NATCloud: Cloud-Assisted NAT-Traversal Service

Hanna Kavalionak
CNR-ISTI, Pisa, Italy
hanna.kavalionak@isti.cnr.it

Amir H. Payberah
SICS, Sweden
amir@sics.se

Alberto Montresor
University of Trento, Italy
alberto.montresor@unitn.it

Jim Dowling
SICS, Sweden
jdowling@sics.se

ABSTRACT

Although over the last decade large efforts have been done to design efficient peer-to-peer (P2P) protocols, very few of them have taken into account the problem of firewalls and network address translators (NAT). Most of the existing P2P systems do not work properly when a high percentage of nodes are behind NAT. While a few P2P systems tackled the NAT problem, all of them employ third party nodes to establish a connection towards nodes behind NAT, and these may become bottlenecks, menacing the health of the entire system. A possible solution to this problem is to rent extra resources from the cloud. This paper presents NATCLOUD, a cloud-assisted NAT-traversal service, where rented cloud resources are added on demand to the overlay, as third party nodes, to help other nodes to make connections to nodes behind NAT. We show the feasibility of integrating our approach with existing gossip-based peer sampling services and evaluate our solution by simulations, conducting extensive experiments under different network conditions.

CCS Concepts

•**Networks** → **Network algorithms;** *Network performance evaluation;* Network properties;

Keywords

Distributed algorithms, Peer-to-peer, Self-adaptive algorithm, Load balancing, NAT-traversal

1. INTRODUCTION

During the last decade, most of the research in the field of peer-to-peer (P2P) networks has been focused on creating and improving overlays. While adopting P2P solutions for implementation of the Internet services and applications, the research community tackled a wide range of important issues, like cost-efficiency [17], reliability [9] and performance [8].

SAC 2016,April 04-08, 2016, Pisa, Italy

http://dx.doi.org/10.1145/2851613.2851640

Despite their effectiveness, most of the proposed approaches are based on the assumption that all nodes can directly communicate with each other. Unfortunately, over the last decade the Internet IP architecture has undergone steady changes, such as the spreading of NATs gateways and firewall systems, which have progressively led to the loss of end-to-end addressability. Nowadays the percentage of the Internet nodes that are behind NAT, and therefore cannot be accessed directly, is so high that the well-being of existing P2P protocols is at risk. An example of such protocols is the *peer sampling service* (PSS) [24], a service widely used in the context of gossip-based networks. The PSS continuously provides nodes with a uniform sample of the live nodes in the network, in fact a partial view of the entire network. PSS' are usually exploited as building blocks for P2P networks, as they help keeping the network updated and connected. Moreover, PSS' increase the resilience of the overlay to network churn. However, in the presence of NATs, a large fraction of nodes cannot establish direct connections to each other. Hence, NATted nodes become under-represented in partial views, and traditional PSS become biased [10].

Recent papers have extended existing PSS service to work under NAT [16, 2, 5]. In these works, the main role is played by public P2P nodes that can directly access the network. These nodes are used as relaying [16] or rendezvous [5] servers. However, state-of-art solutions so far did not consider public nodes overloading and service reliability. In case the amount of public nodes is small, such nodes may become overloaded by the increasing intensity of requests from the nodes behind NATs. Moreover, if there are no available public nodes, NATs nodes become isolated and the whole network is exposed to the risk of failure. These limitations can significantly decrease the reliability of the distributed applications relying on PSS services.

To overcome these limitations, we designed a novel approach, called NATCLOUD, that can be applied to resolve the NAT issue for real network P2P applications. The vision of NATCLOUD is to combine P2P and cloud computing technologies for an effective NAT-traversal peer-sampling. The problem of node overloading is solved by allowing NATCLOUD to exploit available P2P resources, and relying on the cloud whenever P2P resources are not enough to support a given quality of service. Considering the cost of cloud resources, it is crucial to provision the required cloud resources properly. If too few cloud resources are available, connectivity to some private nodes may be lost. On the other hand, if too many resources are rented, the cost will be unnecessarily high.

To demonstrate the feasibility of our solution, we integrated NATCLOUD with CLOUDCAST PSS [13]. We performed extensive sim-

ulations and evaluated the integrated protocol under dynamic settings and with different network sizes. The evaluation shows that NATCLOUD has a negligible impact on the PSS when the P2P resources are enough. On the other side, when the P2P public resources are under-represented or not available, NATCLOUD successfully involves cloud resources in order to preserve the PSS quality of service.

2. RELATED WORK

Over the last decades the Internet IP architecture has undergone steady changes. One of the most important changes is the spreading of NAT approaches, which have progressively led to the loss of end-to-end addressability. Therefore in the last years it became crucial to study techniques able to cope with NAT systems [18, 5, 6].

