to update the routing table taking as parameter the entries that have been received during a view shuffling. This method adds an entry in the routing table for each exchanged node reference, and specifies that the RVP for each of the references to natted nodes among them to be the node with which the shuffle has just been performed. Finally, the method `decrease_routing_table_ttls()` is used to (periodically) decrease the TTLs of the existing routing table entries accounting for the elapsed interval since the last gossip execution. The routing table entries whose TTL drops to zero are discarded from the routing table. Furthemore, every local view entry referencing one of these nodes or having as RVP one of these nodes, is discarded as well.

3.2.4 Discovery of NAT type and hole timeout

As we can note in the pseudo-code, nodes need to be aware of their NAT type and TTL duration, as well as those of the nodes they want to gossip with, in order to be able to decide which communication strategy to adopt: direct, hole punching, relaying.

$\mathcal{N}ylon$ does not directly provide a built-in solution to discover the NAT type and TTL durations, but it can rely on existing methods. In fact, a possible viable and practical solution for a node to discover its NAT type before joining the $\mathcal{N}ylon$ overlay, is to run the client part of the STUN protocol [72], and contact a public STUN server available on the Internet[5]. STUN is a client/server protocol which allows a client to discover whether it is sitting behind a NAT, and if that is the case, what is the type of the NAT and the public IP address that the NAT device assigned to the client as part of the mapping procedure. Similarly to the way a node can discover its NAT type, the node can also discover the duration of its NAT hole timeout by having the server attempt to delay sending a message to the client (and this one, sending back an ack to attest the reception) [82].

The NAT type information (either manually set or discovered), and its initial TTL duration, can then respectively be stored in the entry data type in the fields `nat_type` and `ttl`. And, by means of the regular view shuffling, nodes can spread their NAT type information and be aware of the NAT type of their neighbors.

We point out that the aforementioned methods to discover a node's NAT type and TTL duration could be implemented within the $\mathcal{N}ylon$ protocol itself (having each node implement both the client and server side part), and be run by a node upon joining the overlay network.

[5]The STUN servers are listed in [81].

3.3 Optimizations

We present in the following a set of optimizations to $\mathcal{N}ylon$. Their purpose is to reduce the length of the chains of RVPs, and thus the latency in establishing direct communication towards natted nodes.

3.3.1 Entry optimization predicate

Let us assume that a node s has an entry P in its local view referencing the natted node q. When s receives an entry E referencing node q, it updates its local entry if: *(i)* E has a TTL greater or equal than that of P, and *(ii)* E has an "estimated path length" towards q which is strictly lower than that currently stored in P.

The motivation behind the first point is to guarantee the invariant condition that the TTL of a hop along the chain of RVPs is equal or greater than that of the previous hop(s). Violating this invariant can cause routing problems: OPEN_HOLE messages might in fact be dropped along the chain of RVPs because routing tables entries have been discarded due to an expired TTL at one of the hops.

The motivation behind the second point, requiring the strictly lower estimated path length, is the following. If a node s receives from a node d an entry E which references a natted node q, the length of the path to reach q from s using d as its next RVP would then in fact be one hop longer than that from d itself to q. In order to provide a node with the information about the (estimated) length of the chain of RVPs towards a given natted node, the entry data type is extended with an `estimated_path_length` field, which is initialized as follows. Taking again Figure 3.3 as example, at first node $n1$ successfully communicates directly with the node $n2$. At this point, both $n1$ and $n2$ have an estimated path equal to 1 towards each other. When $n2$ gives to $n3$ its entry referencing $n1$, the node $n1$ being natted, $n3$ stores $n2$ as its RVP towards $n1$ and increments by one the estimated path information received from $n2$.

The optimizations described in the following sections aim at improving the accuracy of the `estimated_path` field, and at using chains of RVPs which lead to shorter paths towards natted nodes. For each of the optimizations, we first present their rationale, then we illustrate their functioning via a graphical example, and finally we provide the details of how to implement the optimization within the protocol.

3.3.2 Optimization 1: Shuffled entries optimization

Rationale. This optimization leverages the shuffling among nodes. It allows a node to possibly optimize its local entries referencing the same destinations referenced by the entries received via shuffling. In fact, if a received entry has a path to a destination q which is shorter than the path previously known by the receiver node, this node will update its (local view and/or routing table) entry referencing q to route messages along the same path used by the sender node.

Example. Figure 3.5 illustrates the "shuffled entries optimization". All nodes represented in the example are natted. Node s participates in a view exchange with node d, and receives d's entry referencing node q. The path to reach q using d as the next RVP

is one hop shorter ($\rightarrow d \rightarrow q$) than the path previously known by s ($\rightarrow n1 \rightarrow n2 \rightarrow q$). Consequently, s updates its entry referencing q to use d as its next RVP.

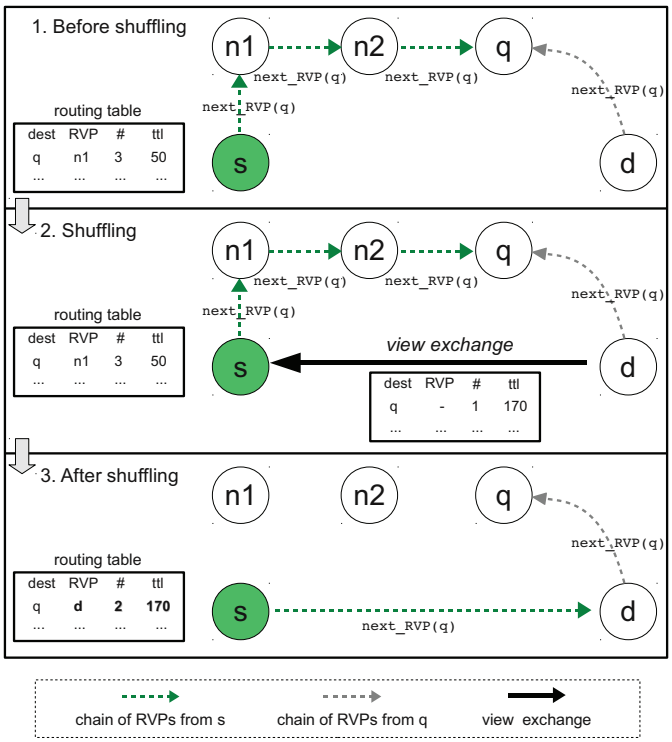


Figure 3.5 – Shuffled entries optimization: example. Node s optimizes its path towards node q, passing from 3 hops ($\rightarrow n1 \rightarrow n2 \rightarrow q$) to 2 hops ($\rightarrow d \rightarrow q$).

Details. When a node receives a shuffle request/response, it tests each entry contained in the shuffle request/response against the optimization predicate. When the predicate holds, the data of the received entry are used by the node to update its local entry. Note that this optimization leverages all the entries received via shuffling, but only the ones kept from the view merge and truncate procedure are stored in the local view. The other entries only end up being updated in the routing table.

3.3.3 Optimization 2: Backward optimization

This optimization leverages the transit of an OPEN_HOLE message. We describe the optimization in two parts: the "backward optimization towards the source", and the "backward optimization towards the destination". Note that successfully applying the first optimization is a prerequisite to apply the second one.

3.3.3.1 Backward optimization towards the source

Rationale. This optimization allows each node forwarding an OPEN_HOLE message along the chain of RVPs to possibly optimize its entry referencing the source node s which originated the OPEN_HOLE. In fact, if the path traversed by the OPEN_HOLE from the source node s to a node p is shorter than the path p knows to reach s, then p

will update its entry referencing s to route messages backward along the path traversed so far by the OPEN_HOLE. To do so, p will use the previous hop as its next RVP for s.

Example. Figure 3.6 illustrates the "backward optimization towards the source". All nodes represented in the example are natted. Node s originates an OPEN_HOLE message targeted to node d, which is forwarded along the chain of RVPs. Each RVP along the chain increments in the OPEN_HOLE the number of hops traversed so far from s. Node p receives the OPEN_HOLE from $n1$, and learns that the path to reach s passing via $n1$ is shorter ($\rightarrow n1 \rightarrow s$) than the path via $n2$ it previously knew ($\rightarrow n2 \rightarrow n3 \rightarrow s$). Consequently, p updates its entry referencing the source node s to use $n1$ as its next RVP.

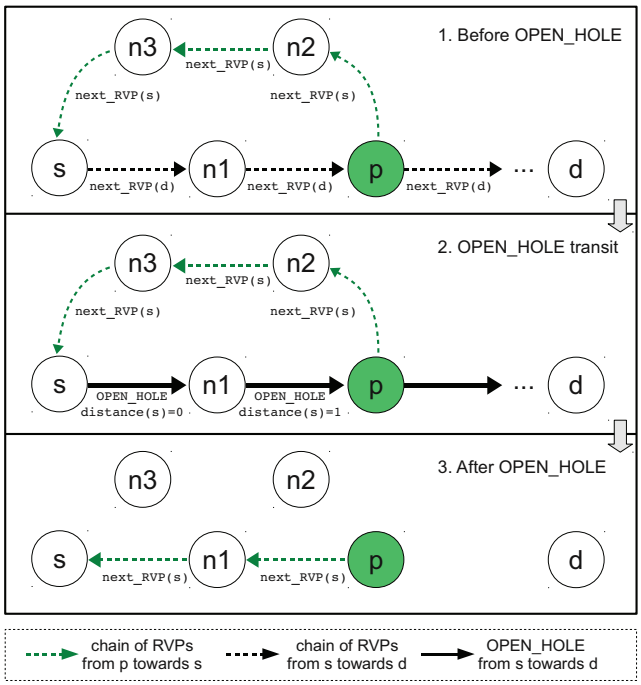


Figure 3.6 – Backward optimization towards the source. Node p optimizes its path towards the source node s, passing from 3 hops ($\rightarrow n2 \rightarrow n3 \rightarrow s$) to 2 hops ($\rightarrow n1 \rightarrow s$).

Details. The OPEN_HOLE message data type is extended with a `backward_entry` field. Whenever a hop along the OPEN_HOLE chain forwards the OPEN_HOLE message to the next hop, it also stores in this field its local entry referencing the source node s[6]. Upon reception of the OPEN_HOLE, the current hop tests the contained `backward_entry` against the optimization predicate. Analogously to what explained in Section 3.3.2, if for the `backward_entry` the optimization predicate holds, the current hop will update its local entry using the data contained in the received entry.

[6]When it is the source node s sending the OPEN_HOLE to the first hop of the chain, the field `backward_entry` is left empty. In fact, the first hop is in direct communication with the node s.

3.3.3.2 Backward optimization towards the destination

Rationale. The OPEN_HOLE in transit can also be leveraged to optimize the path towards the destination node d, reaching it backwardly through the source node s. In fact, once the hole punching procedure completes, the source node s and the destination node d can communicate directly, hence "extending" the validity of the backward path (from a hop along the chain of RVPs, towards the source node s) up to the destination node d itself. According to this, if the path (plus one additional hop) traversed by the OPEN_HOLE from the source node s to a node p is shorter than the path p currently knows to reach d, then p will update its entry referencing d to route messages backward along the path traversed by the OPEN_HOLE. To do so, p will use the previous hop as its next RVP for d.

Example. Figure 3.7 illustrates the "backward optimization towards the destination". Again, all nodes in the example are natted. When node p receives the OPEN_HOLE from s, it learns that the path to reach d via s ($\rightarrow s \rightarrow d$) is shorter than the path via $n1$ it previously knew ($\rightarrow n1 \rightarrow n2 \rightarrow d$). Consequently, p updates its local entry referencing the destination node d to use s as its next RVP.

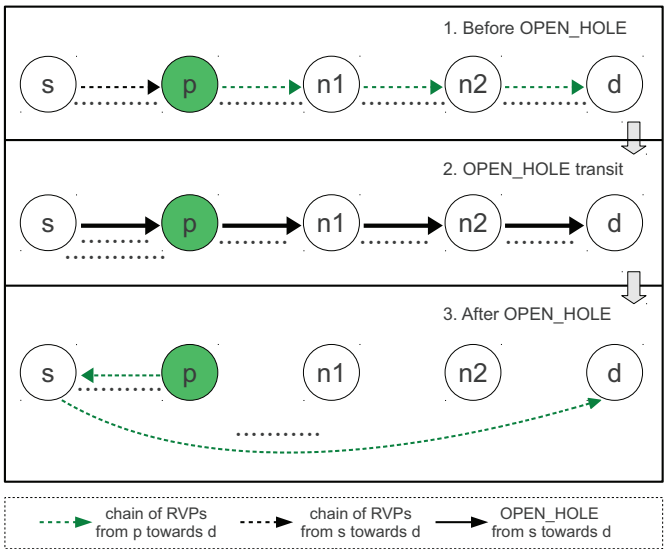


Figure 3.7 – Backward optimization towards the destination. Node p optimizes its path towards the destination node d, passing from 3 hops ($\rightarrow n1 \rightarrow n2 \rightarrow d$) to 2 hops ($\rightarrow s \rightarrow d$).

Details. First, a hop node p along the chain of RVPs creates a "candidate" entry E referencing d. This entry E leads to a path which passes through s, going backward along the same path which the OPEN_HOLE traversed to reach p. The entry E is initialized in the following way. The RVP and TTL are those of p's local entry referencing s. As estimated path length value, that of its local entry referencing s, incremented by one unit. Then, in order for a hop node p to actually replace its previous entry referencing d with this candidate entry E, two conditions must hold. First, the node p is the first RVP along the chain, or it has successfully applied the backward optimization towards the source s (cfr. previous Section 3.3.3.1). Second, the candidate entry E referencing d

passes the optimization predicate. Finally, if the optimization is applied, the concrete changes to the entry referencing the destination node d are delayed, by a gossip period. In fact, direct communication between the source node s and the destination node d is only possible once the hole punching procedure has completed. This is not the case yet while nodes are still forwarding the OPEN_HOLE message. An immediate change to the routing table entry referencing the node d would then result in a possible (transient) invalid route to d.

3.3.4 Optimization 3: Forward optimization

Rationale. This optimization is the "symmetric" version of the backward optimization previously described. It still aims at optimizing the entries referencing the source node s and destination node d of the OPEN_HOLE message. But, in this case, the optimized paths towards s and towards d are oriented along the same direction of the path traversed by the OPEN_HOLE. After receiving the OPEN_HOLE, the destination node d originates a so called ACK_OPEN_HOLE message. This message is sent to s by traversing backward the very same path originally traversed by the OPEN_HOLE. If the path traversed by the ACK_OPEN_HOLE from the destination node d to a node p is shorter than the path p knows to reach d, then p will update its local entry referencing d to route messages backward with respect to the path traversed by the ACK_OPEN_HOLE (i.e., along the same direction of the path originally traversed by the OPEN_HOLE to d). To do so, p will use its previous ACK_OPEN_HOLE hop as its next RVP for d. Similar reasoning applies to optimize the path towards s by passing through d.

Example. Figure 3.8 illustrates the "forward optimization". Again, all nodes in the example are natted. When node p receives the ACK_OPEN_HOLE from $n3$, it learns that the path to reach d via $n3$ is 2 hops distant. Consequently, p updates its (inaccurate) knowledge of the distance to d. This also means that the path to reach s passing through d ($\rightarrow n3 \rightarrow d \rightarrow s$) is shorter than the path via $n4$ that p previously knew ($\rightarrow n4 \rightarrow n5 \rightarrow n6 \rightarrow s$). Consequently, p updates its local entry referencing the source node s to use $n3$ as its next RVP.

Details. To realize this optimization, a new message type is introduced, which is called ACK_OPEN_HOLE, and it is used as follows. Once the OPEN_HOLE message reaches the destination node d, this node originates an ACK_OPEN_HOLE message which is sent along the very same chain of RVPs that originally forwarded the OPEN_HOLE. In order to be able to traverse backward the original path, when nodes forward an OPEN_HOLE message, they (temporarily) associate to its message id and source node id, the identifier of the previous RVP along the chain. Analogously to what done in the case of the backward optimization, when a node forwards the ACK_OPEN_HOLE message to the next hop along the path towards the source node s, it stores in a `forward_entry` field its local entry referencing the destination node d[7]. In a first place, upon reception of the ACK_OPEN_HOLE, a hop along the chain of RVPs attempts to optimize its local entry referencing the destination node d. This is actually the "forward optimization towards the destination". If it succeeds in applying this optimization, it then attempts to optimize its local entry referencing the source node s and passing through the node d.

[7] When the destination node originates the ACK_OPEN_HOLE, the field `forward_entry` is empty.

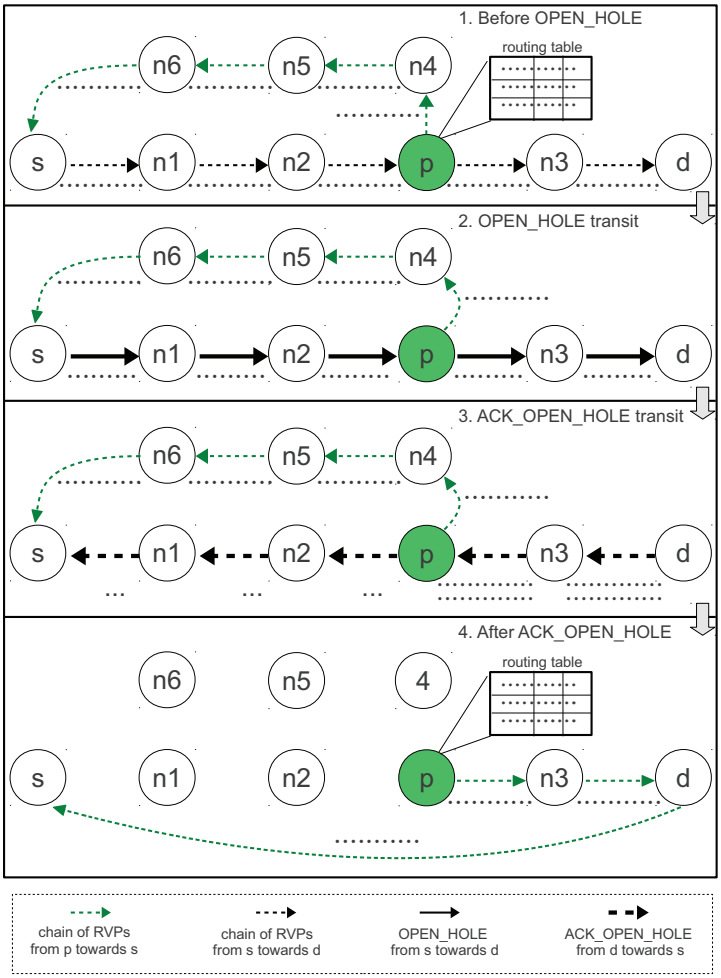


Figure 3.8 – Forward optimization towards source and destination. After the transit of the ACK_OPEN_HOLE, p updates its knowledge of the path length towards d (from the inaccurate 3 hops, to the actual 2 hops), and optimizes its path towards the source node s passing from 4 hops ($\rightarrow n4 \rightarrow n5 \rightarrow n6 \rightarrow s$) to 3 hops ($\rightarrow n3 \rightarrow d \rightarrow s$).

This is actually the "forward optimization towards the source". The conditions to apply them are the same ones described in Section 3.3.3.1 and Section 3.3.3.2.

3.4 Evaluation

In this section, we report the results of the evaluation of the $\mathcal{N}ylon$ protocol. We evaluate in both the base and optimized variants (that we refer to as $\mathcal{N}ylon$++). We show that the protocol: *(i)* achieves a uniform random peer sampling, *(ii)* constructs chains of RVPs with reasonable length, *(iii)* is highly resilient to churn, and *(iv)* induces little overhead and homogeneously balances the load among natted and public nodes.

Experimental settings. To the best of our knowledge, existing P2P simulators do not take into account NATs. We thus developed a Java cycle-based event-driven simulator à la PeerSim [62], which takes into account the four types of NATs described in Section 2.2. Message latency is set to $50ms$, the NAT hole timeout is set to $90s$ (a fair estimate of real values [50]), and the shuffling period is set to $5s$.

Although we experimented with all four types of NATs, experiments with FC NAT are not reported. In fact, as explained in Section 2.2.1, nodes behind FC NATs behave similarly to public nodes as long as they frequently send or receive messages. The distribution we used is the following: 50% of RC NATs, 40% of PRC NATs, and 10% of SYM NATs. This distribution mostly accounts for the fact that SYM NAT behavior, which imposes relaying to be overcome, constitutes the smallest proportion of deployed NATs, according to findings and recent measurements [50]. Note that we evaluated other distributions and obtained comparable results. Experiments were conducted on a 1,000 nodes system, with a view size set to 10 and shuffle length set to 5. Unless explicitly stated differently, all experiments were run with 30 different seeds, and the results reported are the average of those 30 runs. Finally, experiments lasted a long enough time to observe, most of the time, a negligible variance. However, any non negligible observed variance is indicated in the graphs.

We do not report the preliminary analysis results: we indeed first checked that during execution there were no network partitions in absence of churn, and that no stale references were present in the views of the nodes. We start then by presenting an analysis of the randomness of the protocol. Then, we study the average latency (expressed in terms of length of the chain of RVPs) incurred in establishing a communication towards natted nodes. Afterwards, we study the resiliency of $\mathcal{N}ylon$ to churn. Finally, we show the network bandwidth overhead with respect to standard gossip peer sampling protocols.

3.4.1 Randomness

Similarly to what is done in [27], we assessed the randomness properties of the graph formed by nodes: each node is a vertex, and we consider that there is a directed edge from node p to node q when q is in the local view of p. To determine whether the graph is a random graph, we study the *in-degree* of nodes and *clustering coefficient* of the graph. Moreover, to compare with results presented in Section 2.3.3, we present the

percentage of non-stale references towards natted nodes in views.

We present the results obtained for the following three protocols: *(i)* a reference (push/pull, rand, swapper) instance of the generic peer sampling framework; *(ii)* the base $\mathcal{N}ylon$ protocol; *(iii)* the $\mathcal{N}ylon$ protocol with optimizations ($\mathcal{N}ylon$++). While for the first case we simulated an overlay without NATs, in the latter two cases the percentage of NATs was set to 70% of the nodes of the system, which is a realistic value on the Internet [49].

Following the study conducted in [27], to study the stability and convergence of the graph properties during the execution of the protocols, they are evaluated under the following three different bootstrapping scenarios:

Growing. There is initially only one node (the bootstrapping node). At each shuffling period, 50 new nodes are added, with their views initialized only containing the bootstrapping node. The addition of nodes continues periodically until the total number of nodes is reached.

Ring. Nodes are randomly ordered on a ring space, and each node's view is filled with an equal number of closest nodes on its *left* and *right* side on the ring. In the cases of $\mathcal{N}ylon$ and $\mathcal{N}ylon$++, whenever a natted node is put into the view of a node as part of the bootstrapping procedure, a NAT hole is artificially opened (and routing tables are updated accordingly) in order to guarantee that the reference will not be initially stale.

Random. Views of the nodes are initially filled with randomly chosen nodes. In the cases of $\mathcal{N}ylon$ and $\mathcal{N}ylon$++, we artificially open NAT holes as described in the previous scenario.

As suggested in [27], to properly show the dynamic properties of the protocols, results show the behavior from a single run. Nevertheless, we ran all the scenarios using 30 different seeds obtaining very similar results. The run duration was set to 300 shuffling periods, which was enough to observe the convergence of the behavior of the different protocols.

3.4.1.1 In-degree

The in-degree of a node p is equal to the number of nodes storing a reference to p in their local view. From a load-balancing perspective, nodes should have a comparable in-degree. We depict: *(i)* the standard deviation of the in-degrees during the protocol execution; *(ii)* the distribution of the in-degree at the end of the execution.

In-degree standard deviation. Figure 3.9 shows the standard deviation of the in-degree of nodes, during the protocol execution, for the three bootstrapping scenarios. We can make three remarks. First, taking individually each protocol, it consistently converges around the same value of the standard deviation in each bootstrapping scenario. This means that the starting topology does not affect the behavior exhibited by the protocols. Second, the standard deviation of the in-degree of nodes is fairly low. This means that the load on the nodes is fairly well balanced. Third, $\mathcal{N}ylon$ and

$\mathcal{N}ylon$++ manifest very similar behavior with respect to the reference protocol. This means the modifications to the original peer sampling protocol, to take into account NATs, have not affected the original properties of the resulting graph.

In-degree distribution. Figure 3.10 shows the in-degree distribution of the various protocols at the end of the simulation, for the three bootstrapping scenarios. Two things can be noted. First, we observe that all protocols manifest very comparable behavior. Second, the in-degree distribution has a quite narrow width. In fact, the in-degree value corresponding to the peak of the curves is slightly below the view size used in the configuration (10 nodes), and very few nodes have a much higher in-degree. This means the load on nodes is fairly well balanced.

3.4.1.2 Clustering coefficient

The clustering coefficient of a graph is defined as follows [83]. Consider a vertex p and its neighbors. Then, consider the number of all edges which can exist among these neighbors. The clustering coefficient C_p of p is defined as the fraction of these edges which actually exist. The higher the value of C_p, the more the neighbors of p resemble a clique. The clustering coefficient C of the whole graph is then computed by averaging the clustering coefficients of all its vertices. From the point of view of fault tolerance, it is preferable to have a low clustering coefficient C, as this means that the graph is not clustered.

Figure 3.11 shows the clustering coefficient during protocol execution, for the three bootstrapping scenarios. We can make three observations. First, taking each protocol individually, the clustering coefficient converges around the same value in each scenario. Second, all protocols behave very similarly: the modifications to the protocol to handle NATs have not impacted the behavior of the protocol with respect to the clustering coefficient. Third, the average clustering coefficients are very low (between 0.02 and 0.03), which, as explained before, is a desirable characteristic.

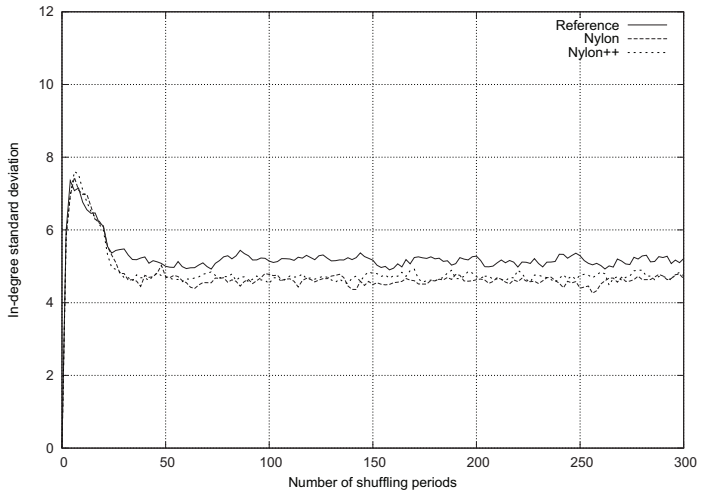
3.4.1.3 Non-stale references towards natted nodes

Figure 3.12 shows the percentage of non-stale references towards natted nodes in the local views of nodes during the execution of $\mathcal{N}ylon$ and $\mathcal{N}ylon$++. Recall that there are 70% of natted nodes in the system. We do thus expect local views of nodes to approximately contain around 70% of non-stale references towards natted nodes. Figure 3.12 shows that this is indeed the case for all the three bootstrapping scenarios.

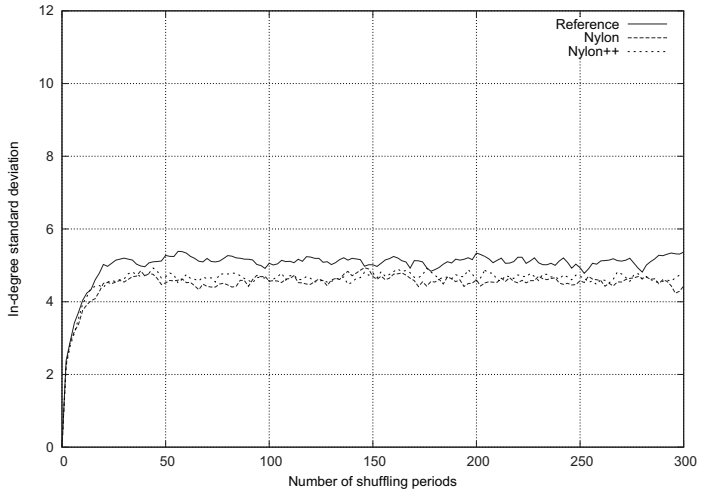
3.4.2 RVP chains length

We study the average length of the chain of RVPs between two nodes. The longer the length, the larger the time to initiate a message exchange with a natted node. In Figure 3.13 we observe that the number of RVPs increases with the percentage of NATs. Also, the optimizations in $\mathcal{N}ylon$++ reduce the path length with respect to the base protocol up to a 15% in a configuration with 90% of NATs. Overall, the average length of the chains of RVPs, even in presence of high percentage of NATs, remains very reasonable (smaller than four hops, considering its optimized flavor).

