On the Efficiency of P2P Service Trading through Social Networks

Yong Liu[†], Hao Hu[†], Zhengye Liu[†], Markus Mobius[*], Keith Ross[‡] and Yao Wang[†]
[†]Electrical & Computer Engineering, Polytechnic Institute of NYU
[*]Harvard University and NBER
[‡]Computer and Information Science, Polytechnic Institute of NYU

January 15, 2009

1 Service Trading through Social Networks

We consider a set U of users connected in a social network $G_S = (U, F)$, where the social link set $F \subset U \times U$ defines the friends relations between users. If user i treats j as his friend, i connects to j through a directed link $\langle i, j\rangle \in F$. Suppose each user can provide some services, and also want to consume some services provided by other users. We can define a demand graph $G_D = (U, D)$, where $D \subset U \times U$ is the service demand profile among users. For a service demand $d = (k, l) \in D$, k is the provider and l is the consumer of the service. User k charges user l a cost of $h^{(k,l)}$ for providing the service. We define the associated $|U| \times |U|$ demand matrix $H = [H(k, l)]$ as $H(k, l) = h^{(k,l)}$ if $(k, l) \in D$, and $H(k, l) = 0$ if $(k, l) \notin D$.

Assume there is no centralized bank and no common currency in the system. User k and l can trade their services synchronously if they need services from each other and $h^{(k,l)} = h^{(l,k)}$. However, this synchronous trading is too restrictive. To relax the synchronous requirement, the trust between friends can be utilized to allow friends exchange services asynchronously. Specifically, we introduce pairwise credit between a pair of friends $\langle i, j\rangle$. User j can obtain a service $d = (i, j)$ from i by borrowing $h^{(i,j)}$ amount of credits from i. Let b_{ij} be the amount of credit j owes to i. Automatically, we have $b_{ij} = -b_{ji}$. Initially, there is no credit balance between i and j, $b_{ij}(0) = b_{ji}(0) = 0$. After i serves j once, the balance becomes $b_{ij}(1) = h^{(i,j)}$. If the next time i obtains a service (j, i) from j, the balance becomes $b_{ij}(2) = b_{ij}(1) - h^{(j,i)} = h^{(i,j)} - h^{(j,i)}$. Let C_{ij} be the maximum amount of credits j can borrow from i. Then the constraint on the credit balance is $b_{ij} \le C_{ij}$, and $b_{ji} \le C_{ji}$. Since $b_{ij} = -b_{ji}$, the credit balance constraints between i and j can be summarized as

$$-C_{ji} \le b_{ij} \le C_{ij}$$

With pairwise credits, friends can exchange services asynchronously as the mutual credit limits allow. However, users still cannot obtain services provided by users outside of his direct friends group. To address this problem, we introduce a credit transfer mechanism inside social network to allow a user to obtain services from a remote user who is indirectly connected to him through a path of friend links. Specifically, if l wants to obtain some service provided by k, l first tries to find a path from k to l in the social network, $p(k, l) = \{k = r_0 \to r_1 \to r_2 \to \cdots r_{m-1} \to r_m = l\}$. Then l can initiate a sequence of credit transfers in the reverse direction of path $p(k, l)$: r_n borrows $h^{(k,l)}$ credits from r_{n-1}, $n = m, \cdots, 1$. After the credit transfers, k can provide the service to l and the credit balance on each node is updated as $b_{r_n r_{n+1}}(t + \delta) = b_{r_n r_{n+1}}(t) + h^{(k,l)}$.[1]

[1]Pairwise credits between different friend pairs are not exchangeable. Therefore, the credit balances between different friend pairs cannot be merged.

