**Distributed Computing manuscript No.**
(will be inserted by the editor)

**Indranil Gupta · Mahvesh Nagda · Christo Frank Devaraj**

# The Design of Novel Distributed Protocols from Differential Equations

Received: date / Accepted: date

**Abstract** This paper proposes a framework to translate certain subclasses of differential equation systems into practical protocols for distributed systems. The generated protocols are intended for large-scale distributed systems that contain several hundreds to thousands of processes. The synthesized protocols are state machines containing probabilistic transitions and actions, and they are proved to show equivalent stochastic behavior to the original equations. The protocols are probabilistically scalable and reliable, and have practical applications in large-scale distributed systems, e.g., peer to peer systems. In order to illustrate the usefulness of the framework, it is used to generate new solutions for the problems of (i) responsibility migration (giving rise to a novel model of dynamic replication), and (ii) majority selection. We present mathematical analysis of these two protocols, and experimental results from our implementations. These two protocols are derived from natural analogies that are represented as differential equations - endemics and the Lotka-Volterra model of competition respectively. We believe the design framework could be effectively used in transforming, in a very systematic manner, well-known natural phenomena into protocols for distributed systems.

This research was supported in part by NSF CAREER grant CNS-0448246 and in part by NSF ITR grant CMS-0427089.

I. Gupta
Department of Computer Science
University of Illinois at Urbana-Champaign.
201 N. Goodwin Ave., Urbana, IL, USA 61801.
Ph: 1 217 265 5517. Fax: 1 217 265 6494.
E-mail: indy@cs.uiuc.edu

M. Nagda
Wolverine Asset Management.
3555 Scottsdale Circle, Naperville, IL 60564.
E-mail: mnagda@wolve.com

C.F. Devaraj
Microsoft Corporation.
2400 Elliott Ave, 214, Seattle, WA 98121.
E-mail: chdevara@microsoft.com

**Keywords** Science of Protocol Design, Differential Equations, Distributed Protocols, Scalability, Reliability, Replication, Endemics, Voting, LV, Probabilistic Protocols.

## 1 Introduction

Much attention has been paid to studying the scalability and reliability of distributed protocols that run inside large-scale distributed systems containing hundreds to thousands of processes, such as peer to peer systems, the Grid, the Internet, and the Web. However, the *science of design* of scalable, efficient, and reliable protocols for such large-scale distributed systems remains a difficult problem. In this paper, we describe a framework to translate sets of differential equations (called a "system of differential equations", or abbreviated as "equation system") into a protocol for a large-scale distributed system. The techniques generate a state machine where states are derived from variables in the original equations, and actions and transitions are derived from the terms in the original equations. The source equation systems are required to be in a certain form - we describe this through a taxonomy, and also present equation rewriting techniques to enable conversion of other equation systems to the appropriate mappable form.

In order to illustrate practical utility of the presented methodology, we present two case studies where natural processes represented as differential equations are used to design practicable protocols for a dynamic and migratory responsibility migration scheme and for probabilistic majority selection. The two protocols are respectively called endemic replication and the Lotka-Volterra (LV) protocol.

Previously, differential equations have been used to study and analyze algorithmic solutions to problems such as independent vertex sets and degree distributions [28], 3-SAT [1], randomized load balancing [20], etc. The analyzed solutions are usually randomized algorithms.

However, little of the above work has addressed the converse direction, i.e., a design methodology to translate differential equations into distributed protocols. Such a methodology can be invaluable because scientists and researchers in various scientific disciplines express their ideas and results by using the language of differential equations. These ideas can then be systematically converted into distributed protocols that consequently have well-known behavior (by virtue of the original equations).

Our protocols can be seen as a variant of probabilistic I/O automata [29], however the cited work gives a broad specification of protocols, rather than a framework for building and analyzing protocols. Several randomized protocols have been proposed to circumvent the FLP impossibility of consensus result [12], starting from Rabin's work in [22]. Distributed computing with infinitely many processes, and the relation to finite groups, was studied in [17,19,20]. Strogatz's textbook [26] gives a thorough coverage for the analysis of non-linear dynamic equation systems.

The protocols generated by our framework have the common characteristics that they offer probabilistic reliability, utilize low and scalable bandwidth at each process, and have low convergence times. Most importantly, the distributed protocol inherits the stochastic behavior of the original differential equation system, e.g., the existence of stable equilibrium points in the original equations implicitly map into self-stabilizing behavior in the protocol. The mathematical analysis of these protocols borrows techniques from the study of non-linear dynamics. One particular style using *phase portraits* and simple perturbation analysis is appropriate for our goals, and has not been used before to study distributed protocols. The protocols are intended for asynchronous network settings, however our analysis makes certain simplifying assumptions.

In summary, the contributions of this paper are:
(i) A novel framework to translate differential equations, from two subclasses, into equivalent distributed protocols;
(ii) A natural taxonomy of differential equations, and equation rewriting techniques that facilitate translation;
(iii) Use of this framework to design new scalable protocols for majority selection and responsibility migration;
(iv) Use of analytical techniques from the study of non-linear dynamics in analyzing distributed protocols; and
(v) A toolkit called *DiffGen* that when given differential equations in Mathematica-like format, spews out compilable and executable C code.

**System Model:** We assume a closed group $G$ of $N$ processes, communicating over an asynchronous network. The closed group assumption means that there are no joins by new processes; simulations show that our protocols work in open groups. For simplicity, our analysis assumes that clock drifts are negligible; our results also extend to systems with clock drift. Our initial analysis assumes that processes do not fail, and all messages are delivered within a timeout; however, we discuss how to account for known rates of crash-stop failure and network message loss. Each process also knows about the maximal group membership, i.e., the remaining $(N-1)$ processes [1]. Our protocol analysis will show that if the group size $N$ is finite but large enough, this gives equivalent behavior to when $N$ is infinite. Our specification of the translated distributed protocols uses several variables to represent fractions of processes in a given state of a finite state machine. Thus, our equivalence proof allows us to assume that variation of these variables is continuous in time (as opposed to discrete).

**A Motivating Example - Epidemics:** The epidemic spread of diseases and rumors can be represented through differential equations [4]. In a closed group (i.e., no participant joins) of $N$ participants, let the *fractions* of infected and susceptible processes be $y$ and $x$ respectively; $x = 1 - y$. Let variable $t$ represent time. Then:

$$\frac{dx}{dt} = \dot{x} = -xy$$
$$\frac{dy}{dt} = \dot{y} = xy \tag{0}$$

A *pull* epidemic protocol can be designed from these equations. This is done by creating a state machine with two states - $x$ (susceptible) and $y$ (infected). The actions in the state machine involve susceptible processes periodically sampling the group for infectives (average protocol period fixed at all processes). This is nothing but the canonical pull epidemic protocol, which is well-known, analyzed, and used in systems such as Clearinghouse [11]. For unfamiliar readers, the canonical pull epidemic algorithm works by having each process $p$ that has not received the multicast (i.e., is susceptible) periodically contact one other process selected uniformly at random from the group. If the remote process has received the multicast (i.e., is infected), it sends the multicast to $p$, which then turns infected. The analysis predicts that as $t \to \infty$: $x(t) \to 0, y(t) \to 1$, and it takes an expected $O(log(N))$ rounds to reach $x \simeq O(\frac{1}{N})$ w.h.p.

We remind the reader that epidemics as discussed above are only an example to motivate discussion of the design framework. Epidemics are *not* the focus of this paper; epidemics were analyzed by Demers et al. in [11].

The rest of the paper is organized as follows. Section 2 gives a taxonomy of differential equation systems. Section 3 presents techniques to map a subset of polynomial differential equation systems (from the taxonomy) into distributed protocols. Sections 4 and 5 depict the power of the framework in generating scalable protocols for useful problems, along with analysis and experimental results for these protocols. Section 6 extends the protocol design framework to differential equation systems

[1] Well-known results can be used to reduce this size to logarithmic in group size.

with polynomial and arbitrary terms. Section 7 presents equation rewriting techniques that extend the framework to differential equations not initially expressed in a mappable form. Section 8 presents details of a design toolkit called DiffGen that automates the entire above framework, allowing rapid and automatic generation of deployable protocol code. We conclude in Section 9.

## 2 A Taxonomy of Differential Equation Systems

In order to clearly classify the differential equation classes that can be converted into distributed protocols, we first describe a taxonomy of differential equation systems. We first consider systems of equations that have a single differential per equation, and are of order and degree 1. Section 7 discusses translations for equations of higher order and degree.

Let $X$ denote a set of $m$ independent variables, where each individual variable lies in the interval $[0, 1]$. Let $\bar{X}$ be a finite-sized vector of these variables. We are concerned only with systems of differential equations in the form $\dot{\bar{X}} = \bar{f}(\bar{X})$. Here, $\bar{f}$ refers to an $m$-sized vector of functions, with each function in $m$ variables and each function is *finite*, i.e., it has a finite number of occurrences of each variable from $X$. $\dot{\bar{X}}$ refers to a vector of the $m$ variables in $\bar{X}$, each differentiated once with respect to time, i.e., $\dot{\bar{X}} = \frac{d}{dt}(\bar{X})$. We also denote the equation for variable $x \in X$ as $f_x(\bar{X})$.

We further limit ourselves to only differential equations that make sense solely in the interval $\bar{X} \in [0, 1]^m$, and that have $\Sigma_{x\in X} x = 1$. Notice that since each individual variable $x$ in $X$ is such that $x \in [0, 1]$, $f$ is defined only on $[0, 1]^m$. This restriction means that for each $x$ in $X$: (i) whenever $x = 1.0$, it should be the case that $f_x(\bar{X}) \leq 0$, while (ii) whenever $x = 0.0$, it should be the case that $f_x(\bar{X}) \geq 0$. Each of these equations is thus of order 1 (highest derivative) and degree 1 (power of highest derivative). Equation system (0) is an example, where $\bar{X} = \{x, y\}$, $\bar{f} = \{-xy, xy\}$. All the case studies considered in this paper (i.e., the endemic protocol and the LV protocol) satisfy this limited definition.

We wish to express each $f_x(\bar{X})$ as a *sum* of elementary *terms*, where each term is either positive or negative. Based on the actual definition of *term* in $\bar{f}$, we can identify the following two classes of polynomial equations:

**Polynomial Equation System:** An equation system, that follows the above limited definition, is said to be *polynomial* if for each equation $\dot{x} = f_x(\bar{X})$ in the system, $f_x(\bar{X})$ can be written as a sum of *polynomial terms*. Each polynomial term $T$ is defined to be of the form $\pm c_T \Pi^{y\in X}(y^{i_{y,f_x,T}})$, where all $i_{y,f_x,T}$'s are non-negative integers. $c_T$ is a constant specific to the term, and we restrict $c_T \in [0, 1]$.

**Non-polynomial System:** An equation system, that follows the above limited definition, is said to be *non-polynomial* if there is an $f_x(\bar{X})$ (for some $x \in X$) that cannot be written as a sum of polynomial terms. We define a *non-polynomial term* as an elementary unit that appears in $f_x$ ($x \in X$), but is neither a polynomial term, nor can be written as a sum of component polynomial and non-polynomial terms [2].

Equation systems belonging to a particular subclass of polynomial equation systems are amenable to translation into distributed protocols:

**Restricted Polynomial Equation System:** An equation system $\dot{\bar{X}} = \bar{f}(\bar{X})$ is restricted polynomial if (i) it is polynomial, and (ii) for each $f_x(X)$, it is true that each negative term $-T = -c_T \Pi^{y\in X} y^{i_{y,f_x,T}}$ that occurs in it ($c_T \in [0, 1]$) has $i_{x,f_x,T} >= 1$. Equation system (0) is an example of a restricted polynomial equation system.

Finally, based on the structure of $f$, equation systems can also be classified into two kinds:

**Complete Equation Systems:** An equation system $\dot{\bar{X}} = \bar{f}(\bar{X})$ is said to be *complete* if and only if $\Sigma^{x\in X}\dot{x} = \Sigma^{x\in X} f_x(\bar{X}) = 0$. In other words, the right hand sides all sum to zero. Without loss of generality, we will henceforth assume $\Sigma^{x\in X} x = 1$.

**Completely Partitionable Equation Systems:** An equation system $\dot{\bar{X}} = \bar{f}(\bar{X})$ is said to be *completely partitionable* if and only if (i) it is complete, and (ii) the multi-set consisting of all terms in $\bar{f}(\bar{X})$ can be written as a union of mutually disjoint subsets, where each subset contains exactly two terms, and the two terms in each subset add up to zero (i.e., one term is the negative of the other in the pair). For example, equation system (0) is completely partitionable.

## 3 Mapping an Equation System that is Completely Partitionable and Restricted Polynomial

The key idea in translating a *completely partitionable* differential equation system, into a distributed protocol, involves creating a state machine that contains (a) one state per basic variable in the original equation system, and (b) *actions* mapped onto these states. We denote the corresponding state for a given variable $x$ simply as "state $x$".

Behaviorally, each given variable $x$ is mirrored in the protocol as the fraction of processes in the system that

[2] This definition is left intentionally loose since this paper does not deal with rigorous translation techniques for such non-polynomial terms.

are in state $x$. Terms in the equation system are mapped to actions relevant to the states in the state machine, and the actions ensure that for each variable $x \in X$, the rate of inflow of processes into a state $x$ is always $f_x(\bar{X})$.

**Translating a Completely Partitionable Equation System:** Since a completely partitionable system can be split into pairs of positive-negative terms, we translate each such term pair into an action. If the negative term occurs in $f_x$ and the positive term occurs in $f_y$, the action is executed only by a process in state $x$, and if successful, causes a transition of that process into state $y$.