(a) Growing scenario



(b) Ring scenario



(c) Random scenario

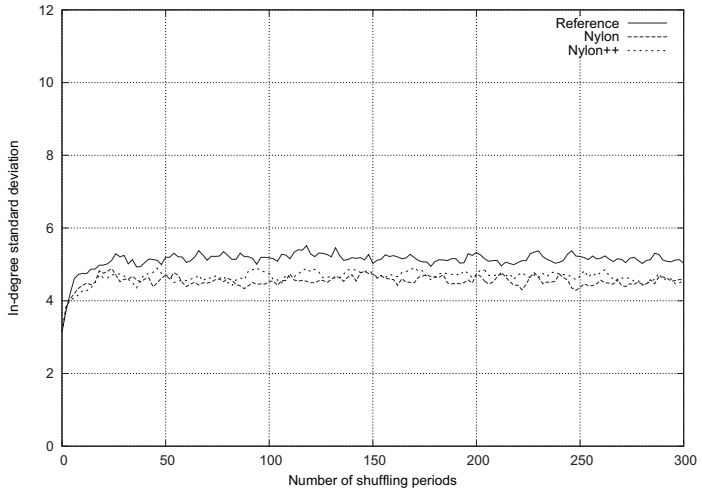
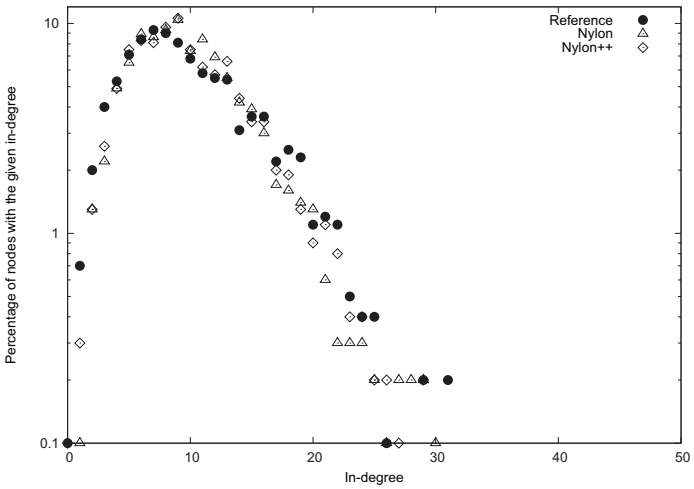
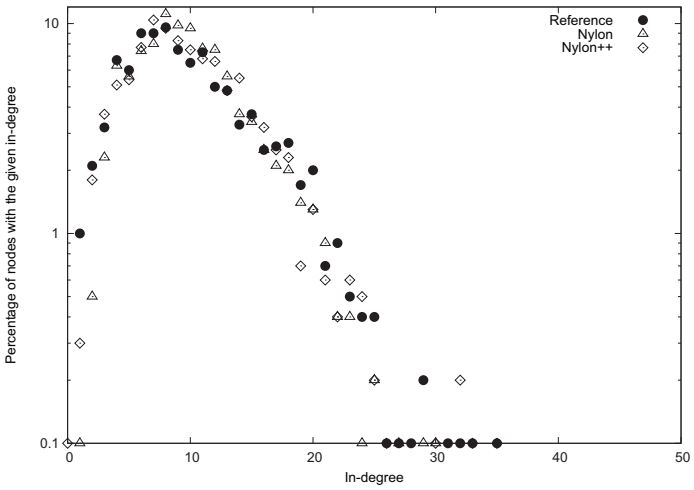


Figure 3.9 – In-degree standard deviation during protocol execution for the three bootstrapping scenarios.

(a) Growing scenario



(b) Ring scenario



(c) Random scenario

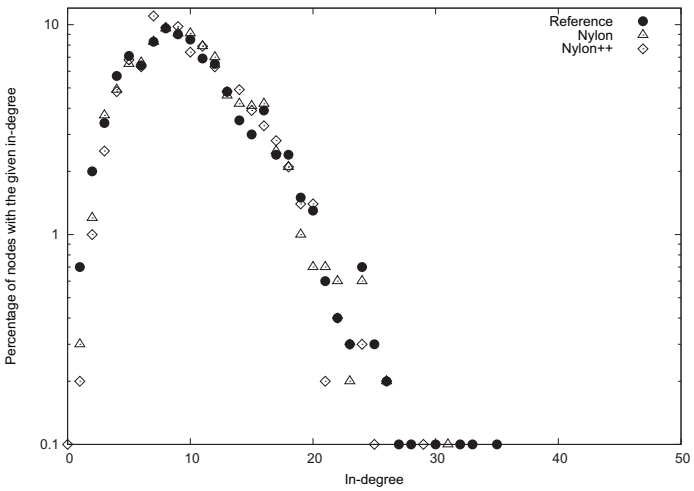
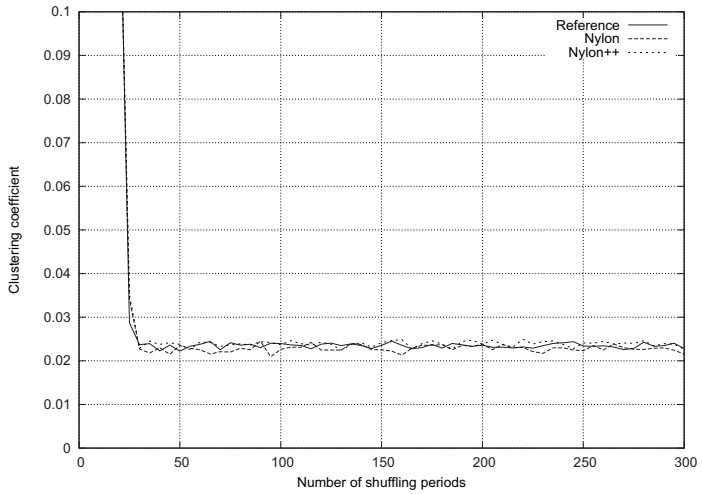
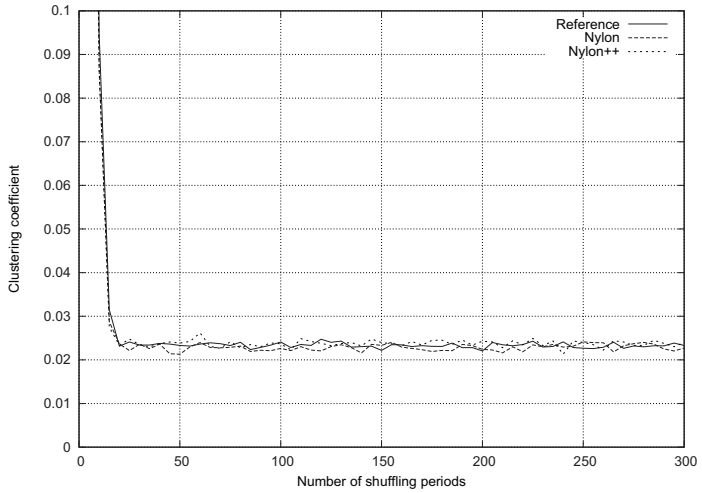


Figure 3.10 – In-degree distribution (Y axis in logarithmic scale) for the three bootstrapping scenarios at the end of the simulation.

(a) Growing scenario



(b) Ring scenario



(c) Random scenario

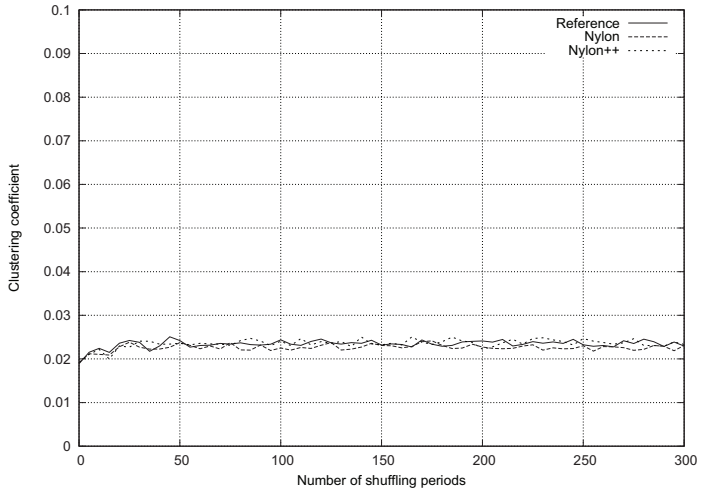
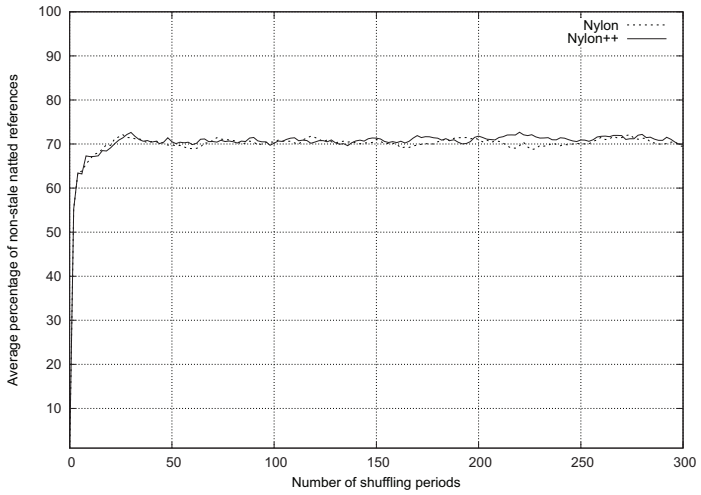
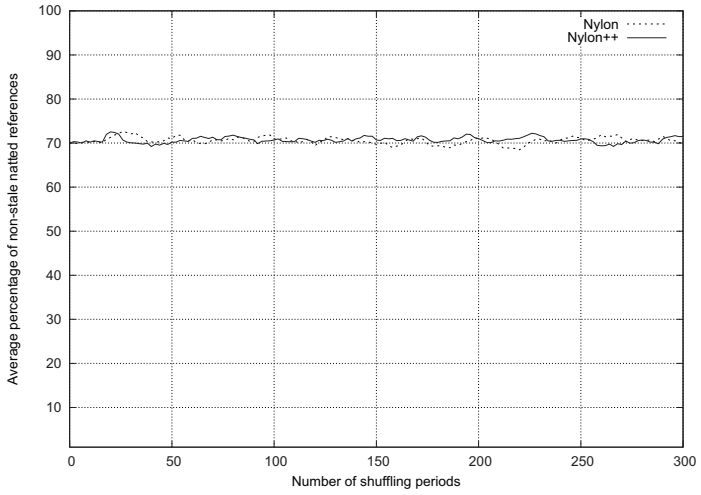


Figure 3.11 – Clustering coefficient during protocol execution for the three bootstrapping scenarios.

(a) Growing scenario



(b) Ring scenario



(c) Random scenario

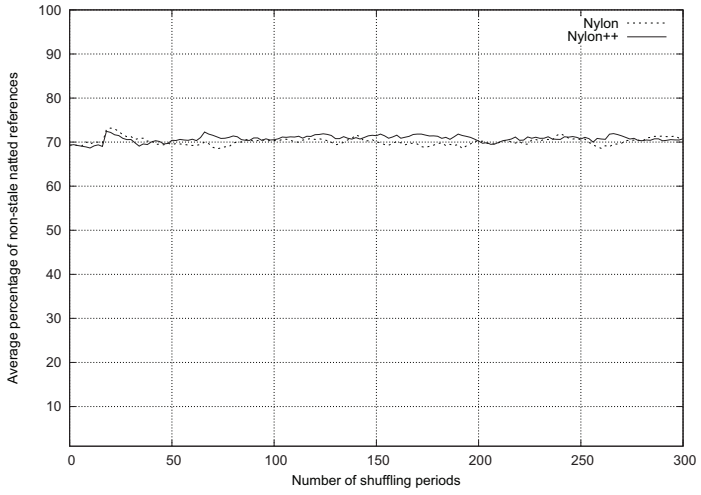


Figure 3.12 – Percentage of non-stale references towards natted nodes in the views during protocol execution, for the three bootstrapping scenarios.

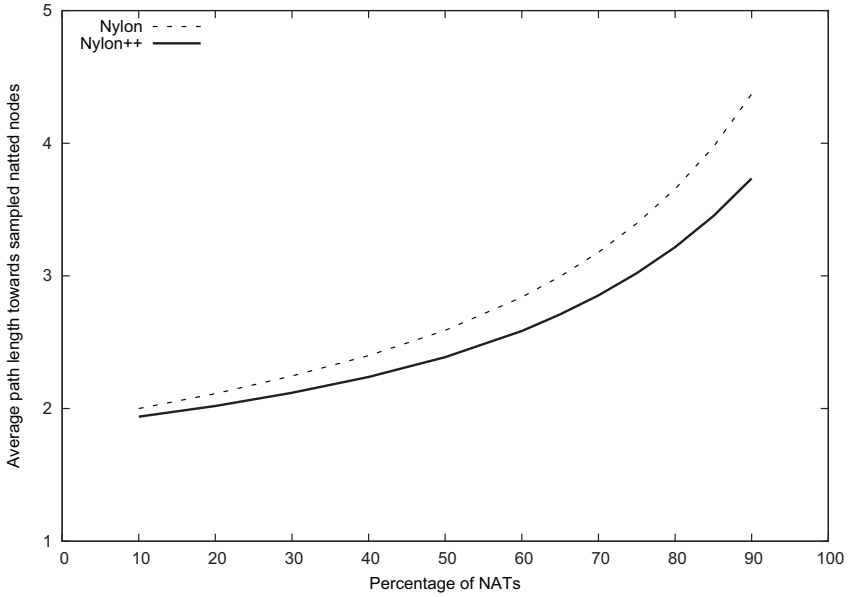


Figure 3.13 – Path length towards selected natted nodes.

3.4.3 Churn resiliency

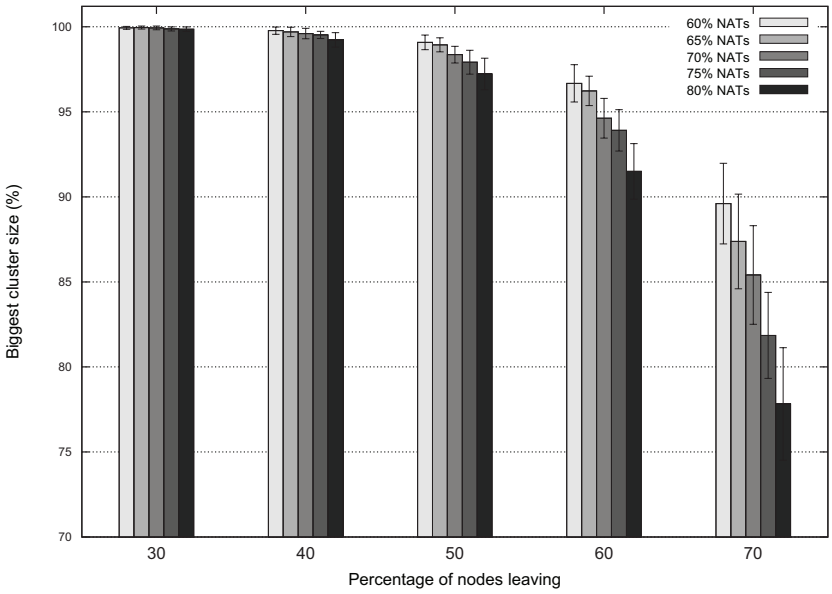
We evaluate the resiliency of the protocols to churn. For that purpose, we study to which extent the graph of nodes remains connected when a large percentage of nodes leave at the same time. We convey the resiliency of the protocol in the form of the size of the biggest connected cluster formed by the remaining nodes.

The experiments were conducted adopting the random bootstrapping scenario, and removing a varying percentage of nodes after each of them had performed 1000 shuffling periods. Public and natted nodes were removed proportionally to their number in the system. We present results in Figure 3.14. The different bar types correspond to different percentages of NATs. On the X axis is represented the percentage of nodes that have left the system. The Y axis represents the size of the biggest cluster 200 shuffling periods after the massive churn took place. We observe that the protocol is highly resilient to churn, tolerating the departure of 40% of the nodes basically without partitioning. Even with higher percentages of nodes leaving the system, the protocol exhibits very good performances. This result can be explained by the fact that each node can be reached by different chains of RVPs at the same time, thus reducing the likelihood of broken paths due to departed nodes. Also, $\mathcal{N}ylon$++ presents in general a slightly better churn resiliency with respect to the base protocol. This is a consequence of the fact that its chains of RVPs are shorter.

3.4.4 Network bandwidth consumption

We evaluated the bandwidth consumed by $\mathcal{N}ylon$ and $\mathcal{N}ylon$++. For comparison, in the plots we indicate also the bandwidth consumed by the reference (push/pull, rand, swapper) implementation of the peer sampling framework. As explained in Section 3.2, one of the objectives of $\mathcal{N}ylon$ is to ensure that all nodes contribute almost equally to

(a) $\mathcal{N}ylon$



(b) $\mathcal{N}ylon$++

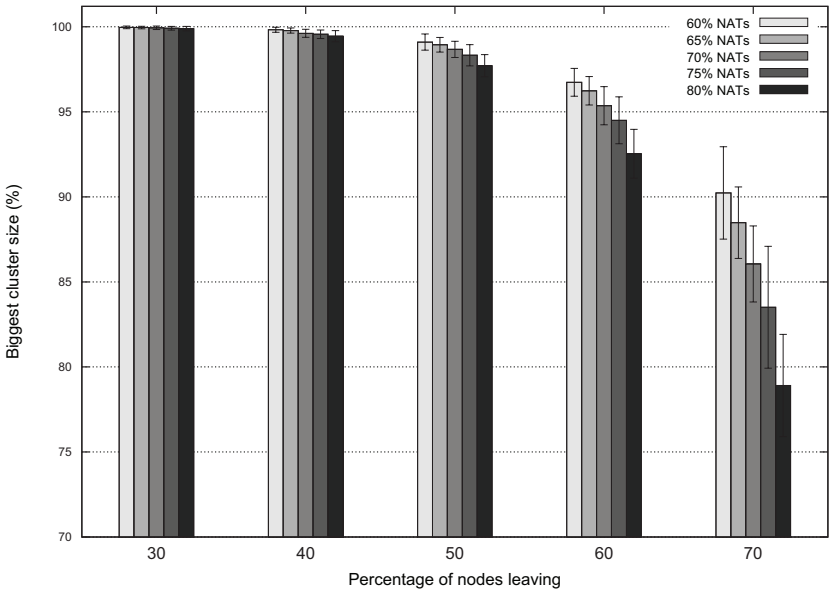


Figure 3.14 – Biggest cluster size after massive churn.

the protocol[8].

Figure 3.15 and Figure 3.16 show respectively the average number of bytes per second that each node sends and receives as a function of the percentage of NATs. Both $\mathcal{N}ylon$ and $\mathcal{N}ylon$++ consume less than 150B/s in download and upload. Albeit $\mathcal{N}ylon$++ has shorter paths towards natted nodes (cfr. Section 3.4.2), its exchanged neighbor entries contain additional fields needed by the optimizations. Hence the higher bandwidth with respect to the base $\mathcal{N}ylon$.

Moreover, we observe that the bandwidth consumption does not evolve linearly with the number of NATs. This comes from the fact that the length of the chain of RVPs does not evolve linearly with the number of NATs (cfr. Section 3.4.2). This is reflected in the figures, which show the average number of bytes per second sent and received by public and natted nodes. We observe also that public nodes send and receive slightly less than natted nodes. This comes from the fact that albeit all nodes can act as RVP, public nodes *(i)* do not receive OPEN_HOLE messages for themselves, and *(ii)* do not send PONG messages.

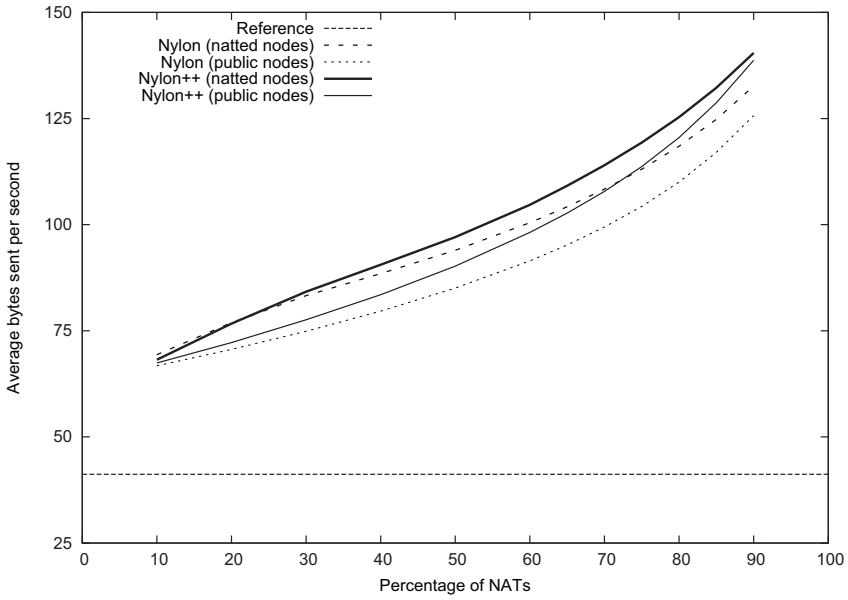


Figure 3.15 – Average number of bytes/s sent by a node.

3.5 Conclusion

A large fraction of computers on the Internet sit behind NATs. This impacts potentially many peer-to-peer protocols relying on the assumption that any node can communicate with any other node provided it knows its IP address and port number. In particular, this impacts gossip protocols that traditionally assume that each node in the

[8]The only exception being that messages sent and received by nodes sitting behind SYM NATs must be relayed by public nodes.

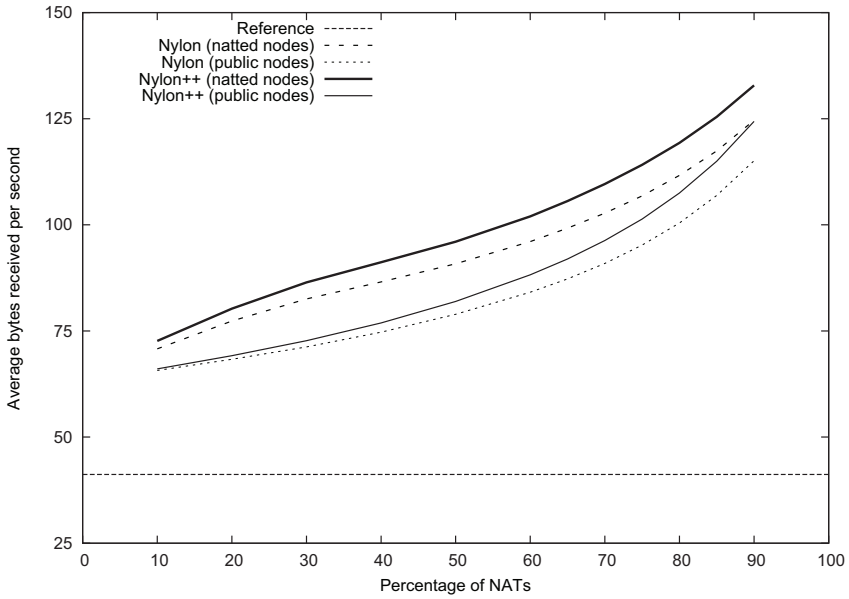


Figure 3.16 – Average number of bytes/s received by a node.

system can communicate with any other node.

We have designed and evaluated $\mathcal{N}ylon$, a fully decentralized NAT-resilient gossip peer sampling protocol. $\mathcal{N}ylon$ leverages the fact that in a gossip protocol each node actually communicates with only a subset of nodes in the system. This enables to use a *reactive* hole punching protocol, which creates a path of relay nodes to setup communications. Experiments have shown that $\mathcal{N}ylon$ accommodates a large proportion of NATs without impacting the properties of the peer sampling. Moreover, $\mathcal{N}ylon$ evenly spreads the overhead induced by NATs between public and natted nodes and is highly resilient to churn.

Part II

Spam-resilient Gossiping in the BAR Model

4

Spam dissemination in presence of Byzantine and rational behavior

Contents

Internet communication and collaboration platforms such as Web groups (e.g., [84]) or Questions and Answers (Q&A) Web sites (e.g., [85, 86]) are commonly used by Internet users every day. An appealing way to realize such services in a distributed manner, and to effectively disseminate the content produced by users, is by relying on gossip-based dissemination protocols (e.g., [25, 30, 33]). In gossip-based dissemination protocols, each node forwards all the messages it receives to a randomly chosen subset of nodes. The advantages of gossip-based dissemination protocols are that they are simple to design, and yet they allow fast and reliable dissemination of messages to large networks of nodes.

Nevertheless, an implicit assumption of gossip-based dissemination protocols is that nodes well-behave in their participation. Nodes are assumed to generate content which is of interest for all other nodes, and they are assumed to follow the specifications

of the protocol by forwarding in turn the messages they receive. But, in the context of P2P Internet collaboration services, there is no central authority controlling what they do. In this "wilder" environment, gossip-based dissemination protocols face scenarios they were not traditionally designed up front to account for. For instance, nodes can act maliciously and leverage the gossip reliable dissemination to actually disseminate spam messages [87]. Or, nodes can act selfishly and deviate from the protocol to maximize their own benefit while reducing their costs in the participation [60], e.g., by not forwarding to other nodes the messages they receive.

This chapter is organized as follows. Section 4.1 provides some background on gossip-based dissemination, illustrating a probabilistic dissemination protocol based on the gossip peer sampling service described in Chapter 2. Section 4.2 studies the impact of spam on gossip-based dissemination protocols. Section 4.3 provides some background on fault models and details a few existing P2P systems designed to take into account malicious behavior, or malicious and selfish behavior altogether. With the characteristics of these existing systems in mind, this section ends identifying a set of requirements for a spam-resilient gossip protocol accounting for malicious and selfish behavior, and a summary of whether the presented existing systems match these requirements. Section 4.4 concludes this chapter.

4.1 Gossip-based dissemination

The gossip paradigm owes its origins to the probabilistic mathematical studies on the diffusion of an epidemic [88]. Inspired by the way an epidemic is spread, gossip protocols disseminate information among nodes similarly to how humans infect each other in case of a disease. This analogy makes gossip protocols usually be referred to also as *epidemic protocols* [29]. In fact, as in the diffusion of an epidemic, once a node receives a message for the first time (when it gets *infected*), it forwards the message to other random nodes (it *infects* other nodes), which in turn do the same, contributing to quickly and effectively spread the message (the *infection*) to a large fraction of the system population.

This simple approach turns out to be very effective. In fact, taking again the epidemic analogy, as pointed out in [89], once an epidemic starts, it is very hard to stop. It is enough for very few people to start infecting other people, to make the disease spread among a large fraction of a group, despite the fact that people may die before infecting other people, or despite the presence of immunized people who do not contribute in spreading the disease. Sharing these similar properties, gossip protocols represent then a simple, yet robust approach to spread information in the system.

In the following, we first illustrate the general functioning, parameters and analytical results underlying the gossip-based dissemination approach. Then, we discuss a simple and effective gossip-based dissemination protocol which leverages the gossip-based peer sampling framework we illustrated in the previous chapters.

4.1.1 Probabilistic dissemination

In [89] the authors discuss the parameters[1] underlying the gossip-based dissemination approach. Each time a node receives a message for the first time, it forwards the message to a random subset of f target nodes of the system, which is usually called the dissemination *fanout*. A node performs periodically this forwarding to f random nodes for a limited amount of r times, which is usually called the number of dissemination *repetitions*. Moreover, each subsequent hop in the dissemination path from the node which originated a given message is usually called dissemination *round*.

Still following the epidemic analogy, the dissemination scheme where $r = 1$ is usually referred to as the "infect-and-die" model, as it mimics the epidemic scenario in which somebody dies just after infecting other individuals. Figure 4.1 illustrates how information is spread among nodes according to this type of dissemination scheme. In the example, once a node receives the message for the first time, it forwards just *once* the message to a fanout f composed of *two* randomly chosen nodes. After four dissemination rounds, all the nodes have received the message. As we can infer from the example, proactively selecting random communication targets introduces redundancy in the message dissemination: nodes possibly receive the same message multiple times.

