Integration of Assisted P2P Live Streaming Service in Community Network Clouds

Mennan Selimi*, Nuno Apolónia*, Ferran Olid*, Felix Freitag*, Leandro Navarro*, Agusti Moll†, Roger Pueyo†, Luís Veiga ‡

* Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
{mselimi, apolonia, folid, felix, leandro }@ac.upc.edu

† Fundació Privada per a la Xarxa Lliure, Oberta i Neural Guifi.net, Barcelona, Spain
{agusti.moll, roger.pueyo}@guifi.net

‡ Instituto Superior Técnico (IST), INESC-ID Lisboa, Lisbon, Portugal
{luis.veiga}@inesc-id.pt

***Abstract*—Wireless community networks (CNs) are large-scale, self-organized and decentralized communication infrastructures built and operated by citizens for citizens. Community network cloud infrastructures have been recently introduced to run services inside the network, without the need to consume them from the Internet. We have developed a Linux-based distribution code-named Cloudy, which fosters the service deployment and automation in community network clouds. In this paper we present two ways provisioned by Cloudy to integrate the services and improve the users QoS in these clouds. First, we present a distributed service discovery mechanism that helps users with service quality metrics to choose the best service from a pool of instances. Second, we experiment with a live video streaming service deployed in CN environments, using more than 50 real CN nodes across Europe for the evaluation. Our analysis shows that tuning the vital parameters of this service as neighborhood peer selection strategies, and source node dispersion strategy, improves the video streaming QoS in the CNs. Our results indicate that both ways help the user to experience improved service performance. Automated service selection, needed once the number of micro service providers becomes larger, is the next step that can be built upon our results.**

***Index Terms*—community network cloud; p2p live streaming; service discovery**

I. Introduction

Wireless community networks (CNs) are large scale, self-organized and decentralized networks, which are deployed and maintained by their own users. Different from the traditional business-focused model of telecommunication operators, each user in community networks is owner of a part of the total infrastructure. Community networks are normally open, free and neutral. Local stakeholders develop community services, mainly local networking and Internet access [1]. There are several large community networks in Europe having from 500 to more than 28.000 nodes, such as Guifi.net[1] in Spain, AWMN[2] in Greece, Ninux[3] in Italy, Seattle Wireless[4] in USA and many more worldwide. Most of them are based on Wi-Fi technology (ad-hoc networks, IEEE 802.11a/b/g/n access points in the first hop, long-distance point-to-point Wi-Fi links for the trunk network), but also optical fiber links have become used in some areas. Guifi.net is considered to be the largest CN worldwide having today more than 28.000 operational nodes [2].

Resource sharing in CNs from the equipment perspective refers in practice to the sharing of the nodes bandwidth. This enables the traffic from other nodes to be routed over those of different node owners. This is done in a reciprocal manner which allows CNs to successfully operate as IP networks. The sharing of other services like storage, video streaming, VoIP, which is now common practice in today's Internet through cloud computing, hardly exists in CNs. We argue however that it can be made possible through clouds in community networks, i.e. *community network clouds*, a cloud deployment model in which a cloud infrastructure is built and provisioned for an exclusive use by a specific community of consumers with shared concerns and interests.

Deployment of services in CNs allows to offer resources and applications, which are of value for the users and meet their particular needs and interests. Among the services that are very appealing, P2P live streaming is an important candidate, as can be seen by the growing success and usage of commercial systems such as PPLive, SopCast. P2P live streaming systems allow to watch live streams such as events or television channels over a network, granting anyone to become a content provider.

To enable these types of services within CN nodes is very challenging, since community networks are diverse and dynamic networks with limited capacity of wireless links and often low-resource and cheap devices. Streaming applications, however, have high demands of bandwidth, they require low and stable latency and only withstand low packet loss.

Our motivation begins with the integration of a cloud-like system in community networks which gives users the opportunity to use services (e.g. video streaming) in their constraint devices (home gateways), without relying on the commercial clouds. Furthermore, we extend our motivation towards providing the service ease of usage and optimization of QoS on the challenging environment of CNs.

The contributions of this paper are the followings:

[1]http://guifi.net
[2]http://www.awmn.gr
[3]http://wiki.ninux.org/
[4]http://seattlewireless.net/

- We integrate the P2P live streaming service in the Cloudy distribution[5], and enable the automation and provision of this service in community network clouds.
- We implement a search service based on Serf[6] that allows the P2P live streaming service to be published and discovered by users in the community network cloud. Furthermore, we add a QoS-aware service selection algorithm that allows users to choose the best service from a pool of instances, according to network metrics.
- We evaluate the performance of PeerStreamer as a P2P live streaming service deployed over 55 geographically distributed real community network nodes. We then study the effects of different parameters of PeerStreamer on its performance in the community network environment.

The rest of the paper is organized as follows. Section II defines the community networks clouds and describes the Cloudy distribution. Section III explains the live streaming service and its integration in the Cloudy distribution. In section IV we explain the Serf service discovery implementation and we show how it is used to publish and discover live streaming services in community network clouds. Section V describes the experimental setup. In section VI we analyze and discuss our results. Section VII describes related work and section VIII concludes and indicates future research directions.

