DPCP: A Protocol for Optimal Pull Coordination in Decentralized Social Networks

Huanle Xu, Pili Hu, Wing Cheong Lau, Qiming Zhang, Yang Wu
Department of Information Engineering, The Chinese University of Hong Kong

***Abstract*—Social Networking Service has become an essential part of our life today. However, many privacy concerns have recently been raised due to the centralized nature of such services. Decentralized Social Network (DSN) is believed to be a viable solution for these problems. In this paper, we design a protocol to coordinate the pulling operation of DSN nodes. The protocol is the result of forward engineering via utility maximization that takes communication layer congestion level as well as social network layer centrality into consideration. We solve the pulling rate control problem using the primal-dual approach and prove that the protocol can converge quickly when executed in a decentralized manner. Furthermore, we develop a novel "drumbeats" algorithm to estimate node centrality purely based on passively-observed information. Simulation results show that our protocol reduces the average message propagation delay by 15% when comparing to the baselined Fixed Equal Gap Pull protocol. In addition, the estimated node centrality matches well with the ground-truth derived from the actual topology of the social network.**

I. Introduction

Social Networking Service (SNS) has become an essential part of life nowadays. Due to the poor privacy control configurations/policies offered by centralized SNS, developers and researchers are now seeking a decentralized solution, where users can regain full control of their own data. Although various solutions have been proposed and implemented, like Diaspora and OStatus, their adoption rate is very low compared with the existing centralized SNS. One previous study [3] shows that the most challenging problem in the movement towards Decentralized Social Network (DSN) is to address possible loss of social connections during the migration process. Towards this end, [3] have built a middleware called SNSAPI that can unify interfaces and data structures of many existing SNS's as well as conventional communication channels like Email and RSS. SNSAPI allows its users to not only connect with old friends on heterogeneous centralized services but also directly communicate with each other via various kinds of supported channels. The flexibility provided by that middleware enables a smooth transition path from the current centralized paradigm to a decentralized one. [3] shows that the SNSAPI solution can scale to thousands of DSN nodes which exchange information with each other in a pull-based manner. That is, nodes post messages to their own local storage and each node periodically polls their followees for new messages. We refer this approach as the Fixed Equal Gap Pull (FEGP) protocol. Its worth to note that the alternative push-based model is not as attractive as it would require additional infrastructure support like a PubSub bus, relays servers or structured P2P overlay, which will hinder the initial boostrapping and incremental deployment of the DSN. While FEGP may be adequate for jumpstarting the initial deployment of a DSN, it is likely to encounter scalability problems as the number of nodes in the DSN grows beyond the range of thousands. In particular, there are three major drawbacks of the FEGP protocol: Firstly, as a DSN grows, operation parameters like the pull-gap, i.e. the periodic polling interval, should be adjusted automatically instead of being kept constant as in the case of FEGP. Secondly, the pulling operations of different nodes under FEGP are not coordinated at all. This can potentially cause congestion in some part of the DSN. Worse still, the number of followers typically exhibit a highly skewed distribution in a social network. This means popular nodes are likey to be overwhelmed by a large volume of non-coordinated pulling operations. To tackle this problem, nodes must reduce their pulling rate if congestion is detected. Thirdly, FEGP is totally unaware of the topology of the social network. Depending on the position of a node in the social graph, its importance, in terms of ability to disseminate information within the DSN, can be quite different. It is desirable to let more important nodes pull faster than others, especially in the event of resource contention. The importance of a node within a social network can be quantified by various network centrality metrics such as PageRank of a node within the friendship graph or the in-degree of the node in the activity graph.

With the above observations in mind, we proceed to forward engineering the so-called Distributed Pull Coordination Protocol (DPCP) via an optimization and control-theoretic framework. We first formulate a utility maximization problem by constructing a utility function to capture both data rate in the communication layer and the importance of a node in the DSN. We then apply the primal-dual approach to yield a distributed algorithm which is guaranteed to converge to an optimal solution. Under this algorithm and its corresponding protocol, each DSN node simply adjusts its pull gap to maximize its own local utility according to *passively* observed feedbacks such as network delay, packet loss rate, connection loss probability. In the basic version of DPCP, centrality measures enter the utility function as constants. In practice, this corresponds to the scenario where centrality is known a priori from domain knowledge, or separately estimated by another auxiliary protocol. In the advanced version of DPCP, we can estimate the centrality purely based on passive observations on the behaviour of neighboring nodes, i.e. on how the pulling rates of followers of a node change over time. In particular, we provide an example of estimating the eigenvector centrality metric for each node via a "drumbeats" algorithm.

One distinguishing feature of DPCP is that the utility func-

tion captures not only the data rate of the communication layer but also the importance of the node in the DSN. Furthermore, DPCP can signal the nodes to back off purely based on passively observed information. Lastly, DPCP eliminates the need of a separate, auxiliary protocol for centrality estimation. These properties make DPCP readily and incrementally deployable on the SNSAPI-based DSN. Although the design of DPCP is motivated by improving FEGP used in SNSAPI-platform [3], the protocol is generally applicable for other DSNs. DPCP can even be used for centralized social networks where thousands of users are requesting data from a single congested server. Note that our work is solving a utility maximization problem and differs from works that minimize message propagation delay [4]. Interested readers can refer to [4] for more work in that direction. To summarize, we have made the following technical contributions:

- After reviewing the related work in the literature in Section III, we illustrate the importance of considering the centrality of a node in the social network during message-pulling coordination.
- In Section IV, we formulate a utility maximization problem that takes both link-layer congestion control and social network centrality into consideration.
- In Section V, we adopt the primal-dual approach to solve the optimization problem and give a delay-based instantiation that direct a node to gracefully back off based on passively observed information.
- In Section VI, we develop a "drumbeats" algorithm for nodes to estimate their own eigenvector centrality by passively observing the behaviour of neighbouring nodes. This algorithm removes the dependence of prior knowledge of centrality or the need of any auxiliary protocol.
- In Section VII, we discuss some real implementation issues for the protocol. We evaluate the performance of DPCP via an extensive simulation study in Section VIII and illustrate its convergence behavior during various operating conditions. We also demonstrate the effiacy of the proposed "drumbeats" algorithm in providing an accurate estimate of node centrality. We conclude our findings and discuss future works in Section IX.

II. Related Work

Although the optimization framework and methodology used by some previous works are similar to ours, there are two significant differences in our protocol design due to the special context of DSN. Firstly, message forwarding in a DSN is selective and hop-by-hop in nature. In particular, the forwarding/propagating of a posting is not automatic but requires explicit user action (and interest) Strictly speaking, each forwarding operation actually creates a new message. On the contrary, in the context of network multicasting and P2P content distribution, all the nodes have the same objective, i.e. to acquire the same content as fast as possible. For example, [10] studies the message propagation delay from one source to the whole network under different message propagation methods. [7] proposes an efficient mechanism for one-to-many content distribution to improve the throughput. [8] presents an optimal scheduling algorithm to minimize the completion time for file dissemination in P2P networks. On the contrary, the back-off protocol in our work is restricted to per-hop optimization.