Each process uses a parameter called *protocol period duration*, denoted as $\tau$ time units. All processes use the same value for $\tau$. Processes run completely asynchronously, and there is no time synchronization across processes. Our analysis assumes that all clock drifts are negligible; our results also extend to systems with clock drifts or varying protocol periods (see Corollary 1.B in Section 3.1). In addition, our protocols are not sensitive to protocol period variation at a small number of processes in the system. The protocols do not require either global clocks, or global synchronization, or agreement.

Each process uses protocol period durations *in order to decide what time the next action will be executed* at that process. We describe the actions of a typical process $p$ here. For each $y \in X$, denote as $k_y$ the number of *negative* terms occurring in $f_y$. Suppose $p$ is currently in state $x$. Process $p$ decides the time of its next action according to a memoryless (exponential) probability distribution that has an average of $\frac{\tau}{k_x}$ time units. In other words, the time to next action $T_{next}$ is selected from a cumulative distribution function $Pr.(T_{next} \leq T) = 1 - e^{-\frac{T}{\tau} \cdot k_x}$. Process $p$ then waits for another $T_{next}$ time units, at which it picks a *random action* from among the $k_x$ actions for each of the *negative terms* appearing in $f_x$.

This use, by a process in state $x$, of both the memoryless distribution with average $\frac{\tau}{k_x}$ time units and the picking of a random action from among $k_x$ actions, gives us the following observation.

*Observation:* Consider $N \to \infty$. At any time, for each $x \in X$, if $x =$ fraction of nodes currently in state $x$, for each action $A$ defined for a negative term appearing in $f_x$ (and its positive counterpart term), the expected number of $A$ actions executed throughout the group during an infinitesimal time interval $dt$ is $\frac{dt}{\tau}.N.x.k_x \cdot \frac{1}{k_x} = N.x.\frac{dt}{\tau}$.

Finally, note that the above discussion applies to any equation system that is completely partitionable, regardless of the nature of its terms. In Section 3.1, we describe how to generate actions for restricted polynomial terms. Section 6 describes how to generate actions for other term types.

## 3.1 Generating Actions from Polynomial Terms

Consider a polynomial term of the form $-T = -c.\Pi^{y \in X} y^{i_{y,f_x,T}}$, occurring in $f_x$, with the corresponding $+T$ term occurring in $f_y$. If $-T = -c.x$, it can be translated into a *Flipping* action, otherwise it is translated into a *One-time-sampling* action.

**Flipping:** A term of the type $-c.x$ ($c \in [0,1]$) occurring on the right hand side (henceforth "r.h.s.") of $\dot{x} = f_x(\bar{X})$ is mapped to a flipping action. A process in state $x$ *locally* tosses ("flips") a biased coin that has heads probability $c$. Only if the coin turns up heads, does the process transition out of state $x$; the new state is $y$, where $\dot{y} = f_y(\bar{X})$ contains the corresponding $+c.x$ term, and $y \neq x$. Notice that for one execution of the flipping action at a given process, the probability of success, and thus state transition, is $c$.

**One-time-sampling:** A term $-T$ of the type $-T = -c.x^{i_{x,f_x,T}}.\Pi^{y \in X-\{x\}}(y^{i_{y,f_x,T}})$ ($c \in [0,1]$) that occurs on the r.h.s. of $\dot{x} = f_x(\bar{X})$ with $i_{x,f_x,T} >= 1$ is mapped into the following action. A process in state $x$ samples $(i_{x,f_x,T} - 1 + \Sigma^{y \in X-\{x\}}(i_{y,f_x,T}))$ other processes, each such target process selected uniformly at random from across the group. In addition, the process also flips *locally* a biased coin that has heads probability $c$. Let the variables $y \in X$ be ordered lexicographically. Then the process makes a transition out of state $x$ (and into the corresponding state with the $+T$ term) if and only if (i) each of the first $i_{x,f_x,T} - 1$ target choices happen to be in state $x$; and (ii) for each $j : 1 \leq j \leq (\Sigma^{y \in X-\{x\}}(i_{y,f_x,T}))$, we have that the $j^{th}$ process sampled is in the same state as the $j^{th}$ variable in $\Pi^{y \in X-\{x\}}(y^{i_{y,f_x,T}})$ (when ordered lexicographically); and (iii) the flipped local coin falls heads [3]. Notice that for one execution of this one-time-sampling action at a given process, the probability of success, and thus state transition, is $c.x^{(i_{x,f_x,T})-1}.\Pi^{y \in X-\{x\}}(y^{i_{y,f_x,T}})$.

Below, we analyze the behavior of the derived protocol when $N \to \infty$ (call this the *infinite analysis*: see Corollary 1.A). However, before doing so, we need to first show that this analysis is in fact equivalent to when one does the *finite analysis* with finite $N$ and then applies the limit $N \to \infty$ to this finite analysis. For this purpose, we utilize the following theorem by Kurtz.

*Definition - Density-dependent Family of Markov Chains [17, 20]:* Suppose we are given a set of transitions $L \subseteq Z^D$ (for a fixed, given $D$), and a collection of non-negative functions $\beta_{\bar{l}}$ for $\bar{l} \in L$ defined on a subset $E \subset R^D$. Then, [17, 20] define a *density-dependent family of Markov chains* $\bar{X}_N$ (where $N$ is some parameter) as a sequence $\{\bar{X}_N\}$

[3] The idea behind this condition is of course similar to that behind the well-known *Law of Mass Action*. Flipping is in fact a special case of One-time-sampling.

of jump Markov processes such that the state space of $\bar{X}_N$ is $E_N = \{E \cap \{\frac{h}{N} : h \in Z^D\}\}$, and the transition rates of the unscaled process $(N.\bar{X}_N)$ are given as:

$$q_{\bar{k},\bar{k}+\bar{l}} = N.\beta_{\bar{l}}(\bar{k}/N).$$

where, $\bar{k}, \bar{k}+\bar{l} \in N.E_N$.

Notice that $L, q_{.,.}$ are both unscaled, while $\bar{X}_N$, $E_N$, $\beta_{\bar{l}}(.)$ are all scaled.

**Kurtz's Theorem [17]:** Consider a density-dependent Markov process $\bar{X}_N$ as defined above. Next, suppose this process $\bar{X}_N$ satisfies the following two conditions (A) and (B):

**(A)** Suppose $\bar{X}_N$ satisfies the Lipschitz condition, i.e., for some constant $M$, and for all $\bar{z}, \bar{y}$ lying in $E_N$ (i.e., the range of $\bar{X}_N$), it is true that:

$$|F(\bar{z}) - F(\bar{y})| \leq M.|\bar{z} - \bar{y}| \quad \text{(A)}$$

where, $F(\bar{y}) = \Sigma_{\bar{l}} \bar{l}.\beta_{\bar{l}}(\bar{y})$.

**(B)** Now, define $\bar{X}(t)$ to be the appropriate limiting process (corresponding to $\bar{X}_N$), expressed as:

$$\bar{X}(t) = \bar{x}_0 + \int_0^t F(\bar{X}(u))du, t \geq 0$$

where $Lim_{N\to\infty} \bar{X}_N(0) = \bar{x}_0$.

Finally, consider the path $\{\bar{X}(u) : u \leq t\}$ for some fixed $t \geq 0$ and assume there exists a neighborhood $K$ around this path satisfying:

$$\Sigma_{\bar{l}} |\bar{l}|.sup_{\bar{y}\in K} \beta_{\bar{l}}(\bar{y}) < \infty \quad \text{(B)}$$

Now, if all the above conditions (A) and (B) are satisfied, then it is the case that:

$$Lim_{N\to\infty} sup_{u\leq t} |\bar{X}_N(u) - \bar{X}(u)| = 0.$$

In other words, when $N \to \infty$, the finite analysis (i.e., $\bar{X}_N(t)$) approaches the results of the infinite analysis (i.e., $\bar{X}(u)$). $\diamond$

**Theorem 1** (*The infinite analysis gives the same result as the finite analysis with $N \to \infty$.*)**:** The protocols derived from differential equations that are both completely partitionable and restricted polynomial: (I) are density-dependent Markov processes, and (II) satisfy Kurtz's theorem.

**Proof:** Here, we show that the system-wide run of the protocol defined in Section 2 (prior to Kurtz's theorem) fits into Kurtz's theorem. To do so, we: (I) first discuss the basics of the mapping to Kurtz's theorem, then (II) verify the conditions (B) and (A) from Kurtz's theorem (in that order).

**(I)** We start with the basics, and show why the protocol just described (Section 2) is indeed a density-dependent Markov process. In other words, we need to define $N$, $\bar{X}_N$, $E_N$, $D$, $L$, $\beta_{\bar{l}}(.)$ (and thus $F(.)$ in Kurtz's theorem).

First, notice $N$ is the finite number of processes in the distributed system. Next, $\bar{X}$ (defined at the start of Section 2) is the same as the (scaled) $\bar{X}_N$ defined in Kurtz's theorem above. Thus, with $E = [0,1]^m \subset R^m$, the state space of $\bar{X}_N$ is in $E_N = E \cap \{\frac{h}{N} : h \in Z^m\}$, therefore we have $D = m$ in Kurtz's theorem.

Next, we define $L$ and $\beta_{\bar{l}}(\bar{y})$. First, the set of transitions $(\bar{l} \in) L \subseteq Z^m$ is always a vector with all but two of the $m$ coordinates as zeroes; the remaining two coordinates are a -1 and a +1. This corresponds to a single process in the system changing its state. Thus $|\bar{l}| = \sqrt{2}$ always. Second, notice that each $\bar{l} \in L$ is an unscaled vector while each $\bar{y} \in E_N$ is a scaled vector (i.e., the former represents numbers of processes in different states, the latter represents these as fractions). Now, for a given $\bar{l}$ (and any $\bar{y} \in E_N$), we have that $\beta_{\bar{l}}(\bar{y})$ is non-zero if and only if that transition is allowed by a pair of terms in the differential equations, but is zero otherwise. More specifically, suppose $\bar{l}$ has its $i^{th}$ coordinate a -1 and its $j^{th}$ coordinate a +1. Then, suppose there is a negative term $-T_{\bar{l}}(\bar{y})$ occurring in $f_{x_i}$ and a corresponding matching positive term $+T_{\bar{l}}(\bar{y})$ occurring in $f_{x_j}$ (where $\bar{y} \in E_N$, and $x_i, x_j \in X$ are the $i^{th}, j^{th}$ coordinates respectively in $\bar{X}$). If no such terms exist for the given $\bar{l}$, then we have that $T_{\bar{l}}(\bar{y}) = 0$. In either case, we can concretely define:

$$\beta_{\bar{l}}(\bar{y}) = T_{\bar{l}}(\bar{y})$$

This is because of the following argument. Suppose a term $-T = -T_{\bar{l}}$ occurs in $f_{x_i}$, and that $f_{x_i}$ contains a total of $k_{x_i}$ negative polynomial terms. From the preceding discussion before Theorem 1, at any process $p$ that is currently in state $x_i$ (and the fraction of processes in state $x_i$ is also denoted as $x_i \in [0,1] \cap \{\frac{k}{N} : k \in Z^m\}$), each execution of the action for term $-T$ has a probability of $T/x_i$ of being successful, i.e., of causing a state change - this follows from the Flipping and One-time-sampling actions. The current number of processes in state $x_i$ is $N.x_i$. Thus, if the protocol period is $\tau$ time units, then the expected number of actions (for $-T : +T$) executed during an infinitesimal time interval of length $dt$, throughout the system is $\frac{dt}{\tau}.Nx_i.k_{x_i}.\frac{1}{k_{x_i}} = Nx_i.\frac{dt}{\tau}$. Due to term $T$, the number of processes in state $x_i$ changes during $dt$ by $d(N.x_i) = -Nx_i.\frac{dt}{\tau}.T/x_i = -N.T.\frac{dt}{\tau}$, or $dx_i = -\frac{T}{\tau}.dt$. This gives the effect of term $T$ on fraction $x$ as $dx_i = -\frac{T}{\tau}.dt$, which is $=-T.dt$ for a normalized protocol period of $\tau = 1$ time unit.

**(II)** Here, we verify the two conditions (B) and (A) of Kurtz's theorem (in that order for convenience).

For verifying condition (B), the above discussion gives us that (i) for any $\bar{l} \in L$, $|\bar{l}| = \sqrt{2}$, and (ii) for the corresponding term $T_{\bar{l}}(.)$, the positive constant in this polynomial term is denoted as $c_{T_{\bar{l}}}$. Thus :

$$\begin{aligned}\Sigma_{\bar{l}}|\bar{l}|.\beta_{\bar{l}}(\bar{y}) &\leq \Sigma_{\bar{l}}\sqrt{2}.|c_{T_{\bar{l}}}| \\ &\leq \sqrt{2}.\Sigma_{T\in\bar{f},T \text{ is a positive term in } \bar{f}}|c_T| \\ &< \infty\end{aligned}$$

Notice that the last sum above adds up the constants of *all positive terms* appearing in *all* component functions of $\bar{f}$. By definition, $\bar{f}$ contains a finite number of positive polynomial terms – therefore, the above sum is a finite quantity. This verifies condition (B) of Kurtz's theorem.

