Stimulating Participation in Wireless Community Networks

Elias C. Efstathiou, Pantelis A. Frangoudis, and George C. Polyzos
Mobile Multimedia Laboratory, Department of Computer Science
Athens University of Economics and Business
Athens, Greece
{efstath, pfrag, polyzos}@aueb.gr

***Abstract*—Wireless Community Networks (WCNs) are wide-area wireless networks whose nodes are owned and managed by volunteers. We focus on the provision of Internet access to mobile users through WCN-controlled wireless LAN access points (APs). We rely on reciprocity: a person participates in the WCN and provides 'free' Internet access to mobile users in order to enjoy the same benefit when mobile. Our reciprocity scheme is compatible with the distinctive structure of WCNs: it does not require registration with authorities, relying only on uncertified free identities (public-private key pairs). Users sign digital receipts when they consume service. The receipts form a receipt graph, which is used as input to a reciprocity algorithm that identifies contributing users using network flow techniques. Simulations show that this algorithm can sustain reciprocal cooperation. We have implemented our algorithm to run on common APs.**

I. INTRODUCTION

Wireless Community Networks (WCNs) are wide-area wireless networks whose nodes are owned and managed by volunteers. WCNs include Seattle Wireless [1], NYCwireless [2], Consume (London, UK) [3], and the Athens Wireless Metropolitan Network (Athens, Greece) [4]. WCNs offer various services to their participants and to the public. In this paper, we focus exclusively on the provision of Internet access to mobile users through WCN-controlled wireless LAN access points (WLAN APs). We want to stimulate participation in WCNs so that more WLAN APs become available. WLAN-enabled mobile phones are now on the market [5, 6], and WCNs could complement 3G networks in metropolitan areas.

We will assume that all WCN-controlled WLAN APs are connected to metered DSL connections. This assumption does not hold true for many WCNs that bypass wired ISPs and interconnect their nodes with point-to-point wireless links. We anticipate, however, that—with a further drop in DSL prices—our assumption will hold true for metropolitan WCNs. We will also assume that ISP contracts permit the not-for-profit sharing of DSL connections with nearby mobile users over a WLAN. Sharing-friendly ISPs exist today [7, 8].

Why should volunteers set up WLAN APs and share their metered connections with nearby mobile users? Altruism is one answer. However, we assume that there are not enough altruists to allow WCNs to rival 3G networks in citywide coverage, or to make WCNs robust to failures through redundancy.

This research is supported in part by the project "Mobile Multimedia Communications" (EP-1212-13), funded by the program "Herakleitos—Fellowships for Research at the Athens University of Economics and Business," which is co-financed by the Greek Government and the EU through the program "EPEAEK II."

Instead of altruism, we rely on *reciprocity*. The idea is that a person participates in the WCN and provides free Internet access to mobile users—who also participate in the WCN—in order to enjoy the same benefit when mobile; mobile users who do not participate in the WCN are excluded from the service. We assume that a mobile user benefits when using a foreign WLAN/DSL connection for free, and that this benefit offsets the cost of allowing other mobile users to access his or her home WLAN/DSL connection. Our reciprocity scheme encourages participants to have consumption–contribution ratios that are near 1:1. By *consumption* we refer to the volume of Internet traffic a mobile user transfers through foreign WLAN/DSL connections, and by *contribution* to the volume of Internet traffic the user relays for others through his or her home WLAN/DSL connection. In our scheme, individuals can also group in *teams* whose *aggregate* ratio is near 1:1.

The reciprocity scheme that we will propose is compatible with the distinctive structure of WCNs. For example, we do not rely on authorities that punish or reward because WCNs rarely have a powerful centralized authority. Participants do not register with a Trusted Third Party and are free to choose their own system identifiers (IDs). We do not restrict the rate of ID generation by requiring, for example, that IDs contain solutions to cryptographic puzzles. Participants may use an unlimited number of free (and unforgeable and unique) IDs. This important requirement makes our reciprocity enforcement problem different from similar problems in the area of ad hoc networks where exactly one (unforgeable and unique) ID per node is assumed. Furthermore, we assume that participants can modify the modules that implement the reciprocity protocol and we do not rely on tamperproof hardware. We assume that participants do not know or trust most of the other participants, but also that participants can form teams with other participants that they do trust. Finally, two or more teams can collude.

We will present the design, evaluation, and implementation of a protocol that encourages cooperation in WCNs and conforms to the requirements stated above. We call it *Peer-to-Peer Wireless Network Confederation* (P2PWNC) protocol. We have already presented a basic P2PWNC reciprocity algorithm [9] and have used evolutionary simulations to evaluate it [10]. Here we extend our previous results with (1) an improved collusion-proof reciprocity algorithm, (2) a centralized version of our scheme—with minimal demands on the center—that could be practical for certain WCNs, (3) comprehensive

simulation results for the performance of our algorithms in centralized and decentralized mode, (4) specification of our protocol, and (5) performance results from our implementation on the Linksys WRT54GS WLAN AP.

A. Overview

Our scheme works as follows. We assume that WCN participants divide into *teams* of a few tens of *members* each. Members of the same team must know and trust each other. Teams own and manage a number of *APs* (WLAN access points, connected to DSL lines) at locations throughout the city. We assume that aggregate team consumption rates are homogeneous. This is an important assumption that we will return to in Section VII. We gave the precise meaning of *consumption* and *contribution* before; we say now that a *team* consumes when one of its members uses an AP of another team, and *contributes* when a member of another team uses an AP of this team. The objective of our reciprocity protocol is to encourage teams to match their consumption with at least an equal amount of contribution. Free-riding teams that contribute much less than they consume will find it hard to obtain service. And only short-term history is important: teams must contribute continuously in order to be able to consume continuously.

Members sign digital *receipts* when they consume service from another team. The receipts form a *receipt graph*, which is used as input to a *reciprocity algorithm* that identifies contributing teams using *network flow* techniques. Simulations show that this algorithm can sustain reciprocal cooperation. Receipts are stored either in a *central server*, or they are distributed among multiple *team servers* with a *gossiping protocol*. Two (non-colluding) teams never exchange information over the Internet: team servers are not accessible to outsiders. Different teams interact only when a member is physically close to the AP of another team and requests its services. We will show that gossiping is beneficial then to both teams.

The receipt graph may contain fake receipts—the result of collusion among two or more teams, or of a *Sybil attack*[1]. All member IDs and team IDs are unique public/private key pairs. We assume that it is computationally infeasible to break the digital signature scheme used to sign receipts and *member certificates* (member certificates are signed by the *leader* of each team). We do not rely on a Public Key Infrastructure and we do not assume that teams know the IDs of other teams.

We use "peer-to-peer" in our protocol's title; however, no *searching* for resources occurs. We assume that members, when mobile, try to access any AP that is close to their current location. Also, we assume that members, if granted access, will tunnel all their traffic to a trusted Internet gateway, so we are not concerned with the security of the wireless link or with APs that spy on foreign traffic. Our *receipt generation protocol* (Section IV-D) ensures that hijacking of the wireless session cannot occur.

We wanted to design a reciprocity protocol that is simple to implement directly on WLAN APs. Our implementation (Section VI) that runs on the Linksys WRT54GS AP proves this.

In our scheme, WCNs are meant to complement 3G networks in metropolitan areas. The growth of WLAN/DSL deployments makes our scheme relevant—at least as long as DSL rates remain lower than 3G rates. We support that our protocol is simple to implement and can achieve its basic goal, which is to stimulate participation in WCNs while respecting their open and self-organized nature.

B. Organization of the Paper

The remainder of this paper is organized as follows. Section II reviews and compares with related work. Section III specifies system models, including the trust model and the uncooperative behavior model, and presents the system entities involved. Section IV describes our reciprocity algorithm and communication protocols. Algorithm performance is evaluated via simulations in Section V. Section VI presents our reference implementation of the P2PWNC protocol on the Linux-based Linksys WRT54GS AP, along with measurements of its performance. We discuss several important issues in Section VII, where we also conclude the paper.

II. Related Work

A. Distinctive Characteristics of our Approach

The area of economics-informed design of peer-to-peer systems attracts a lot of research interest. Packet relaying in ad hoc networks is a standard example of a function that requires sustained cooperation among independent and selfish peers. At first, the Wireless Community Network context seems different only in two details: (1) there is only one intermediate relaying node (the WLAN AP), which is also connected to a fixed power supply and to the Internet; (2) the functions of provider and consumer are split: APs are providing and mobile users are consuming. A P2PWNC "peer" is a distributed entity comprising APs and members that are all part of the same team.

Two additional requirements, however, go beyond what is usually encountered in the literature. First, peers do not register with authorities and our peer (team) IDs are free; a peer may use multiple IDs if it is in its interest. This makes recognition difficult. We do not impose other artificial start-up costs on new IDs by requiring, for example, that IDs contain solutions to cryptographic puzzles. To bootstrap our reciprocity algorithm, we require some contribution from a new team/ID *before* this new ID can start consuming; therefore, IDs may be free, but new peers must contribute to the system *first*. Our bootstraps are designed to be relatively cheap, however, because they may have to happen often: since we rely only on *short-term* history, if a peer (team) stops participating for some time it will have to go through the equivalent of the bootstrap phase upon reappearing, irrespective of a change in ID.