Here we distinguish two main techniques that are used to communicate with private nodes behind a NAT: *hole punching* [20, 4] and *relaying* [12]. Hole punching can be used to establish direct connections that traverse the private node's NAT, while relaying can be used to send a message to a private node via a third party relay node that already has a connection with the private node. With relaying, all data transmitted between two nodes behind the NATs will pass through a third party node. Hence, this approach consumes a lot of bandwidth and guaranties little or no privacy. The hole punching technique allows the nodes behind the NATs to communicate with the help of a rendezvous server. This server is required only for the exchange of the addresses, and that creates a little impact on the bandwidth consumption. For example, in case of the hole punching procedure implemented by the NatTrav library [4], the peers willing to receive connections from outside of NAT system register themselves with an intermediate connection broker.

Gossip-based PSS' are widely used as building blocks for P2P overlay networks. In networks where all nodes can directly communicate with each other, PSS' can provide nodes with a uniform random sample of the nodes in the network [24, 7, 13]. However, in the Internet, where a high percentage of nodes are behind NATs and firewalls, traditional gossip-based PSS' become biased or the network may even become partitioned [10]. The first PSS that addresses the problem of NATs was ARRG [3]. In ARRG, each node maintains a list of nodes with whom it has had a successful gossip exchange in the past. When a node view exchange fails, it selects a different node from this list. This approach however, biases the PSS, since the nodes in the list are selected more frequently for gossiping. Nylon [10] is another NAT-aware PSS that uses all existing nodes in the system (both private and public nodes) as rendezvous servers. Renesse et al. presented an approach to fairly distribute relay traffic over public nodes [11].

Gozar is another NAT-aware PSS that replaces rendezvous server chains with one- hop relaying to all private nodes [16]. Private nodes discover and maintain a redundant set of public nodes that act as relay nodes on their behalf. Croupier provided an alternative NAT-aware PSS without relaying [2]. Instead of routing the messages to private nodes, the messages are sent only to public nodes that execute the shuffling of the descriptors on behalf of both public and private nodes. An alternative NAT-aware PSS is proposed by Roverso et al. [21]. The authors propose an algorithm where each private node randomly chooses a public node and systematically places samples of itself on nodes in the neighborhood of this public node. Pouwelse et al. propose a peer-to-peer client Tribler that uses social phenomena to connect peers and find content in the network [19]. Usurp is another NAT traversal distributed solution similar to our work that is designed to support the connectivity of overlay networks [15]. The authors propose a solution where the public nodes are organized in a structured overlay network. Each private node is assigned to a key in the overlay, and the public node who manages such key also acts as a rendezvous and relay server to the private node.

While the described NAT-aware methods demonstrate promising results for PSS support in P2P networks, all of them are limited by the percentage of available public nodes in the network. In this paper we propose a solution that combines together both P2P and Cloud Computing technologies in a way that allows preserving the overlay connectivity even in case of absence of available public nodes in the network. We employ an adaptive selection of the rendezvous servers that allows us to optimize resources utilization in the network, while minimizing the negative impact on the P2P public nodes bandwidth consumption and the cloud computing service costs.

3. PROBLEM STATEMENT

We consider a network of peer machines that contains two types of nodes: *private* and *public*. Private nodes, i.e., *client* nodes, are the nodes behind NAT and cannot be accessed directly. Public nodes, i.e., *proxy* nodes, are open to end-to-end communication. To allow client nodes to participate in a P2P protocol, each client node keeps a list of proxy nodes willing to bootstrap communication channels on behalf of the client.

In this work, we adopt the so-called NAT-traversal "hole punching" technique [5]. To establish communication with a client node, one of the proxy nodes associated with the client needs to be contacted and acts as *rendezvous server*. We do not consider here the details of the hole punching technique that depends on the different types of NAT systems and communication protocols [22]. Instead, we focus on the problem of rendezvous servers overloading and reliability. The number of proxy nodes assigned to each client clearly affects the availability of such clients. We define the *availability factor* (AF) of a proxy node as the probability of being up and running, capable to serve its clients by helping in the establishment of new connections. The *availability factor* of a client node is defined as the probability of being able to accept new communication requests. It thus depends on the availability factor of the proxy clients associated with it.

The goal of this work is to maximize the availability factor for all client nodes belonging to the system. Ideally, boosting clients' AF requires a large number of proxy nodes. Unfortunately, this is not generally possible due to the limited number of them. To overcome this limitation and increase the AF for clients, we propose to adopt the concept of *cloud proxy*, as an always-available proxy node hosted on a cloud provider. While renting a cloud proxy can ideally solve all our problems, it does come with a cost. Intensive cloud utilization can hinder in the economical sustainability of the approach. Therefore, it is important to strike a balance between the amount of cloud resources used and the overall system QoS.

