Network Management and Control Mechanisms to Prevent Maliciously Induced Network Instability[1]

Ronald Skoog and Nicholas Jasinski
Telcordia Technologies
331 Newman Springs Road
Red Bank, NJ 07701

Mark Shayman, Rana Ghahremanpour and Mehdi Kalantari
Department of Electrical and Computer Engineering
University of Maryland
College Park, MD 20742

Abstract: Large networks relying on real-time processing can be driven into unstable modes of operation (e.g., routing system failures, routing flaps, congestion and deadlock scenarios, system crash chain reactions, etc.). In the past, unintentional system faults have led to frame relay networks, SS7 signaling networks, and PSTNs going into unstable modes that have led to major service disruptions. A serious concern is that a malicious party could induce similar instabilities. The vulnerability of a network to instabilities may be due to unrecognized design flaws or hidden software bugs. Since these details are not known in advance, effective control mechanisms tailored to the specifics of the vulnerability are virtually impossible to achieve. However, it is our contention that there are a limited number of "generic propagation mechanisms" that enable these network instabilities to occur. By enumerating these propagation mechanisms and designing network management and control mechanisms to mitigate them, it would be possible to stabilize networks against malicious attack even when the details of the network vulnerability being exploited are unknown. In this paper, we focus on a single example of a generic propagation mechanism that can occur in IP and ATM networks using link state routing protocols. The propagation mechanism is overload propagation in the control plane caused by excessive route updates. Network management and control mechanisms for mitigating this propagation mechanism are developed and validated through simulation of both the control and data planes.

1. Introduction

The research reported in this paper is part of program whose goal is to develop a capability to detect and stabilize a network that is under attack, and to accomplish this without knowing ahead of time what specific mechanisms are being used to drive the network into an unstable state. Rather, generic propagation mechanisms are used to classify the types of network instability that can occur, and the control techniques used to stabilize the network are designed to prevent the generic propagation mechanisms from remaining active and allowing the network to continue to operate in an unstable mode.

The approach of this program is to build the desired network control capabilities through a process of four steps. The first step is to identify and define a set of generic fault propagation mechanisms that have occurred in telecommunications and other cyber networks. A number of specific examples of network instability events have occurred in telecommunications networks, and these specific events have been used to define the generic propagation mechanisms that were involved. Also, other cyber network events have been explored to identify additional propagation mechanisms.

The second step is to develop a modeling framework based on the generic propagation mechanisms. This modeling framework will allow control structures to be designed without regard to what specific attack scenarios might take place.

The third step is to develop a measurement system that provides the ability to detect the activation of a generic propagation mechanism, and to also provide the required information for the control system to stabilize the network.

[1] Research supported by DARPA under contract N66001-00-C-8037, by the Laboratory for Telecommunications Sciences (LTS), and by ARDA.

The fourth step is to develop control structures, based on the generic propagation mechanisms and the measurement system, that would provide the means to detect and classify an active propagation mechanism and exert controls that would prevent the propagation mechanism from remaining active. Thus, such controls would prevent network instabilities from persisting and causing serious network damage.

There have been a number of large scale network failures. Examples include the 1990 outage of the AT&T switched network, RBOC Signaling System No. 7 (SS7) Common Channel Signaling Network (CCSN) outages in 1990 and 1991, and failures of the AT&T and MCI-Worldcom frame relay networks in 1998 and 1999, respectively. However, due to competitive concerns, almost no nonproprietary analysis of these incidents has been reported in the technical literature. An exception is [HMS+94] which analyzes the 1990 AT&T switched network failure. The April 1994 issue of the IEEE JSAC [BKPS94] was motivated by the considerable efforts, after the 1990-91 RBOC CCSN failures, that went into analyzing CCSN design and related issues such as distributed processing, software reliability, disaster prevention and recovery, topological design, network engineering, and congestion control. The only paper in that issue that explicitly addresses the fault propagation problem is [HJM94], which investigates alternatives to achieve software diversity (a technique that carriers have rejected due to its complexity and significant cost). To our knowledge, there is no previous work that explores the idea of overlaying a control capability on a network with 'unreliable' components to prevent the propagation of failures. Virtually all of the work related to this problem is directed at how to design systems and protocols to be robust assuming complete knowledge and control of the underlying protocols and systems. A typical example is the design of system and network congestion controls. There seems to be no previous work on developing techniques for containing undesirable network behavior under the assumption that network elements and protocols are not robust due to unknown flaws or malicious alterations.

Since fault propagation generates alarm propagation, techniques for alarm correlation are relevant to the third step in our program--detecting activation of a generic propagation mechanism. There is a large body of literature on alarm correlation. (Some examples include [JW93], [JW95], [KS95], [N95], [YK+96], [HSV99] among others.) We also note that in the MAGDA Project [FBJ+00], the interaction of network elements in fault propagation is modeled using Petri nets.