II. Community Networks Clouds

Our proposition is to deploy the PeerStreamer[7] service on cloud-based resources within community networks (CNs). These resources are given as CN clouds. CN users contribute computing resources to the cloud. The resources are therefore heterogeneous, geographically distributed, and often with resource constraints. Home gateways located in user homes can become cloud resources and they are integrated as Community Home Gateways (CHGs). From an administrative perspective, these CHGs are peer-to-peer infrastructures. The community network cloud we envisioned consists therefore of user-contributed infrastructures, such as home gateways, connected to the cloud in a peer-to-peer fashion, used for the collective provision of services that are of interest for the community.

Our model fits to the general cloud computing deployment categories. Besides public, private and hybrid models of cloud computing, a community network cloud differs from the others in that it is designed with a specific community in mind, and where costs and responsibilities are shared among the community members. Such community network cloud model assumes that cloud users can be classified into communities, where each community of users has specific needs in terms of services. We identify in CN such a community as a *micro-cloud*. A micro-cloud has a reduced number of nodes which are close as in Figure 1. This closeness in the context of CNs can be of technical and social nature. Cloud nodes within a micro-cloud announce their services and discover other nodes within the micro-cloud they belong to.

[5]http://cloudy.community/
[6]https://www.serfdom.io/
[7]http://peerstreamer.org/

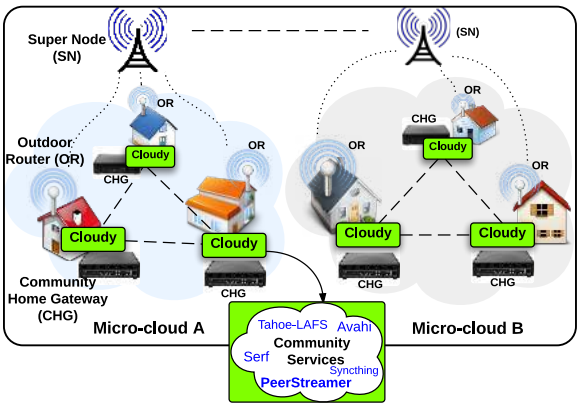

Figure 1. Community network cloud nodes, grouped into micro-clouds. The nodes of micro-clouds are spread on different locations inside the CNs, forming a meta cloud environment (community network cloud).

A. Cloudy

Cloudy is a Linux distribution containing open-source software services for our cloud platform for CNs. Cloudy also integrates platform services that were developed for CN users, including decentralized storage (a key-value store), video streaming, video-on-demand, search services and service discovery. Cloudy's main components can be considered a layered stack with services residing both inside the kernel and the user-level. Figure 2 indicates some of the already integrated types of services on the Cloudy CN distribution. An example of these services are the ones we consider in this paper, the video streaming service such as PeerStreamer, and the discovery service named Serf.

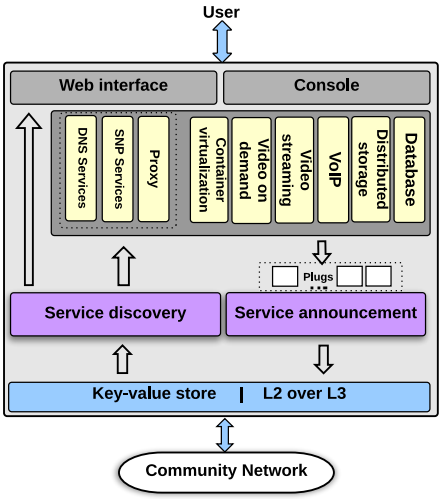

Figure 2. Cloudy architecture

1) Cloudy architecture: The internal architecture of the Cloudy distribution is depicted in Figure 2, inside the central rectangle. On the bottom part, the virtual Layer 2 over Layer 3 network provides the overlay to interconnect all the servers (nodes) in a micro-cloud. This overlay network is used in the service announcement and discovery processes, that respectively publish local information to the cloud and receive data from other cloud nodes.

Another special service module in the Cloudy instance is the

distributed announcement and discovery of services. On the lower layer it provides the mechanisms and the infrastructure to other services to publish their information all over the CN. This is a valuable resource to orchestrate the CN cloud itself as it allows room for self-discovery, management and federation of services and resources. On the user interaction layer, the DADS allows the end user to discover the available cloud services in the CN and decide which service provider to choose according to certain metrics (e.g. network round-trip time (RTT) to the services and number of hops).

The main block of Cloudy comprehends the CN services, stressing the important role of cloud services in the center of the diagram (see Figure 2). These services are the ones that benefit from or embrace the CN cloud environment to operate or offer a richer quality of experience (the list in the diagram is non-exhaustive, but mentions key services like distributed storage or different ways to reach video contents). Among them, virtualization is a special case. While other services focus on interaction and contents for the end user, provision of Infrastructure as a Service (IaaS) by means of virtual machines focuses on fostering the deployment of other services that run on top of this infrastructure.

