Combatting Maelstroms in Networks of Communicating Agents

James E. Hanson and **Jeffrey O. Kephart**
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{hanson,kephart}@watson.ibm.com

Abstract

Multi-agent systems in which agents can respond to messages by automatically generating and multicasting other messages are inherently vulnerable to a phenomenon that we call a *maelstrom*. We define a maelstrom to be a self-sustaining chain reaction in which a single message can unintentionally trigger the generation of a rapidly growing, potentially infinite number of messages, quickly incapacitating the communications network. There is reason to fear that modest advances in agent technology and usability could lead to spontaneous maelstroms on the Internet in the near future, particularly in the realm of electronic mail. In this article we describe various classes of maelstroms that may arise due to automated forwarding of messages and propose a novel and practical means of combatting them.

Introduction

The rapidly growing literature on multi-agent systems (e.g., [Demazeau, 1998]) is largely concerned with the design or behavioral study of systems that have been engineered from a global or systemic standpoint. By this we mean that, in addition to defining the individual agent behaviors, the system designers carefully specify the agents' roles in the collective and choreograph the sequences of interaction among them. Often, the agents are endowed with an intimate understanding of other agents that may be present in the system. The focus is typically on demonstrating that the individual agents and the multi-agent system as a whole behave as intended by the designer.

Much less well investigated, however, are what might be called *emergent* multi-agent systems, which arise through the haphazard aggregation of individual agents that can communicate, even if only marginally, with one another. By definition, the agents in such a system were designed separately, in a variety of ways, to pursue a variety of goals. Furthermore, the set of agents and agent types typically changes over time as existing agents become inactive and new agents become active. No designer can have knowledge of all the agent types that might interact with his agent, let alone the power to modify the other agents individually or in aggregate.

The Internet appears to be a conducive environment for the development of this more informal type of multi-agent system. Apart from the underlying protocols that make communication possible, the information content, usage patterns, computational properties, and goals and intentions of agents on the Web are unconstrained and, in practice, unknowable. This is especially true in the context of electronic commerce, where organizational boundaries act as barriers to the gathering of information and the imposition of global controls.

To the extent that agent technology comes into common usage, the number, variety, and sophistication of agents on the Internet will continue to grow. Inseparable from this development is a correspondingly growing potential—which is both threat and promise—for the emergence of novel and unforseen collective behaviors in the population of agents. Even in engineered systems, designing against destructive collective behaviors can be a subtle and difficult task; but at least in principle, it can be done through sufficiently careful adjustment of the agents' individual behaviors, of their interactions, or both. This kind of global legislation is simply impossible in an emergent system. Therefore it is vitally important to understand any collective modes of behavior that may arise in such systems, and to develop methods whereby an individual agent may, so far as is possible, exploit those which are favorable and combat those which are unfavorable.

In this paper we describe one potentially disastrous class of collective behavior that may arise in a multi-agent milieu of the sort just described, and propose a method whereby individual agents may combat it. For definiteness, we model an agent as an information processing system that receives messages from other agents, processes them in some way, and possibly sends messages to other agents as a result. In a population of such agents, it is natural to suppose that a single message could trigger a chain reaction of messages, and that, under certain circumstances, this chain reaction might be self-sustaining. Manber [Manber, 1990] has reviewed a number of real-world chain reactions of this type, which range from the Internet worm of 1988 to infinite loops of low-level error messages in a local area network, to the cycling of e-mail messages among a user's multiple accounts. In all the examples he gives, the chain reaction was due to some hardware or software failure at one or more of the workstations involved, or to an unnoticed flaw in one of the protocols used. With the advent of user-programmable intelligent agents, however, comes the possibility of a new type of chain reaction, in which the actions of any individual agent, considered in isolation, are entirely useful, intentional and benign. We will refer to self-sustaining chain reactions in networks of agents, whether due to a failure or a supposedly innocent action, as *maelstroms.*

The rest of the paper is organized as follows. A scenario for how a maelstrom of electronic mail messages might develop is given in section 2, followed by a partial taxonomy of maelstrom types. Previous methods for preventing maelstroms are critiqued in section 3. In section 4, we describe a novel anti-maelstrom technique that has several advantages. The performance of an implementation of the algorithm is evaluated in section 5, and simulation studies are used in section 6 to demonstrate that sufficient deployment within the agent population will successfully prevent maelstroms. We close with a summary and discussion of future work.