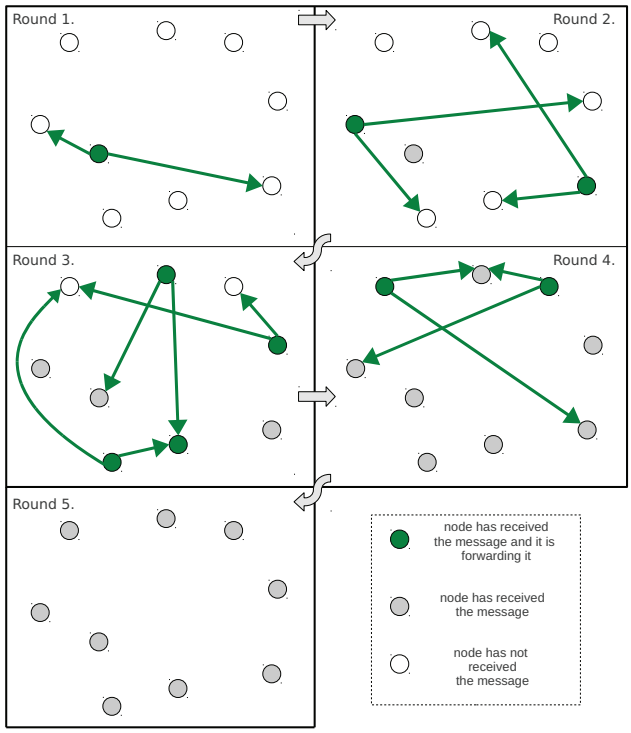


Figure 4.1 – Gossip-based dissemination example: infect-and-die model. Upon receiving a message for the first time, a node forwards it just once (r=1) to two (f=2) random target nodes.

[1]The authors describe also a buffer capacity parameter and possible buffer management strategies. We consider this an orthogonal issue with respect to the description and functioning of the protocols presented in the rest of this document, and we thus omit its discussion in the following.

In the following paragraphs, we first report the analytical results[2] on the reliability properties and latency behavior of gossip-based dissemination protocols. Then, we discuss the crucial point upon which these analytical results are based: the possibility for each node to select f random dissemination targets.

Atomic dissemination. Given the number N of nodes comprising the system population, the choice of the parameter f is crucial for the reliable dissemination of messages. Based on the results by Erdös and Renyi on the branching factor of nodes to ensure connectivity in a graph [70], Kermarrec et al. [30] have analytically and experimentally studied the probability of achieving atomic dissemination (all the nodes receive the message) in the case of gossiping. Their results show that, in the "infect-and-die" model, when the fanout f of random targets is chosen in the order of $ln(N) + c$, the probability of achieving atomic dissemination follows the function:

$$\boxed{P_{atomic} = e^{-e^{-c}}} \tag{4.1}$$

In other words, this result can be read as follows: adequate redundancy of message dissemination (achieved by tuning the fanout f) ensures reliable dissemination with very high probability. Moreover, their study proves that this probabilistic guarantee is ensured when the fanout used by nodes is *on average* $O(ln(N))$, even if individually each node employs a different fanout value.

Latency of dissemination. When the fanout f is chosen in the order of $ln(N) + c$ to provide atomic dissemination, an important analytical result has been provided concerning the latency of dissemination. In fact, Bollobás [28] has shown that, in the "infect-and-die" model, the number R of rounds it takes to achieve atomic dissemination follows the function:

$$\boxed{R = \frac{ln(N)}{ln(ln(N))} + O(1)} \tag{4.2}$$

This result, together with the first one in Formula 4.1, prove the scalability of gossip protocols with respect to the system size. Fanout and latency in fact increase only logarithmically with the system size.

Ensuring random choices of the targets. From what discussed so far, gossip-based protocols stand as a simple, yet scalable and (probabilistically) highly reliable solution for information dissemination. Nevertheless, an assumption upon which the analytical results are based, is that a node is actually able to select uniformly at random f dissemination targets from all nodes of the system. We have discussed in the previous chapters of this document how it is possible to sample nodes uniformly at random in a scalable way by relying on gossip-based peer sampling protocols (e.g. [26, 27, 55]). These protocols are in fact capable of maintaining partial views which contain a

[2]The reader can find in the work by Eugster et al. [89] description and references of several mathematical results in the theory of epidemics which inspired gossip protocols.

(continuously changing) random sample of the whole system, and that these views are able to quickly adapt to node arrivals and departures.

We illustrate in the next section an example of a gossip-based dissemination protocol which builds on top of the gossip-based peer sampling framework [27] for the selection of random dissemination targets.

4.1.2 The RandCast protocol

The gossip-based dissemination protocol presented in [30] relies on each node having only a partial view of the system. This partial view is assumed to be assigned by a membership protocol which is able to ensure that each node's view contains a uniform random sample of the nodes. In [90], the authors propose to instantiate the gossip-based dissemination protocol presented in [30] (which they refer to as *RandCast*[3]), on top of the gossip-based peer sampling framework [27] discussed in previous chapters of this document. The gossip-based peer sampling ensures in fact that nodes have a continuously changing uniform random sample of the nodes in their local (partial) views. From the analytical results reported in the previous section, it follows that the peer sampling protocol can then be configured such that the size of the partial view maintained at each node is at least equal to the fanout value required by the dissemination protocol to achieve reliable dissemination with high probability.

The functioning of the *RandCast* dissemination protocol can then be summarized by the pseudo-code of Figure 4.2. When a node originates a message, or when a node receives a message for the first time, it disseminates the message to f random nodes. The choice of these targets is encapsulated in the `select_targets()` function. This function, which can be provided by a modular component of the dissemination protocol, selects the f random targets relying either on a complete or on a partial view of the system, depending on the underlying view maintenance protocol.

4.1.3 Summary

Gossip-based dissemination protocols are a simple, yet reliable and scalable, approach for disseminating information in large-scale peer to peer systems. Nevertheless, the very same underlying principle —random choice of gossiping targets— which ensures the reliable dissemination among all nodes with very high probability, has also its drawbacks. First, the random choice of targets introduces a lot of redundancy in the message dissemination, where nodes can receive the same message multiple times [30]. Second, it is not trivial to stop the dissemination of a message: all copies of a message should in fact be deleted [29].

We discuss in the next section how gossip-based dissemination protocols are affected by the injection in the network of messages with junk content, i.e. spam messages.

[3]For convenience, in the rest of this document we keep using *RandCast* to refer to the gossip-based probabilistic dissemination protocol described in [30].

```
1  on originate_message(m)
2    disseminate(m)

3  on receive ⟨Broadcast,m⟩ from p do
4    // If first time this message was received
5    if m not in buffer then
6      disseminate(m)

7  def disseminate(m)
8    add m to buffer
9    targets ← select_targets()
10   foreach target in targets:
11     send ⟨Broadcast, m⟩ to target

12 // Modular targets selection function
13 def select_targets()
14   return f random neighbors
```

Figure 4.2 – RandCast protocol: pseudo-code

4.2 Impact of spam in gossip-based dissemination

Gossip-based dissemination protocols can be an effective solution to disseminate information in a collaborative service. But, in a collaborative service where there is no central authority controlling the actions of nodes, these latter might act maliciously and deviate from the guidelines of the service they participate in. As studied in [87], nodes might for instance exploit the high reliability of gossiping to inject spam messages (i.e., messages with unsolicited or junk content) in the network and just expect them to be spread to a very large fraction of nodes.

To the best of our knowledge, the work in [87] represents the most relevant study on the impact of spam in gossip-based dissemination protocols. In the following section, we first illustrate this existing solution to limit spam dissemination, and then we discuss the drawbacks of the approach.

4.2.1 Canning Spam in Wireless Gossip Networks

In [87], the authors study the impact of spam in wireless ad-hoc networks, when gossiping is used for information dissemination. In a wireless ad-hoc network, there is generally no preexisting routing infrastructure. Each node exchanges information with nodes in its network proximity, resulting in nodes collaborating to route data to a given destination. The given study, conducted in the context of wireless ad-hoc networks, is generalizable to the case of gossiping in peer-to-peer networks as well.

In their work, the authors first study the impact of spam on the wireless network node population. To do so, they simulate a system in which nodes exchange information by leveraging a push/pull gossip-based dissemination protocol. Each node periodically selects one node among their current set of wireless neighbors as the gossip target. The node sends to the target node a subset of the messages it has currently stored in its local

buffer, and receives in turn a subset of the messages in possession by the target node.

The study shows that due to the randomness and redundancy of gossiping, it is sufficient to have just a few nodes inject spam, to have it progressively spread among the whole node population. The authors propose to limit spam dissemination based on the following assumption on the malicious nodes behavior. As message dissemination relies on nodes forwarding the message among each other, a malicious node could modify the content of a message and forward a spam-modified version in place of the original published content. They suggest then to tackle the spam problem employing digital signatures and integrity checks along the path. According to this, the source node can digitally sign the messages it publishes[4]. Then, potentially each node along the dissemination path can check the digital signature of each message it receives.

To reduce the computational cost on the wireless network nodes, which typically have constrained resources, the authors propose to employ a probabilistic check of the received messages. Figure 4.3 illustrates a possible pseudo-code of the proposed solution. Nodes only check a received message with a fixed probability P_{check} (line 16), and if the message is not valid, it is discarded (lines 17-18). The approach turns out to limit the dissemination of tampered messages: the more the distance from a node which has tampered a message, the more the probability for this message to be discarded along the path. Furthermore, the overall amount of tampered messages in the network decreases when increasing the probability of checking P_{check}.

In a subsequent work [91], the authors have enhanced the above solution by making each node dynamically adjusts its P_{check} probability based on the amount of spam it perceives from its neighbors. Moreover, this dynamically adjusted probability is maintained on a per-neighbor basis, as different neighbors could have different (spam) behavior. Thanks to this approach, the nodes which are closer to the nodes injecting spam perform the resource consuming task of checking the integrity of messages and filtering them, whereas more distant nodes can simply leverage on the action taken by the former nodes. To reduce the workload induced on the nodes which are close to the nodes injecting spam, nodes keep track of the spam messages they receive on a per-neighbor basis. This allows a node to detect if a neighbor is consistently presenting a malicious behavior, and if so, the node can refuse to communicate with it.

Drawbacks. We identify the following drawbacks in the proposed solution. First, on the Internet, the problem of spam cannot considered merely an integrity check problem, which can be solved by signing and (probabilistically) verifying messages. In fact, nodes can inject false information on an Internet forum which can be detected as junk only by human users. For example, in a discussion group dedicated to football, a user could post a message claiming to know that a given player has signed for a team, while this information is not true, thus polluting the content of the collaborative service. Or even worse, on a technical Questions&Answers web site, a user could ask a question on how to fix a configuration problem of its OS, and a malicious user could reply by giving a false answer which could damage the computer of the user who posted the question.

Second, not all users are capable, or willing, to effectively detect and filter junk

[4]The solution assumes the use of a *Certification Authority* (CA), which certifies the public keys used by nodes.

```
1  every shuffling_period units do
2    target ← selectPeer()
3    msgs ← selectMsgsToSend()
4    send ⟨Gossip, msgs⟩ to target

5  on receive ⟨Gossip, msgs⟩ from source do
6    msgs_resp ← selectItemsToSend()
7    send ⟨GossipResponse, msgs_resp⟩ to source
8    verify(msgs)
9    buffer ← selectMsgsToKeep(buffer, msgs)

10 on receive ⟨GossipResponse, msgs_resp⟩ from source do
11   verify(msgs_resp)
12   buffer ← selectMsgsToKeep(buffer, msgs_resp)

13 def verify(msgs)
14   foreach m in msgs:
15     // Verify each message with probability P_check
16     if rand(0,1) < P_check
17       if m is not valid:
18         msgs ← msgs \ {msg}
19   return msgs

20 def selectPeer()
21   return a node among the wireless network neighbors
```

Figure 4.3 – Pseudo-code for the push/pull gossip-based dissemination protocol used in "Canning Spam in Wireless Ad-Hoc Networks".

content. That is, users have a different "pollution awareness" [58]. Moreover, to actually be encouraged in participating in the filtering of spam in a collaborating service, nodes should be rewarded for filtering spam. That is, users contributing more in filtering should receive less spam in return.

Third, in a P2P context, nodes gossip with targets choosing uniformly at random among the *whole* node population, thus limiting the effectiveness of such an approach based on fixed neighbor relationships, and relying only on direct first-hand experience with malicious nodes.

Taking into account the points discussed so far, we study in the next section the impact of spam on the *RandCast* protocol which we have described in Section 4.1.2.

4.2.2 Impact of spam on RandCast

From the points discussed in the previous section, we have studied the impact of spam dissemination in a P2P context where nodes actually gossip with random nodes taken from the whole node population. We have used the *RandCast* gossip-based dissemination protocol [30], which we have modified so that each node i has a (fixed) probability P_i to detect (and stop forwarding) each spam message it receives for the first time. In Figure 4.4, we present the results of a simulation study on a system composed of $1,000$ nodes. Moreover, we have evaluated the dissemination of spam using three

different distributions of the nodes' probability to detect and filter spam:

- uniform distribution: nodes have a probability to filter spam which is uniformly spread in the range $[0, 1]$;
- power-law distribution: most of the nodes have a probability to filter spam which is rather low;
- inverse of the power-law distribution: most of the nodes have a probability to filter spam which is rather high.

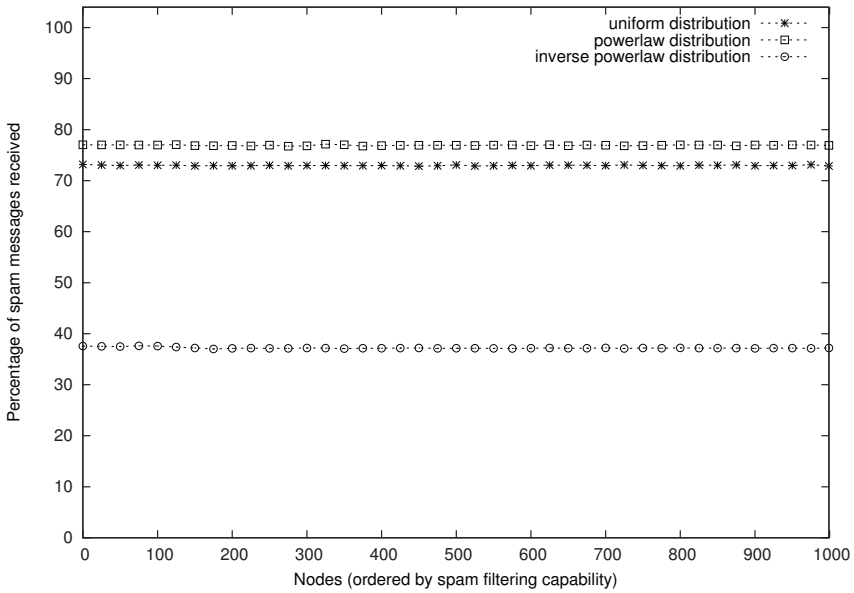


Figure 4.4 – RandCast: percentage of spam messages received by nodes for three different spam filtering capability distributions.

We can observe two things from the results. First, regardless of the actual different capability of nodes to filter spam, the percentage of spam received by each node is roughly the same. This is due to the randomness and redundancy of gossiping. Second, the amount of spam received by nodes is quite high. In fact, as pointed out in [89], once an epidemic starts, it is hard to stop.

4.2.3 Summary

We have seen how gossip-based dissemination protocols are ideal vectors to disseminate spam messages. Indeed, simple strategies consisting in having each node locally filter the messages it detects as spam do not work. The reason is that gossip-based protocols are highly redundant and random: each node receives messages multiple times from different nodes. Consequently, it is enough that a small subset of nodes do not filter a spam message to have it received by a large fraction of nodes.

In order to design a gossip protocol able to limit the dissemination of spam, special care should be taken when considering the possible behavior of the participant nodes in

a context like the Internet. In the next section, we describe which behavior nodes can have in practice in the Internet, and we detail the mechanisms that existing practical P2P solutions adopt to tolerate/discourage this behavior.

4.3 Byzantine and rational behavior

P2P Internet collaborative services are based on the participating nodes contributing their resources to the overall functioning of, and getting the benefits from, the collaborative services themselves they participate in. The design of protocols implementing such distributed systems should account then for the possible behaviors of the participating nodes. In the simplest of the scenarios in fact, nodes might silently disconnect without notice, or just fail because of a crash of the machine. This behavior is called the *crash model*, and protocols designed accounting for this behavior are said to be *crash fault tolerant* protocols. For what we have illustrated in the previous sections, gossip-based dissemination protocols are by nature crash-tolerant. In fact, the randomness and redundancy of the probabilistic dissemination approach ensures reliability with high probability even in case of node disconnections [30].

But, as pointed out in [60], in a practical context like the Internet, where there is no central authority controlling what nodes do, nodes may also deviate from the protocol for the following two other reasons. First, nodes can deviate because of a bug, a misconfiguration, or even because an intruder takes control of their machine and uses it to perform an attack to a given node or to the system as a whole. This more generic fault behavior is called the *Byzantine model* [92], and protocols designed to tolerate this behavior are said to be *Byzantine fault tolerant* protocols. Second, nodes may also selfishly deviate from the protocol to maximize their benefit in the participation while reducing their costs, e.g., by getting a "free-ride" and not forwarding messages [93, 94]. This behavior is in game theory referred to as *rational behavior* [95]. The BAR model [60] is a fault model accounting for Byzantine and rational behavior altogether, and protocols designed to tolerate these behaviors are said to be *BAR fault tolerant* protocols.

Figure 4.5 illustrates the fault model resiliency hierarchy that we refer to in the following. A protocol which is crash-tolerant is not necessarily also Byzantine-tolerant. In turn, a Byzantine-tolerant protocol is not necessarily also BAR-tolerant.

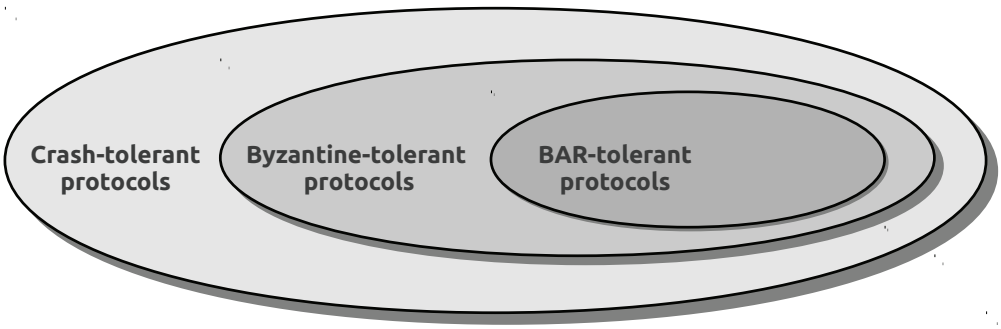


Figure 4.5 – Fault tolerance hierarchy.

In the following sections, we first detail the functioning of two existing P2P systems designed under the Byzantine model. Then, we discuss the BAR model [60], which accounts for Byzantine and rational behavior altogether, and we detail the functioning of two existing P2P systems designed under this model. With the characteristics of these four existing systems in mind, the section ends identifying a set of requirements for a spam-resilient gossip protocol in the BAR model, and it summarizes whether the presented four systems match these requirements.

4.3.1 Byzantine-tolerant systems

In the Byzantine Fault Tolerance (BFT) model [92], it is usually assumed an upper bound of nodes which might deviate from the protocol. Such nodes, called Byzantine nodes, can deviate for arbitrary of reasons: they might for instance be bugged, misconfigured, or even compromised. Byzantine Fault Tolerant protocols are then those protocols designed to tolerate the Byzantine behavior, that is, designed to ensure the desired system properties even in presence of the maximum allowed number of Byzantine nodes deviating from the protocol.

In the following two sections, we provide a detailed description of the functioning of two existing P2P systems considering the Byzantine fault tolerance model: SecureStream [96], which implements a live streaming protocol on top of the Fireflies [97, 98] membership protocol, and Nysiad [99], a system which translates a crash-tolerant protocol into a Byzantine-tolerant one.

4.3.1.1 Fireflies/SecureStream

SecureStream [96] is a live streaming protocol built on top of Fireflies [97, 98], a Byzantine-tolerant membership protocol. Fireflies ensures that, despite the presence of a possible upper bound of nodes exhibiting Byzantine behavior, each node has a *complete view* of the live nodes of the system. Such membership is leveraged by the SecureStream streaming protocol for the choice of dissemination partners.

Despite the fact that in Fireflies nodes maintain a complete view of the system, the protocol actually requires nodes to communicate with (and about) just a small subset of the nodes. These characteristics make the protocol a suitable solution for applications where the system size is in the order of hundreds or thousands of nodes.

Fireflies architecture. Fireflies relies on a probabilistic approach to provide each node with an up-to-date global view, and it is composed of the following three protocols:

- an adaptive **pinging protocol** to detect if nodes are live or not;
- a **membership protocol** which relies on *accusations* and *rebuttals* to ensure nodes maintain an up-to-date global view of the system;
- a **gossiping protocol** to reliably disseminate the accusation and rebuttal membership messages among non-Byzantine nodes.

Nodes are logically organized into $2t + 1$ rings. The organization of nodes on the rings has two properties. First, nodes have a different position on each ring. The

position of a node on a ring is in fact given by hashing with a collision resistant hash function H the identifier of the node concatenated with that of the given ring:

$$\boxed{pos[ring\_id] = H(node\_id||ring\_id)} \tag{4.3}$$

Second, the use of such a hash function deterministically specifies the global ordering of nodes on each ring, and hence the relative positions among nodes. In Fireflies, such ordering specifies which nodes each node monitors and from which nodes it is itself monitored, which plays a role in the functioning of the membership protocol described next. Taking Figure 4.6 as an example, it illustrates a possible ordering of 7 nodes on 3 rings for a given hash function. In this example, node A is a monitor for nodes B, D and F (respectively, on the outer, middle and inner most ring), while at the same time being monitored by the nodes F, E and G (respectively, on the outer, middle and inner most ring).

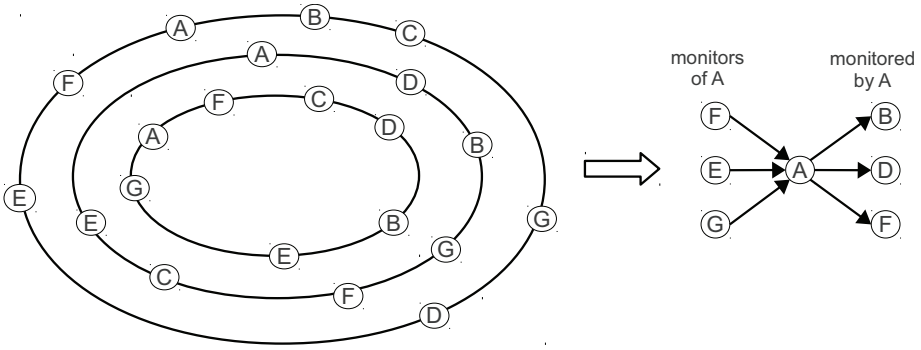


Figure 4.6 – Fireflies: example of a possible ordering of 7 nodes on 3 rings. Node A is monitored by nodes F, E, G and it monitors nodes B, D, F. (*Based on:* [96])

Fireflies group membership. On each of the $2t + 1$ rings, each node n_i monitors the first successor node n_j which, according to its view of the system, it knows to be live. To do so, n_i performs an adaptive probing by means of a pinging protocol[5]. If the monitored node n_j does not reply to the ping, the node monitor n_i sends an accusation (failure notice) for node n_j. The accusation is progressively diffused to all nodes by means of a gossip-based dissemination protocol (detailed later), whose latency is probabilistically upper bounded by a 2Δ time period. Thanks to this Δ, when an accused node n_j receives an accusation about itself, it can send —still via the aforementioned gossip-based dissemination protocol— a rebuttal message meant to state its liveness to all nodes, thus invalidating the accusation. A node n_k which has received an accusation about a node n_j in fact will remove n_j from its view only after a 2Δ time period from the first reception of the accusation about n_j.

Fireflies utilizes the following mechanisms to prevent Byzantine monitors from causing non-Byzantine nodes to be removed form the views of non-Byzantine nodes. First, nodes digitally sign the accusations and rebuttals they send. Second, an accusation

[5]The reader can refer to the Fireflies paper [97] for details of the adaptive ping protocol.

for a node n_j can only be sent relative to the most recent rebuttal that the node n_j has sent in the overlay. Third, a rebuttal contains an `enabled` bitmap which specifies which $t+1$ rings among the $2t+1$ are allowed to actually send an accusation for that rebuttal, being otherwise the accusation not considered valid by any non-Byzantine node. The number of Byzantine monitors that each node (and hence the protocol) can tolerate depends then on the value of t.

To choose a proper value for t, Fireflies applies the following reasoning. If by system configuration any live node can be Byzantine with probability P_{byz}, it is possible to calculate the number of $2t+1$ rings to compose the Fireflies overlay such that the probability for a node to have more than t live Byzantine monitors is very small. Fireflies chooses this minimum value of t by solving the cumulative binomial distribution reported in the Formula 4.4:

$$\boxed{min(t) : P = B(t, 2t+1, 1-P_{byz}) < \epsilon} \tag{4.4}$$

Fireflies membership protocol leverages the fact that, if the probability of having more than t Byzantine monitors is very low, then: *(i)* a non-Byzantine node, by disabling t rings in the `enabled` bitmap, can effectively prevent all the (possible) Byzantine monitors from maliciously accusing itself, and *(ii)* a Byzantine node, by disabling t rings in the `enabled` bitmap, will still nevertheless have (at least) a non-Byzantine monitor able to send an accusation for it.

The accusation and rebuttal messages are diffused to all nodes using gossiping. A node originating a message, sends it to f random nodes selected from the view, where f is the dissemination fanout. Nodes that receive a message will in turn forward it to f randomly chosen nodes. As previously discussed in Section 4.1.1, if the fanout used by each node is (on average) $ln(N)+O(1)$, then all nodes receive the message with very high probability [30], and the upper bound Δ to have all the nodes receive the message is in the order of $\frac{ln(N)}{ln(ln(N))}+O(1)$ gossip dissemination rounds [28].

SecureStream. SecureStream [96] is a Byzantine-tolerant streaming protocol built on top of Fireflies. It relies on Fireflies to maintain membership even in presence of a bounded number of Byzantine nodes, and it leverages Fireflies multiple rings topology to build a Byzantine-tolerant pull-based live-streaming protocol. Nodes are in fact allowed to request for the most up to date packets to their predecessors on the various rings. The approach ensures the following two properties. First, nodes cannot be overwhelmed with requests, because the number of nodes which are actually allowed to pull packets from them is limited to their successors on the rings. Second, as the ordering on each ring is dictated by a pseudo-random hash function specific for each ring, the graph formed by connecting the various nodes according to the predecessor/successor relationship on the various rings turns out to resemble a random graph. For the probabilistic properties of the choice of the fanout in gossip-based dissemination [30] which we have illustrated in the previous sections, it follows that by adequately tuning the number of rings (and hence, the number of predecessors from which it is possible to receive packets), the dissemination service is ultimately resilient

to omission attacks by the Byzantine nodes.

The protocol also provides an auditing mechanism to ensure nodes participate in uploading at least a certain percentage of the received stream. This mechanism involves a two-level auditing architecture. First, the nodes themselves act as "local auditors" by exchanging among each other "receipts" of messages received from their neighbors. This allows to determine if a node has not uploaded the minimum percentage of stream or it is reporting false information about data received from another node. Second, "external auditors" are in charge of collecting complaints from local auditors, and to decide to punish misbehaving nodes accordingly.

4.3.1.2 Nysiad

Nysiad [99] is a system which is able to translate a P2P protocol which is only crash-tolerant into one which is Byzantine-tolerant, by adopting an approach based on State Machine Replication (SMR) [100]. In State Machine Replication a node is replicated on multiple nodes, called the *replicas*. The replicas run a replication protocol which ensures that they remain synchronized on the state of the replicated node. This is achieved by executing in the same order the requests to the replicated node, even in case of possible networking issues or failures.

Nysiad makes use of State Machine Replication as follows. Each participant node is modeled by a replicated state machine, and if the replicated node does not faithfully follow the protocol, its replicated state machine is halted. In such latter case the node would be considered as crashed by the other nodes, which is a behavior that is ultimately tolerated if the original protocol being translated is crash tolerant.

Nysiad architecture. The state machine replicas of each node are assigned to (at least) $3t + 1$ nodes, including the node itself. These nodes are called the *guards* of the replicated node. Furthermore, each two nodes have (at least) $2t + 1$ common guards, which are called the *monitors* of the two nodes. Nysiad makes the assumption that an upper bound t of guards of a node can be Byzantine, and that message communication between non-Byzantine guards is reliable. Also, the guards assignment is made by a (logically) centralized 3rd party trusted entity called the "Olympus".

The overall system is composed of two main protocols: one run by the various replicated nodes, while the other one is run by the Olympus. The participant nodes run a replication protocol which is composed of a base *reliable ordered broadcast* protocol, plus *attestation* and *credit* sub-protocols. These three protocols ensure the following:

1. **Reliable ordered broadcast protocol**: the guards (the replicas) are synchronized on the state of the node they replicate.
2. **Attestation protocol**: only messages corresponding to a faithful protocol execution are delivered to the guards (the replicas) of a node.
3. **Credit protocol**: a node is considered as crashed by other nodes if it does not fairly process all its input.

The Olympus does not take part to the replication protocol. It runs a replica maintenance protocol, to assign and maintain the replicas for the various nodes. In

the following, we first illustrate the various parts of the replication protocol. Then, we discuss the role of the Olympus.