III. Live Streaming Service

PeerStreamer is an open source live P2P video streaming service, and mainly used in our Cloudy distribution as the live streaming service example. This service is built on a chunk-based stream diffusion, where peers offer a selection of the chunks that they own to some peers in their neighbourhood. The receiving peer acknowledges the chunks it is interested in, thus minimizing multiple transmissions of the same chunk to the same peer. Chunks consists of parts of the video to be streamed (by default, this is one frame of the video). At the beginning of the streaming process, these chunks are all from the same peer (since only one peer is the source), then the source sends m copies of the chunks to random peers ($m = 3$ by default), creating an overlay topology with all peers [3] in order to exchange chunks between them. The whole architecture and vision of PeerStreamer is described in detail in [4].

A. PeerStreamer Assumptions and Notation

We call the community network the *underlay* to distinguish it from the *overlay* network which is built by PeerStreamer. The underlay network is supposed to be connected and we assume each node knows whether other nodes can be reached (next hop is known). We can model the underlay graph as:

$$G^{ug} = (S, L^{ug}) \tag{1}$$

where S is the set of super nodes present in community network and L^{ug} is the set of wireless links that connect them. This is the global level.

In the micro-cloud level we have a set of outdoor routers (OR) that are connected to each other in the same micro-cloud as shown in Figure 1,

$$G^{um} = (OR, L^{um}) \tag{2}$$

where OR is the set of outdoor routers present in the micro-clouds of the CNs and L^{um} is the set of wireless links that connects them.

The nodes of the underlay (connected to super nodes through outdoor routers) run an instance of the PeerStreamer and are called *peers*. Each peer P_i at time t chooses a subset of the other peers as a set of neighbours that are called $N_i(t)$. The peer P_i exchange video frames (chunks) only with peers in $N_i(t)$, and the union of all the $N_i(t)$ and the related links defines the network topology of the application, also represented as graph and called *overlay*. The overlay built by PeerStreamer is a directed graph:

$$G^{og}(t) = (P_{set}, L^{og}(t)) \tag{3}$$

where P_{set} is the set of peers and

$$L^{og}(t) = (P_i, P_j) : P_j \in N_i(t) \tag{4}$$

is the set of edges that connect a peer to its neighbours. The main difference between the overlay and the underlay is that the underlay is determined by the network topology, on which PeerStreamer does not have control, while the overlay is generated by PeerStreamer.

B. PeerStreamer integration in Cloudy

The version of PeerStreamer that is bundled with Cloudy, only features UDP streaming for video input, which is an acceptable transport protocol for video streaming. Therefore, we need to consider this fact in our stream provision. Either an online stream can be used (with the help of other applications) or a local video streamed to a local port is used. However, most of the video streams in the Internet do not use directly the network-level UDP protocol, instead it is more common to use an application-level protocol, such as RTSP/RTP[8]. In order to include PeerStreamer in Cloudy we choose the lightweight PeerStreamer version since we have low-resource machines in our community cloud deployment.

IV. Service Discovery

Cloud services in the context of CNs are built and operated in a decentralized way, and need a common place for both providers and users respectively, to publish their services and learn about their availability. In Guifi.net, the available network services are normally declared on the web page, by manually submitting the details (type of service and specific characteristics, location, IP address, terms of usage, etc.). The lack of automated methods for publishing services, and also for conveniently finding out which are the ones closest to the user, has led to a couple of drawbacks: not all the services are declared on the website (although they are announced by other means, like users mailing lists) and when a service is temporarily or permanently unavailable, it still appears on the website as *online* until it is manually removed from the list. In this section we show how we implement and use the automatic service discovery based on Serf to discover services such as PeerStreamer in Cloudy instances.

[8] http://www.ietf.org/rfc/rfc2326.txt

A. Serf Implementation

The distributed announcement and discovery of services (DADS) operates in parallel at both the global community network cloud level and at the micro-cloud level. On each of these two levels a different technological approach is used. Cloudy includes a tool to announce and discover services in the CN clouds based on Serf, a decentralized solution for cluster membership, failure detection, and orchestration. Serf relies on an efficient and lightweight gossip protocol to communicate with other nodes that periodically exchange messages between each other. This protocol is, in practice, a very fast and extremely efficient way to share small pieces of information. An additional byproduct is the possibility of evaluating the quality of the point-to-point connection between different Cloudy instances. This way, Cloudy users can decide which service provider to choose based on network metrics like RTT, number of hops or packet loss (Algorithm 1). The second level of DADS occurs in the micro-cloud, where a number of Cloudy instances are federated and share a common, private Layer 2 over Layer 3 network built with Getinconf[9]. At that level, Avahi[10] is used for announcement and discovery. Originally this solution was to be applied to the whole CN but as more Cloudy instances started to appear it became clear that the solution would not scale further than the tens of nodes as we explain in [5]. However, in the context of an orchestrated micro-cloud, it can be used not only for publishing cloud services but also other resources like network folder shares, etc.

When Serf finds different services (including services of the same type) we need to provide a QoS-aware service selection approach that will help users to choose the best quality of service among all instances. It is worth noting that a service with consistently good QoS performance is typically more desirable than a service with a large variance on its QoS performance. This would allow users to choose the best service available ranked according to some important community network parameters.