Maelstroms

One type of user-programmable agent that already exists in widely-used commercial electronic mail software packages is a mail forwarding agent. The forwarding agent scans the header and body of an incoming mail message for specified keywords. If the right combination of keywords is found, the agent automatically forwards the message to a designated set of recipients.

Now consider the following scenario: A typical computer user ("Fred") is one of a small group of friends who exchange jokes with one another via e-mail. He decides to automate distribution of jokes. He instructs his mail agent to forward any incoming mail with the word "Joke" in the subject line to his friends.[1] Over time, this idea occurs independently to more and more users. Jokes start getting forwarded several times, from mailing list to mailing list. One day, the social network of automated forwarders closes upon itself, and one of the jokes that Fred's agent had forwarded is sent back to Fred. It is of course automatically forwarded, and begins the second of an endless succession of trips around the cycle. Every time it goes around, everyone who originally got the joke gets it again, and forwards it again. Furthermore, because both the original message and each copy are forwarded independently, the number of copies of the message grows rapidly with time. Before long, the network used for e-mail delivery is swamped, and can't be used to transmit useful information to Fred or anyone else—even those not involved in the mail loop.

[1] It took us 25 seconds to program an agent in Microsoft Outlook Express to perform exactly this operation. Similar functionality is available in Lotus Notes.

The above is an example of a *simple maelstrom,* in which messages are forwarded verbatim. We may identify other specialized types of maelstrom as follows:

Additive maelstroms. This is automatic message forwarding in which additional information is added to the message before it is sent. The additional information may be of any nature from the most insignificant, such as an extra blank line added to the bottom of the original message, to the most important, such as a complete disavowal of the original by its author.

Combinatorial maelstroms. This involves forwarding in which several messages or parts of messages are combined to form a single new message prior to forwarding. An example of such a message is an automatically generated personalized newspaper that can be received by another agent and in turn used by it as fodder for its own automatically generated newspaper.

Maelstroms with finitary transformation. As the message is forwarded from agent to agent, it is transformed into a succession of variations of which there is a finite (usually small) number of types. Simple examples include conversion of the message to all capital letters or all lower case letters, adding or removing whitespace characters, or applying simple character encodings.

Previous ID-based solutions

Previous approaches to preventing chain reactions in networks have centered on inserting identifiers into header fields of messages. One such approach was proposed by Manber [Manber, 1990]. The key concept is to assign a unique ID to each newly generated message in the network, and to insert this ID into the header of the message prior to forwarding. At each forwarding step, each outgoing message, however transformed, is given the same ID as the message that triggered it. All agents maintain a list of all IDs of messages sent, against which every incoming message is checked. If the ID of an incoming message is found in the list, it is not forwarded. When this prescription is strictly adhered to, no message is forwarded twice by any agent. No maelstrom occurs.

A second approach was proposed by Spagna specifically for e-mail messages [Spagna, 1997]. In this case, instead of assigning a unique ID to each message, each agent inserts its own unique ID into the header of each message that it sends. When it receives a message, the agent searches the header message for its own ID. If the ID is found, then it does not forward the message. This prescription also prevents maelstroms, and it has the advantage of reducing the amount of data each agent must store in order to recognize messages. It is slightly less powerful than Manber's approach because it fails to detect multiple copies of a single message that have reached an agent for the first time along distinct paths. The agent is incapable of recognizing that the incoming messages are duplicates, so it forwards both copies.

While ID-based approaches such as these can work in certain contexts, there are several important situations in which the general concept of deliberately inserting a unique identifier of some sort into a message is either inappropriate or ineffective:

1. If the agent doing the forwarding is written as an add-on to an existing system. In this case, the agent may not have write access to the message header, in which case it can't insert or manipulate IDs.
2. If the message is transmitted to other domains that employ protocols other than the one used to encode identifiers. In such situations the message header containing the inserted ID may get lost in the translation. When the message is re-injected into the system that checks for the identifiers, it is treated as new, reinitiating the maelstrom.
3. If the agent modifies the message in a way that is important to some of the other agents in the network, but unimportant to others. In this case, only those agents to whom the modification is important should resend the modified version. The identifier method prevents this, or at best severely limits it.
4. If an agent wishes to ignore some types of modification and pay attention to others when it decides whether to forward a modified version of a message. As in the previous case, the identifier method prevents or severely limits this.

