Emergent Algorithms — A New Method for Enhancing Survivability in Unbounded Systems

David A. Fisher & Howard F. Lipson
CERT® Coordination Center
Software Engineering Institute
dffisher@cert.org   hfl@cert.org

Abstract

Traditional security approaches are not sufficient to deal with the protection and survivability of highly distributed information systems operating in unbounded networks. This paper discusses the need for and importance of survivability, defines unbounded network, and examines the characteristics that differentiate survivability from other software quality attributes and nonfunctional properties of systems. It introduces emergent algorithms as an approach to problem solving in unbounded networks and suggests a methodology for their development. Emergent algorithms are philosophically and methodologically different from traditional approaches. The characteristics of emergent algorithms are examined. A strategy for the development of high performance solutions using emergent algorithms is illustrated in outline form for a problem in Internet routing.

Keywords: agents, distributed systems, electronic commerce, emergent algorithms, emergent computation, emergent properties, infrastructure protection, mission requirements, networks, network security, nonfunctional properties, nonfunctional requirements, protocols, software quality attributes, security, survivability, survivable systems, unbounded networks, unbounded systems

1. The Need for Survivable Systems

Key sectors of our society are becoming increasingly dependent upon highly distributed information systems that operate in unbounded networks, such as the Internet. This includes electronic commerce and many of the information system components of our nation’s critical infrastructures [PCCIP 97]. As a result, the risks to our society associated with successful intrusions into these information systems have never been greater.

Unfortunately, traditional security approaches are not sufficient to protect large-scale, highly distributed systems that operate in unbounded networks. In the realm of networked information systems, the natural escalation of threats versus countermeasures has demonstrated time and again that practical systems are always vulnerable to attack. Evolution of attack techniques is particularly well supported when systems include commercial off-the-shelf (COTS) software and public domain components, because widespread knowledge about system internals is readily available to the community at large. In the presence of these threats, there is a compelling need to augment traditional security approaches with techniques that allow systems to survive, limit damage, recover, and operate robustly in the presence of attacks that cannot be completely repelled [ISW 97, ISW 98]. A fundamental assumption underlying survivability is that no individual component of a system can be made immune to all attacks, accidents, and design errors.

We define survivability as the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents [EFL 97]. We use the term “system” in the broadest possible sense, and include networks and large-scale “systems of systems.” Although survivability is generally considered in the context of attacks by intelligent adversaries, knowledge of the cause, the perpetrators, or their intent is seldom known at the time of an attack. Methods to achieve survivability must be effective whether or not the cause is accidental or intentional, whether the cause is a software design error, user error, hardware malfunction, or an orchestrated attack, whether or not the perpetrators are insiders, and regardless of whether their intent is malicious or mischievous. We will use the term incident to mean any of attack, failure, or accident that, alone or in combination, threatens the ability of a system to fulfill its mission. Because determination of the cause of an incident can take arbitrarily long, survivable systems must be able to protect against and react to the effects of an incident before the cause has been identified.

© CERT is registered in the U.S. Patent & Trademark Office
The notion of essential services is central to survivability. These services must continue while an incident is in progress, or be restored within a limited time after an incident has occurred. Related concepts include resistance (traditional security augmented by techniques to limit damage), recognition (intrusion detection plus the capacity to assess damage), recovery (the ability to restore services in a timely manner), and evolution (improving the system’s capacity to resist, recognize, and recover from future incidents) [LML 98, EFL 97]. Issues of fail-soft, fail-safe, and graceful degradation are often included in requirements for survivability. Recoverability, however, is the characteristic that most clearly distinguishes survivability from other software quality attributes. Most attributes of survivability are characteristics of a system as a whole and not of its individual components. The final arbiters of survivability are the requirements that define the system’s mission and its success criteria.

Redundancy and diversity likely will be central to any effective method for achieving survivability [Lin 99]. An obvious analogy in this regard can be seen in social interactions and life processes. Genetic algorithms [For 91a] often demonstrate survivable characteristics. Genetic algorithms often achieve survivability through evolution, random mutation, and local interactions with their environment in the context of a reward system in which only the fittest survive.