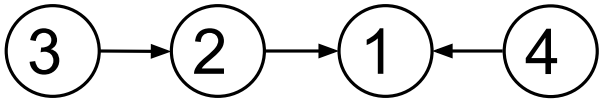

Fig. 1. An illustrated example of a social network where each "$\rightarrow$" represents a following friendship, e.g. User 2 is the follower of User 1.

Secondly, a normal P2P network adapts its topology for the benefit of data transfer rate. For example, in a Bit-Torent network, nodes periodically switch from slow peers to fast peers. In [14], researchers propose a Markov-chain guided topology hopping algorithm to dynamically select the peers to maximize the broadcast rate. However, the topology of DSN, in terms of social network layer, is fixed. The follower-followee relationship is not setup arbitrarily and the number of followers can be highly skewed. Towards this end, node-level congestion (in end-host), instead of link-level congestion (in network) become as a major problem which is the focus to be addressed by our proposed protcol.

III. The Importance of Considering Node Centrality in Social Network

Before delving into the detailed mathematical formulation, we illustrate in this section, by a simple example, why it is important to take node centrality into consideration. The topology is depicted in Fig. 1. Node 1 creates a new message every unit of time. Each user pulls the messages from its followees. For simplicity, we assume each user can only handle one request per unit of time, i.e. only one message pulling request from a random follower can succeed and all other requests are blocked. Moreover, each pulling request can obtain multiple new messages.

When FEGP is used, all the users pull the messages from its followees with a same fixed gap which is assumed to be one unit time in this example. Node 1 receives 2 requests every unit of time and each request can succeed with a probability $1/2$. There are four cases depending on which neighbour succeeds in each round:

- Case 1: Node 2 pulls the messages successfully in both time units. In this case, Node 2 gets two messages and Node 3 gets one message.
- Case 2: Node 4 succeed to pull the messages in both time units. Then, only Node 4 gets two new messages.
- Case 3: Node 2 and Node 4 succeed to get the messages in the first and second time unit respectively. In this case, both Node 2 and Node 3 get one message. However, Node 4 can get two messages for its pull.
- Case 4: Node 4 and Node 2 succeed to get the messages in the first and second time unit respectively. In this case, Node 4 only gets one message and Node 2 gets two messages while Node 3 cannot get any messages.

Each case can happen with a probability $1/2 * 1/2 = 1/4$. Thus, the expected number of messages which can be propagated in two units of time is $1/4 * (3 + 2 + 4 + 3) = 3$.

Now we consider the alternative that Node 2 and Node 4 pull messages from Node 1 with different gaps of x and y, respectively. In this case, each request of Node 2 can succeed with a probability $y/(x + y)$. For ease of description, we denote $\alpha = y/(x + y)$ as a shorthand. Consider the four cases discussed above which occur with probability $\alpha^2, (1 - \alpha)^2, \alpha(1 - \alpha)$ and $\alpha(1 - \alpha)$ respectively. Hence, the expected number of messages propagated within two units of time is $3\alpha^2 + 2(1 - \alpha)^2 + 4\alpha(1 - \alpha) + 3\alpha(1 - \alpha) = -2\alpha^2 + 3\alpha + 2$. The maximum is attained when $\alpha = 3/4$, i.e. $x = y/3$.

One can see that Node 2 pulls faster than Node 4 in the optimal solution. The reason is that Node 2 is more important in the social network due to its follower, namely, Node 3. This simple example shows that one needs to consider node importance in the social network when allocating pulling rate. We propose to use "network centrality" to capture the notion of social importance, which is a class of well-defined and widely-used metrics on nodes or edges. Common metrics for node centrality include the in-degree-, out-degree-, betweenness-, closeness-, eigenvector- and PageRank- centrality. While, in this simple example, the importance of Node 2 can be well-captured by in-degree-centrality, it is not necessarily the case in general. Instead of pinning down one specific centrality measure, we will start by an abstract notation and defer the specific choice of centrality metric to Section VI.

IV. Utility Maximization Formulation

We model the decentralized social network as a directed graph $G = (V, E)$ where each node $i \in V$ represents a user and E is the set of directed edges between users, i.e., $(i, j) \in E$ if there is an edge from i to j. Denote by $N(i)$ the set of followees of user i, namely, $N(i) = \{j \in V : (i, j) \in E\}$. Denote by x_s the aggregate pulling rate of user s from its followees in the pull-based model and c_s by the centrality of user s in the social graph. In the communication layer, each link $l \in L$ has a capacity C_l. Define N_l as the set which contains all the communication pairs that go through link l. Denote by R the set which contains all the physical machine and each user is hosted on a particular machine. Define $M(r)$ as the set which contains all the users hosted on machine r. To model the resource contention in application layer, we assume machine $r \in R$ has a processing capacity T_r which is the maximum number of pulling requests that machine r can handle in one unit of time. Further, we denote by $U_s(c_s, x_s)$ the utility of user s which is a concave and strictly increasing function of both x_s and c_s. Moreover, when $x_s \leq 0$, $U_s(c_s, x_s) = -\infty$. This guarantees that x_s is nonnegative when optimal solution is attained and we can omit the constraint $x_s \geq 0$ in the following discussion for clarity. The utility function can be interpreted as the contribution of each individual user in the social network.

We formulate the optimization problem with the objective to maximize the total utility of the network under the resource constraint in both communication layer and application layer. The DPCP protocol is the outcome of this optimization problem which can be formulated as P1 in the following:

$$\max \quad \sum_{s \in V} U_s(c_s, x_s) \tag{1a}$$

$$s.t. \quad x_s = \sum_{o \in N(s)} x_s^o \quad \forall s \in V \tag{1b}$$

$$\sum_{o \in M(r)} \sum_{s: o \in N(s)} x_s^o \leq T_r \quad \forall r \in R \tag{1c}$$

$$\sum_{(s,o) \in N_l} x_s^o \leq C_l \quad \forall l \in L \tag{1d}$$

In the formulation, c_s is a centrality metric which serves as a parameter of the utility function and x_s^o is the pulling rate of user s from its followee o. The first constraint states that the pulling rate of user s is the aggregation of all the individual pulling rates from its followees. The second constraint models the application layer bottleneck, e.g. the capacity of each host in terms of processing the message pulling request. The last constraint corresponds to the bottleneck of each link in the communication layer. In this constraint, we assume all the pulling requests incur the same amount of traffic and C_l is normalized to the maximum number of requests that can go through link l in one time unit.