Second, we do not assume that it is incentive compatible to perform distributed accounting tasks. One standard use of accounting is to detect and punish free-riders. However, *second-order free-riders* [12] may not be willing to spend resources in detecting or punishing free-riders. The dilemma of second-order free-riders is: "why help others in punishing free-riders and improving system performance when I can free-ride on the efforts of others that do that?" This is similar to the dilemma of free-riders: "why help others and improve system performance when I can free-ride on the efforts of others that

[1] Sybil attack [11]: the creation of multiple identities per real entity; a fundamental problem in open and self-organizing electronically mediated communities without identity-certifying authorities. Sybil attacks can invalidate any number of system assumptions and make collusion-based attacks much easier.

do that?" We either need an accounting system to punish second-order free-riders (which may lead to third-order free-riding, and so on), or all distributed accounting tasks must have an obvious and direct benefit to the peer that performs them. Also, "punishments" should be limited to refusals to cooperate.

B. Literature Survey

Catch [13], CONFIDANT [14], and CORE [15] are examples of incentive schemes for multi-hop wireless networks that assume nodes: (1) have exactly one ID, (2) do not collude, and (3) are willing to perform distributed accounting tasks (cooperating nodes also cooperate in punishing free-riders "for the good of the community"). Sprite [16], like our P2PWNC protocol, is a receipt-based protocol that assumes nodes can collude; however, unlike P2PWNC, it requires a powerful Credit Clearance Service that determines the real-world charge and credit to each node. Follow-on work [17] still relies on centralized clearing. The Nuglet approach [18] requires that nodes have tamper-resistant security modules, manufactured by a limited number of trusted manufacturers who also cross-certify each other's public keys. A formal model for cooperation in *static* multi-hop wireless networks based on game theory and graph theory is presented in [19]. Simulation results show that, in practice, the conditions for cooperation without incentives are virtually never satisfied, and that cooperation needs to be encouraged.

Our reciprocity algorithm is a generalization of the algorithm we presented in [9, 10], and an extension to the *maximum-flow* decision function presented by Feldman *et al*. [20] (itself inspired by older work on authentication metrics [21, 22]). The context studied by [20] is similar to ours: there are no identity-certifying authorities, IDs are free, and collusion is possible. However, colluders in [20] are *naïve*; they only engage in false trading with each other and never contribute to anyone outside the colluding group. Our reciprocity algorithm is robust against less naïve colluders who may contribute to others outside their group. A follow-on work of [20] is [23], where the effect of free IDs on the cooperation strategies studied by Axelrod (*Tit-for-Tat*) [24] and Nowak and Sigmund (*Image*) [25] is analyzed. Their theoretical results confirm the result of Friedman and Resnick [26] that free IDs incur a social cost. Our experimental results in this paper also show this fundamental problem.

The maximum-flow decision function [20] that we extended belongs to a class of cooperation strategies that rely on *multi-way exchanges*, a generalization of 2-way exchanges and a special (cyclical) case of *indirect reciprocity* [27]. Multi-way exchanges have been adopted as an incentive mechanism for file sharing [28], storage sharing [29], and in our own previous work [9, 10]. However, our algorithm in [9, 10] was simple: it did not consider receipt weights and it could not discourage peers that consumed more than they contributed if their consumptions occurred in quick succession and at different peers.

Earlier work on partially observable distributed graphs includes [30, 31]; reference [30] does not focus on incentive issues; reference [31] addresses the incentives of nodes: they *store* accounting information only if it is in their interest; however, reference [31] also assumes that nodes are simply willing to *share* their stored accounting data with others.

Micropayment-based incentive techniques for peer-to-peer systems include PPay [32] and Karma [33]. PPay requires a central authority to generate currency and to check for double spending. Karma requires other peers to keep track of a peer's account balance, and assumes that distributed accounting is incentive compatible. Also, Karma is susceptible to the Sybil attack: a peer can repeatedly join the system and obtain new start-up funds each time. The cryptographic puzzle that new entrants must solve limits only the *rate* of new ID generation.

Reference [34] is a work with the same objective as ours (fuelling WLAN deployment), and focuses also on QoS. Wireless ISPs have multilateral roaming contracts and must register with a central authority that maintains reputation records, which are updated with QoS reports submitted by the roamers.

Several problems faced by system designers who consider incentive issues in their designs are discussed in [35, 36].

III. System Models and Design

In this section, we specify the trust model and the uncooperative behavior model for the Peer-to-Peer Wireless Network Confederation (P2PWNC) protocol, and describe the relevant P2PWNC entities. Detailed algorithms follow in Section IV.

A. Entities and Trust Relationships

We consider citywide Wireless Community Networks (WCNs) comprising tens of thousands of Wireless LAN access points (WLAN APs). Each AP is connected to an always-on metered DSL line. The APs and DSL lines are owned by different WCN participants. APs provide WLAN coverage to the publicly accessible areas that surround the households of their owners. We do not assume the existence of a WCN backbone with point-to-point wireless links.

1) Teams: WCN participants divide into teams. The members of a team must know and trust each other so we anticipate that teams will be small, with a few tens of members. Team members may reside close to each other, but a team can also be distributed across the city. By *team consumption* we refer to the aggregate consumption of its members, meaning the aggregate volume of Internet traffic that its members, when mobile, transfer through APs that belong to other teams. We assume that—even though individual consumption rates may vary—the aggregate team consumption rates are homogeneous.

The idea behind using teams—and enforcing reciprocity between teams and not individuals—is that individual WCN participants may reside in areas where there are not many mobile users to serve. Because our reciprocity algorithm encourages consumption–contribution ratios near 1:1, these participants would receive little value from our scheme. However, if we allow them to team with other participants who trust them and who contribute a lot, then a team as a whole could still maintain a consumption–contribution ratio near 1:1. We do not specify how members settle their different consumption and contribution rates within a team. Our protocol allows teams to monitor the consumption and contribution rates

of their individual members; because there is no anonymity within a team and teams are small, we assume that teams will use social pressure or intra-team monetary transfers to balance their aggregate consumption-contribution ratio, aided also by careful selection of their members. In general, however, our scheme does pose problems for those who reside in neighborhoods that are rarely visited by mobile users.

A team requires a *team founder*. The role of the founder is to generate the *team identifier,* which is a unique public key whose corresponding private key is kept secret by the founder. The founder uses this private key to sign a *member certificate* for each team member. Each member certificate binds a unique member identifier (ID) to the team's ID (public key). A member ID is also a unique public key, whose corresponding private key is known only to the member who generated the key pair. We do not require a Public Key Infrastructure—team public keys remain uncertified. Using their certificates, members request service in the name of their team. We assume that teams do not trust most other teams, and that they may not even know of the existence of other teams.

2) Receipts: A receipt is evidence that WLAN service was contributed and consumed. Receipts are signed by team members when they use an AP of another team, and are transferred to the contributing team during the WLAN session. We will describe our *receipt generation protocol* in Section IV-D. This protocol ensures that none of the two parties is at a significant disadvantage during the exchange. Fig. 1 shows the format of receipts. Receipts contain: (1) the public key of the contributing team, (2) the certificate of the consuming member, which contains the public key of the consuming team, (3) a *timestamp* noting the start time of the WLAN session, (4) a *weight* noting the volume of traffic the member transferred through the AP during the WLAN session, and (5) the member's signature, that is, a hash of the above fields encrypted with the consuming member's private key.

We can verify the two signatures that the receipts contain (the team signature on the member certificate and the member signature on the receipt) using information on the receipt itself (the team public key and the member public key). If both verifications succeed we are sure that a team with a given ID (found on the member certificate) has authorized the member who signed the receipt to consume in the team's name.

Receipts form a logical *receipt graph*. The vertices of this graph are team IDs (public keys) and the weighted directed edges point from the consuming ID to the contributing ID. An edge's weight is equal to the volume of traffic the source has consumed from the destination, that is, the edge weight is equal to the sum of the weights of the corresponding receipts. The direction of the edge denotes the "owes to" relation.

3) Receipt Servers: After a WLAN session is over, the contributing team will store the newly generated receipt in a receipt server. Our protocol supports two types of servers: a *central server*, and multiple *team servers.* These correspond, respectively, to a *centralized* and a *decentralized mode* of operation. We compare the two modes of operation in our simulations (Section V). Our reference implementation (Section VI) supports both modes. The centralized mode requires that all teams of a WCN agree on the same central server—which may be impossible. Decentralized mode does not have this problem.