To verify condition (A) of Kurtz's theorem (i.e., the Lipschitz condition), we have:
$F(\bar{y}) = \Sigma_{\bar{l}}\bar{l}.\beta_{\bar{l}}(\bar{y}) = \Sigma_{T_{\bar{l}}\in\bar{f},T_{\bar{l}} \text{ is a positive term in } \bar{f}}(T_{\bar{l}}(\bar{y}).\bar{l})$

For the Lipschitz condition, given $\bar{z},\bar{y}\in E_N$, we need to calculate the quantity $|F(\bar{z})-F(\bar{y})|$. This quantity can be written as:

$$\begin{aligned}|F(\bar{z})-F(\bar{y})| &= |\Sigma_{\bar{l}}\bar{l}.\beta_{\bar{l}}(\bar{z}) - \Sigma_{\bar{l}}\bar{l}.\beta_{\bar{l}}(\bar{y})| \\ &= |\Sigma_{\bar{l}}\bar{l}.(\beta_{\bar{l}}(\bar{z}) - \beta_{\bar{l}}(\bar{y}))| \\ &\leq \Sigma_{\bar{l}}|\bar{l}|.|(\beta_{\bar{l}}(\bar{z}) - \beta_{\bar{l}}(\bar{y}))|\end{aligned}$$

The above quantity can be bounded as:
$\leq \Sigma_{T_{\bar{l}}\in\bar{f},T_{\bar{l}} \text{ is a positive term in } \bar{f}}(\sqrt{2}.|(T_{\bar{l}}(\bar{z})) - (T_{\bar{l}}(\bar{y}))|)$,

Now, Lemma 1.1 below shows that for each term $T_{\bar{l}}\in\bar{f}$, there exists a constant $M_{\bar{l}}$ such that
$|(T_{\bar{l}}(\bar{z}))-(T_{\bar{l}}(\bar{y}))| \leq M_{\bar{l}}.|\bar{z}-\bar{y}|$. This completes the theorem's proof since the constant $M$ required to fulfill condition (A) of Kurtz's theorem (the Lipschitz condition) can be written as the finite quantity:

$$M = \sqrt{2}.\Sigma_{T_{\bar{l}}\in\bar{f},T_{\bar{l}} \text{ is a positive term in } \bar{f}}(M_{\bar{l}}).$$

**Lemma 1.1:** For each positive (polynomial) term $T_{\bar{l}}\in\bar{f}$, and $\bar{z},\bar{y}\in E_N$, there is a constant $M_{\bar{l}}$ such that $|(T_{\bar{l}}(\bar{z}))-(T_{\bar{l}}(\bar{y}))| \leq M_{\bar{l}}.|\bar{z}-\bar{y}|$.
**Proof:** For each $\bar{x}\in E_N$, denote $\bar{x}=(x_1,x_2,\ldots,x_m)$. Let the term $T_{\bar{l}}$ be written as: $T_{\bar{l}}(\bar{x}) = +c.\Pi_{i=1}^m(x_i^{s_i})$, where each $s_i$ is a non-negative integer.

Now, define a series of terms $< v_0, v_1, v_2, \ldots v_{m-1}, v_m >$ such that for each $j=0$ to $m$, we have
$v_j = +c.\Pi_{i=1}^j(y_i^{s_i}).\Pi_{i=j+1}^m(z_i^{s_i})$. In other words, $v_j$ and $v_{j+1}$ differ only in the value of the $(j+1)^{th}$ coordinate in the product, and the series represents a coordinate-by-coordinate transition from $\bar{z}$ to $\bar{y}$. Thus, notice that $v_0 = T_{\bar{l}}(\bar{z})$, and $v_m = T_{\bar{l}}(\bar{y})$.

With this, we can write:

$$\begin{aligned}&|(T_{\bar{l}}(\bar{z})) - (T_{\bar{l}}(\bar{y}))| \\ &= |v_0 - v_m| \\ &\leq \Sigma_{j=0}^{m-1}(|v_j - v_{j+1}|) \\ &= \Sigma_{j=0}^{m-1}[c.\Pi_{i=1}^j(y_i^{s_i}).\Pi_{i=j+2}^m(z_i^{s_i}).|(z_{j+1}^{s_{j+1}} - y_{j+1}^{s_{j+1}})|] \\ &\leq \Sigma_{j=0}^{m-1}[c.|(z_{j+1}^{s_{j+1}} - y_{j+1}^{s_{j+1}})|] \\ &= \Sigma_{j=1}^{m}[c.|(z_j^{s_j} - y_j^{s_j})|] \\ &\leq \Sigma_{j=1}^{m}c.s_j|z_j - y_j|,\end{aligned}$$

where the last step used the inequality $|(z_j^{s_j} - y_j^{s_j})| \leq s_j|z_j - y_j|$ for $z_j, y_j\in[0,1]$ and non-negative integer $s_j$.

Denote, for each $j$, $\delta_j = |z_j - y_j|$. The above inequality can then be written as:

$$\begin{aligned}&|(T_{\bar{l}}(\bar{z})) - (T_{\bar{l}}(\bar{y}))| \\ &\leq \sqrt{(\Sigma_{j=1}^m c.s_j.\delta_j)^2} \\ &= c.\sqrt{\Sigma_{j=1}^m(\delta_j^2.s_j^2) + \Sigma_{j\neq r,j=1,r=1}^{j=m,r=m}(\delta_j.\delta_r.s_j.s_r)}\end{aligned}$$

Assume, without loss of generality, that we have $\delta_1 \geq \delta_2 \geq \ldots \delta_m$, i.e., arrange $s_j$'s in non-increasing order of $\delta_j$'s. Then, we can continue the above derivation as:

$$\begin{aligned}&|(T_{\bar{l}}(\bar{z})) - (T_{\bar{l}}(\bar{y}))| \\ &\leq c.\sqrt{\Sigma_{j=1}^m(\delta_j^2.s_j^2) + \Sigma_{j\neq r,j=1,r=1}^{j=m,r=m}(\delta_j.\delta_r.s_j.s_r)} \\ &= c.\sqrt{\Sigma_{j=1}^m(\delta_j^2.s_j^2 + \Sigma_{j+1\leq r\leq m}2\delta_j.\delta_r.s_j.s_r)} \\ &\leq c.\sqrt{\Sigma_{j=1}^m(\delta_j^2.(s_j^2 + \Sigma_{j+1\leq r\leq m}2s_j.s_r))} \\ &\leq c.\sqrt{\Sigma_{j=1}^m(\delta_j^2.(s_j^2 + \Sigma_{r\neq j,1\leq r\leq m}2s_j.s_r))} \\ &\leq c.S_{T_{\bar{l}}}.\sqrt{\Sigma_{j=1}^m\delta_j^2}\end{aligned}$$

where we have defined
$S_{T_{\bar{l}}} = max_{j:1\leq j\leq m}\{\sqrt{(s_j^2 + \Sigma_{r\neq j,1\leq r\leq m}2s_j.s_r)}\}$. Notice that the value of this $S_{T_{\bar{l}}}$ depends only on the exponent values in the term $T_{\bar{l}}$, and not on the values of $\delta_j$'s. Thus $S_{T_{\bar{l}}}$ is a constant.

Finally, as $\sqrt{\Sigma_{j=1}^m\delta_j^2} = |\bar{z}-\bar{y}|$, we continue and wrap up with $M_{\bar{l}} = c.S_{T_{\bar{l}}}$, since:

$$|(T_{\bar{l}}(\bar{z})) - (T_{\bar{l}}(\bar{y}))| \leq c.S_{T_{\bar{l}}}.|\bar{z}-\bar{y}|.$$

Hence proved (Lemma 1.1 and Theorem 1). □

**Corollary 1.A (*Infinite Analysis*):** Flipping and One-time Sampling are sufficient in mapping equation systems that are both completely partitionable and restricted polynomial, into distributed protocols that have equivalent behavior in infinite sized groups. Equivalence means that the equilibrium points w.r.t. $\bar{X}$ are preserved, while the timed trajectories of $\bar{X}$ are slower by a factor of $\tau$, where $\tau$ = the protocol period duration.

The above corollary follows from Theorem 1 and the fact that the limiting process $\bar{X}(t)$ (Kurtz's theorem) is the same as trajectory of the differential equation system itself.

Henceforth in the paper, we perform *only* the infinite analysis for the individual protocols derived from differential equations (endemic protocol, LV protocol). This means that *we are free to assume that $N\rightarrow\infty$ in each step of our future analyses of this class of protocols.* By the above theorem, this is the same as the finite analysis when $N\rightarrow\infty$.

**Corollary 1.B:** When different processes have different protocol periods, Theorem 1 and Corollary 1.A continue to hold, with $\tau$ standing for the inverse of the harmonic mean of all processes' protocol periods. This is easy to see because the term $\frac{dt}{\tau}.Nx_i.k_{x_i}.\frac{1}{k_{x_i}}$ appearing in Theorem 1's proof above is replaced by $\Sigma_{p\in ProcessGroup}(\frac{dt}{\tau_p}.x_i.k_{x_i}.\frac{1}{k_{x_i}})$, where $\tau_p$ is the protocol period effective at process $p$. Intuitively, this corollary is true because Theorem 1's proof depends on the *system-wide rate of actions*, not on individual protocol periods.

The next theorem bounds the error between the infinite and the finite analysis of our protocols.

**Corollary 1.C:** The error in system state, i.e., $\bar{X}$, as predicted by the infinite analysis (Theorem 1) and the finite analysis (Kurtz's theorem's $\bar{X}_N$) is $O(\frac{1}{\sqrt{N}})$.

**Proof:** Kurtz points out in [17] that:

$$X_N(t) = \bar{X}(t) + O(\bar{X}_N(0) - \bar{x}_0) + O(\frac{1}{\sqrt{N}}).$$

The error $|\bar{X}_N(t)-\bar{X}(t)| = |O(\bar{X}_N(0)-\bar{x}_0)+O(\frac{1}{\sqrt{N}})|$. The first term $O(\bar{X}_N(0)-\bar{x}_0)$ is an $m-$coordinate vector with each coordinate $\leq \frac{1}{N}$. Thus we have:

$$|O(\bar{X}_N(0) - \bar{x}_0)| \leq \frac{\sqrt{m}}{N}.$$

Hence, the error is bounded by $O(\frac{1}{\sqrt{N}})$ (since $m$ is a known constant). □

**Message Complexity:** The message complexity of the protocol is bounded by the size of the original equations. The expected number of sampling messages sent out by a process that is in state $x$, per protocol period, equals the sum of the number of occurrences of all variables in negative terms in $f_x(\bar{X})$, less the number of negative terms in $f_x(\bar{X})$.

**The Effect of Failures:** Message delivery losses and process failures modify the equation that is modeled by the protocol. If fractions $x \in X$ are fractions of alive processes in respective states, each original term $T$ for a one-time-sampling action takes on a multiplicative factor of $(\frac{1}{1-f})^{|T|-1}$, where $f$ is the known group-wide failure rate per connection attempt, and $|T|$ is the total number of variable occurrences in term $T$. In order to faithfully model the original equations for a system with a known $f$, it is enough to increase the heads probability of the flipped coin for one-time-sampling terms by a multiplicative factor of $(\frac{1}{1-f})^{|T|-1}$. If this causes the values of some constants $c_T$ in the equations to increase beyond 1, then the equation may have to be rewritten (using the techniques described in Section 7).

## 4 Translation Case Studies

We explore two case studies to demonstrate application of the design framework. This section describes protocols for probabilistic responsibility migration and probabilistic majority selection.

### 4.1 Case Study I: Responsibility Migration

We define a responsibility migration problem that aims at selecting a subgroup of processes from the group and migrating membership of this subgroup. The subgroup members could be used to share responsibility for a task, e.g., storing replicas of a given file, buffering multicasts received by the group, running consensus, etc. In this paper, we discuss the responsibility migration problem in the context of partial replication of files, and specifically, for a persistent distributed file system. We call this a model of *migratory replication.*

The responsibility migration problem can be formally stated as follows:

**Distributed Responsibility Migration Problem:**

At any point of time in a group $G$ of processes, each process is either a *responsible* process, or not. A non-responsible process can turn responsible only after contacting another process that is responsible. Further,

`Safety:` The number of responsible processes in the system never becomes zero.

`Liveness:` A process that is currently responsible will eventually become non-responsible.

`Fairness (optional):` Over a long time of running, each process in the system bears responsibility for an equal fraction of time.

For a persistent distributed file system application (e.g., a concept similar to the eternity storage service introduced by Anderson in [3]), where each object is a file, each file has a responsibility migration protocol running on its behalf, at each process. At any time, the responsible processes for a file are the only ones storing replicas of the file. `Safety` ensures that a file, once inserted, never disappears from the system. `Liveness` ensures that a responsible process deletes the file eventually[4].

Partial data replication has been studied for many years, e.g., in databases [11,13], email and file systems [16,24,25], and more recently, in peer to peer systems [7]. Gray et al. [13] argue against the dangers of large-scale replication and warn that maintaining consistency among a large number of replicas might counter the goal of scalability. Replication has two tasks: (a) replica location (placement), and (b) replica management. Replica location decides, for a given object, "how many" and "where" replicas of the object are located. Replica management deals with how replicas are accessed and updated in a consistent manner, e.g., active and passive replica management are well known techniques [8].

[4] This does not preclude the process from becoming responsible again at some later point of time.