This formulation is similar to the multipath TCP utility maximization problem when provided the centrality of each user. We will present the primal-dual approach to solve this optimization problem in the next section.

V. Pulling Rate Adjust Based on Delay

In this section, we assume c_s is a constant obtained from either prior knowledge or an auxiliary protocol. In this way, pulling rate x_s is the only optimization parameter and we design a delay-based algorithm to solve it optimally.

A. Characterizing the optima of P1

First, we write down the Lagrangian dual problem of P1 in the following and refer to it as P2.

$$\min_{p_r, \lambda_l} D(p_r, \lambda_l), \quad D(p_r, \lambda_l) = \max_{x_s^o} f(p_r, \lambda_l, x_s^o) \tag{2}$$

$$f(p_r, \lambda_l, x_s^o) = \sum_{s \in V} U_s(c_s, x_s) - \sum_{l \in L} \lambda_l \cdot \Big(\sum_{(s,o) \in N_l} x_s^o - C_l \Big) - \sum_{r \in R} p_r \cdot \Big(\sum_{o \in M(r)} \sum_{s: o \in N(s)} x_s^o - T_r \Big)$$

where p_r and λ_l are Lagrangian dual variables associated with the application layer and communication layer constraints respectively.

P1 is a convex optimization problem and there is no gap between the optimal values of the primal and dual problem. Furthermore, the optimal solutions must satisfy the following KKT equations:

$$\frac{\partial U_s(c_s, x_s^*)}{\partial (x_s^o)^*} - p_r^* - \sum_{l: (s,o) \in N_l} \lambda_l^* = 0; \quad \forall (s, o) \in E \tag{3}$$

$$p_r^* \cdot (\sum_{o \in M(r)} \sum_{s:o \in N(s)} (x_s^o)^* - T_r) = 0; \quad \forall r \in R \tag{4}$$

$$\lambda_l^* \cdot (\sum_{(s,o) \in N_l} (x_s^o)^* - C_l) = 0; \quad \forall l \in L \tag{5}$$

where $(x_s^o)^*$, p_r^* and λ_l^* are the optimal solutions to the dual problem. Denote by $\mathbf{x} = [x_s^o]_{s \in V, o \in N(s)}$, $\mathbf{p} = [p_r]_{r \in R}$, $\lambda = [\lambda_l]_{l \in L}$ and the set containing all $(\mathbf{x}, \mathbf{p}, \lambda)$ which satisfy the above KKT equations as ψ.

B. Primal-dual approach

P1 is similar to the multi-path TCP problem but not exactly. There are a lot of existing distributed algorithms which solve the multi-path TCP problem iteratively. For example, [9] reviews a branch of existing multi-path TCP algorithms which are packet-loss based. These algorithms are motivated from TCP Reno in the single-path case. To guarantee the convergence of these algorithms, the theoretical analysis in [9] imposes a rather strong assumption, namely, the utility function (if exist) must be strictly concave of all the variables. However, this condition does not hold in our optimization problem. Prior to [9], [1] analyzes the packet-loss based algorithm in Data Center Network which is also extended from TCP Reno. In theory, the packet-loss based algorithm corresponds to the primal approach [11]. Applying the primal approach in P1 is difficult due to the fact that it is nontrivial to define a good penalty function to incorporate constraints in both application layer and communication layer. Other than the primal approach, the standard dual subgradient algorithm can also solve this problem [5], [15]. However, the convergence of those dual variables in these approaches are typically slow.

On the other hand, the primal-dual approach was studied widely to develop distributed algorithms. The primal-dual approach in [12] can converge even when varying the link capacity. In this way, the problem considered in [12] is different from that of Eq. (3). [2] adopts the primal-dual approach to solve the problem of utility maximization in P2P systems and the algorithm can converge exponentially fast under some mild conditions. The solutions proposed in [2] can also be applied to the multi-path TCP problem.

In this way, we adopt the prima-dual approach to solve P2. The update of the primal and dual variables satisfy the following fluid-flow equations:

$$\dot{x_s^o} = \rho(x_s^o)[\frac{\partial U_s(c_s, x_s)}{\partial x_s^o} - p_r - \sum_{l:(s,o) \in N_l} \lambda_l]_{x_s^o}^+ \tag{6}$$

$$\dot{p_r} = \tau(p_r)[\sum_{o \in M(r)} \sum_{s:o \in N(s)} x_s^o - T_r]_{p_r}^+ \tag{7}$$

$$\dot{\lambda_l} = \eta(\lambda_l)[\sum_{(s,o) \in N_l} x_s^o - C_l]_{\lambda_l}^+ \tag{8}$$

where $\rho(x_s^o)$, $\tau(\lambda_r)$ and $\eta(p_l)$ are the stepsizes of the fluid-flow equations and $[x]_y^+$ is x if $y > 0$ and $\max\{x, 0\}$ otherwise. Every saddle point of $P2$ is an equilibrium of the above system. Observe that each user only needs to collect the information of dual variables locally and updates their pulling rate accordingly. Hence, we conclude this algorithm can be implemented in a distributed manner.

The following lemma shows that this primal-dual algorithm converges to a set, over which the aggregate pulling rate of each user is optimal. This can be proven by adopting the method of Lyapunov stability theory.

Lemma 1. *All $(\mathbf{x},\mathbf{p},\lambda)$ trajectories of the system in* (6), (7) *and* (8) *converge to a set which is denoted by $S = \{(\widetilde{\mathbf{x}},\widetilde{\mathbf{p}},\widetilde{\lambda}) : \widetilde{x_s} = \sum_{o \in N(s)} \widetilde{x_s^o} = x_s^*\}$.*

Proof. Refer to Appendix A. □

As $U_s(c_s, x_s)$ is a strictly concave function of x_s, $\widetilde{x_s}$ must be unique. In this way, the aggregate pulling rate of each user reaches the optima after the algorithm runs for a long enough time. However, it is possible that S contains some $(\widetilde{\mathbf{x}},\widetilde{\mathbf{p}},\widetilde{\lambda})$ that is not in ψ. If $(\widetilde{\mathbf{x}},\widetilde{\mathbf{p}},\widetilde{\lambda})$ moves onto these points, the system will keep oscillating and may never converge. One way to guarantee S not to contain these singular points is to apply the fact that $\widetilde{x_s} = \sum_{o \in N(s)} \widetilde{x_s^o} = constant\}$. This can be achieved by choosing appropriate values for $\rho(x_s^o)$, $\tau(\lambda_r)$ and $\eta(p_l)$ as described in the following section.

C. Processing Delay based Algorithm Design

The constraint (1c) and (1d) are mathematically equivalent to each other. This can be verified by creating a dummy node for each host. Following this observation, we omit the constraint (1d) to simplify the convergence analysis of the primal-dual approach. In practice, the link capacity in the communication layer is large enough comparing to the processing constraint in the application layer as well.