When a Cloudy client issues a find service request, Serf obtains the service list available and related service availability degree. Service availability may include many aspects to service i as S_i, we denote as A_{i1}, A_{i2}, A_{i3},...A_{ij},..., A_{im}, where m is the attribute number of each service. The services can have attributes as RTT, packet loss, throughput etc. We use W_{ij} to denote the importance weight of every attribute of service i, where j=1, 2, 3 ...,m and ϵ as a preference weight of the user for a given type of service. Taking into account this, the service availability can be described as A_i, in Equation 5. We specify also a service availability threshold λ, which denotes that if a service with A_i is greater than specified λ, then the service is available and it is added to the available service list set.

$$A_i = \sum_{j=1..m} (W_{ij} A_{ij}) - \lambda \quad (5)$$

[9]https://github.com/Clommunity/getinconf/

[10]https://avahi.org

Algorithm 1 ServiceSelection(S_i, W_{ij}, A_{ij})

```
// S_i ← service in the cloud, A_ij ← the jth attribute value
   of service i, W_ij ← the weight of importance degree, ε
   ← user preference weight, λ ← the availability threshold;
procedure S−SELECTION
    AvSet={};
    for each S_i in the Community Cloud do
        if S_i is in Micro-Cloud then
            W_i * ε where ε > 0 ;
        end if
        calculate A_i with equation (5);
        if A_i ≥ 0 then AvSet = AvSet U {(S_i,A_i)}
    end for
    sort(AvSet) order by descending;
end procedure
```

By default, Serf is used in Cloudy in order to simplify the process of service discovery for the users by utilizing the QoS-aware service selection algorithm (Algorithm 1).

V. Experiment Setup

For the experimental research, our main configuration includes geographically distributed CN nodes from Guifi.net in Spain, AWMN in Greece and Ninux in Italy. These nodes are co-located in either users homes (as home gateways, set-top-boxes etc) or within other infrastructures around each city. Nodes are deployed to use the wireless links of each community network that operate in the ISM frequency bands at 2.4 GHz and 5 GHz. The connectivity between CN nodes varies significantly. Two CNs (Guifi.net and AWMN) are connected on the IP layer via the FEDERICA[11] (Federated E-infrastructure Dedicated to European Researchers) infrastructure, enabling network federation. The nodes of Ninux CN in Italy are not connected to FEDERICA, therefore we experiment with them separately (without including other CN nodes). In our experiments the nodes from UPC (Technical University of Catalonia) are a subset of Guifi.net CN nodes which are distributed in our UPC campus in Barcelona. We use these nodes as a baseline in order to be able to better understand the effects of the network given by the statistical data gathered from the community networks.

In order to deploy the PeerStreamer application in a realistic community network cloud setting, we use the Community-Lab [1] infrastructure which is a distributed infrastructure provided by the CONFINE[12] project, where researchers can deploy experimental services and perform experiments in a real and production CN.

Our experimental evaluation is comprised of 55 physical nodes distributed across Europe, among the working nodes available from the three CNs. Most of the nodes are built with an Intel Jetway device equipped with an Intel Atom N2600 CPU (2 cores), 4GB of RAM and 120GB SSD and running a custom Linux OS (based on OpenWRT),[13] which makes them resource constrained devices at the edges of the network.

[11]http://www.fp7-federica.eu/

[12]https://confine-project.eu

[13]https://openwrt.org

Table I shows the number of nodes used in three community networks, their location and type of devices deployed.

In our experiments we connect a live streaming camera (maximum 512 kbps bitrate, 30 fps) to a local PeerStreamer instance which acts as the *source* for the P2P streaming. We choose as a source a stable node with good connectivity and bandwidth to the camera in order to minimize the video frame loss from the networked camera. The source is responsible for converting the video stream into chunk data that is sent to the peers. In the default configurations of PeerStreamer a single chunk is comprised of one frame of the streaming video. Also, the source PeerStreamer node sends three copies ($m = 3$) of the same chunk to the peers, meaning that only three peers receive the chunks directly from the source at a given time. Thus, each peer that receives the chunks exchange with other peers in order to form the P2P exchange network.

The evaluation metrics presented were chosen in order to understand the network behavior, quality of service and quality of experience. Thus, for network behavior section VI-A explains in details the network measurements obtained. On the quality of service side, we measure the number of chunks that are received by peers and the chunk loss percentage in order to understand the impact of the network on the reliable operation of this type of service. On the quality of experience, we gather statistical data from the chunks that are played out locally by each of the peers to understand the quality of the images that the edges show to the users. These metrics, show the impact of such networks when using streaming services while also guaranteeing the image quality that each node can display on average. Regarding the network interference issues of other users' concurrent activity which can impact the results of the experiments, we reference to [6] and is out of the scope of this paper.