The last two cases above illustrate a *semantic* drawback to ID-based methods: they effectively prescribe a fixed convention to be applied to all messages modified in any way by any agent. The decision of "same" or "different", which determines whether the original ID is preserved or a new one is generated, is made once and for all by the originating agent prior to transmission. This prevents exactly the sort of contextual, individualized decision-making that is one of the central benefits of using intelligent agents in the first place. For example, Manber's scheme requires that the forwarded message, however modified, have the same identifier as the original—or at minimum as one of a predefined, strictly limited set of variants of it. One consequence of this (pointed out by Manber) is that it severely constrains extensive interagent "conversations" (i.e., series of automatically generated messages passing between two agents). Spagna's scheme, on the other hand, requires that the modified message always be treated as new. This prevents the agents from ignoring trivial changes in a message already encountered.

Signature-based maelstrom solution

We propose a maelstrom prevention method that is based directly on the contents of the message itself, rather than a message or agent ID imbedded in the header. The content-based solution avoids the cited drawbacks of the ID-based methods. It requires no manipulation of the message header, and does not rely on the header remaining inviolate. It gives individual agents the freedom to make their own decisions about whether a new message is sufficiently distinct from a previous one to warrant forwarding.

In order to explain the content-based solution, we now follow the sequence of events that ensues when a new message M is received by the agent.

1. **Determine eligibility.** First, M is examined to determine whether it is eligible for forwarding. If M does not meet the forwarding criteria, no further processing is necessary. It may be desirable to filter out incoming duplicate messages even if they are not going to be forwarded by the agent, in which case control passes to the signature extraction step below.
2. **Match text.** If M is eligible for forwarding, a text matching algorithm that employs signature scanning is used to locate any previously forwarded messages or significant portions thereof that M may contain.
3. **Generate summary.** A *summary* of M is created from the matches located in the previous step. The summary expresses M as a combination of one or more previously forwarded messages (or portions thereof), plus any text that is unique to M.
4. **Heuristically evaluate summary.** A heuristic procedure is applied to M's summary to determine whether forwarding is warranted, and if so the forwarding takes place automatically.
5. **Extract signature, update database.** M's status as a new or previously forwarded message is reflected by updating the signature database. This may involve the automatic extraction of one or more signatures and auxiliary information from M or perhaps just the previously unencountered portions of M.

The remainder of this section is devoted to more detailed descriptions of text matching, summary generation, heuristic evaluation, and signature extraction and updates to the signature database.

Text matching

The text matching procedure attempts to locate previously forwarded messages (or portions thereof) within M. In order to reduce sensitivity to common insignificant textual transformations that occur when mail is forwarded, such as the addition of right angle brackets to indicate quoting, extra blank lines, etc., M's text is first filtered. In a current implementation, the filtering entails removing header data, replacing multiple consecutive whitespace characters with a single whitespace, removing (most) non-alphanumeric characters, and mapping all characters to lower case. In what follows, this transformed text shall be referred to as M'.

Next, M' is to be matched against similarly transformed versions of all previously forwarded messages. The naive approach of applying something like the Unix `diff` operation sequentially to M' and each of the hundreds or even thousands of previously forwarded messages P is clearly much too inefficient. Instead, a highly efficient signature scanning procedure originally developed for detecting computer viruses is used to scan M'. The relatively slow text matching procedure only needs to be invoked for those messages P that are associated with signatures located within M. If the signatures (which consist of sequences of s characters occurring in the transformed version of a given message) are selected carefully, they will hardly ever be found by chance, so that if M' contains entirely new material it is likely that no signatures will be located, and expensive matching will be avoided altogether. The careful selection of signatures is performed by an automated signature extraction technique described more fully below.

A potential problem arises when a signature for P has been located within M', but P itself has been deleted. In such a case, it is still possible to determine the location and approximate extent of the match between P and M'. The trick is to extract a small amount of extra information from P before deleting P. In a current implementation, we extract 4-byte checksums for a series of roughly concentric regions of text centered around each signature in P. Associated with each checksum is an offset of the signature from the beginning of the region and the number of characters in the region; this information constitutes a "textblock triple". Each region is roughly twice the size of the previous one, up to and including the entire message. Thus, if the transformed version of a message is 580 characters in length and the signature length is $s = 20$, there will be 5 checksummed regions with lengths of 40, 80, 160, 320, and 580 bytes.