2 Credit Transfer Routing

More generally, a service demand can be served as long as a legitimate credit transfer flow can be established from the provider of the service to the consumer of the service. Let $x^d_{\langle i,j\rangle}$ be the amount of credit transfer for service d on link $\langle i,j\rangle$, then $\{x^d_{\langle i,j\rangle}, \langle i,j\rangle \in F\}$ should satisfy *flow conservation* on all nodes in the network:

$$\sum_{i:\langle i,u\rangle\in F} x^d_{\langle i,u\rangle} - \sum_{j:\langle u,j\rangle\in F} x^d_{\langle u,j\rangle} = \begin{cases} -h^d & \text{if } u = provider(d) \\ h^d & \text{if } u = consumer(d) \\ 0 & \text{otherwise} \end{cases}, \qquad \forall u \in U, \forall d \in D \tag{1}$$

When there are multiple simultaneous service demands, the total aggregate credit transfers on link $\langle i,j\rangle$ and $\langle j,i\rangle$ can be calculated as $\sum_{d\in D} x^d_{\langle i,j\rangle}$ and $\sum_{d\in D} x^d_{\langle j,i\rangle}$ respectively. Therefore the resulted credit balance between user i and j after all credit transfers is $b_{ij} = \sum_{d\in D}(x^d_{\langle i,j\rangle} - x^d_{\langle j,i\rangle})$. We have to make sure that the resulted credit balances on all links after executing all services are bounded by their credit limits:

$$-C_{ji} \le \sum_{d\in D}(x^d_{\langle i,j\rangle} - x^d_{\langle j,i\rangle}) \le C_{ij}, \qquad \forall \langle i,j\rangle \in F. \tag{2}$$

Given a social network $S = (U, F)$ with links weighted by credit limits $\{C_{ij}, \langle i,j\rangle \in F\}$, and the set of service demands D with the associated demand matrix H, the credit transfer routing problem is to find credit transfer flows $\mathcal{X}(H) \triangleq \{x^d_{\langle i,j\rangle}, d \in D, \langle i,j\rangle \in F\}$ such that constraints defined in (1) and (2) are all satisfied.

3 Feasibility Results for Static Service Demands

Lemma 1 *If a set of demands form a loop in the demand graph G_D and the demand costs are the same, then all demands in the set can be executed simultaneously as long as users involved in the demand sets are connected in the social network G_S.*

Proof: Let $C = \{u_0 \to u_1 \to \cdots u_m \to u_{m+1} = u_0\}$ be a loop in the demand graph, and $h^{(u_n,u_{n+1})} = h, 0 \le n \le m$. For $0 \le n \le m-1$, find a path P_n in G_S from u_n to u_{n+1}. Obviously $P_{-m} \triangleq \bigcup_{0\le n\le m-1} P_n$ is a path (might with loops) from u_0 to u_m. Then $P_m \triangleq \{\langle j,i\rangle |$ the reverse link $\langle i,j\rangle \in P_{-m}\}$ forms a path from u_m to u_0. Now allocate credit flows for demands in set $\{(u_n, u_{n+1}), 0 \le n \le m\}$ as follows:

$$\{x^{(u_n,u_{n+1})}_{\langle i,j\rangle} = h, \forall \langle i,j\rangle \in P_n, 0 \le n \le m\}$$

It can be easily verified that the flow conservation in (1) is maintained for all demands, and

$$\sum_{n=0}^{m} x^{(u_n,u_{n+1})}_{\langle i,j\rangle} = \sum_{n=0}^{m} x^{(u_n,u_{n+1})}_{\langle j,i\rangle} \qquad \forall \langle i,j\rangle \in \bigcup_{0\le n\le m} P_n.$$

In other words, the credit balance resulted from the executions of this set of service demands is zero on all social links involved. Therefore the credit balance constraints (2) are satisfied automatically.

In summary, a demand loop in the demand graph can be executed synchronously in a social network without leaving any balance on any social link. ■

As illustrated in Figure 1, five users form a social network with a chain topology: $A \leftrightarrow B \leftrightarrow C \leftrightarrow D \leftrightarrow E$. If C wants service from A, and E wants service from C, A wants service from E. Service providers and consumers are not directly connected in the social network. By setting up credit transfer path $A \to B \to C$ for service (AC), $C \to D \to E$ for service (CE), and path $E \to D \to C \to B \to A$ for service (EA), all demands are executed, and the credit balances on all links remain zero.