In this paper, we focus only on replica location - these strategies can be combined with appropriate replica management strategies, although such issues are the subject of a future article. Most existing solutions to replica location [23, 24] locate replicas *statically* and *reactively*, i.e., the subset of hosts selected to hold replicas of a given object does not change unless a special event happens, e.g., one of the hosts could crash. This approach has the following three disadvantages: (1) it can be expensive in systems containing millions of hosts, where a large fraction have short lifetimes (O(several minutes)) and rejoin multiple times (6.4 times/day as reported in the Overnet system [5]) [5]; (2) From a security stand-point, static and reactive strategies allow an attacker to easily locate and attack all the individual replicas of an object. A malicious host could track the current replicas for a given object, and then subject each of them simultaneously to a kind of directed attack (e.g., a DOS attack on each host, sustained until the host crashed, would suffice) in such a way that all copies of the object are destroyed; (3) Static and reactive strategies attempt to satisfy safety but neither liveness nor fairness.

Dynamic and migratory replication strategies avoid the above drawbacks, and provide other interesting properties. These strategies proactively move replicas of the object among different hosts in the system. Thus, an attacker finds it difficult to locate all the replica hosts, and even if the attacker does locate them, it has only a short window before the subgroup membership changes. Further, the availability of the object is not affected by either short host availability periods or massive failures in the system. Contrary to intuition, endemic protocols can offer scalable and reliable behavior w.r.t. liveness, safety, and fairness, while incurring only a constant message overhead at each process in the group.

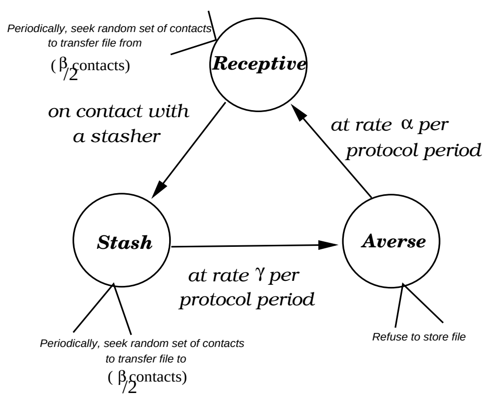
The following section discusses the endemic protocol and its analysis. In order to avoid diluting the focus of our paper, we concentrate on the scalability and efficiency for only the migration of replicas. Details of the replica lookup and replica update, and a detailed security analysis, are left to a later article on the persistent distributed file system.

The definition of the distributed responsibility migration problem leads to the following impossibility result:

**Observation 1** *(Impossibility of achieving* `Safety`*)***:** No responsibility migration protocol can achieve `Safety`. This is because there exists a run where at some instant of time, all responsible processes crash simultaneously.

This leads us to define a probabilistic version of the problem:

[5] One could restrict replicas to be stored only on hosts with at least a threshold availability [6], but this does not address the issue (2).



Periodically, seek random set of contacts to transfer file from
( β/2 contacts)
Receptive
on contact with a stasher
at rate α per protocol period
Stash
Averse
at rate γ per protocol period
Periodically, seek random set of contacts to transfer file to
( β/2 contacts)
Refuse to store file

**Fig. 1** *The Push-Pull Variant of the Endemic Protocol.*

**Probabilistic Distributed Responsibility Migration Problem:** This is the same as the distributed responsibility migration problem, except that the `Safety` property is redefined as:
`Probabilistic Safety:` The probability of the number of responsible processes in the system becoming zero, is close to 0.

### *4.1.1 A Simple Solution, and its Drawback*

Consider a replica migration protocol where a process storing an object replica hands it off to another process after a while and immediately deletes the object. A crash-stop failure of the former process before the transfer effectively destroys an object replica. Over time, the number of replicas of a given object will then go down to zero (unless there is a periodic refresh).

### *4.1.2 Endemic Protocol*

We present an *endemic protocol* for probabilistic distributed responsibility migration, designed from differential equations for endemic infectious diseases in a closed human population. This natural analogy is appropriate because the desired behavior of the protocol bears resemblance to the persistent survival of common cold in human populations and folklores in human society, both for centuries. Let $x, y, z$ be the fractions of susceptible, infected, and immune individuals, respectively. We consider the following endemic equations:

$$\begin{aligned}\dot{x} &= -\beta xy + \alpha z \\ \dot{y} &= \beta xy - \gamma y \\ \dot{z} &= \gamma y - \alpha z\end{aligned} \tag{1}$$

Here, $\beta, \gamma, \alpha$ respectively stand for the rates of infected-susceptible contact, recovery, and susceptibility. $\alpha$ and $\gamma$ each lie in the interval $(0, 1]$, and $\beta > \gamma$.

This equation system is restricted polynomial and completely partitionable. Using the framework of Section 3, the state machine derived has three states - $x$ (susceptible or *receptive*), $y$ (infected or *stash*), $z$ (immune or *averse*). *A process is responsible if and only if it is in the stash state.* The actions are as follows.

Assume that $\beta \in (0, 1]$. State actions can be defined to be executed periodically, as discussed earlier in the methodology. (i) ($\gamma y$ term) A process $p$ in the stash state tosses a coin with a biased heads probability $\gamma$ - if the coin falls heads, process $p$ changes its state to averse. This transition is accompanied by a deletion of the object replica at $p$. (ii) ($\alpha z$ term) A process $p$ in the averse state tosses a coin with heads probability $\alpha$, and changes state to receptive if this coin falls heads up. (iii) ($\beta xy$ term) A process $p$ in the receptive state chooses 1 target uniformly at random, and if the target is in the stash state, and a locally flipped coin (with heads probability $\beta$) falls heads simultaneously, process $p$ changes its state to stash (after an object transfer).

While the above protocol is of the *pull* type (receptive processes attempt to pull responsibility to themselves), it is also possible to design a *push-pull* variant of the above protocol. This variant also allows responsible processes to push responsibility to other processes. *The endemic protocol analysis of Section 4.1.3 applies to both pull and push-pull variants.* The experimental results of Section 5.2 study only the push-pull variant.

In this new push-pull variant, $\beta$ is an even positive integer[6]. Figure 1 illustrates this variant, where the action (iii) is replaced by two actions (iii)* and (iv)*. The variant protocol uses a small, fixed parameter $b = \beta/2$. (iii)* ($\beta xy$ term) A process $p$ in the receptive state chooses $b$ targets uniformly at random, and if any of these targets is in the stash state, process $p$ changes its state to stash (after an object transfer). (iv)* ($\beta xy$ term) A process $p$ in the stash state chooses $b$ processes uniformly at random; any of the selected target processes that is receptive immediately transitions to the stash state (after an object transfer).

This variant protocol still follows the equation system (1). The contact rate $\beta$ can be recalculated as follows. The probability that a given stash process and another given receptive process will make contact is $(1 - (1 - \frac{b}{N})^2) \simeq \frac{2b}{N}$. The change in the value of $x$ due to such contacts is $\frac{Nx.Ny.\frac{2b}{N}}{N} = \beta xy$, giving $\beta = 2b$.

A few observations are in order before we analyze the endemic protocol. First, both the above protocols (pull and push-pull) satisfy equation system (1). Further, notice that the push-pull variant allows us to account for aborted file transfers and process failures with an extra multiplicative factor for $b$. Finally, the third averse state is required as it probabilistically ensures that there is a time interval after a process deletes a replica (i.e., stops being in stash state) when it will not store the file again.

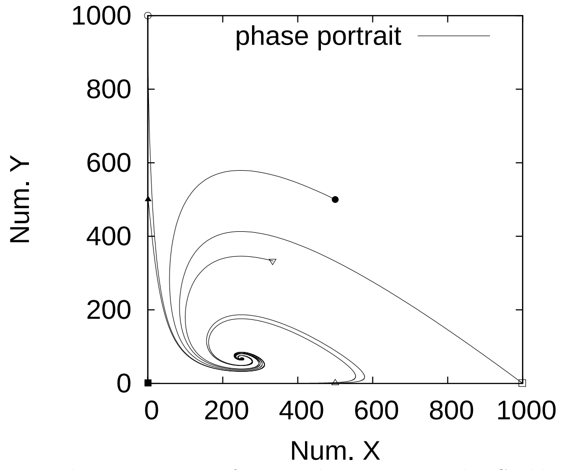


**Fig. 2** *Phase Portrait of an Endemic Protocol - Stable Spiral: The plot shows the phase portrait obtained by simultaneously plotting the numbers of susceptibles and infectives $(X, Y) = (Nx, Ny)$ over time, in a system with $N = 1000$ processes. This plot shows the system started at these initial points - (X,Y,Z)=blank square (999,1,0), dark square (0,1,999), blank circle (0,1000,0), dark circle (500,500,0), blank triangle (500,1,499), dark triangle (1,500,499), and blank inverted triangle (333,333,334). For the parameter setting $N = 1000, \alpha = 0.01, \beta = 4, \gamma = 1.0$, the non-trivial equilibrium point above is a stable spiral.*

### *4.1.3 Endemic Protocol Analysis*

We analyze the endemic protocol[7], with the goal of demonstrating how the framework-generated protocols are to be studied. First, we note that the endemic protocol of Figure 1 satisfies `Liveness` if $\gamma > 0$. Since the protocol is symmetric across processes, it also satisfies `Fairness`. The `Probabilistic Safety` of the protocol is discussed at the end of this section.

Equilibrium in the endemic equations occur when $(\dot{x}, \dot{y}, \dot{z}) = 0$. Substituting in the above equations gives us two equilibria:

$$(x_\infty, y_\infty, z_\infty) = (1, 0, 0), \text{ and}$$
$$(x_\infty, y_\infty, z_\infty) = (\gamma/\beta, \frac{1 - \gamma/\beta}{1 + \gamma/\alpha}, \frac{1 - \gamma/\beta}{1 + \alpha/\gamma}) \qquad (2)$$

We call these respectively as the *first equilibrium point* and the *second equilibrium point*. The first equilibrium results in all copies of the object disappearing; all processes eventually turn receptive. The second equilibrium is more desirable, since it guarantees safety.

Now, we study stability and convergence properties of these equilibrium points. Since an explicit solution for

[6] Although it will be evident from our discussion that $\beta$ can be generalized to be any positive quantity.

[7] To the best of our knowledge, equation (1) has not been analyzed elsewhere in the literature.

time-dependent variation of the variable values is difficult to obtain, we resort to an indirect analysis using *perturbation analysis*.

**Is the First Equilibrium Self-Correcting?** Let us assume a small deviation from the first equilibrium point. Let us start the system in the state:

$$(x_0, y_0, z_0) = (1 - u - v, u, v)$$

where $|u|, |v| \ll 1$. Substituting into the second equation in system (1), we get:

$\dot{u} = \beta u - \gamma u$

which has the solution $u = u_0.e^{(\beta-\gamma)t}$,where $u_0$ is the initial value of $u$. Thus, if $\alpha, \beta, \gamma > 0$ and $1 > \frac{\gamma}{\beta}$, the first equilibrium point $(x, y, z) = (1, 0, 0)$ is a *saddle point*. This means that it is only *partly* stable; as long as $y$ remains 0, the system converges back to the equilibrium. However, inclusion of even a single stasher $(y > 0)$ will cause the system to diverge away from this first equilibrium point (and towards the second equilibrium point, as we describe below).

Finally, notice that if $1 < \frac{\gamma}{\beta}$ and $\alpha, \beta, \gamma > 0$, then given that $x, y, z > 0$, then the equilibrium point $(1, 0, 0)$ is stable. We always choose the parameters $\frac{\gamma}{\beta} < 1$ so that this situation never arises.

**Is the Second Equilibrium Self-Correcting?** Let us assume a small deviation from the second equilibrium point. Let us start the system in the state:

$$(x_0, y_0, z_0) = (x_\infty(1 + u), y_\infty(1 + v), z_\infty(1 + w))$$

where $|u|, |v|, |w| \ll 1$, and
$(x_\infty, y_\infty, z_\infty) = (\gamma/\beta, \frac{1-\gamma/\beta}{1+\gamma/\alpha}, \frac{1-\gamma/\beta}{1+\alpha/\gamma})$.

**Lemma 1:** The only equilibrium point in equation system (4) is $(t, u) = (0, 0)$.
**Proof:** Substituting the above into equations (1) and simplifying, we get:

$$\frac{\gamma}{\beta}\dot{u} = -\beta\frac{\gamma}{\beta}\frac{1-\gamma/\beta}{1+\gamma/\alpha}(1 + u)(1 + v) + \alpha\frac{1-\gamma/\beta}{1+\alpha/\gamma}(1 + w)$$

$$\text{or } \dot{u} = \frac{\beta-\gamma}{1+\gamma/\alpha}(w - u - v) \quad (3a)$$

where the $uv$ term is ignored as $|u|, |v| \ll 1$. Similarly,

$$\dot{v} = \gamma u \quad (3b)$$

$$\dot{w} = \alpha(v - w) \quad (3c)$$

where we have assumed that $y_\infty = \frac{1-\gamma/\beta}{1+\gamma/\alpha} \neq 0$. Eliminating $w$ and $v$ from equations (3a-c), we get the following differential equation for $u$:

$$\frac{1}{\sigma\alpha}\ddot{u} + \dot{u}(\frac{1}{\sigma} + \frac{1}{\alpha}) + u(1 + \frac{\gamma}{\alpha}) = 0.$$

where $\sigma = \frac{\beta-\gamma}{1+\gamma/\alpha}$. This can be written as a system of two linear differential equations by introducing a new variable $t = \dot{u}$. In matrix form, the equations are

$$\dot{T} = A \cdot T, \quad (4)$$

where

$$T = \begin{bmatrix} t \\ u \end{bmatrix}, \qquad A = \begin{bmatrix} -(\sigma + \alpha) & -\sigma(\gamma + \alpha) \\ 1 & 0 \end{bmatrix}$$

Finally, $\dot{u} = 0 \Rightarrow t = 0$, and then $\dot{t} = 0 \Rightarrow u = 0$. Hence proved. □

From the discussion in the above proof, the equation system (3(a,b,c)) has only one equilibrium point $(u, v, w) = (0, 0, 0)$. This is easy to see by substitution of the equilibrium value $u = 0$ into equations (3(b,c)).