Emergent algorithms share some of the characteristics of genetic algorithms. Emergent algorithms, however, are concerned not just with survivability, but with simultaneous satisfaction of mission requirements which typically include the functionality of the mission and a variety of additional constraints in the form of software quality attributes including global nonfunctional properties. These attributes include most importantly, and in contrast with genetic algorithms, constraints on performance and storage costs.

Emergent and genetic algorithms share the characteristic of being self-stabilizing [SP 97, DO 93]. They also frequently share many of the environmental constraints of unbounded networks. These include, as will be discussed below, direct communications only with local neighbors, absence of central control, and the inability of individual participant components to view the state of the system from a global perspective.

2. Survivability

Survivability shares goals with other areas of system and software quality including security, reliability, safety, dependability, modifiability, and fault tolerance [BKL 95, BBF 97]. Advances in these other areas of software quality often contribute to survivability. For example, increased security can improve the general capacity of a system to resist attacks. Improvements in the reliability of code can eliminate bugs (such as buffer overflows) that could be exploited by an intelligent adversary. Fail-soft and fail-safe techniques to enhance safety often parallel the actions required for recognition and recovery in survivable systems. Ease of modification can allow vulnerabilities to be quickly repaired, and can support the evolution of a system to maintain survivability over time.

However, survivability is not simply a combination of other software quality attributes. A system may have a set of requirements defining its survivability goals and another set defining its safety goals. Some of the requirements in these two sets may be similar, some complementary, and some conflicting. For example, reversion to a safe state that interrupts essential functionality will not be sufficient to meet the criteria for survivability unless it is followed by rapid recovery of the mission’s essential services. Requirements for survivability and fault tolerance can also sometimes be in conflict. A fault that does not jeopardize the mission may be acceptable from a survivability standpoint, but the resources devoted to addressing such a fault may be needed to satisfy other survivability requirements.

When conflicts occur among requirements for software quality attributes, tradeoffs must be made [KMB 98]. A common, but unnecessary, source of conflict is from overly specific requirements and especially those that specify how a goal is to be achieved. Another important source of conflict is design decisions that strongly support one goal but impede satisfaction of another. This often occurs when the most natural data representations and algorithms to support functional requirements conflict with the required software quality characteristics. Another source of conflicts arises from implicit, assumed, and unintended requirements that are easy to satisfy or nice to have, but interfere with satisfaction of explicit requirements.

Resolution of conflicts among requirements for software quality attributes is particularly difficult when global nonfunctional properties are involved, when solutions are highly intertwined with the satisfaction of orthogonal requirements such as functionality and performance, and when a repertoire of well known solutions approaches does not exist. Survivability is in all three of these categories.

The term nonfunctional property [Ebe 97, CNY 96] has essentially the same meaning as software quality attribute, but suggests a global rather than local perspective. A global viewpoint is particularly appropriate for survivability. Survivability involves inherently global properties that relate to the overall mission and often do not have a corresponding interpretation within individual components. Survivability, like performance and security, cannot be achieved in isolation and is highly intertwined with the implementation details of the system’s functionality.
This latter characteristic also creates conflicts and opportunities for tradeoffs among survivability, security and performance goals. Finally, survivability is a relatively new area of concern that has tended to borrow techniques from a variety of areas including security, safety and reliability. Survivability does not currently have an established repertoire of demonstrably effective techniques.

The desire to achieve inherently global properties, combined with a need for solutions that are unavoidably intertwined with the functionality and implementation details of a system, necessitates a non-traditional approach. We begin with the view that the purpose of a computational system is not to compute a value, but to continuously fulfill a mission. The requirements of the mission define constraints on any satisfactory implementation. In addition to the mission’s functionality, the requirements may specify a variety of software quality attributes in the form of nonfunctional local and global properties that the system must maintain. Mission goals may include absolute or relative time constraints. They may be influenced by dynamically changing conditions within a system and its environment. The global properties defining a mission must be continually maintained in real-time through an on-going computation [DS 93, DO 93].