In centralized mode, all APs communicate through the Internet with the central server. In decentralized mode, each AP is configured to connect to its team server. In our reference implementation the team server is co-located with one of the team APs—teams do not have to maintain another dedicated device. The APs store the receipts they earn in the server. When a receipt server is full, it deletes the receipt with the oldest timestamp in order to store a new one. In decentralized mode, a team's members may contact their team server in order to receive an *update* with the latest receipts, which they will use when *gossiping* (see Section IV-C). We assume that servers are always online; however, this is not crucial (members can use stale data). Fig. 2 shows both modes of protocol operation.

B. Uncooperative Behavior Model

We assume that members are selfish but not malicious. However, we do not consider selfish MAC-layer behavior. We do not consider localized Denial-of-Service (DoS) attacks like physical layer jamming. Also, we do not consider Internet DoS attacks on the receipt servers; the central server is the obvious target, however, even team servers can be attacked if their Internet address is revealed. However, an important characteristic of our protocol is that it does not require that different teams communicate through the Internet, so the probability of DoS attacks due to publicized IP addresses is lowered.

We assume that standard cryptography works, and that attackers cannot steal private keys in order to break our digital signature schemes. Such private keys are contained in the WLAN clients used by members (because these clients sign receipts). There is also one private key per team—kept secret by the team founder—that is used to sign member certificates. APs do not store private keys, only their team's public key, which they advertise during receipt generation (Section IV-D).

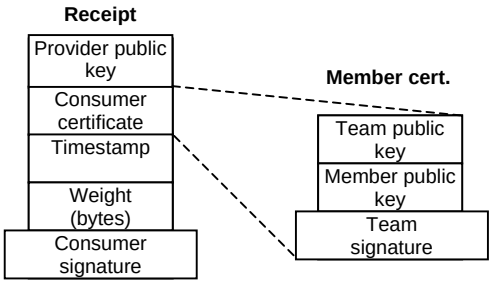

Figure 1. Receipt format. Receipts are self-verifiable signed documents that report a completed WLAN session in which a total of *weight* bytes was relayed. The WLAN session had started at the time encoded in *timestamp.*

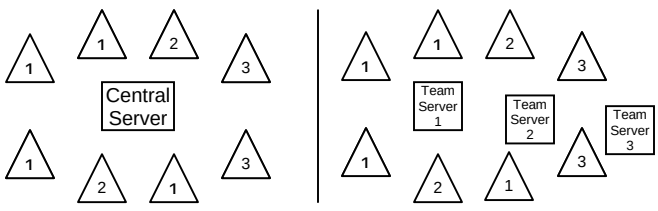

Figure 2. Centralized and decentralized mode. In decentralized mode, team APs communicate with their own team server (which may also be an AP).

We assume that teams will attempt to cheat the reciprocity algorithm and contribute less than they consume. To do that, naïve teams will simply refuse all service requests. Less naïve teams will engage in *false trading*, that is, they will generate fake receipts. They can do this either on their own, by creating fake team IDs in a Sybil attack, or in collusion with other teams. Even less naïve teams can combine false trading with (some) actual contribution. In terms of our receipt graph, an attacker *cannot* add a new outgoing edge or change the weight of an existing edge that starts from a vertex that corresponds to a real team with which the attacker is not colluding—this would require access to the private key of the team, or to the private key of one of the team's members, which we assumed is impossible. Fig. 3 shows potential attacks on the graph.

IV. Algorithms and Protocols

A. Reciprocity Algorithm

The reciprocity algorithm we will present is a generalization of our algorithm in [9, 10]. It provides a YES/NO answer to the question "should team *P* provide service to team *C*?" *P* is short for *provider* and *C* is short for *consumer*. The algorithm uses the receipt graph as input. In centralized mode, the central server executes the algorithm on its view of the graph; APs send their queries to the central server; queries contain the IDs of *P* and *C*. In decentralized mode, APs send their queries to their team servers. Our objective with this algorithm is to encourage cooperation by cooperating only with contributors.

The decentralized algorithm in [9,10] does this by searching the receipt graph for *directed paths* starting from *P* and ending at *C*. If such a path is found, the algorithm decides that *P* owes service to *C* (directly, or *indirectly*, that is, by owing to teams that directly or indirectly owe service to *C*), and, therefore, *C* is a contributor. To maintain a balance between contribution and consumption, *P* never considers the receipts of this path in the future, and so *C* cannot repeatedly take advantage of the same path to *P*. Two limitations of this algorithm are: (1) it does not consider the weights on the receipts; (2) receipt discarding is local to *P*, which means that *C* can potentially take advantage of the same receipts elsewhere (if they do not expire [9, 10]). The idea behind finding a path from *P* to *C* is to prevent false trading attacks. If *P* does not find such a path, it cannot trust any of the other receipts (or chains of receipts) that point to *C*: they could all be the result of collusion or of a Sybil attack.

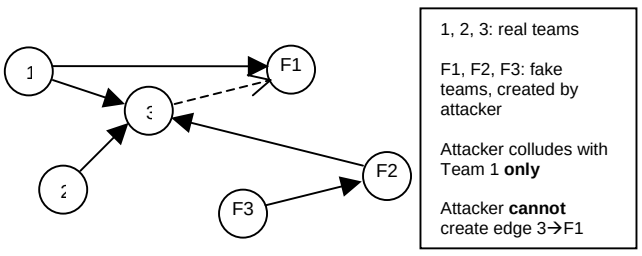

Figure 3. Receipt Graph. Attackers can create fake vertices (teams) and edges (receipts), but cannot create outgoing edges from real teams that are not in collusion with the attacker because they do not have access to the private signing key of the team, nor to the private signing key of one of its members.

Feldman *et al.* [20] presented a decision function that is based on *maximum flow* (maxflow), which does not have the two limitations mentioned above. Reference [20] shows via analysis and simulation that this decision function is robust to collusion and can encourage cooperation in communities with free IDs. In this paper, we adopt the algorithm from [20], and extend it, making it robust against less naïve colluders who combine false trading with (some) real contribution.

1) Basic Algorithm: The decision function in [20] is probabilistic. Peer *P* provides service to *C* with probability:

$$p = \min\left(\frac{mf(P \to C)}{mf(C \to P)},\ 1\right) \tag{1}$$

where $mf(\cdot)$ is the result of the maxflow algorithm for a pair of vertices. In [20], the vertices represent peer IDs and the edges represent amount of service consumed. In our context, the graph vertices are team IDs and the edges show the volume of transferred traffic the source team ID "owes to" the destination team ID; weights are measured in bytes. We also define that if the denominator in Eq. (1) is 0, *P* provides service only if the numerator is strictly positive.

This algorithm takes indirect consumption of the prospective consumer into account (in the denominator), balancing consumption and contribution without the need for receipt discarding. The simulations in [20] show that peers who adopt this function in an environment where (naïve) collusion is possible perform better (in a manner to be defined in Section V) than peers who follow other strategies like *Always Cooperate* or *Always Defect*. The algorithm is robust to (naïve) collusion: non-contributors cannot appear to be contributing if they falsely trade with non-contributors only. See also Fig. 4.

The maxflow function is, however, vulnerable to another attack. We give an example: Let *C* be a contributing team; however, instead of consuming using ID *C*, each time *C* consumes it uses a fresh ID C_i, $i > 0$, which *C* never reuses. This simple Sybil attack does not benefit *C* because $mf(\cdot \to C_i)$ is 0 for all *i* (because there are no receipts pointing *to* the fresh ID). Therefore, *P* will refuse to cooperate with C_i.

However, if *C* falsely trades with the new ID and creates a new $C \to C_i$ edge, with weight *w*, then, if $w > mf(P \to C)$, we have that $mf(P \to C_i) = mf(P \to C)$, that is, by the definition of maxflow, all the flow from *P* that reaches *C* will also reach C_i if the $C \to C_i$ weight is enough to carry it, and $mf(C_i \to P) = 0$ (because C_i is fresh). In effect, ID C_i earns all the "contribution reputation" of ID *C* but none of its "consumption reputation." In our context, this means that team *C* can recruit an arbitrary number of members because even if one member *can* consume, then any number of members can consume, as long as they erase the consequences of their consumption using fresh IDs as described above. This creates the wrong incentives for teams because they can recruit members disproportionately to their contribution. Note that ID *C will* indirectly owe more to others as a result of these Sybil attacks, however, because the members of *C* appear with a fresh team ID C_i each time, the denominator of Eq. (1) will always equal 0, irrespective of the value of $mf(C \to P)$. Fig. 5 shows this attack.

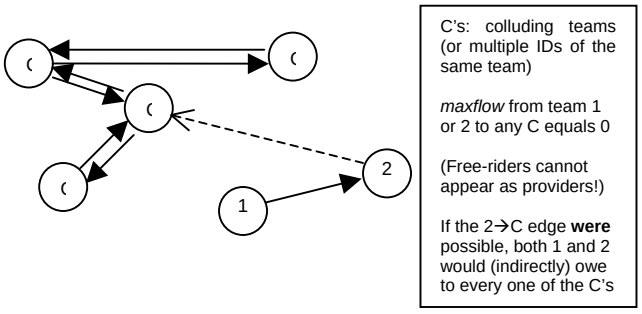

Figure 4. The maximum-flow decision function of [20] is robust to naïve collusion in which colluders falsely trade only with each other and never contribute to outsiders.