**Theorem 2** *(The endemic protocol is always self-stabilizing around the second equilibrium point)***:** For the system of differential equations (1), the equilibrium point $(x_\infty, y_\infty, z_\infty) = (\gamma/\beta, \frac{1-\gamma/\beta}{1+\gamma/\alpha}, \frac{1-\gamma/\beta}{1+\alpha/\gamma})$ is *always stable*, if both $\alpha, \beta, \gamma > 0$ and $1 > \frac{\gamma}{\beta}$.
**Proof:** The stability of the equilibrium point $(u, v, w) = (0, 0, 0)$ depends on the trace and determinant of matrix $A$. If the trace is negative and the determinant is positive, the equilibrium point is stable. However, if the trace and determinant were both positive, the equilibrium would be unstable. Finally, if the determinant is negative, the equilibrium is a saddle point (i.e., some trajectories in the vicinity converge to it, and the rest diverge) [26].

We can calculate:
$\tau$ = trace$(A)$ = sum of leading diagonal elements in $A = -(\sigma + \alpha)$, and

$$\Delta = \det(A) = \sigma(\gamma + \alpha) \quad (5)$$

By our choice of parameters, $\sigma > 0$ and $\alpha, \gamma > 0$. Therefore, we *always have that* $\tau < 0, \Delta > 0$. This means that the solution of $u = 0$ for equations (4) is *always stable*.

From equations (3(b,c)) and Lemma 1, we can say that any perturbations ($(u, v, w)$ in the values of $(x, y, z)$) around the second non-trivial equilibrium point will die out, and the system is self-stabilizing around the second equilibrium point. □

The above discussion shows that when $\alpha, \beta, \gamma > 0$ and $1 > \frac{\gamma}{\beta}$, starting with a positive number of stashers $(y > 0)$ will always drive the system to the second stable equilibrium point, while starting with no stashers eventually leads to the first equilibrium point. Figure 2 illustrates this behavior through a *phase portrait*, which depicts the simultaneous variation of three variables $x, y, z$ from several initial points. The trajectories around the second equilibrium point for this setting of parameters is a stable spiral.

Perturbation analysis can also be used to calculate *how quickly* the distributed group converges towards a stable equilibrium point, when it is started in a close-by operating point. This convergence speed is defined as follows.

**Convergence Complexity (Definition):** The convergence complexity for a stable equilibrium point $P$ of a protocol state machine with $m$ states is an $m-$sized vector of functions in variable $t$ that describe the variation of fractions of processes in the $m$ respective states, if the system is started in some neighborhood of point $P$.

For endemics, the nature of the trajectories around the second stable equilibrium point, as well as the time taken by the protocol for convergence, both depend on the eigenvalues and eigenvectors of the matrix $A$. The eigenvalues of $A$ are $\lambda_1 = \frac{\tau+\sqrt{\tau^2-4\varDelta}}{2}$, and $\lambda_2 = \frac{\tau-\sqrt{\tau^2-4\varDelta}}{2}$.

From equations (5), we can calculate and simplify:

$\tau^2 - 4\varDelta = (\frac{\beta-\gamma}{1+\gamma/\alpha} - \alpha)^2 - 4\frac{\beta-\gamma}{1+\gamma/\alpha}\gamma$

The time-dependent variation of $u$ is a function of the initial value $u_0$ and $\dot{u_0}$ at time $t = 0$. Three cases arise, and we apply the results from [26] to analyze in the following manner (only the most significant terms are accounted for in this analysis, and comparatively insignificant terms are ignored):

1. $\tau^2 - 4\varDelta < 0$ (eigenvalues distinct and complex): The variation of the displacement $u$ in the number of susceptibles $x$, as a function of time, can be calculated as:

$u = u_0 e^{-\frac{t(\sigma+\alpha)}{2}} cos(t\sqrt{\sigma\gamma - \frac{(\sigma-\alpha)^2}{4}})$

where $u_0$ is the initial value of $u$. Notice that with time, $u$ decreases exponentially fast to 0. The cosine term causes a (damped) oscillation in the value of $u$. This leads to a *stable spiral*, which means that the convergence takes the form of a damped oscillation.

2. $\tau^2 - 4\varDelta > 0$ (eigenvalues distinct and real): The variation of $u$ as a function of time is given by

$u = \frac{\dot{u_0}-\lambda_2 u_0}{\lambda_1-\lambda_2} e^{t\lambda_1} + \frac{\dot{u_0}-\lambda_1 u_0}{\lambda_2-\lambda_1} e^{t\lambda_2}$

where $u_0$ and $\dot{u_0}$ are the initial values of $u$ and $\dot{u}$ respectively. Notice that since both eigenvalues will be negative (when $\tau < 0$), this equation shows exponentially fast convergence.

3. $\tau^2 - 4\varDelta = 0$ (eigenvalues equal and real): The variation of $u$ as a function of time can be calculated as $u = u_0 e^{-t\frac{\sigma+\alpha}{2}}$, where $u_0$ is the initial value of $u$.

Thus, the system converges exponentially quickly from the neighborhood of the second equilibrium point.

`Probabilistic Safety` **- Longevity of Object Replicas:** In any computer system, there is always a non-zero probability of all replicas of an object disappearing completely. While a full-scale analysis of the multi-dimensional Markov chain generated by our endemic protocol is infeasible, we present a back of the envelope calculation of the likelihood of all file replicas disappearing in an endemic protocol that is at equilibrium. Each of the $y_\infty.N$ stashers creates new stashers at a rate $\beta x_\infty 1 = \gamma$. Each stasher is also turning averse at the same rate $\gamma$, thus it is equally likely to die before creating any new stashers. The likelihood that none of the $y_\infty.N$ stashers will create any new replicas before turning averse is $= (\frac{1}{2})^{y_\infty \cdot N}$.

If protocol parameters $\alpha, \gamma, b$ are chosen so that $y_\infty = \frac{c.log_2 N}{N}$, the probability of all stashers dying before creating new ones is $\frac{1}{N^c}$. If the protocol period duration is 6 minutes long, $N = 1024$ and 50 replicas gives us an expected object longevity of $1.28 \times 10^{10}$ years. With $N = 2^{20}$ and 100 replicas, we get an object lifetime of $1.45 \times 10^{25}$ years.

### 4.2 Case Study II: Majority Selection Problem

Implementations of distributed systems often need to *select* between two choices of voting processes in the system based on the *majority* of voting processes in the system. Examples include deciding between two differing replica types with the same filename in a distributed digital library application such as LOCKSS [18], for distributed replica management, etc. We state the problem as:

**Majority Selection Problem:** *In a distributed group of N processes, each process initially proposes either 0 or 1. The Majority Selection protocol ensures that all processes agree on which of the two values (0 or 1) has a majority of non-faulty proposers.*

This problem is related to the Consensus problem, where each of $N$ processes initially proposes a value (0 or 1), but eventually sets its output variable exactly once, and to the same value as other non-faulty processes. Article [12] shows that the Consensus problem is impossible to solve in an asynchronous system.

*Observation:* Majority selection is impossible to solve in an asynchronous system, since a solution could be used to implement consensus.

Below, we give a specification for probabilistic majority selection. The probabilistic majority selection runs forever, and it maintains a *running* decision variable with possible values 0 or 1 or $b$ (undecided).

**Probabilistic Majority Selection Problem:** *In a distributed group of N processes, each process initially proposes either 0 or 1. The Majority Selection protocol ensures that decision variables at all non-crashed processes eventually agree, and with high probability (depending on the difference in the numbers of 0's and 1's), this is the same as the initial majority value.*

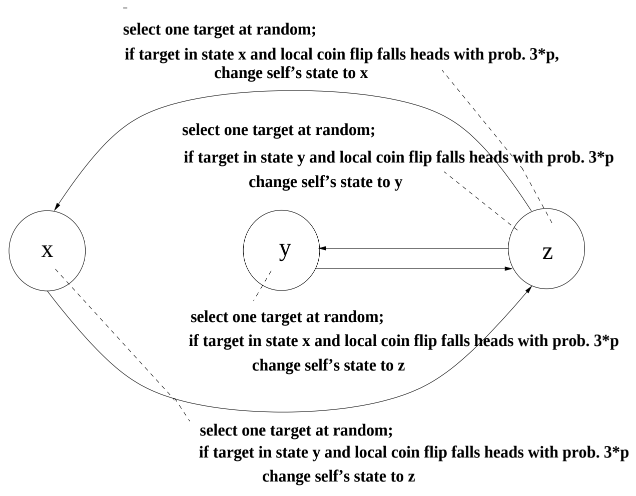

select one target at random;
if target in state x and local coin flip falls heads with prob. 3*p,
change self's state to x
select one target at random;
if target in state y and local coin flip falls heads with prob. 3*p
change self's state to y
x
y
z
select one target at random;
if target in state x and local coin flip falls heads with prob. 3*p
change self's state to z
select one target at random;
if target in state y and local coin flip falls heads with prob. 3*p
change self's state to z

**Fig. 3** *The LV protocol for Probabilistic Majority Selection.*

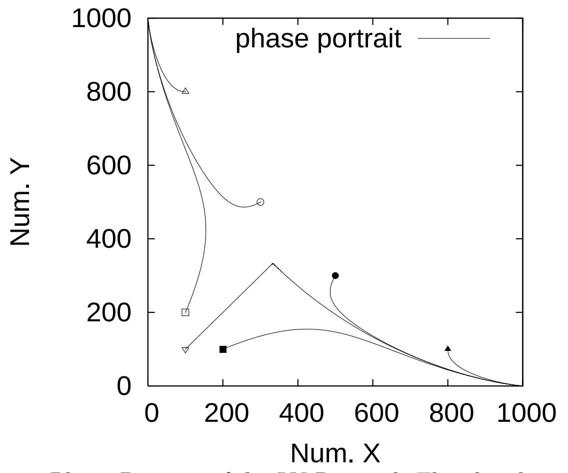


**Fig. 4** *Phase Portrait of the LV Protocol: The plot shows the phase portrait obtained by simultaneously plotting $(X,Y) = (Nx,Ny)$ over time. This plot shows a system of $N = 1000$ processes started at these initial points - $(X,Y,Z)$=blank square (100,200,700), dark square (200,100,700), blank circle (300,500,200), dark circle (500,300,200), blank triangle (100,800,100), dark triangle (800,100,100), and blank inverted triangle (100,100,800). All initial points with $x < y$ converge to (0,1000,0), and all initial points with $x > y$ converge to (1000,0,0). Initial points with $x = y$ move towards (333.3,333.3,333.3), but may then move arbitrarily towards one of the two stable points.*

Probabilistic majority selection is useful for applications that need to reach consensus eventually, but where the decision value is allowed to be set multiple times, e.g., in LOCKSS [18][8].

### *4.2.1 The LV Protocol*

The *Principle of Competitive Exclusion* says that "Two species competing for the same limited resource typically cannot coexist" [21,26]. The classic *Lotka-Volterra (LV) model* presents a mathematical modeling of the above phenomenon[9]. In the model, a variable $x$ denotes the number of rabbits, and variable $y$ denotes the number of sheep, in a given ecosystem. The LV model encapsulates the competition in a system of differential equations. We use the following new equations because they are appropriate to our purpose[10]:

$$\dot{x} = 3.x.(1 - x - 2y)$$
$$\dot{y} = 3.y.(1 - y - 2x) \qquad (6)$$

To make this equation system complete, we add a new variable $z = 1 - x - y$, and the equation

$$\dot{z} = -\dot{x} - \dot{y}$$

A glance at equation system (6) shows a $+3x$ term on the r.h.s. of $\dot{x}$, indicating that the equations may not be partitionable. However, let us rewrite these equations as:

$$\dot{x} = +3xz - 3xy$$
$$\dot{y} = +3yz - 3xy$$
$$\dot{z} = -3xz - 3yz + 3xy + 3xy$$

This equation is now completely partitionable. To convert this equation system into one that is also restricted polynomial, we multiply *all terms* on all the right hand sides above with a *normalizing parameter* $p \in (0, 1/3]$, thus ensuring that the constants in all terms are $\leq 1$.

$$\dot{x} = +3pxz - 3pxy$$
$$\dot{y} = +3pyz - 3pxy$$
$$\dot{z} = -3pxz - 3pyz + 3pxy + 3pxy \qquad (7)$$

Here, $p$ is a fixed constant, $p \in (0, 1/3]$. The above procedure has effectively *rewritten* equation system (6) into a form that is both completely partitionable and restricted polynomial. Section 7 discusses equation rewriting techniques in detail. We can now apply Flipping and One-time-sampling on equation system (7) to generate the state machine shown in Figure 3.

### *4.2.2 LV Protocol Analysis*

Assuming the fractions of processes in different states to be $x, y, z$ respectively, the equilibrium points for equation (6) (equivalent to equation (7)) are obtained by setting $(\dot{x}, \dot{y}) = (0, 0)$. They are $(x_\infty, y_\infty) = (0, 0)$, $(0, 1)$, $(1, 0)$, and $(1/3, 1/3)$.

[8] This does not violate the consensus impossibility proof [12] since in such probabilistic majority voting, it is difficult to decide *when* to finalize the decision variable.

[9] Notice that this is *not* the same as Lotka-Volterra's predator-prey model.

[10] These exact equations have not been presented elsewhere in literature.