In this paper, we focus on a single example of a generic propagation mechanism: overload propagation in the control plane caused by excessive route updates. This propagation mechanism is one that occurs in IP networks using routing protocols like OSPF (Open Shortest Path First) and in ATM networks using the PNNI (ATM Forum Private Network Network Interface) routing protocol. It has been observed in real network outages. The AT&T frame relay network outage in 1998 [ATT98] and the MCI frame relay network outage in 1999 [SM99] were caused by an underlying ATM network going unstable due to this propagation mechanism. In each case the ATM network used a PNNI type routing protocol (different implementations in the AT&T and MCI case). This type of routing protocol has the link state message flooding and processing attributes this propagation mechanism requires. BGP networks have also experienced instabilities similar to this propagation mechanism [L97].

The work reported in this paper focuses on the second and fourth steps of the general research program as they relate to overload propagation in the control plane. In order to investigate this propagation mechanism, we used two simulators: (1) A simple simulator programmed in C that is used to simulate only the control plane and the interaction of the control plane with network management. (2) An OPNET-based simulator that is used to study the impact on the data plane of the events occurring in the control plane. The simple simulator was used to explore the role of network topology and parameters on the susceptibility of the network to this instability propagation mechanism. It was then used to develop "supervisory controls" to prevent the instability propagation from occurring. Those supervisory control techniques that appeared promising were then further validated by using the OPNET simulator to examine the effect on the behavior of the data plane.

Two distinct control strategies have emerged from the research. In the first strategy, "backoff" mechanisms are used to prevent multiple routers from reinitializing at the same time. This prevents large bursts of route update packets from occurring when crashed routers reboot. In contrast, the second strategy does not delay

router reboots; instead, it performs shaping of the route update traffic associated with both router crashes and reboots. Our results thus far suggest that from the perspectives of both performance and implementation, the control traffic shaping approach is more promising than the backoff approach.

2. Network Model

The network model used to study the propagation of congestion and system failure through link status flooding messages is as follows:

- The network consists of nodes and links interconnecting nodes.
- Each node has a control processing function with a real-time processing capacity of $C_{control}$.
- When a link fails or recovers (i.e., changes state), the nodes connected to that link detect the failure/recovery and flood link state messages to the other nodes in the network informing them of the link state change.
  - The flooding mechanism consists of the initiating node (i.e., the node detecting the failure/recovery) sending a link state message on each working link to which the node is connected. When a node receives a link state message it queues it for processing by the control processing function. When the link state message is processed it is discarded if it is a copy of a previously received message; otherwise it floods the message out on all its links except the one on which it was received, and it updates the routing table to reflect the link state change.
  - Detection of a link failure/recovery by the control processing function can take some time (a detection time) after the failure/recovery event. This detection time can be random or deterministic, and it can depend on the state of the node (e.g., its congestion level). There are also different possible mechanisms used to detect link failure/recovery like using keep alive messages over the link to the adjacent node and responses or indications from Layer 1 or 2.

The propagation mechanism is modeled as follows:

- A trigger event occurs that causes a number of nodes to fail (or appear to fail) at about the same time. A node failure causes all links connected to that node to appear to the network as failed links.
- This trigger event gives rise to a large number of link state messages being flooded through the network at the same time, and this can produce an overload condition in the control processing function in some or all of the working nodes in the network.
  - The overload results because link state messages arrive faster than the node's control processing function can process the messages and do the required routing updates.
- There is a mechanism in the node processing capability that when its control processing function is in overload for some random or deterministic time period, the node will fail and initialize.
  - An alternative mechanism is the control processing can delay the processing of keep-alive messages long enough (when it is sufficiently overloaded) that it appears to fail because it does not respond fast enough.
  - The resulting real or apparent node failures cause the adjacent nodes to the 'failed' node to detect link failure and flood corresponding link state messages.
- The above mechanism can maintain a sufficiently high level of link state packets flooding through the network that other nodes will have control processing congestion and fail, causing more flooding and maintaining a sustained network instability in which there are rolling node failures and recoveries.

The ability of a network to go unstable due to this propagation mechanism (and the dynamics of the instability) depends on many parameters such as network size (number of nodes and links), control processing function capacity $C_{control}$, the processing time of link state update messages, the processing time of routing table updates in response to link state messages, the time for the control level to detect a link failure/recovery, the time for a failed node to reinitialize, etc.

Most of the parameters that govern this propagation mechanism are implementation dependent. It is also possible for this propagation mechanism to be realized from software 'bugs' or malicious activity to implant corrupted code. An example of how implementation decisions can affect this propagation

mechanism is in the scheduling of processing activities related to handling link state messages. The routing tables could be updated for each new link state update message, or if there is a queue the routing table update could be done only after some number of new update messages have been processed. There are different scheduling priorities that can be used between message flooding activities and routing update processing. For example, a simple scheme would be to do sequential processing of link state update messages (i.e., do all processing related to a message as one batch process) on a first-come-first-served basis; alternatively, higher priority could be given for the message flooding activity and the routing table updates given lower priority.

3. Simple Simulation Model

The simple simulation model was developed and implemented as a C program. It models a single routing domain. Only the control plane is simulated. Furthermore, the only control traffic modeled consists of link state updates. The network topology is specified in an input file.

Each router has a processing queue with FIFO service discipline for arriving link state advertisements (LSAs). Time is discretized into slots. The duration of a slot corresponds to the processing time (assumed deterministic) for an LSA; a typical duration would be 1 ms. Thus, a router with nonempty queue completes the processing of one LSA per slot. Each queue has a threshold R and crash probability P-crash. If the queue occupancy exceeds R, the router crashes in the current slot with probability P-crash (independent of the history of buffer occupancy). Each link has a propagation delay that is a configurable number of slots.