Table I
NODES IN THE CLUSTER AND THEIR LOCATION

Nr. of nodes	Cat.	Location	Type
23	UPC	Barcelona, Spain	Physical nodes and VMs
8	Guifi.net	Catalonia, Spain	Physical nodes
12	AWMN	Athens, Greece	Physical nodes
12	Ninux	Rome, Italy	Physical nodes

A. Scenarios

To assess the applicability of PeerStreamer in CNs, the following describes a chosen scenario that reflects a use case of live video streaming in CNs. Also, we augment our findings with a scenario reflecting different parameters of PeerStreamer usage, in order to understand possible improvements of the overlay network created by the PeerStreamer instances. The parameters used in the scenarios are summarized in Table II.

For the first scenario we choose the default parameters of PeerStreamer and run in the challenging environment of CNs. One of the nodes, which has the best connectivity to the camera stream is chosen to be the source peer, while the rest of the available nodes will initially contact the source in order to enter the P2P network for chunk exchange. Since the Ninux group of nodes do not have connectivity in IPv4 to other CNs (they are not part of FEDERICA), we deliberately executed the experiment apart from the other CNs, in order to understand different CNs network behaviors. The experiment ran on this group was different because of the non-connectivity to the camera stream, therefore another solution was devised. We introduced a live TV streaming channel as the streaming source, transcoded to 512 kbps bitrate, 30 fps on average similar to the camera stream. However, this stream also included audio, which made the exchange of data between peers higher than the peers of other CNs. Each experiment is composed of 20 runs, where each run has 10 repetitions, and averaged over all the successful runs (90% of the runs were successful). In the 10% of the runs the source was not able to get the stream from the camera, so peers did not receive the data. The measurements we present consists of 3 weeks of experiments, with roughly 300 hours of actual live video distribution and several MBytes of logged data.

Table II
SUMMARY OF OUR SCENARIO PARAMETERS

Scenario	1 and 2
Total number of nodes	55
Groups of nodes	UPC, Guifi.net, AWMN, Ninux
Tests time-frame	T1 = 30m | T2 = 1h | T3 = 2h
Source 1 Send Rate (chps)	T1 = 31 | T2 = 32 | T3 = 31
Source 2 Send Rate (chps)	T1 = 55 | T2 = 55 | T3 = 49
Metrics	Peer Receive Ratio, Chunk Loss Chunk Playout, Neighborhood Size

We then establish three experiments shown for 30 minutes, 1 hour and 2 hours of continuous live streaming from the PeerStreamer source. This was done in order to gather statistical information within different time-frames and to use as initial step towards live events coverage on CNs. Other nodes were started at the same moment in time, 10 seconds after the source started, in order for the source to gather enough data to be able to exchange with the peers. This also allows the randomization of the nodes that the source PeerStreamer will first push the chunks to, and thus on all experiments the peers that begin receiving chunks from the source will be different (PeerStreamer overlay topology changes in every run of experiments). In all experiments we try to guarantee the number of nodes to remain constant. However, since we are dealing with a very dynamic and challenging environment, there is an issue of churn rate of nodes. This happens in the CNs because most of the nodes are connected wirelessly and their connectivity depends on many factors (such as weather, electric failures, router connectivity, among others). PeerStreamer for its own overlay performs operations to manage the peer churn rate by constantly updating each peer neighborhood, an important feature for the potentially unstable and dynamic nodes that we find in community networks.

For our second scenario, the evaluation performed includes the findings of different configuration parameters of PeerStreamer, which results in better quality streaming. This was done in order to understand the different behaviors of the PeerStreamer algorithms such that the overlay network that it constructs can be optimized. The different parameters chosen

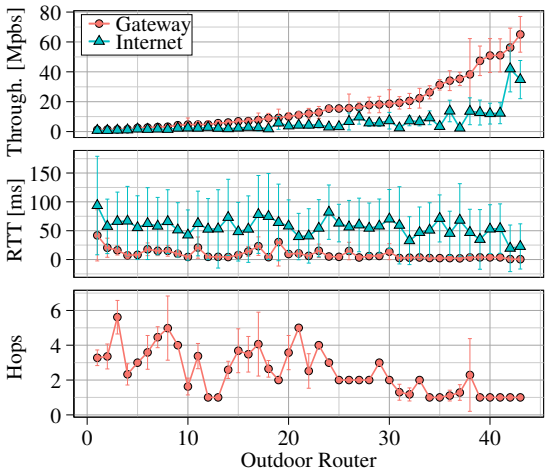

Figure 3. Average throughput and RTT to the gateway/Internet and number of hops to the gateway.

include sending different amount of copies of the chunks from the PeerStreamer source ($m = 5$, $m = 1$); keeping the best peers in the neighborhood in between topology updates of the overlay that PeerStreamer creates ($TopoKeepBest$); and the addition of the peers that can be selected to the neighborhood by extending the default RTT (10 ms) of the peer selection metric [4] to 20 ms .

VI. Results

A. Characterizing the Network Performance

Typically, CN users have an outdoor router (OR) with a Wi-Fi interface on the roof, connected through Ethernet to an indoor AP (access point) as a premises network. In Guifi.net where nodes are located, OSPF, BGP, BMX6 [7] or combination of them is used as a routing protocol. In AWMN and Ninux BGP and OSPF are mainly used between outdoor routers. Most of the super nodes (the ones routing the traffic between the different zones) are working in AP mode. The nodes (home gateways) where the PeerStreamer application is running are connected to these super nodes through their outdoor routers. A few super nodes are placed strategically on third party locations, e.g. telecommunication installations of municipalities, to improve the community network's backbone. In order to gain insight for network behavior in community networks we monitored the network for a period of 30 days.