Second, we classify these equilibrium points as either stable or unstable. To do this, we use the Linearization technique described by Strogatz in [26]. Linearization effectively approximates the original equation system (that is non-linear) with a linear equation system. The stability of equilibrium points in the linearized system (and hence the original equation system) depend on eigenvalues of the Jacobian.

The Jacobian for the equation system 1 is:

$$J_{(x,y)} = \begin{bmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial y} \\ \frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial y} \end{bmatrix} = p. \begin{bmatrix} 3-6y-6x & -6x \\ -6y & 3-6x-6y \end{bmatrix}$$

The value of the Jacobian at the equilibrium points are:

$J_{(0,0)} = p. \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$, with a sole eigenvalue of $3p$.

$J_{(0,1)} = p. \begin{bmatrix} -3 & 0 \\ -6 & -3 \end{bmatrix}$, with a sole eigenvalue of $-3p$.

$J_{(1,0)} = p. \begin{bmatrix} -3 & -6 \\ 0 & -3 \end{bmatrix}$, with a sole eigenvalue of $-3p$.

$J_{(1/3,1/3)} = p. \begin{bmatrix} -1 & -2 \\ -2 & -1 \end{bmatrix}$,

with eigenvalues $p$ and $-3p$.

$\underline{(0,0):}$ $(0,0)$ has a single eigenvalue that is positive. This means that $(0,0)$ is an *unstable* equilibrium point. In other words, all points of operation $(x,y)$ that are in the vicinity of $(0,0)$ diverge away, as time progresses. In the context of the protocol of Figure 3, this is interpreted as "The distributed process group moves away from indecision", a desirable property.

$\underline{(1/3,1/3):}$ The equilibrium point $(1/3,1/3)$ has two eigenvalues - one is positive and one is negative. The product of the eigenvalues is negative and their sum is positive. From [26], these conditions mean that $(1/3,1/3)$ is a *saddle point*. In the context of the distributed protocol for majority selection (state machine - Figure 3), this is interpreted as "*Some* operations points in the vicinity of $(1/3,1/3)$ converge towards it, and the rest diverge away from it."

The details of this behavior depend on the eigenvectors of the Jacobian: for the negative eigenvalue $-3p$, the eigenvector is $(1,1)$ and the eigenvector for $+p$ is $(1,-1)$. From known results [26], all points lying on the line through the equilibrium point $P$, with a slope given by the eigenvector for the negative eigenvalue (line called the "stable manifold"), converge to $P$. Further, all points on the line through $P$ that has a slope equal to the eigenvector for the positive eigenvalue (line called the "unstable manifold") eventually diverge from $P$. Points that are on neither manifold eventually diverge away from $P$.

Thus, all points lying on the line $x = y$ (eigenvector for negative eigenvalue) through $(1/3,1/3)$ converge towards $(1/3,1/3)$, and all points on $x+y = 2/3$ (eigenvector for positive eigenvalue) diverge away from the point. Further, all other points in the plane eventually converge towards the unstable manifold, i.e., $x + y = 2/3$.

$\underline{(0,1) \text{ and } (1,0):}$ Each of $(0,1)$ and $(1,0)$ have a single eigenvalue that is negative. Unfortunately, this means that it is possible that the conclusions of the Linearization analysis that we have done so far may not hold [26] (such equilibrium points are called "degenerate nodes"). Therefore, we need to resort to *perturbation analysis* to find out the nature of these last two equilibrium points.

**Lemma 2:** Each of the equilibrium points $(0,1)$ and $(1,0)$ is stable, i.e., for each point, there is a neighborhood around it from where the system converges exponentially quickly back to the point.

**Proof:** We first present the proof for $(0,1)$. Let $u$ and $v$ be small displacements (perturbations) along the $x$ and $y$ directions respectively from the point $(0,1)$, i.e., $(x,y) = (u, 1-v)$, where $|v|, |u| \ll 1$ and $v \geq u$. From equation system (6), we have

$\dot{x} = \dot{u} = 3pu(1-2+2v-u) \simeq -3pu$

$\dot{y} = -\dot{v} = 3p(1-v)(1-2u-1+v) \simeq 3pv - 6pu$

where the final approximations assume that $|2v-u| \ll 3$ and $|v| \ll 1$.

These equations have a closed form solution:

$u = u_0 e^{-3pt}$

$v = (6u_0pt + v_0)e^{-3pt}$

where $u_0, v_0$ are the initial values of $u, v$ at time $t = 0$. It is easy to see that as $t \to \infty$, both $u, v \to 0$. Hence, $(0,1)$ is a stable equilibrium point.

By symmetry of equation system (6), it is easy to show similarly that $(1,0)$ is also a stable equilibrium point. □

This means both these points are stable equilibria. In other words, all points of operation in the vicinity of either $(1,0)$ or $(0,1)$ converge towards the respective point, as time progresses. In the context of the protocol for majority selection, this is interpreted as: "The distributed process group converges towards the nearest point of consensus." These points represent desired final states for our process group.

**Theorem 3** *(Correctness of LV protocol)***:** $(x,y) = (0,1)$ and $(1,0)$ are stable points, while $(0,0)$ is an unstable point and $(1/3,1/3)$ is a saddle point. Further:
1. If the system starts from any initial point that is to the right of $x = y$, i.e., has $x_0 > y_0 \geq 0, x_0 + y_0 \leq 1$, it will eventually converge towards $(1,0)$.
2. If the system starts from any initial point that is to the left of $x = y$, i.e., has $x_0 < y_0 \geq 0, x_0 + y_0 \leq 1$, it will eventually converge towards $(0,1)$.
3. If the system starts from any initial point that lies on $x = y$, i.e., has $x_0 = y_0 \geq 0, x_0+y_0 \leq 1$, it will eventually converge towards $(1/3,1/3)$.

Thus, an infinite-sized process group eventually self-stabilizes.

For finite-sized groups, if condition (3) above theorem is true, it takes only one process to be perturbed to cause the system to deviate from $x = y$. Thus, a finite-sized group eventually converges towards one of the two points $(x, y) = (0, 1)$ or $(1, 0)$. □

Figure 4 depicts the phase portrait of the system for several initial operating points.

In a finite-sized group of processes, the $x = y = 1/3$ point would be unsustainable, because randomization will eventually push the system into either $x < y$ or $x > y$. Section 5 studies LV protocol in finite (and large) sized groups.

**Self-Stabilization, Open Groups:** Since the LV protocol runs forever, by Theorem 3 above, it is also *self-stabilizing*. Even in an open group, it proactively continues to converge back to an equilibrium point in spite of dynamic changes (e.g., new processes) that may perturb the operation point.

**Convergence Complexity:** For the LV state machine of Figure 3, we calculated the convergence complexity of stable point $(0, 1)$ as $(x(t), y(t)) = (u_0 e^{-3pt}, 1 - (6u_0pt + v_0)e^{-3pt})$, in the neighborhood $|u|, |v| \ll 1$. A symmetric result holds for $(1, 0)$. The LV protocol thus has an exponential convergence complexity in the vicinity of each of the stable equilibrium points.

## 5 Experimental Studies

In order to understand the implications of the endemic protocol and the LV protocol in practical scenarios with finite but large-sized process groups, we present experimental results. These results are based on C implementations of the LV protocol, and endemic protocol designed for the application of a persistent distributed file system.

The protocols were tested in a simulated environment, with multiple instances that are running synchronously (protocol periods are fixed, and start times are the same across processes) over a simulated network, all on a single machine (1.7 Ghz Intel Celeron CPU, 256 MB RAM, WinXP Pro). We are able to report numbers in 100,000-host groups. The Mersenne Twister pseudorandom generator is used for random number generation. In all the plots, the "Time" variable on the horizontal axis is normalized in time units (3 minute intervals).

In our discrete simulator, each process processes messages and sends messages out once every time unit. The default protocol period is set to an expected value of 2 time units. This means that $T_{next}$ is chosen in the following manner: a process in state $x$ (for any given $x$), at the top of each time unit, flips a local coin with heads probability of $\frac{0.5}{k_x}$. If this coin turns up heads, the process executes a random action from among the set of $k_x$ actions defined for state $x$, otherwise the process waits until the next time unit. If an action is chosen and causes a transition, the process executes no more actions during the rest of this time unit, instead waiting until the next time unit starts.

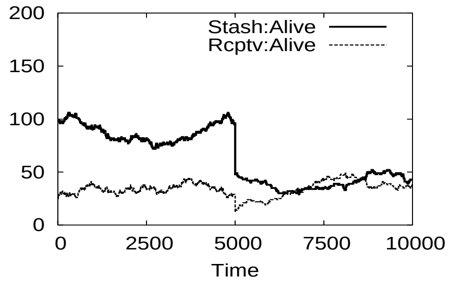


**Fig. 5** *Endemic Protocol - Massive Failures: In a setting with* $N = 100,000$*,* $b = 2, \alpha = 10^{-6}, \gamma = 10^{-3}$*, the number of stashers and replicas in a 100, 0000 host system is very small. Massive failure of 50% of the hosts at time* $t = 5000$ *causes the system to stabilize quickly. The value oscillations are more sluggish after the failures since many sampling messages go unanswered.*

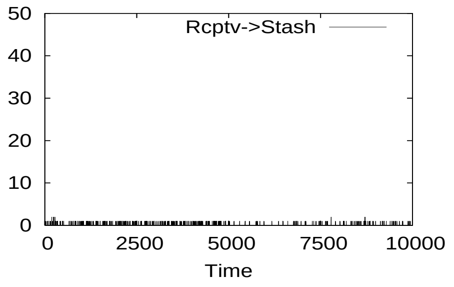


**Fig. 6** *Endemic Protocol - File Flux Rate = number of file transfers per protocol period. This is for the same experiment as in Figure 5. Occurrence of a massive number of failures at time* $t = 5000$ *does not affect file flux rate drastically.*

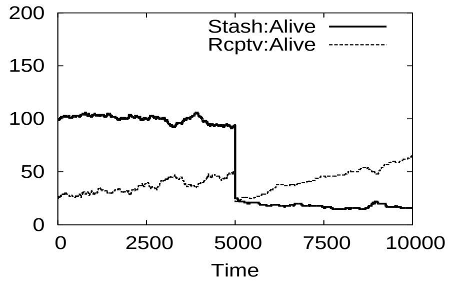


**Fig. 7** *Endemic Protocol - Massive Failures: Same setting as Figure 5, but with a protocol period of 4 time units. The convergence is more sluggish, but is less prone to perturbations.*

### 5.1 Endemic Replication - Experiments

This is an implementation of the protocol from Section 4.1. The time unit is fixed at 3 minutes at each host (so the protocol period is 6 minutes). Values of $b, \gamma, \alpha$ vary for different experiments. All numbers are for a single file only.

**Overhead, Fault-tolerance and Self-Stabilization:** A 100,000 host system initially at equilibrium is subject to failure of a random 50% of the hosts. $b = 2, \alpha = 10^{-6}, \gamma = 10^{-3}$ are used. After the failures at time $t =$

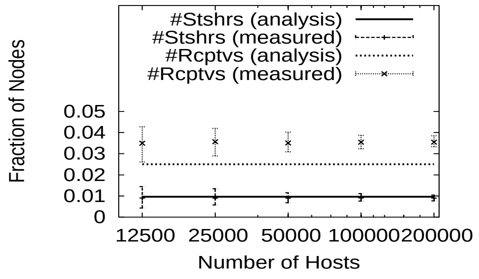


**Fig. 8** *Endemic Protocol - Accuracy of Continuous Time Analysis: The experimental results from the simulation match well with those predicted by the mathematical analysis. With* $b = 2, \gamma = 0.1, \alpha = 0.001$*, the above plot shows the median number (over a time interval 2000 periods long) and the analytically expected numbers of both receptives and stashers. The minimum and maximum measured values over this interval are also shown.*

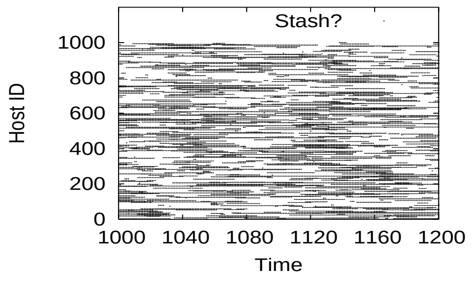


**Fig. 9** *Endemic Protocol - Replica Untraceability and Load Balancing: For a population of* $N = 1000$ *hosts and* $b = 2, \gamma = 0.1, \alpha = 0.01$*, the plot shows which hosts are stashers at the end of every protocol period. The absence of significant horizontal lines indicates good load balancing (no hosts store a replica for very long). The absence of any correlations w.r.t. time or hostid in the figure shows the difficulty faced by an attacker who seeks to destroy all replicas of the file.*

5000, the number of stashers (Figure 5) and the number of averse (not shown) each drop by a factor of about two. The number of stashers stabilizes very quickly. The number of receptives does not change since after the failure, 50% of the contacts initiated by any alive host are directed at a crashed host, and are hence fruitless (this reduces the effective value of $b$ by 2, thus doubling the original equilibrium fraction $x_\infty = \frac{\gamma}{\beta}$). Figure 6 shows that the communication overhead and network traffic stays low too, and there is no wild variation in the number of file transfers in spite of the massive failure.

The oscillation in the numbers of stashers and receptives, occurring due to random choice of contacts in the endemic protocol, eventually die out. The oscillations are more sluggish after the massive failure as fewer contacts are fruitful.