When a router crashes, neighbor nodes detect the event and transmit LSAs on each active link. When a down router completes a reboot, the links connected to the router change to the up state and LSAs are flooded for each of these links by each neighbor node. In addition, the rebooted router floods an LSA. When a router receives an LSA from a neighbor, that LSA is placed at the end of the processing queue. When the LSA reaches the head of the queue, it is processed. If it is identified as a duplicate update, it is discarded; otherwise, it is forwarded on each outgoing link other than the link on which it arrived. For simplicity, it is assumed that the processing time is one slot regardless of whether or not it is a duplicate. However, by making minor modifications in the code, this assumption could be relaxed so that the processing time of duplicate LSAs is less than that of new LSAs.

In this model we assume that the only control packets in the network are the link state packets that are generated because of the crash or reboot of some routers. In other words no other packets can occupy the buffers. For this reason, we assume a small number for the threshold R since we are only looking at part of the control packets in the buffer that we are interested in.

The simulation user externally specifies the reboot policy. The default policy is to initiate reboot of a down router in the slot immediately following the slot in which it crashes. It is assumed that rebooting takes an integer number of slots, which is a deterministic parameter that may be specified by the user.

The simulation user specifies a number of routers to be initially in the down state. The program then randomly chooses which routers are initially down. This models the trigger event--i.e., a set of routers that have been crashed by a malicious intruder. The program simulates the propagation mechanism by which other routers may crash due to the LSA processing overloads created by rebooting of down routers and by secondary crashes. We assume that there is no external mechanism that may cause the routers to crash once the propagation mechanism starts. If the simulation reaches the state where all routers are up, all buffers are empty, and there are no LSAs in transit on the links, then no further crashes can occur and the simulation has reached a terminal state of stability. However, this may take a long time or, in some cases never occur. In this case we say that the network did not reach stability.

In order to study control methods that involve traffic shaping in the control plane, the simulator is extended to include an output buffer. When a router generates a new link state packet (following detection of a change of link status) or when it completes processing of a nonduplicate link state packet received from a

neighbor, the link state packet is placed in a FIFO "output buffer". A specifiable control policy determines when a packet is removed from the output buffer, replicated, and copies placed in the link buffers for transmission on outgoing links. Thus, a link state packet received from a neighbor is first placed in the processing buffer, then in the output buffer, and finally in the transmission buffers for every link except the one on which the packet arrived.

4. Network Instability Results without Controls

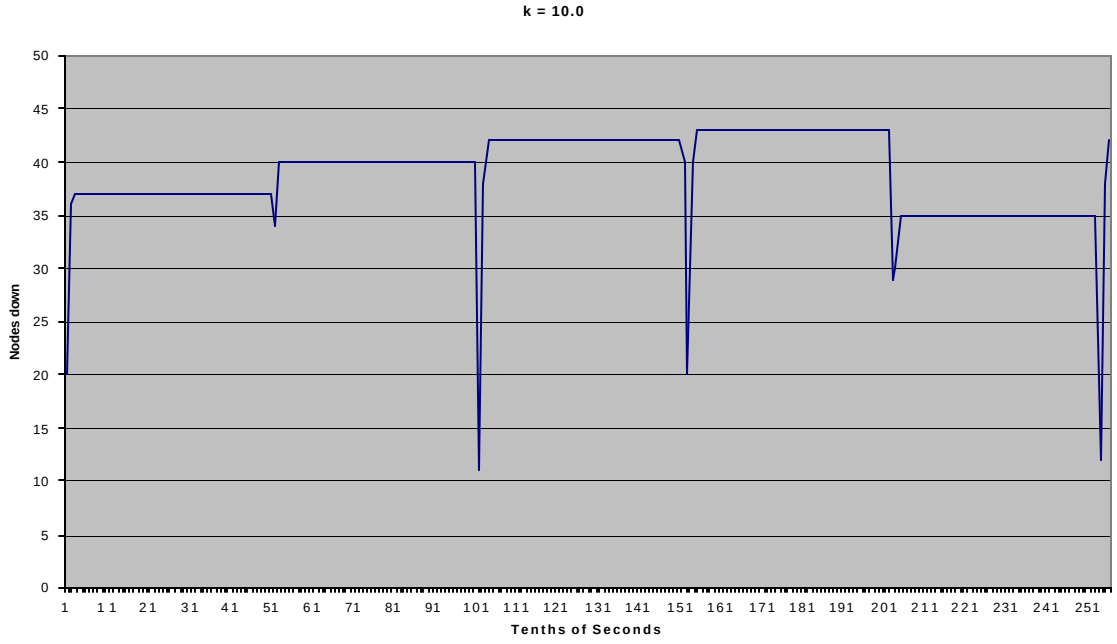
The control plane simulator was used to investigate how various network parameters affect the vulnerability to the overload propagation mechanism in the absence of an externally imposed control policy--i.e., a reboot is initiated immediately after the router crashes. Simulations were done with different parameter values to see their effect on when the network would go unstable.