Nodes state machine replication protocol. We illustrate in the following the functioning of the three sub-protocols forming the state machine replication protocol.

(i) Reliable ordered broadcast protocol. This protocol ensures the guards (the replicas) are synchronized on the state of the node they replicate. To achieve this, the replicated node uses a reliable ordered broadcast protocol for communication with its guards. This protocol, depicted in Figure 4.7, works as follows. When a node n_i wants to send an input message m to its guards, it first sends an *order request* message to its guards containing the hash of m. Guards reply with an *order certificate* message, which includes a sequence number c that they maintain on behalf of the replicated node n_i. The node n_i is able to collect (at least) $G_i - t$ order certificates, G_i being the number of guards of n_i. In fact, for what specified previously, t is the assumed upper bound of Byzantine guards of a node. A consistent sequence number in $G_i - t$ order certificates constitutes an *order proof* for the message m to be delivered. At this point, n_i delivers the messages m to its own running replica of its state machine, and sends the message m with the order proof to all its guards. If a guard assesses the order proof is valid, it delivers the message m to its running replica of n_i's state machine, and gossips it with the other guards of n_i to ensure that if a non-Byzantine guard delivers a message, also all other non-Byzantine guards will be able to do so.

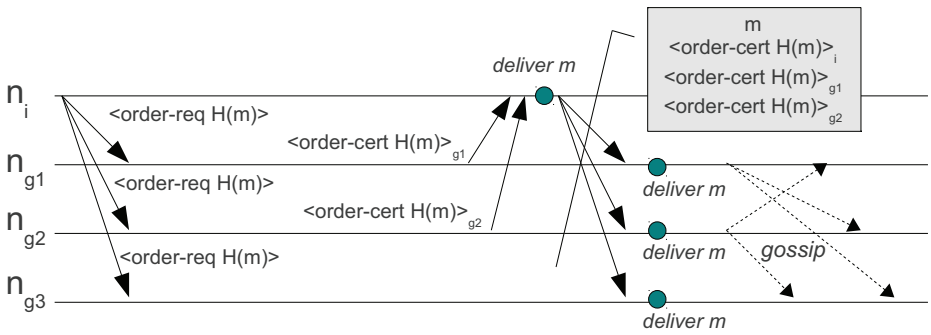


Figure 4.7 – Nysiad: reliable ordered broadcast protocol. Node n_i initiates a reliable ordered broadcast. The guard node n_{g3} is Byzantine and does not participate in the protocol. (*Based on:* [99])

The reliable ordered broadcast protocol ensures synchronization among guards (the replicas) of a node, but it does not preclude to a node the possibility of forging or ignoring inputs. The attestation and credit protocols described next provide the complementary pieces to take these two misbehaviors into account.

(ii) Attestation protocol. This protocol ensures that only messages corresponding to a faithful protocol execution are delivered to the guards (the replicas) of a node, thus precluding to a node the possibility to forge invalid inputs. If for example in the original protocol the node n_i was sending a message m to node n_j, this in the translated protocol would correspond to having the state machine of n_i sending m to the state machine of n_j. To preclude the possibility to n_j of forging arbitrary inputs, the guards of n_j need

also a "proof of validity" with every message that is sent by n_j via the reliable ordered broadcast described before. This proof of validity that n_j has to provide is a set of $t+1$ *attestations* from nodes which are guards of both n_i and n_j (their monitors). In fact, each (non-Byzantine) guard of n_i runs n_i's state machine replication, and has thus the same state for n_i which led to sending the message m. Each (non-Byzantine) guard hence sends an attestation message to n_j. These attestations are required only in the last part of the reliable ordered broadcast, so they can be requested in parallel with the request of order certificates.

(iii) Credit protocol. This protocol ensures a node is considered as crashed by other nodes if it does not fairly process all its input. Crashing is a behavior that the original crash tolerant protocol being translated can handle. The credit protocol relies on the following principle: a node must get $t+1$ *credits* from its guards to able to send new inputs to its guards by means of the reliable ordered broadcast. Non-Byzantine guards provide these credits if they have seen the replicated node faithfully processing the previous inputs. Furthermore, an attempted reliable ordered broadcast message sent by a node without the valid credits constitutes a proof of misbehavior. Such messages are reported to the Olympus, and their consequences are described in the next paragraph.

Olympus: replica maintenance protocol. A logically centralized trusted entity[6], called the "Olympus", is in charge of assigning and maintaining guards (the state machine replicas) for nodes. The Olympus is unaware of which protocol is being replicated at each node, and —apart from collecting proofs of misbehavior— it does not take part to any of the three replication sub-protocols described previously. Its functioning is based essentially on monitoring two things. First, the Olympus checks if nodes are live by means of running a simple ping/pong protocol. Second, as part of the credit protocol, it collects the proofs of misbehavior from the guards of a node, which assess with evidence that a node has not followed the protocol. Then, according to the fact that a node is no longer live, or that a node has to be evicted from the system because it has provably misbehaved, the Olympus updates the guards. To do so, it signs and sends to nodes *epoch certificates*, which attest a node's identifier and its current guards.

4.3.2 BAR model: accounting for Byzantine and rational behavior

The BAR model has been introduced in [60] pointing out how, in Internet collaborative services, nodes might actually deviate from a given protocol for two different reasons. In fact, not only a bounded number of nodes could deviate because bugged, misconfigured or compromised (as in the Byzantine fault tolerance model [92]), but also possibly all nodes could deviate motivated by a *rational behavior* [95], that is, with the intent of maximizing their own benefit, for instance attempting to reduce their costs in the participation by "free-riding" [93, 94].

In the rest of this section, we first provide the concepts (categories of nodes, classes of protocols in the BAR model, game theory background to understand rational behavior) underlying the BAR model based on the original formalization found in the

[6] In the original description [99], the authors suggest a Byzantine-tolerant implementation of the "Olympus", e.g., replicated with a protocol like PBFT [101].

BAR seminal paper [60] and present in Martin's dissertation [102]. Then, we provide the superset of assumptions on rational behavior found in the first two P2P protocols designed in the BAR model [42, 60].

Categories of nodes in the BAR model. The name BAR stands for "Byzantine-Altruistic-Rational" model, as it classifies nodes into the following homonymous three categories [60, 102]:

- **Byzantine nodes**. They are nodes which can exhibit arbitrary deviations from the protocol specifications: they could be bugged, misconfigured, compromised by an attacker.
- **Altruistic nodes**. They are nodes which *always* follow each one of the steps mandated by the protocol specifications.
- **Rational nodes**. They are selfish nodes which are willing to deviate from the protocol specifications if by doing so they can increase their utility, i.e., if they can maximize their benefits while reducing their costs.

Ensuring system properties in the BAR model. To ensure the properties of a system to all non-Byzantine nodes, the following two classes of protocols in the BAR model have been defined [60, 102]:

- **Incentive-Compatible Byzantine Fault Tolerant (IC-BFT)** protocols: they are protocols which tolerates a bounded number of Byzantine nodes deviating from the protocol, and all rational nodes expect to obtain the highest utility by faithfully following the protocol.
- **Byzantine Altruistic Rational Tolerant (BAR-Tolerant or BART)** protocols: they are protocols which tolerate a bounded number of Byzantine nodes and possibly all rational nodes deviating from the protocol.

As it can be inferred from the description, and as summarized graphically by Figure 4.8, IC-BFT protocols are a subset of BART protocols.

Modeling rational nodes: game theory. As explained in [102], in the BAR model the behavior of rational nodes is modeled in a game theoretic framework [95]. Rational nodes are modeled as players of an infinitely repeating game. In such a game, a rational player seeks to modify its game playing strategy in order to maximize its benefit, but only if by doing so it does not risk compromising its benefit in the next play of the game. A game is then said to be a Nash equilibrium [61] if no player can increase its utility by changing playing strategy while the other players continue following their strategies.

This drives the design of Incentive-Compatible Byzantine-Fault-Tolerant (IC-BFT) protocols in the BAR model along two points [60, 102]: *(i)* an IC-BFT protocol must account for, and tolerate, a possible upper bound of Byzantine nodes deviating from the protocol for arbitrary reasons (Byzantine Fault Tolerance); *(ii)* an IC-BFT protocol must provide incentives to discourage rational nodes from deviating from the protocol, i.e. by designing the protocol to be a Nash equilibrium (Incentive Compatibility).

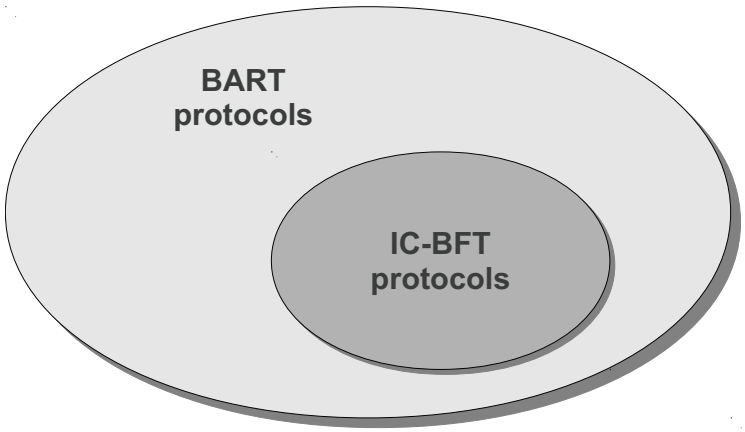


Figure 4.8 – BAR model: classes of protocols. All Incentive-Compatible Byzantine Fault Tolerant (IC-BFT) protocols are also BAR-Tolerant (BART) protocols.

Assumptions on rational behavior. According to the aforementioned points, when designing a protocol in the BAR model, the behavior of rational nodes in a collaborative service is usually characterized by the following set of assumptions [42, 60]:

- rational nodes seek long term benefit in their participation.
- rational nodes are conservative in their actions: they expect the upper bound of Byzantine nodes to behave in the way which could harm the most their utility (assuming that all other non-Byzantine nodes follow the protocol).
- rational nodes have no benefit in delaying sending a message if they are nevertheless going to send it.
- rational nodes have no interest in receiving (or even more, sending) messages which are not part of the protocol specifications.
- rational nodes consider the benefits of the service much more important than the costs of participation.
- rational nodes have no interest in deviating from the protocol if by following it they receive the same utility.
- rational nodes will always follow the protocol if this latter provides a Nash equilibrium.
- rational nodes do not collude[7], only Byzantine nodes can collude.
- rational nodes suppose the other (non-Byzantine) nodes are altruistic and follow the protocol[8].

In the following two sections, we provide a detailed description of the functioning of the first two P2P systems which have been proposed in the BAR model: BAR-B [60], a collaborative backup service implemented on top of a generic framework, and BAR Gossip [42], a gossip-based live streaming application.

[7] This is a Nash equilibrium limitation.
[8] This is a Nash equilibrium artifact proof.

4.3.2.1 BAR-B

BAR-B [60] is the first P2P system in the BAR model. It proposes a generic 3-level architecture to build collaborative services in the BAR model, and it presents a collaborative backup service as example of instantiation of this architecture. The 3-level architecture, from the lowest to the highest level, is made of:

- **Level 1: Primitives**. This level ensures reliable communication among nodes by means of an IC-BFT replicated state machine built on top of them.
- **Level 2: Work Assignment**. This level allows to reliably request work to do to nodes.
- **Level 3: Application (Backup)**. This level implements a given collaborative service, e.g., a collaborative backup.

This architecture is aimed at providing a practical BAR framework to realize collaborative services made from tens to a few hundreds of nodes. Moreover, it assumes an upper bound of $\frac{n-2}{3}$ Byzantine nodes in the system. The layered-architecture allows simplified design for the upper layer(s), by leveraging low-level mechanisms provided by the layer(s) below. In the following paragraphs, we thus detail more on the functioning of the two lowest, and most important, levels of the architecture.

Level 1: Primitives. This level provides an IC-BFT Replicated State Machine which allows nodes to issue commands via an IC-BFT Terminating Reliable Broadcast (TRB) protocol. The TRB protocol rotates the node which can send a command (the *leader*) after each executed instance of the protocol. This ensures that each node has its possibility to issue a command to the replicated state machine by attempting the three-phase commit (agree/write/show-quorum) depicted in Figure 4.9.

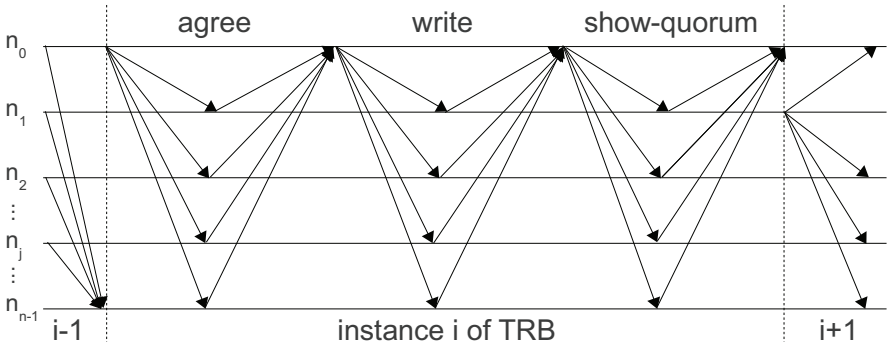


Figure 4.9 – BAR-B: Terminating Reliable Broadcast. (*Based on:* [60])

A set of mechanisms are then put into place at this level of the architecture to discourage rational nodes from: *(i)* omitting sending the messages they are actually supposed to send; *(ii)* sending the less costly message, among various possible valid

responses, instead of the "faithful" (and possibly bigger in size) message; and *(iii)* delaying sending the messages. These mechanisms are described in the following.

Discouraging message omission. Each node locally maintains message queues for the communication to and from any other node in the system. These queues ensure *predictable communication patterns* by working accordingly to the "If you don't talk to me, then I won't talk to you" principle [103]: a node will refuse any incoming or outgoing communication with a given node if this latter has not complied to sending a message the former node is expecting. The use of these message queues, with their unilateral blacklisting, encourages every rational node r to follow the protocol. In fact, when it is r's TRB turn, if the upper bound of f Byzantine nodes have blacklisted r, and even only one non-Byzantine node has blacklisted r because of misbehavior, then r will not be able to reach a TRB protocol quorum of $n - f - 1$ nodes to actually issue a command in the state machine. For what illustrated in the previous section on the assumptions on rational behavior, a rational node r seeks long term benefit in the system and does not want to risk it by attempting to reduce the costs. This means the rational node r is encouraged to send the messages expected by other nodes, in order to be able to issue commands via the state machine when it is its turn.

Discouraging sending cheap valid messages. The design of messages follows a *balanced cost* principle: the valid messages for a given protocol step are all of the same size. In this way, sending the message which faithfully corresponds to following the protocol, or sending another (still valid) message, would result having the same network cost. For what illustrated among the assumptions on a rational node's behavior, this means a rational node will have no interest in not following the protocol.

Discouraging late responses. Nodes measure the timeliness of messages sent by other nodes, and can unilaterally punish non-timely nodes via a *penance* mechanism which roughly works as follows. When a node is the TRB leader, it includes in the command that is proposed to the state machine a fixed size *untimely vector* (a bitmap) of nodes it has seen to be untimely. Once the command is executed by the SMR, all nodes (but the leader) expect to receive (leveraging the message queues mechanism described above) a *penance message* from the untimely nodes. Again, the use of a fixed-size for this vector encourages a rational node to actually specify the nodes it has perceived as non timely: in fact, not specifying any would result in the same network cost. Furthermore, when a node p detects a node q is misbehaving (e.g., q has not sent a message p was expecting), p marks q as faulty in its local *badlist vector*. As the untimely vector, this vector is diffused together with the command issued by node p when p is the leader in the TRB protocol. Non-Byzantine nodes will then stop communicating with any node q that is observed to be present in at least $f + 1$ different badlist vectors diffused as part of the TRB protocol. A rational node is then encouraged to follow the protocol.

Level 2: Work Assignment. This level allows requesting work to do to nodes (e.g., storing data), and ensuring that a node responds accordingly to the request, or that there is evidence that the node did not answer. It relies on the abstraction of a trusted altruistic node, called the "witness node", for the communication between nodes. The altruistic witness node abstraction is implemented on top of the participant nodes by means of

the IC-BFT replicated state machine of the underlying layer[9]. The Work Assignment protocol is composed of the following three sub-protocols.

(i) Guaranteed Response protocol. This protocol ensures rational nodes will provide a response to the requests they receive. When a node p has to send a request to node q, p can first attempt to send the request directly to q. If q does not reply, node p can then rely on the witness node (the replicated state machine) for delivering the request and obtaining the response. This latter way is more costly for q than the direct response. The *credible threat* of having to incur in higher communication costs encourages a rational node q to actually prefer the first way and reply to a direct request from a node. Moreover, if q is not responding for a request issued through the replicate state machine, the state machine after a given timeout will provide p the evidence that q did not reply.

(ii) Periodic Work protocol. This protocol allows leveraging the witness node to expect a periodic work to be performed by nodes (whose type depends on the application level services). If a node does not comply to this periodic work, the witness node can generate a Proof Of Misbehavior (POM) (described next) about the misbehaving node. A rational node has then the incentive to always follow this protocol.

(iii) Authoritative Time Service protocol. This protocol allows nodes to keep a consistent and updated time. This time is used by nodes to estimate the timeliness of messages and detect misbehaving nodes.

Level 3: Application (Backup). This level apart from defining the actual application benefits to nodes (i.e., fruition of the backup service), it allows also nodes to be able to detect invalid responses (e.g., incorrect stored data). An example suggested by the authors is the following: a node might first adhere to store a file, by returning a signed hash of the stored file. But later, upon a request to read that file, the node replies with a signed message that contains data not corresponding to the hash. These two messages together represent a POM that the node has misbehaved.

The role of POMs. POMs are periodically distributed to other nodes via the *Work Assignment* periodic work sub-protocol described previously. Leveraging this, it is possible to enforces nodes to send a POM if they have it, or a NOPOM message if they do not. This NOPOM message, following the balanced cost principle, has the same size of a POM message, hence rational nodes have no interest in not reporting a POM. By distributing the POMs, each node of the system can evict the misbehaving nodes.

4.3.2.2 BAR Gossip

BAR Gossip [42] is the first P2P live video streaming application designed in the BAR model, and it relies on the gossip-based paradigm for disseminating chunks of the stream among nodes.

In this work, the authors point out how gossip protocols owe their robustness to the randomness which drives the choice of partners to gossip with, but that this very same randomness poses problems in a model where nodes might deviate from the protocol.

[9]Details of how to implement the "witness node" via the SMR can be found in the original paper [60].

Thanks to this non determinism, nodes could for example justify the fact of exchanging chunks of the stream with only certain nodes. Or, nodes could free-ride, omitting to forward chunks to other nodes without being detected and blamed for this. In BAR Gossip the proposed approach is then to rely on a *verifiable pseudo-random* selection of the gossip partner. In this way, non determinism is avoided, and the robustness and scalability properties of gossip are still ensured.

BAR Gossip works as follows. An altruistic broadcaster node streams a live event. Client nodes sign up for the event *before* the start of the streaming, after which joining is not allowed. On signing up, each client generates a public/private key pair and sends *both* of them to the broadcaster node. Before the start of the live streaming, the broadcaster node publishes the list of participating client nodes together with their identity (e.g., IP address) and public key. Client nodes are in fact required to digitally sign all messages they send, thus being accountable for what they do [104].

The broadcaster diffuses the stream in the form of chunks of fixed size that in BAR Gossip are called *updates*. Time is divided into rounds of duration $T + \delta$. Here, T is the assumed upper bound length necessary to complete the gossip exchange described next. Instead, δ is assumed to be the maximum difference in the clocks of (non-Byzantine) nodes. A client node plays the streaming data contained in an update once its expiration time has elapsed, and then discards the update.

At each round, the broadcaster sends the current updates to a certain number of random client nodes. The authors point out that this number should be chosen to ensure that, with very high probability, the updates are broadcast to at least a non-Byzantine client, so that subsequent exchange to the rest of the nodes is ensured via the following two protocols (illustrated in Figure 4.10) which are described next:

- **Balanced exchange protocol.** The two gossiping client nodes exchange an *equal* number of updates between each other.
- **Optimistic push protocol.** The two gossiping client nodes exchange an *unequal* number of updates between each other.

Balanced exchange protocol. This protocol encourages rational nodes to fairly exchange stream chunks between them. The authors have in fact designed, and proved, that each phase of the protocol is a Nash equilibrium [61], meaning that the whole protocol is a Nash equilibrium. In a Nash equilibrium, rational nodes are encouraged to follow the protocol, because deviating from it does not increase their utility in the participation in the system. As previously discussed, under the Nash equilibrium framework rational nodes do *not* collude, and they assume other nodes follow the protocol.

The protocol is made of four phases. We briefly describe them illustrating why, in each phase, a rational node is encouraged to follow the protocol.

(1) Partner selection. In the first phase (not represented in the figure), a node chooses a gossip partner to exchange updates with. The choice is: *(i) deterministic*, because a pseudo-random generator (shared by all client nodes) is fed with the current streaming round number, and *(ii) unpredictable*, because this round number is signed by the initiator node. The chosen gossip partner, from the partner selection message itself, can

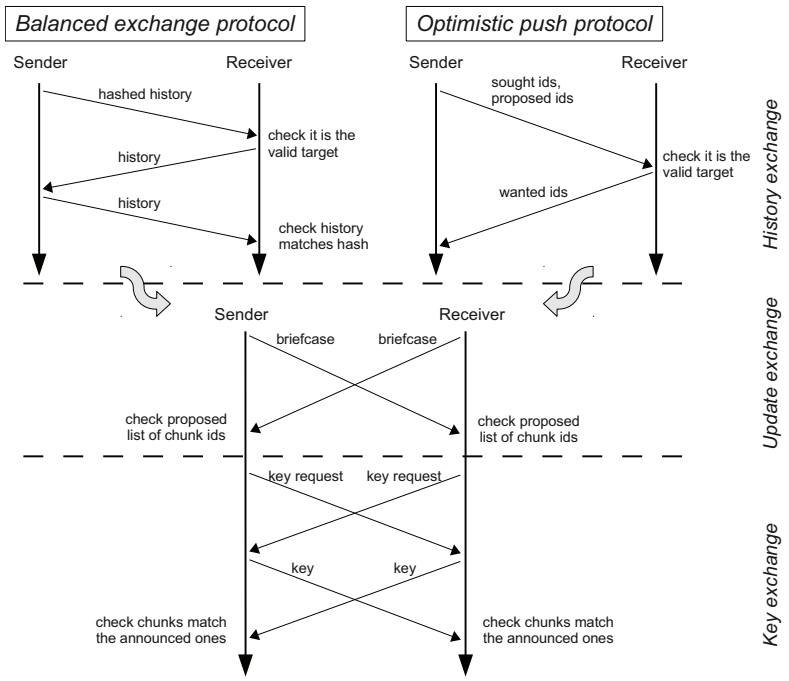


Figure 4.10 – BAR Gossip: Balanced Exchange and Optimistic Push protocols. (*Based on:* [42])

then decrypt and verify it is the intended valid gossip partner. If it is not the case, this received message, signed and thus non repudiable by the sender, represent a *Proof Of Misbehavior* (POM) of the initiator node. We describe afterwards the role of POMs.

(2) History exchange. Once a receiver node has accepted gossiping with the sender node, the two nodes exchange their list of the unexpired updates they have. In particular, the sending partner first sends a hashed history, and only after obtaining the other partner's history (in clear) it sends its history in clear. Again, this allows the receiver nodes to verify that the sender has correctly followed the protocol by transmitting valid update identifiers. Being this not the case, these two last messages represent a POM for the sender node.

(3) Update exchange. On reception of the updates histories, the two client nodes compute the maximum number k of updates that a node is missing and the other node can offer, thus leading to an exchange based on an equal number of updates. This implies the *history exchange* phase encourages nodes to announce all their unexpired updates in order to benefit from the maximum number of unexpired updates from the receiver node. Each of them then sends a *briefcase* message to the other node specifying the update ids being shipped and the updates themselves in encrypted form. A rational node is encouraged to ship in the briefcase message the update ids corresponding to the k updates supposed to be exchanged given the already sent history exchange message, making otherwise these two messages represent a POM against itself.

(4) Key exchange. A node requests to the gossip partner the key it used to encrypt the updates shipped in the briefcase exchanged in the former phase, so that it will then

be able to decrypt them. A rational node is encouraged to reply including in the key exchange message the key it used to encrypt updates in the briefcase message, otherwise these two messages altogether would represent a POM against itself. Also, if a node does not respond to a key request, the requester node continues sending the request. A rational node is then encouraged to provide the key in order to avoid wasting its download bandwidth by receiving the key requests from the other node.

The role of POMs. In BAR Gossip, a centralized trusted entity (which works in cooperation with the broadcaster) periodically collects POMs from nodes. The interaction relies, as in BAR-B [60], on the *balanced cost* principle: the reply of nodes must be of a fixed size. Sending or not the POMs thus does not increase the benefit of rational nodes (e.g., they cannot save bandwidth). Furthermore, also the identity of non replying nodes is communicated to the broadcaster. These nodes are then *evicted* from the live streaming by means of an eviction list (of fixed size) which is sent together with the updates diffused by the broadcaster. Because of these mechanisms, a rational node is thus encouraged to follow the protocol.

Optimistic push protocol. The balanced exchange protocol requires that clients can exchange an equal number of updates, which can be a limitation if clients genuinely have no updates to exchange (e.g., they did not get the latest updates due to networking issues). The optimistic push protocol comes into play to overcome this situation. The protocol takes its name from the fact that a gossiping node is exchanging updates with the selected gossip partner with no guarantees provided by the protocol that this latter will do the same. Its functioning is similar to that of the balanced exchange protocol, but it allows an *unequal* number of updates to be exchanged between the two gossiping nodes. The optimistic push protocol is in fact *not* designed as a Nash equilibrium (as the balanced exchange protocol), and it allows more flexible participation of rational nodes in the *history exchange*: the authors experimentally showed that is beneficial for rational nodes to faithfully participate in it.

Tolerating Byzantine behavior. In BAR Gossip, due to the signature used by broadcaster when diffusing updates, Byzantine nodes cannot compromise the streaming *content* by forging the updates. Also, the authors have shown through experimental evaluations that non-Byzantine nodes perceive high *quality* (predictable throughput and low latency) of the streaming service despite a large portion (20%) of Byzantine nodes.

Dynamic membership and optimizations: FlightPath. FlightPath [105] is a refinement of BAR Gossip, in which the authors modify the original protocol in order to cope with dynamic membership and improve performances, e.g., jitter reduction and better bandwidth utilization. To achieve that, the protocol is designed to be an *approximate Nash equilibrium* [95] instead of a strict Nash equilibrium: rational nodes have some flexibility in the execution of the protocol. For example, this makes it possible to reduce jitter, by allowing an unbalanced exchange of stream updates. And also, it makes nodes better use their bandwidth, by allowing them to redirect the requests to other nodes if they are unable to respond to them.

4.3.3 Summary

In the previous sections, we have discussed the Byzantine and the BAR model. Moreover, we have detailed the functioning of a few practical P2P systems tolerating Byzantine behavior (Fireflies/SecureStream and Nysiad), or Byzantine and rational behavior altogether (BAR-B and BAR Gossip). We provide in the following the set of requirements that would ideally drive the design of a spam-resilient gossiping protocol accounting for both Byzantine and rational behavior. Then, we discuss why the previously illustrated systems do not provide a solution which satisfy all of these requirements.

4.3.3.1 Requirements of a spam-resilient gossip protocol

We provide here a list of the desired characteristics of a gossip protocol able to limit spam, and accounting for Byzantine and rational behavior.

- **Spam-resilient dissemination.** The dissemination protocol should be able to limit the diffusion of spam messages, while still ensuring reliable dissemination of non-spam messages. Furthermore, the protocol should be designed tacking inspiration by reputations systems [106], in which nodes build a reputation during time. As such, nodes which send less spam and contribute more to the filtering of spam messages, should receive less spam in return.
- **Tolerating an upper bound of Byzantine nodes.** The protocol should be designed in order to ensure the system properties assuming an upper bound of Byzantine nodes might misbehave for arbitrary reasons (provided other non-Byzantine nodes follow the protocol).
- **Incentives to follow the protocol.** Accordingly to the assumptions about a rational node's behavior discussed in Section 4.3.2, nodes seek long term benefit in the participation. Hence, the protocol should be designed so that by deviating a node should fear to have its benefits reduced. The protocol should then provide incentives to encourage rational nodes to follow the protocol.
- **Incentives to monitor nodes and report misbehaviors.** Related to the previous point, the protocol should also provide incentives for nodes to monitor other nodes' behavior, and report in case of misbehavior, in order to ensure the monitored nodes follow the protocol.
- **Scalable number of monitors per node.** For the monitoring to be scalable, it should be limited to few monitors per node.
- **No external entity to assign monitors.** For the protocol to be scalable, it should not rely on an external (possibly distributed) entity to assign monitors to node. The assignment of monitors should be done in a decentralized fashion by means of a protocol run by the participant nodes themselves.
- **No external entity to punish misbehaviors.** For the protocol to be scalable, it should not rely on an external (possibly distributed) entity to take eviction decisions either. Participant nodes, in particular the monitors of a node, should decide about whether to punish a monitored node or not.