Figure 7 shows that increasing the protocol period (by choosing to execute an action in state $x$ with probability $\frac{0.25}{k_x}$ per time unit) does not change the equilibrium points, but merely has the effect of making convergence slower. Slower convergence has the advantage of reducing the amplitude of perturbations in comparison to when a shorter protocol period were used (Figure 5).

Notice that for both protocol period settings above, the behavior at times $t > 5000$ is also characteristic of a *heterogeneous* setting, where half the hosts are chronically averse to storing the file or even perhaps to running the protocol.

**Reality Check:** In a 100,000-host system, each host would be storing a given file for 0.1% of the time (since number of stashers $\simeq 100$), effectively once every 100,000 hundred hours, or 4166 days. The file would be stored for an average duration of 100 hours (a little over four days) each time (expected time for a stasher to turn averse is $\frac{1}{\gamma} = 1000$ periods). If one assumes an average file size of 88.2 KB as in [23], a 6 minute protocol period implies a bandwidth utilization of $3.92 \times 10^{-3}$ bps per file per host.

**Analysis vs. Real Behavior:** Our analysis in Section 4.1.3 assumed an infinitely large group, and the use of memoryless distributions to decide when the next action will be executed. In comparison, our simulations have finite but large groups and use fixed protocol period lengths. Figure 8 compares the former (analysis) with the latter (measured). The measured and the analyzed fractions of stashers tally very closely, and the measured fraction of receptives stays constant and close to the fraction of receptives expected from the analysis. Further, the system-wide fractions of stashers and receptives are constant, and the standard deviation stays low. This shows that the considered group sizes are large enough for the analysis to apply.

**Untraceability of Replicas:** Figure 9 shows the distribution of stashers over time in a population of $N = 1000$ hosts, with protocol parameters $b = 2, \gamma = 0.1, \alpha = 0.01$. The stable number of stashers is 88.636. The distribution of the stashers (dark dots) does not appear to have correlations either across hostid's or for extended periods of time at each hostid. Unless the attacker knows about the location of all the replicas of a file at a given point of time *and* destroys all these copies before any new stashers are created (with the current parameters, one stasher is created on average every 40.6 seconds), the file will survive inside the system. The plot also illustrates load balancing.

**Trace-based simulations - Effect of Host Churn:** Real protocol deployments are also required to tolerate dynamic stresses such as host churn, i.e., rapid arrival and departure. We model the worst effect of host failure - a host loses all stored file replicas when it fails or departs. When it rejoins the system, it is in the receptive state towards all files but does not participate in any startup file transfers. Figures 10 and 11 show the behavior of endemic replication in a 2000 host system under host churn, injected in the form of availability traces taken from the

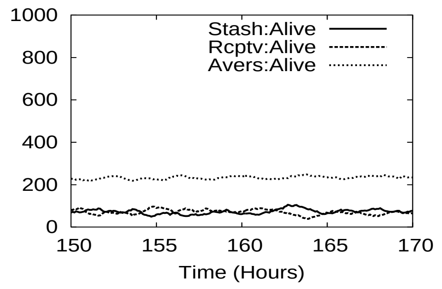


**Fig. 10** *Endemic Protocol - Effect of Host Churn A: Churn traces were injected continuously into the system, with the protocol period set to 6 minutes at each host.* $N = 2000, b = 32, \gamma = 0.1, \alpha = 0.005$. *This plot shows that the numbers of stashers, averse and receptives remain stable in the system, and the number of stashers stays low.*

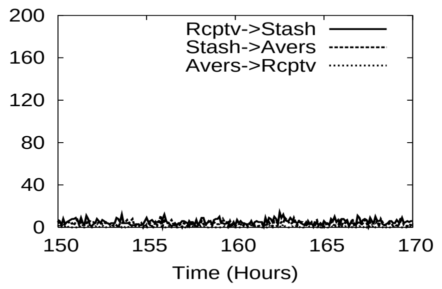


**Fig. 11** *Endemic Protocol - Effect of Host Churn B: For the experiment in Figure 10, this plot shows the number of state transitions, per protocol period, across hosts.*

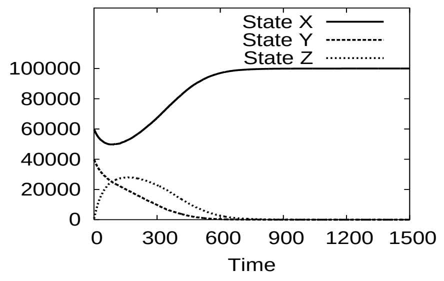


**Fig. 12** *LV Protocol - Variation of Populations:* In a 100,000 process group, starting with 60,000 processes in state $x$ and 40,000 processes in state $y$.

Overnet system [5]. The original availability traces were taken once every hour; for our experiments, these were spread out over each hour. The protocol period was set to 6 minutes, and the values of $\alpha = 0.005, \gamma = 0.1, b = 32$ used. Hourly churn rates in the traces were typically between 10% to 25% of the system size. The endemic protocol is seen to maintain a stable number of stashers and low file flux rate, and is thus churn-resistant.

## 5.2 LV Protocol - Experiments

We present excerpts from simulation results of the push-pull variant of the LV protocol implementation in Section 4.2. The normalizing parameter is set as $p = 0.01$. The protocol period is chosen to be an expected 2 time

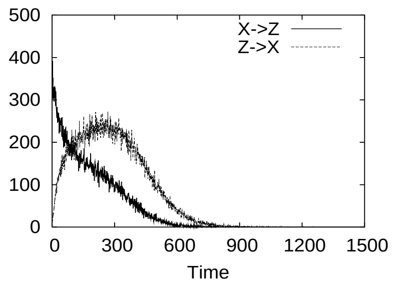


**Fig. 13** *LV Protocol - Rate of State Transitions:* For Figure 12, the number of state transitions per protocol period.

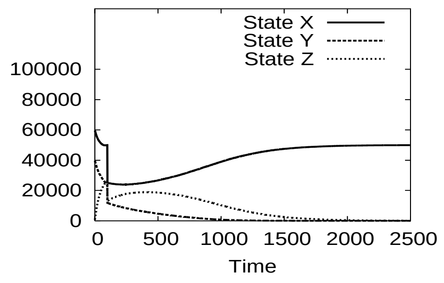


**Fig. 14** *LV Protocol - Effect of Massive Failures: For the same initial conditions as in Figure 12, half the processes (selected at random) are caused to fail at time* $t = 100$. *Convergence still occurs, although a little late.*

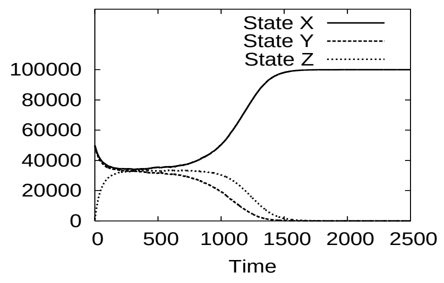


**Fig. 15** *LV Protocol - Effect of Finite Group Size: An initial difference of 800 processes in a 100,000 process system converges towards the "correct" equilibrium.*

units, as described right before Section 5.1.

**Convergence:** Figure 12 shows that a 100,000 process group started with 60,000 processes in state $x$ and 40,000 processes in state $y$ converges quickly towards X=100,000. The convergence time is 1782 time units; with a protocol period of (say) 1 s, this means convergence in about 15 minutes. Recall that the message overhead on each process, due to the protocol, is always constant and independent of group size. Figure 13 shows that the corresponding number of transitions (per protocol period) is low.

**Effect of Massive Failures:** For the same initial conditions as in Figure 12, half the processes, selected at random, are caused to crash at time $t = 100$. Figure 14 shows that the system converges. The convergence time is delayed, but occurs eventually.

**Resolution:** The normalizing protocol parameter $p$ ($p \in [0,1]$) can be used to (1) dampen the perturbations in the system; and (2) control the resolution of the protocol, i.e., its sensitivity to small initial differences between the numbers of processes in states $x$ and $y$. A smaller value for $p$ means smaller perturbations. We do not attempt to mathematically analyze the effect of varying $p$, but instead intuitively explain the advantages and disadvantages of small $p$ values.

Recall that each term in the equation system is multiplied with $p$, thus affecting the heads probability of the coin flips for these terms. Higher values of $p$ have lower resolution, and a greater likelihood of converging towards the "wrong" equilibrium (i.e., towards state $x$ although initially $X < Y$, and vice versa). This can happen due to the randomness of coin flips in a finite-sized group. A smaller value of $p$ reduces this likelihood. For $p = 0.01$, an initial difference (between the numbers of processes in states $x$ and $y$) of 800 for a 100, 000 process system converges in the correct direction - see Figure 15. A smaller initial difference has a lower probability of converging to the correct equilibrium, and may require an even smaller value of $p$. Finally, the downside of using small $p$ values is that the convergence is substantially slowed down – this is intuitively equivalent to the system being "careful" about making a decision.

## 6 Mapping Equation Systems that are Completely Partitionable but not Restricted Polynomial

### 6.1 Equations with Polynomial Terms

Flipping and one-time-sampling may not suffice to translate equation systems that are polynomials but not restricted polynomials, even if they are completely partitionable. For example, an equation $\dot{x} = f_x(\bar{X})$ might contain a $+c$ term, or a term $-\Pi^{y \in X}(y^{i_{y,f_x,T}})$ with $i_{x,f_x,T} = 0$.

Flipping and One-time-sampling do not work for mapping such terms. We informally describe a third type of action called Tokenizing that can be used to *approximately* map such equations. "Approximate" means that equivalence to the original equations cannot be guaranteed, but would nevertheless hold under many circumstances.

**Tokenizing:** In a completely partitionable and polynomial equation system, consider a term $-T = -c_T.\Pi^{y \in X}(y^{i_{y,f_x,T}})$ that occurs on the r.h.s. of the equation for variable $\dot{x}$, where $c_T \in [0,1]$. Further, $i_{x,f_x,T} = 0$. (Otherwise, Flipping or One-time-sampling can be used.) A *Tokenizing* action can be combined with Flipping or One-time-sampling to map this term into an action. This is done as follows - choose a variable $w \in X$ such that $i_{w,f_x,T} \neq 0$. If no such variable exists, term $T$ is a constant $c$ - in that case, rewrite $+T$ as $+c.(\Sigma^{v \in X} v)$ and $-T$ as $-c.(\Sigma^{v \in X} v)$, and repeat these steps.

Now, create an appropriate Flipping or One-time-sampling action for processes in state $w$. This action is created in a similar manner as in Section 3. However, modify the action as follows: when all conditions are satisfied by the Flipping or One-time-sampling action (coin turns up heads, or three conditions are true, respectively), then the process in question does not transition (since it is in state $w$), but instead creates a *token* specifying the action. It immediately passes the token on to another process that it knows is in state $x$. On receipt of the token, the process in state $x$ transitions to the state of the variable that has the corresponding $+T$ term. If no processes in the system are in the state $x$, the token is dropped.

**Limitations of Tokenizing:** There are several reasons why the proposed Tokenizing is only approximate. The Tokenizing action may, in some cases, not lead to state-to-state transition rates that are equivalent to those in the original differential equations. We describe some of these situations below.

After a token is generated at a process $p$, the ability to pass this on to another process that is in the needed target state for the token, requires continuous maintenance of knowledge (at $p$) of which states other processes are in (or at least one process for each possible state). This could be achieved by using a scalable membership protocol such as SWIM [10], yet may involve additional overhead to maintain this information, especially if transitions in the group are occurring very rapidly.

An alternative is to associate each token with an integer-valued time-to-live (TTL) and pass it along a random walk among the processes until a process in the target state is encountered. A finite TTL has a probability of expiring before an appropriate target is met, while an infinite TTL may cause state transitions to occur too late. In either case, the behavior of the protocol may be different from the original equation system. However, the new behavior can still be analyzed by modifying the original equation system with multiplicative terms in tokenized actions that account for the likelihood of the generated token being effective.

### 6.2 Equations with Arbitrary Terms

A completely partitionable differential equation that is not polynomial may contain such terms as $T = \frac{x}{y}$, $e^{z-2y}$. A generic Sampling technique is described below for *approximately* mapping such terms to actions. "Approximate" means that equivalence to the original equations cannot be guaranteed. This sampling technique generates tokens at a rate of $T$.

**Sampling:** For a term $-T = -g(\bar{X})$ that appears in $f_x(\bar{X})$ and is not mappable using one of Flipping, One-time-sampling or Tokenizing, the following action is used.

When a process is in state $x$, it samples a number (say $l$) of other processes uniformly at random for their current states. From this sample, it estimates the fraction $z$ of processes in state $z$ present throughout the system, for each variable $z$ that appears in term $T$. From this, and an appropriate Flipping action, the process decides probabilistically whether to generate a token for the transition to the target state. The token is passed to a process known to be currently in state $x$. If no such process in state $x$ exists, the token is dropped.

The limitation of the Sampling action is that $l$ needs to be very large (e.g., $O(N)$) in order to ensure that the differential equation is faithfully mirrored in the protocol. Otherwise, there may be errors in sampling the fractions of processes in different states. However, a large value for $l$ makes the protocol unscalable.

## 7 Equation Rewriting: Mapping Equations that are not Completely Partitionable

We describe techniques that translate an equation system into a completely partitionable form.

**Rewriting an equation into a Complete form:** Any equation system $\dot{\bar{X}} = f(\bar{X})$ can be rewritten into an equivalent equation system that is complete, by (i) introducing a new variable $z \notin X$ so that $z = 1 - \Sigma^{x \in X} x$, and (ii) considering instead the modified equation system, for variables in $X' = X \cup \{z\}$, consisting of the union of the original equation system $\dot{\bar{X}} = f(\bar{X})$ and the equation $\dot{z} = \Sigma^{x \in X}(-f_x(\bar{X}))$.
*E.g.,* In Section 4.2, the original LV equation system (6) was rewritten into a completely partitionable form (7).

