A Control Theory Approach to Self-stabilization in Large Distributed Systems

Hui Fang and Yong Meng Teo
Singapore-MIT Alliance
National University of Singapore
3 Science Drive 2, Singapore 117543
email: {g0404400, teoym}@comp.nus.edu.sg

February 9, 2005

Abstract

The stability of large distributed systems plays an important role in fault tolerance. However, it is generally difficult to evaluate the distributed stability in practice. This paper proposes to divide the stability into node stability and global stability. The node stability depicts the correct routing information acquired by a node due to network change, for example: node failure or leave. The stability of the system called global stability is derived by each node's stability. We show that this approach reflects the real stability of the overall system. To study the stability, we model a large distributed system as a dynamic system consisting of two parameters p and q, which impacts the stability in contrary way. For example: p and q can in practice model the query rate and flush-stale-node rate, respectively. From the properties induced by asymptotic analysis, a self-stabilizing algorithm is proposed based on (p, q) feedback system. This algorithm shows that each node can asynchronously achieve the predefined stability in finite time. And global stability can be obtained by setting a global threshold.

1 Introduction

Self-stabilization is an attractive property for modeling fault-tolerance in large distributed systems. The conception of self-stabilizing distributed computation was first proposed and explored by Dijkstra [2] in 1974. More recent survey of self-stabilization can be found in [12]. [10] gave a formal view of fault tolerance in asynchronous distributed computing environments. A distributed system is *self-stabilizing* if a) when started from an arbitrary initial state, it is guaranteed to reach a legitimate state b) once in a legal state, the system does not switch to an illegal state in the absence of failures. The first property is called *convergence*, the latter is called *closure*. It is important that when the system goes to unstable state, proper actions are taken to put the system back to stable state. This property is highly desirable for fault tolerance in distributed systems, without global or external intervention, i.e., the system corrects itself automatically and distributively, in finite time from perturbations or failures.

The self-stabilizing protocols based on Dijkstra state transition model are discussed as in [9]. Some other less formal methods such as leader election algorithm on tree graphs, and many different graph algorithms are also discussed in [7, 16]. A distributed system can be constructed using uniform [9] or non-uniform networks. Specifically, Peer to Peer(P2P) system favors the

uniform algorithm which allows each node to perform similar actions [9, 16]. Furthermore, when noticing that in P2P networks all the network structure information inconsistency is caused by the node actions (leave, join and fail) and routing updates, we quickly focus our attention on the message-passing mechanism in the network, which directly tells the sender or receiver the partial network structure information and impacts the whole network stability. Some practical protocols based on automatic routing operations are proposed as in [1].

Since the theory of dynamic systems focuses on the system stability, it is also natural to apply control theory to distributed computer systems. Taylor and Tofts gave a control theory perspective on self managed systems [13] and considered the basic conceptions such as convergence, oscillations, and chaotic behavior. However, it is not easy to find "good" control functions, in other words, control model to describe the dynamically changing system. The physics-style approach [15] attempts to analyze the perturbation behavior of distributed networks under self-repair policies by using intensive variables. The experiments show that the specified variable may be the critical network parameter related to stability.

Since it is generally very hard to solve the dynamic equations and even harder to obtain useful information from the complicated solution format, some researchers use the Lyapunov second method to achieve the qualitative analysis results on system stability [11, 8, 14]. The key of Lyapunov second method is to find a suitable Lyapunov function V that satisfies the positive-definite around $(0, 0)$ state and the differential of V on time t is negative-definite. Then we can guarantee the dynamic system is asymptotically stable.

The paper is organized as follows. In Section 2 we present our overall idea on the model of system stability. Section 3 defines global stability and node stability. Section 4 introduces the two important parameters related to system stability and gives an analysis on the evaluation mechanism based on control theory. Then we design a feedback system by measuring the parameters. An algorithm is proposed for this purpose. In section 5 we draw a conclusion on our technique and discuss the future work.

2 Our Approach

In a large distributed system such as grid, P2P system, the computing elements or nodes exchange information by asynchronous message passing. Every node locally maintains state information about other nodes. Global state can be derived as the sum of the local states of all the nodes in the system. Each node has a partial view of the global state, and this depends on the connectivity of the system and the propagation delay of different messages. We acknowledge the stability a kind of networks structure information consistency, and try to find the crucial parameters related to network stability. Based on these parameters, we design the uniform self-stabilizing algorithm that is run on each node to maintain overall state stable.