Each user still needs to measure p_r in Equation (6). For ease of implementation, we let $\tau(T_r) = \frac{1}{T_r}$. Similar to the queuing delay in the TCP Vegas design [11], the price p_r can be interpreted as the processing (queuing) delay in the application layer. In this way, the primal-dual approach can be readily be implemented in practice. With the help of the following theorem, the pulling rate for each user can converge to its global optimal value asymptotically.

Theorem 1. *When choosing $\rho(x_s^o) = 1$ for all s and $o \in N(s)$ with $\tau(p_r) = \frac{1}{T_r}$, the primal-dual approach can converge globally asymptotically to the optimal solution under some mild conditions.*

Applying the fact that $\widetilde{x_s} = \sum_{o \in N(s)} \widetilde{x_s^o} = constant$, we first prove that when choosing such a stepsize, p_r can converge to the optimal. Based on this result, we prove that each individual x_s^o also converges to the optimal. For the detailed proof, please refer to Appendix B.

VI. Distributed Centrality Estimation

The basic version of DPCP introduced in Section V assumes the centrality is known a priori or obtained from another auxiliary protocol. This makes the resultant protocol difficult to deploy on a SNSAPI-based ad-hoc DSN. Towards this end, we propose to enhance DPCP by estimating centrality while requiring passively-observed information only.

A. Compute Social Network Centrality Measures in a Distributed Fashion

Different centrality metrics capture nodal importance from different aspects. Examples are like degree-, closeness-, betweenness- and eigenvector- centrality. Degree centrality, either in-degree or out-degree, is easy to compute, because the information is locally available for all DSN nodes. However, the calculation of some centrality measures requires the knowledge of the entire social graph topology, which is not readily available in the DSN scenario. One way to solve this problem is to design an iterative algorithm so that local knowledge can eventually propagate to all other nodes. The challenge here is how to achieve this goal without an auxiliary protocol. That is, how can one node learn its own centrality based on passively-observed information only, e.g. by observing the changing behavior of its neighboring nodes. Although the goal sounds mythical at first glance, we show in the following subsection a first positive result, namely, the eigenvector centrality can be estimated in this way. Since the design could be case-by-case depending on the centrality metric of choice, the exploration of other centrality merics is left for future work.

Before delving into the details of estimating eigenvector centrality, we need to note that the centrality estimation runs on top of delay-based rate adjustment. That is, nodes first run enough number of iterations to get the optimal x_s. Then, nodes update c_s accordingly and to obtain a converged value of x_s again. This process continues until the whole system converges to an operating point such that: 1) c_s reflects the target centrality of choice and 2) x_s is the optimal pulling rate under this centrality metric.

B. Estimating Eigenvector Centrality via "Drumbeats" Algorithm

The eigenvector centrality for node o is defined as

$$c_o = \sum_{i:o\in N(i)} \frac{c_i}{|N(i)|}$$

It is also considered a specific version of PageRank, namely, without the escaping probability. The intuition is that the more important one node is, the more important its followee is. It improves degree-centrality by taking the importance of followers into consideration. This centrality metric is easy to calculate in either centralized or decentralized manner by iterating the above formula. With the same intuition, we propose to estimate it via the following framework:

$$\frac{c_s(k)}{m_s} = f(x_s^o(1), x_s^o(2), \cdots,) \ \forall s \in V; o \in N(s) \quad (9)$$

$$c_o(k) = \sum_{i:o\in N(i)} \frac{c_i(k-1)}{m_i} \quad \forall o \in V; \quad (10)$$

where k denotes the iteration and $m_i = |N(i)|$ denotes the number of followees. The question remaining is how to design such an f and the corresponding strategy for adjusting x_s^o.

The key idea is to encode the information of eigenvector centrality into the pulling rate. Note that the pulling rate alone means nothing because it reflects not only the centrality but also the degree of congestion. However, the change of pulling rate matters. That is, we can embed one node's current eigenvector-centrality into the step size when it executes Eq. (6). Recall that nodes update pulling rate according to

$$x_s^o(j+1) = x_s^o(j) + \alpha_s g(j) \quad (11)$$

where α_s is the step size that encodes c_s/m_s and $g(j) = [\frac{\partial U_s(c_s, x_s(j))}{\partial x_s^o(j)} - p_r]^+_{x_s^o(j)}$ is the gradient at step j. By observing the difference of pulling rates between two consecutive iterations, we can get $\alpha_s g(j)$. In order to decode the centrality, next question is how to cancel out $g(j)$. The idea is to divide one step in original formula to two sub-steps, which we call the even step and odd step:

$$x_s^o(2l+1) = x_s^o(2l) + \alpha_s^{(even)} g(2l) \quad (12)$$

$$x_s^o(2l+2) = x_s^o(2l+1) + \alpha_s^{(odd)} g(2l) \quad (13)$$

Since the g parts are the same, we can cancel it out by division. We want to carefully design the even/odd step sizes (a.k.a "Drumbeats") so that it has two properties: 1) the centrality is embedded in $\alpha_x^{(even)}/\alpha_x^{(odd)}$; 2) when the two sub-steps are combined, the resultant step size leads to convergence. One possible choice is:

$$\alpha_s^{(even)} = \frac{(1 + \frac{c_s/2}{c_s+m_s})}{c_s}, \alpha_s^{(odd)} = \frac{1}{2c_s + 2m_s} \quad (14)$$

The first property is natural by showing the f that estimates follower's centrality:

$$f(x_s^o(1), x_s^o(2), \cdots) = ave_l \left\{ \frac{2}{\frac{x_s^o(2l+1) - x_s^o(2l)}{x_s^o(2l+2) - x_s^o(2l+1)} - 3} \right\} \quad (15)$$

where $ave_l\{\cdot\}$ represents the average value. The second property is guaranteed by Theorem 2, which can be proven based on arguments similar to those of Theorem 1.

Theorem 2. *When choosing $\rho(x_s^o) = (1 + \frac{c_s}{c_s+m_s})/c_s$ for all s and $o \in N(s)$ with $\tau(p_r) = \frac{1}{T_r}$, the system determined by Equation (6),(7) can converge globally asymptotically to the optimal.*

Proof. Refer to Appendix C. □

VII. Practical Issues in the Implementation of DPCP Protocol

In this section, we address several practical issues for implementing the DPCP protocol.

A. Utility function design

Logarithm function is a widely adopted utility function to ensure both efficiency and fairness [13]. Thus, we adopt the following for our protocol design:

$$U_s(c_s, x_s) = \begin{cases} c_s \cdot \log x_s & x_s > 0 \\ -\infty & otherwise \end{cases} \quad (16)$$

It can be readily verified that the above utility function satisfies the requirements stated in Section IV.