A base case for the simulation studies was chosen with parameter choices that result in an unstable network after the initial trigger event. Parameter variations around this base case were then explored to see how behaviors changed with different parameter choices. The base case parameters are:

- 100 node network
- Medium connectivity topology (regular topology in which each internal node has four neighbors)
- LSA Buffer threshold = 20 LSAs
- Probability of system crash when threshold is exceeded = 0.04
- Reboot time = 5 seconds
- Trigger event = 20 nodes down
- Link propagation delay = 0

Figure 1 illustrates the network behavior for the base case.

Figure 1 - Down nodes as a function of time for the Base Case



This chart shows that at t = 0 the trigger event of 20 down nodes occurs. Very soon after that additional nodes fail due to the large number of LSAs flooding the network. The network settles down with 37 down nodes. After the reboot time of 5 seconds, the down routers from the initial trigger event complete their

reboot, and there is another flurry of LSAs. This causes more nodes to fail, and the network settles down with 40 rebooting nodes. This process continues and the network remains unstable.

The behavior of the network is highly dependent on the following parameters:

1. The number of nodes
2. The network connectivity
3. The initial condition (trigger event size)
4. The congestion threshold of the LSA processing buffer
5. The probability of crash when the LSA processing buffer congestion threshold is exceeded

The results show that a larger network is more unstable than a smaller network. The topology of the network is another important factor in stability. In general, when comparing similar network structures, we found the network with higher connectivity to have less stability. This is due to the fact that in the case of high connectivity, a large number of copies are generated for any link state packet in the network. With a larger number of LSAs flooding through the network, the queues for LSA processing in the routers get larger, resulting in higher likelihood for router failure. The results show that, in general, more connectivity leads to more instability. However, a network with a small number of routers is normally stable and does not become unstable when the connectivity is high. On the other hand, for a larger network, the connectivity can change the behavior significantly.

The effect of the initial number of down routers (the trigger event size) on the stability behavior of the network was studied. If the network parameters are such that it can go into a sustained instability, then there is a low range of initially down routers that will not cause a sustained instability, an intermediate range where a sustained instability may or may not occur, and an upper range where a sustained instability always occurs. These range values are affected by the other parameter values.

Regarding the last two parameters, LSA congestion threshold size and the probability of a crash with a congested LSA buffer, it was found that networks become less stable as the LSA buffer congestion threshold decreases and as the probability of a crash increases.

5. Control Approaches

Our objective is to find a robust control law that will stabilize the network regardless of the system parameters. Among robust control laws, we seek one that minimizes performance degradation. Two different approaches are taken to control the propagation of system failure resulting from the flooding of link state advertisements.

The first method is to randomize the reboot times of the crashed nodes so that a large amount of traffic is not generated simultaneously. Randomization is used to control the average rate at which routers reboot without requiring a centralized controller to determine the times at which the various down routers are rebooted. Thus, distributed implementation is possible. The average reboot rate is state dependent, depending on the current number of down routers.

The second approach is to use traffic shaping algorithms for control packets. Instead of controlling the rate that routers reboot, this method controls the release of link state packets in order to prevent congestion in the control plane. The control traffic is shaped by implementing a token bucket controller.

5.1 Randomizing the reboot time

Each down router is rebooted in any time slot with a probability *P* that depends on the number of down routers in the network. The control law is distributed in the sense that the decision whether to reboot a given router in a slot does not depend on the decisions made for other routers. However, it assumes that the local agent making the control decision knows the current number of down routers in the whole routing domain. This information might be difficult to obtain during times when many routers have crashed and the network is flooded with link state packets.

Let the number of down routers in the network be N_d. We begin by considering the parameterized family of control laws of the form

$$P = \frac{k}{N_d + bN_d^2}$$

k and *b* are parameters to be chosen to optimize the control law. The expected number of down routers that start rebooting in a time slot is

$$\frac{k}{1+bN_d}$$

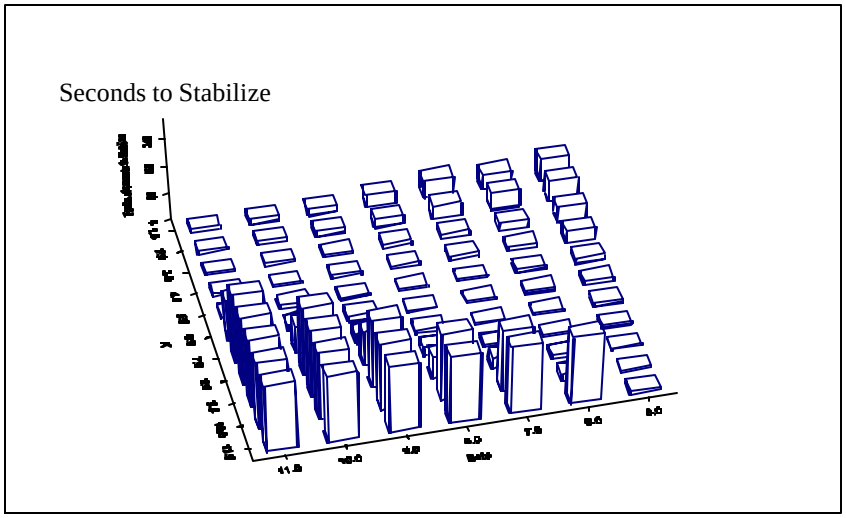
These control laws have the desirable feature that the expected number of routers that reboot in a given slot decreases as the number of down routers increases.