In this paper, we address the issue of autonomously regulating the utilization of resources between the cloud and a set of available P2P public nodes, while maximizing the clients' AF. To this end, we propose a self-regulation mechanism that focuses on proxy nodes management in cloud-assisted NAT-traversal. Our target is to meet a desired level of QoS while minimizing the economical cost. We

consider the AF as the critical parameter for selecting the proxy. As a consequence, the problem of self-regulation can be divided into two parts: (i) availability factor modeling and (ii) cost minimization.

4. NAT-TRAVERSAL MODEL

Each node in the system is represented by a descriptor $\{N_{id}, NAT_{type}, \{P_i\}\}$, where N_{id} is the unique node identifier, NAT_{type} indicates the node NAT type, i.e., public or private, and $\{P_i\}$ lists the assigned proxies to the client. This list is empty for public nodes. The proxy nodes can be chosen from the public nodes or the cloud proxy. Client nodes can substitute their proxies autonomously among public nodes and the cloud proxy. However, since each communication with the cloud is associated with a cost, client nodes try to choose their proxies from public ones, if the number of available public nodes is enough to guarantee the minimum level of availability. Instead, when client nodes cannot achieve the required AF with the existing public nodes, they add a cloud proxy to their proxy list.

If all proxy nodes of a client node fail simultaneously, the client node becomes inaccessible from other nodes. Therefore, it is crucial to have a large enough proxy list to decrease the risk of simultaneous failure. However, a client node sends a keep-alive message to its proxy nodes periodically to keep the connection open. Hence, the more proxy nodes are assigned to a client, the more network traffic will be generated. Therefore, to avoid network overhead we introduce the concept of *redundant threshold* that limits the maximum number of proxy nodes that is reasonable to keep per client.

The Availability Factor. We present here a model for the availability factor that allows the client nodes to compute their number of proxy nodes according to the desired availability. In order to compute the AF of a client, we need to compute the probability that all its proxy nodes are off-line at the same time τ. We consider two issues that can lead to a proxy failure: (i) sub-network failure and (ii) network churn rate. We assume that each of the nodes belongs to some sub-network, for example to same Autonomous System (AS). We consider AS to be a connected group of some IP prefixes that is run by one or more network operators with a single routing policy. As it is shown in the work of Sriram et al. [23], ASs can be subject to malicious attacks that can lead to AS partitioning. In this case, isolated AS cannot support client nodes. We also consider the churn rate of the network, that shows the percentage of nodes that leaves the network over a specific time interval.

More formally, each proxy node belongs to some AS_i that has a failure rate α_i. Moreover, each of the proxy nodes P_j has a failure rate π_j due to churn. Consider, for example, a client node q that has three proxy nodes P_1, P_2, P_3, such that $P_1, P_2 \in AS_1$ and $P_3 \in AS_2$. To compute the AF of q, we have to define the probability that at least one of these proxies is available during the next τ time interval. The probability that P_1 and P_2 are not available at the same time is defined with the multiplication of their nodes failure rates: $\pi_1\pi_2$. Hence, the probability that at least one of them is available is $1 - \pi_1\pi_2$. At the same time, if an AS is not available, all the nodes belonging to it are not available as well. Therefore, to have at least one of the proxy nodes P_1 or P_2 available, AS_1 should be available as well: $(1 - \alpha_1)(1 - \pi_1\pi_2)$. In all other cases the proxies P_1 and P_2 are not available, either because of their failure or their AS failure: $1 - (1 - \alpha_1)(1 - \pi_1\pi_2) = \alpha_1 + \pi_1\pi_2(1 - \alpha_1)$. With the same reasoning for AS_2, the probability that proxies belonging to AS_2 are not available is $\alpha_2 + \pi_3(1-\alpha_2)$. Hence the probability that during the τ time no proxies are available is $(\alpha_1 + \pi_1\pi_2(1-\alpha_1))(\alpha_2 + \pi_3(1-\alpha_2))$. Thus, the AF for this client node q is $AF = 1-(\alpha_1+\pi_1\pi_2(1-\alpha_1))(\alpha_2+\pi_3(1-\alpha_2))$.

To simplify the computation, we assume failure probability of the public nodes are similar and equal r: $\pi_1 = \pi_2 = \cdots = r$. We can, then, write the formula for AF of a client node as $AF = 1 - \prod_{i \in AS}[\alpha_i + r^{k_i}(1 - \alpha_i)]$, where k_i is the number of proxy nodes belonging to AS i.

The Load Factor. The support our NAT-traversal protocol, client nodes should keep all connections to their proxies open. To do that, client nodes send keep-alive messages periodically to their proxies. The higher the number of proxy nodes in a descriptor of a client, the more bandwidth is needed to support the keep-alive messages exchange.