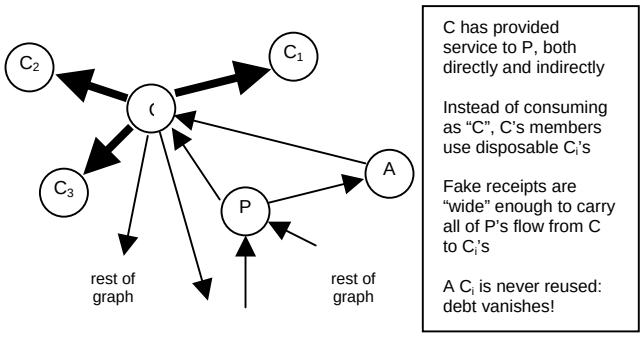

Figure 5. Less naïve collusion. The numerator in Eq. (1) is the same for C and the C_i's, but the denominator is 0 for all C_i's (fresh IDs with no history).

2) Extended Algorithm: To limit the consequences of this attack we observe that an attacking *C* will tend to have many high-weight outgoing edges. Also, we observe that, compared to *C*, the C_i's will be farther away from *P* in terms of path length. How much weight is "high" and what path distance is "far"? To answer this, we defined *GMF*, a *generalized maxflow* [37] algorithm with an estimator that detects excessive false trading.

If we assume that most teams are homogeneous in their consumption rates and that the probability of two teams interacting is uniform across all pairs of teams, we hypothesize that some uniformity will be evident in the receipt graph. We therefore define GMF as the following variant of maxflow: in GMF, the weight w_e of an edge e encountered during the standard maxflow computation is *discounted* depending on the distance, $d(P, S_e)$, of e's source vertex, S_e, from the maxflow source, *P*. w_e is further discounted depending on $q(P, S_e)$, which is equal to the following ratio: the sum of the weights of all the outgoing edges that originate from S_e, divided by the sum of the weights of all the outgoing edges that originate from *P*. The exact discounting occurs as follows: w_e is first divided by $g^{d(P, S_e)}$, where *g* is a GMF constant (we use $g = 2$); w_e is then further divided by $q(P, S_e)$, but only if $q(P, S_e) > 1$.

Fig. 6 shows a GMF example. GMF discounts the value of flow as we go farther away from the maxflow source, and it also discounts flow when it passes through vertices with a high

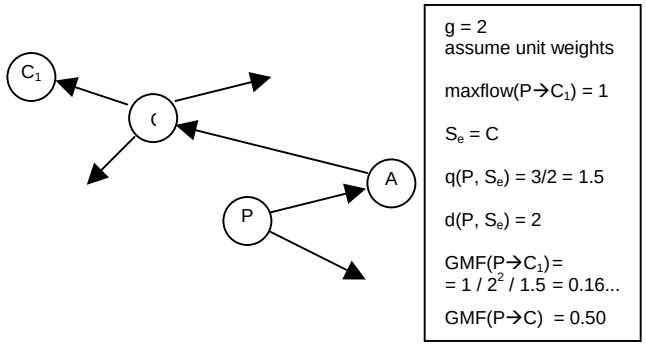

Figure 6. GMF example. In GMF, flow is discounted for 2 reasons: for being more than 1 hop away from the maxflow source, or for having passed through vertices with more total outgoing flow compared to the maxflow source.

sum of outgoing edge weights (compared to the sum of outgoing edge weights originating from the maxflow source). In the attack described above, GMF($P \rightarrow C_i$) will be less than GMF($P \rightarrow C$), and their difference will become even greater if *C* adds more outgoing edges.

The idea is to use the result of GMF to (subjectively) judge prospective consumers. A GMF result that is much smaller than the GMF results that *P* usually sees in the community should alert *P* to the possibility of false trading. *P* must employ the GMF test *before* the test of Eq. (1); *P* must first check for the GMF to *C*, and compare it to a running GMF average that *P* maintains as *P* interacts with the community. *P* only allows a request from *C* to proceed to the test of Eq. (1) with probability:

$$p = \min\left(\frac{g \times GMF(P \rightarrow C)}{gmf}, 1\right) \quad (2)$$

where *g* is the constant mentioned above (we use $g = 2$). Think of *g* as *P*'s *generosity*; *gmf,* on the other hand, is updated every time a new, strictly positive GMF result is computed by *P*. This update follows a standard formula for running estimators:

$$gmf = a \times gmf_{old} + (1\text{-}a) \times GMF(P \rightarrow C) \quad (3)$$

and we use $a = 0.5$. See also Algorithm 1 below.

Algorithm 1 RECIPROCITY ALGORITHM. Should *P* provide service to *C* ?

```
1: gmf_sample ← GMF(P→C)
2: if gmf_sample > 0 then
3:   gmf ← α × gmf + (1-α) × gmf_sample
4: end if
5: Apply Eq. (2) test: if pass then
6:   Apply Eq. (1) test: if pass then
7:     return YES
8:   end if
9: end if
10: return NO
```

B. Bootstrapping

The reciprocity algorithm requires unconditional cooperation in order to bootstrap. New teams (or teams that return

after an absence, whose transaction history is forgotten) must contribute unconditionally first without using the reciprocity algorithm. This must happen because teams, in their early life, do not have outgoing receipts (since they have not yet consumed). In such a case, the result of Eq. (1) will always be 0, and new teams would never cooperate; therefore, they would be no different from free-riders. To break this cycle, teams must contribute unconditionally at first; meanwhile, we assume that their members will try their luck in the community. We define the following: after the members of a new team finally manage to consume service for a total of *patience* times, the team can safely assume that it has finally become "known" for its contribution to the community, and it can start to use the reciprocity algorithm properly.

C. Gossiping Protocol

If the teams cannot agree on a central server, receipt dissemination through *gossiping* is required. Without gossiping, each team has a limited view of the receipt graph: it can only see its incoming edges (having earned the corresponding receipts directly from consuming members). This situation is equivalent to the *private history* of [20]. If we applied the maxflow heuristic in such a setting, it would be capable of measuring direct debt only. A community with this restriction would find it difficult to reach citywide scale; teams would have to be very few in number, with large membership, and a large number of APs, so that consuming members could always hope to access APs of teams that owe them directly.

At the opposite end of the spectrum of choices is our *centralized mode*, where the central server has knowledge of all receipts—at least all the recent ones that can fit in its memory.

Our gossiping protocol and *decentralized mode* is a third option: there is no central server, but teams attempt to create a consistent view of the receipt graph by sharing their views whenever they interact. A similar technique is studied in [30]. More specifically, members carry in their portable devices a small repository of receipts. We will see (Section VI-G) that the size of a receipt is in the order of 100 bytes, so modern mobile phones can store thousands of receipts. Members then, occasionally, request an *update* with the latest receipts by communicating with their own team server (over the community WLAN system or over 3G for example). Then, they share these new receipts with their prospective providers before they request service from them. Assume for example that team C is about to request service from team P. If the member of C shares with P the newly earned receipts of C, these would correspond to graph edges that point *to* C, which can only *increase* the result of $mf(P \rightarrow C)$ that P would calculate.

In addition, team C could also keep in its team server (and transmit to its members) other *random* receipts—themselves obtained by previous visiting members through the procedure we described above. Interestingly, the providers found on these receipts would have recently been consuming from C (which is how C got these receipts in the first place). If they had been consuming, then they would also have signed a receipt to C. Therefore, by sharing random receipts along with receipts that point to C, C shows to P essentially a *tree* of receipts, all the nodes of which directly or indirectly point to C, and thereby potentially increasing the result of $mf(P \rightarrow C)$. In this process, which we call *merging*, P's team server, when full, will never replace a receipt with an older receipt, only a newer one.

As a side effect, both teams are informed about (some of) their outgoing edges through this process. Note that members do not have to report the receipts they sign to their team server, but these receipts are required for the numerator of Eq. (1) or Eq. (2) to be strictly positive. Also, C does not reveal its outgoing edges to P: doing so would potentially increase the result of $mf(C \rightarrow P)$, and C does not want that. However, again, P learns about (some of) C's outgoing receipts through the same gossiping process. Therefore, in decentralized mode, P's calculation of both $mf(C \rightarrow P)$ and $mf(P \rightarrow C)$ is done in a balanced way: P has a lot of information for edges near the target vertices for both maxflow calculations, but less information for edges near the source vertices. The ratio of maxflow is not influenced much because this loss of information is caused by the same underlying process.

One difficulty with gossiping is that because members may present many receipts *before* requesting service, team servers may not want to spend the time to verify the signatures on all of them in "real-time" (for signature verification times see Section VI-G). Team servers, however, could verify receipts in a background process. Then, however, consuming members have an incentive to show forged receipts that appear to be signed by P even if they would never pass signature verification. However, P's team server can allow unverified receipts to be stored and used in maxflow calculations, as long as all receipts that *are* part of the maxflow-discovered paths *are* verified. The number of these receipts will generally be small.