We make the following assumptions:

- Node can only communicate with neighbors that have pointer in its routing table
- Both the links and node may fail and recover during normal operation
- The recovery should be without global intervention, but system will consider the stability in global state sense
- Each node keeps to some extent the knowledge about others to avoid lost, such as a list

of neighbors

First, we define global stability and node stability. Second, we model the dynamic system stability using two parameters, p and q. Third, we analyze the asymptotical property on system stability and convergence time limit of the equations and propose an algorithm that dynamically maintains the global stability at a certain level. Finally, the conclusion is given and further possible research is elaborated. The main contribution of this paper lies in the clear definition of stability and the uniform self-stabilizing algorithm based on control theory.

3 Stability Definitions

A distributed system is modeled as a directed graph, G = (V, E), where V is a set of nodes denoting peers, or grid resources, and E is the set of edges which denote the virtual connection among nodes. $|V| = n$ is a stationary network size. To be convenient, the n can be limited, or chosen as big as enough to contain all possible expanding nodes. The edge $e_{ij} = \langle v_i, v_j \rangle$ means node v_i knows node v_j. The stability is defined as the accumulation of knowledge of all nodes about other nodes. At initial state E_0, each node may know little about others. When time goes on, each node may know more neighbors and their parameters. The knowledge could also be impacted or decreased due to network failure(node leaving/failure and link failure). The stability is regarded higher when each node knows well about each other. Here we try to depict the stability of distributed system by considering the edge and node failure separately.

- d_{ij}: The degree of stability of an edge $e = \langle v_i, v_j \rangle, range \quad [0, 1]$. In the simple case, $d_{ij} = 1$ if node v_i contains the pointer to v_j without considering other impact. If no recorded pointer, the value of d_{ij} can be equal to 0. This can be improved by another ideal but more complicated measurement which is to assign the value of $\frac{shortest\ path\ length}{network\ diameter}$. This parameter reflect the interconnected stability of the nodes. It is impacted by virtual connectivity.

- u_i: The average failure rate of node i. For one specified node, it can be sampled and regarded as a fixed number of the node, or we can say, a property of the node. This parameter reflect the inner stability of the node. It is impacted by physically causes.

- w_i: The weight of stability of node i. Here $w_i = 1 - u_i$.

- w_{ij}: The weight added to d_{ij} caused by the possible failure of node v_i and v_j. Obviously, $w_{ij} = w_i * w_j = (1 - u_i) * (1 - u_j)$.

The following two matrices define the stability of G:

$$D = \begin{pmatrix} d_{11} & \dots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nn} \end{pmatrix}, w = \begin{pmatrix} w_1 \\ w_2 \\ ... \\ w_n \end{pmatrix}.$$

Node stability, x_i is defined as:

$$x_i = \frac{w_i}{n} \sum_{j=1}^{n} d_{ij} * w_j, and\ X = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} = \frac{1}{n} w \cdot (D \star w)$$

Here we use the point product to help to denote the concept of X. Thus, global stability, S, based on D and w is:

$$S(D,w)=\frac{1}{n}\sum_{i=1}^{n}x_i=\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}d_{ij}*w_{ij}=\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}w_i*d_{ij}*w_j=\frac{1}{n^2}w'Dw$$

If G is an undirected graph, matrix D is symmetric. If we assume the node failure rate is a constant u for all nodes (This is a reasonable assumption for P2P network), then

$$S(D,w)=\frac{(1-u)^2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}d_{ij}=\frac{(1-u)^2}{n^2}\sum_{i=1}^{n}d_i,\quad where\quad d_i=\sum_{j=1}^{n}d_{ij}.$$

In this case, S depends on the sum of each node knowledge about its neighborhood. We can connect d_i from the routing table information in each node. If

$$D=\begin{pmatrix}1 & \dots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{pmatrix},$$

every node has a complete routing information about others. In this case the system can be regarded as much stable and the stability S only depends on the probability of node failure. Another example is that there is no node failure and d_{ij} is binary valued(u=0 and $d_{ij}=1$ if v_{ij} is in G), seen in Figure 1. The stability is indicated in the following Table 1.