We define the *load factor (LF)* as the average upload bandwidth at proxy nodes that consumed in keep-alive message exchanges. The LF can be computed as follows:

$$LF = \frac{m_{NAT} \cdot \overline{N_{proxy}} \cdot N_{private}}{N_{public} \cdot U} = \frac{m_{NAT} \cdot \overline{N_{proxy}}}{\gamma \cdot U} \quad (1)$$

where $\gamma = \frac{N_{public}}{N_{private}}$, m_{NAT} is the bandwidth rate of keep-alive messages, $\overline{N_{proxy}}$ is the average number of proxy nodes per client node, $N_{private}$ is the number of private nodes in the network, N_{public} is the number of public nodes in the network and U is the average available upload bandwidth per proxy node.

In order to limit the possible proxy overloading, we introduce the *maximum load factor*, LF_{max}, as the maximum percentage of upload bandwidth at a proxy node that can be used in keep-alive messages. We also define the *redundant* threshold as the maximum number of proxy nodes that a client node is allowed to have. In order to evaluate this value we consider Equation 1, where $LF = LF_{max}$ and $\overline{N_{proxy}} = redundant$:

$$redundant = \left\lfloor \frac{LF_{max} \cdot \gamma \cdot U}{m_{NAT}} \right\rfloor \quad (2)$$

5. NAT-TRAVERSAL PROXY NODES

The architecture of our system can be described as follows. Each client node executes three tasks: (i) peer-sampling, (ii) proxy-client connection support and (iii) NAT-traversal proxy nodes management. As a PSS, we adopted CLOUDCAST, a hybrid cloud-P2P protocol capable to "scale-down" to a very limited number of nodes thanks to cloud support. To support the proxy-client connection, clients periodically exchange keep-alive messages with their associated proxies. While this is a simple adaptation of existing solutions, the real core of our work is the autonomic management of proxy nodes.

Algorithm 1 describes the pseudo-code of our algorithm for the management of proxy nodes. The algorithm execution is divided in periodic rounds of length ΔT, which is a parameter of the system. At the start of each round, a client requests information about the current ratio γ_i between public and private nodes in the network through function getCurrentRatio(). According to the data from previous $(i-1)$-th round and the current i-th round, func-

```
repeat
    γ_i ← getCurrentRatio();
    γ_{i+1} ← getEstimation(γ_{i−1}, γ_i);
    redundant_{i+1} ← ⌊ (LF_max γ_{i+1} U) / m_NAT ⌋;
    if redundant_{i+1} ≤ Critical and Descriptor.Proxies !
    contain(cloud) then
        addProxy(cloud);
        [candidates] ← findBetterProxy(view);
        proxyOptimization([candidates]);
        return;
    AF_i ← getCurrentAF();
    if Descriptor.Proxies contain(cloud) and N_proxies ≥ (Critical + 1)
    and AF_i > AF_Critical then
        removeProxy(cloud);
    if N_proxies ≥ redundant_{i+1} then
        removeProxy(N_proxies − redundant_{i+1});
        [candidates] ← findBetterProxy(view);
        proxyOptimization([candidates]);
    if N_proxies ≤ redundant_{i+1} then
        if AF_i < AF_Critical and Descriptor.Proxies ! contain(cloud)
        then
            addProxy(cloud);
        findNewProxy();
    wait ΔT;
```

Algorithm 1: Algorithm executed by client nodes

tion getEstimation() estimates γ_{i+1}; it then computes the redundancy threshold $redundant_{i+1}$ for the next protocol round $i+1$ (Equation 2). If (i) the estimated redundancy threshold is smaller or equal than the critical threshold, and (ii) the client descriptor does not contain a cloud proxy, then the client includes a cloud as one of its proxies, optimizes the rest of the proxies and exits the algorithm until the next round. The optimization of the proxies is composed by the following steps. First, each client monitors the partial $view$ given by the peer sampling to find proxy nodes with better characteristics (such as the load factor in our case) through method findBetterProxy(). Such proxies are called *candidates*. Afterwards, a client replaces the proxies that are in the descriptor with better ones from the candidates by calling proxyOptimization($[candidates]$).

Otherwise, in case the number of proxies is sufficiently high, therefore the current availability factor and the number of proxies are higher than the critical threshold, a client can eliminate the cloud proxy from the list. If the current number of proxy nodes is higher than the expected redundancy threshold, then a client removes redundant proxy nodes from its descriptor. The decision about the proxy to remove is based on their load. In particular, the most loaded proxies are the candidates to be removed. The other proxies are optimized according to their load factors.

Finally, in case the number of proxies is smaller than the redundant threshold, a client promotes additional proxy nodes from the peer sampling service, through the partial view $view$; if the availability factor is less than critical, it also adds a cloud proxy to the proxies list.