Global nonfunctional properties of systems, including those needed to fulfill mission requirements for survivability, performance and safety, often arise (or falter) through interactions among components of the system. Global properties that prevail for a system as a whole, but cannot or do not exist within individual components of that system, are called emergent properties. Survivability, for example, cannot be achieved at the level of atomic system components because each component represents a single point of failure for its own survival. Thus, distribution and implementation in unbounded networks, which often characterize the problem space of missions with survivability requirements, also may be essential to solutions satisfying those requirements.

3. Why an Emergent Approach is Needed for Survivability

The primary paradigm for software design traditionally has been hierarchical decomposition of functionality. Hierarchical decomposition serves to partition problems into smaller, more manageable parts that can be developed independently. The decomposition process is applied recursively producing larger numbers of smaller and simpler components. The partitioning most appropriate for functionality (or for any particular software quality goal) often conflicts with the best partitioning to achieve other needed software quality goals. Communications among the components is managed along the hierarchical structure of the design. The resulting implementations display the same hierarchy in their functional composition. In sequential computations, hierarchical decomposition offers organizational advantages with minimal overhead. In parallel systems with multiple processors and central control, it enables parallel processing at the leaves with somewhat increased communications overhead. In loosely coupled distributed systems with multiple independent processors, the leaf processors can be more heterogeneous and the leaf processes more diverse in their implementation, but coordination and communications significantly increases the overall cost. Nevertheless, for reasons of tradition and analogy with sequential systems, hierarchical decomposition remains the primary paradigm for distributed systems.

The first widespread use of alternative paradigms has been on the Internet. Applications include, most conspicuously, the use of chat rooms, overall governance of the Internet, combining of independently developed search tools in ways unanticipated by their authors, and the process of Internet routing. These mechanisms have arisen on a largely ad hoc and consensus basis, with individuals and automated processes at a variety of nodes contributing both computation and control, typically without insight to the overall computation or to its complete purpose and goals. Contributing sites seldom anticipate or learn the full extent of applications and purposes to which their data or computations contribute. Individual computations combine skills, specialized computations, and information available only at a few nodes of the network to achieve applications not previously or otherwise possible. These computations exhibit emergent characteristics, but because of their ad hoc development, their ability to satisfy software quality attributes cannot be assured.

Hierarchical decomposition also imposes rigidity on system structures by requiring that boundaries between logical components of the system be determined at design time. This lack of flexibility conflicts with adaptations often needed, for example, in recovery and evolution of survivable systems.

Hierarchical decomposition for a given software quality attribute typically involves developing the desired property in one or more leaf computations and then recursively composing those computations in a way that preserves the desired property throughout the composition. In a sequential computation, for example, allowed performance costs for each component can be allocated among their immediate subcomponents at each level of the hierarchy. Security, in the form of firewalls, can be applied to the communications of leaf components at the edge of a system, and their protection maintained through every compositional level of the system by prohibiting direct external communication by all other components and compositions. Such methods, however, do not work when the desired global property cannot be
generated in leaf components (as with survivability) or cannot be preserved through the process of composition (as with safety).

Effective solutions must be implemented on an unbounded network. If a desired global nonfunctional property cannot be achieved locally in individual components or cannot be maintained through functional composition, then there must be multiple active entities within the system that can work cooperatively to generate and maintain that property. For our purposes, a network is taken to mean any execution environment in which there are large numbers of (physically or logically) autonomous communicating processes, hereafter called nodes.

We further restrict our domain of interest to unbounded networks including the Internet, intranets and extranets. An unbounded network is characterized by distributed administrative control without central authority, by limited visibility beyond the boundaries of local administration, and at any individual node, by incomplete information about the network's topology and functionality [EFL 97]. In an unbounded network there is neither a fixed bound on the number of nodes nor on the configuration of interconnections among them. Furthermore, changes in the numbers of nodes and the interconnections among them is not under the control of any single node or entity. As the numbers of nodes and interconnections among them changes, so does the topological architecture of the system. Physical implementations impose a small upper bound on the number of immediate neighbors for each node of the network. In unbounded networks, such as the Internet, there is no unified administrative control, and no one node has complete knowledge of the topology of the network at any given time.