If a signature for P has been located within M', then each textblock triple is tested against the M' to identify the longest matching region, and this information is passed to the summary generation step. If there are no matching triples, then the length of the partial match is taken to be the signature length. Note that some loss of information is expected because the length of the matching region will be underestimated consistently.

Summary generation

The list of matching regions generated in the previous step is used to construct a summary that replaces each matching block of text with a short identifier that refers to the original message in which that text occurred. A sample message and message summary are shown in Fig. 1. If there are multiple possible matches for a given region of M', preference is given to the longest match that contains that region.

Heuristic evaluation

The generated message summary is used by a heuristic that decides whether to permit M to be forwarded. [2] One can contemplate a range of possible heuristics that vary widely in computational feasibility and effectiveness. One simple heuristic would forbid the forwarding

[2]If automatic forwarding of M is prevented, a user interface could still offer the option of manual forwarding.

	Message Summary	ID
a	Meeting today, 3pm	1426
b	FYI: *[copy of message 1426]*	1465
c	CANCELLED: *[copy of message 1426]*	1466

Figure 1: Schematic depiction of message summaries. The original message (a) is not summarizable. Messages (b) and (c) are forwarded versions of (a) with some prepended text.

of M solely on the basis of the form of the message summary, not its textual content. For example, the heuristic could forbid forwarding only if M exactly contains in full a previously encountered message P, and contains no more than 10 additional characters.

More sophisticated heuristics would take into account both the form of the summary and any block of text that is unique to M. Plausible heuristics might search for the presence of keywords in the non-matching text regions, or alternatively apply a text classifier to the non-matching text regions and map the result of the classifier into a yes/no forwarding decision. As an example of the first approach, a keyword-based algorithm might decide against forwarding message (b) in Fig. 1 because the additional text consists of a recognizably unimportant word "FYI", but it would permit message (c) to be forwarded because of the recognizably important keyword "CANCELLED".

The heuristic could be adjusted or trained to the tastes of the individual user. As with any heuristic, there will be false positives and false negatives; the effects of false negatives will be investigated below.

Signature extraction and updating

The scanner uses a database that contains signatures and their associated textblock triples, pointers to messages containing the signatures, a TimeLastSeen field, whose purpose will be discussed below, and possibly other relevant information.

If M is a new message with no relationship to a previously encountered message, then one or more signatures for M are extracted and added to the database. As has been discussed, it is important to select signatures that are unlikely to appear at random in ordinary messages, so that the expensive matching operation between M and a previously forwarded message P is only called upon when there is a very strong likelihood that M and P have a significant block of text in common.

We have adapted an automatic signature extraction procedure described in [Kephart & Arnold, 1994], originally developed for computer virus signatures. At infrequent intervals (perhaps once per year), a large corpus of mail messages (for example, those stored in the user's mail archive or database) is filtered as described above. The number of occurrences of each unigram a, bigram ab, and trigram abc in the filtered database is tallied and recorded in tables as $t(a)$, $t(ab)$, and $t(abc)$, respectively, along with the corpus size Z. For English and several other languages, there are at most 128 different ASCII characters that occur in text, so this requires storage of 2^{21} 4-byte integers for the trigram table, or about 8 megabytes; the sizes of the bigram and unigram tables are relatively insignificant.

When M is submitted to the signature extraction procedure, it is first filtered to obtain M'. Next, all s-byte sequences in M are considered as candidate signatures. (Typically, s is set to approximately 20.) For each candidate signature, the tallied n-gram statistics are used to estimate the likelihood for that s-byte sequence to appear by chance in an ordinary (filtered) mail message. In our implementation, we ignore possible correlations that span more than 3 characters, allowing us to estimate the likelihood of the sequence $c_1c_2\ldots c_s$ as

$$p(c_1c_2\ldots c_s) \approx \frac{t(c_1c_2c_3)\ldots t(c_{s_2}c_{s-1}c_s)}{Zt(c_2c_3)\ldots t(c_{s-2}c_{s-1})}, \qquad (1)$$

If any trigram frequencies are zero, they are replaced with estimated trigram frequencies based on bigram and unigram frequencies.

Finally, the candidate signature or signatures that minimize $p(c_1c_2\ldots c_s)$ are selected and recorded in the database, and the TimeLastSeen field is initialized to the time at which M was received. If more than one signature is chosen, the selection criterion is modified slightly to encourage the selected signatures to be as far apart as possible. For each selected signature, a textblock triple is also computed as described above, although this step may be deferred until M is deleted.