A number of simulation runs were done for the base case network with different values for *K* and *b*. The objective was to determine how the control behavior changed and what optimal choices were for *K* and *b*. We then repeated these types of runs varying *K* and *b* with perturbations from the base case to see how the 'optimal' control law changed as network parameters changed.

Figure 2 shows the time it takes to stabilize the base case network with different *K* and *b* pairs. The results show that for each value of *b*, as *K* is increased from 0.005 the time to stabilize the network decreases to a minimum and then rises rather sharply. The optimal value for *K* increases as *b* increases.

The sharp rise in time to stabilize near the optimal control point is due to the fact that as the control law speeds up the recovery, the network gets closer to secondary node failures due to the recovering nodes.

Figure 2 - Time to stabilize the base case network



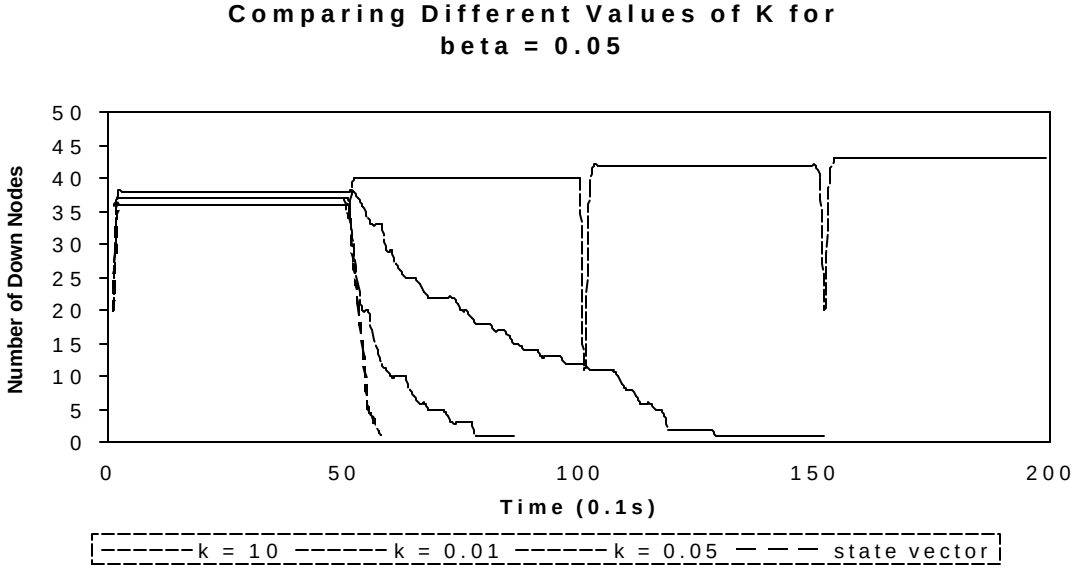
The observed behavior of the optimal choices of K and β are as follows:

- To optimize the (K, β) pair it is better to fix β and optimize over K. This is because the recovery time as K varies gives a more predictable minimum. This is seen by the variability shown in Figure 2.
- The optimal (K,β) pair depends on the different network parameters.

- Varying trigger event sizes shows that the best choice for β is 0.05, and the best value for K varies with the trigger event size. The best K value varies from 0.5 to 2 as the trigger event varies from 20% to 100% of the nodes.
- Varying the density (node degree) parameter shows (K,β) pairs (0.1, 0), (0.5, 0.05) and (1, 0.1) are optimal across density variation.
- Varying crash probability shows that β in the range 0.05-0.1 and K=0.5 gives the best performance.
- These results allow us to conclude that (K,β) = (0.5, 0.05) is optimal for varying density and crash probability, but not for varying trigger event sizes.

In general, these results show that the optimum values of *K* and *b* depend on the initial number of down routers. This suggests that improved performance and robustness can be obtained by permitting *K* and *b* to be state-dependent--i.e., to depend on the current number of down routers. For this reason a 'state vector' control law is considered. In this method the number of down routers is divided into a number of ranges and an optimum probability is associated to each range using the optimum *K* and *b* associated with the initial value of down routers in that range. This leads to a more stable network than that obtained using fixed values for *K* and *b*. This is illustrated in Figure 3, where the state vector control law is compared to the (*K, b*) control laws for the base network simulation configuration with *b* held fixed at 0.05.

Figure 3 - Comparison of State Vector Control Law with (K,b) Control Law



5.2 Shaping the control traffic

In this approach we use a token bucket [LW2000, p. 524] to shape the flow of control packets out of each router in order to prevent congestion. When a router generates an LSA to be flooded, or completes the processing of a non-duplicate LSA that it receives from a neighbor, the LSA is placed in a FIFO output buffer. A token bucket (TB) regulates departures from the output buffer. Tokens are generated at the rate of one per time slot. A packet needs to wait for *T* tokens to be available before it can depart from the output buffer and be transmitted on the appropriate set of outgoing links. Tokens are collected in the bucket if there are no packets to be sent out. The capacity of the bucket is $T + \tau$ tokens.