D. Receipt Generation Protocol

During a WLAN session, APs request receipts from all WLAN-connected foreign members periodically. The requests contain the public key of the contributing team. Members are required to sign the appropriate receipts, which *must* contain the weight the provider is measuring. As a side-effect, by receiving such a receipt, the AP is sure that the session is not hijacked because nobody else has the appropriate private key, which must correspond to the member certificate that was presented initially. APs will only store the last receipt in such a series. Every receipt is uniquely identified in the system by the {*provider public key*, *consumer certificate*, *timestamp*} set of fields, so another receipt with different weight but the same timestamp (which always encodes the session *start time*) will be seen as the same. The AP has an incentive to hold the one with the biggest weight—the last one. By requesting receipts periodically, none of the two parties is at a significant disadvantage. To guard against members that attempt to avoid signing the first receipt, the AP, in the beginning of a session, can ask for receipts at a higher rate, and lower it after the member has proven trustworthy. There is no concept of members refusing to sign a receipt: this indeed happens as part of normal operation and such a *timeout* indicates that the session has ended. The AP will store the last receipt from that session in its team server or in the central server (depending on the operational mode), and will block further member traffic.

V. Simulation Results

In this section we present results from simulations that evaluate the performance of the reciprocity algorithm, the bootstrap algorithm, and the gossiping protocol, including the reciprocity algorithm's robustness to attacks.

A. Simulation Environment and Parameters

We programmed our custom simulator in Java, which can be downloaded from:

`http://mm.aueb.gr/research/P2PWNC`

Our objective was to model a peer-to-peer environment and examine the robustness of our reciprocity algorithms, not to perform low-level WLAN simulations. Time in our simulations consists of *rounds*. During a round, teams are randomly matched, and each team gets exactly one chance to consume and exactly one chance to contribute. The number of these *matches* per round is equal to the number of available teams. The number of teams changes: at round 1, we start with 2 teams, and at the end of each round a new team joins the community. This models community growth, and it continues up to a maximum number of n teams. For simplicity we assume that all WLAN sessions result in a new receipt with unit weight. If, in a match, the prospective contributor decides to provide service, its *score* is reduced by c. At the same time, the score of the consumer is increased by b. This models the cost and benefit of providing service. This is a standard setup for similar (evolutionary) games in sociology and biology; see, for example [25], or the peer-to-peer evaluation framework of [20]. This benefit-to-cost ratio, b / c, models how much the teams value the Wireless Community Network service. Using current 3G and DSL rates as an indicator, we set this ratio to equal 10 (with $b = 10$ and $c = 1$). Our results are, however, qualitatively the same for a wide-range of b / c ratios (not presented here).

The *rating* of a *strategy*, following [20], is the average of the running averages of scores per round of its followers, with each term weighted according to how many rounds the team has been using the strategy: "veterans" of a particular strategy carry more weight that than "amateurs" when calculating a strategy's rating. This is useful when teams change strategies (in the experiments where this is supported) because the early scores of amateurs do not affect much the overall rating of the strategy. For cooperative strategies, the top rating is 9 ($= b - c = 10 - 1$). This corresponds to a situation in which, for every round played, all the teams following the strategy contributed once and also managed to consume once per round. In Table I we give nominal values for our simulator parameters, as well as the Section of the paper that first discusses them.

TABLE I. Simulation Parameters

Parameter	Nominal value	Section
Number of teams (n)	150	V-A
Mode of operation	Decentralized	III-A-3
Server repository size (receipts)	1500	III-A-3
Patience (successful consumptions)	10	IV-B
Receipts to merge when gossiping	150	IV-C

B. Reciprocative Strategy Against Itself

1) Repository Sizes (Fig. 7): For our first experiment we wanted to see how our Reciprocative strategy (called *RECI* from now on) performs under various receipt repository sizes. An instance of a receipt repository is located either on the central server, or there is a different instance of the repository on every team server; this depends on the mode of operation. Fig. 7 shows the rating of RECI (remember that 9 is optimal, which equals $b - c$) for a community that grows to $n = 150$ teams, and two different repository sizes: 1500 and 500 receipts (all receipts are assumed to be unit-weight).

Having short-term history (that is, a finite repository with oldest-receipt-out replacement rule) is important because it encourages continous contribution—fresh receipts replace old ones and teams that delay in replenishing their incoming receipts find it harder to obtain service. However, a small repository size of 500 (small compared to the number of teams which equals 150, after the last team joins at round 150) causes the maxflow and GMF tests to mistake contributors for non-contributors. This starts a vicious cycle which ends with the collapse of cooperation. The two cases with 1500 receipts per repository do not have this problem, and approach the maximum rating of 9. Note that in centralized mode there is only one repository of 1500 receipts. In decentralized mode there are 150 team repositories *each one* storing 1500 receipts.

Note also that a rating of 9 will never be reached. This is because of the occasional mistakes made by the GMF and maxflow tests. These mistakes, and their result, represent the "social cost" of free IDs and of our algorithms.

2) Merging Receipts (Fig. 8): In decentralized mode of the previous experiment, consuming teams merged *all* their receipts with the provider. This is a burden for the provider who must verify them, and a burden for the member who must store and present a large number of receipts to prospective providers. We see, however, that even when merging only 50 receipts, the rating of RECI stays consistently above 8, that is, the information loss is not crucial. After ratings stabilize, near round 350, merging only 50 receipts corresponds to a running average of 90% success for RECI teams. That is, 1 in 10 requests is not satisified. However, increasing this to 500 merged receipts leads to 98% success; that is, the probability that a RECI does not recognize another RECI as a contributor is 2%. (During a merge, the consuming member shows the latest receipts from the consumer's team server to the provider; also, the consumer *hides* any receipts in which the consuming team appears as consumer. See also Section IV-C).

3) The Effect of Patience (Fig. 9): Here we examine how the initial patience parameter (see Section IV-B) affects the ratings later in the game. With the exception of the very small patience value of 1, the remaining patience values result in high levels of cooperation in the steady state, practically converging on the same rating; patience does not appear to be a crucially significant parameter. A patience value of 1 does not allow enough time to new teams to build a repository of incoming and outgoing receipts before they start using the

reciprocity algorithm, which makes them unecessarilty strict, causing a vicious cycle.

4) Merging and Repository Sizes Revisited (Fig. 10): We see here how the community performs once steady state has been reached (at round 500). Centralized mode does marginally better, followed by decentralized. We see again the effect of the repository size and its relation to the number of teams. In all our experiments, the best results were consistently obtained for repository sizes whose size was 10 times larger than the number of teams. The receipts to merge can be in the order of the number of teams.

C. Reciprocative Strategy against Free-Riders, Under-Providers, and Uncondtional Cooperators

1) Under-Providers and Evolutionary Learning (Fig. 11): Here, RECI confronts three other strategies: *Unconditional Cooperators* (ALLC), *Unconditional Defectors* (ALLD), and *Random* (RAND), a strategy that cooperates 50% of the time. In this experiment we simulated evolutionary learning [20]: at the end of each round, each team would consider the ratings of all the other strategies, pick one at random, and, if the rating for that strategy was higher than the rating for its own, it would switch to that strategy with a probability proportional to the difference in rating between the two strategies. Each team that joins chooses one of the 4 available strategies with probability 0.25. As time progresses, the winning strategies are RECI and ALLC, that is, the cooperative strategies. ALLD is doing less well because unconditional free-riders are easily recognized by the GMF test immediately (GMF to them is 0). However, ALLC and RAND followers may cooperate with them, and this is why ALLD followers persist for a while. ALLD is indeed decreased in the population, which, in turn, allows ALLC to endure because their potential exploiters are driven away through the "efforts" of RECI. RAND does well, but consistently worse than RECI. Providing service once every two rounds did not result in an increase in RAND's rating because their under-provision was detected by the GMF and maxflow tests. At round 500, the running average of RECI-RECI cooperations is 96.5%, and the running average of RECI-RAND cooperations (percentage of times RECI contributed to RAND) is only 68.4%. Due to evolutionary learning, at the end of round 500 the team population consists of 171 RECIs, 100 ALLCs, 20 RANDs, 5 ALLDs and 4 pre-RECIs (for a total of n = 300 teams). "Pre-RECIs" are RECI followers that undergo bootstrap because they switched to RECI only recently. Pre-RECIs are patient with their initial low ratings, expecting that they will soon switch to RECI.

2) Under-Providers without Learning (Fig. 12): Here we experimented with the probability with which RAND players cooperate. We test 5 different values, for two different strategic mixes. For low probability values RAND more closely resembles ALLD, while for high values it more closely resembles ALLC. RAND's best choice is shown to be 0.5, which is still an inefficient way to increase its payoff: RECI followers still do better.