6. EVALUATION RESULTS

In the first part of this section we present a theoretical analysis of the critical availability factor and the redundancy threshold based on the model described above (Section 4). Then, we present the validation and evaluation results of NATCLOUD. In order to validate the approach, we integrated NATCLOUD with the CLOUDCAST PSS protocol. CLOUDCAST is a protocol that integrates cloud storage with gossip protocols in order to maintain the overlay connected in case of small network sizes and high network churn rate. Moreover it is also able to maintain the number of accesses to the cloud under control. Hence, we validate NATCLOUD by controlling that integration of NATCLOUD and CLOUDCAST preserves these characteristics.

In order to evaluate the performance of NATCLOUD we evaluate the influence of metrics like ratio γ (1, 0.05, 0, 01, 0.0025), maximum allowed Load FactorLF_{max} (2%, 1%, 0, 5%, 0.25%), network size n (oscillating between 0 and 1024 nodes during the day period) and $churn$ rate (1%, 0.1%, 0, 01%, 0.00%), on performance metrics like CLOUDCAST cloud in-degree, availability factor AF and monetary costs. The simulations were executed using PEERSIM [14]. In our evaluation we have considered the average upload bandwidth per public node $U = 512$ Kbps and keep-alive messages rate $m_{NAT} = 11$ bps. We set the critical availability factor $AF_{critical} = 0.9$.

Theoretical evaluation

This section provides a theoretical evaluation of two important system parameters: the critical availability factor and redundancy threshold. The evaluation is based on the theoretical model that is described in Section 4.

Availability Factor Evaluation. Figure 1 shows the impact of ASs with different failure rates (0.001, 0.01, 0.1) on AF of a client. To simulate a stress situation, we consider an high churn rate $r = 0.1$. In our test case we consider the worst case of proxies layout in which all the proxy nodes belong to the same AS.

As we can see, in all the cases AF converges to its maximum when the number of proxy nodes is around 3. However, the maximum possible AF for each AS is different and limited by the AS failure rate. Therefore the proxies layout in terms of AS does not significantly influence on the optimum number of proxies for a client, but impacts on the maximum reachable AF. Arguing as above, we see that a number of proxy nodes greater than 3 yields on high AF even in case of high network churn. Therefore, we set the critical threshold to 3.

Load Factor Evaluation. In order to evaluate the LF model, we consider a "stress test" for the system, where keep-alive messages are sent every 30 seconds. In a real scenario, the interval between two consecutive keep-alive messages depends on the NAT type and can vary from seconds to minutes. Therefore, to evaluate the upload bandwidth consumption caused by NAT-traversal support, we chose a 30 seconds keep-alive interval as a reasonable stress condition. A generic keep-alive message contains no payload and it is composed only by the header, which results in a TCP/IP message of 41 bytes.

Figure 2 shows a graphical representation of Equation (1), where $U = 512$ Kbps and $m_{NAT} = 11$ bps and represents the influence of a ratio between private and public nodes in the network on a proxy node overload. The curves are shown for different $\overline{N_{proxy}}$ per client descriptor. While the larger number of proxies assigned to a client $\overline{N_{proxy}}$ increases LF the actual impact is not significant. However, when γ is low (less than 0.5) and so proxy nodes are few in the network, the risks of overloading increases.

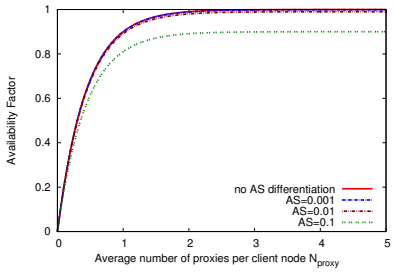


Figure 1: **AF of a client vs. number of its proxy nodes for different AS failure rates.**

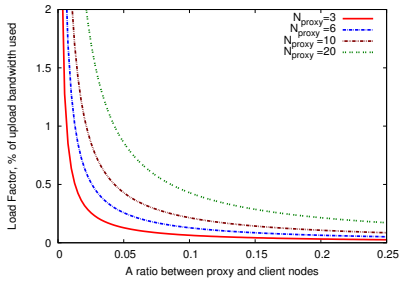


Figure 2: **The LF impact for different ratio between proxy and client nodes for different $\overline{N_{proxy}}$.**

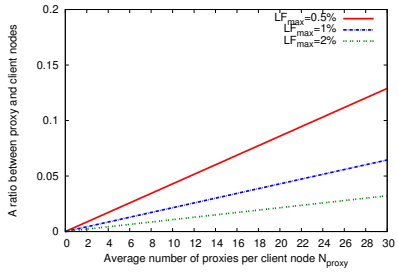


Figure 3: **Maximum number of proxy nodes in a client descriptor for maximum allowed LF_{max}**

Figure 3 shows the required minimum γ to support an average number of proxy nodes per client $\overline{N_{proxy}}$ with a maximum allowed LF_{max}. It follows that a client node with more than $\overline{N_{proxy}}$ proxy nodes in the descriptor should reduce its number of proxies to avoid overtaking LF_{max}. At the same time, the number of assigned proxies cannot be less than the critical threshold (which is around 3 according to the AF model).