4.3.3.2 Drawbacks of existing solutions

We briefly summarize here why the existing systems presented so far do not satisfy the aforementioned requirements.

Fireflies [97]/SecureStream [96]. Fireflies is a Byzantine-tolerant group membership protocol on top of which an application layer, like the SecureStream streaming protocol, can be implemented. In SecureStream nodes forward an update to the successor nodes on each Fireflies ring. We can comment on these two protocols individually. Concerning the Fireflies protocol, it is "only" Byzantine-tolerant, and not Incentive-Compatible Byzantine Fault Tolerant. In fact, no mechanisms encourage a rational node to actually forward the gossip messages it receives. This could possibly impact the reliability of gossip if a large percentage of nodes does not participate in the gossiping. Concerning SecureStream, the fact of forwarding to the multiple rings successors ensures reliability due to the randomness and redundancy of the multiple rings overlay. Yet, as we have discussed in Section 4.2, this turns out to be the ideal condition to also effectively spread spam messages. Also, SecureStream assumes that generally only a limited fraction of nodes could behave selfishly, and proposes a simple auditing mechanism which relies on external auditing entities.

Nysiad [99]. It translates a crash-tolerant protocol in a Byzantine-tolerant protocol by replicating each node with a state machine composed of a few guards, which are assigned by a logically centralized entity. Furthermore, Nysiad replication protocol is not IC-BFT. In fact, as an example, the replicas could choose to omit sending the gossip messages in the final phase of the reliable ordered broadcast. This could possibly compromise the effective synchronization of the replicas in the case in which a Byzantine primary has only sent the order certificate to a subset of its replicas.

BAR Replicated State Machine [60]. The first two layers of the BAR-B architecture provide an IC-BFT replicated state machine. This state machine abstracts a trusted altruistic "witness node", and it is implemented by all nodes. Whenever a node does not comply to the protocol, other nodes have and share evidence of the misbehavior and can individually refuse to communicate with the misbehaving node, without relying on a 3rd party entity. The approach does not scale over tens or few hundreds of nodes, as all the communication made by a node passes through the state machine composed by all other nodes. Also, nodes can issue commands in turns, according to a round-robin leader rotation.

BAR Gossip [42]. It is an IC-BFT one-to-many broadcasting protocol. It allows preserving the randomness and robustness of gossiping by employing a deterministic, yet (pseudo) random, choice of the dissemination targets. Furthermore, an incentive-compatible balanced exchange protocol ensure nodes exchange stream chunks, otherwise a proof of their misbehavior is collected by a 3rd party centralized entity, which will lead to the eviction of misbehaving nodes. Nevertheless, the choice of dissemination targets still follows a random scheme, which from what we said in Section 4.2, makes gossiping an ideal vector for spreading spam if BAR Gossip is used for a many-to-many

dissemination service. Moreover, the cooperation on a short timescale (the time of a single gossip exchange) limits nodes to having only short-term reputation. Yet, in a collaborative service where nodes have to filter spam, nodes should be able to build a long-term reputation so that the nodes which filtered the most of spam are rewarded by receiving less spam in the future.

Table 4.1 shows, for each requirement of a spam-resilient gossiping protocol, whether the existing systems presented in this chapter provide or not a satisfactory solution. From the table, we observe that none of the described systems provide a comprehensive solution allowing to implement a scalable and reliable gossip protocol which is able to limit the diffusion of spam messages in presence of Byzantine and rational behavior.

Property \ *System*	**Fireflies / Secure-Stream**	**Nysiad**	**BAR State Machine**	**BAR Gossip**
Spam-resilient dissemination	X	X	-	X
Byzantine-tolerant	√	√	√	√
Incentives to follow the protocol	X	X	√	√
Incentives to monitor nodes and report misbehaviors	X	X	√	√
Scalable # of monitors per node	√	√	X	-
No external entity for assignment of monitors for the nodes	√	X	-	-
No external entity for punishing misbehaviors	X	X	√	X

Table 4.1 – Properties of the presented Byzantine and BAR tolerant P2P systems. Legenda: (√): prop. holds; (X): prop. does not hold; (-): prop. does not apply.

4.4 Conclusion

An effective way to disseminate information in P2P collaborative services (e.g., a P2P Web forum or Q&A Web site) is by employing gossip-based dissemination protocols. In gossip-based dissemination protocols, once a node receives a message for the first time, it forwards it to a random subset of nodes. This simple approach has been shown to be highly scalable and reliable in spreading the information in the network.

Nevertheless, gossip-based dissemination protocols have a drawback: they are unable to limit the dissemination of spam messages. Indeed, messages are randomly and redundantly disseminated in the system, and it is enough that a small subset of nodes forward spam messages, to have them received by a large percentage of nodes.

To make things worse, in a practical context like the Internet, where there is no authority controlling what nodes do, nodes can deviate from the protocol for different reasons. Nodes can arbitrarily deviate from the protocol due to a bug or misconfiguration, or even because compromised by a malicious attacker who took control of their machines (Byzantine behavior). But also, nodes can deviate from the protocol following

a selfish choice, seeking to maximize their net benefit while reducing their costs, e.g., by not forwarding messages they are supposed to forward (rational behavior). The BAR model is a model accounting for both these two classes of faults.

To the best of our knowledge, no currently existing gossip-based dissemination protocol is able to limit the dissemination of *spam*, while accounting for Byzantine and rational behavior. In the next chapter, we present $\mathcal{F}ire\mathcal{S}pam$, a novel gossiping protocol designed in the BAR model, that is able to limit the diffusion of spam.

5

FireSpam: Spam-resilient Gossiping in the BAR Model

Contents

We have shown in the previous chapter that gossip-based dissemination protocols are ideal vectors for the diffusion of spam. In fact, the straightforward approach of letting nodes locally filter the messages they detected as spam is not effective. This is due to the fact that gossip-based dissemination protocols are highly redundant and random: nodes possibly receive the same message several times and from different nodes. It is then enough that a few nodes do not stop the dissemination of a spam message, to have it spread to a significant fraction of nodes.

To the best of our knowledge, no currently existing gossip-based dissemination protocol is able to limit the dissemination of *spam*. In this chapter we present $\mathcal{F}ire\mathcal{S}pam$, a

novel gossiping protocol that is able to limit spam dissemination. In fact, in $\mathcal{F}ire\mathcal{S}pam$ nodes are organized in a ladder topology according to their capability to filter spam: nodes having a high (resp., low) filtering capability are located at the top (resp., bottom) of the ladder. Messages are disseminated from the bottom to the top of the ladder, which acts as a progressive spam filter. The rationale behind this topology is that nodes that actively filter spam (i.e., those with a high filtering capability) progressively climb the ladder and will eventually be less overwhelmed by spam messages.

While organizing nodes in a ladder topology can easily be achieved if all nodes in the system behave correctly [40, 41], it appears challenging to build such a topology in the presence of nodes acting maliciously or selfishly. Nodes with such behaviors are common in P2P systems and they do thus need to be taken into account when designing $\mathcal{F}ire\mathcal{S}pam$ for it to be usable in practice. Consequently, we have designed $\mathcal{F}ire\mathcal{S}pam$ considering the BAR model [60]. This model states that there are three kinds of nodes: *altruistic* nodes that strictly follow the protocol, *Byzantine* nodes that can behave arbitrarily (hence, to the extreme, maliciously), and *rational* nodes that behave selfishly and are willing to deviate from the protocol if there is a gain in doing so. We have thus designed $\mathcal{F}ire\mathcal{S}pam$ as an Incentive-Compatible Byzantine Fault Tolerant (IC-BFT) protocol [60] in order to tolerate Byzantine nodes and discourage rational nodes from acting selfishly. In fact, $\mathcal{F}ire\mathcal{S}pam$ design choices are such that it tolerates an upper bound of Byzantine nodes, which by misbehaving cannot harm the system properties. Moreover, $\mathcal{F}ire\mathcal{S}pam$ mechanisms allow misbehaving nodes to be detected and evicted from the system. Furthermore, $\mathcal{F}ire\mathcal{S}pam$ encompasses a set of incentive-compatible mechanisms that make the protocol a strict Nash equilibrium [61]. More precisely, the incentive mechanisms used in $\mathcal{F}ire\mathcal{S}pam$ ensure that it is in a rational node's best interest to always follow the protocol, as it will have no gain in not doing so. We have built $\mathcal{F}ire\mathcal{S}pam$ on top of the Fireflies [97] Byzantine-tolerant overlay network. The Fireflies protocol tolerates Byzantine nodes but is not explicitly designed for coping with rational behavior. In order then to actually have a full Incentive-Compatible Byzantine Fault Tolerant protocols stack ($\mathcal{F}ire\mathcal{S}pam$ on top of Fireflies), we illustrate in Appendix A the modifications to Fireflies to discourage rational behavior and thus to make it an IC-BFT protocol.

We assessed the robustness of $\mathcal{F}ire\mathcal{S}pam$ both theoretically and practically. From a theoretical point of view, we proved that the protocol is a strict Nash equilibrium: rational nodes will follow the protocol as they cannot expect a gain by deviating, or worse, they fear to compromise their benefit, by deviating. From a practical point of view, we assessed through an extensive simulation study that $\mathcal{F}ire\mathcal{S}pam$: *(i)* reliably delivers good messages; *(ii)* drastically limits the dissemination of spam messages; *(iii)* cannot be harmed by a set of Byzantine nodes colluding to break the ladder topology; and *(iv)* has reasonable bandwidth costs due to the robustness properties.

The rest of this chapter is organized as follows. In Section 5.1 we present the system model underlying the $\mathcal{F}ire\mathcal{S}pam$ protocol. We describe the design and functioning of the protocol in Section 5.2. Its robustness is discussed in Section 5.3 (a sketched proof of its incentive-compatibility is further given in Appendix B). We present the performance evaluation in Section 5.4 before concluding in Section 5.5.

5.1 System Model

In this section we present the system model underlying the $\mathcal{F}ire\mathcal{S}pam$ protocol. The system model decomposes into a message model, a spam filtering capability model, a fault model, and a set of system assumptions.

Messages. Nodes can generate two types of messages: *good* messages and *spam* messages. *Good* messages are of interest to all the nodes and must be reliably disseminated in the network. Instead, *spam* messages should be filtered during the dissemination process in order to reach as few nodes as possible. By spam message we do not refer merely to junk content, for which automatic spam filtering solutions could be employed from the artificial intelligence / machine learning field. We consider in fact the more general (and subtle) type of spam message whose content is irrelevant, inappropriate or just misleading for the other participants, but which nevertheless could pass undetected an automatic spam filter. Taking as example the case of an Internet forum dedicated to a single soccer team, one malicious user might spread a false information about the club or its players, yet this message would pass through an automatic filter.

Node filtering capability. Each node in $\mathcal{F}ire\mathcal{S}pam$ has a *spam filtering capability* (also called "pollution awareness" in [58]) that expresses the ability of a node to detect *spam* messages. We assume that the messages classified by nodes as *good* or *spam* effectively fall into that category with a very high probability. That is, nodes do very few (i.e., less than 5%) false positive or false negative message classification (*good* messages classified as *spam* and *spam* messages classified as *good*, respectively).

Fault model. We consider the BAR model [60], in which nodes can be Byzantine, altruistic or rational. *Altruistic* nodes behave exactly as dictated by the protocol, and they can only fail by crashing. On the other side of the spectrum, *Byzantine* nodes can deviate from the protocol for any reason (e.g., a failure, a bug, a threat) and in doing so, they can take arbitrary decisions (e.g., dropping *good* messages). Furthermore, Byzantine nodes can collude together. Finally, *rational* nodes aim at maximizing their benefit according to a known utility function. We suppose that rational nodes join and remain in the system for a long time and seek a long-term benefit. Moreover, rational nodes do not collude and assume that other nodes are altruistic. A rational node can deviate from the protocol if the generated utility increases accordingly (e.g., they can decide not to forward messages for saving bandwidth). We consider the utility to be proportional to the amount of *good* messages received and conversely proportional to the amount of *spam* messages received as well as to the amount of bandwidth consumed by receiving/forwarding messages from/to other nodes. Specifically, the utility perceived by a rational node can be represented along the following axes:

1. (G) Receiving as much as possible (possibly, all) of the *good* messages disseminated in the system;

2. (S) Receiving as little as possible (possibly, none) of the *spam* messages disseminated in the system;

3. (F) Forwarding as little as possible (possibly, none) of the received messages.

We can thus informally define the utility function as:

$$\boxed{B = \alpha G + \beta S + \gamma F} \tag{5.1}$$

where $\alpha \gg \beta \gg \gamma$, intuitively meaning that nodes do not want to trade-off the reliable reception of good messages to receive less spam or consume less bandwidth. Furthermore, for a node it is of more benefit receiving less spam, rather than saving some bandwidth.

System assumptions. We assume a cryptographic identification of nodes as it is assumed in many practical gossip-based content dissemination protocols (e.g., [42]). Each message sent in the network is therefore signed using the sender's cryptographic key. We also assume cryptographic primitives cannot be subverted, which *(i)* exclude the impersonation attacks [107], and *(ii)* make senders accountable [104] for the messages they send. Furthermore, we assume that non-Byzantine nodes maintain clocks synchronized within δ seconds and communicate over reliable links. Moreover, we assume that messages sent by a sender to a given receiver are always received within a bounded time. We also assume that trivial Denial-of-Service (DoS) attacks can be detected and suppressed (e.g., [108]).

5.2 The $\mathcal{FireSpam}$ protocol

We present here $\mathcal{FireSpam}$, a spam resilient gossiping protocol in the BAR model. $\mathcal{FireSpam}$ organizes the nodes in a ladder topology, and it employs a set of mechanisms to tolerate Byzantine behavior and to discourage rational behavior. We start by a description of the ladder topology as a progressive spam filter approach. Then, we describe the challenges raised by the construction of the ladder in the presence of Byzantine and rational nodes. Afterwards, we illustrate the basic mechanisms underlying the $\mathcal{FireSpam}$ protocol and finally we explain its functioning in more detail.

5.2.1 Ladder topology

In $\mathcal{FireSpam}$, nodes are organized according to their filtering capability in a ladder topology. Nodes at the bottom of the ladder have the lowest filtering capability, whereas nodes at the top of ladder have the highest filtering capability. Messages produced by a node p are first sent to a set of nodes located at the bottom of the ladder (called the *Publication View* of p). When a node q receives a given message for the first time, it decides whether to filter it (when it considers it as a spam message) or to forward it to a set of nodes which surround it in the ladder (called the *Forwarding View* of q). Good messages will thus eventually reach all nodes in the ladder, whereas spam messages will be progressively filtered by nodes along the ladder. The rationale behind this ladder topology is that nodes that actively filter spam (i.e., have a high filtering capability), will progressively climb the ladder and be located at the top of it. Consequently, they will receive less spam than nodes with a lower filtering capability. Nodes are thus rewarded to filter spam.

Figure 5.1 shows an example of message dissemination in $\mathcal{FireSpam}$. In this figure, a node p generates a spam message m and sends it to its publication view (in

the figure, the three nodes at the bottom of the ladder). Nodes that are able to filter m are represented with a thick border, whereas nodes represented with a thin border are not able to filter m. Note that as nodes are organized in the ladder according to their filtering capability, it is more likely to find nodes able to filter m at the middle and top of the ladder rather than at the bottom of it. In the example, nodes that do not filter m, forward it to nodes in their forwarding view. We observe that m is eventually filtered as it progressively reaches nodes that are able to filter it, thus preventing nodes at higher positions of the ladder from receiving it (in the figure, unfilled circles).

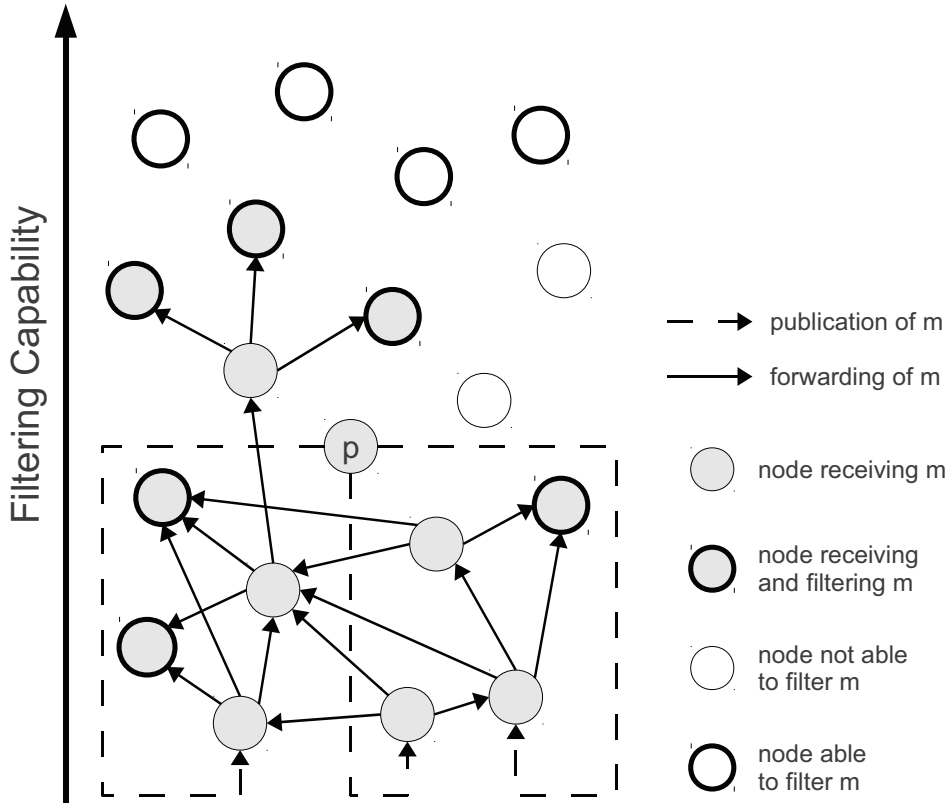


Figure 5.1 – $\mathcal{F}ire\mathcal{S}pam$ ladder topology: example of spam message dissemination and filtering.

5.2.2 Ladder construction challenges

Organizing nodes in a ladder topology can easily be achieved if all nodes in the system behave correctly. In fact, gossip-based protocols like T-Man [40] / Vicinity [41] can be employed to organize nodes according to a given proximity function. In such protocols, nodes gossip their value and that of their current proximity function optimal neighbors. In this way, nodes can adjust their views with better neighbors (according to the proximity function) and ultimately make the overlay converge to the desired topology. For the case of our desired ladder topology, nodes would exchange their (self-announced) spam filtering capability and that of their current neighbors. On merging the views, the proximity function would return the nodes which have closest filtering

capability, thus converging ultimately to a ladder topology.

But, it appears challenging to build, and leverage for message dissemination, such a ladder topology in the presence of rational and Byzantine nodes. In particular, for $\mathcal{F}ire\mathcal{S}pam$ to function under this possible behavior of the participant nodes we need to ensure:

- **Reliable dissemination:** $good$ messages must be forwarded by all nodes. Consequently, rational and Byzantine nodes should be discouraged or punished when dropping $good$ messages.
- **Correct node evaluation:** in order for the ladder to have a correct topology, it is crucial that the filtering capability of nodes is regularly and correctly assessed. For instance, rational and Byzantine nodes should not have ways to forge filtering capability assessment.
- **Correct view assignment:** the publication and forwarding views of nodes must be correctly assigned (i.e., according to their filtering capability). For instance, rational and Byzantine nodes should not be able to forge view assignments in order to take a larger benefit from the protocol (e.g., choosing as neighbors nodes with a high filtering capability).

The three concepts are tied together, as summarized in Figure 5.2. In fact, the view assignment determines to which nodes messages should be disseminated. Message dissemination —how a node contribute in forwarding good and/or spam messages— on the other side, is the criterion upon which a node is evaluated with respect to other nodes. And, according to the results of the node evaluation, a node is supposed to have a (forwarding) view assigned which is made of nodes with similar filtering capability.

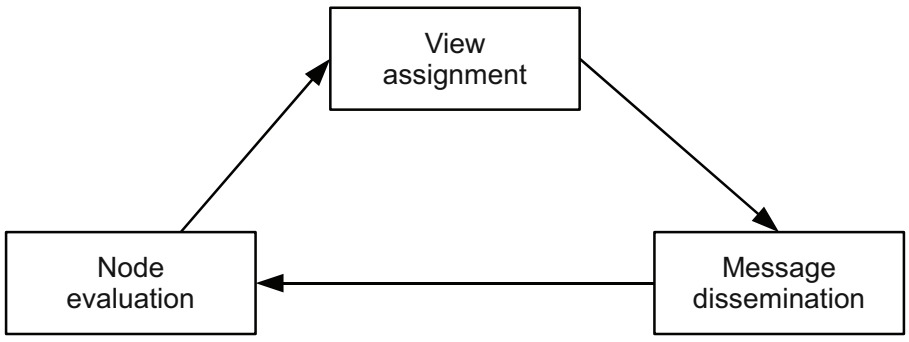


Figure 5.2 – $\mathcal{F}ire\mathcal{S}pam$ main concepts.

5.2.3 Protocol mechanisms

In order to deal with the aforementioned challenges, $\mathcal{F}ire\mathcal{S}pam$ decomposes in a set of mechanisms enabling the reliable construction of the ladder topology and the reliable dissemination of (good) messages within it.

To enable the reliable dissemination of *good* messages, nodes classify messages in three categories, as illustrated in Figure 5.3. The *good* and *spam* categories are

used when the node can assess with evidence that the message is a good or a spam message, respectively. Instead, the *undetermined* category is used when the node is not able to assess with certainty whether the message is a spam or a good message. This classification allows detecting nodes that intentionally filter good messages. Indeed, a node is expected to send all messages that it classifies in the *good* and *undetermined* categories. Consequently, if a node does not receive a message m it classified as *good* from a node p it is in the forwarding view of, it will conclude that p intentionally filtered the message m. In this case, as we will see, it will take actions to punish p.

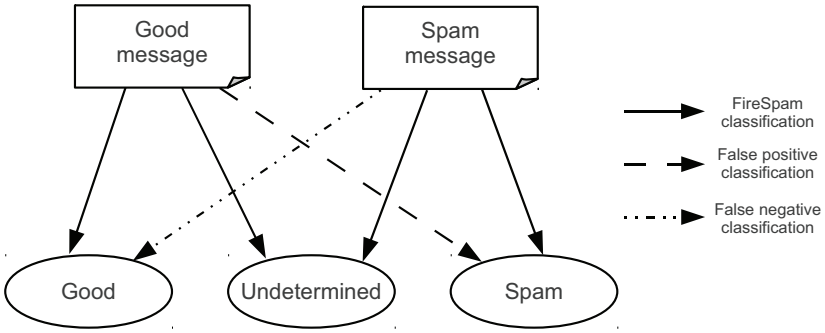


Figure 5.3 – $\mathcal{F}ire\mathcal{S}pam$ message classification categories.

To enable correct node evaluation and correct view assignment, a node is deterministically assigned a set of *node monitors*. The reason why $\mathcal{F}ire\mathcal{S}pam$ relies on node monitors for node evaluation and view assignment is the following: a node obviously should not self-assess its filtering capability. Otherwise, rational nodes could simply claim they have the highest possible filtering capability. An intuitive idea would be to rely on nodes that are in the view of the node to be assessed. Indeed, these nodes receive all messages sent by p and seem thus to be good candidates to evaluate its capability to filter spam. Nevertheless, these nodes could act rationally by forging the evaluation of nodes with high filtering capabilities in order to keep them close by in the ladder and benefit from their filtering capability. It is thus necessary to rely on third party node monitors and to define incentive mechanisms guaranteeing that these node monitors will behave correctly. Node monitors of a node p are actually responsible for the following tasks:

- Assessing the filtering capability of p.
- Assigning the publication and forwarding views of p.
- Detecting any misbehavior of p and possibly blacklisting and evicting p from the system.
- Detecting any misbehavior of other node monitors of p and possibly evicting them from the system.

Node monitors for every node in the system are allocated using the Fireflies overlay network [97]. The Fireflies protocol relies on a multiple rings topology and allows to deterministically associate to each node a set of nodes (node monitors in the case

of $\mathcal{FireSpam}$) such that, with very high probability, a majority of them are non-Byzantine. Note that in $\mathcal{FireSpam}$ the node monitors also use the gossip broadcasting sub-protocol provided by Fireflies in order to reliably disseminate a message among all the non-Byzantine node monitors. This gossiping sub-protocol ensures that messages are reliably delivered, despite the presence of Byzantine nodes, and within a bounded time[1]. As we previously pointed out, we illustrate in Appendix A the modifications to Fireflies to make it an IC-BFT protocol.

5.2.4 Protocol description

The $\mathcal{FireSpam}$ protocol is decomposed in a set of *seven* steps that are illustrated in Figure 5.4 and that are described in the following. The design choices which make the protocol tolerate Byzantine nodes as well as the incentive mechanisms that are used to encourage rational nodes to follow each one of the protocol steps are further discussed in the next section.

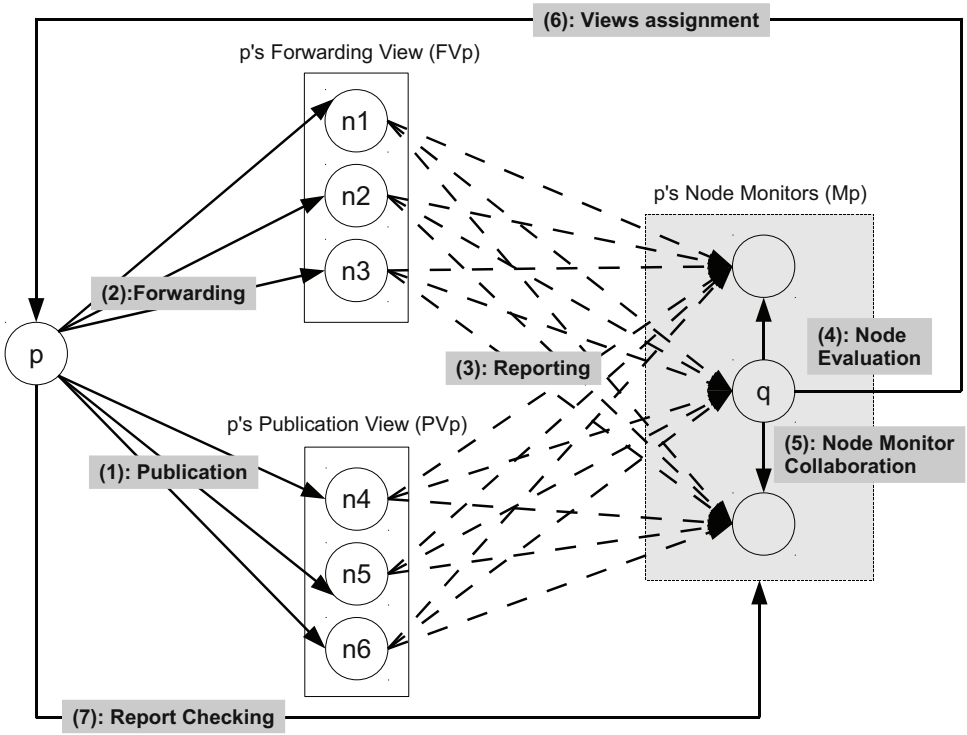


Figure 5.4 – $\mathcal{FireSpam}$ roles and protocol steps.

(1) Message publication. To publish a message m, node p sends it to all the nodes in its publication view, i.e., nodes n_4, n_5, and n_6.

(2) Message forwarding. When node p receives a message m, if it does not classify it as spam, it forwards it to all nodes in its forwarding view, i.e., nodes n_1, n_2, and n_3.

(3) Message reporting. When nodes receive a message from p, they send a REPORT

[1]Timeouts can be dynamically set based on the monitoring of the current message delivery latency across the gossip-based platform.

message to p's node monitors.

(4) Node evaluation. Node monitors use the reports received about p to assess its filtering capability. This is made possible by the fact that all messages are broadcast in the entire network. Consequently, it is possible for a node monitor to compare the filtering capability of two nodes by simply comparing the overall number of messages they sent during the same time interval: a node p sending less (resp., more) messages than a node q if it has a higher (resp., lower) filtering capability than q. Recall that we make the assumption that a node does not incur in more than 5% misclassifications when assessing whether messages are spam or good. If a node was evaluating other nodes throughout the lifetime of a system, it could thus theoretically achieve a 5% precision on the evaluation of other nodes. In practice, however, nodes are evaluated on finite time intervals. To obtain a reasonable precision, we evaluate nodes on time intervals during which a large number of messages are broadcast (i.e., 1000 messages in the evaluation presented in Section 5.4).