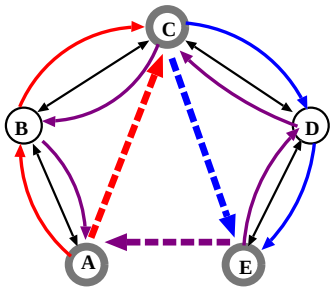


Figure 1: Credit transfer flows for a balanced demand set

Definition 1 *Balanced Demand Matrix: a demand matrix H is called balanced if for any user in the demand matrix, the total cost of service demands generated by him equals to the total cost of service demands requested from him by other users in the demand set:*

$$\sum_{l \in U} H(k,l) = \sum_{i \in U} H(i,k), \forall k \in U$$

Lemma 2 *Any balanced demand matrix can be decomposed into finite demand loops in the demand graph.*

Proof: In the demand graph G_D, assign the cost of a service demand as the weight of the corresponding demand link. For any node in the graph of a balanced demand set, the total weight on its ingress links equals to the total weight on its egress links. One can traverse the graph in the following way: starting from any node u_0, pick any egress link, say $\langle u_0, u_1 \rangle$, with positive weight, move to u_1; since u_1 has at least one positive weight ingress link, due to the balanced ingress and egress link weights, it must have at least one positive weight egress link, then pick any positive weight egress link of u_1, say $\langle u_1, u_2 \rangle$, move to u_2, using the same argument, the trip can continue until at some step n, the trip goes back to a previously visited node u_i, $0 \leq i < n$, then we find a demand loop $u_i \rightarrow \cdots u_{n-1} \rightarrow u_n = u_i$. Let $h = \min_{i \leq l \leq n-1} h^{(u_l, u_{l+1})}$, remove the identified loop from the graph by subtracting h from the weights of all links in the loop. The ingress and egress link weights on all nodes are still balanced after the loop removal. We can repeat the process until all link weights go to zero, or equivalently all demands have been decomposed into demand loops ■

Theorem 3 *Any balanced demand matrix can be executed simultaneously in a social network G_S as long as users involved in the demand sets are connected in G_S.*

Proof: According to Lemma 2, we can decompose a balanced demand set into demand loops. According to Lemma 1, each demand loop can be executed independently without generating credit balance on any link. After executing all demand loops, all demands in the original balanced set are executed. ■

Definition 2 *Reduced Demand Matrix: for a demand matrix H, the reduced demand matrix $\bar{H}$ is defined as*

$$\bar{H}(k,l) = H(k,l) - \min(H(k,l), H(l,k))$$

The reduced demand matrix captures the net demand between a pair of users.

Corollary 4 *A demand matrix H is executable in a social network G_S if and only if the corresponding reduced demand matrix $\bar{H}$ is executable in G_S.*

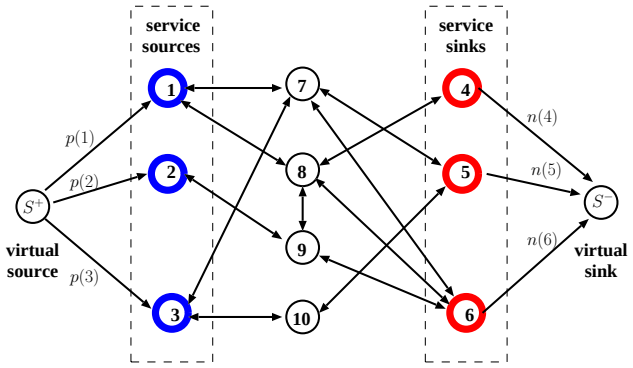


Figure 2: Extended Social Graph for Unbalanced Demand Set

Proof: **IF :** $\Leftarrow$ If $\bar{H}$ is executable, let $\mathcal{X}(\bar{H})$ be the associated credit flow. Define $\hat{H} = H - \bar{H}$, where the subtraction is taken on each element. $\hat{H}$ defines a balanced demand set. Due to Theorem 3, it is executable with some credit flow $\mathcal{X}(\hat{H})$ with zero credit balance on all links. It can be easily verified that $\mathcal{X}(\bar{H})+\mathcal{X}(\hat{H})$ defines a legitimate credit flow for demand $H = \bar{H} + \hat{H}$
ONLY IF : $\Rightarrow$ If H is executable, let $\mathcal{X}(H)$ be the credit flow. For each pair of users (k, l), merge credit flows for two demands (k, l) and (l, k). It can easily verified that the combined flow implement the net demand between (k, l) (possibly with credit transfer loops). ■