Figure 3 shows the average throughput and RTT to the gateway (proxy) and the Internet, and the number of hops to the gateway obtained for every OR. The values are sorted by the throughput to the gateway. Standard deviation error bars are also given. Internet values are measured using a server located outside of Guifi.net. The figure also reveals that the throughput to the Internet and the gateway are not linearly correlated. The average throughput to gateway is 17.4 Mbps and to the Internet 6.3 Mbps. This is because one of the gateways in CNs has a better connection to the Internet. Thus, even if the throughput to the gateway is high, those nodes using the second gateway in other parts of the network have a low throughput to the Internet. Furthermore, it demonstrates that the RTT has a stronger correlation with the number of hops than the throughput (average RTT to the gateway is 9.26 ms and to the Internet 56.3 ms). Error bars reveals that some nodes have an average number of hops with noticeable deviations. This variability has two causes: change in the routes, and selection of a different gateway.

B. Scenario Results

Figure 4 depicts the amount of chunks on average the peers receive. Knowing that Source 1, sends out to the peers around 31 chunks per second (chps), we notice that the distant groups (Guifi.net and AWMN) in relation to the source, receive less chunks than the closer group (UPC), in relation to the source. This is because of the network impact on the delivery time of the chunks. Thus, more chunks arrive out of the time allotted, the farther the chunks have to travel. We also notice that the number of chunks received on average increases with longer time-frames, this occurs because the peers can gather more statistical information about each other and therefore update their neighboring peers accordingly, while securing a subset of peers in which they can rely on to receive the chunks in the time allotted to be displayed. We also show that on Ninux

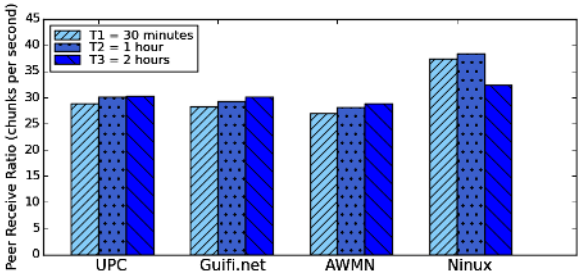

Figure 4. Average Peer Receive Ratio

side the amount of chunks received tends to be higher that of the other CNs. This is due to the fact that we use a different stream (Live TV channel stream), in which Source 2 sends around 55 chps instead (accounting with the added audio part of the stream). We also notice a drop of receiving chunks for longer times, because of the inherited instability of this group of nodes, where the loss of data is more constant/visible when dealing with longer times.

Figure 5 shows the average chunk loss for each group of peers. We can see that the loss is greater for shorter time-frames (loss in UPC 7%, Guifi.net 9% and AWMN 13%) and are amortized for longer time-frames (loss in UPC 2%, Guifi.net 3% and AWMN 7%). We also notice that distant groups (distant from the source stream) are more affected by the diminished rate of chunks received, which demonstrates the influence the network has to the amount of data that is lost (either by losses on the network or by not arriving on time to be displayed). As for the Ninux group, as previously mentioned, the network behavior is more volatile since there is a higher packet loss. Therefore, we notice that since Source 2 sends more chunks per second (around 55) than Source 1, the loss of chunks in the peers is greater than in other groups and in longer time-frames the network instability has a higher impact on the data exchanged (34% loss).

Figure 6 illustrates the quality (chunks played) of video offered on the peers side. The closer groups display more

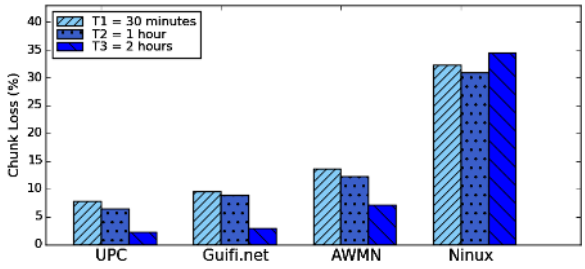

Figure 5. Average Chunk Loss

chunks, because the loss between farther nodes is greater than closer nodes and since the network plays a big role on the delivery of chunks. We also notice that the longer time-frames have on average a better chunk playout because more chunks arrive on time to be displayed (UPC 98%, Guifi.net 98%, AWMN 92%). For the Ninux group we see a more stable chunk playout for each of the time-frames, which means that since the network instability occurs during the whole evaluation the same amount of chunks (on average 71%) arrive to be displayed, also meaning that the network bandwidth/throughput between nodes (on average) is lower than on other CNs and remains constant over time.