In order to compute the number of messages actually sent by a node p, node monitors rely on MAJORITYREPORT messages. These reports are sent by p's node monitors whenever they receive a majority of REPORT messages stating that p sent a message m. On reception of the first MAJORITYREPORT message from a node monitor of node p, a node monitor updates the evaluation of p by incrementing by one the number of messages sent by p.

Two things must be noted. First, MAJORITYREPORT messages are sent to all node monitors in the system in order to inform them that p has actually sent the message m to a majority of nodes in its view. Second, the reception of REPORT messages is also used by node monitors to detect if a monitored node p did not forward *good* messages. Specifically, node monitors of a node p check that p correctly forwarded the messages that it considers *good*. If that is not the case, it considers that p has not followed the protocol and can take actions against p.

(5) Node monitor collaboration. In order to ensure that node monitors follow the protocol, each node monitor of node p is responsible for controlling whether other node monitors are behaving correctly or not. Specifically, each time a node monitor sends a message to other node monitors, it expects to receive a "similar" message from other node monitors (i.e., if it sends a MAJORITYREPORT for the node p, it expects the other monitors to send a MAJORITYREPORT for p as well, possibly made with REPORT messages produced by different neighbors of p). The reason is that they follow a deterministic protocol that produces the same outputs provided they receive the same inputs. If a node monitor does not receive an expected message from one of the other node monitors, it collaborates with the other node monitors of p in order to evict it from the system (if a majority of node monitors are willing to do it).

(6) View assignment. Nodes belonging to the set of node monitors of node p periodically compute a forwarding view and a publication view for p. They then send the computed views to the other node monitors of p, which check their correctness. If a node monitor q does not send a view or sends a wrong view, the other node monitors can possibly evict q (if a majority of node monitors are willing to do it).

The computation of views is performed as follows: every node monitor knows the

set of nodes in the system and their filtering capability (thanks to the MAJORITYREPORT messages it receives). Views are then assigned using the same mechanism than the one used in Fireflies to assign monitors to nodes [97]. This mechanism allows to deterministically associate to each node sets of $2t + 1$ nodes, out of which there is a probability $\mathcal{P}$ that at most t nodes are Byzantine. The larger the set of nodes from which the $2t + 1$ nodes are chosen, the higher the probability $\mathcal{P}$. In the remaining of this paper, we call *candidate set* the set of nodes from which the $2t + 1$ nodes are chosen. Additionally, we impose that the forwarding and publication views of any node be disjoint from its node monitors set. This condition is necessary to ensure that node monitors have no incentive in not adhering to the protocol for their own good.

(7) Report checking. As rational nodes assume that other nodes are altruistic, a rational node may decide to save bandwidth by omitting to send reports. In order to avoid this behavior, nodes sending messages periodically query their node monitors to check that nodes in their views correctly reported about the messages they sent. Node monitors having received REPORT messages send them back to senders. Note that these replies have *fixed size* in order to encourage node monitors to actually reply with the correct information: there is indeed no incentive in sending wrong information as this will consume the exact same amount of bandwidth.

5.3 Robustness

Assessing the robustness of $\mathcal{F}ire\mathcal{S}pam$ against both Byzantine and rational behaviors implies showing that the protocol is actually **Incentive-Compatible Byzantine Fault Tolerant** (IC-BFT) [60]. This means that the protocol should satisfy the following two requirements:

1) Byzantine fault tolerance. The desired protocol properties (i.e., reliable dissemination, correct node evaluation and correct view assignment) are guaranteed even in the presence of a bounded number of Byzantine nodes, *provided that non-Byzantine nodes follow the protocol*.

2) Incentive-compatibility. Rational nodes follow the protocol as they have no incentives in deviating from it: in deviating *(i)* they expect to get the same or an inferior utility from the participation in the system, or even worse *(ii)* they fear to be blacklisted by some nodes or evicted from the system.

5.3.1 Tolerating Byzantine behavior

As discussed in Section 5.2, $\mathcal{F}ire\mathcal{S}pam$ design choices guarantee the desired protocol properties in the presence of a bounded amount of Byzantine nodes, provided that all other nodes follow the protocol. Indeed, forwarding and publication views, as well as node monitors are assigned in a way that guarantees that they comprise a majority of non-Byzantine nodes. In particular, as in $\mathcal{F}ire\mathcal{S}pam$ the node monitors of a node are allocated using the Fireflies [97] overlay network, we first report the mechanism used in Fireflies to assign to a node with high probability a majority of non-Byzantine node monitors selected from the whole node population. Then, using the same mechanism, we describe how to assign to a node with high probability a majority

of non-Byzantine nodes in the view, this time selected from a small candidate set of nodes surrounding the node in the ladder.

Tolerating Byzantine nodes in the node monitors set. Consider the underlying Fireflies multiple rings overlay network[2]. This underlying network is made of $2k+1$ rings, such that the position, and hence the ordering, of the nodes on each ring is deterministically dictated by a pseudo-random hash function: $H(node\_id||ring\_id)$. In $\mathcal{F}ire\mathcal{S}pam$, the predecessors of a node p on each of the $2k+1$ Fireflies rings form the node monitors set for the node p. Suppose now that the bounded amount of Byzantine nodes which can be tolerated in the system is b. This means that for any node taken in the *whole* system made of n total nodes, the probability for it to be a Byzantine node is $b/n = P^n_{byz}$. Given this probability P^n_{byz}, we have discussed in Section 4.3.1.1 the approach employed in Fireflies to choose the number $2k+1$ of rings, and hence the number of the pseudo-random node monitors assigned to each node, in order to select the smallest k for which the probability of having more than k Byzantine nodes in the node monitors set is very small. This minimum value of k can be obtained by solving the cumulative binomial distribution in Formula 5.2:

$$\boxed{min(k) : P = B(k, 2k+1, 1 - P^n_{byz}) < \epsilon} \tag{5.2}$$

As an example, if we aim at tolerating at most 5% of Byzantine nodes among all the nodes in the system, choosing $k = 2$ would lead to a 99.88% probability of having (at least) a majority of non-Byzantine nodes in a node monitors set of size $2k + 1 = 5$. Nevertheless, as noted in Fireflies [97], if node monitors rely on their successors on the multiple rings to gossip messages intended to be delivered to all the node monitors of the system, the choice of the number of the rings —and hence of the gossip targets— must be (on average) in the order of $ln(N) + O(1)$ to ensure reliable atomic dissemination to all nodes [30]. This means for example that for a system size of 1,000 nodes, a $k = 4$ would be a suitable value.

Tolerating Byzantine nodes in the views. The probabilistic assignment of a majority of non-Byzantine nodes in the views follows from similar probabilistic arguments as the one discussed before for node monitors sets. Accordingly to the ladder topology of $\mathcal{F}ire\mathcal{S}pam$ design, the forwarding view of a node p should be composed of the nodes whose filtering capability surrounds that of p. Consider then that we might want to select $2t+1$ (random) nodes to form the forwarding view of p from a candidate set of c nodes surrounding p, e.g., taken from a contiguous range which is half immediately below p and the other half immediately above p in the ladder. Then, if $c \leq b$ (the bounded amount of Byzantine nodes), in the worst case where all Byzantine nodes form a contiguous block together, the probability to find a Byzantine node in this range is 100%. Increasing c to be $r * b$, then the probability to find a Byzantine node in the contiguous candidate set is reduced by a factor r, thus becoming $P^c_{byz} = 1/r$. It follows that the candidate set size (and hence, the $P^c_{byz} = 1/r$), together with the view size $2t+1$, must be appropriately chosen such that the probability P to have at most t

[2]The reader can find in Section 4.3.1.1 of this document a description of the functioning of Fireflies, and can refer to the original Fireflies paper [97] for additional details.

Byzantine nodes out of the $2t + 1$ in the view is very low. This probability P is again expressed by a cumulative binomial distribution, as indicated by Formula 5.3:

$$\boxed{min(t) : P = B(t, 2t + 1, 1 - P^{c}_{byz}) < \epsilon} \tag{5.3}$$

As an example of how to pick the candidate set size c and the view size $2t + 1$, we consider again the case where we aim at tolerating at most 5% Byzantine nodes among all the system nodes. If we choose $c = 5 * b$ and $t = 6$, there would be a 99.29% probability of having (at least) a majority of non-Byzantine nodes in a view of size $2t + 1 = 13$.

As final considerations on the views, the following three things can be noted. First, they are *asymmetric*: if a node p is in node q's forwarding view, the opposite is not necessarily true. Second, the very same aforementioned discussion about majority of non-Byzantine nodes in forwarding views applies also to the assignment of publications views. In fact, in this case the candidate set for the publication view of a node p is made of the first c nodes in the ladder, instead of the c nodes surrounding p. Third, the majority value $t + 1$ should be a proper fanout to ensure the reliable dissemination among all the nodes of the system. In fact, $\mathcal{F}ire\mathcal{S}pam$ dictates to forward a message to all the $2t + 1$ nodes in its view: this accounts for the fact that only the $t + 1$ non-Byzantine nodes among them might forward it afterward. In the evaluations presented in Section 5.4, we have experimentally found that view size made of 13 nodes ensured reliable dissemination in a system with 1,000 nodes.

5.3.2 Discouraging rational behavior: incentive-compatibility

In order to show that rational nodes follow the protocol, we prove in a game theoretic framework along the lines of systems like BAR-B [60] and BAR Gossip [42] that $\mathcal{F}ire\mathcal{S}pam$ is a strict Nash equilibrium [61]. Towards this purpose, we need to demonstrate that each of the seven steps of the protocol (presented in Section 5.2) is a strict Nash equilibrium, meaning that a rational node cannot expect to increase its utility by deviating from the protocol. As done in BAR Gossip [42], we point out that in the rest of the discussion a rational node assuming other nodes to behave altruistically is a Nash equilibrium artifact proof.

We informally present here the incentive mechanisms provided in $\mathcal{F}ire\mathcal{S}pam$ to discourage rational nodes from deviating from the protocol, while instead we sketch a more formal proof in Appendix B.

(1) Incentives for message publication. A rational node r publishes a message m by sending it to all nodes in its publication view PV_r. In fact, as it is in r's interest to have its message m reliably delivered to all nodes of the system, and for what discussed in Section 5.3.1 on the required dissemination fanout to tolerate Byzantine nodes in the views, the rational node r will then follow the protocol and send the message m to all the nodes in its publication view PV_r.

(2) Incentives for message forwarding. Upon the reception of a message m which is not classified as *spam*, a rational node r always forwards m to all the nodes in its forwarding view FV_r. In fact, r forwards m, otherwise it risks:

- To be blacklisted by a node p in FV_r to which r did not forward m. Specifically, if p receives a message from another node than r and classifies it as $good$, it will detect r's misbehavior and blacklist r.

- To be blacklisted and suggested for eviction by a node p among its node monitors set M_r. Specifically, if p receives m and classifies it as $good$, it will expect the reception of a majority of REPORT messages from nodes in r's forwarding view. If p does not receive such reports, it will blacklist r and suggests to evict it. A majority of node monitors suggesting for eviction of r will then cause r removal from the system.

(3) Incentives for message reporting. Upon reception of a message m from a node p, a rational node r will always send a REPORT message to all the node monitors of p about the reception of m. In fact, p periodically checks on its node monitors and for each message it has forwarded, that all the nodes in its forwarding view sent the corresponding REPORT message. The node monitors reply to the sender with a *fixed size* response that contains the signed evidence of the reception of those REPORT messages. If r did not send a REPORT message, p will detect it and will consequently blacklist r.

(4) Incentives for correct node evaluation. At the heart of correct node evaluation is the MAJORITYREPORT dissemination performed by node monitors. Once a rational node monitor r has collected a majority of REPORT messages about a monitored node p for a given message m, the node monitor r will disseminate this information in the system. In fact, all p's node monitors (including r) have collected an equivalent majority of REPORT messages. Thus, as the dissemination in the node monitors overlay is reliable despite the presence of an upper bound number of Byzantine nodes (as previously discussed in Section 5.3.1), the other node monitors can observe whether r has actually disseminated the report or not. In this latter case, r will be blacklisted.

(5) Incentives for monitor collaboration. A rational node monitor r of a node p always sends correct messages to other node monitors of the same node. In fact, each time a node monitor sends a message to other node monitors, it expects to receive the same message from other node monitors (e.g., MAJORITYREPORT messages). Thus, if a node monitor does not receive an expected message from r, it collaborates with the other node monitors of p in order to evict r from the network.

(6) Incentives for correct view assignment. Similarly to the correct evaluation of nodes, biased view assignment can be very easily detected by node monitors as this process is deterministic. Hence, a node monitor that computes and disseminates a wrong forwarding or publication view risks eviction by other node monitors of the same monitored node.

(7) Incentives for report checking and answering to report checks. A rational node r periodically checks on its node monitors the existence of REPORT messages sent by nodes in r's forwarding view. If r does not check for the reception of those reports, its evaluation may be biased (e.g., by Byzantine nodes) and it also risks blacklisting and eviction. Hence, report checking allows a node to ensure that its node monitors are aware of its forwarding activity and discourage neighbors from not sending reports. Furthermore, a rational node monitor r, once queried by a monitored node p about

reports sent by one of its neighbors, always replies and with the correct answer. In fact, if r does not reply, p will blacklist it. Then, the incentive to provide the correct answer is given by the following two mechanisms. First, the response must be of a given fixed size, otherwise r will be blacklisted by p. Second, as the monitor node r and the monitored node p can not be in each other's forwarding view, then r has no interest in not providing the correct answer to p.

5.4 Evaluation

We performed a simulation-based evaluation of $\mathcal{F}ire\mathcal{S}pam$. To that purpose, we developed a C++ event-based simulator à la PeerSim [62]. We simulated a system composed of 1,000 nodes for three different filtering capability distributions: uniform, power-law, inverse of power-law. Moreover, we varied the percentage of Byzantine nodes that are in the system and that $\mathcal{F}ire\mathcal{S}pam$ is configured to tolerate: 1%, 3%, 5%.

In all experiments, the candidate sets from which to select the nodes for views has a size which is 5 times the number of Byzantine nodes to tolerate in the system. From what already said in Section 5.3.1, this guarantees that in a system with at most 5% Byzantine nodes, selecting 13 nodes in each view will ensure with probability 99.29% that a majority of them are non-Byzantine nodes. From similar reasoning, as discussed in Section 5.3.1, choosing a number of node monitors per node made by 9 nodes, it guarantees with probability 99.99% that each node has a majority of non-Byzantine node monitors when at most 5% of the nodes of the system are Byzantine.

We compare $\mathcal{F}ire\mathcal{S}pam$ against the gossip based dissemination protocol presented in [30], and that —as previously done by [90]— we refer to as *RandCast* in the following. In *RandCast* each node forwards the messages it receives to a set of nodes that continuously changes and that represents a random sample of the network. It has been proved that this set must have a size of $log(N) + c$ (where N is the size of the network and c is a constant) to ensure reliable delivery of messages [30]. In our experiments, this size is set to 13 to mimic the behavior of $\mathcal{F}ire\mathcal{S}pam$, in which each node forwards a message to *every* node in its view. Moreover, as we have described and studied in Section 4.2.2, we have further modified the *RandCast* protocol to take into account spam filtering capabilities of nodes, so that they do not forward the messages they detect as spam.

We first assess the correctness of the forwarding views that $\mathcal{F}ire\mathcal{S}pam$ assigns to nodes. Our experiments show that each node has in its forwarding view a set of nodes actually selected from the candidate set of c nodes in the surroundings of its position in the ladder, as explained in Section 5.3.1. We then show that albeit both $\mathcal{F}ire\mathcal{S}pam$ and *RandCast* ensure reliable dissemination of good messages, $\mathcal{F}ire\mathcal{S}pam$ presents an increased latency due to its ladder topology, and within it, depending on the percentage of Byzantine nodes to tolerate. Afterwards, we show the evaluation of the average percentage of spam messages received by each node as a function of the distribution of filtering capabilities and of the percentage of Byzantine nodes in the system. This evaluation shows that in all cases $\mathcal{F}ire\mathcal{S}pam$ drastically reduces the percentage of spam messages received by the nodes that are in the upper part of the ladder. It also shows that the higher the filtering capability of a node, the lower the

percentage of spam messages it receives. We then assess the behavior of $\mathcal{F}ire\mathcal{S}pam$ under an eclipse attack [109], i.e. when a set of Byzantine nodes collude to break the ladder topology. This experiment shows that under an eclipse attack, $\mathcal{F}ire\mathcal{S}pam$ successfully maintains the ladder topology, ensures reliable dissemination of good messages, and keeps filtering spam messages in an efficient way. Finally, we assess the cost of $\mathcal{F}ire\mathcal{S}pam$ in terms of bandwidth consumption with respect to *RandCast*.

We point out that, in our simulations, Byzantine nodes do not forward good messages, do not filter spam messages, send randomly generated reports, and take random decisions when they act as node monitors. To measure their impact, blacklisting and eviction has been disabled in all the evaluation contexts, except for the evaluation of Byzantine nodes colluding together and attempting to break the ladder.

5.4.1 Correctness of forwarding views

Figure 5.5 illustrates the forwarding views assigned to nodes by $\mathcal{F}ire\mathcal{S}pam$, configured to tolerate an upper bound of Byzantine nodes respectively comprising 1%, 3% and 5% of the nodes of system. We observed very similar results for the three spam filtering distributions, we thus only report results for the uniform distribution. The X and Y axes both represent nodes, ordered by filtering capability. The forwarding view of a given node on the X axis is made by the set of nodes on the Y axis for which a point is plotted in the graph.

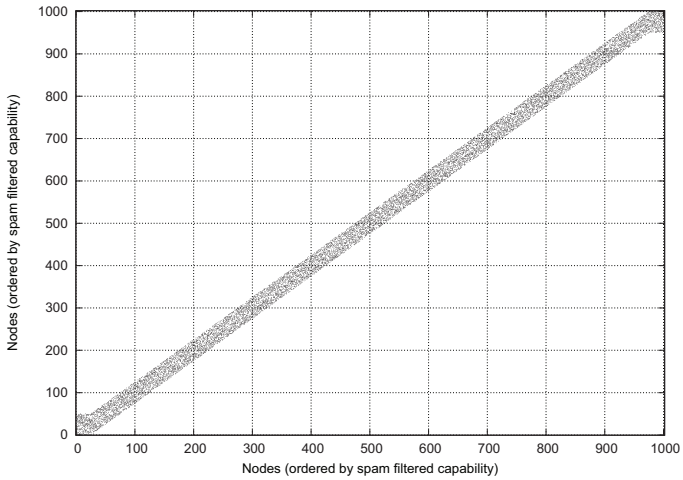
As explained before, each forwarding view is made of 13 nodes that are selected from a candidate set which is 5 times the number of Byzantine nodes to tolerate in the system. This means that the candidate set for a given node p is made of, respectively, the 50, 150, and 150 nodes surrounding p in the ladder, when $\mathcal{F}ire\mathcal{S}pam$ is configured to tolerate an upper bound of, respectively, 1%, 3%, and 5% of the nodes of the system being Byzantine. We indeed observe that $\mathcal{F}ire\mathcal{S}pam$ correctly assigns forwarding views: every node has a forwarding view that comprises 13 nodes that are randomly chosen from the candidate set of nodes surrounding it in the ladder.

5.4.2 Reliability and latency of good messages delivery

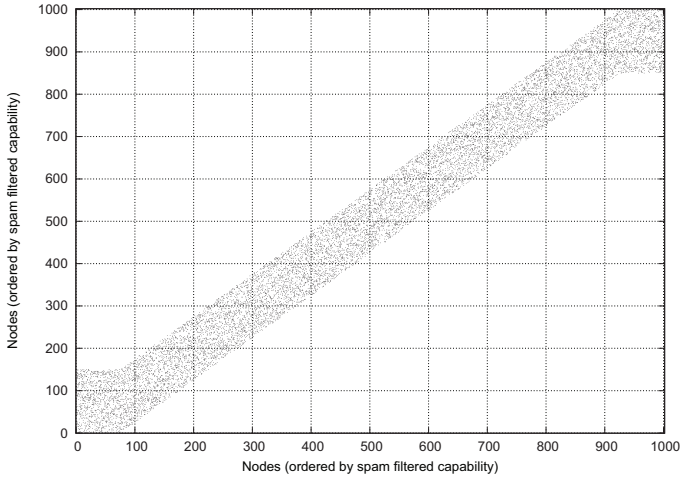
Figure 5.6 compares *RandCast* and $\mathcal{F}ire\mathcal{S}pam$ with respect to the average percentage of nodes reached by a good message as a function of the number of hops from the publisher node. Again, for $\mathcal{F}ire\mathcal{S}pam$ we show the results obtained in the presence respectively of 1%, 3%, and 5% of the nodes of the system being Byzantine (i.e., not forwarding good messages). The observed results were very similar for the three spam filtering distributions, as a consequence of the fact that they all lead to similar forwarding view assignments, as noted in the previous section. We thus only show the latency results for the uniform distribution.

We observe that both protocols achieve 100% reliability in delivery. Nevertheless, we observe that the average latency (expressed in the number of hops) to reach all nodes is higher for $\mathcal{F}ire\mathcal{S}pam$ than for *RandCast*. This comes from the usage of the ladder, in which messages are progressively disseminated from the bottom to the top. Moreover, we observe that the latency decreases with the number of Byzantine nodes that are tolerated by $\mathcal{F}ire\mathcal{S}pam$. This comes from the fact that this decreases the size

(a) 1% Byzantine nodes



(b) 3% Byzantine nodes



(c) 5% Byzantine nodes

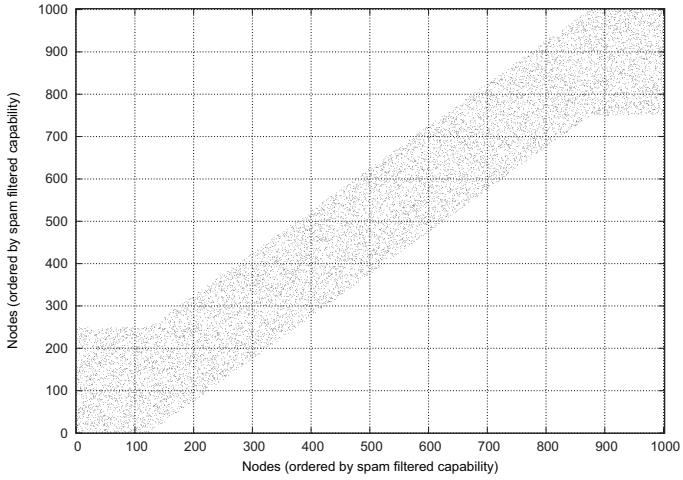


Figure 5.5 – Forwarding views assigned by $\mathcal{F}ire\mathcal{S}pam$ as a function of the node filtering capability for three different percentage of Byzantine nodes.

of the candidate set from which forwarding views are assigned, as shown previously in Figure 5.5. As a consequence, the height of the ladder increases, and thus the time it takes to disseminate messages.

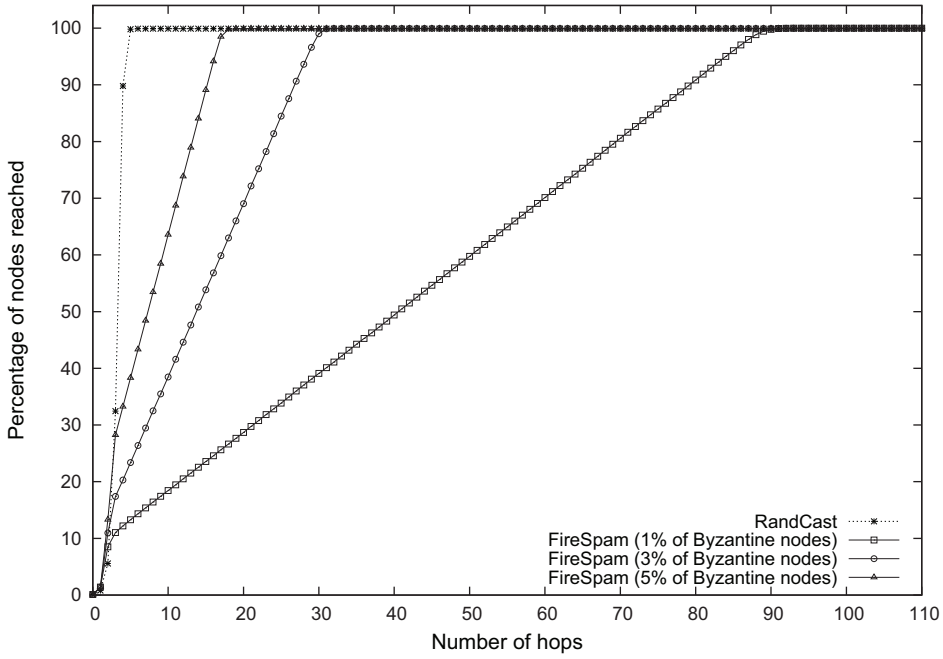


Figure 5.6 – Cumulative distribution of the average percentage of nodes reached by a good message, after a given number of hops from the publisher node.

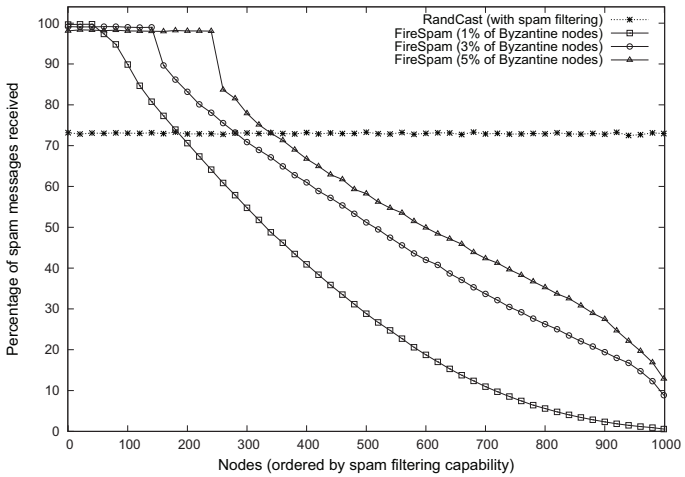
5.4.3 Percentage of spam messages received

Figure 5.7 shows the percentage of spam messages that are received by every node. The X axis represents nodes, ordered by filtering capability. Nodes on the left (resp., right) of the X axis are thus located at the bottom (resp., top) of the ladder. We plot results for *RandCast* and $\mathcal{F}ire\mathcal{S}pam$ for the three spam filtering capability distributions. Regarding $\mathcal{F}ire\mathcal{S}pam$, we plot results obtained when the system is made of, and $\mathcal{F}ire\mathcal{S}pam$ is configured to tolerate, respectively 1%, 3% and 5% of the nodes being Byzantine.

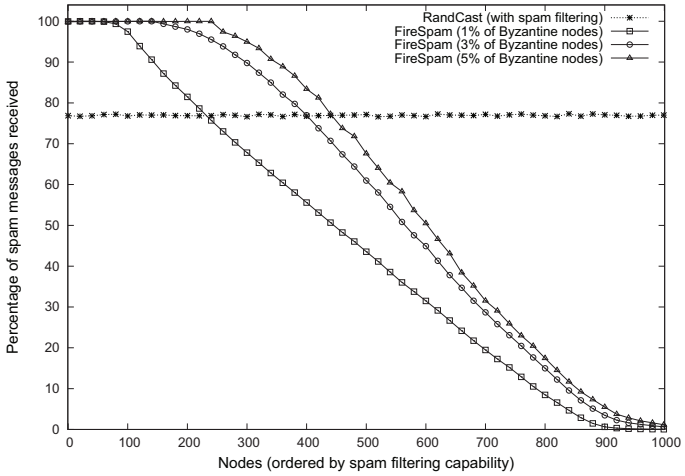
We can first observe that, when using *RandCast*, the percentage of spam messages that are received is the same for all nodes. This comes from the fact that each node has the same probability to communicate with all nodes in the network. Thus, the benefit of spam filtering is uniformly spread among nodes. Not surprisingly, we also observe that the percentage of spam messages that are received is much lower when using the inverse power-law distribution (38%) than when using other distributions (around 75%). This comes from the fact that in the former case, many nodes have a very high filtering capability.