Simulation results

The first evaluation validates NATCLOUD by integrating it into the existing CLOUDCAST PSS and then we evaluate the integration of the protocols in different network conditions. Finally, we evaluate the monetary costs.

Integration of* NATCLOUD *and* CLOUDCAST *PSS. In order to validate the NAT-traversal algorithm, we verified whether the scalability properties of CLOUDCAST are preserved. We considered a network whose size oscillates between 0 and 1024 nodes over the period of one day. Figure 4 shows the results of 2 simulated days.The figure demonstrates that the in-degree of a peer in the network oscillates around some average value and does not follow the growth of the network size. At the same time even in case of empty network, the cloud entity remains awake and facilitates the network bootstrapping afterwards. Hence, we show that the integration of NATCLOUD in CLOUDCAST does not have a negative impact on the scalability and the described periodic behavior corresponds to the behavior of the original CLOUDCAST protocol.

Figure 5 evaluates the ability of the protocol to support different network churn rates. We have evaluated NATCLOUD for various size of the network and compared them with the original CLOUDCAST protocol (Figure 6). For the evaluation we set the maximum load factor (LF_{max}) to 1% and the ratio between available proxy and client nodes in the network (γ) to 0.05. The NATCLOUD protocol performs well even in case of churn. The average cloud in-degree remains the same (apart for a small deviation) even for large network sizes. Nevertheless, as we can see from both figures, the average in-degree for the CLOUDCAST integrated with NATCLOUD is higher than without it. We explain this by the small availability of P2P proxies resulting in possible delays when contacting client peers. Moreover, as well as in the original CLOUDCAST work, the system cannot cope with the 1% network churn, which corresponds to extremely short peer lifetime. The validation suggests that NATCLOUD can be applied to the PSS without a significant negative impact on its properties.

NATCLOUD *evaluation.* We evaluate now the influence of LF, γ and $churn$ on the availability factor, CLOUDCAST cloud in-degree and the economical cost. We considered the average cloud in-degree of CLOUDCAST PSS with the integrated NAT-traversal mechanism for different values of maximum load factor LF_{max} and ratio between proxy and client nodes γ (Figures 7, 8). As we can see, with higher value of γ and LF_{max}, the average in-degree for the cloud in the considered PSS decreases. Nevertheless, in case $\gamma = 0.0025$ (Figure 7), which corresponds to extremely low concentration of proxy resources in the network, our experiments show a relatively low average cloud in-degree. We explain it with the fact that in case there are not enough proxy resources in the network, the clients more frequently use the cloud proxy and the communication is mostly organized with its support. On one side, this allows us to reduce the negative impact of using P2P nodes and keep the cloud in-degree for CLOUDCAST PSS low. On the other side, as we show later, it brings additional economical costs to the system.

In both cases we see that a large size of the network does not significantly influence the in-degree. Instead, the smaller sizes of the network are characterized with higher value of cloud in-degree. We explain it with the fact that low levels of available proxy resources in case of small network size influences on the work of PSS and as a result the average in-degree of a cloud increases. Instead, starting from some network size, there is enough of proxy resources to serve the PSS and the average in-degree decreases to a stable value.

In order to evaluate the quality of service supported by the proposed solution, we evaluated the availability factor of client nodes with multiple network sizes and under different churn rates (Figure 9). The experiments show that the proposed approach is able to sustain the critical availability factor almost for all the clients. Nevertheless, with 1% and 0.1% churn rates, the availability factor of some clients drops below the threshold. We explain this decreasing with the extremely small lifetime of the nodes and the time needed for a client to bootstrap the NATCLOUD (to find the proxies).

Costs evaluation. We have evaluated our algorithm using the Amazon S3 storage service [1]. Given the amount of data involved, we assume that storage costs are negligible. Hence, we base the cloud costs evaluation just on the prices of the GET and PUT requests [1].

Figures 10 demonstrate the approximate costs of cloud utilization for NATCLOUD part in the integration with CLOUDCAST. The fig-

[1] 0.004$ for 10000 GET requests; 0.005$ for 1000 PUT requests; (accessed March-2015, Region EU (Ireland))

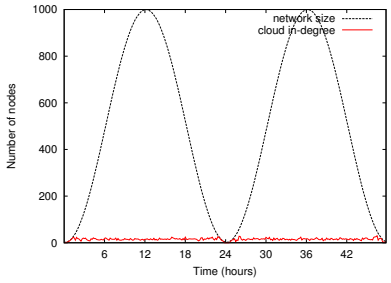


Figure 4: **Scalability properties of CLOUDCAST with NATCLOUD;** $\gamma = 0.05$, $LF_{max} = 1\%$