In the context of unbounded networks, the limitations of traditional software design approaches, such as hierarchical decomposition, motivate the exploration of emergent algorithms. Emergent algorithms have the potential to generate and maintain global nonfunctional properties in the context of unbounded networks whether or not those properties can be generated locally in individual components. Although survivability is an obvious application for emergent algorithms, emergent algorithms also may be beneficial for achieving other nonfunctional global properties including those of performance, safety, security, and reliability. In the context of networked and distributed systems, emergent algorithms offer an alternative that generally has lower communications cost than would a hierarchically decomposed solution of the same functionality and may prove beneficial even where traditional algorithms are well known.

4. Characteristics of Emergent Algorithms

The emergent character of attributes for survivability and the need for more effective methods to generate and maintain other global nonfunctional properties in unbounded networks, suggest an approach analogous to those of natural processes in biological systems, social behavior, and economic systems in generating emergent properties. The current fascination with emergent computation [For 91a, For 91b, Res 96] arises to a great extent from observing global properties that are naturally generated, but do not exist anywhere locally within components of those systems.

An emergent algorithm produces global effects through cooperative local actions distributed throughout a system. The task here is to refine this notion into guidelines that are useful in developing and analyzing emergent algorithms that fulfill explicit functional and nonfunctional requirements for practical applications. A first step in this process is to characterize emergent algorithms in a manner that differentiates them from other computations in a formal and testable manner.

Distribution of Results

A useful definition of emergent algorithm would include computations that produce global properties that do not exist locally, but should not exclude similarly structured computations in which the properties exist locally as well as globally. This line of thought quickly leads to the realization that any attempt to characterize emergent algorithms entirely by the effects they produce will be either too limiting or will include any computation that generates global properties. Their characterization must include aspects of how the computation is organized, structured, and carried out.

Emergent algorithms cannot be just those computations with global nonfunctional properties that do not exist locally. If an emergent algorithm is extended to measure the global properties it generates and to report those results locally to each node, it should still remain emergent. Neither should that same algorithm become non-emergent if it destroys the generated global properties after measuring and reporting them locally. Likewise, a conventional algorithm should not be considered emergent because it redundantly produces the same result locally within each node of a system. The final distribution of the results of a computation cannot serve as a useful criterion to distinguish emergent algorithms.

An emergent algorithm may produce results that are distributed globally but do not exist locally, that are entirely local, or that are represented both locally and globally.
**Immediate Neighbors**
Intuitively, emergent computations involve local actions and distributed cooperation. The concept of “local actions” includes the idea that nodes involved in a computation can communicate directly only with their immediate neighbors. There is no need to place limitations on the topology of the interconnections among nodes and their neighbors, but there must some limitation on the number of immediate neighbors per node. Any number proportional to the total number of nodes would fail to impose a formal limit. It is convenient and often desirable to limit the number of immediate neighbors per node to a fixed constant independent of the number of nodes. Any value less than proportional to the number of nodes, however, could be used.

Each node executing an emergent algorithm can communicate directly only with a number of immediate neighbors that is less than proportional to the total number of nodes in the system.

**Global Visibility**
Global visibility is the ability to observe the state of nodes throughout a system. If an emergent algorithm could read the state of nodes beyond its immediate neighbors (i.e., nodes proportional in number to the size of the system), it would violate the communications requirement above. Similarly, central control is the ability to alter the state of nodes throughout a system. If an emergent algorithm could write to nodes beyond its immediate neighbors, it likewise would violate the communications requirement above.

Each participant in an emergent computation can neither read nor write directly to nodes proportional in number to the size of the system. An emergent algorithm can make use of neither global visibility nor central control.