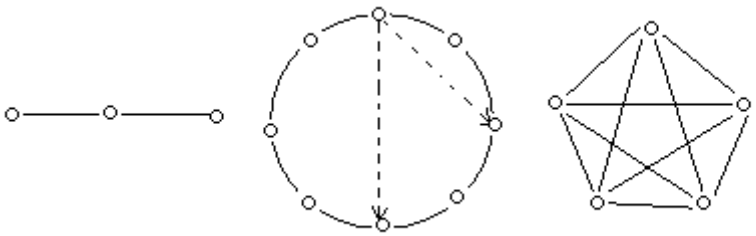

Figure 1: Different kinds of graph

Graph	Stability
Thread: n nodes connected in a line	$S=(2/n+3/n*(n-2)+2/n)/n=(3n-2)/n^2$
Chord: each node contains log(n) fingers	$S=log(n)/n$
Graph with n node full connected	$S=1$

Table 1: Graph stability comparison

As we know, the $log(n)/n$ stability degree in Chord ensures that we can retrieve the data efficiently [5]. Some researchers also relax the $O(logn)$-state-per-node restriction to unlimited state [4]. However, in many cases we may only need to achieve stability on certain level other than the full stability. Therefore the current problem is whether we can and how to achieve a predefined stability in finite time.

4 System Model and Analysis

Each node contributes to global stability in different proportions. So our problem now is how to evaluate and maintain the stability of each node? We at first analyze why each node lose connection or how to get updates. As the formula defined in section 3, the stability of node i is x_i. Consider x_i as a function of time t, we have $x_i = x_i(t)$. To be concise without misunderstanding, in the following context the $x(t)$ is used instead of $x_i(t)$. Consider the routing inconsistency. There are in general two ways to change the routing entries.

- An incoming message updates or adds new routing entries (new pointer to other node). If a new joined node wants to know more routing information about others, it can periodically send out query messages and then wait for the responding messages to find more useful routes. This kind of resource discovery will consume bandwidth to some extent due to extra messages. More mechanisms about resource discovery in distributed networks can be found in [21].
- A node flushes the outdated entries in its routing table, in terms of out-going message timeout or specific policy, because the incorrect or out of date routing entries influence the stability of the system routing.

Hence, we propose to model the impact on system stability using two parameters:

- p: Models a factor that improves stability, such as the number of searching messages the node has sent out that results in the activation of new routing entries.
- q: Models a factor that decreases stability, such as the frequency at which the specified routing entries are deleted due to time out.

The stability difference between new $x(t)$ and previous $x(t)$ is caused by two parts: The portion found in the instability $(1 - x(t))$, i.e. $p(1 - x(t))$; the portion lost from the current stability $x(t)$, i.e. $-qx(t)$. In case where both p and q are constant, we have the following differential equation(The case when p, q change over time will be discussed later):

$$\frac{dx(t)}{dt} = p(1 - x(t)) - qx(t), \qquad where\ p, q\ in\ [0, 1]$$

The above linear time-invariant ODE has solution and steady point x*:

$$x(t) = \frac{p}{p+q}(1 - ke^{-(p+q)t}), \qquad where\ k = 1 - \frac{p+q}{p} * x(0)$$

k is a constant that is dependent on the x(t) initial state. The trajectory curve of $x = x(t)$ is shown in Figure 2.

The meaning of p and q can be further understood in the following way:

- If p, the query rate, is increased, the final value of node stability $\frac{p}{p+q}$ also increases. When $p >> q$, the stability converges to 1, i.e., if query operations are frequent, stale routing entries will always be replaced by new-added entries. For example, Epichord [4] uses the parallel lookup to get quick updates of the routing table.
- If q, the flush rate, is increased, the stable value of node stability will decrease at first sight. However we also notice that: Although increasing q decreases the stability, it also decreases the convergence time to steady point!