D. Reciprocative Strategy against Sophisticated Colluders

In this experiment (Fig. 13) we simulated an attack similar to that of Fig. 5. Team 100 appears in the system at round 100. It follows the RECI strategy until round 200, at which time, every time it asks for service it does so *twice*, at different teams, each time using a new fake ID, which it never reuses. Because we use unit weights, we used 5, 10, or 15 receipts to "carry" the flow from the real ID to each of the two fake ones, each time. With 5 receipts, the fake IDs may not receive all the flow (maxflow averaged 10 units in the community of this experiment), but they risk less from the GMF test of excessive consumption. With 10 receipts, approximately all the incoming maxflow of Team 100 will be carried to its fake IDs. In all three situations Team 100 did badly: no team cooperated with Team 100 after a few tens of rounds (hence the regular drop in Team 100's running average rating). Team 100 would not do a lot worse if it simply *stopped* contributing. The GMF test was successful in detecting and punishing the most moderate use of concurrent false IDs, which is 2. If Team 100 tried to use 3 or more false IDs, excessive consumption would be recognized by the others even sooner. The GMF test achieved this without compromising RECI behavior in the previous experiments.

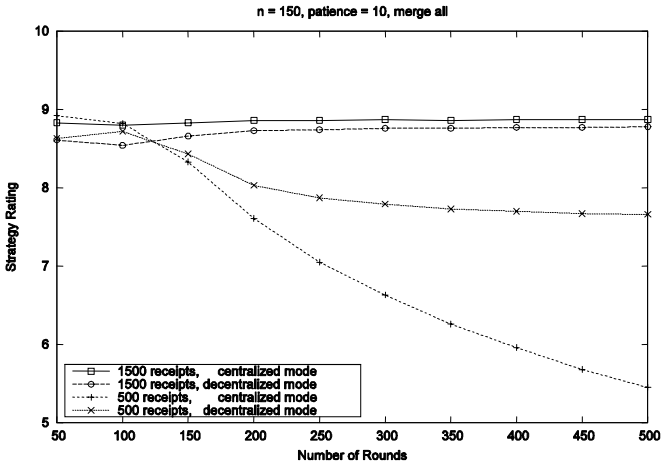

Figure 7. Repository sizes experiment (Section V-B-1)

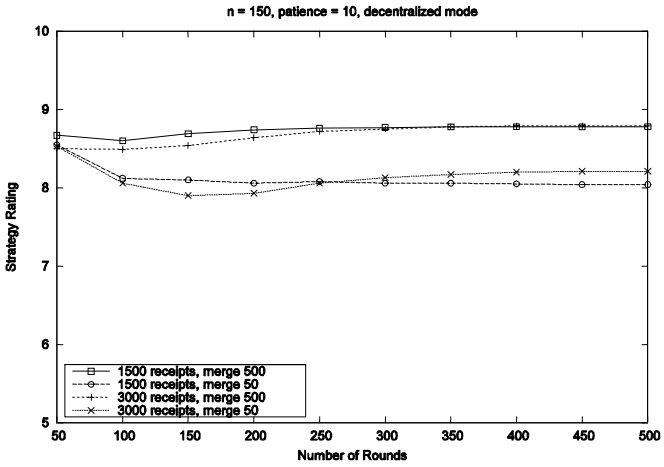

Figure 8. Merging receipts (Section V-B-2)

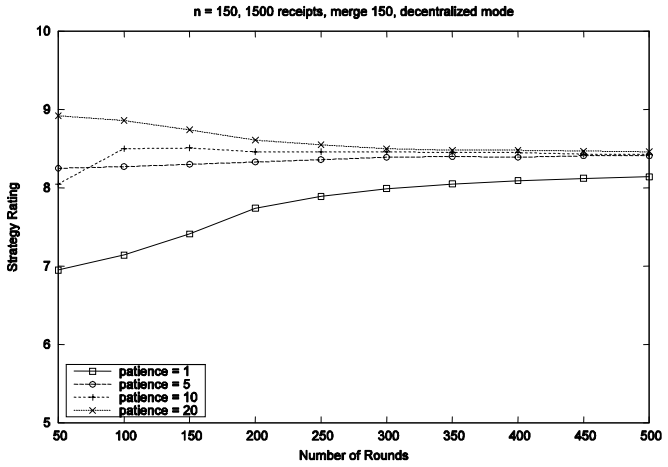

Figure 9. The effect of patience (Section V-B-3)

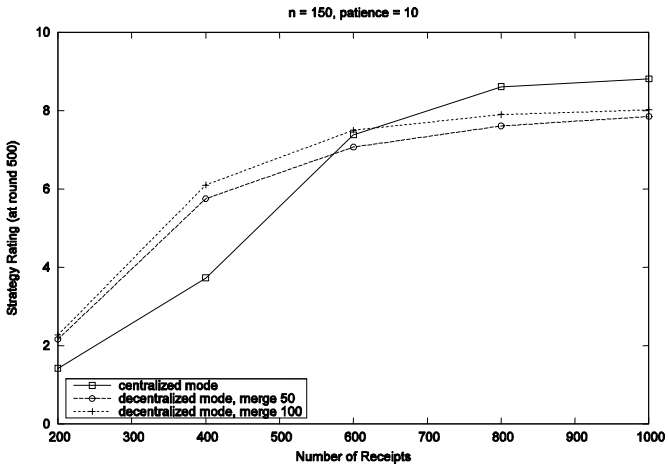

Figure 10. Merging and repository sizes (Section V-B-4)

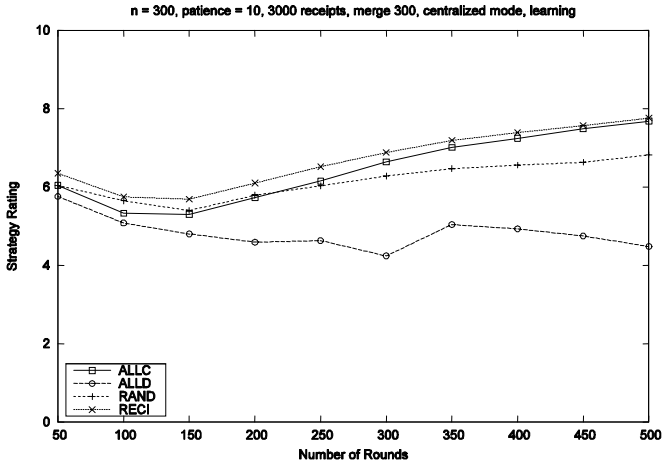

Figure 11. Under-providers and evolutionary learning (Section V-C-1)

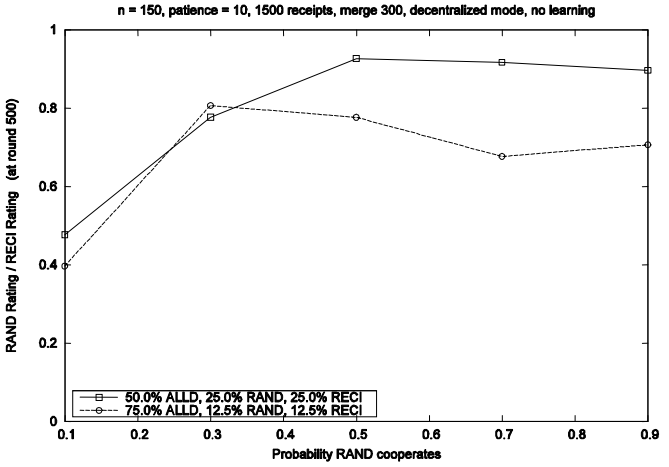

Figure 12. Under-providers without learning (Section V-C-2)

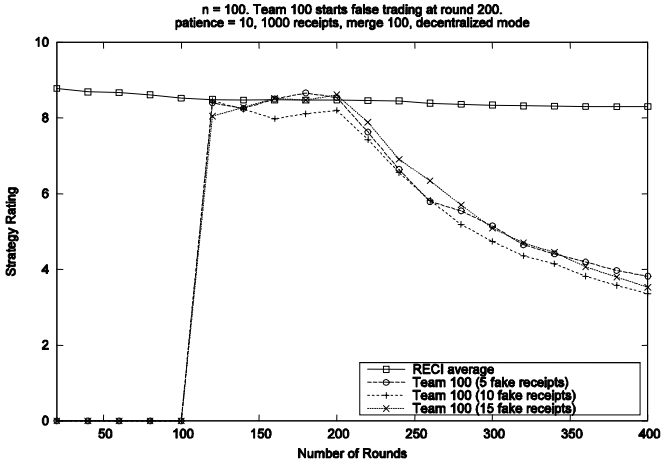

Figure 13. Sophisticated colluders and the GMF test (Section V-D)

VI. IMPLEMENTATION

This section summarizes the main results of the technical report [38].