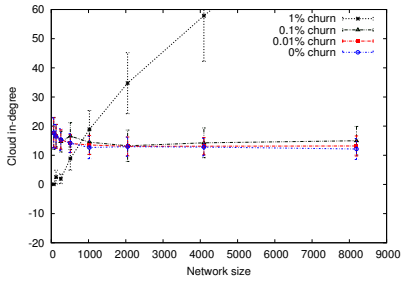


Figure 5: **Average cloud in-degree in CLOUDCAST with NATCLOUD;** $\gamma = 0.05$, $LF = 1\%$

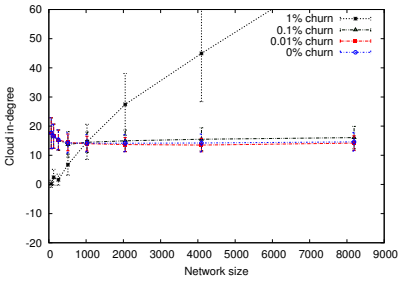


Figure 6: **Average cloud in-degree in original CLOUDCAST protocol**

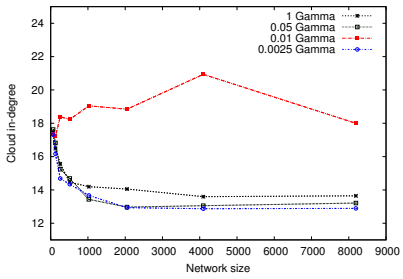


Figure 7: **Average cloud in-degree for different** γ**;**0.01% **churn rate,** $LF_{max} = 1\%$

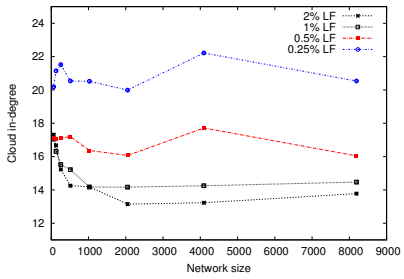


Figure 8: **Average cloud in-degree for different** LF_{max}**;**0.01% **churn rate,** $\gamma = 0.05$

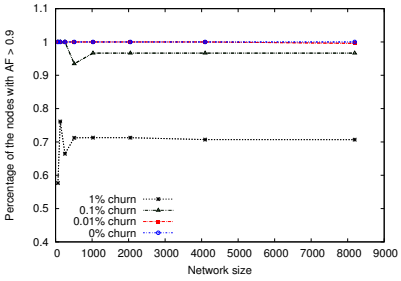


Figure 9: **% of the client nodes with availability factor greater then 0.9;**$LF_{max} = 1\%$, $\gamma = 0.01$

ures present the simulation of a network with a churn rate of 0.01. Figure 10(a) demonstrates the influence of different values for the γ and different network sizes on the cloud utilization costs per node. The maximum load factor parameter was set up to 1%. As we can see from the results the higher the concentration of the potential proxy nodes in the network, the lower the actual cloud costs. The Figure 10(b) demonstrates the influence of different network churn rates on the costs. In the experiment, the ratio between proxy and client nodes γ is set to 0.05. The experiments show that increasing the network churn rate increases the intensity of cloud utilization, while the impact of the network size is negligible.

Finally we would like to notice that the prices in the long term are not negligible. Nevertheless, NATCLOUD self-adapts to the current network state. As we can see on the Figures 10, in case the network parameters (γ and churn rate) are regular, the NATCLOUD costs are close to zero. Instead, in extreme cases when the network is under high churn or lack of P2P proxies, NATCLOUD allows with a reasonable price to support the regular functioning of PSS. Here we would like to mention the perturbation of the costs in case of small networks. We connect this phenomenon with the leak of proxy bandwidth resources in case of small network sizes, so that the system is more sensitive for the variation of parameters.

7. CONCLUSION

We approached the problem of NAT-traversal in P2P services by proposing NATCLOUD, a fully distributed approach that combines public nodes and cloud resources. We designed a cloud-assisted solution for NAT-traversal that preserves the effectiveness of P2P-based services in the real Internet deployments. Together with a mathematical model of the protocol we validated and evaluated the approach with PEERSIM simulations.

To demonstrate the applicability of our approach, we successfully integrated NATCLOUD with CLOUDCAST, an existing peer sampling service, without significant negative impact on its characteristics. The evaluation of the protocol proved NATCLOUD to be self-adaptive to the state of the network. In case the network parameters are regular and public nodes are abundant, the economical impact of NATCLOUD is practically zero. Instead, in the extreme case of very high churn rate or lack of public nodes, NATCLOUD successfully supports the NAT-traversal at a reasonable cost.

Following these results, we believe that NATCLOUD is a candidate to solve the problem of NAT-traversal for distributed applications and services.