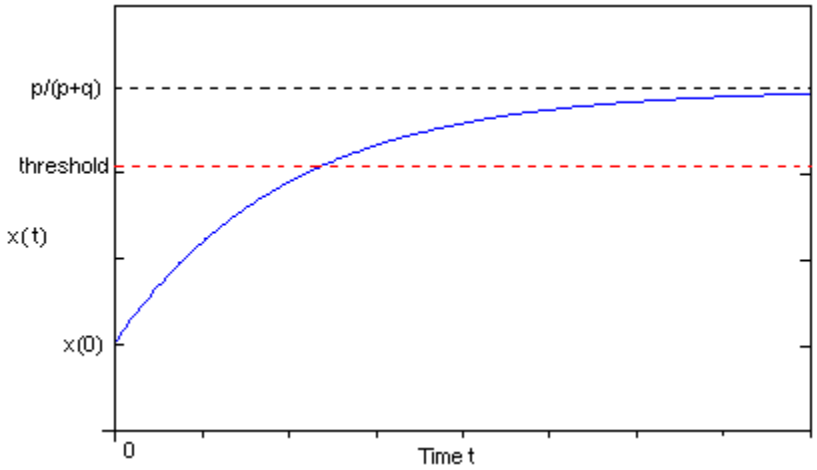

Figure 2: x(t) curve when p and q are constant

- Although the maximum stability $\frac{p}{p+q}$ is not reachable, node can still choose one threshold value which is bigger than $x(0)$ and less than maximum as a goal to achieve.

Based on the above observations, we further analyze p and q as functions of abstract time t, as follows.

$$\frac{dx(t)}{dt} = p(t)(1 - x(t)) - q(t)x(t), \ \ where\ p(t), q(t)\ in\ [0, 1]. \tag{1}$$

Property 1: If $\frac{q(t)}{p(t)}$ has lower bound C, and if x* exists, then $x^* > \frac{1}{1+C}$.

Proof: Equation (1) is a time-variant, linear differential equation. The solution and stationary point is in the form:

$$x(t) = \frac{\int_0^t p(\xi) e^{\int_0^\xi (p(\tau)+q(\tau))d\tau} d\xi + x0}{e^{\int_0^t (p(\tau)+q(\tau))d\tau}},$$

$$x* = \lim_{t\to\infty} x(t) = \lim_{t\to\infty} \frac{p(t)}{p(t)+q(t)} > \frac{1}{1+C}. \qquad (Use\ \ L'Hospital\ Rule) \quad \square$$

This means if we keep $\frac{q(t)}{p(t)} = O(1)$, i.e. the query rate is higher than constant times flush rate, when the state turns to stable, the stability will certainly be higher than a positive constant.

Property 2: If $\frac{q(t)}{p(t)} = C$ is constant, and if there exists $\varepsilon : p(t) > \varepsilon > 0, when\ \ t > t_\varepsilon$, then

1. x(t) is asymptotically stable.
2. x(t) is increasing and x* $= \frac{1}{(1+C)}$ is upper bound, i.e. for all x(t), $x(t) < x^*$.
3. For all constant $s : 0 < s < x^*$, the state s is reachable and the convergence time $Ts < \frac{1}{\varepsilon(1+C)} * \log \frac{1-(1+C)x_0}{1-(1+C)s}$.

Proof: Substitute q(t) = C*p(t) into Equation (1), we have Equation

$$\frac{dx}{dt} = p(t)[1 - (1 + C)x(t)], \tag{2}$$

Set $\xi = \frac{1}{1+C} - x$, then $\frac{d\xi}{dt} = -(1+C)p(t)x(t)$. Then we create function $V(\xi) = \xi^2$. It has two properties: 1) $V(0) = 0; V(\xi) > 0 \quad when \quad \xi \neq 0$,where 0 is zero-solution; 2) $\frac{dV}{dt} = \frac{dV}{d\xi} * \frac{d\xi}{dt} = -2(1+C)p(t)\xi^2$. Since $p(t) > \varepsilon$, we have $\frac{dV}{dt} < 0$. So $V(\xi)$ is a Lyapunov Function. Based on Lyapunov Second Method, $\xi = \xi(t)$is asymptotically stable for zero-solution, i.e., x(t) is asymptotically stable on point $\frac{1}{1+C}$.