A. Platforms

We implemented the P2PWNC protocol (AP and Team Server) on the Linux-based Linksys WRT54GS wireless AP [39] (currently retailing for less than $70). We included our software modules on the AP's firmware. We also implemented the client side of the protocol as a Java application that runs on Windows and Linux, and as a C application that runs on Linux. We left the implementation of a smart-phone client for future work.

We conducted experiments to test the performance of the software running on the Linksys AP. For comparison, we also ran the software on an AMD Athlon XP 2800 laptop. Table II shows the specifications of these two platforms.

TABLE II. PLATFORM SPECIFICATIONS

Characteristic	Athlon XP 2800	Linksys WRT54GS
CPU speed	2.08 GHz	200 MHz
CPU type	AMD Athlon XP 2800	Broadcom MIPS32
RAM	512 MB	32 MB
Storage	60 GB HD	8 MB Flash (read only) 32 KB NVRAM
Operating system	Linux kernel 2.4.18 (Red Hat Linux 8.0)	Linux kernel 2.4.18 (Linksys specific)
Cryptographic Library	OpenSSL 0.9.8 beta 5	OpenSSL 0.9.8 beta 5
Compiler	gcc 3.2	gcc 3.2
Compiler Optimizations	-O3	-O3 –mcpu=r4600 –mips2

B. Protocol Messages

The P2PWNC protocol comprises a set of 7 text-based messages, and operates on top of TCP/IP. A Base64 encoder is used to convert binary data to the text-based wire format. Table III shows the protocol messages, the entities that exchange them, and a short description for each message.

TABLE III. P2PWNC PROTOCOL MESSAGES

Message	Description	Direction
CONN	WLAN session initiation request	Client → AP
CACK	WLAN session initiation response	AP → Client
RREQ	Receipt request	AP → Client
RCPT	Receipt	Client → AP AP → Central or Team Server Team Server → Client
QUER	Query the Central or Team Server	AP → Central or Team Server
QRSP	Query response	Central or Team Server → AP
UPDT	Client receipt update request	Client → Team Server

Fig. 14 shows the format of an RCPT message that contains a receipt (in its Base64 wire format) signed using the Elliptic Curve Digital Signature Algorithm (ECDSA).

```
RCPT P2PWNC/1.0
Content-length: 357
Algorithm: ECC160
Timestamp: Tue, 24 May 2005 17:26:41 +0000
Weight: 6336
BNibmxStfJlod/LnZubH6pzWHQqKyZFcSMjnZurmTe4KjCRkllhV93MEegPvCsxz
2oe/hqevoPSrwO1JLO/36J8HTIeyeKQqTCfx+EPxweAvYC/ZFb8URLa2faIbvSgD
3lm6Wa1S4cYlSWeSNmFzS/ebDFfzakqNSEsERefwEcdWJD9gzIXafL4pojhhfP5b
rS4QPtHzBl58POfKdx9AqCDMBxRoGALKJSJYYXlsrwtiyZJKvPlU5B3lWrFuL25P
d+kv2iMVRElXk/4=
```

Figure 14. RCPT message. It contains the receipt encoded in its Base64 wire format. The *timestamp* and *weight* fields of a receipt are contained in human-readable form also.

C. Network Access Control

The AP software uses the Linux *iptables* firewall for network access control. We built our own Linux kernel module for measuring the volume of forwarded traffic per client. When WLAN clients associate with the AP they are assigned dynamic IP addresses from an address pool via DHCP, and are denied Internet access until the AP receives a positive QRSP from the Central or Team Server. When a session ends (RREQ timeout) the particular IP address is once again blocked, and measurement of traffic for or from that address is stopped.

D. Receipt Server Implementation

We implemented the team server to run on the AP itself. The code is only slightly different from the central server version that runs on PCs. We assume that teams will pick one of their APs for the role of team server.

A receipt server needs to support dictionary and graph operations. It should support efficient receipt insertion, deletion, search, and maximum flow computation. A composite data structure was built to accommodate this. The structure includes hash tables that store pointers to receipts and team IDs in order to achieve fast look-ups. A red-black tree [40] structure keeps receipts sorted by their timestamp. Each tree node corresponds to a receipt. A red-black tree supports logarithmic-time insertion, deletion and searching for receipts based on their timestamp. Finally, there is an adjacency-list representation of the receipt graph, with each red-black tree node also storing a pointer to the respective graph edge. A FIFO variant of the *push-relabel* maximum flow algorithm [41] has been implemented. Its $O(V^3)$ worst case running time is long; therefore, we used the *global relabeling heuristic* [41, 42], which yielded dramatic performance improvements.

E. Maximum-flow Performance

We measured the performance of the FIFO-based push-relabel algorithm with the global relabeling heuristic for various random graph instances. In our experiments, we created random directed graphs comprising 1000 and 10 000 receipts (edges), and 100 and 1000 teams (vertices). Table IV shows the pure CPU time spent on executing the algorithm (measured with the Linux *times* function). Each reported value is the average time spent on the execution of the maximum flow algorithm for 20 random source-destination pairs of the same graph. Time is measured in milliseconds.

TABLE IV. MAXIMUM FLOW ALGORITHM PERFORMANCE

	Athlon XP 2800		Linksys WRT54GS	
Number of receipts	100 teams	1000 teams	100 teams	1000 teams
1000	0.43 ms	0.23 ms	12.64 ms	3.75 ms
10 000	5.88 ms	12.72 ms	59.27 ms	134.04 ms

F. Cryptographic Parameters

The P2PWNC protocol supports both the RSA and Elliptic Curve Digital Signature Algorithm (ECDSA). RSA cryptosystem parameters include the bit length of the keys and the public exponent value (for us, fixed to 65537). For ECDSA, we used verifiably random curves over the F_p finite field. More specifically, we used the *sec160r1*, *secp192r1*, *secp224r1*, and *secp256r1* named curves [43, 44] for key lengths of 160, 192, 224, and 256 bits, respectively.

G. Receipt Sizes and Cryptographic Performance

We performed tests to study the performance of both the RSA and the ECDSA signature schemes. The operations of interest are the generation and verification of digital signatures. In our protocol, clients sign receipts and APs verify them. Tables V and VI shows the results of these tests. ECDSA is faster for signatures than it is for verifications and is appropriate for use in battery-powered devices like mobile phones. Also, the use of ECDSA results in smaller receipts, since ECDSA keys and signatures are shorter than their RSA counterparts for the same security level. The smallest receipt in P2PWNC, when all keys and signatures use 160-bit Elliptic Curve Cryptography (ECC) requires 211 bytes (3 ECC public keys, represented without point compression, requiring 41 bytes each, two ECDSA signatures requiring 40 bytes each, and two 4-byte integers representing the *timestamp* and *weight* fields). With 1024-bit RSA keys the receipt size is 648 bytes.

TABLE V. CRYPTOGRAPHIC OPERATION PERFORMANCE : SIGNING

	Athlon XP 2800		Linksys WRT54GS	
Bit length (RSA/ECC)	RSA	ECC	RSA	ECC
1024/160	9.0 ms	1.3 ms	300.6 ms	20.3 ms
1536/192	25.9 ms	1.2 ms	655.6 ms	18.5 ms
2048/224	47.3 ms	1.4 ms	1529.0 ms	23.4 ms
3072/256	149.1 ms	1.7 ms	3939.0 ms	73.1 ms

TABLE VI. CRYPTOGRAPHIC OPERATION PERFORMANCE : VERIFICATION

	Athlon XP 2800		Linksys WRT54GS	
Bit length (RSA/ECC)	RSA	ECC	RSA	ECC
1024/160	0.4 ms	6.5 ms	12.3 ms	114.7 ms
1536/192	0.8 ms	6.0 ms	21.4 ms	99.9 ms
2048/224	1.3 ms	7.1 ms	37.9 ms	135.7 ms
3072/256	2.8 ms	8.6 ms	75.3 ms	453.0 ms

H. Routing Performance under Verification Load

We carried out experiments to test the Linksys AP routing behavior under CPU-intensive ECDSA signature verifications. We transferred a 223 MB file via FTP through the AP's wired router interface and performed 160-bit ECDSA signature verifications at the same time. Table VII shows the effect of these verifications on throughput. From Table VI, the verification time for one ECC-160 signature is 115 milliseconds.

I. Reference Implementation Distribution

Our reference implementation is available to download from `http://mm.aueb.gr/research/P2PWNC`

There, firmware for the Linksys WRT54GS AP that includes configuration and management utilities is available in binary and source-code format. For client software, both our C and Java implementations are available. We used GNU tools *autoconf* and *automake* to package the software. Program sources are ready to compile and install.