The second observation we can make is that using $\mathcal{F}ire\mathcal{S}pam$, nodes receive a percentage of spam messages that is conversely proportional to their spam filtering

(a) Uniform distribution



(b) Power-law distribution



(c) Inverse power-law distribution

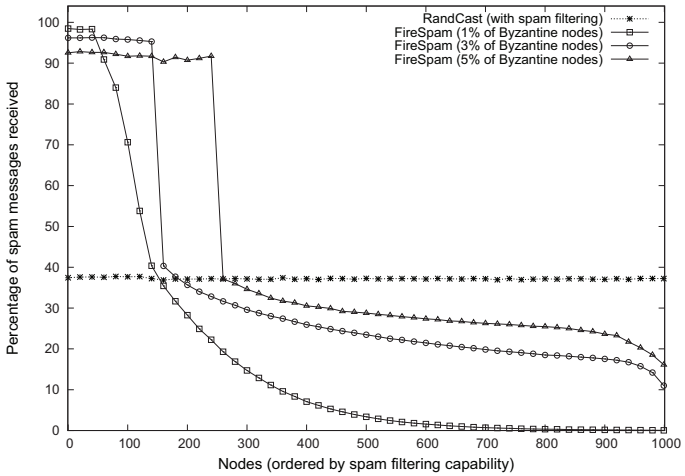


Figure 5.7 – Percentage of spam messages *received* by nodes for the three different spam filtering capability distributions: (a) uniform, (b) a power-law, (c) inverse power-law.

capability. We do thus see that nodes have a clear incentive to filter spam messages, as this will consequently decrease the percentage of spam messages they receive. For instance, in all three distributions, the very best nodes receive at least 5 times less spam messages than the worst nodes.

We also observe that nodes that are at the bottom of the ladder receive more spam than using *RandCast*, but also much more spam than nodes that follow them in the ladder. This is explained by the fact that these nodes receive all messages that are generated in the system because they are in the publication views of nodes. Once they have received messages, the filtering process starts and nodes located higher in the ladder receive fewer spam.

Finally, we can also observe that in all three distributions, increasing the number of Byzantine nodes that can be tolerated decreases the overall spam filtering efficiency. This is explained by the fact that the candidate sets used to assign forwarding views become bigger, as previously shown in Figure 5.5. This in fact results in a faster dissemination from the bottom to the top of the ladder topology, thus reducing the probability for spam messages to be filtered while being disseminated using the ladder.

5.4.4 Behavior under an eclipse attack

In this section, we study the behavior of $\mathcal{F}ire\mathcal{S}pam$ when 5% of the nodes are Byzantine and collude with the aim of harming the ladder. We consider a uniform distribution of the spam filtering capabilities. In order to harm the ladder, Byzantine nodes behave as follows: during a long enough period of time, they all behave similarly, filtrating and forwarding the same messages. Consequently, they end up at the same location in the ladder (in the middle in the presented experiment). Once they have all reached the middle of the ladder, they all start the attack simultaneously. During the attack, they do not forward good messages, and they do not filter spam messages. Figure 5.8 shows the percentage of spam messages that are actually received by every node in the system. Nodes are depicted in the X axis, ordered by spam filtering capability. We plot three different lines: before the attack (T0), during the attack (T1), and after the attack (T2).

Before the attack (line T0), the percentage of spam messages that are received is the same as the one depicted in Figure 5.7 for the uniform distribution and 5% of Byzantine nodes. During the attack (line T1), we observe that the percentage of spam messages that are received by nodes in the upper part of the ladder (i.e. on the right of the X axis) slightly increases. This is due to the fact that the 50 colluding nodes no longer filter spam messages. These colluding nodes are located between $x = 475$ and $x = 525$, hence the increase of the percentage of spam received by nodes located above the colluding nodes in the ladder (i.e. nodes located on the right of $x = 525$). Finally, we observe that some time after the attack (line T2), colluding nodes have been evicted from the network. The percentage of spam messages received by nodes in the upper part of the ladder is thus very close to the one that was observed before the attack (line T0). The slight difference comes from the fact that before the attack, colluding nodes were participating in the filtering of spam messages.

Moreover, we assessed the reliability of good messages delivery before, during

and after the attack. Our results show that $\mathcal{F}ire\mathcal{S}pam$ was able to reliably deliver good messages to all nodes during all the experiment, even when under attack by the colluding nodes.

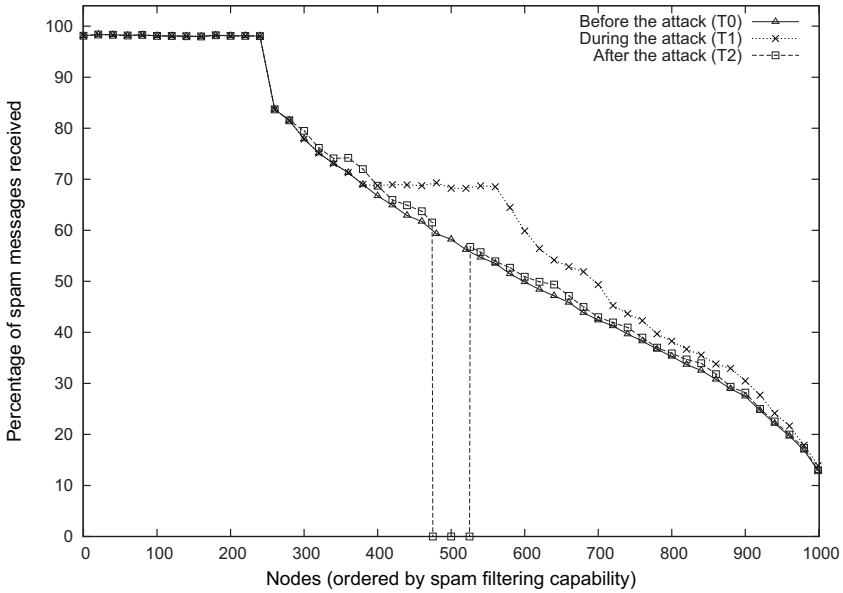


Figure 5.8 – Percentage of spam messages that are received by nodes before the attack, during the attack, and after the attack (uniform distribution of the spam filtering capabilities).

5.4.5 Bandwidth consumption

We evaluated the bandwidth consumption of $\mathcal{F}ire\mathcal{S}pam$ by assessing its overhead in message dissemination with respect to the *RandCast* protocol which we used as a reference. We have simulated a system with 1,000 nodes, with $\mathcal{F}ire\mathcal{S}pam$ configured to tolerate 5% of Byzantine nodes. In the experiment, we simulated the scenario of an Internet discussion group where a uniformly selected participant creates a new discussion topic, or answers to an existing one, on average every 5 minutes. Furthermore, we assumed that such created user messages carry an average content (i.e., the text being published) of size 4 KB.

We observe that message dissemination alone (i.e., *RandCast*), requires an upload bandwidth cost of 180 B/s. Adding to that the REPORT and MAJORITYREPORT messages dissemination, the upload bandwidth cost reaches 6.574 KB/s. A large fraction of the generated traffic in $\mathcal{F}ire\mathcal{S}pam$ is in fact due to fault tolerance, in particular to disseminate the MAJORITYREPORT messages. For what said in the protocol description, this is in fact required to properly asses the filtering capability of all nodes of the system, by ordering them by the number of forwarded messages. The correct node evaluation is indeed ensured by the fact that all (non-Byzantine) monitors of a monitored node disseminates a MAJORITYREPORT for each single message sent by one of their monitored nodes. Furthermore, this MAJORITYREPORT created by a single

monitor is reliably broadcast in all the system using the Fireflies gossiping service on top of the multiple rings overlay. Note that our design choices in $\mathcal{F}ire\mathcal{S}pam$ are dictated by favoring robustness against rational and byzantine behaviors, while performance has been a secondary concern. A number of optimizations can be investigated to reduce the overhead generated by $\mathcal{F}ire\mathcal{S}pam$.

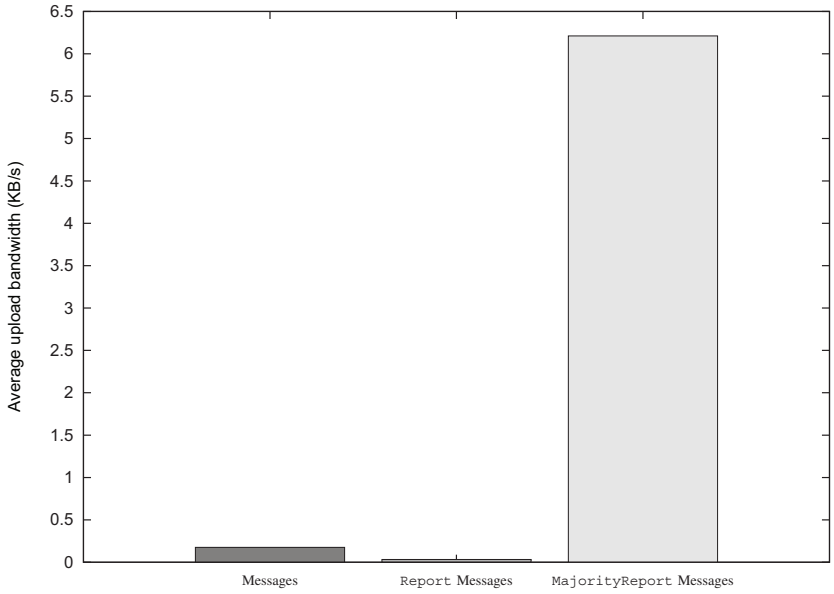


Figure 5.9 – $\mathcal{F}ire\mathcal{S}pam$: upload bandwidth consumption.

5.5 Conclusion

Gossip-based dissemination protocols are known to allow reliable and fast delivery of information in large-scale networks. Nevertheless, they cannot limit the amount of spam. We have designed and evaluated $\mathcal{F}ire\mathcal{S}pam$, a novel gossiping protocol able to limit spam dissemination. $\mathcal{F}ire\mathcal{S}pam$ organizes nodes in a ladder topology: nodes with low spam filtering capability are at the bottom of the ladder, whereas nodes with a high filtering capability are at the top. $\mathcal{F}ire\mathcal{S}pam$ has been designed in the BAR model: it can thus tolerate a bounded number of Byzantine nodes, and it provides incentive-compatible mechanisms to discourage rational nodes to deviate from the protocol. We have extensively evaluated $\mathcal{F}ire\mathcal{S}pam$ using simulations. Our results show that it allows drastically to limit the dissemination of spam, without hurting the reliability of good message delivery.

6

Conclusion

This chapter concludes the document by providing first a summary of the contributions of this thesis, and then illustrating some possible open research directions.

6.1 Contributions of this thesis

In this thesis, we have tackled practical problems that gossip protocols face when deployed on the Internet. In particular, we focused of the following two axes: NAT-resilient gossip peer sampling, and spam-resilient gossiping in the BAR model.

NAT-resilient Gossip Peer Sampling. In gossip-based peer sampling, each node maintains a partial local view of the system and periodically exchanges a subset of its view with another node, selected from its view. This periodic exchange ensures that the local views of nodes represent a continuous uniform random sample of all the nodes in the system [27]. Nevertheless, gossip-based protocols assume that any node is able to communicate with any node picked from its view. This is not the case in practical deployment scenarios as the Internet, where a large fraction of nodes may very well be sitting behind a Network Address Translator (NAT). NATs implement firewall-like mechanisms which drop unsolicited incoming packets if no prior message was sent from behind the NAT. Consequently, the presence of NAT prevents direct communication among nodes. In Chapter 2, taking the case of the generic gossip-based peer sampling framework [27], we have studied how the mere presence of NATs severely impacts the properties of gossip-based peer sampling. The randomness of the returned sample and the connectivity of the overlay network are affected by the presence of NATs: nodes end up having many stale references in their view, and the graph of nodes gets partitioned because of many unreachable references. In Chapter 3 we have presented $\mathcal{N}ylon$ [55], a novel NAT-resilient gossip peer sampling protocol. $\mathcal{N}ylon$ is built on the gossip-based peer sampling framework [27] and it is based on a decentralized hole punching [56, 57] approach. The approach allows to traverse NATs by establishing a path of relay nodes to initiate the communication towards natted nodes. We have extensively evaluated

$\mathcal{N}ylon$ showing that: *(i)* it ensures the properties of the peer sampling service; *(ii)* it fairly balances the communication costs on the nodes (whether they are natted or not); *(iii)* it is highly robust to churn.

Spam-resilient Gossiping in the BAR model. In a gossip-based dissemination protocol, once a node receives a new message, it forwards it to a random subset of its neighbors. The advantages of such a scheme are its simplicity and reliability, ensured by the randomness and redundancy, where nodes possibly receive the same message multiple times [30]. Such protocols can be used to implement effectively Internet collaborative services (e.g., a Q&A web site or a forum) in a distributed manner, where participant nodes contribute to the information dissemination. Nevertheless, in practical deployment scenarios as the Internet, nodes may deviate from the guidelines of the protocol and from those of the collaborative service itself. In fact, nodes might publish spam messages which are of no interest for the participants. In Chapter 4 we have discussed how the characteristics of gossip-based dissemination protocols, namely randomness and redundancy, make them ideal vectors to disseminate spam messages to a large fraction of nodes. To design a spam-resilient gossip-based dissemination protocol, special care must be given to the actual behavior that nodes may exhibit when participating to a collaborative service, where no central authority controls what they do. We considered the BAR model [60] as a practical fault tolerance model to cope with the possible behaviors of nodes. Accordingly to the BAR model in fact, together with (*altruistic*) nodes who follow the protocol, there might be (*Byzantine*) nodes deviating for arbitrary reasons (due to a bug or possibly attempting to harm other nodes), or even (*rational*) nodes that seek maximizing their net benefit in the participation (e.g., by not forwarding messages they should have to). In Chapter 5 we have presented $\mathcal{F}ire\mathcal{S}pam$ [59], a novel spam-resilient gossip protocol designed in the BAR model and accounting for the three aforementioned classes of behavior. $\mathcal{F}ire\mathcal{S}pam$ encompasses a set of mechanisms ensuring that Byzantine nodes are detected and evicted from the system. These mechanisms assume a maximum number of Byzantine nodes. Furthermore, $\mathcal{F}ire\mathcal{S}pam$ encompasses a set of incentive-compatible mechanisms that make the protocol a strict Nash equilibrium [61]. These incentive mechanisms ensure that it is in a rational node's best interest to always follow the protocol.

6.2 Future directions

The BAR (Byzantine-Altruistic-Rational) model [60] is of valuable importance to characterize the behavior of participant nodes in Internet collaborative services. With respect to designing and building practical distributed systems in the BAR model, we define the following axes for a possible future research investigation.

General purpose and scalable P2P BAR Framework. Almost all the P2P systems designed in the BAR model are suited to the specific application scenario they address: BAR Gossip [42] and FlightPath [105] for one-to-many live-streaming, FireSpam [59] for many-to-many spam-resilient information dissemination. The only work providing a more generic framework is the 3-levels architecture employed by BAR-B [60]. In

this framework, an incentive-compatible replicated state machine is implemented on top of all nodes, so that it abstracts what the authors call a trusted altruistic "witness node", enforcing each non-Byzantine node to follow the protocol, and ensuring that any misbehavior can be detected. Nevertheless, this solution does not scale over a few hundreds nodes, and has the limitations we have examined in Section 4.3.3. Thus, the following natural question arises:

> *Can we design a scalable P2P BAR framework to build BAR tolerant versions of existing (e.g., collaborative backup, broadcasting, ..) or novel application-level protocols?*

In particular, we wonder if it is possible to abstract generic API from the stack $\mathcal{F}ire\mathcal{S}pam$ on top of Fireflies, and build a general purpose Incentive-Compatible Byzantine Fault-Tolerant P2P framework.

BAR model and social links. In all systems designed so far in the BAR model [42, 59, 60, 105], all nodes are classified accordingly to these three classes of behavior: Byzantine, Altruistic, Rational behavior. This classification accounts for the behavior of nodes taken individually. A protocol designed in the BAR model tolerates a maximum number of misbehaving or malicious Byzantine nodes, and provides incentive mechanisms so that all the remaining non-Byzantine nodes do not have an interest in rationally deviating from the protocol to gain more benefit. In the BAR model, rational nodes do not collude together, only Byzantine nodes possibly can. Nevertheless, in the age of online social networks, the graph of nodes and the interactions among them are usually driven by existing social relationships among the participants. This typically means that a node's behavior might legitimately be different with respect to the node it interacts with. The following natural question arises:

> *How should the BAR model account for social links among nodes?*

In particular, we wonder if it is possible to simplify/adapt the incentive mechanisms and the assumptions used by BAR protocols when modeling interactions between "friends" or, transitively, "friends or friends".

BAR model and anonymous communications. We have discussed how, in the BAR model [60], rational nodes will follow the protocol as long as there is a "credible threat" for them to be caught while misbehaving, and consequently punished for their actions. Nevertheless, existing BAR protocols [42, 59, 60, 105] rely on the fact that nodes are accountable [104] for what they do. Yet, recent systems like Gossple [110] have highlighted how, in the age of large scale social networks, users could benefit from the information contained in what the authors call "network of anonymous social acquaintances". Put simply, users could increase the quality of their search results by querying the information contained in the profiles of users who area *not* necessarily their friends in the real life, but who nevertheless share similar interests in the social network. To implement such a communication protocol, Gossple leverages anonymous

communication exchanges à la onion routing [111]. In onion routing, nodes exchange messages through a sequence of hops which ensure that no node can tell that two nodes are indeed exchanging data. The following natural question arises:

> *How can a protocol relying on anonymous communications be designed in the BAR model?*

In particular, a recent work (Dissent [112]) shows that it is possible to ensure both anonymity and accountability in a system, and we wonder how the two properties could be ensured in a large-scale communication protocol designed in the BAR model.

Bibliography

[1] "Gnutella protocol specification v0.4," www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf.

[2] Cisco, "Cisco Visual Networking Index: Usage Study," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/Cisco_VNI_Usage_WP.html, October 2010.

[3] YouTube, http://www.youtube.com.

[4] Vimeo, http://www.vimeo.com.

[5] B. Cohen, "Incentives Build Robustness in BitTorrent," in *First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[6] ICQ, http://www.icq.com.

[7] Jabber, http://www.jabber.org.

[8] Skype, http://www.skype.com.

[9] Wuala, http://www.wuala.com.

[10] SopCast, http://www.sopcast.com.

[11] PPLive, http://pplive.allp2ptv.org.

[12] Veetle, http://www.veetle.com.

[13] Diaspora, http://www.joindiaspora.com.

[14] R. Rodrigues and P. Druschel, "Peer-to-Peer Systems," *Communications of the ACM*, vol. 53, no. 10, pp. 72–82, October 2010.

[15] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, February 2003.

[16] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the International Middleware Conference (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[17] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *Proceedings of the Internal USENIX Annual Technical Conference (ATEC)*, Boston, MA, USA, June 2004, pp. 127–140.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the International Special Interest Group on Data Communication (SIGCOMM)*, San Diego, CA, United States, August 2001, pp. 161–172.

[19] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, March 2002, pp. 53–65.

[20] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proceedings of the International Symposium on Theory of Computing (STOC)*, El Paso, TX, USA, May 1997, pp. 654–663.

[21] M. F. Kaashoek and I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proceedings of the International Workshop Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, USA, February 2003, pp. 33–44.

[22] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service," in *Proceedings of the International Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001, pp. 71–76.

[23] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility," in *Proceedings of the International Symposium on Operating System Principles (SOSP)*, Banff, AB, Canada, October 2001, pp. 188–201.

[24] J. W. Mickens and B. D. Noble, "Exploiting Availability Prediction in Distributed Systems," in *Proceedings of the International Conference on Networked Systems Design & Implementation (NSDI)*, San Jose, CA, USA, May 2006, pp. 73–86.

[25] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Transactions on Computer Systems*, vol. 21, no. 4, pp. 341–374, November 2003.

[26] S. Voulgaris, D. Gavidia, and M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[27] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based Peer Sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, August 2007.

[28] B. Bollobas, *Random Graphs*. Cambridge University Press, 2001.

[29] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Database Maintenance," in *Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, Vancouver, BC, Canada, August 1987, pp. 1–12.

[30] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh, "Probabilistic Reliable Dissemination in Large-Scale Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 248–258, March 2003.

[31] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized Rumor Spreading," in *Proceedings of the International Symposium on Foundations of Computer Science (FOCS)*, Redondo Beach, CA, USA, November 2000, pp. 565–574.

[32] D. C. Oppen and Y. K. Dalal, "The clearinghouse: a decentralized agent for locating named objects in a distributed environment," *ACM Transactions on Information Systems*, vol. 1, no. 3, pp. 230–253, July 1983.

[33] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal Multicast," *ACM Transactions on Computer Systems*, vol. 17, no. 2, pp. 41–88, May 1999.

[34] R. van Renesse, Y. Minsky, and M. Hayden, "A Gossip-style Failure Detection Service," in *Proceedings of the International Middleware Conference (Middleware)*, The Lake District, UK, September 1998, pp. 55–70.

[35] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based Computation of Aggregate Information," in *Proceedings of the International Symposium on Foundations of Computer Science (FOCS)*, Cambridge, MA, USA, October 2003, pp. 482–491.

[36] M. Jelasity, A. Montresor, and Ö. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, August 2005.

[37] M. Jelasity and A.-M. Kermarrec, "Ordered Slicing of Very Large-Scale Overlay Networks," in *Proceedings of the International Conference on Peer-to-Peer Computing (P2P)*, Cambridge, UK, October 2006, pp. 117–124.

[38] A. Fernandez, V. Gramoli, E. Jimenez, A.-M. Kermarrec, and M. Raynal, "Distributed Slicing in Dynamic Systems," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Toronto, ON, Canada, June 2007.

[39] V. Gramoli, Y. Vigfusson, K. Birman, A.-M. Kermarrec, and R. van Renesse, "Slicing Distributed Systems," *ACM Transactions on Computer Systems*, vol. 58, no. 11, pp. 1444–1455, November 2009.

[40] M. Jelasity, A. Montresor, and Ö. Babaoglu, "T-Man: Gossip-based Fast Overlay Topology Construction," *Computer Networks*, vol. 53, no. 13, pp. 2321–2339, August 2009.

[41] S. Voulgaris and M. van Steen, "Epidemic-Style Management of Semantic Overlays for Content-Based Searching," in *Proceedings of the International Euro-Par Parallel Processing Conference*, Lisbon, Portugal, August 2005, pp. 1143–1152.

[42] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "BAR Gossip," in *Proceedings of the International Symposium on Operating*

Systems Design and Implementation (OSDI), Seattle, WA, USA, November 2006, pp. 191–204.

[43] D. Frey, R. Guerraoui, A.-M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma, "Heterogeneous Gossip," in *Proceedings of the International Middleware Conference (Middleware)*, Urbana, IL, USA, November 2009, pp. 42–61.

[44] A.-M. Kermarrec and M. van Steen, "Gossiping in Distributed Systems," *Operating Systems Review*, vol. 41, no. 5, pp. 2–7, October 2007.

[45] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations," in *Proceedings of the International Middleware Conference (Middleware)*, Toronto, ON, Canada, October 2004, pp. 79–98.

[46] B. Pittel, "On Spreading a Rumor," *SIM Journal on Applied Mathematics*, vol. 47, no. 1, pp. 213–223, March 1987.

[47] M. Jelasity, W. Kowalczyk, and M. Van Steen, "Newscast Computing," Vrije Universiteit Amsterdam, Tech. Rep., November 2003.

[48] Kermarrec, A.M. and Leroy, V. and Thraves, C., "Ensuring Uniformity in Random Peer Sampling Services," INRIA / INSA , Tech. Rep., April 2010.

[49] M. Casado and M. J. Freedman, "Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification," in *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*, Cambridge, MA, USA, April 2007.

[50] G. Halkes and J. Pouwelse, "UDP NAT and Firewall Puncturing in the Wild," TUDelft, Tech. Rep., 2010.

[51] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the New Coolstreaming: Principles, Measurements and Performance Implications," in *Proceedings of the International Conference on Computer Communications (INFOCOM)*, Phoenix, AZ, USA, April 2008, pp. 1031–1039.

[52] G. Maier, F. Schneider, and A. Feldmann, "NAT Usage in Residential Broadband Networks," in *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*, Atlanta, GA, USA, March 2011.

[53] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica, "Non-Transitive Connectivity and DHTs," in *Proceedings of the International Workshop on Real, Large Distributed Systems (WORLDS)*, San Francisco, CA, USA, December 2005, pp. 55–60.

[54] A. Ganguly, P. O. Boykin, D. I. Wolinsky, and R. J. Figueiredo, "Improving Peer Connectivity in Wide-Area Overlays of Virtual Workstations," in *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, Boston, MA, USA, June 2008, pp. 129–140.

[55] A.-M. Kermarrec, A. Pace, V. Quéma, and V. Schiavoni, "NAT-resilient Gossip Peer Sampling," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Montreal, QC, Canada, June 2009, pp. 360–367.

[56] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *Proceedings of the Internal USENIX Annual Technical Conference (ATEC)*, Anaheim, CA, USA, April 2005.

[57] P. Srisuresh and B. Ford, "State of Peer-to-Peer (P2P) Communication Across Across Network Address Translators (NATs)," RFC 5128, March 2008.

[58] U. Lee, M. Choi, J. Cho, M. Y. Sanadidi, and M. Gerla, "Understanding Pollution Dynamics in P2P File Sharing," in *In Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, USA, February 2006.

[59] S. Ben Mokhtar, A. Pace, and V. Quéma, "FireSpam: Spam Resilient Gossiping in the BAR Model," in *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)*, New Delhi, India, October 2010, pp. 225–234.

[60] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "BAR Fault Tolerance for Cooperative Services," in *Proceedings of the International Symposium on Operating Systems Principles (SOSP)*, Brighton, UK, October 2005, pp. 45–58.

[61] J. Nash, "Non-Cooperative Games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, September 1951.

[62] A. Montresor and M. Jelasity, "PeerSim: A Scalable P2P Simulator," in *Proceedings of the International Conference on Peer-to-Peer (P2P)*, Seattle, WA, September 2009, pp. 99–100.

[63] Z. Bar-Yossef, R. Friedman, and G. Kliot, "RaWMS - Random Walk Based Lightweight Membership Service for Wireless Ad Hoc Networks," *ACM Transactions on Computer Systems*, vol. 26, no. 2, June 2008.

[64] V. King and J. Saia, "Choosing a Random Peer," in *Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, St. John's, NL, Canada, July 2004, pp. 125–130.

[65] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer Counting and Sampling in Overlay Networks: Random Walk Methods," in *Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, Denver, CO, USA, July 2006, pp. 123–132.

[66] D. Aldous and J. Fill, "Reversible markov chains and random walks on graphs," Draft available at: http://stat-www.berkeley.edu/users/aldous/RWG/book.html.

[67] E. L. Merrer, A.-M. Kermarrec, and L. Massoulié, "Peer to peer size estimation in large and dynamic networks: A comparative study," in *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, Paris, France, June 2006, pp. 7–17.

[68] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, "Estimating Aggregates on a Peer-to-Peer Network," Computer Science Department, Stanford University, Tech. Rep., 2003.

[69] M. Klamkin and D. Newman, "Extensions of the birthday surprise," *Journal of Combinatorial Theory*, vol. 3, no. 3, pp. 279–282, 1967.

[70] P. Erdos and A. Renyi, "On random graphs I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[71] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, January 2001.

[72] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs)," RFC 3489, March 2003.

[73] N. Drost, E. Ogston, R. V. van Nieuwpoort, and H. E. Bal, "ARRG: Real-World Gossiping," in *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, Monterey, CA, USA, June 2007, pp. 147–158.

[74] J. a. Leitão, R. van Renesse, and L. Rodrigues, "Balancing Gossip Exchanges in Networks with Firewalls," in *Proceedings of the International Workshop on Peer-to-Peer systems (IPTPS)*, San Jose, CA, USA, April 2010.

[75] J. Maassen and H. E. Bal, "Smartsockets: Solving the Connectivity Problems in Grid Computing," in *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, Monterey, CA, USA, June 2007, pp. 1–10.

[76] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, pp. 164–206, May 2003.

[77] M.-J. Lin and K. Marzullo, "Directional Gossip: Gossip in a Wide Area Network," in *Proceedings of the European Dependable Computing Conference on Dependable Computing (EDCC)*, Prague, Czech Republic, September 1999, pp. 364–379.

[78] Skype, "Post-mortem on the Skype outage (Skype Blog)," http://blogs.skype.com/en/2010/12/skype_downtime_today.html, December 2010.

[79] M. Lee, H. Choi, and S. Park, "DONet-P: A Streaming Overlay Network Protocol with Private Network Support," in *Proceedings of the Region 10 Conference TENCON*, Taipei, Taiwan, November 2007.

[80] H. Yoshimi, N. Enomoto, Z. Cui, K. Takagi, and A. Iwata, "NAT Traversal Technology of Reducing Load on Relaying Server for P2P Connections," Las Vegas, NV, USA, January 2007.

[81] "Public stun servers," http://www.voip-info.org/wiki/view/STUN.

[82] B. Rekhtman, "UDP Hole Punch Timeout Discovery Algorithm Over Network Address Translation Connection," US Patent App. 12/402,153, March 2009.

[83] D. Watts and S. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[84] Google, "Google groups," http://groups.google.com.

[85] StackOverflow, http://www.stackoverflow.com.

[86] Quora, http://www.quora.com.