Definition 3 *Unbalanced Demand Matrix: a demand matrix H is called unbalanced if there is at least one user such that the total cost of service demands generated by him does not equal to the total cost of service demands requested from him by other users in the demand set:*

$$\sum_{l \in U} H(k, l) \neq \sum_{i \in U} H(i, k), \exists k \in U$$

Definition 4 *Service sources and sinks in unbalanced demand set: for a user u in a unbalanced demand set, calculate the difference between the cost of service that other users request from him and the cost of the service that he requests: $w(u) = \sum_{l \in U}(H(u, l) - H(l, u))$. If $w(u) > 0$, we call user u a service source, and $p(u) = w(u)$ is its net service contribution; if $w(u) < 0$, we call user u a service sink, and $n(u) = -w(u)$ is its net service consumption.*

Definition 5 *For a social network $G_S = (U, F)$ with a unbalanced demand set D, let U^+ be the set of service sources, and U^- be the set of service sinks, we construct an extended social network $G'_S = (U', F')$ as follows:*

$$U' = U \cup \{s^+, s^-\}; \tag{3}$$
$$F' = F \bigcup_{u \in U^+} \langle s^+, u \rangle \bigcup_{v \in U^-} \langle v, s^- \rangle; \tag{4}$$
$$C_{\langle s^+, u \rangle} = p(u), \quad \forall u \in U^+; \tag{5}$$
$$C_{\langle v, s^- \rangle} = n(v), \quad \forall v \in U^-. \tag{6}$$

Figure 2 plots an extended social graph for an original social network graph with ten users, where three users are service sources, and three users are service sinks, four users are balanced.

Theorem 5 *An unbalanced demand matrix D is executable in a social network G_S if and only if the min-cut between s^+ and s^- in the extended social network G'_S is greater than or equal to $M = \sum_{u \in U^+} p(u) = \sum_{v \in U^-} n(v)$.*

Proof: $\{$*SKETCH*$\}$
ONLY IF : $\Rightarrow$ In any credit transfer solution for D, the net credit flow from set U^+ to U^- is $M = \sum_{u \in U^+} p(u)$. Therefore the min-cut in G'_S has to be at least M.
IF : $\Leftarrow$ Given a min-cut between s^+ and s^- in G'_S with size M, construct a flow from s^+ and s^- with volume $M = \sum_{u \in U^+} p(u)$, due to the link capacity constraint on link $\langle s^+, u \rangle, u \in U^+$, the flow routed from s^+ to a service source $u \in U^+$ is exactly $p(u)$, likewise, the flow routed from a service sink $v \in U^-$ to s^- is exactly $n(v)$. On the virtual source node s^+, put different labels to flows distributed through service source nodes. On each service sink node, calculates the volume of flows received from different service sources, denoted as $f(u,v), u \in U^+, v \in U^-$, obviously we will have

$$\sum_{v \in U^-} f(u,v) = p(u), \quad \forall u \in U^+; \qquad \sum_{u \in U^+} f(u,v) = n(v), \quad \forall v \in U^-; \tag{7}$$

Construct a new demand matrix $\hat{H}$ with $\hat{H}(u,v) = f(u,v)$, the maximum flow directly transfer credits for $\hat{H}$ in G_S. So demands in $\hat{H}$ can be synchronously executed in G_S. It can be easily verified that

$$\sum_{l \in U} (\hat{H}(k,l) - \hat{H}(l,k)) = \sum_{l \in U} (H(k,l) - H(l,k)), \quad \forall k \tag{8}$$

Roughly speaking, $\hat{H}$ and H have exactly the same service unbalance distribution at the node level.

Now construct another demand matrix $\tilde{H}$ by "subtracting" $\hat{H}$ from H. More specifically,

1. $\tilde{H}(k,l) = H(k,l) - \hat{H}(k,l)$;
2. $d(k,l) = \tilde{H}(k,l) - \tilde{H}(l,k) > 0$ set $\tilde{H}(k,l) = d(k,l)$ and $\tilde{H}(l,k) = 0$; otherwise set $\tilde{H}(k,l) = 0$ and $\tilde{H}(l,k) = -d(k,l)$.