If M is not entirely original, but is still regarded as sufficiently distinct from its ancestors, then it is desirable to reduce the potential for confusion during the scanning phase by extracting a signature that distinguishes M from its ancestors. In this case, signature extraction proceeds as described above, except that the candidate signatures are drawn only from the non-matching text regions in M's message summary.

In cases where M is identical or nearly identical to a previously encountered message P, it may not be worthwhile to extract a new signature for it. The signature database is simply updated by recording the current time in the TimeLastSeen field of each signature located within M. This keeps track of how "current" the signatures are. In order to bound the growth of the database, signatures and associated data that are deemed sufficiently old may be purged from the database.

Standalone Performance

For a single agent, the essential performance measures of the anti-maelstrom method are the storage requirements and processing time. To estimate these, we describe the application of the method to a fixed corpus of over 10,000 e-mail messages consisting of several years' mail received by one of us.

In two experiments conducted on a model F50 RISC System 6000 workstation with a 166 MHz processor, we generated two signatures per message, each $s = 20$ bytes in length. For each signature, the storage requirement was 20 bytes for the signature, a 4-byte pointer to the original message, a 4-byte LastTimeSeen field, and a number of 12-byte textblock triples (4 bytes each for the checksum, signature offset, and length) that depended on the size of the message.

In a first experiment involving 3569 messages, 6889 signatures were extracted; there were some failures due to short messages. The number of textblock triples was 39,393. Thus the total storage required was $(20 + 4 + 4) * 6889 + 12 * 39,393 = 665,608$ bytes, or less than 100 bytes per signature. If messages are purged from the database after a year of inactivity, a user receiving 5,000 messages per year requires just a one-megabyte signature database.

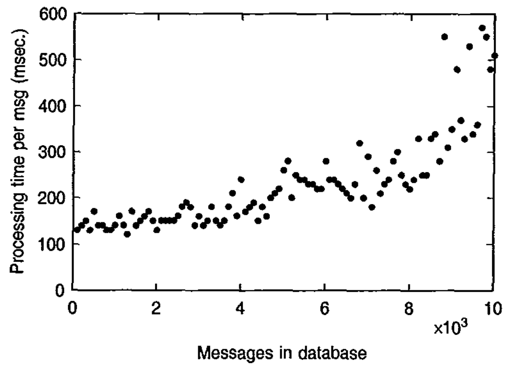

Figure 2: Processing time vs. number of scanned files when signatures are simultaneously being extracted.

In a second experiment, we built a signature database from scratch by running through 10,000 mail messages. Each message was scanned using the current signature database, and whenever a signature was found the appropriate full-text match was carried out and a message summary was generated. Then, two signatures and their textblock triples were extracted for the message, the database was updated, and the next message was processed. The amount of time required to process each message (averaged in blocks of 100 messages) is plotted in Fig. 2. Early on, the processing time per message is somewhat less than 150 msec. Practically all of this time is taken by signature extraction. As the number of messages database grows in size towards a few thousand, the processing time per message grows noticeably because the scan starts to become slower; at 5,000 messages, it takes 200 to 250 msec per message. This would be the asymptotic performance for a database that is kept pruned to this size. Especially when one considers that no algorithmic tuning has been done, this suggests that the method is entirely practical, and would have very little impact on storage or CPU usage.

Performance in a Network

The second test of an anti-maelstrom method is in its performance as deployed across a network. In the context of an emergent multi-agent system, we cannot hope to universally suppress redundant messages. At best, a "successful" solution is one which reduces the number of undesirable messages to a level sustainable by the network and by the agents themselves. Here, we can only present the beginnings of such an evaluation pending more detailed investigation. We will present preliminary results on the behavior of maelstroms as a function of two separate parameters: (i) the rate of misidentifications induced by the anti-maelstrom method; and (ii) the fraction of agents employing it.

As noted, the anti-maelstrom method will induce a certain degree of error due to misclassification of messages by the AI heuristic. These will take the form of *false positives,* in which a message that is not part of a maelstrom is misclassified as being one that is, and *false negatives* in which a "bad" message is misclassified as "good". Obviously the actual false positive and false negative rates will depend sensitively on the nature of the transformations of the messages being sent and on the actual heuristics in use. In practice, the transformations will be the forcing function that drives the evolution of the heuristic. Here, we do not speculate on the form they will take, but rather show the behavior of the system as a function of the error rates, however they may come about.