Fig. 2. The model of pulling message

B. Update the pulling rate for each user

In our enhanced DPCP protocol, we adopt the eigenvector centrality described in Section VI as the centrality metric. Users pull the messages from their followees by dynamically adjusting their pull gap. This can be achieved by taking the pull gap as the inverse of x_s^o which is determined by Equation (12) and (13). After the pulling rate has stabilized, the followee updates its new centrality estimated according to Equation (15) and (9). The message pulling process begins with user s updating its current centrality as shown in Fig. 2. After the first pulling-request for new messages, user s waits for $\Delta_1 = \frac{1}{x_s^o(1)}$ units of time before sending the second request to user o. It then waits for another Δ_1 time before sending the third request. To request again, user s needs to wait for $\Delta_2 = \frac{1}{x_s^o(2)}$ time. So on and so forth, user s follows the model in Fig. 2 to send its following requests, namely, $\Delta_{2l+1} = \frac{1}{x_s^o(2l+1)}$ and $\Delta_{2l} = \frac{1}{x_s^o(2l)}$. We design the two successive pulls with the same pull gap to distinguish between the even step and odd step in Equation (12) and (13). As a result, when the followee observes three pulling requests of which the time gap are the same, it can easily match each pulling request to the corresponding item in Equation (15).

C. Measuring the delay in two layers

Measuring the processing delay accurately is particularly important for the protocol design. Once a user sends a request to one of its followees, it then measures the duration of the time period till it receives the response. This duration is the sum of the propagation delay, queuing delay in communication layer and processing delay in application layer. The propagation delay can be treated as a constant value. By subtracting the duration of the shortest time period in the history which is treated as the propagation delay, we can get a measurement for the sum of the current queuing delay and processing delay in the two layers.

VIII. Simulation Results

In this section, we conduct a set of simulation experiments under the NS-3 environment to evaluate the performance of the proposed DPCP protocol and "Drumbeats" Algorithm.

A. Message generation and dissemination model

Assume that each node periodically generates new messages at a constant rate. Once a follower sends a pulling request to its followee, the followee will reply with the latest new messages generated by itself or pulled from others. If there is no new message, the followee just replies a null message. In addition, there is a limit for the number of messages contained in each reply. For social networks like Facebook, Weibo and Renren, when users forward a message, they usually add some of their own comments. This technically makes the forwarded message a new one. Our dissemination model also takes this factor into account. For example, if node B and node C both forward a message from A, the two actions will generate two different new messages.

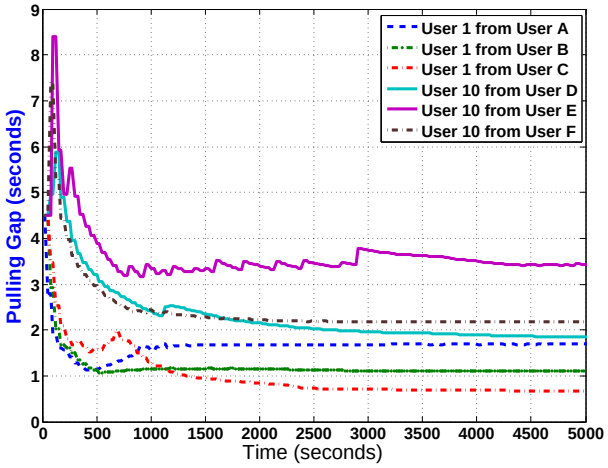

Fig. 3. The updates of the pulling gap in DPCP.

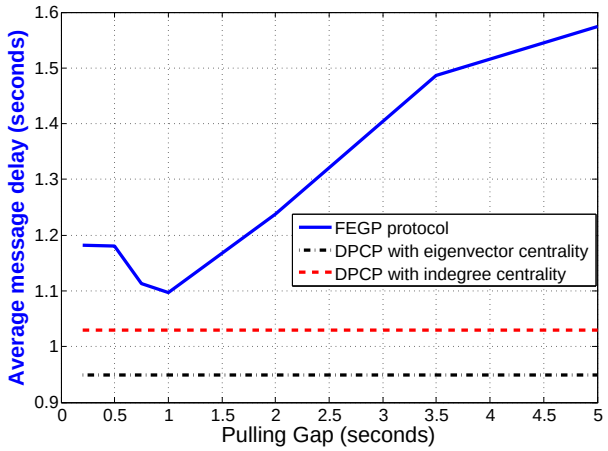

Fig. 4. The comparison of the average message propagation delay between DPCP and FEGP protocol.

B. Performance evaluation for DPCP

Simulation topology: We crawled 600 users from a real social network to obtain the simulation topology. Every user is hosted on its own physical machine and the machine processing capacity is uniformly distributed between five and ten requests per second.

Evaluation Metric: Note that we derive the optimal solution for the optimization problem and the objective value under DPCP is therefore expected to be better than that of the baseline FEGP approach. Below, we first use the average message delay as the performance metric for evaluation. In the message generation model described in Section VIII-A, every *original* message will eventually be propagated throughout the entire network. We record the elapsed time for every message to be propagated to the whole network and take use the corresponding average over all the messages as the evaluation metric.

Baseline: We adopt the FEGP protocol as the baseline under which every node pulls the message from its followee at a

constant rate. We then adjust the pull-rate of FEGP over a wide range and compare the results with the DPCP protocol.

Centrality: To evaluate the rate adaptation performance of DPCP, we first use the ground-truth value of the particular centrality metric computed according to the topology of the social network. Two different centrality metrics are considered, namely, the node-in-degree-centrality and eigenvector-centrality.

1) Convergence performance: We first evaluate the convergence behaviour of DPCP. The communication layer is omitted for brevity because our protocol depends only on the measurement of the end-to-end delay and is oblivious to the physical topology. The simulation result is shown in Fig. 3. Here we randomly choose two users where each user has three followees to illustrate the updates of their pulling gaps.

2) Delay performance: We adopt the in-degree centrality and eigenvector-centrality in DPCP to caputure the delay performance. We try different pulling gaps under FEGP to determine the best one. Results depicted in Fig.4 show that the average message delay under the eigenvector-centrality is shorter than that under the in-degree centrality. Even when FEGP uses its best parameter, DPCP still reduces the average message delay by 15%.

C. Performance evaluation for "Drumbeats" algorithm

To evaluate the performance of the "Drumbeats" algorithm, we use a small topology for ease of demonstration. The simulation topology is illustrated in Fig. 5.

In real implementation, the update of centrality and pulling rate can be interleaved with each other. More explicitly, once a user detects the update of the eigenvector-centrality from one follower through the pulling gap, it then updates its new centrality immediately without waiting for the convergence of pulling gap of this follower(s). Moreover, the update does not rely on the centrality measure of the other followers.