Due to (8), it can be verified that $\tilde{H}$ is a balanced demand set. Due to Theorem (3), we can find credit transfer flows for $\tilde{H}$ without change the credit balance on links. Combine the credit flows for $\hat{H}$ and $\tilde{H}$, we have a credit flow for $\hat{H} + \tilde{H}$. It can be checked that the reduced demand matrix for $\hat{H} + \tilde{H}$ is the same as the reduced matrix for the original demand set H. Due to Corollary 4, H is also executable. ■

4 Credit Transfer Strategies for Dynamic Service Demands

In a time-slotted system, at time slot k, a new set of demands $H(k)$ are generated among users. Given the credit balance on all social links resulted from previous services, one need to find credit transfer flows to maximally execute current demand set.

Theorem 6 *Given a set of executed demands in history $H(i), 1 \leq i \leq k-1$, a new demand set $H(k)$ is executable in G_S if and only if the aggregate demand up to time k, $A(k) = \sum_{i=1}^{k} H(i)$, is executable in G_S with zero credit balance on all links. In other words, the executability of $H(k)$ is independent of how credit flows were set up for demand sets $H(i), 1 \leq i \leq k-1$, that have been executed in the past.*

Proof: Denote by $\mathcal{X}(i)$ the credit flow for demand set $H(i)$, $1 \leq i \leq k-1$.

ONLY IF : $\Rightarrow$

If $H(k)$ is executable at time k, let $\mathcal{X}(k)$ be the corresponding credit flow. Naturally we have a credit flow for the aggregate demand $A(k)$ in the initial social graph by merging $\{\mathcal{X}(i), 1 \leq i \leq k\}$ into one set of credit flows.

IF : $\Leftarrow$

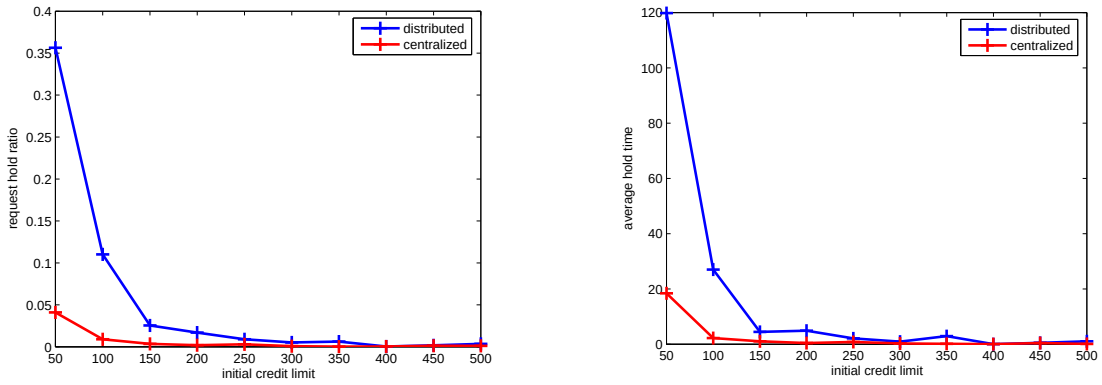
Let $\mathcal{Y}(k)$ be the credit flow for the aggregate demand set $A(k)$ in G_S with zero credit balance one all links.

Similar to the previous argument, all credit flows configured up to time $k-1$ execute the aggregate demand up to time $k-1$: $A(k-1) = \sum_{i=1}^{k-1} H(i)$. Let $\mathcal{Y}(k-1)$

be the aggregate credit flow. Construct a new demand $\check{A}(k-1)$ by reversing the directions of all demands of $A(k-1)$, construct a new flow $\check{\mathcal{Y}}(k-1)$ by reversing the directions of all credit flows in $\mathcal{Y}(k-1)$, immediately $\check{\mathcal{Y}}(k-1)$ implements all demands in $\check{A}(k-1)$.