8. REFERENCES

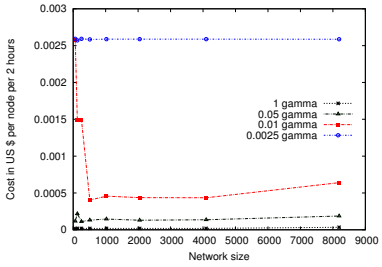
[1] Amazon simple storage service (Amazon S3). http://aws.amazon.com/s3/, [Online; accessed March-2015].

[2] J. Dowling and A. Payberah. Shuffling with a croupier: Nat-aware peer-sampling. In *Proc. of ICDCS'12*, pages 102–111. IEEE, 2012.

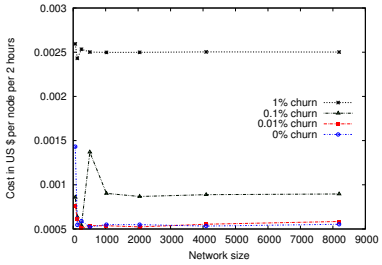
[3] N. Drost, E. Ogston, R. van Nieuwpoort, and H. Bal. ARRG: real-world gossiping. In *Proc. of HPDC'07*, pages 147–158. ACM, 2007.

[4] J. L. Eppinger. TCP connections for P2P apps: A software approach to solving the NAT problem. In *Technical Report CMU-ISRI-05-104*. Carnegie Mellon University, 2005.

[5] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-Peer



(a) Evaluation of γ influence; 0.01% churn rate, $LF_{max} = 1\%$



(b) Evaluation of churn rate influence; $\gamma = 0.05$, $LF_{max} = 1\%$

Figure 10: **NATCLOUD cloud cost estimation for 2 hours simulation per node for different network size and parameters.**

communication across network address translators. In *Proc. of ATC'05*, pages 179–192. USENIX, 2005.

[6] S. Guha and P. Francis. Characterization and measurement of TCP traversal through NATs and firewalls. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, Berkeley, CA, USA, 2005. USENIX Association.

[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3), Aug. 2007.

[8] H. Kavalionak, E. Carlini, L. Ricci, A. Montresor, and M. Coppola. Integrating peer-to-peer and cloud computing for massively multiuser online games. *Peer-to-Peer Netw. and App. (PPNA)*, pages 1–19, 2013.

[9] H. Kavalionak and A. Montresor. P2P and Cloud: A marriage of convenience for replica management. In *Proc. of IWSOS'12*, pages 60–71. Springer, 2012.

[10] A.-M. Kermarrec, A. Pace, V. Quema, and V. Schiavoni. NAT-resilient gossip peer sampling. In *Proc. of ICDCS'09*. IEEE, 2009.

[11] J. Leitão, R. van Renesse, and L. Rodrigues. Balancing gossip exchanges in networks with firewalls. In *Proc. of IPTPS'10*, page 7. USENIX, 2010.

[12] R. Mahy, P. Matthews, and J. Rosenberg. Traversal using relays around NAT: Relay extensions to session traversal utilities for NAT, 2010.

[13] A. Montresor and L. Abeni. Cloudy weather for P2P, with a chance of gossip. In *Proc. of P2P'11*. IEEE, 2011.

[14] A. Montresor and M. Jelasity. PeerSim: A scalable p2p simulator. In *Proc. of P2P'09*, pages 99–100. IEEE, 2009.

[15] S. Niazi and J. Dowling. Usurp: Distributed nat traversal for overlay networks. In *Distributed Applications and Interoperable Systems*, volume 6723 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011.

[16] A. Payberah, J. Dowling, and S. Haridi. Gozar: NAT-friendly peer sampling with one-hop distributed NAT traversal. In *Proc. of DAIS'11*, pages 1–14. Springer, 2011.

[17] A. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi. Clive: Cloud-assisted p2p live streaming. In *Proc. of P2P'12*, pages 79–90. IEEE, 2012.

[18] M.-A. Poulin, L. R. Maldague, A. Daigle, and F. Gagnon. NAT traversal in peer-to-peer architecture. In *Technical Report SCE-12-04*. Carleton University, Canada, 2012.

[19] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: A social-based peer-to-peer system: Research articles. *Concurr. Comput. : Pract. Exper.*, 20(2):127–138, Feb. 2008.

[20] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for nat (STUN), 2008.

[21] R. Roverso, J. Dowling, and M. Jelasity. Through the wormhole: Low cost, fresh peer sampling for the Internet. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10, Sept 2013.

[22] R. Roverso, S. El-Ansary, and S. Haridi. Natcracker: Nat combinations matter. In *Proc. of ICCCN'09*. IEEE, 2009.

[23] K. Sriram, D. Montgomery, O. Borchert, O. Kim, and D. Kuhn. Study of BGP peering session attacks and their impacts on routing performance. *IEEE Journal on Selected Areas in Com.*, 24(10):1901–1915, 2006.

[24] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.