Note that our protocol can automatically handle the cases when a user join or leave the DSN. Whenever a new user joins the DSN, it initializes its own eigenvector centrality to be a constant and begins to pull messages from its followees. When a user leaves, other users can rapidly adapt their own eigenvector centrality.

Fig. 6 shows the estimation process of all the nodes. (The behavior of node 7 is similar to node 5 and is therefore excluded from the figure). Observe that the estimated value of the eigenvector centrality stabilizes within 20000 seconds where the initial centrality for each node is 1. Furthermore, the estimation results match well with the ground-truth centrality directly derived from the network topology. This demonstrates that the effectiveness of the "drumbeats" algorithm.

IX. Conclusions and Future works

In this paper, we have formulated a utility maximization problem to coordinate nodes in a pull-based Decentralized Social Network (DSN). One distinguishing feature of our formulation is the consideration of social network layer centrality. This is in sharp contrast to previous work in P2P content

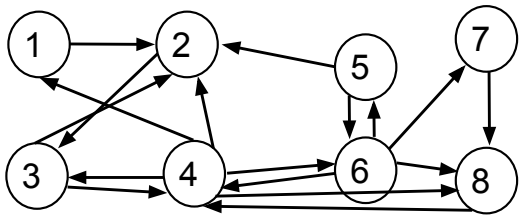

Fig. 5. Experiment topology for "drumbeats" algorithm. Each arrow represents a following friendship, for example, User 1 follows User 2.

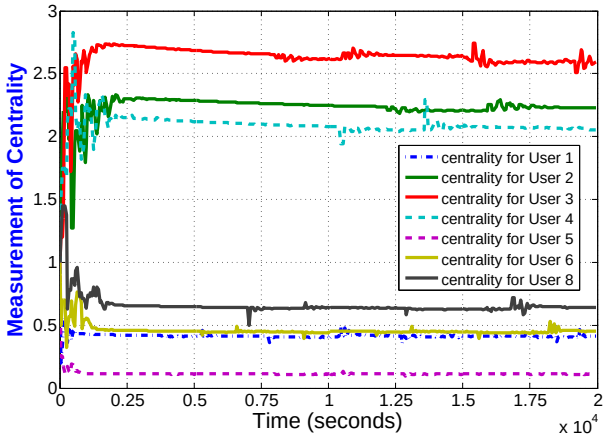

Fig. 6. The measurement process of centrality for each node according to "Drumbeats" Algorithm.

delivery which focuses on maximizing the content delivery data rate. We have also developed a novel "drumbeats" algorithm that let nodes estimate their own eigenvector centrality based on passively-observed information only. Previous study has shown that migration is the biggest challenge in moving towards a decentralized paradigm. Towards this end, the ability of DPCP to support incremental deployment is critical for bootstrapping DSN.

There are several extensions of the current work. First, we can compare the DPCP protocol with other baselines, including p-persistent style back-off protocol and plain TCP style back-off protocol (without consideration of node importance in social network). The deployment of DSN nodes, e.g. via SNSAPI, may support both pull and push based model. So we can also develop a protocol for push-based backbone in the future. In the current work, only centrality on static graph is evaluated, i.e. PageRank on friendship graph. One extension is to consider node centrality in the dynamic activity graph, e.g. PageRank using effective message delivery as edges. Note that the current work does not account for resource limit of follower side. When the follower has limited bandwidth or CPU cycles, it may allocate different rate to different followees. One example is to also consider the message posting frequencies of followees, so that resources are further saved.

References

[1] M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of DCTCP: Stability, convergence, and fairness. In *Sigmetrics*, June 2011.

[2] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou. Utility maximization in peer-to-peer systems. In *Sigmetrics*, June 2008.

[3] P. Hu, Q. Fan, and W. C. Lau. SNSAPI: a cross-platform middleware for rapid deployment of decentralized social networks. In *CoRR abs/1403.4482*, 2014.

[4] B. Jiang, N. Hegde, L. Massoulie, and D. Towsley. How to optimally allocate your budget of attention in social networks. In *IEEE Infocom*, April 2013.
[5] K. Kar, S. Sarkar, and L. Tassiulas. Optimization based rate control for multirate multicast sessions. In *IEEE Inforcom*, April 2001.
[6] Lefechetz, S., Lasalle, and J.P. *Stability by Lyapunov's direct method with applications*. Academic Press, New York, 1961.
[7] J. Li, P. A. Chou, and C. Zhang. Mutualcast: An efficient mechanism for content distribution in a peer-to-peer (p2p) network. In *Proceedings of Acm Sigcomm Asia Workshop*, April 2004.
[8] J. Mundinger, R. Weber, and G. Weiss. Optimal scheduling of peer-to-peer file dissemination. *Journal of Scheduling*, 11(2):105–120, April 1993.
[9] Q. Peng, A. Walid, and S. H. Low. Multipath TCP algorithms: Theory and design. In *Sigmetrics*, June 2013.
[10] S. Sanghavi, B. Hajek, and L. Massoulie. Gossiping with multiple messages. *IEEE Transactions of Information Theory*, 53(12), December 2007.
[11] S. Shakkottai and R. Srikant. *Network Optimization and Control (Foundations and Trends(r) in Networking)*. Hanover, MA, USA, January 2008.
[12] T. Voice. *Stability of congestion control algorithms with multi-path routing and linear stochastic modelling of congestion control*. PhD Dissertation, University of Cambridge, Cambridge, UK, May 2006.
[13] L. Ye, Z. Wang, H. Che, H. B. C. Chan, I. Ghosh, and C. M. Lagoa. Utility function of TCP and its application to the design of minimum rate guaranteed control law, 2009.
[14] S. Zhang, Z. Shao, and M. Chen. Optimal distributed p2p streaming under node degree bounds. In *International conference on network protocols*, October 2010.
[15] H. Zhao, C. H. Xia, Z. Liu, and D. Towsley. Distributed resource allocation for synchronous fork and join processing networks. In *IEEE Infocom*, March 2010.