**Distributed Information**
Emergent computations are distributed and cooperative. For convenience and by custom, distributed usually means that the number of nodes involved in a computation is proportional to the total nodes in the system. For formal reasons it will sometimes be necessary to permit lower bounds that are still dependent on the total system size. A constant lower bound, however, would be inappropriate because it would not necessarily exclude any computation regardless of the number of nodes involved. By cooperative we mean that there is communication among the nodes involved in a computation. Conceivably, an algorithm could involve arbitrary numbers of nodes in unnecessary communications and irrelevant computations that would not influence the results. To eliminate this possibility, we require that the lower bounds on the number of participating nodes apply to those nodes that are involved of necessity. That is, at some point during the computation, state information must be distributed as an inherent requirement of the solution, and often of the problem.

Results of an emergent algorithm must be critically dependent on information that is distributed over multiple nodes whose number is not bounded by any constant independent of the size of the system.

**Protocols**
Whether in real-life situations, genetic algorithm, or networked computer systems, the topology of the interconnections that give meaning to the term immediate neighbor can change frequently. In the physical world, immediate or near neighbor is typically defined by either distances or line of sight, both of which change as the participants change position. Computerized networks are regularly reconfigured or expanded to satisfy changing needs. Thus, although every node participating in an emergent algorithm must be able to communicate with its immediate neighbors, the algorithm itself should not depend on knowledge of the overall network topology. The critical role of communications and relative unimportance of topology suggests that many aspects of emergent algorithms could be expressed as protocols.

In the absence of global visibility and control, the only aspects of emergent computations not encompassed by protocols are state transitions occurring entirely within individual nodes. The state transitions (i.e., local algorithms or processes) within each node could be identical throughout the system or unique to each node. In practice, the state transitions within nodes can most conveniently be viewed as shared by all nodes, but with local adaptations as a function of either static and dynamic local conditions.

An emergent algorithm can be expressed as a protocol together with a set of local state transitions.

**Undirected Communications**
Perturbation and undirected communications can be beneficial to the success and performance of emergent computations. Computations cannot adapt or converge to solutions until they have access to the appropriate information. In the context of emergent algorithms, information can only be transmitted through sequences of immediate neighbor communications. The available
sources and needed destinations of information are often unknown locally. Strategies that depend on sending information only when and only to where it is known to be needed or on requesting information only when needed and only from where it is known to be available, may be unsuccessful. Performance of an emergent algorithm will sometimes be improved by using otherwise idle communications capacity to transmit available information that might be needed elsewhere.

Emergent computations can be stochastic. The presence of undirected communications and decision making in the absence of complete information permits algorithms that sometimes but not always succeed in satisfying their goals. Emergent algorithms with stochastic properties of this kind may not only help model social and biological systems with similar behavior, but enable less than perfect solutions where perfection is impossible or unaffordable. Few believe that any algorithm could guarantee a particular node of the Internet absolute protection from information-based attacks, but there may be emergent algorithms that will ensure arbitrarily high probabilities of protection to missions that are distributed over the Internet.

Undirected communications within an emergent algorithm may produce a more efficient computation, but with greater variance. Activity, in the form of frequent communications among nodes, has the effect of rapidly diffusing information to where it may be needed, including situations in which the appropriate information sources and destinations are unknown.

Unpredictable Environment

Even without undirected communications, emergent computations may demonstrate stochastic behavior. Emergent algorithms are inherently distributed in character, often involve unreliable information and untrustworthy participants (automated or human) including those outside the administrative control of the system, may be implemented on platforms that are dynamically changing and not fully known, and may involve components that have been compromised. These unknown and unknowable characteristics of the realm of emergent algorithms can lead to random or at least not fully predictable results. Thus, this form of stochastic behavior from emergent computations is not inherent in the algorithms themselves, but as a practical matter is often dictated by characteristics of the application. Any application with survivability or security requirements, with accompanying assumption of compromised components, will have a similar degree of unpredictability whether or not it is implemented with an emergent algorithm.

An emergent algorithm may demonstrate stochastic behavior or have outcomes that are guaranteed only as a function of some statistical properties of the problem space.

Definition of Emergent Algorithm

Given all of the above considerations, we are led to the following definition.