Notice that Equation(2) is solvable and the solution is:

$$x(t) = \frac{(1 - k * e^{-(1+C)\int_0^t p(\tau)d\tau})}{(1+C)} < \frac{1}{1+C}.$$

Let x(t) = s. We already have $p(t) > \varepsilon > 0$, that implies

$$Ts < \frac{1}{\varepsilon(1+C)} * \log \frac{1-(1+C)x_0}{1-(1+C)s} \qquad \square$$

We note that the result (1) and (2) from Property-2 is not dependent on the initial value of x(t). With respect to result (3), if we do not care the initial state, the following inferred formula can also be used:

$$Ts < \frac{1}{\varepsilon(1+C)} * \log \frac{1}{1-(1+C)s}$$

The result (3) shows that convergence time depends on how stability we already get (we will achieve), and the proportion C.

The above results lead us to design a feedback system and algorithm that dynamically adjusting factor p(t) and q(t) in each step. Thus, the stability can be maintained at a specified level efficiently. Since the discrete systems are abstractions of a continuous world, we believe the design of self-stabilizing algorithm based on the continuous model yield the insight into the world for the discrete case. The similar question is also motivated in the paper [17]. The feedback (p,q) system is described as Difference Equation:

$$x_{t+1} = x_t + p_t(1 - x_t) - q_t x_t$$

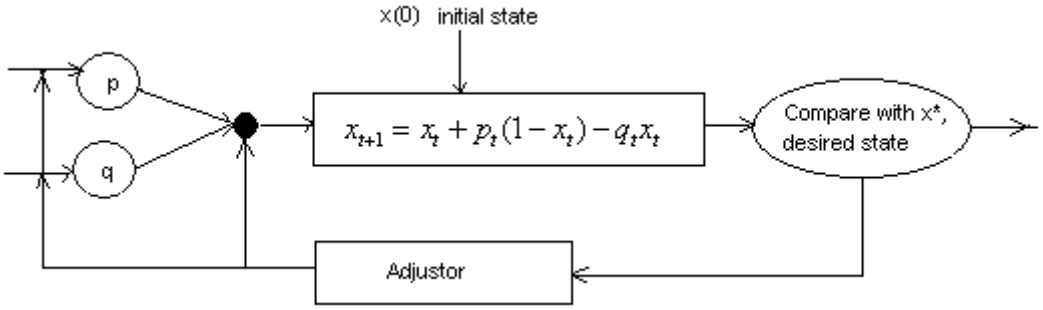

Figure 3: (p,q)-Feedback system

The Adjustor in Figure 3 is to compare the x(t) with desired state x* and adjust the parameters p and q. The algorithm is in Figure 4.

According to Property 2, this algorithm running on the node can achieve the predefined stability s* in limit time, if we notice that the algorithm keeps the invariance $\frac{p}{p+q} = s*$ and there always

Initialization step: Upon joining, each node keeps its recent known neighbor information $x(0)$, and the local parameters (p, q) related to query rate and flush rate: $p = p_0 > 0, q = q_0 > 0$;

Maintenance step: Repeat each time, by adjusting the p, q and computing new local stability. For each time t:

if value of x(t) < desired s* then
Increase p;
increase the of initiated query messages
Decrease q, satisfying $\frac{p}{p+q} = s*$;

else if value of x(t) > desired s* then
Contrary action

else **value (x(t)) = desired s* ** then
Keep the p and q
endif
endif

All incoming message:
Record the pointer of source node without duplication.
** as a result: p increased as expected **
if Δp is not enough
Send out searching messages for routing discovery
$x = x + \frac{\Delta p}{n}$

All outgoing message:
if no response back in timeout, then
Mark as "to be flushed"
endif
while current flush < q
Flush the stale routing entries
endwhile
$x = x - \frac{\Delta q}{n}$

Figure 4: Algorithm-1 of achieving node stability x* in finite time

is $p(t) > 1$. Obviously it is true that not all the nodes reach the maximum local stability at the same time. So whenever we need a global stability degree, one possible method is to gather and sum up all the current local stabilities, then compare with an expected value. The sum operation will become simple if we consider the P2P network in which each node obeys the same node failure rate. There's no need to check the stability before the computed time limit. This action can be deferred to a certain time limit when almost all nodes reach the stable state. So our idea is to disseminate the information of goal s* and convergence time T in the whole network; Then for each node,if it knows its stability is much lower than s*, it will speed up the increasing stability process, adjusting (p, q).