APPENDIX

A. Proof of Lemma 1

We prove Lemma 1 by adopting the method of Lyapunov stability theory [6]. We first define the following energy function:

$$V(x_s^o, p_r, \lambda_l) = \sum_{s\in S}\sum_{o\in N(s)}\int_{(x_s^o)^*}^{x_s^o}\frac{(\xi-(x_s^o)^*)}{\rho(x_s^o)}d\xi + \sum_{r\in R}\int_{p_r^*}^{p_r}\frac{(\xi-p_r^*)}{\tau(p_r)}d\xi + \sum_{l\in L}\int_{\lambda_l^*}^{\lambda_l}\frac{(\xi-\lambda_l^*)}{\eta(\lambda_l)}d\xi \tag{17}$$

where $(x_s^o)^*$, p_r^* and λ_l^* are the optimal solution to P2. It can be shown that $V(x_s^o, p_r, \lambda_l)$ is positive definite. Denote by $\dot{V}(x_s^o, p_r, \lambda_l)$ the derivative of $V(x_s^o, p_r, \lambda_l)$ with respect to time. We proceed to prove that $\dot{V} \le 0$ in the following:

$$\begin{aligned}\dot{V} = &\sum_{s\in V}\sum_{o\in N(s)}[h(x_s^o) - p_r - \sum_{l\in L}\lambda_l]_{x_s^o}^+\cdot(x_s^o-(x_s^o)^*)\\ &+\sum_{r\in R}[\sum_{o\in M(r)}\sum_{s:o\in out(s)}x_s^o - T_r]_{p_r}^+\cdot(p_r-p_r^*)\\ &+\sum_{l\in L}[\sum_{(s,o)\in N_l}x_s^o - C_l]_{\lambda_l}^+\cdot(\lambda_l-\lambda_l^*)\end{aligned} \tag{18}$$

where $h(x_s^o) = \frac{\partial U_s(c_s,x_s)}{\partial x_s^o}$. It's straightforward to get

$$\begin{aligned}\dot{V} \le &\sum_{s\in V}\sum_{o\in N(s)}(h(x_s^o) - p_r - \sum_{l\in L}\lambda_l)\cdot(x_s^o-(x_s^o)^*)\\ &+\sum_{r\in R}(\sum_{o\in M(r)}\sum_{s:o\in N(s)}x_s^o - T_r)\cdot(p_r-p_r^*)\\ &+\sum_{l\in L}(\sum_{(s,o)\in N_l}x_s^o - C_l)\cdot(\lambda_l-\lambda_l^*)\end{aligned} \tag{19}$$

For ease of presentation, let G denote the right-hand side of Equation (19), then

$$\begin{aligned}G = &\sum_s(\sum x_s^o - \sum(x_s^o)^*)(U'(\sum x_s^o) - U'(\sum(x_s^o)^*)\\ &+\sum_o\sum_{s:o\in N(s)}(x_s^o-(x_s^o)^*)(U'(x_s^o)^* - p_r^* - \lambda_l^*)\\ &+\sum_{r\in R}(\sum_{o\in M(r)}\sum_{s:o\in N(s)}(x_s^o)^* - T_r)\cdot(p_r-p_r^*)\\ &+\sum_{l\in L}(\sum_{(s,o)\in N_l}(x_s^o)^* - C_l)\cdot(\lambda_l-\lambda_l^*)\le 0\end{aligned}$$

Observe that in the above equation, $G = 0$ holds only when $\sum_{o\in N(s)}x_s^o = \sum_{o\in N(s)}(x_s^o)^*$. Then based on Lasalle's Principle in [6], we obtain lemma 1 immediately.

B. Proof of Theorem 1

When the communication layer is not the bottleneck, the update of the pull rate can be modeled as

$$\dot{x_s^o} = \rho(x_s^o)[\frac{\partial U_s(c_s,x_s)}{\partial x_s^o} - p_r]_{x_s^o}^+ \tag{20}$$

$$\dot{p_r} = \tau(p_r)[\sum_{o\in M(r)}\sum_{s:o\in N(s)}x_s^o - T_r]_{p_r}^+ \tag{21}$$

Under the assumption that the projection is inactive, the above system is a linear system.

Here, we prove a more general result, i.e., when $\rho(x_i^j) = \rho(x_k^j) = a_j\ \forall i,k$ and $j\in N(i)\cap N(k)$ and $\tau(p_r) = \frac{1}{T_r} = q_r$. Moreover, $a_j = a_k$ if user j and user k are hosted on the same physical machine. The system determined by Equation (20) and Equation (21) will converge to the optimal.

Denote by $X_{n\times n} = [x_1^1, x_1^2, \cdots, x_1^n; \cdots; x_n^1, x_n^2, \cdots, x_n^n]$ where x_i^j is the pull rate of user i from its followee j and $n = |V|$. Further define $\mathbf{e} = [1; 1; \cdots; 1]$ with length n. Following the result of Lemma 1, we have

$$X\cdot\mathbf{e} = constant \tag{22}$$

Hence, the following equation also holds:

$$\dot{X}\cdot\mathbf{e} = \mathbf{0} \tag{23}$$

where $\dot{X}$ is the derivative of X with respect to time. Denote by $A = dig[a_1, a_2, \cdots, a_n]$, then $\dot{X}$ can be represented by

$$\dot{X} = \begin{pmatrix} h(x_1^1)-p_1 & h(x_1^2)-p_2 & \cdots & h(x_1^n)-p_n\\ h(x_2^1)-p_1 & h(x_2^2)-p_2 & \cdots & h(x_2^n)-p_n\\ \cdots & & & \\ h(x_n^1)-p_1 & h(x_n^2)-p_2 & \cdots & h(x_n^n)-p_n\end{pmatrix}\cdot A$$

where $h(x_i^j) = \frac{\partial U_i(c_i,x_i)}{\partial x_i^j}$ and $p_i = p_j$ if user i and user j are hosted on the same machine. Actually, when the algorithm runs a long enough time, the aggregate pulling rate of each user is optimal, thus $h(x_i^j) = h((x_i^j)^*)$. Let $\mathbf{p} = [p_1, p_2, \cdots, p_n]$ denote the row vector, we get

$$\mathbf{p}\cdot A\cdot\mathbf{e} = constant \tag{24}$$

Similarly,

$$\dot{\mathbf{p}}\cdot A\cdot\mathbf{e} = 0 \tag{25}$$

where $\dot{\mathbf{p}}$ is the derivative of $\mathbf{p}$ with respect to time. Denote by $Q = dig[Q1, Q2, \ldots, Q_r]$ where Q_r is a $M(r) \times M(r)$ matrix with all the elements being q_r. Based on Equation (21), $\dot{\mathbf{p}}$ can be expressed as

$$\dot{\mathbf{p}} = \mathbf{e^T} \cdot X \cdot Q \tag{26}$$

Combine Equation (25) and (26), it holds that

$$\mathbf{e^T} \cdot X \cdot Q \cdot A \cdot \mathbf{e} = 0. \tag{27}$$

Take the derivative of Equation (27), we get

$$\mathbf{p} \cdot (AQ) \cdot A \cdot \mathbf{e} = constant \tag{28}$$

More generally, we get that

$$\mathbf{p} \cdot (AQ)^l \cdot A \cdot \mathbf{e} = constant \quad \forall l \in \mathbb{N} \tag{29}$$

Denote by $z_i = |M(i)|$ and $m = |R|$, it's easy to check that

$$(AQ)^l A\mathbf{e} = [\underbrace{(z_1 a_1 q_1)^l a_1, \ldots (z_1 a_1 q_1)^l a_1}_{z_1}, \ldots, \underbrace{(z_m a_m q_m)^l a_m}_{z_m}]^T \tag{30}$$