An emergent algorithm is any computation that achieves formally or stochastically predictable global effects, by communicating directly with only a bounded number of immediate neighbors and without the use of central control or global visibility.

As the definition indicates, an emergent algorithm produces global effects through cooperative local actions distributed throughout a system. And, as discussed earlier, emergent algorithms can be expressed as a protocol together with local state transitions.

5. Emergent Algorithms for Practical Applications

If one is modeling natural phenomena, biological processes, or social systems that involve emergent properties, performance may not be a major concern. Our primary concern, however, is with practical use of emergent algorithms in applications where conventional algorithms are unavailable or too expensive. In practical applications, execution and storage cost considerations impose additional constraints on the design.

Emergent algorithms appear most useful in unbounded networks where input data or results are distributed throughout the system. The Internet in particular provides an obvious example of a large and growing system of constantly changing size and topology. Each node lacks global visibility and is connected to only a small number of immediate neighbors. No node or small group of nodes can exercise control over any significant portion of the network. Examples of applications where emergent solutions appear particularly appropriate include message routing, detection of intruder incidents, and domain name translation on the Internet; protection of critical infrastructures such as the electrical power grid, air traffic control system, and interbank financial transactions from electronic attack; and safety critical application involving highly distributed components.

Distributed solutions that merely partition sequential solutions and then distribute the parts over a network often require resources in each node proportional to the size of the network. Emergent algorithms offer the possibility of simultaneously fulfilling survivability,
performance, and other nonfunctional software quality goals in highly distributed systems operating on unbounded networks.

**Design Strategy**

The strategy underlying any emergent algorithm is that each node should act independently to contribute to global objectives in a way that if sufficiently many other participating nodes acted with similar objectives, satisfaction of the global objectives would be assured. Each node must not only contribute directly to the objectives based on the resources that it has access to, but must anticipate the needs and potential contributions of its neighbors. Information that must be transferred among nodes to fulfill the mission goals can be communicated whenever it is available, but additional synchronization and communication to support the structure of the algorithm itself is not needed. Thus, emergent algorithms generally have lower communications costs than conventional distributed algorithms. In the special context of survivability, each node also must assume that its neighbors can never be fully trusted and at times may be rogues.

**Practical Considerations**

What performance costs are anticipated for emergent algorithms? Under what circumstances would it practical to use an emergent algorithm? Because they can exploit the concurrency of execution in multiple nodes simultaneously without the need for extra synchronization and communications imposed by hierarchical decomposition, emergent algorithms may be appropriate for applications where traditional distributed solutions are limited by the elapsed time of the computation. They may be appropriate in applications such as Internet routing where conventional algorithms require storage in each node proportional to the size of the network. Because emergent algorithms often have greater redundancy of computation than traditional distributed implementations, they may be inappropriate where the primary limitations are total system-wide performance costs. A possible compensating factor is reduced computational costs for synchronization and communications. Also, emergent algorithms would generally be expected to be preferable where the data and results are distributed, while a more conventional distributed approach would be preferable where the data and results are local to a few prescribed nodes. Although emergent algorithms can be used to generate stochastic properties, stochastic results are not inherent in the use of emergent algorithms.

Better insight into the applicability of emergent algorithms can be gained by examining the practical constraints on distributed and networked computations. Ideally, the total execution cost (measured in processing cycles) of a distributed implementation would be the same as the total execution cost of the best possible sequential implementation of the same functionality, but with uniform distribution of those costs over all the participating nodes. This allocation would result in identical total execution cost for the most execution efficient sequential and distributed implementations, and elapsed time for the distributed computation proportional to the total processing time divided by the number of participating nodes. This ideal situation can never be achieved because the distributed solution will have additional costs for inter-node synchronization and communication. Also, unless the data can be partitioned into independent parts, at least some of the data must be redundantly stored. The idealized bounds on elapsed time cannot be realized unless (1) execution costs in the distributed implementation can be precisely balanced among the nodes, (2) communications overheads are nonexistent, and (3) redundant computation is unnecessary.