Figure 7 demonstrates the chunk loss gathered during 30 minutes experiment, with different parameters given to the PeerStreamer. The parameters shown ($TopoKeepBest$, $RTT = 20ms$ and $m = 5$) have been selected in order to predict the behavior and improvements that PeerStreamer can have when executed in CNs. We notice that increasing the

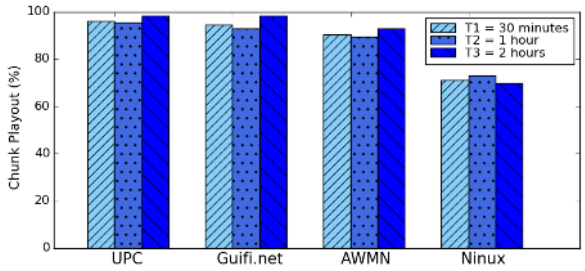

Figure 6. Average Chunk Playout

RTT for the overlay topology gives the peers higher probability to receive chunks in time and therefore decreasing the chunk loss in each of the groups. The other parameters have a higher impact on losing chunks, especially when the source only sends one copy ($m = 1$) of the chunks to peers (not shown in the figure). We also notice that keeping the best neighbors on topology overlay updates, lowers groups loss chunks (as in UPC case) that have nodes closer to each other, in which the selection of peers for exchanging chunks will have higher probability to choose the best nodes from previous topology updates. For the Ninux group we notice that when keeping the best nodes on topology updates there is a greater improvement (23% in loss, comparing with default parameters where we got 32% loss), because the probability of choosing the best nodes will be higher, since the nodes on this CN have worst connectivity. Also for Ninux, giving a RTT of 20 ms has mostly the same average as the previous experiments (with default parameters) since the nodes are farther apart (in RTT terms), meaning that there will be no significant changes in the neighborhood created for these peers. We also show that there is improvement when changing the number of chunk copies Source 2 sends to peers ($m = 5$). This is because of the resources that Source 2 has at its disposal, which makes it able to send more copies without losing bandwidth and computation time (against Source 1 as a low-power device); and also, since the network has more packet loss than in other CNs, flooding the network with more copies makes a higher probability for peers to be able to receive more chunks on time.

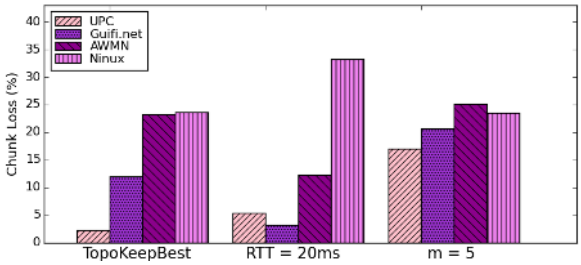

Figure 7. Average Chunk Loss with different parameters

C. Discussion

We started our evaluation by demonstrating the performance PeerStreamer has on CNs, with the default parameters, in order to understand what improvements can be achieved in CNs. We found that PeerStreamer neighborhood selection lacks accountability for network instability and therefore PeerStreamer can perform poorly in CNs. The metric for randomly selecting a subset of peers for the neighborhood reduces the probability to receive chunks in time, since peers can select the worst neighbors. This metric can be good for reducing time spent on initial costs however over time the CN selected peers need to be within either the best or with a greater range in RTT, as shown with different parameters scenario in Figure 7. We also found that while modifying the number of chunk copies that the source sends, can have beneficial results, guaranteeing that the chunks will travel to more nodes and be available to be traded in the P2P network over more peers. However, since the wireless links of CNs have a high diversity in bandwidth, this issue can arise and should be studied more thoroughly. Regarding the amount of data exchanged between peers we consider that in current wireless CNs and using P2P networks the high quality video streams (i.e., 1080p) are affected by the performance of the network links since more data or sizable data needs to pass through the network to the peers, and may even congest it. While using standard quality video streams, as shown in our evaluation the amount of loss is lower and more efficiently exchanged between peers in CNs.

Furthermore, by enhancing the performance of live video streaming, with the opportunity for users to choose the preferable services for them (based on the services' attributes such as RTT) can augment the probability for optimizing the QoE/S in these environments and therefore the combination of our contributions can achieve higher quality of service than an ad-hoc solution.

VII. Related Work

In terms of evaluating the performance of PeerStreamer in unreliable networks, the work of Baldesi et al. [8], [9] is the most relevant to our work. The authors evaluate PeerStreamer, a P2P video streaming platform, on the Community-Lab, the wireless community network (WCN) testbed of the EU FIRE project CONFINE. Their experiments highlight the feasibility of P2P video streaming, but they also show that the streaming platform must be tailored ad-hoc for the WCN itself to be able to fully adapt and exploit its features and overcome its limitations. However they evaluated with a limited number of nodes (16 Guifi.net nodes), which were located in the city of Barcelona and they do not use live video stream. A recent PhD dissertation [10] includes some discussion on P2P streaming on WCNs, but does not elaborate on live streaming, but consider streaming of Video on Demand (VoD) retrieval.

Another work [11] studies different strategies to choose neighbours in a P2P-TV system (PeerStreamer). The authors evaluate PeerStreamer on a cluster and on Planetlab. In wireless networks PULLCAST [12], is a cooperative protocol for multicast systems, where nodes receive video chunks via multicast from a streaming point, and cooperate at the application level, by building a local, lightweight, P2P overlay that supports unicast recovery of chunks not correctly received via multicast.