Deterministic Analysis Based on Worst-Case: Assuming that no more than one packet can be generated in a time slot by a node, the maximum burst of outgoing packets *M* happens when the bucket is full and the node sends out one packet per time slot. It can be shown that

$$M = \left\lfloor \frac{t}{T-1} + 1 \right\rfloor$$

After a burst of *M* consecutive time slots during each of which one packet is transmitted (on each of the appropriate outgoing links), the packets are sent out with a rate of one per *T* time slots.

Assume that each node has exactly *N* neighbors. The worst case is when all the *N* neighbors of a node are sending a burst at the same time, that is, for *M* time slots, all the *N* links are sending one packet per time slot. Therefore the receiving node receives *N* packets at each time slot for *M* consecutive slots. It can process only one packet at each time slot, so it will accumulate *N-1* packets at each slot. Therefore, *M(N – 1)* packets are added to its buffer. To ensure that this does not cause the buffer to overload, M should satisfy

$$MN - M + 1 \le R$$

or

$$M \le \frac{R-1}{N-1}$$

where *R* is the queue threshold for overload. After a burst of M, it is possible that each neighbor sends a packet every *T* time slots. In other words, at any time slot, *N/T* packets are added to the buffer and one packet is processed. To ensure that the number of packets accumulated in the buffer is not increasing without bound, T should satisfy

$$\frac{N}{T} \le 1$$

The preceding pair of inequalities can be used to calculate the parameters of the token bucket such that the probability of secondary crash is zero. This analysis is similar to the deterministic treatment of admission control for ATM using the concept of effective bandwidth [WV2000].

The throttling mechanism allows all routers to recover after their reboot time, but the network recovery may be adversely affected since the throttling of the link state advertisements can delay the convergence of the routing algorithm. In order to capture this aspect of the problem it is necessary to model the data plane and observe the data throughput. Consequently, investigation of this issue required the development of the OPNET simulation model.

The token bucket controller has two key advantages over the probabilistic backoff controller. Firstly, the TB controller is truly distributed since it does not require global knowledge of the current number of down routers. Secondly, the TB controller can prevent overloads caused by both crashes and reboots, while the backoff controller can affect only the congestion caused by reboots.

6 OPNET Simulation Model

The simulation model is based on OPNET's generic IP models with alterations to allow nodes to be failed and rebooted based on the state of the network. The first step was to develop a model of the nodes that would replicate the results we had obtained with the simple simulation model described above. The second step was to develop the data plane model capabilities.

The simulated network for congestion experiments consists of 48 nodes arranged in a grid 6 nodes wide and 8 nodes high. Connected to the right side of the grid are 8 workstations. On the left are 8 servers. The workstations and servers exchange UDP traffic that models many small database requests. The average traffic is about 45 data packets per second. For disconnect experiments the workstations and servers are distributed around the periphery of the grid.

The basic IP node model (router) consists of 4 links (receiver/transmitter pairs) connected to a central IP processor. The IP processor is also connected to high level processors for routing control like OSPF. To control the node, queue processors were inserted between the IP processor and each of the 8 transmitters

and receivers. In addition, one central processor was added to control the 8 queue processors. With this, the following capabilities were added to the IP nodes:

- Starting a node down at the beginning of the simulation shortly after the data traffic starts.
- Queuing of OSPF packets as they arrive at the node.
- Processing of OSPF packets at a rate specified by the user.
- Failing a node based on the size of its OSPF queues.
- Rebooting a node after a specified time, or rebooting a node based on a Probability Control Law.
- Queuing outgoing UDP and OSPF traffic and allowing the user to specify the rate that the node can process and send packets.
- Dropping outgoing traffic when queues are filled.

These modifications enable simulation of both a Probability Control Law and Throttle Control Law. The Throttle Control Law is a simplified Token Bucket Controller in which the parameter $\tau = 0$. Thus, it imposes an absolute maximum rate and does not allow bursts that exceed this rate.
The initial studies using the OPNET simulation model were focused on two network models: one to examine the network behavior when failures cause the links in the network to go into congestion, and the other to examine what happens when the network becomes disconnected.

The primary measure on the data plane performance is the throughput of the UDP packets from workstations to servers. The network simulations were run for four cases:

- No routers are allowed to fail after the initial trigger failure events – this is the best case for data plane throughput since all trigger event failures will recover after the reboot time and the network will be fully operational.
- No control law is used – this gives the worst case since the network is unstable and never reaches a point where all routers are recovered.
- The (K, β) probability control law is used.
- The throttle control law is used.

An example result of this congestion case is provided in Table 1. The different scenarios are compared with the ideal case when routers are not allowed to fail after the trigger event. Without a control law the network is unstable, and the highest percentage loss is seen (43.3%). The network is stabilized with the probability control law and the throttle control law, with the throttle control law showing a significant improvement. Further studies are needed to establish that the throttle control law will always do better.