TABLE VII. LINKSYS WRT54GS ROUTING PERFORMANCE UNDER SIGNATURE VERIFICATION LOAD

Verifications / s	Average TCP throughput (KB/s)	% of "no load" performance
0.0 ("no load")	3965	100%
0.7	3858	98%
1.3	3600	91%
2.6	3145	79%
3.8	2783	70%
8.7 (maximum rate)	—	—

VII. DISCUSSION AND CONCLUSIONS

The guiding vision behind the design of the Peer-to-Peer Wireless Network Confederation scheme and its algorithms was that of a broadband subscriber who purchases a P2PWNC-compatible Wireless LAN access point, sets it up at home in "sharing" mode, and earns the right to roam the streets of the city, enjoying ubiquitous high-speed wireless access through similar access points. We wanted the scheme to be: (1) open to all, with no registration procedure; (2) free to use, with reciprocity as the only driving force; (3) simple to implement.

Support for free IDs is the most challenging aspect of our design. This is followed by the problem of providing incentives for distributed accounting. Our work on the analysis, simulation, and implementation of P2PWNC is in its early stages. There are still restrictive assumptions, and team homogeneity tops the list; we need to analyze the effect of non-homogeneity on the GMF algorithm, and possibly extend it.

On the theoretical level, formal analysis of the effect of free IDs is required. The theoretical work on the area so far has been about proving impossibilities [11, 26]. However, there is still a huge space of available tradeoffs to explore before we dismiss free IDs as a useful identity and accounting system. It is important that we pursue this for two reasons: a system with free IDs has built-in privacy support, and the freedom of not having to register with authorities may permit massively distributed systems to grow organically.

We are concerned with the scalability of our system. Can it scale to large cities or just small towns? And how does user mobility affect this? Can QoS be useful? Currently, we do not examine the QoS that contributors offer to consumers. Also, can some of the parameters (the sizes of receipt repositories and the number of receipts to merge) be adjusted dynamically?

And what happens when team members do *not* trust each other? Excess consumption by members can be detected if the team just looks at the available graph, but what about more sophisticated attacks? In many ways, cooperation enforcement is like security engineering [36]. In this paper we showed how we deal with some obvious attacks first. We still require a systematic way to approach attacks against incentive schemes.

ACKNOWLEDGMENT

We thank our fellow participants in the Athens Wireless Metropolitan Network for their advice and support, and the anonymous reviewers whose comments helped improve the quality of the paper.

REFERENCES

[1] Seattle Wireless. http://www.seattlewireless.net

[2] NYCwireless. http:/www.nycwireless.net

[3] Consume. http://www.consume.net

[4] Athens Wireless Metropolitan Network. http://www.awmn.net

[5] Motorola CN620. http://www.motorola.com/wlan/solution_cn620.html

[6] Nokia 9500. http://www.nokia.com/nokia/0,,54106,00.html

[7] Electronic Frontier Foundation (EFF) Wireless friendly ISP list. http://www.eff.org/Infrastructure/Wireless_cellular_radio/wireless_friendly_isp_list.html

[8] Speakeasy NetShare. http://www.speakeasy.net/netshare/

[9] E. C. Efstathiou and G. C. Polyzos, "Self-organized peering of wireless LAN hotspots," European Transactions on Telecommunications, vol. 16, no. 5 (special issue on Self-Organization in Mobile Networking), in press.

[10] E. C. Efstathiou and G. C. Polyzos, "A self-managed scheme for free citywide wi-fi," In Proc. 1st IEEE WoWMoM International Workshop on Autonomic Communications and Computing, Taormina-Giardini Naxos, Italy, June 13, 2005.

[11] J. Douceur, "The Sybil attack," in Electronic Proceedings 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, MA, March 7-8, 2002.

[12] K. Panchanathan and R. Boyd, "Indirect reciprocity can stabilize cooperation without the second-order free rider problem," Nature, vol. 432, pp. 499-502, 2004.

[13] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Sustaining cooperation in multi-hop wireless networks," In Proc. 2nd USENIX Symposium on Networked System Design and Implementation (NSDI '05), Boston, MA, May 2-4, 2005.

[14] S. Buchegger and J.-Y. Le Boudec, "Performance analysis of the Confidant protocol (cooperation of nodes–fairness in dynamic ad-hoc networks)," In Proc. 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002), Lausanne, Switzerland, June 9-11, 2002.

[15] P. Michiardi and R. Molva, "Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," In Proc. IFIP TC6/TC11 6th Joint Working Conference on Communications and Multimedia Security, Portoroz, Slovenia, 2002.

[16] S. Zhong, J. Chen, and Y. R. Yang, "Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks," In Proc. IEEE INFOCOM, San Francisco, CA, March 30-April 3, 2003.

[17] S. Zhong, L. Li, Y. Liu, and Y. R. Yang, "On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks," In Proc. ACM MOBICOM, Cologne, Germany, August 28-September 2, 2005, in press.

[18] L. Buttyán and J.-P. Hubaux, "Stimulating cooperation in self-organizing mobile ad hoc networks," ACM/Kluwer Mobile Networks and Applications, vol. 8, no. 5, pp. 579-592, 2003.

[19] M. Félegyházi, J.-P. Hubaux, and L. Buttyán, "Nash equilibria of packet forwarding strategies in wireless ad hoc networks," IEEE Transactions on Mobile Computing, in press.

[20] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," In Proc. ACM Conference on Electronic Commerce (EC'04), New York, NY, May 17-20, 2004.

[21] R. Levien and A. Aiken, "Attack-resistant trust metrics for public key certification," In Proc. of the USENIX Security Symposium, 1998.

[22] M. K. Reiter and S. G. Stubblebine, "Authentication metric analysis and design," ACM Transactions on Information and System Security, vol. 2, no. 2, pp. 138-158, 1999.

[23] M. Feldman and J. Chuang, "The evolution of cooperation under cheap pseudonyms," In Proc. 7th IEEE Conference on E-Commerce Technology (CEC), München, Germany, July 19-22, 2005, in press.

[24] R. Axelrod, "The evolution of co-operation," Penguin Books, London, 1990. (First published by Basic Books, New York, 1984.)

[25] M. A. Nowak and K. Sigmund, "Evolution of indirect reciprocity by image scoring," Nature, vol. 393, pp. 573-577, 1998.

[26] E. Friedman and P. Resnick, "The social cost of cheap pseudonyms," Journal of Economics and Management Strategy, vol. 10, no. 2, 1998.

[27] B. Greiner and M. V. Levati, "Indirect reciprocity in cyclical networks – an experimental study," Discussion Papers on Strategic Interaction, 2003-15, Max Planck Institute of Economics, 2003.

[28] K. G. Anagnostakis and M. B. Greenwald, "Exchange-based incentive mechanisms for peer-to-peer file sharing," In Proc. 24th International Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan, March 23-26, 2004.

[29] L. P. Cox and B. D. Noble, "Samsara: honor among thieves in peer-to-peer storage," In Proc. 19th ACM Symposium on Operating System Principles (SOSP'03), Bolton Landing, NY, October 19-22, 2003.

[30] S. Čapkun, L. Buttyán, and J.-P. Hubaux, "Self-organized public-key management for mobile ad hoc networks," IEEE Transactions on Mobile Computing, vol. 2, no. 1, 2003.

[31] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative peer groups in NICE," In Proc. IEEE INFOCOM, San Francisco, CA, March 30-April 3, 2003.

[32] B. Yang and H. Garcia-Molina, "Ppay: micropayments for peer-to-peer sytems," In Proc. 10th ACM Conference on Computer and Communications Security (CCS), Washington, DC, Oct. 27-30, 2003.

[33] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: a secure economic framework for p2p resource sharing," In Electronic Proceedings 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, June 5-6, 2003.

[34] N. Ben Salem, J.-P. Hubaux, and M. Jakobsson, "Reputation-based wi-fi deployment," Mobile Computing and Communications Review (MC2R), July 2005, in press.

[35] E. Huang, J. Crowcroft, and I. Wassel, "Rethinking incentives for mobile ad hoc networks," In Proc. SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS), Portland, OR, August 30-September 3, 2004.

[36] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Experiences applying game theory to system design," In Proc. SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS), Portland, OR, August 30-September 3, 2004.

[37] É. Tardos and K. D. Wayne, "Simple generalized maximum flow algorithms," In Proc. 6th International Conference on Integer Programming and Combinatorial Optimization, pp. 310-324, 1998.

[38] P. A. Frangoudis, "The peer-to-peer wireless network confederation protocol: design specification and performance analysis," Technical Report 2005-MMLAB-TR-02, Mobile Multimedia Laboratory, Athens University of Economics and Business, June 2005. Available at http://mm.aueb.gr/technicalreports/

[39] Linksys Wireless-G broadband router. http://www.linksys.com

[40] L. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," In Proc. 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 8-21, 1978.

[41] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," Journal of the ACM, vol. 35, no. 4, pp. 921-940, 1988.

[42] B. V. Cherkassky and A. V. Goldberg, "On implementing the push-relabel method for the maximum flow problem," Algorithmica, vol. 19, no. 4, pp. 390-310, 1997.

[43] Standards for Efficient Cryptography group, "SEC1: Elliptic curve cryptography," September 2000. http://www.secg.org

[44] Standards for Efficient Cryprography group, "SEC2: Recommended elliptic curve domain parameters," September 2000. http://www.secg.org