The impact of uncooperative peers on video discontinuity and latency during live video streaming using PlanetLab is studied in [13]. The paper in [14] investigates the impact of peer bandwidth heterogeneity on the performance of a mesh based P2P system for live streaming.

In our work we emphasis on studying the video streaming applications in a real deployment scenario within community networks, in order to understand their performance and operation feasibility under real network conditions. Furthermore, we automate the PeerStreamer deployment using Cloudy distribution.

VIII. Conclusion and Outlook

A distributed community network cloud was presented, as the environment for the experimental evaluation of PeerStreamer, a P2P live streaming service, integrated through the Cloudy distribution. An important aspect for the ease of usage of community network clouds is the automatic announcement of services, such as PeerStreamer, and their discovery by other cloud nodes. A service announcement mechanism based on Serf was used to allow end users to discover active PeerStreamer instances in the cloud and join a live streaming event. Furthermore, we designed an algorithm to help users choose the service with the better QoS available to them. This was our contribution done on the users perspective, which improves the underlay network. The service discovery and the ease of usage that the Cloudy environment provides for end users, is considered an important element that envisages the users to participate in the streaming service.

On the overlay network level our goal was to have a feasible system that can utilize the resources scattered on CNs in order to achieve a live video streaming service. The part of PeerStreamer that can be modified is the construction of the overlay network. Our evaluation showed that using PeerStreamer with the default settings can achieve lower rating in terms of QoS in the CN environment where the network instability is prominent. We showed that in different CNs the results obtained in terms of loss of data between peers is distinctive. For this reason, we augmented our findings by running PeerStreamer with different configuration of parameters so that we can understand the best behaviour that PeerStreamer can provide to its users. Our evaluation showed that modifying the number of chunk copies that the source sends to peers and modifying the neighbourhood selection policies such as metrics for peer selection as RTT and keep best peers ($TopoKeepBest$) can have beneficial results for live video streaming in the high diversity environment of a CN.

Based on the successful operation and performance of PeerStreamer in community networks and the service discovery mechanism provided through community network clouds, our experimental deployment of these applications should now transform into a permanently available cloud-based streaming service used by community network participants.

Acknowledgement

This work was supported by the European Framework Programme 7 FIRE projects CONFINE (FP7-288535), CLOMMUNITY (FP7-317879), by the UPC-BarcelonaTech and the Spanish Government through the Delfin project, TIN2013-47245-C2-1-R.

References

[1] B. Braem *et al.*, "A case for research with and on community networks," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 68–73, Jul. 2013.

[2] "Guifi.net: Open, Free and Neutral Network Internet for everybody." [Online]. Available: http://guifi.net

[3] R. Birke *et al.*, "A delay-based aggregate rate control for p2p streaming systems," *Computer Communications*, vol. 35, no. 18, pp. 2237 – 2244, 2012.

[4] R. Birke, E. Leonardi, M. Mellia *et al.*, "Architecture of a network-aware p2p-tv application: the napa-wine approach," *Communications Magazine, IEEE*, vol. 49, no. 6, pp. 154–163, June 2011.

[5] M. Selimi, F. Freitag, R. P. Centelles, A. Moll, and L. Veiga, "TROBADOR: Service discovery for distributed community network micro-clouds," in *29th IEEE International Conference on Advanced Information Networking and Applications (AINA 2015)*, Mar. 2015.

[6] L. Cerdà-Alabern, A. Neumann, and P. Escrich, "Experimental evaluation of a wireless community mesh network," in *Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '13)*. New York, NY, USA: ACM, 2013, pp. 23–30.

[7] A. Neumann, E. Lopez, and L. Navarro, "An evaluation of bmx6 for community wireless networks," in *IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2012, pp. 651–658.

[8] L. Baldesi, L. Maccari, and R. Lo Cigno, "Live p2p streaming in communitylab: Experience and insights," in *13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*, June 2014.

[9] L. Baldesi, L. Maccari, and R. Lo Cigno, "Improving P2P Streaming in Community-Lab Through Local Strategies," in *10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Larnaca, Cyprus, October 2014, pp. 33–39.

[10] A. Alasaad, "Content sharing and distribution in wireless community networks." Univeristy of British Columbia, PhD Thesis, 2013.

[11] S. Traverso *et al.*, "Experimental comparison of neighborhood filtering strategies in unstructured p2p-tv systems," in *IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*, Sept 2012, pp. 13–24.

[12] A. Russo and R. Cigno, "Pullcast: Peer-assisted video multicasting for wireless mesh networks," in *10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, March 2013.

[13] J. Oliveira *et al.*, "Can peer-to-peer live streaming systems coexist with free riders?" in *P2P'13*, 2013, pp. 1–5.

[14] A. Couto da Silva, E. Leonardi, M. Mellia, and M. MEO, "Exploiting Heterogeneity in P2P Video Streaming," *IEEE Transactions on Computers*, vol. 60, p. 667–679, May 2011.