Table 1 - Summary of Throughput Results for Congestion Case

Scenario	UDP Packet Throughput (% Loss)
No Failures after Trigger Event	15202 (26.8%)
No Control Law	10333 (43.3%)
Probability Control Law	12292 (36.1%)
Throttle Control Law	13962 (29.5%)

Figure 4 shows time profiles for the four cases summarized in Table 1. The four curves presented in each scenario are, going top to bottom:

- the number of down nodes (i.e., nodes rebooting),
- the number of UDP packets being received per second by all servers,
- the number of UDP packets being dropped per second in the network, and
- the number of UDP packets being dropped per second due to IP routing not being able to route the packets.

Figure 4a shows the unstable case with no control used to stabilize the network. In this case after the trigger event of 21 routers down, the UDP throughput drops more than half and stays at a significantly reduced level of throughput. It is also seen that the UDP drop rate jumps up to around 20 and stays there most of the time. There is a dip around the time the trigger event routers finish their reboot. At that time it is seen there

is a pulse of packet drops due to IP routing reconfiguration, and the UDP packets dropped due to congestion goes down. The total drop rate, however, remains about the same (at 20).

Figure 4b shows the ideal case when all routers recover after the reboot time and there are no subsequent failures. There is a time delay between when all routers recover and the UDP packet throughput starts to increase. This is due to the time it takes the network to recognize the nodes are up and reconfigure the routing to use all the links in the network. As more links get used, the congestion reduces and the throughput increases. During the time the network is reconfiguring, it is also seen that IP routing is dropping packets as in the previous case (Figure 4a). In this case, however, after that reconfiguration period the drop rate quickly decreases to zero.

Figures 4c and d show the results for the throttle and probability control laws. The throttle control law has a faster network recovery, where network recovery is defined as the time when the network throughput rate returns to normal. The bars at the top of the charts compare the recovery times of the ideal case, throttle case and probability case.

Further studies are being done to explore the congestion case further and also study the case when the network becomes disconnected. Some initial results on the disconnected case indicate the probability law might be better in some cases. Another important area being investigated is setting the throttling rate for the Throttle Control Law. It appears to be feasible to have an adaptive mechanism to adjust the throttling rate to achieve very good performance. Details of the adaptive techniques will be reported in future publications.

7. Conclusions

In this paper we have examined one generic propagation mechanism that makes networks using link state routing protocols (e.g., OSPF and PNNI) vulnerable to being triggered into unstable behavior. Such instability behavior has been observed in commercial networks, and when it has occurred the impact has been severe. Simulation models have been developed to demonstrate how these instabilities arise and what the impact of different control mechanisms can be. It has been shown that there are various control laws that can be used to stabilize the networks after a trigger event and prevent a sustained instability. Two basic techniques were studied: one that controlled when systems would be allowed to reboot (probability control law) and another that throttled the LSAs to prevent LSA processing overload in the network. Our initial results indicate that the throttling control laws seem to work best for most situations. Further work is being done to see if this is always true. Throttling laws have the advantage that they do not depend on knowledge of the current number of down nodes. We also note that various attacks on OSPF [QVW+98] can elicit increased flooding of LSAs. Throttling control laws can prevent such attacks from triggering overload propagation. Another area of future work is to develop techniques to adaptively optimize the controls. The major challenge in this regard is that the underlying system parameters that govern the instability are not known. Our studies have shown that if the system parameters are known, the probability and throttling control laws can be tuned to give very effective controls. However, our basic hypothesis in this work is that we do not know the specific implementation and parameters of the system. We only know the generic propagation mechanism. Therefore, adaptive techniques are needed to tune the controls based on observed network behavior. Current work has shown there are some promising techniques for doing this with the throttling controls.

REFERENCES

[ATT98] AT&T, "AT&T announces cause of frame-relay network outage," News Release, April 22, 1998, http://www.att.com/press/0498/980422.bsb.html.

[BKPS94] V.A. Bolotin, P.J. Kuhn, C.D. Pack, and R.A. Skoog Guest Editors, *Common Channel Signaling Networks: Performance, Emgineering, Protocols, and Capacity Management*, IEEE JSAC, vol.12, No. 3, April 1994.

[FBJ+00] E. Fabre et al., "MAGDA: Alarm supervision in telecommunication networks," http://www.irisa.fr/sigma2/benveniste/pub/magda2000.html.

[HJM94] N.L. Hung, A.R. Jacob, and S.E. Makris, "Alternatives to Achieve Software Diversity in Common Channel Signaling Networks," IEEE JSAC, vol. 12, No. 3, April 1994, pp. 533-538.

[HSV99] M. Hasan, B. Sugla and R. Viswanathan, "A conceptual framework for network management event correlation and filtering systems," *IEEE/IFIP Symposium on Integrated Network Management,* 1999, 233-246.

[HMS+94] D. J. Houck et al., "Failure and congestion propagation through signaling control," *International Teletraffic Congress,* 14 (1994), 367-376.

[JW93] G. Jakobson and M. Weissman, "Alarm correlation," *IEEE Network,* 7 (6), November 1993.

[JW95] G. Jakobson and M. Weissman, "Real-time telecommunication network management: extending event correlation with temporal constraints," *IEEE/IFIP Symposium on Integrated Network Management,* 1995.

[KS95] I. Katzela and M. Schwartz, "Schemes for fault identification in communication networks," *IEEE Transactions on Networking,* 3 (6), 1995.