At time k, we first add in credit flow $\check{\mathcal{Y}}(k-1)$ to execute $\check{A}(k-1)$, then all links' credit balance go back to zero. Then we add in $\mathcal{Y}(k)$ to implement $A(k)$. In two steps, we construct a credit flow for the composite demand $\check{A}(k-1) + A(k)$, which has the same reduced demand matrix as $H(k)$. According to Corollary 4, $H(k)$ is also executable at time k. ■

When a peer cannot find a credit flow for his demand, he can either drop the demand request, or keep the request in buffer and wait for credit balances change triggered by transactions from other peers in the network. To verify the efficiency of social network based P2P trading, we conducted a preliminary simulation



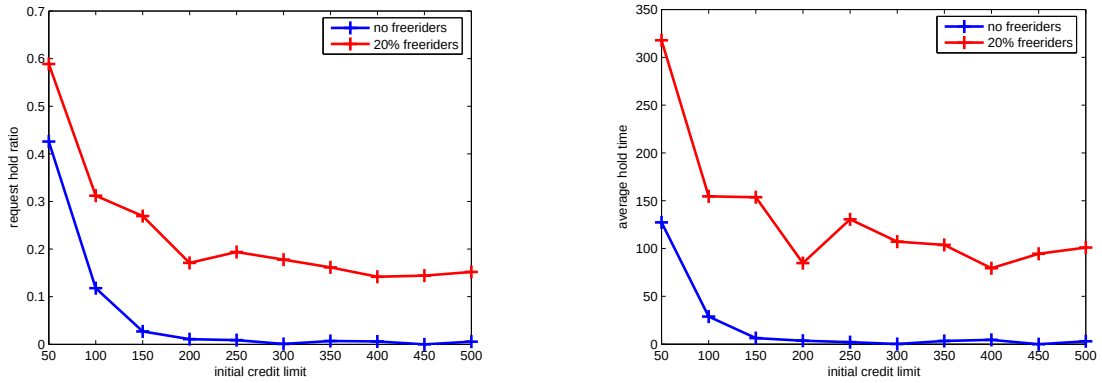
(a) Holding Probability as Function of Credit Limits

(b) Waiting Time as Function of Credit Limits

Figure 3: Efficiency of Social Network Trading with Dynamic P2P Service Demands

study using a synthesized social network with 400 peers. Peers randomly connect to each other. Peer degree follows truncated power law with coefficient $\alpha = 2$ in the range of $[5, 100]$. The average peer degree is 5.2. We assign homogeneous credit limit K to all peering links. Each peer generates service demand to a random peer in the network. The cost of demand follows a truncated normal distribution in the range of $[10, 120]$. The average cost of demand is 50, and the standard deviation is 50. The demand inter-arrival time on each peer is uniformly distributed in $[0, 400]$. We simulated both *bufferless* and *buffered* peer request mode. In the buffered mode, peers hold unfulfilled demand requests and wait for a chance to execute them in a later time. Given credit balances on all links, a demand is executed if and only if a credit transfer flow can be established between the supplier and consumer of the demand. For comparison, we also simulated the same P2P trading with a centralized bank: each peer has a balance and credit limit with the bank; a peer's balance increases if he provides a service and decrease if he consumes a service; a peer's service got hold if his

negative balance reach his credit limit. We set the credit limit of a peer with the bank to be the summation of his credit limits with all his friends in the social network. Figure 3(a) plots the probability that demands got hold in buffer as a function of credit limits. As we increase the credit limit from 50 (roughly one demand) to 150 (roughly three demands) on all links in the network, the holding probability quickly decreases from 35% to 3%. The hold probability approaches that of the centralized bank as the credit limit increases. Figure 3(b) plots the average waiting time for demands to be executed. Again, there is quick reduction when the credit limit is increased from 50 to 150. The waiting time approaches zero when credit limit is large. We repeat the same simulation with the setting that 20% randomly chosen peers behave like free-riders. They only generate service demand, never serve demands from other peers, and never participate in credit transfer. As illustrated in Figure 4, the trading efficiency of the whole social network degrades significantly. Both the hold ratio and request wait time in the buffer increases.



(a) Holding Probability as Function of Credit Limits

(b) Waiting Time as Function of Credit Limits

Figure 4: Efficiency of Social Network Trading with Free Riders