For the simulations presented here, we generated a random digraph with $n = 1000$ nodes and $k = 4000$ edges, in which each edge's source and destination were selected at random, but in which loops (self-edges) were not permitted. Then we removed all nodes and edges outside the giant strongly connected component, which resulted in a graph with $n = 969$ nodes and $k = 3878$ edges in which a communication path existed between each pair of nodes. At each time step, all nodes were updated synchronously. When updated, a node processed all incoming messages, determining whether each was new or whether it had been received already. If classified as new, the message was forwarded verbatim to each of the node's downstream neighbors (but not back to the sender); if not, the message was not forwarded. The network was seeded with a single outgoing message from a randomly selected node.

To measure the behavior under imperfect classification, the classifier was set to randomly misidentify messages with false negative rate r_N. Then the number of messages extant in the system was measured as a function of time. Figure 3 shows the number of messages plotted vs. time for $r_N = 0.0$ to $r_N = 1.0$ in increments of 0.1, plus $r_N = 0.25$. Most notable is the qualitative change in behavior at $R_N \approx 0.25$. For values of r_N greater than this critical false negative rate, the number of messages grew without bound. (In practice, the number of messages is bounded by the maximum carrying capacity of the network.) For $r_N < R_N$, after an initial period of exponential growth, the number of

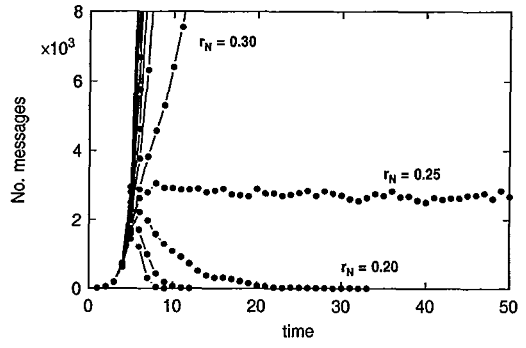

Figure 3: Number of messages vs. time for different false negative error rates.

messages eventually fell back to 0. At $r_N = 0.25$, the system reached a steady state in which the maelstrom persisted indefinitely but the number of messages did not grow. Because the average outdegree of each agent is approximately 4, at $r_N = 0.25$, each agent sends about as many messages as it receives.

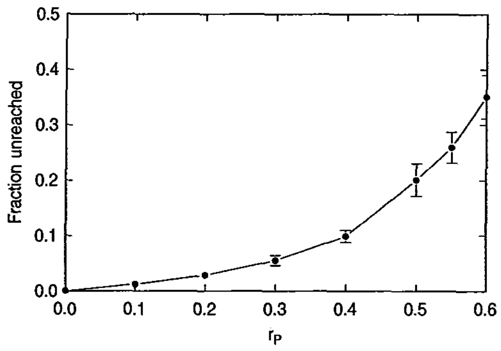

Figure 4: Fraction of nodes unreached vs. false positive error rate.

The effect of false positives is summarized in fig. 4, which shows the fraction of nodes that failed to receive even one copy of the message as a function of the false positive rate r_P. The data were averaged over 10 trials using different random number seeds. The mean values at each r_P are plotted, bracketed by error bars showing the standard deviations. Beyond $r_P = 0.6$, the fraction of unreached nodes quickly approached 1.0.

When both r_N and r_P are nonzero, the redundant messages generated by positive r_N and the unsent messages due to positive r_P tend to counteract each other, but only partially. For example, when $r_N = 0.10$, the fraction of unreached nodes at various r_P is typically reduced by approximately one half. But the critical false negative rate R_N remains at approximately 0.25 for values of r_P up to about 0.25.