Based on the above dynamically self-stabilizing and the existing diffusing algorithms [3], the revised version of algorithm can be easily written out in order to adjust the global stability in specified time if we know it is achievable. The only problem which may be encountered in implementation depends on how we keep the invariance, or the precondition of Property-2 at all time.

The algorithm of achieving the specified global stability s is described in Figure 5.

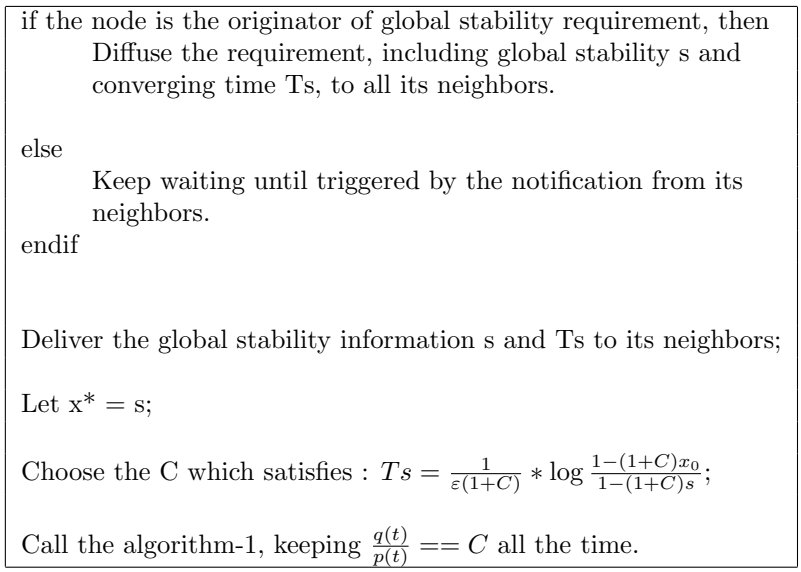

Figure 5: Algorithm-2 of achieving global stability s in finite time

Since this algorithm requires the global stability information to be distributed to every node in the network, the convergence time is actually summed up by two parts: One is the time execution for distribution of global stability seed, the other is the time execution for node stability adjustment.

One of the advantages of the algorithm is that once after the node receives the value of global stability, the rest work is to merely adjust the node stability locally. At this time it is not necessary for each node to coordinate with each other, before the new requirement on stability arrives. This property is especially useful in the asynchronous distributed environment [3] to achieve stable global state. There is no need to make periodical global snapshot to verify the termination-detection, because each node only focuses to finish its own job, that is, achieve a specified stability in finite time, then done. Since the expected convergence time can be evaluated by the result of control theory, it is quite promising that all the nodes in the connected network could reach the stability in the same predefined time. The center server administration and coordination between peers are apparently reduced to the smaller phase.

5 Conclusions and Future Work

In this paper, we analyze the node behavior in large distributed system and give a practical evaluation on the global stability and node stability, which we believe fully denotes the inner structure of system stability distribution. Then we sort out two important parameters which impact the node stability, and further show by differential equation that if keeping the two co-related parameters vary in certain bound, the goal stability for each node can be achieved in limit time. The proof is based on the Lyapunov second method about the stability of dynamical system. This technique is especially attractive when we have set up a suitable model [8, 14].

To make a system having a self-stabilizing property, a practical algorithm was proposed to achieve the specified stability degree on each node. The key is to satisfy the invariance condition defined in the theorem proof. Based on the proof result, the further global stability can be

predicted by coordinating the predefined goal among network in advance. Future work will be extended to the experiment verification which is not done in this paper.

The global stability of network may not fully depend on a certain threshold value. For example: although the stability degree is very low such as thread-like network structure, it can still tolerant some node failures such as two end-points failure. Actually the global stability also depends on the network topology, or stability distribution matrix we specified in previous section. We can imagine the stability Matrix D as a kind of *stability distribution*. In some areas (sub-matrix) the density of stability may be higher, while other areas with lower stability. In particular, a sparse stability distribution may have lower fault tolerance. Some nodes with locality may also comprise super-node. Hence we will consider the stability of local nets and the stability between two local areas.