**Performance Bounds**

Practical bounds, however, can be established for acceptable performance in a distributed implementation. Let C be the total execution cost and S be the total storage requirements for the best known sequential algorithm that achieves the desired functionality. Let N be the number of nodes participating in a distributed implementation. By the above arguments, the total execution cost of a distributed implementation must be at least O(C)\(^1\). The execution cost per node must be at least O(C/N). The elapsed time for the total computation must be at least O(C/N). These establish lower bounds on what is achievable, with the possible exception that time vs. space tradeoffs in the sequential algorithm may have imposed greater values on S than are inherently required by its information content. The lower limit on space per node is thus O(I/N) where I is the minimal storage for the information content of S.

When analyzing performance of any application that is distributed over a large network, it is convenient to focus on the costs per node rather than the total system costs. In fact, in the absence of global administrative control, few are concerned with total network costs. Instead, most are concerned with the execution and storage costs for nodes under their control. If a distributed algorithm had total costs per node greater than O(C) for execution, O(S) for storage, or O(C) for elapsed time, it would generally be unacceptable because replication of the sequential algorithm would do at least as well. It is also undesirable for local (i.e., per node) costs to be proportional to N for an unbounded N. Thus, if a distributed algorithm had per node costs of O(N) or greater for any of execution, storage, elapsed time, or communications bandwidth, local resources requirements

---

\(^1\) Hereafter, the symbol "O" refers to order of a computation. O(N), for example, may be read as "is proportional to N."
would be proportional to the size of the network and impractical to implement. This establishes the upper and lower bounds on practical implementations.

For practical implementations in an unbounded network, an emergent algorithm must have execution costs per node not less than \( O(C/N) \) and less than \( O(\min(C,N)) \), storage costs per node not less than \( O(I/N) \) and less than \( O(\min(S,N)) \), elapsed time in each node not less than \( O(C/N) \) and less than \( O(C) \), and per node communications bandwidth requirements less than \( O(N) \), where \( N \) is the number of nodes, \( C \) and \( S \) are respectively the execution time and storage requirements for the most efficient sequential algorithm of the same functionality, and \( I \) is the minimum storage required for the information in \( S \).

These bounds are highly constraining and thus provide strong guidance to the design of any practical emergent algorithm. Development of an emergent algorithm is beyond the scope of this paper, but the approach is illustrated in outline form for the example below.

**Internet Routing Example**

Internet routing is a computation that, given a source node and a specified destination node, determines a sequence of immediate neighbor connections linking them. Let \( P \) be the length of the shortest path between two nodes, \( A \) be the number of bits in each nodes address, \( M \) be the total number of messages throughout the network, and \( K \) be an upper bound on the number of immediate neighbors per node. For the most (execution) efficient sequential routing algorithm, \( S = O(N^P*A) \), \( C = O(M*A*\log_2 N) \), and \( I = O(N*K*A) \). We note that \( P \) and \( A \) are \( O(\log_2 N) \), \( K \) is constant, and \( M \) is likely \( O(N) \).

Using these values, our criteria say that the storage cost per node for any distributed algorithm cannot be less than \( O(I/N) = O(K*A) \) and for an affordable emergent algorithm should be less than \( O(\min(S,N)) = O(N) \).

The latter is a very severe constraint requiring that each local routing table not know of even a constant fraction of the network and precluding any partitioning of the representation used for \( S \). That is, any practical solution must have a per node routing table proportional at most to some power of \( \log_2 N \). The lower bound of \( O(K*A) = O(\log_2 N) \) suggests that this may be possible, although not achieved in the most efficient sequential algorithm. We also know that any solution must be able to route messages to any destination address.