[L97] C. Labovitz et al., "Internet Routing Instability," *Proceedings of the ACM SIGCOMM,* Nice, France, August, 1997.

[LW00] A. Leon-Garcia and I. Widjaja, *Communication Networks,* McGraw Hill, Boston, 2000.

[N95] Y. A. Nygate, "Event correlation using rule and object based techniques," *IEEE/IFIP Symposium on Integrated Network Management,* 1995.

[QVW+98] D. Qu et al. "Statistical anomaly detection for link-state routing protocols," *Proceedings of IEEE Conference,* September 1998, 62-70.

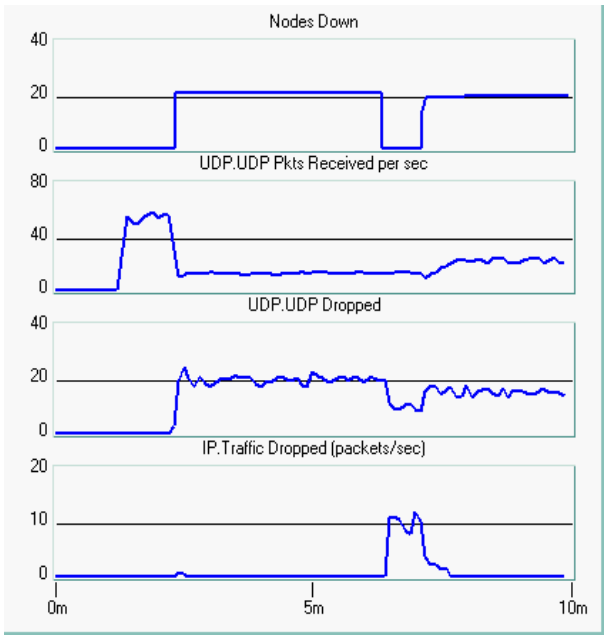
[SM99] T. Sweeny and C. Moozakis, "MCI frame net melts down," *Tech Web,* August 12, 1999, http://content.techweb.com/wire/story/TWB19990812S0013.

[WV00] J. Walrand and P. Varaiya, *High Performance Communication Networks*, 2nd ed., Morgan Kaufmann, San Francisco, 2000.

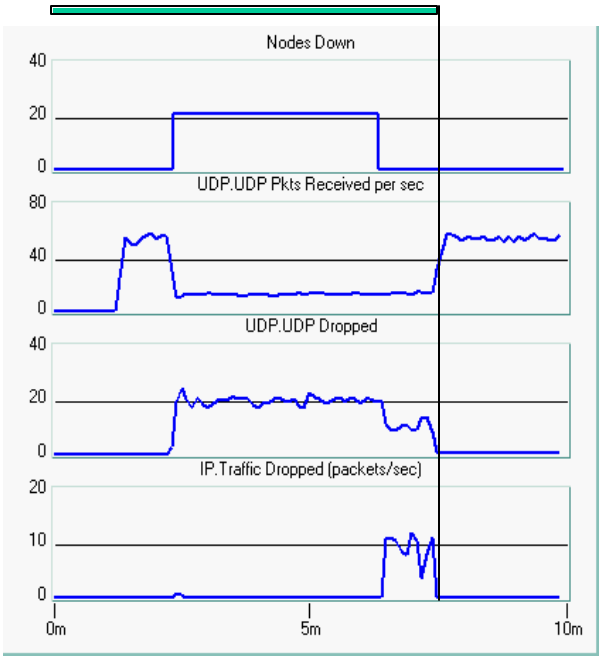
[YK+96] S. Yemini et al., "High speed and robust event correlation," *IEEE Communications Magazine,* May 1996.

Figure 4--Curves for the Four Cases in Table 1

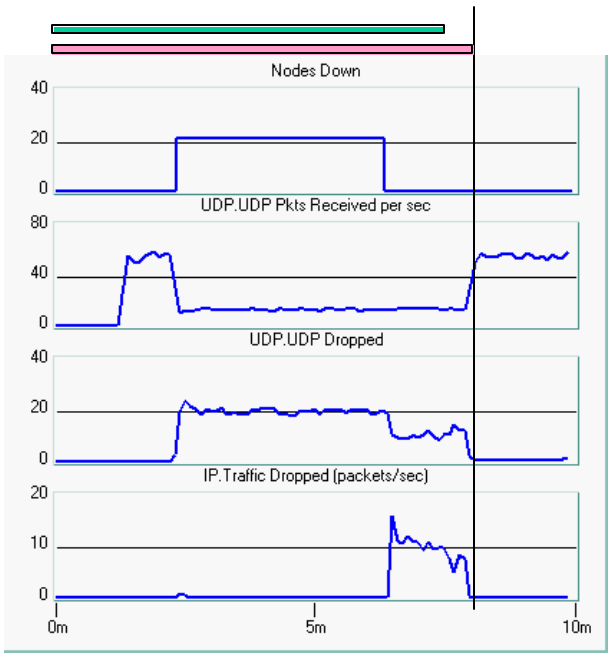
(a) No Control Law



(b) No Failures



(c) Throttle Control Law



(d) Probability Control Law