[87] D. Gavidia, G. Jesi, C. Gamage, and M. van Steen, "Canning Spam in Wireless Gossip Networks," in *Proceedings of the International Conference on Wireless on Demand Network Systems and Services (WONS)*, Oberguyrgl, Austria, January 2007, pp. 30–37.

[88] N. Bailey, *The mathematical theory of infectious diseases and its applications*. Hafner Press, 1975.

[89] P. Eugster, R. Guerraoui, A. Kermarrec, and L. Massoulié, "Epidemic Information Dissemination in Distributed Systems," *IEEE Computer*, vol. 37, no. 5, pp. 60–67, May 2004.

[90] S. Voulgaris and M. Van Steen, "Hybrid Dissemination: Adding Determinism to Probabilistic Multicasting in Large-Scale P2P Systems," in *Proceedings of the International Middleware Conference (Middleware)*, Newport Beach, CA, USA, November 2007, pp. 389–409.

[91] D. Gavidia and M. van Steen, "Enforcing Data Integrity in Very Large Ad Hoc Networks," in *Proceedings of the International Conference on Mobile Data Management*, Mannheim, Germany, May 2007, pp. 77–85.

[92] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, July 1982.

[93] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica, "Free-riding and whitewashing in peer-to-peer systems," in *Proceedings of the International Workshop on Practice and theory of incentives in networked systems (PINS)*, Portland, OR, USA, September 2004, pp. 228–236.

[94] E. Adar and B. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, pp. 2–13, 2000.

[95] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[96] M. Haridasan and R. van Renesse, "SecureStream: An intrusion-tolerant protocol for live-streaming dissemination," *Computer Communications*, vol. 31, no. 3, pp. 563–575, February 2008.

[97] H. Johansen, A. Allavena, and R. van Renesse, "Fireflies: scalable support for intrusion-tolerant network overlays," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Leuven, Belgium, April 2006, pp. 3–13.

[98] H. Johansen, "Intrusion Tolerant Membership Management for Peer-to-Peer Overlay Networks," Ph.D. dissertation, University of Tromsø, November 2007.

[99] C. Ho, R. van Renesse, M. Bickford, and D. Dolev, "Nysiad: practical protocol transformation to tolerate Byzantine failures," in *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, April 2008.

[100] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, December 1990.

[101] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, November 2002.

[102] J.-P. Martin, "Byzantine Fault Tolerance and Beyond," Ph.D. dissertation, UT Austin, December 2006.

[103] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "BAR fault tolerance for cooperative services - Extended Technical Report," UT Austin, Technical Report TR-05-10, October 2005.

[104] A. Yumerefendi and J. Chase, "The Role of Accountability in Dependable Distributed Systems," in *Proceedings of the International Workshop on Hot Topics in System Dependability (HotDep)*, June 2005.

[105] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin, "FlightPath: Obedience vs. Choice in Cooperative Services," in *Proceedings of the International Conference on Operating Systems Design and Implementation (OSDI)*, San Diego, CA, USA, December 2008, pp. 355–368.

[106] A. Jusang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, 2007.

[107] J. R. Douceur, "The Sybil Attack," in *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, March 2002, pp. 251–260.

[108] G. Badishi, I. Keidar, and A. Sasson, "Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 1, pp. 45–61, 2006.

[109] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending Against Eclipse Attacks on Overlay Networks," in *Proceedings of the ACM SIGOPS European Workshop (EW)*, Leuven, Belgium, September 2004, p. 21.

[110] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy, "The Gossple Anonymous Social Network," in *Proceedings of the International Middleware Conference (Middleware)*, Bangalore, India, November 2010, pp. 191–211.

[111] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Proxies For Anonymous Routing," in *Proceedings of the International Computer Security Applications Conference (ACSAC)*, San Diego, CA, USA, December 1996.

[112] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable Anonymous Group Messaging," in *Proceedings of the International Conference on Computer and Communications Security (CCS)*, Chicago, IL, USA, October 2010, pp. 340–350.

[113] G. Hardin, “The tragedy of the commons. The population problem has no technical solution; it requires a fundamental extension in morality.” *Science*, vol. 162, no. 859, pp. 1243–1248, 1968.

List of Figures

List of Tables

Making Fireflies IC-BFT

We have illustrated the Fireflies [97] Byzantine-tolerant membership protocol in Section 4.3.1.1. While the protocol has originally been shown to be Byzantine fault tolerant in (probabilistically) guaranteeing up to date view membership and reliable gossip dissemination, one might wonder if it is also Incentive-Compatible Byzantine Fault Tolerant (IC-BFT) in the BAR model [60]. This implies that not only the desired protocol properties (i.e., correct view membership and reliable dissemination in the Fireflies overlay) are guaranteed in the presence of a bounded number of Byzantine nodes (Byzantine Fault Tolerance), but also that it must be in the best interest of rational nodes to actually follow each step of the protocol (Incentive-Compatibility).

We question then here if Fireflies provides incentives for a rational node to actually follow each step of the protocol. That is, a rational node must have incentives to:

1. ping and issue an accusation for a successor if this does not reply to a ping;
2. reply to a ping by sending back a pong;
3. issue a rebuttal note for an accusation about itself;
4. forward a gossip message to other k random nodes taken from the view.

It is enough for a rational node to deviate from a single step of the protocol to make Fireflies not incentive-compatible. First, we discuss in Section A.1 why the protocol is not incentive-compatible. Then, in Section A.2, we illustrate the modifications to make the protocol incentive-compatible.

A.1 Is Fireflies incentive-compatible?

We consider individually each step of the protocol and we discuss whether or not a rational node has incentives to actually follow the protocol at that step. As said

previously, it is enough for a rational node to have incentives to deviate from a single step of the protocol to make Fireflies not incentive-compatible.

Question (1): Does a rational node r have incentives to ping a successor node s and issue an accusation for s if this is not responding to the ping?

An accusation for node s can only be issued relative to the most recent note that the node s has sent. Also, a note has an `enabled` bitmap which specifies which $t + 1$ among the $2t + 1$ rings are allowed to actually issue an accusation for that note, being otherwise the accusation not valid. As we have detailed in Section 4.3.1.1, the rings construction guarantees with very high probability that each node has (at least) $t + 1$ non-Byzantine predecessor nodes. This means that a Byzantine node s could exclude t out of the $t + 1$ non-Byzantine predecessors from issuing an accusation.

Let's consider the point of view of a rational node r which is the remaining non-Byzantine predecessor of s not excluded by the `enabled` bitmap. This rational node r has an incentive in incurring the (neglectable) costs of pinging the node s and that of issuing an accusation for s if it is not responding. In not doing so in fact, it risks to cause keeping a disconnected node in its and each other node's view. Because the very same views are the ones used to select nodes to gossip not only the accusations, but also the rebuttal notes, by keeping disconnected nodes in the views of the members, a rational node would risk impacting the reliability of the very same messages (i.e., the rebuttal notes) it might disseminate itself in the overlay. This turns out to be ultimately not in its best interest, hence a rational node is encouraged to follow the protocol.

$\Longrightarrow$ *Answer (1): YES.*

Question (2): Does a rational node r have incentives to reply to a ping from a predecessor node p situated on a ring enabled in the last note issued by r?

A rational node r has clearly an incentive in replying to a ping from a predecessor p which is on a ring which is enabled in the last note sent by r. In fact, if r does not reply to a ping (with a cheap pong message), it expects p to issue an accusation for it. At that point, being the accusation valid, the node r would be obliged to issue a new note message as a rebuttal for this accusation, otherwise it would be considered no more alive. This would go against the assumption that a rational node seeks long term benefit by its permanence in the system.

$\Longrightarrow$ *Answer (2): YES.*

Question (3): Does a rational node r have incentives to issue a rebuttal note for a valid accusation about itself?

A rational node r has clearly an incentive in sending a rebuttal note about a valid accusation. In fact, if it does not do that, it would be considered disconnected by the other nodes of the overlay. This would go against the assumption that a rational node seeks long term benefit by its permanence in the system.

$\Longrightarrow$ *Answer (3): YES.*

Question (4): does a rational node r have incentives to forward a message received via gossiping?

We described in Section 4.3.1.1 the rationale behind the reliable dissemination of accusations and notes in the Fireflies overlay. In fact, to ensure the timely and reliable dissemination of accusations and notes, Fireflies relies on nodes to pick f nodes to gossip with from their complete view of the system. If f is (on average) $ln(N) + O(1)$, then all the nodes are ensured to receive a given message with high probability [30]. Because of these probabilistic guarantees, a rational node might decide to omit forwarding messages received via gossiping from other nodes, in order to save bandwidth. In fact, as it believes other non-Byzantine nodes act altruistically, the redundancy of gossiping would still guarantee that these messages are diffused to all nodes.

$\Longrightarrow$ *Answer (4)*: **NO**.

But, if Byzantine nodes behave maliciously and omit forwarding, and if all rational nodes apply this strategy to maximize their utility, then the "tragedy of commons" [113] happens, where nobody forwards messages and thus nobody receives the messages. The protocol needs then to be modified to encourage all non-Byzantine nodes to follow the protocol and forward the messages they receive.

A.2 Making Fireflies incentive-compatible

We have discussed why the Fireflies protocol is not incentive-compatible with respect to the gossiping of accusation and rebuttal notes, as rational nodes could omit forwarding these messages in order to save bandwidth. We describe here the modifications to actually encourage rational nodes to forward accusations and rebuttal notes. This makes gossiping accusations and rebuttal notes on Fireflies incentive-compatible, thus resulting in making the protocol as a whole incentive-compatible from what said in the previous section. The modifications leverage the concepts of *predictable communication patterns*, *credible threats* and *balanced costs* which can be found in the seminal BAR paper [60].

The modifications to the gossiping sub-protocol are an extension of the approach proposed by Fireflies to prevent nodes from being overhelmed with load from malicious nodes. In particular, Fireflies proposes to leverage the multiple rings to impose the fixed set of nodes, specifically its rings predecessors, that a node can gossip accusations and rebuttals with. The same approach is used by SecureStream [96], a live-streaming protocol built on top of Fireflies, to impose the fixed set of nodes that a node can query —via gossiping— to get updated chunks of the live stream.

From what pointed out in the previous section about the suitable choice of the f random dissemination targets to probabilistically ensure reliable dissemination, it follows that the number of successors of a node, and hence the number of the Fireflies overlay rings, should account for this value. Then, in order to make the gossip sub-protocol of Fireflies incentive-compatible the following modification could be done. A credible threat should encourage rational nodes to follow this gossip sub-protocol and hence actually forward messages to their successors. The rationale behind this credible threat is explained through the example illustrated in Figure A.1. Here node A

publishes a message by sending it to all its successors nodes on the various rings. Upon reception of the message for the first time, a node forwards it in turn to all its successors. In the example, we see how the probabilistic guarantees provided by gossiping on the multiple rings allow for the reliable dissemination of the message despite the fact that the node D omits forwarding.

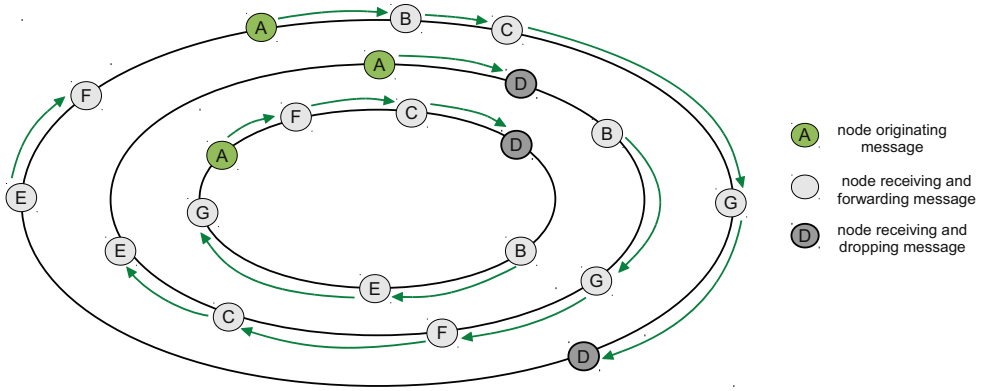


Figure A.1 – Fireflies: possible ordering of 7 nodes on 3 rings (*Based on:* [96]). Here node A publishes a message which is disseminated via gossiping to all the successors on the Fireflies multiple rings. Message is delivered to all nodes despite node D omitting forwarding the message.

Because of the probabilistic reliable dissemination on the overlay formed by the rings, a node n expects to receive the same message being forwarded by all of its predecessors on the rings. If n does not receive the message from a predecessor p[1], then the node n can employ a local blacklisting mechanism à la BAR-B [60]. In order for this blacklisting incentive mechanism to be effective, we propose to enhance the messages disseminated via the gossip sub-protocol with an additional fixed-sized `blacklisted` bitmap which specifies which nodes have been blacklisted by the node sending the message. The blacklisted nodes are supposed not to be given the message during its dissemination on the rings. The fixed size length of the bitmap provides a balanced cost incentive for rational nodes to actually specify the blacklisted nodes when they create and send the message. In fact, not setting their bits would not decrease the communication cost in transmitting the message.

We conclude this study of how to modify Fireflies to make it IC-BFT by pointing out two things. First, the proposed extension makes the gossiping sub-protocol used to disseminate Fireflies-specific messages (accusations and rebuttal notes) incentive-compatible. Second, the very same gossip sub-protocol can also be used to disseminate messages published as part of an application-level protocol running on top of Fireflies, thus resulting in having a general purpose incentive-compatible gossip-based dissemination service to be used by Fireflies and on top of Fireflies.

[1]Nodes can adjust an adaptive timeout for messages to be received by their predecessors, e.g., how long to wait for the reception of the messages after one predecessor has forwarded it.

B

FireSpam Incentive-Compatibility: proof sketch

In this appendix we provide a more formal description of $\mathcal{F}ire\mathcal{S}pam$ incentive-compatibility than that given in Section 5.3.2. In fact, we sketch here the demonstration of why it is in the rational nodes' best interest to follow the protocol, based on the assumptions usually made by BAR systems (e.g., [42, 60, 102]), and listed in Section 4.3.2 of this document. We provide here a short summary for convenience.

We recall that we do not assume a bounded number of rational nodes: all non-Byzantine nodes could behave rationally if they are able to maximize their utility function in doing so. The only assumption we made is that rational nodes do not collude. Relaxing this assumption is possible future work.

We assume rational nodes seek long term benefit from participating in the system. In fact, a rational node would join and remain in the system to be able to reliably receive and publish messages, as it would be the case for a user forum for example. Furthermore, as already conveyed by Formula 5.1, we assume rational nodes do not want to risk decreasing their benefit while attempting to reduce the participation cost, as the value of the service is considered more important than its associated communication costs. Also, in case of same net benefit in following or deviating from the protocol, we assume rational nodes decide to follow it.

We also assume that rational nodes adhere to the *promptness principle* [60]: if they have no benefit in delaying the sending of a message, they will send it as soon as possible.

We assume rational nodes are conservative in their actions with respect to the estimated impact of Byzantine nodes. This means that a rational node assumes that the maximum number of Byzantine nodes will possibly act in a malicious way, and that a Byzantine node will take the protocol action which could harm the most the rational node.

B.1 Preliminary definitions

Before proving the protocol, we introduce some preliminary definitions which we reuse throughout the proof.

Definition B.1 (Correct forwarding view assignment)**.** *A node p has a correctly assigned forwarding view if this contains nodes which, according to the ordering given by the filtering capabilities of nodes, fall into the candidate set composed of the nodes which surround p in the ladder.*

Definition B.2 (Correct publication view assignment)**.** *A node p has a correctly assigned publication view if this contains nodes which, according to the ordering given by the filtering capabilities of nodes, fall into the candidate set composed of nodes located at the bottom of the ladder.*

Definition B.3 (Correct forwarding view membership)**.** *A node s has a correct forwarding view membership if, for each node p whose forwarding view s is member of, s does not violate p's forwarding view correctness definition (cfr. Definition B.1).*

Definition B.4 (Correct publication view membership)**.** *A node s has a correct publication view membership if, for each (if any) node p to whose publication view s is member of, s does not violate p's publication view correctness definition (cfr. Definition B.2).*

In the rest, by simply "correct view assignment" we mean *both* correct publication and forwarding view assignment. Similarly, for "correct view membership".

B.2 Demonstration

In order to prove that rational nodes follow the protocol, we prove in a game theoretic framework along the lines of works like BAR-B [60] and BAR Gossip [42]. If it is possible to prove that the protocol provides a strict Nash equilibrium, then it will be in rational nodes' best interest to always follow it [61]. As done in BAR Gossip [42], we point out that in the rest a rational node assuming other nodes to behave altruistically is a Nash equilibrium artifact proof.

The incentive-compatibility proof of the protocol is then described by means of the following two theorems:

Theorem B.1 (Stepping Theorem)**.** *If at time t_i (with $i \geq 0$) a rational node r has a correct view assignment and a correct view membership, then r will follow the protocol to have a correct view assignment and membership at time t_{i+1}, otherwise risking blacklisting by some nodes or even eviction from the system.*

Theorem B.2 (Incentive Compatibility Theorem)**.** *If the Stepping Theorem holds, and if it possible to provide a correct view assignment and membership for a rational node r at time t_0, then r will follow the protocol at any time t_i (with $i \geq 0$).*

Proof sketch. Directly following its enunciation, the Incentive Compatibility Theorem is proved by natural induction. □

For the Incentive Compatibility Theorem to be proved, we need to prove that its two conditions hold. We prove the possibility to give a correct initial view assignment and membership to a rational node by means of the Lemma B.1. This Lemma proves to be even more generic: it is not only possible to guarantee a correct initial view assignment and membership for a rational node at t_0, but to every node in general.

Lemma B.1. *It is always possible to provide a correct initial view assignment and membership for a node p at time t_0.*

Proof sketch. A node p joining the system is considered initially to have very low filtering capability. This implies that its node monitors will be able to provide it a correct view assignment and membership. In fact, the node monitors of p can correctly give to it a forwarding view containing nodes currently with low filtering capability (at the bottom of the ladder). And, the node monitors of p can correctly put p in the forwarding view of nodes with low filtering capability, and might choose it as candidate for the publication view of other nodes in the system. □

To prove the Stepping Theorem instead, we decompose it in various Lemmas which represent the distinct parts of the $\mathcal{F}ire\mathcal{S}pam$ protocol in which we prove why a rational node would follow the protocol. Proving all these parts will have the effect of proving the Stepping Theorem itself, and thus also the Incentive Compatibility Theorem which depends on it.

In the following, to prove the Stepping Theorem provides a Nash equilibrium, we rely on the following principles present in the seminal BAR paper [60]: ensuring long term benefit, predictable communication partners, credible threats, balanced costs. Moreover, we leverage local ternary message classification (good/spam/undetermined) to detect omissions.

Lemma B.2. *A rational node r does not expect a benefit in being blacklisted by a node p.*

Proof sketch. By being blacklisted by a node p, a rational node r will decrease its benefit in the system because it will not be able to receive messages published by p. In fact, the rational node r knows that altruistic nodes will not forward a message m to it as the publisher specified r is blacklisted and hence they will follow the protocol by not forwarding m to r. Also, as for what we said a rational node has conservative reasoning with respect to Byzantine nodes, r will expect that also the Byzantine nodes it is in the forwarding view of will not forward the message m to it. □

Lemma B.3. *A rational node r does not expect a benefit in being evicted from the system.*

Proof sketch. Being evicted would be against the rational node r's long term benefit seek. In fact, as a rational node r's benefit is given by the receiving and publishing of messages, the eviction would prevent it from doing so. □

Lemma B.4. *A rational node r does not accept a message m from a node p if r is not a member of the (publication or forwarding) view of p.*

Proof sketch. The rational node r does not have a net benefit in accepting messages outside of the dissemination protocol guidelines. In fact, even if the message disseminated by p is good, r will anyway eventually receive it from its legitimate neighbors, thanks to the reliable dissemination of (good) messages. For what specified in the assumptions, having the exact same utility[1] in accepting or not the message m, the rational node r will follow the protocol and ignore m. □

Lemma B.5. *A rational node r does not send a message m to a node p if p is not in its (publication or forwarding) view.*

Proof sketch. As a rational node assumes other nodes behave altruistically thus following the protocol, it has no incentive in sending a message to a node which is not in its (publication or forwarding) view as this latter will not accept it. Furthermore, sending message outside of the protocol will increase its cost because of more bandwidth consumption. □

Lemma B.6. *A rational node r, if maliciously blacklisted by a Byzantine node p, does not incur the risk of being blacklisted by non-Byzantine nodes or being evicted.*

Proof sketch. Each message published by p lists r in the blacklist field of the message m. As such, the non-Byzantine forwarding view neighbors of r know that r cannot possibly forward m as previous nodes along the path, by adhering to the protocol, have not forwarded it to r. Thus, they will not blacklist r if they do not receive m from it. A similar reasoning applies for the node monitors: not receiving a MAJORITYREPORT from r's forwarding view neighbors for the given message m, they will neither blacklist r nor suggest r for eviction. In fact, they see that r is blacklisted by the publisher of m and so it could not forward m. □

Lemma B.7. *If a rational node r detects that a node p is not following the protocol, r will always blacklist p.*

Proof sketch. Blacklisting a node does not add an extra cost on the rational node r. In fact, according to the balanced cost principle, the blacklist field in the message being published by a r is of fixed size, so r will not save any byte in not indicating p in the list. Also, in the case r is blacklisted in turn by node p, r does not incur a risk of being blacklisted by other nodes or evicted from the system, as stated by Lemma B.6. □

Lemma B.8. *A rational node r never blacklists a node p if p has followed the protocol.*

Proof sketch. For what proved so far, blacklisting or not a node p does not add an extra cost in itself (cfr. Lemma B.7), as the blacklist field in a message published by r is of fixed size. But, by falsely blacklisting nodes, the rational node r will see its benefit decreased because of two things. First, its published messages will reach a fewer number of nodes. Second, by falsely blacklisting a node p, it may cause the blacklisted node p to seek "vengeance" by blacklisting in return r. This for what said by Lemma B.2 is of no interest for the rational node r. □

[1] Note that in the utility function that we presented in Formula 5.1 we do not consider the fact of receiving "earlier" a message as of increasing benefit, as long as the message is eventually received.

Lemma B.9. *A rational node r never forwards a message m to a node p which is listed as blacklisted by the publisher of the message.*

Proof sketch. The proof comes from similar reasoning of the proof of Lemma B.6. Furthermore, by adhering to the semantic of the blacklisting, the rational node r will save bandwidth by not forwarding to the blacklisted (by the publisher) node. □

Lemma B.10. *A rational node r when it publishes a message m, it sends it to all nodes in its publication view.*

Proof sketch. It is in the rational node r's interest to disseminate its published message m reliably in the system. □

Lemma B.11. *A rational node r never forwards a message m it locally classified as $spam$.*

Proof sketch. By forwarding a $spam$ message, the rational node r would increase its overall amount of forwarded messages, and hence decrease the filtering capability that is assessed by its node monitors. This is against the terms of its utility function for two reasons. First, r would waste bandwidth in forwarding m as it is not supposed to. Second, by making appear as it had a lower filtering capability, it will be assigned a lower position in the ladder topology than the one it would deserve, thus receiving more $spam$ messages. □

Lemma B.12. *Upon the first reception of a given message m not classified as $spam$, a rational node r always forwards it to all the nodes in its forwarding view.*

Proof sketch. By not adhering to the protocol in fact the rational node r knows that it risks:

- to be blacklisted by a node p in its forwarding view to which r did not forward m. Specifically, if p receives the message m from another node than r and classifies it as $good$, p will detect r's misbehavior and blacklist it.
- to be blacklisted and suggested for eviction by a node q among its node monitors. Specifically, if q receives m and classifies it as $good$, it will expect the reception of a majority of REPORT messages from nodes in r's forwarding view. If q does not receive such reports, it will blacklist r and suggest to the other node monitors of r to evict it. A majority of monitors suggesting for eviction of r will then cause the removal of r from the system.

Because of these credible punishments, the rational node adheres to the protocol. Not doing it would lead to no benefit, as already proved by Lemma B.2 and Lemma B.3. □

Lemma B.13. *A rational node r, which is a node monitor for a node p, always replies and with the correct answer to a check request made by p.*

Proof sketch. If r does not reply, p will blacklist r, which is against r's interest as proved by Lemma B.2. Also, as a check response should be of fixed size (according to the balanced cost principle), and as the monitor and the monitored node cannot be in each other forwarding view according to the protocol view assignment construction choices, the monitor node has no interest in not providing the correct answer. □

Lemma B.14. *A rational node r always checks on its node monitors that all its neighbors reported about a given message m it forwarded to them.*

Proof sketch. The rational node r in fact has to check that its monitors receive a consistent majority of REPORT messages about r having forwarded m, otherwise risking blacklisting and even eviction, which is against r's interest as proved in Lemma B.2 and Lemma B.3. As r conservatively assumes in its view there is the maximum allowed of Byzantine nodes, it will have to check for all of the neighbors to have reported about the message m. □

Lemma B.15. *A rational node r always blacklists a neighbor node p if this did not report about r having forwarded a given message m.*

Proof sketch. This is proved as a consequence of Lemma B.14 and Lemma B.7. □

Lemma B.16. *Once a rational node r receives a message m from a node p (being r in p's forwarding view), r always reports to all the monitors of p about the message m received from p.*

Proof sketch. Because the node p checks on its monitors about the reports received and monitors actually provide the answer, the node r would be detected as not reporting about the received message m and would thus be blacklisted by p, which is not in its interest as said in Lemma B.2. □

Lemma B.17. *Once a rational node r has collected a majority of reports about a monitored node p's forwarding for a given message m, it will always disseminate this information in the network.*

Proof sketch. The rational node r disseminates the MAJORITYREPORT message because other monitors in the same monitor set of p have an equivalent report. Thus, thanks to the reliable dissemination in the node monitors overlay[2], they can observe whether each other monitor in the monitors set has indeed disseminated this information. If r does not disseminate the MAJORITYREPORT message, it will then fear to be blacklisted by the other nodes in the monitor set of p, and it has no interest in doing so, as already stated by Lemma B.2. □

Lemma B.18. *A rational node r always checks whether another node monitor q belonging to the same node monitors set for a monitored node p has not disseminated a* MAJORITYREPORT *message about a message m forwarded by p, and if so, r always blacklists q and suggests for eviction of q.*

Proof sketch. From the rational node r point of view there is no extra bandwidth cost[3] in observing whether other nodes have indeed disseminated a MAJORITYREPORT message. Thus, from the assumptions we made in this section, in case of same cost the node will follow the protocol, hence blacklisting the monitor which omitted disseminating (cfr. Lemma B.7). Also, as suggesting for eviction a node is a subject to the balanced cost principle, a node monitor has no benefit in not informing other node monitors that one has not followed the protocol. □

[2]The reliability of message dissemination in the monitors overlay is provided by Fireflies [97], which we show in Appendix A how it can be made incentive-compatible.

[3]We do not consider reducing CPU utilization as a deciding factor here.

Lemma B.19. *A rational node r always sends the eviction suggestion to the other monitors of the same monitor set for a node p, and always says the truth (true/false) about the eviction suggestion of p.*

Proof sketch. The demonstration is similar to the one of Lemma B.13. As an eviction suggestion message is of fixed size and has the rational node r has not interest in not reporting the truth, it will send the correct value. □

Lemma B.20. *A rational node r always blacklists and suggests for eviction a node monitor p which did not participate in the view assignment or that proposed a wrong view.*

Proof sketch. The proof comes from what said in Lemma B.7 and Lemma B.19. □

Lemma B.21. *A rational node r always participates in the periodic view assignment with the other node monitors of a node p, and it always proposes the correct view.*

Proof sketch. The rational node r fears to be blacklisted or evicted if not participating in view assignment or in proposing a wrong view, which is against r's interest as stated in Lemma B.2 and Lemma B.3. In fact, this will be done either by non-Byzantine nodes (as they follow the protocol) or by Byzantine nodes themselves (as the rational node conservatively foresees the Byzantine nodes' behavior that could impact it the most). □

As we have proved so far, a rational node's best interest is to adhere to each part of the protocol. This ultimately proves the Theorem which states that the whole protocol is a strict Nash equilibrium. As a desidered consequence of rational nodes following the protocols and from the design choices about the bounded number of Byzantine nodes, we have the following:

Corollary B.1. *The dissemination of (good) messages is reliable among all nodes of the system.*

Summary. Finally, Table B.1 gives a summary of which main Lemma provide the incentives for each one of the seven protocol steps of $\mathcal{F}ire\mathcal{S}pam$.

Protocol step	**Lemma(s)**
Message publication	B.10
Message forwarding	B.12
Message reporting	B.16
Node evaluation	B.17, B.18
Node monitor collaboration	B.19
View assignment	B.20, B.21
Report checking	B.13, B.14, B.15

Table B.1 – $\mathcal{F}ire\mathcal{S}pam$: summary of protocol steps and associated lemma(s).