<table>
<thead>
<tr>
<th>Entry No.</th>
<th>Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Path from current to ( \alpha )</td>
</tr>
<tr>
<td>2</td>
<td>Path from current to ( \alpha )</td>
</tr>
<tr>
<td>3</td>
<td>Path from current to ( \alpha )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( \log_2 N )</td>
<td>Path from current to ( \alpha )</td>
</tr>
</tbody>
</table>

**Figure 5.a**

Routing table for node at address \( \alpha \)

Combining these two constraints yields a routing table at each node with one entry for each bit in the binary representation of its address (see Figure 5.a). The i-th entry of a table will contain a path to some (preferably the nearest) node whose address is the same as the table’s node in the most significant i-1 bits, but differs in the i-th bit. Thus, every destination address will share a prefix with exactly one entry in each routing table. Furthermore, the routing table at the terminal of a stored path is guaranteed to have an entry with a prefix that matches at least one more bit of the final destination address than does the current node. Such tables will enable routing from any source to any destination node in at most \( O(A) \) steps (see Figure 5.b). The table size at each node will be \( O(P*A*\log_2 N) \). Each table lookup will have execution costs of \( O(A*\log_2 N) \) and processing time for the resulting entry is \( O(P*A) \), for a total of \( O(P*A) \). Because each message may require as many as \( O(P) \) steps with a lookup at each step, the total execution cost per message is \( O(P*A) = O(\log_2 N) \). Both the time and space bounds per node are less than \( O(N) \), and thus the criteria are satisfied. By comparison, a distributed algorithm produced by replicating the sequential algorithm at each node and uniformly partitioning its tables by source node, would yield storage and execution costs per node of \( O(N^P*A) \) and \( O(A*P) \), respectively. The storage requirements would still be unacceptable and the performance no better than the emergent algorithm.

A simplified version of the emergent routing algorithm is given in Figure 5.b below. Each routing request received by a node includes a destination address and a path. Unless the path is null, it must originate at an immediate neighbor of the current node and terminate at any node whose address shares more leading bits with the destination address than does the current node’s address. Generally, routing requests originating within a node would give null as the initial path, so that the algorithm will select a routing path from its own table. It is beyond the scope of this paper to include a full Internet routing algorithm with an accompanying emergent algorithm to generate and update the routing tables.
Additional research is needed to extend this work to must adapt or evolve in response to changing conditions. interconnection, that rely on self-stabilization, or that systems with highly dynamic structure and computations. Emergent algorithms are best suited to characteristics are the natural result of emergent software quality attributes in the form of global nonfunctional properties, or where solutions are being goals include generation and maintenance of global survivability in networks. Emergent algorithms offer the possibility of achieving functional and software quality goals for distributed applications in unbounded networks with per node costs that grow more slowly than the size of the network.

The work reported here focuses on survivability only as a motivating factor for the development of emergent algorithms. Emergent algorithms offer the possibility of satisfying survivability requirements for which there are not currently known solutions. Achieving that potential, however, will require greater understanding of issues relating to the validation and maintenance of trust among nodes. Quantifying the success of emergent algorithms as a new method for enhancing survivability in unbounded systems also will be difficult because of the stochastic nature of the attacks, failures, and accidents that the solutions must counter.

6. Conclusions

This paper introduces the important concept of emergent algorithms in the new field of survivable systems. Emergent algorithms offer the possibility of practical solutions to problems of survivability in unbounded systems and other domains where mission goals include generation and maintenance of global nonfunctional properties, or where solutions are being developed in the context of unbounded networks. We have shown that emergent algorithms can be formally distinguished from other computational paradigms, have developed a strategy for the design of practical emergent algorithms, and have illustrated that strategy in outline form for a problem in Internet routing.

Survivability involves generation and maintenance of software quality attributes in the form of global nonfunctional properties that often cannot exist at the level of individual system components. Such characteristics are the natural result of emergent computations. Emergent algorithms are best suited to systems with highly dynamic structure and interconnection, that rely on self-stabilization, or that must adapt or evolve in response to changing conditions. Additional research is needed to extend this work to evolutionary and self-stabilization aspects of emergent computation, and to build a variety of emergent algorithms that can be used to demonstrate their capabilities, measure their limitations, and refine the methods for their development.

Distributed algorithms that only partition sequential solutions and distribute the parts over a network often require resources in each node proportional to the size of the network. Emergent algorithms offer the possibility of achieving functional and software quality goals for distributed applications in unbounded networks with per node costs that grow more slowly than the size of the network.

7. References