To measure the behavior for different heterogeneous populations, we disabled the anti-maelstrom solution in a randomly selected fraction f_B of nodes ("bozos"), by

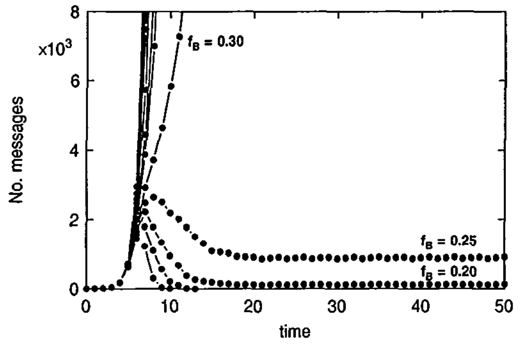

Figure 5: Number of messages vs. time for different fractions of "bozo" agents.

setting $r_N = 1$ and $r_P = 0$; the rest of the nodes had $r_N = r_P = 0$. Figure 5 shows the number of messages extant at a given time, plotted vs. time for $f_B = 0.0$ to $f_B = 1.0$ in increments of 0.1, plus $f_B = 0.25$. One notable finding is the similarity, both qualitative and quantitative, with the data of Fig. 3. For f_B below a critical value $F_B \approx 0.25$, the maelstrom was eventually quenched. For $f_B > F_B$, the number of messages increased exponentially. At $f_B = F_B$, the maelstrom persisted, but the number of messages reached an equilibrium value. Note that at $f_B = 0.25$, each agent—and in particular, each bozo—has on average one bozo for a downstream neighbor.

Obviously, more investigation is needed to verify these findings and to interpret them. Of particular interest are the apparent relationships between the critical values R_N and F_B and the average outdegree, as observed in this data.

Conclusion

In this paper, we have taken notice of a new type of emergent phenomenon that is likely to arise in networks of message-passing agents, with potentially disastrous results. We have presented a solution that hinges on the recognition of similarities in message texts. We have evaluated that solution, both in isolation and, by simulation, in the context of a randomly connected network of mail-forwarding agents.

In isolation, the anti-maelstrom solution was seen to have minimal impact on storage or CPU usage. When deployed among agents in a random network, we found that when the rate at which redundant messages were incorrectly identified was below a threshold value, the maelstrom was eventually quenched. Similarly, when the fraction of agents refusing to adopt the anti-maelstrom method was less than a threshold, the maelstrom was reduced to manageable proportions. Both of these thresholds were linked to topological properties of the network. Even for false positive rates as high as 10%, the fraction of agents in the network that failed to receive the message was negligible. In addition, a nonzero false negative rate was found to mitigate the effect of false positives.

Plans for future work include more detailed numerical experiments on the performance of the solution we proposed, particularly under different network topologies. The connection between collective dynamics and network topology has been investigated in a number of different contexts (see, for example, [Kephart, 1994], [Watts & Strogatz, 1998]); some of those methods and results may be applied or adapted to maelstrom dynamics.

We have barely touched on the heuristic classification methods used by the agents in distinguishing, for example, important variations in a message from unimportant ones. In any practical implementation, of course, a viable heuristic is essential. But for the purpose of understanding collective behavior in a network, it is reasonable to approximate any such heuristic by false positive and false negative error rates. Studies of the sort presented here, if conducted on sufficiently realistic network topologies, are an essential tool in establishing the relation between the "microscopic" observables of a heuristic—e.g., its error rates—and the induced "macroscopic" behavior of the network—such as delivery failures and flooding. Since notions of acceptable behavior are likely to be expressed in terms of macroscopic properties, this relation will help heuristic designers to achieve a reasonable tradeoff between false negative and false positive rates.

Perhaps more important than the details of the particular phenomenon we have studied, however, is the example it provides of a danger innate to emergent multiagent systems: Agents may easily be programmed to perform actions which, considered in isolation, are perfectly innocent, benign, and well-behaved—yet when deployed in a network, wreak collective havoc. Without due attention, this danger could prove to be a significant barrier to the widespread, continued use of agent technology on the Internet.

References

Demazeau, Y., ed. 1998. *Proceedings of the Third International Conference on Multi-Agent Systems.* IEEE Computer Society.

Kephart, J. O., and Arnold, W. C. 1994. Automatic extraction of computer virus signatures. In Duffield, P., ed., *Proceedings of the Fourth International Virus Bulletin Conference*, 179–194. Abingdon, England: Virus Bulletin Limited.

Kephart, J. O. 1994. How topology affects population dynamics. In Langton, C. G., ed., *Artificial Life III*, 447–464. Reading, MA: Addison Wesley.

Manber, U. 1990. Chain reactions in networks. *Computer* 57–63.

Spagna, R. 1997. A method and system for preventing routing maelstrom loops of automatically routed electronic mail. Patent application.

Watts, D. J., and Strogatz, S. H. 1998. Collective dynamics of 'small-world' networks. *Nature* 393:440.