**Normalizing:** A completely partitionable system of differential equations of the form $\dot{\bar{X}} = f(\bar{X})$ may have either variables $x \notin [0,1]$, or $\Sigma^{x \in X} x \neq 1$, or multiplicative constants $c_T$ (for terms $T$) $\notin [0,1]$. Such an equation system needs to be normalized. First, normalize each variable $x \in X$ by *multiplying* it with a constant quantity $M = \Sigma^{x \in X} x$, and then rewriting the equation system. $M$ is a constant since the sum of fractions of processes in different states does not change in a complete system, but it may appear as a constant in the modified equations. In order to further normalize the system, *each* term in *each* $f_x$ may need to be multiplied by a common constant $p \in [0,1)$ so that the constants in each of the terms now lie in $[0,1]$.
*E.g.,* The epidemic equation system (0) in Section 1 were derived from the original system $\dot{x} = -\frac{1}{N}xy, \dot{y} = \frac{1}{N}xy$, where $x$ and $y$ are the *numbers* of susceptibles and infectives. The normalizing constant is $N = x + y$, the (fixed) group size.

**Mapping Differential equations of higher Orders:** *Some* equation systems of higher order (all derivatives w.r.t. $t$ only) can be rewritten, e.g., an equation in a single variable, that has arbitrary order $k \geq 1$ (highest derivative of a variable) and degree 1 (power of highest derivative), can be rewritten as an equivalent equation system by introducing new variables for higher order terms. If the original equation has $m$ variables, the equation system of equations may contain up to $mk$ extra variables.
*E.g.,* $\ddot{x} + \dot{x} = x$ can be rewritten as the completely partitionable system: $\dot{x} = u; \dot{u} = x - u; \dot{z} = -x$.

Finally, *some* equations of arbitrary degree and order can be rewritten. For example, $\dddot{x}^2 + 5\ddot{x} + \dot{x} - 3 = 0$ can be solved for $\dddot{x}$, and then the above techniques applied.

## 8 DiffGen: Automatic Generation of Distributed Protocol Code

The methodology presented so far in the paper is automatable. We have developed an end-user toolkit called **DiffGen** that allows a protocol designer to input a set of differential equations (in a language similar to that of the popular *Mathematica* software [30]), and spews out ready-to-deploy C code that can be run over a standard UDP sockets implementation. We expect the DiffGen toolkit to be useful to distributed protocol designers who wish to *systematically* convert natural phenomena into novel distributed protocols. Some such natural phenomena were already discussed in Section 4; others may include [9,27].

**Related Work:** Previous toolkits for code generation from high level specifications consist mainly of *Model Driven Architectures* (MDA). Major aspects of MDA include UML-based modeling, transformation between an application's overall design models and the models that are specific to the underlying computing architecture like EJB, and generation of code in a specific language. Currently, more than forty such tools are available [2]. Some of these tools like ArcStyler from Interactive Objects Software, GmbH, XDE from IBM Rational, and OptimalJ from Compuware Corp include the UML modeler, the transformation engine and the code generator. Others like Codagen Technology Corp's Architect and Telelogic's Tau take input from existing UML modeling tools such as Rational Rose or No Magic Inc's MagicDraw UML and then generate code from them. Stella [31] allows modeling of existing systems, and subsequent simulation. In all these cases, the problem to be solved is itself not specified, but instead a model for the solution is inputted to the design toolkit. DiffGen has a similar approach; unlike DiffGen however, the above toolkits do not allow a simple and systematic way of generating code for the many distributed protocols based on analogies from natural phenomena.

**DiffGen Internals:** DiffGen consists of three subcomponents - the *Parser*, the *Rewriter*, and the *Code Generator*. The Parser takes in as input differential equations specified in a Mathematica-like input format [30] and converts this into a common internal representation called *diffIR*. This representation is then passed to the Rewriter which attempts to rewrite the equations into a completely partitionable form using the techniques in Section 7. If it succeeds (unless the user terminates the process), the transformed equations are passed to the Code Generator which uses the methodology specified in Sections 3 and 6 to generate C code for the protocol.

The presence of the diffIR representation allows other input formats for differential equations (by extending the Parser). Further rewriting techniques can be incorporated into the Rewriter, and extensions to the translation methodologies presented in this paper would be included in the Code Generator.

The generated protocol code contains clean interfaces and well-defined parameters that allow the designer to modify it systematically if necessary. For example, the random choice of targets for the sampling action could be replaced with a more topologically-aware one. In general, it allows the designer to port the protocol to different underlying communication mechanisms, incorporate it into legacy distributed systems, and compose it with other protocols.

DiffGen allows the user to provide application specific functions for implementing the communication mechanism between nodes (send and receive) and for choosing a random node from the group. Currently, DiffGen provides two default implementations - (i) allowing the protocol code to be run on an internal simulator, and (ii) allowing the protocol code to be run on a UDP sockets implementation, enabling immediate deployment of the code. The simulator allows users to specify environment parameters such as the number of processes that crash-stop or crash-recover, clock drifts, percentage of messages dropped, and the number of nodes to simulate on. The simulator is based on a discrete event simulation model and assumes a FIFO message queue.

The toolkit comes packaged in a GUI with functionality to edit, view and create new files, generate code and analyze the outputted protocol. Further information on the DiffGen toolkit, along with a free download, is publicly available at
`http://www-faculty.cs.uiuc.edu/~ indy/rsrch/diffGen`.

## 9 Conclusion

We have proposed a systematic framework that translates systems of differential equations into practical distributed protocols. The basic equation systems considered are of the form $\frac{d\bar{X}}{dt} = \bar{f}(\bar{X})$, where $\bar{X}$ is a vector of the finite-sized variable set $X$, $\bar{f}$ is an $|X|$-sized vector of functions each in $|X|$ variables, and $t$ is the time variable. If the terms on the right hand sides of these equations are polynomial and can be matched into positive-negative pairs, the protocol generated by the framework has provably *equivalent* behavior to the equation in an infinite-sized distributed group. Our mathematical analysis and experiments confirm this property also holds in finite, large-scale groups.

The framework creates a state machine with one state per variable in the original equations, and probabilistic actions derived from terms in the equations. *Equivalence* means the time-based variation of variable values in the equations is the same as the variation of fractions of nodes in corresponding states in the protocol. Stable equilibrium points in the equation lead to self-stabilizing behavior in the protocol, simplicity of terms leads to scalable communication, and stochastic properties lead to fault-tolerance, resistance to churn, and untraceability.

The framework also includes translation techniques for more general classes of differential equations than the above. However, there are limitations for these general classes, e.g., equivalence to equations with arbitrary terms may be unscalable. Techniques to rewrite differential equations into a format that allows translation by the framework were also presented.

We have used the framework to design new probabilistic protocols for the problems of (i) endemic responsibility migration (e.g., for a model of dynamic and migratory replication), and (ii) majority selection. Both analysis, and experiments with periodic versions of these protocols, reveal that these protocols are probabilistically scalable and reliable. They have possible uses in peer to peer systems, the Grid, and other large-scale distributed systems.

This framework and the DiffGen toolkit provide a valuable and powerful resource to a designer of protocols. They enable systematic conversion of ideas and results from other scientific disciplines into practical distributed system designs.

**Acknowledgements**

The authors thank the reviewers and the editor, as well as Yookyung Jo and Steve Bond, for their advice and insightful comments. This paper is an extended version of our previous conference publication in ACM PODC 2004 [14]

## References

1. D. Achlioptas, "Lower bounds for random 3-SAT via differential equations", *Theor. Comp. Sci.*, 265:1-2, 2001, 159-185.
2. J. Ambrosio, "Tools for the code generation", ADTmag.com, http://www.adtmag.com/article.asp?id=7850, 2004.
3. R. J. Anderson, "The Eternity Service", *Proc. 1st Intl. Conf. Th. and Appl. of Cryptology*, Prague, 1996.

4. N. T. J. Bailey, *Epidemic Theory of Infectious Diseases and its Applications*, Hafner Press, Second Edition, 1975.
5. R. Bhagwan, S. Savage, G.M. Voelker, “Understanding availability”, *Proc. Intnl. Symp. Peer to Peer Syst.*, Feb 2003, 135-140.
6. R. Bhagwan et al., “Total Recall: system support for automated availability management”, *Proc. Usenix Netw. Sys. Design and Implementation*, 2004, 337-250.
7. E. Cohen, S. Shenker, “Replication strategies in unstructured peer-to-peer networks”, *Proc. ACM SIGCOMM*, 2002, 177-190.
8. G. Colouris, J. Dollimore, T. Kindberg, *Distributed Systems: Concepts and Design*, Addison-Wesley, Third Edition, 2001.
9. A. Corradi, L. Leonardi, and F. Zambonelli, “Diffusive Load-Balancing Policies for Dynamic Applications”, IEEE Concurrency, 7:1, Jan 1999, 22-31.
10. A. Das, I. Gupta, A. Motivala, “SWIM: Scalable Weakly-consistent Infection-style process group Membership protocol”, *Proc. IEEE Dependable Syst. and Nets.*, 2002, 303-312.
11. A. Demers et al., “Epidemic algorithms for replicated database maintenance”, *Proc. ACM Princ. of Dist. Comp.*, 1987, 1-12.
12. M. J. Fischer, N. A. Lynch, M. Patterson, “Impossibility of distributed consensus with one faulty process”, *Journ. ACM*, 32:2, Apr 1985, 374-382.
13. J. Gray et al., “The dangers of replication and a solution”, *Proc. ACM SIGMOD*, 1996, 173-182.
14. I. Gupta, “On the Design of Distributed Protocols from Differnetial Equoations”, *Proc. 23$^{rd}$ ACM Symp. Princ. Dist. Comp.*, 2004, pp. 216-225.
15. E. Kreyszig, “Advanced Engineering Mathematics, 8th edition, John Wiley and Sons Publishing Company, NJ, 1998.
16. J. Kubiatowicz et al., “Oceanstore: an architecture for globalscale persistent store”. *Proc. ACM Annl. Symp. Prog. Langs. and Oper. Syst.*, Nov 2000, 190-201.
17. T.G. Kurtz, *Approximation of population processes*, Regional Conference Series in Applied Mathematics, SIAM, 1981, ISBN 0-89871-169-X.
18. P. Maniatis et al., “Preserving peer replicas by rate-limited sampled voting”, *Proc. ACM Symp. Oper. Syst. Principles*, Oct 2003, 44-59.
19. M. Merritt, G. Taubenfeld, “Computing with infinitely many processes”, *Proc. Dist. Comp.*, Springer LNCS 1914, 2000, 164-178.
20. M. Mitzenmacher, “The power of two choices in randomized load balancing”, *IEEE Trans. Parallel and Dist. Syst.*, 12:10, Oct 2001, 1094-1104.
21. R.J. Putman and S.D. Wratten, *Principles of Ecology*, Croom Helm (London), 1984.
22. M.O. Rabin, “Randomized Byzantine generals”, *Proc. Foundations of Comp. Sci.*, Nov 1983, 403-409.
23. A. Rowstron and P. Druschel, “Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility”, *Proc. ACM Symp. Oper. Syst. Principles*, Oct 2001, 188-201.
24. Y. Saito et al., “Taming aggressive replication in the Pangaea file system”, *ACM SIGOPS Oper. Sys. Rev.*, 2001, 15-30.
25. M. D. Schroeder, A. D. Birrell, R. M. Needham, “Experience with Grapevine: the growth of a distributed system”, *ACM Trans. on Comp. Syst.*, 2:1, Feb 1984, 3-23.
26. S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*, Perseus Book Group, First Edition, 2001.
27. E. W. Weisstein, “Heat Conduction Equation”, MathWorld–A Wolfram Web Resource, 2004, http://mathworld.wolfram.com/HeatConductionEquation.html.
28. N.C. Wormald, “Differential equations for random processes and random graphs”, *Annals of Appl. Prob.*, 5:4, 1995, 1217-1235.
29. S.-H. Wu et al., “Composition and behaviors of probabilistic I/O automata”, *Theor. Comp. Sci.*, 176:1-2, Apr 1997, 1-38.
30. Mathematica toolkit, Wolfram Inc., http://www.wolfram.com/products/mathematica/index.html
31. Stella toolkit, ISEE Systems, http://www.iseesystems.com

**Indranil Gupta** is Assistant Professor of Computer Science in the University of Illinois at Urbana-Champaign. He completed his PhD from Cornell University in 2004, and received the NSF CAREER award in 2005. Indranil heads the Distributed Protocols Research Group (DPRG) at UIUC, which studies various distributed systems such as peer-to-peer systems, Grid computing, sensor networks, and design methodologies that span the breadth of these areas. He has served on the program committees for several conferences, including those organized by ACM, IEEE and Usenix.

**Mahvesh Nagda** is a Software Engineer at Wolverine Asset Management in Chicago, IL. She graduated with Honors with a dual Bachelors-Masters of Science in Computer Science from the University of Illinois at Urbana Champaign in 2004.

**Christo Frank Devaraj** is currently a Software Design Engineer at Microsoft Corporation, Redmond. He received his MS degree in Computer Science from the University of Illinois at Urbana-Champaign in 2004. He received his Bachelor of Technology degree in Computer Science from the Indian Institute of Technology, Madras in 2002.