Define matrix G as following

$$G = \begin{pmatrix} z_1 a_1^2 q_1 1 & z_2 a_2^2 q_2 & \cdots & z_m a_m^2 q_m \\ z_1^2 a_1^3 q_1^2 & z_2^2 a_2^3 q_2^2 & \cdots & z_m^2 a_m^3 q_m^2 \\ \cdots & \cdots & \cdots & \cdots \\ z_1^m a_1^{m+1} q_1^m & z_2^m a_2^{m+1} q_2^m & \cdots & z_m^m a_m^{m+1} q_m^m \end{pmatrix}$$

Combine Equation (29) and (30), it follows that

$$G \cdot \mathbf{p^T} = constant \tag{31}$$

G is a Vandermonde matrix, if $z_i a_i q_i \neq z_j a_j q_j \ \forall i, j$; the determinant of G is nonzero which implies that $\mathbf{p} = constant$. In this way, $\dot{\mathbf{p}} = \mathbf{0}$. This further indicates that $h(x_s^o) = p_r \ \forall s$ and $o \in N(s)$; otherwise, $\frac{\partial U_s(c_s, x_s)}{\partial x_s^o} - p_r > 0$ and x_s^o goes to infinity which contradicts with Lemma 1.

Following the above argument, we get $\dot{x_s^o} = 0 \ \forall s$ and $o \in N(s)$ and it completes the proof.

C. Proof of Theorem 2

The idea to prove this theorem is very similar to Theorem 1. However, we need to represent the pulling rate in a column vector instead of matrix, namely, X is represented by the following vector:

$$\mathbf{X} = [x_1^1, x_2^1, \cdots, x_n^1, x_1^2, \cdots, x_n^2, \cdots, x_1^n, x_2^n, \cdots, x_n^n]$$

In $\mathbf{X}$, all the pulling rate from the users hosted on the same machine are put together.

Here, we prove a more general result, namely, when $\rho(x_i^j) = \rho(x_i^k) = \varrho_i \ \forall j, k \in N(i)$ and $\tau(p_r) = \frac{1}{T_r} = \tau_r$, the system determined by Equation (20) and Equation (21) will converge to the optimal. The proof is as follows.

Denote by

$$P_{n^2 \times n} = [I_{n \times n}, I_{n \times n}, \ldots, I_{n \times n}]$$

where $I_{n \times n}$ is the identity matrix with size $n \times n$. According to Lemma 1, it holds that

$$P \cdot \mathbf{X} = constant. \tag{32}$$

Hence, $P \cdot \dot{\mathbf{X}} = 0$. Define $\mathbf{r} = [\varrho_1, \varrho_2, \cdots, \varrho_n]^T$ which is a column vector with dimension n and $R_{n^2 \times m}$ as

$$R = \begin{pmatrix} \mathbf{r} & 0 & \cdots & 0 \\ \cdots & & & \\ \mathbf{r} & 0 & \cdots & 0 \\ 0 & \mathbf{r} & \cdots & 0 \\ \cdots & & & \\ 0 & \mathbf{r} & \cdots & 0 \\ \cdots & & & \\ 0 & 0 & \cdots & \mathbf{r} \end{pmatrix}$$

where every single column corresponds to the stepsize of the pulling rate update from each physical machine. In this way, the following equation holds

$$P \cdot R \cdot \underbrace{[p_1, p_2, \ldots, p_m]^T}_{\triangleq \mathbf{p_1}} = constant \tag{33}$$

thus, $P \cdot R \cdot \dot{\mathbf{p}}_1 = 0$. Further define row vector $\mathbf{d^i}_{(1 \times n z_i)} = q_i \cdot [\underbrace{1, 1, \ldots, 1}_{n \cdot z_i}]$ and matrix $D_{(m \times n^2)} = dig\{d^1, d^2, \ldots, d^m\}$, then $\dot{\mathbf{p}}_1$ can be represented by

$$\dot{\mathbf{p}}_1 = D \cdot \mathbf{X} \tag{34}$$

In this way,

$$P \cdot R \cdot D \cdot \mathbf{X} = constant \tag{35}$$

Similarly, take the derivative of Equation (35), we have

$$P \cdot R \cdot D \cdot R \cdot \mathbf{p_1} = constant$$

Repeat this process, we obtain

$$\begin{pmatrix} P \cdot R \\ P \cdot (RD) \cdot R \\ \cdots \\ P \cdot (RD)^{m-1} \cdot R \end{pmatrix} \cdot \mathbf{p_1} = constant \tag{36}$$

On the other hand, $P \cdot (RD)^{m-1} \cdot R$ is given by the following matrix which is defined as K^{m-1}:

$$\begin{pmatrix} (z_1 \chi q_1)^{m-1} z_1 \varrho_1 & (z_2 \chi q_2)^{m-1} z_2 \varrho_1 & \cdots & (z_m \chi q_m)^{m-1} z_m \varrho_1 \\ (z_1 \chi q_1)^{m-1} z_1 \varrho_2 & (z_2 \chi q_2)^{m-1} z_2 \varrho_2 & \cdots & (z_m \chi q_m)^{m-1} z_m \varrho_2 \\ \cdots & \cdots & \cdots & \cdots \\ (z_1 \chi q_1)^{m-1} z_1 \varrho_n & (z_2 \chi q_2)^{m-1} z_2 \varrho_n & \cdots & (z_m \chi q_m)^{m-1} z_m \varrho_n \end{pmatrix}$$

where $\chi = \sum_{i=1}^n \varrho_i$.

Take the first column of each K^l for $l = 0, 1, 2, \ldots, m-1$ and build a new matrix $\mathbb{K}$ in the following

$$\underbrace{\begin{pmatrix} z_1 & z_2 & \cdots & z_m \\ (z_1 \chi q_1)^1 z_1 & (z_2 \chi q_2)^1 z_2 & \cdots & (z_m \chi q_m)^1 z_m \\ \cdots & \cdots & \cdots & \cdots \\ (z_1 \chi q_1)^{m-1} z_1 & (z_2 \chi q_2)^{m-1} z_2 & \cdots & (z_m \chi q_m)^{m-1} z_m \end{pmatrix}}_{\triangleq \mathbb{K}}$$

Following Equation (36), we get

$$\mathbb{K} \cdot \mathbf{p_1} = constant \tag{37}$$

$\mathbb{K}$ is a Vandermonde matrix, if $z_i q_i \neq z_j q_j \ \forall i, j$; the determinant of $\mathbb{K}$ is nonzero which implies that $\mathbf{p_1} = constant$.

The rest of this proof is completely the same as the proof in Theorem 1.