So our method is to gather the dense area nodes into one super node, compute the stability in the above same way, and then consider the interaction between area A and area B. This is possible further research. The knowledge may cover artificial intelligence, such as Hopfield neural network [20], DCS (Distributed Cell System) may be combined to go on research on stability of P2P networks.

References

[1] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, *Maintenance-Free Global Data Storage*, IEEE Internet Computing, October 2001.

[2] E.W. Dijkstra, *Self-stabilizing Systems in spite of Distributed Control*, Communications of the ACM, 17(11), 643-644, 1974.

[3] N. Lynch, *Distributed Algorithms*, San Francisco, Calif. : Morgan Kaufmann Publishers, Chapter 19, pp617-639, 1996

[4] B. Leong, B. Liskov, and E.D. Demaine, *Epichord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management*, MIT Technical Report. MIT-LCS-TR-963, (Cambridge, MA), Aug. 2004.

[5] A. Gupta, B. Liskov, and R. Rodrigues, *Efficient Routing for Peer-to-peer Overlays*, in Proceedings of the 1st Symposium on Networked Systems Design and Implementation(NSDI 2004), March 2004,pp.113-126.

[6] Y.M. Teo, M. Verdi, and X. Wang, *A Framework for Distributed Grid Resource Indexing and Discovery*, National University of Singapore, September 2004.

[7] G. Antonoiu, and P.K.Srimani, *A Self-Stabilizing Leader Election Algorithm for Tree Graphs*, Computer Science Technical Report,Colorado State University,1996.

[8] O. Theel, *A New Verification Technique for Self-stabilizing Distributed Algorithms based on Variable Structure Systems and Lyapunov Theory*, Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.

[9] M. Flatebo, A.K. Datta, and S. Ghosh, *Self-Stabilization in Distributed Systems*, IEEE Computer Society Press, 1994.

[10] F. Gartner, *Fundamentals of Fault-tolerant Distributed Computing in Asynchronous Environments*, ACM Computing Surveys, vol 31, No 1. March. 1999.

[11] G. Teschl, *Ordinary Differential Equations and Dynamical Systems*, University of Wien, Austria, Section 6.5, pp105, 1991.

[12] M. Schneider, *Self-stabilization*, ACM Computing Surveys, 25:45-67, March 1993.

[13] R. Taylor, and C. Tofts, *Self Managed Systems - A Control Theory Perspective*, SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems, Italy, June 2004

[14] S. Yerramalla, E. Fuller, M. Mladenovski, and B. Cukic, *Lyapunov Analysis of Neural Networks Stability in an Adaptive Flight Control System*, Self-Stabilizing Systems:77-91, 2003

[15] S. El-Ansary, E. Aurell, P. Brand, and S. Haridi, *Experience with A Physics-style Approach for the Study of Self Properties in Structured Overlay Networks*, SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems, Italy, June 2004

[16] H.J. Hoover, and P. Rudnicki, *Uniform Self-stabilizing Orientation of Unicyclic Networks Under Read/Write Atomicity*, Chicago Journal of Theoretical Computer Science, Aug. 1996.

[17] H.J. Hoover, *On the Self-Stabilization of Processors with Continuous State*, Department of Computing Science, University of Alberta, Canada, http://www.cs.ualberta.ca/, 1995.

[18] J. Doyle, *Fundamental Tradeoffs in Robustness of Complex Systems*, Control and Dynamical Systems, Caltech. http://www.cds.caltech.edu/ doyle/CmplxNets/, 2003.

[19] S. Goldenstein, E. Large, and D. Metaxas, *Non-linear Dynamical System Approach to Behavior Modeling*, The Visual computer(1999) 15:349-364,Springer-Verlag 1999.

[20] A. Jagota, *Hopfield Neural Networks and Self-Stabilization*, Chicago Journal of Theoretical Computer Science,vol 1999,article 6,MIT Press,Aug.1999

[21] M. Harchol-Balter, T. Leighton, and D. Lewin, *Resource Discovery in Distributed Networks*, 18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing, Atlanta, May 1999, pp. 229-238.